

Algoritmos e Estruturas de Dados

Questões do ENADE 2005, 2008, 2011, 2014, 2017, 2019, 2021 e 2023

Computação: Ciência da Computação (Bacharelado e Licenciatura), Sistemas de Informação,

Engenharia de Computação e Tecnologia em Análise e Desenvolvimento de Sistemas

Enunciados, gabaritos oficiais e resoluções comentadas

Prof^a Ana Carolina Sokolonski Anton

Instituto Federal de Educação, Ciência e Tecnologia da Bahia (IFBA)

Campus Feira de Santana

Fontes: Cadernos de prova, gabaritos definitivos e padrões de resposta do INEP.

Sumário

Como este material está organizado	2
1 Estruturas de dados lineares: listas, pilhas e filas	4
2 Árvores e <i>heaps</i>	17
3 Tabelas de dispersão (<i>hash</i>)	32
4 Busca e ordenação	36
5 Recursão e análise de complexidade	54
6 Grafos	71
7 Técnicas de projeto de algoritmos	89
8 Lógica de programação com vetores e matrizes	97

Como este material está organizado

Foram selecionadas, das provas de 2005, 2008, 2011, 2014, 2017, 2019, 2021 e 2023, apenas as questões cujo conteúdo principal é *algoritmos* (projeto, análise, recursão, busca, ordenação, grafos) ou *estruturas de dados* (listas, pilhas, filas, árvores, *heaps*, tabelas *hash*). As questões estão agrupadas por tema; o cabeçalho de cada uma indica o ano, o curso e o número original. Questões idênticas em mais de uma prova aparecem uma única vez, com todas as origens indicadas.

Nº	Origem	Tema	Gab.	Tipo
1	2014 CC Bach. Q16	Pilha em vetor (C)	C	Objetiva
2	2017 CC Bach./Lic. D3	Lista encadeada e pilha	–	Discursiva
3	2017 SI Q35	Escolha de estrutura (pilha)	C	Objetiva
4	2021 CC Bach./Lic. Q10	Deque, lista, pilha, fila	B	Objetiva
5	2021 TADS D5	Lista com prioridade	–	Discursiva
6	2023 EC Q15	Vazamento de memória (C)	C	Objetiva
7	2023 EC Q33	Pilha × fila (PID)	D	Objetiva
8	2011 CC Q39	Tipos abstratos de dados	D	Objetiva
9	2011 Comp. Q21	Pilha × fila (sequências)	A	Objetiva
10	2011 TADS Q29	Operações em pilha	A	Objetiva
11	2014 CC Bach. Q13	Árvore binária de pesquisa	B	Objetiva
12	2017 CC Bach./Lic. Q9	Árvore AVL (rotação)	A	Objetiva
13	2021 CC Bach./Lic. Q23	ABB e percursos	C	Objetiva
14	2021 CC Bach. D5	<i>Heap</i> binário	–	Discursiva
15	2019 EC D3	Construção de árvores AVL	–	Discursiva
16	2008 Comp. Q14	Pré-ordem iterativa com pilha	E	Objetiva
17	2011 Comp. Q24	<i>Heaps</i> e filas de prioridade	C	Objetiva
18	2011 Comp. Q30	Caminho de busca em ABB	C	Objetiva
19	2011 Comp. D4	ABB a partir de vetor ordenado	–	Discursiva
20	2005 Comp./SI D39	Percursos e operações em ABB	–	Discursiva
21	2021 SI Q33	Tabela <i>hash</i> (encadeamento)	Anul.	Objetiva
22	2008 Comp. D20	<i>Hash</i> com encadeamento	–	Discursiva
23	2014 SI D3	Busca binária	–	Discursiva
24	2014 SI Q10	Complexidade de busca	B	Objetiva
25	2021 CC Bach./Lic. Q20	Busca sequencial × binária	A	Objetiva
26	2017 CC Bach./Lic. Q18	Ordenação por inserção	D	Objetiva
27	2021 CC Bach. Q32	<i>Quicksort</i>	A	Objetiva
28	2019 EC Q25	Busca binária e recursão de cauda	C	Objetiva
29	2019 EC Q9	<i>Mergesort</i>	E	Objetiva
30	2023 EC D2	Duplicatas, <i>quicksort</i>	–	Discursiva
31	2005 Comp./SI Q13	Propriedades de ordenação	E	Objetiva
32	2005 Comp./SI Q15	Complexidade do <i>mergesort</i>	E	Objetiva
33	2005 Comp./CC-EC Q43	Comparações na busca binária	B	Objetiva
34	2008 TADS Q20	<i>Bubble sort</i> (rastreo)	E	Objetiva
35	2008 TADS Q22	Completar busca binária	A	Objetiva
36	2014 CC Bach. Q12	Complexidade (Horner)	D	Objetiva
37	2014 CC Bach. Q23	Rastreamento de recursão	D	Objetiva
38	2017 CC Bach. Q25	Fibonacci recursivo	C	Objetiva
39	2017 CC Bach. Q33	Recorrência / chamadas	B	Objetiva
40	2014 CC Bach. Q28	P × NP	A	Objetiva
41	2019 EC D4	Produtório iterativo/recursivo	–	Discursiva
42	2023 EC Q14	Recursão em listas (funcional)	E	Objetiva

Nº	Origem	Tema	Gab.	Tipo
43	2023 EC Q18	Vetor dinâmico (amortizada)	B	Objetiva
44	2005 Comp./SI Q21	Recursividade (conceitos)	C	Objetiva
45	2005 Comp./CC-EC Q51	Torre de Hanói	C	Objetiva
46	2005 Comp./CC-EC Q54	Recorrência LCS (PD)	B	Objetiva
47	2005 Comp./CC Q65	Classes P, NP, NP-completo	B	Objetiva
48	2008 Comp. Q18	Chamadas de Fibonacci	C	Objetiva
49	2008 TADS Q27	Fibonacci recursivo	C	Objetiva
50	2011 Comp. D3	Fibonacci iterativo/recursivo	–	Discursiva
51	2011 CC Q36	$P \times NP$	D	Objetiva
52	2017 CC Bach. D5	Busca em profundidade	–	Discursiva
53	2017 CC Bach. Q24	Dijkstra, Kruskal	E	Objetiva
54	2017 SI Q19	Conceitos de grafos	B	Objetiva
55	2021 CC Bach. Q34	Dijkstra (iterações)	C	Objetiva
56	2014 CC Bach. Q30	Árvore geradora mínima	A	Objetiva
57	2014 CC Bach. Q22	Caixeiro viajante	E	Objetiva
58	2019 EC Q11	Coloração de grafos	C	Objetiva
59	2019 EC Q24	Árvore geradora mínima	C	Objetiva
60	2023 EC Q32	Busca gulosa em grafo	B	Objetiva
61	2005 Comp./CC-EC Q44	Conectividade, árvores, grafo completo	C	Objetiva
62	2005 Comp./CC-EC Q48	Ordenação topológica	A	Objetiva
63	2005 Comp./CC Q58	$A^* \times$ busca gulosa	B	Objetiva
64	2008 CC Q31	Busca cega $\times$ heurística	B	Objetiva
65	2008 EC Q51	Busca gulosa (8- <i>puzzle</i>)	A	Objetiva
66	2011 Comp. Q20	Lema do aperto de mãos	D	Objetiva
67	2011 CC Q40	A^* (heurística de Manhattan)	B	Objetiva
68	2014 CC Bach. D3	<i>Backtracking</i> (Sudoku)	–	Discursiva
69	2014 CC Bach. D5	Divisão e conquista	–	Discursiva
70	2017 CC Bach. Q22	Algoritmo guloso (troco)	C	Objetiva
71	2023 EC Q17	Programação dinâmica (LCS)	D	Objetiva
72	2008 TADS Q26	Paradigmas de projeto	Anul.	Objetiva
73	2011 Comp. Q13	Guloso: seleção de intervalos	Anul.	Objetiva
74	2011 Comp. Q28	Paradigmas de projeto	B	Objetiva
75	2014 SI Q9	Laços aninhados em vetores	A	Objetiva
76	2017 SI D4	Matriz (contagem)	–	Discursiva
77	2017 SI Q26	Precedência de operadores	E	Objetiva
78	2021 TADS D3	Vetor: maior, menor, média	–	Discursiva
79	2005 Comp./SI Q14	Matriz em vetor (<i>row-major</i>)	C	Objetiva
80	2008 TADS Q21	Erros em pseudocódigo	B	Objetiva
81	2008 TADS D40	Matriz (rastreo)	Anul.	Discursiva
82	2011 TADS D4	Vetores (rastreo)	–	Discursiva
83	2011 TADS D5	Vetor e acumulador (trigo)	–	Discursiva

CC = Ciência da Computação; SI = Sistemas de Informação; EC = Engenharia de Computação; TADS = Tecnologia em Análise e Desenvolvimento de Sistemas; Comp. = prova única da área de Computação (2005, 2008 e 2011), com o perfil indicado quando a questão era específica de um curso; D = questão discursiva; Anul. = anulada.

1 Estruturas de dados lineares: listas, pilhas e filas

Questão 1

ENADE 2014 – Ciência da Computação (Bacharelado) – Questão 16

Uma pilha é uma estrutura de dados que armazena uma coleção de itens de dados relacionados e que garante o seguinte funcionamento: o último elemento a ser inserido é o primeiro a ser removido. É comum na literatura utilizar os nomes *push* e *pop* para as operações de inserção e remoção de um elemento em uma pilha, respectivamente. O seguinte trecho de código em linguagem C define uma estrutura de dados pilha utilizando um vetor de inteiros, bem como algumas funções para sua manipulação.

```
#include <stdlib.h>
#include <stdio.h>
typedef struct {
    int elementos[100];
    int topo;
} pilha;

pilha * cria_pilha() {
    pilha * p = malloc(sizeof(pilha));
    p->topo = -1;
    return p;
}

void push(pilha *p, int elemento) {
    if (p->topo >= 99)
        return;
    p->elementos[++p->topo] = elemento;
}

int pop(pilha *p) {
    int a = p->elementos[p->topo];
    p->topo--;
    return a;
}
```

O programa a seguir utiliza uma pilha.

```
int main() {
    pilha * p = cria_pilha();
    push(p, 2);
    push(p, 3);
    push(p, 4);
    pop(p);
    push(p, 2);
    int a = pop(p) + pop(p);
    push(p, a);
    a += pop(p);
    printf("%d", a);
    return 0;
}
```

A esse respeito, avalie as afirmações a seguir.

- I. A complexidade computacional de ambas funções *push* e *pop* é $O(1)$.
- II. O valor exibido pelo programa seria o mesmo caso a instrução `a += pop(p);` fosse trocada por `a += a;`
- III. Em relação ao vazamento de memória (*memory leak*), é opcional chamar a função `free(p)`, pois o vetor usado pela pilha é alocado estaticamente.

É correto o que se afirma em

- A) I, apenas.
- B) III, apenas.
- C) I e II, apenas.
- D) II e III, apenas.
- E) I, II e III.

Gabarito oficial: C

I – correta. `push` e `pop` executam um número constante de operações (um teste, um incremento/decremento e um acesso ao vetor), logo são $O(1)$.

II – correta. Rastreando a pilha (topo à direita): `push 2, 3, 4` $\rightarrow [2,3,4]$; `pop` $\rightarrow [2,3]$; `push 2` $\rightarrow [2,3,2]$; `a = pop + pop` $= 2 + 3 = 5 \rightarrow [2]$; `push(a)` $\rightarrow [2,5]$; `a += pop` $\Rightarrow a = 5 + 5 = 10$. Com `a += a`, também se obtém $a = 10$ (o elemento desempilhado era justamente o próprio valor de `a`).

III – incorreta. O vetor `elementos` é um campo da `struct`, e a `struct` inteira foi alocada dinamicamente com `malloc`; portanto ela reside no *heap* e deve ser liberada com `free(p)`.

Observação

O código da prova contém `return pilha;` em `cria_pilha()`, o que não compila (`pilha` é o nome do tipo; o correto seria `return p;`). A questão não foi anulada e o erro não interfere nas afirmativas, mas vale corrigir se o código for usado em sala.

Questão 2

ENADE 2017 – Ciência da Computação (Bacharelado e Licenciatura) – Discursiva 3

Listas lineares armazenam uma coleção de elementos. A seguir, é apresentada a declaração de uma lista simplesmente encadeada.

```
struct ListaEncadeada {
    int dado;
    struct ListaEncadeada *proximo;
};
```

Para imprimir os seus elementos da cauda para a cabeça (do final para o início) de forma eficiente, um algoritmo pode ser escrito da seguinte forma:

```
void mostrar(struct ListaEncadeada *lista) {
    if (lista != NULL) {
        mostrar(lista->proximo);
        printf("%d ", lista->dado);
    }
}
```

Com base no algoritmo apresentado, faça o que se pede nos itens a seguir.

- Apresente a classe de complexidade do algoritmo, usando a notação Θ . (valor: 3,0 pontos)
- Considerando que já existe implementada uma estrutura de dados do tipo pilha de inteiros — com as operações de empilhar, desempilhar e verificar pilha vazia — reescreva o algoritmo de forma não recursiva, mantendo a mesma complexidade. Seu algoritmo pode ser escrito em português estruturado ou em alguma linguagem de programação, como C, Java ou Pascal. (valor: 7,0 pontos)

Padrão de resposta oficial (INEP)

- O estudante deve indicar que o algoritmo tem comportamento assintótico $\Theta(N)$, em que N é o número de elementos da lista linear.
- O estudante pode apresentar qualquer algoritmo que consista em um percurso da lista com posterior impressão dos elementos, mantendo a complexidade linear. Como o algoritmo não possui uma recursão à cauda, necessita de uma pilha auxiliar para armazenar as referências. Exemplos apresentados no padrão:

```
/* Linguagem C */
void mostrar(struct ListaEncadeada *
    lista) {
    Pilha p;
    while (lista != null) {
        p.push(lista);
        lista = lista->proximo;
    }
    while (!pilha.vazia()) {
        lista = p.pop();
        printf ("%d ", lista->dado);
    }
}
```

```
/* Java com java.util.Stack */
public static void mostrar(
    ListaEncadeada lista) {
    Stack<ListaEncadeada> pilha =
        new Stack<ListaEncadeada>();
    ListaEncadeada aux;
    while (lista != null) {
        pilha.push(lista);
        lista = lista.proximo;
    }
    while (!pilha.empty()) {
        aux = pilha.pop();
        print(aux.dado);
    }
}
```

O padrão também aceita uma solução baseada em vetor (conta os N nós, aloca um vetor, copia os dados e imprime de trás para a frente) e a versão em português estruturado:

```
1 mostrar(ListaEncadeada lista)
2     Pilha pilhaAuxiliar;
3     Enquanto (lista <> vazio)
4         pilhaAuxiliar.empilhar(lista.elemento)
5         lista = lista.proximo
6     FimEnquanto
7     Enquanto pilhaAuxiliar.vazia() = falso
8         imprimir(pilhaAuxiliar.desempilhar())
9     FimEnquanto
```

Observação

O exemplo em C do padrão mistura sintaxes (`p.push` e `pilha.vazia()` com nomes diferentes, `null` em vez de `NULL`): é pseudocódigo “com cara de C” e não compila como está. Uma versão em C que compila, usando as operações citadas no enunciado:

```
void mostrar(struct ListaEncadeada *lista) {
    Pilha p;
    inicializar(&p);
    for (struct ListaEncadeada *aux = lista; aux != NULL; aux = aux->proximo)
        empilhar(&p, aux->dado);
    while (!pilhaVazia(&p))
        printf("%d ", desempilhar(&p));
}
```

Questão 3**ENADE 2017 – Sistemas de Informação – Questão 35**

O coordenador geral de um comitê olímpico solicitou a implementação de um aplicativo que permita o registro dos recordes dos atletas à medida que forem sendo quebrados, mantendo a ordem cronológica dos acontecimentos, e possibilitando a leitura dos dados a partir dos mais recentes.

Considerando os requisitos do aplicativo, a estrutura de dados mais adequada para a solução a ser implementada é

- A) o deque: tipo especial de lista encadeada, que permite a inserção e a remoção em qualquer das duas extremidades da fila e que deve possuir um nó com a informação (recorde) e dois apontadores, respectivamente, para os nós próximo e anterior.
- B) a fila: tipo especial de lista encadeada, tal que o primeiro objeto a ser inserido na fila é o primeiro a ser lido; nesse mecanismo, conhecido como estrutura FIFO (*First In – First Out*), a inserção e a remoção são feitas em extremidades contrárias e a estrutura deve possuir um nó com a informação (recorde) e um apontador, respectivamente, para o próximo nó.
- C) a pilha: tipo especial de lista encadeada, na qual o último objeto a ser inserido na fila é o primeiro a ser lido; nesse mecanismo, conhecido como estrutura LIFO (*Last In – First Out*), a inserção e a remoção são feitas na mesma extremidade e a estrutura deve possuir um nó com a informação (recorde) e um apontador para o próximo nó.
- D) a fila invertida: tipo especial de lista encadeada, tal que o primeiro objeto a ser inserido na fila é o primeiro a ser lido; nesse mecanismo, conhecido como estrutura FIFO (*First In – First Out*), a inserção e a remoção são feitas em extremidades contrárias e a estrutura deve possuir um nó com a informação (recorde) e um apontador, respectivamente, para o nó anterior.
- E) a lista circular: tipo especial de lista encadeada, na qual o último elemento tem como próximo o primeiro elemento da lista, formando um ciclo, não havendo diferença entre primeiro e último, e a estrutura deve possuir um nó com a informação (recorde) e um apontador, respectivamente, para o próximo nó.

Gabarito oficial: C

Os registros são inseridos em ordem cronológica e devem ser lidos a partir do mais recente; isto é, o último inserido é o primeiro a ser lido: comportamento LIFO, típico da pilha. Numa pilha encadeada, basta um ponteiro por nó (para o próximo, em direção à base). A fila (B, D) lê primeiro o mais antigo; o deque (A) funciona, mas tem recursos e custo de memória desnecessários; a lista circular (E) não define ordem de leitura a partir do mais recente.

Questão 4

ENADE 2021 – Ciência da Computação (Bacharelado e Licenciatura) – Questão 10

A biblioteca de coleções da linguagem Java disponibiliza implementações de propósito geral para estruturas de dados elementares, como listas, filas e pilhas. Considere as seguintes definições de classes que representam implementações de estruturas de dados disponíveis na biblioteca da linguagem:

- **Classe A:** os objetos são organizados em uma ordem linear e podem ser inseridos somente no início ou no final dessa sequência;
- **Classe B:** os objetos são organizados em uma ordem linear determinada por uma referência ao próximo objeto;
- **Classe C:** os objetos são removidos na ordem oposta em que foram inseridos;
- **Classe D:** os objetos são inseridos e removidos respeitando a seguinte regra: o elemento a ser removido é sempre aquele que foi inserido primeiro.

Nesse contexto, assinale a alternativa que representa, respectivamente, as estruturas de dados implementadas pelas classes A, B, C e D.

- A) Lista circular, lista simplesmente ligada, pilha e fila.
- B) Deque, lista simplesmente ligada, pilha e fila.
- C) Lista duplamente ligada, lista simplesmente ligada, fila e pilha.
- D) Pilha, fila, deque e lista simplesmente encadeada.
- E) Deque, pilha, lista ligada e fila.

Gabarito oficial: B

A: inserção apenas nas duas extremidades → *deque* (fila de duas pontas, **pode inserir e remover do início ou do final** - `ArrayDeque`). B: cada objeto referencia o próximo → lista simplesmente ligada. C: remoção na ordem inversa da inserção → pilha (LIFO). D: remove-se o primeiro elemento inserido → fila (FIFO).

Questão 5

ENADE 2021 – Tecnologia em Análise e Desenvolvimento de Sistemas – Discursiva 5

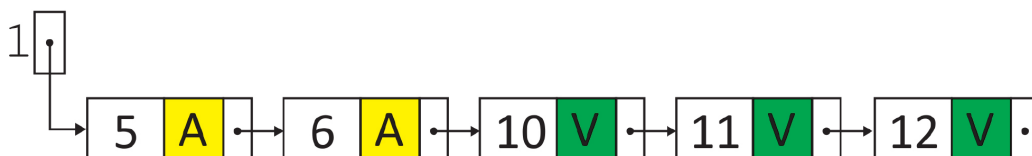
Com a finalidade de melhorar o atendimento e priorizar os casos mais urgentes, a direção de um hospital criou um sistema de triagem em que um profissional da saúde classifica a ordem de atendimento com base numa avaliação prévia do paciente, entregando-lhe um cartão numerado verde (V) ou amarelo (A), que define o menor ou maior grau de urgência da ocorrência, respectivamente. Para informatizar esse processo, o software desenvolvido tem como base o seguinte trecho de código-fonte:

```

1 struct lista {
2     int numero;
3     char cor;
4     struct lista* prox;
5 };
6
7 typedef struct lista Lista;
8
9 Lista* inserir_fim(Lista* l, Lista* no) {
10     Lista* aux;
11
12     no->prox = NULL;
13     aux = l;
14     while (aux->prox != NULL)
15         aux = aux->prox;
16     aux->prox = no;
17
18     return l;
19 }
20
21 Lista* inserir(Lista* l, int numero, char cor) {
22     Lista* no = (Lista*) malloc(sizeof(Lista));
23     no->numero = numero;
24     no->cor = cor;
25
26     if (l == NULL) {
27         no->prox = l;
28         l = no;
29     } else {
30         if (no->cor == 'V')
31             l = inserir_fim(l, no);
32         else
33             l = inserir_prioridade(l, no);
34     }
35
36     return l;
37 }

```

Na linha 21, a função **inserir** recebe o número e a cor do cartão entregue ao paciente na triagem. Pacientes com cartão verde são inseridos no final da fila pela função **inserir_fim** (linhas 9-19). Pacientes com cartão amarelo têm prioridade no atendimento e são inseridos no início da fila, em ordem de chegada, pela função **inserir_prioridade**. Portanto, se são entregues os cartões 10-V, 11-V, 5-A, 12-V e 6-A, nessa ordem, a fila deve ficar assim organizada:



Considerando o processo de triagem descrito e o trecho de código-fonte apresentado, implemente a função `inserir_prioridade` conforme indicado.

```
Lista* inserir_prioridade(Lista* l, Lista* no)
```

(valor: 10,0 pontos)

Padrão de resposta oficial (INEP)

O respondente deve implementar a função `inserir_prioridade`, atendendo a dois fundamentos básicos de inserção de nós em uma lista simplesmente encadeada:

- 1) **Inserção no início:** caso ainda não haja nenhum cartão amarelo na fila, o novo nó é inserido no início, como primeiro elemento da fila;
- 2) **Inserção no meio:** se já houver cartões amarelos na fila, o novo nó é inserido no meio ou no final da fila, a depender, respectivamente, da presença ou da ausência de algum cartão verde; em ambos os casos a lógica é a mesma, pois as variáveis `ant` e `aux` percorrem a lista simultaneamente e atualizam as referências (ponteiros) entre os nós.

```
1 Lista* inserir_prioridade(Lista* l, Lista* no) {  
2     if (l->cartao == 'V') {  
3         no->prox = l;  
4         l = no;  
5     } else {  
6         Lista* ant = l;  
7         Lista* aux = ant->prox;  
8         while (aux != NULL && aux->cartao == 'A') {  
9             ant = aux;  
10            aux = aux->prox;  
11        }  
12        ant->prox = no;  
13        no->prox = aux;  
14    }  
15  
16    return l;  
17 }
```

Serão aceitas respostas em outra linguagem de programação compatível ou pseudocódigo.

Observação

O padrão oficial usa o campo `cartao`, mas a `struct` do enunciado define o campo como `cor`; para compilar, troque `l->cartao` e `aux->cartao` por `l->cor` e `aux->cor`. Note também que `inserir_fim` (linha 12) já anula `no->prox`, mas `inserir_prioridade` não precisa fazê-lo, pois sempre atribui `no->prox`. Rastreando o exemplo: 10-V, 11-V → [10, 11]; 5-A entra no início → [5, 10, 11]; 12-V no fim; 6-A entra depois do último amarelo → [5, 6, 10, 11, 12].

Questão 6

ENADE 2023 – Engenharia de Computação – Questão 15

Memory leak, ou vazamento de memória, é um problema que ocorre em sistemas computacionais quando uma parte da memória, alocada para uma determinada operação, não é liberada quando se torna desnecessária. Na linguagem C, esse tipo de problema é quase sempre relacionado ao uso incorreto das funções `malloc( )` e `free( )`. Esse erro de programação pode levar a falhas no sistema se a memória for completamente consumida.

A partir dessas informações, assinale a opção que apresenta um trecho com *memory leak*.

A)

```
void f( ){
    void *s;
    s = malloc(50);
    free(s);
}
```

B)

```
int f( ){
    float *a;
    return 0;
}
```

C)

```
int f(char *data){
    void *s;
    s = malloc(50);
    int size = strlen(data);
    if (size > 50)
        return(-1);
    free(s);
    return 0;
}
```

D)

```
int *f(int n){
    int *num = malloc(sizeof(int)*n);
    return num;
}

int main(void){
    int *num;
    num = f(10);
    free(num);
    return 0;
}
```

E)

```
void f(int n){
    char *m = malloc(10);
    char *n = malloc(10);
    free(m);
    m = n;
    free(m);
    free(n);
}
```

Gabarito oficial: C

Em C, se `strlen(data) > 50`, a função retorna `-1` antes de chamar `free(s)`: os 50 bytes alocados ficam inacessíveis e nunca são liberados — vazamento de memória.

A: aloca e libera. B: declara um ponteiro, mas não aloca nada. D: a memória alocada em `f` é devolvida ao chamador e liberada em `main`. E: não vaza (ambos os blocos são liberados), mas tem outro defeito: `free(m)` após `m = n` e depois `free(n)` libera o mesmo bloco duas vezes (*double free*); além disso, `char *n` redeclara o parâmetro `n`, o que nem compila.

Questão 7

ENADE 2023 – Engenharia de Computação – Questão 33

Considere que uma empresa planeje desenvolver um sistema de controle automático para manter, em um ambiente industrial, a temperatura constante por meio de um controlador proporcional-integral-derivativo (PID). O controlador PID pode ser implementado em qualquer linguagem que possua estruturas de dados adequadas para a manutenção do histórico e da taxa de mudança do sinal de erro.

Considerando essa situação, avalie as afirmações a seguir.

- I. A implementação do componente Integral requer a soma de todos os erros acumulados desde o início da operação do sistema, o que torna necessário o uso de uma estrutura de dados do tipo fila.
- II. A parte Proporcional do controlador PID é implementada ao ajustar, proporcionalmente, a variável de controle à diferença entre a temperatura desejada e a temperatura atual.
- III. Uma pilha é adequada para o componente Derivativo, pois a taxa de mudança do erro é calculada apenas com a amostra mais recente e a anterior do sinal de erro.

É correto o que se afirma em

- A) I, apenas.
- B) II, apenas.
- C) I e III, apenas.
- D) II e III, apenas.
- E) I, II e III.

Gabarito oficial: D

I – incorreta. A parte integral só precisa da *soma acumulada* dos erros, que cabe numa única variável atualizada a cada amostra (**soma += erro**); não é preciso guardar o histórico numa fila.

II – correta. O termo proporcional é $K_p \cdot e(t)$, com $e(t)$ = temperatura desejada – temperatura atual.

III – correta (segundo o gabarito). O termo derivativo usa só as duas amostras mais recentes, $e_k - e_{k-1}$, que ficam no topo de uma pilha.

Observação

Na prática, a parte derivativa usa apenas duas variáveis (erro atual e erro anterior); uma pilha “funciona”, mas é mais do que o necessário. A questão é boa para discutir em sala quando uma estrutura de dados é de fato exigida e quando bastam variáveis simples.

Questão 8

ENADE 2011 – Ciência da Computação – Questão 39

O conceito de Tipo de Dados Abstrato (TDA) é popular em linguagens de programação. Nesse contexto, analise as afirmativas a seguir.

- I. A especificação de um TDA é composta das operações aplicáveis a ele, da sua representação interna, e das implementações das operações.
- II. Dois mecanismos utilizáveis na implementação de um TDA em programas orientados a objetos são a composição e a herança.
- III. Se S é um subtipo de outro T, então entidades do tipo S em um programa podem ser substituídas por entidades do tipo T, sem alterar a corretude desse programa.
- IV. O encapsulamento em linguagens de programação orientadas a objetos é um efeito positivo do uso de TDA.

É correto apenas o que se afirma em

- A) I. B) II. C) I e III. D) II e IV. E) III e IV.

Gabarito oficial: D

I – incorreta. A *especificação* de um TDA descreve apenas as operações e seu comportamento; a representação interna e a implementação ficam escondidas — é exatamente essa separação que define o TDA.

II – correta. Em OO, um TDA pode ser implementado compondo objetos (por exemplo, uma pilha que contém uma lista) ou herdando de uma classe existente.

III – incorreta. O princípio de substituição (Liskov) diz o contrário: onde se espera um T, pode-se usar um S (o subtipo substitui o supertipo), não o inverso.

IV – correta. O encapsulamento (ocultar a representação e expor só as operações) decorre diretamente da ideia de TDA.

Questão 9

ENADE 2011 – Computação (todas as áreas) – Questão 21

No desenvolvimento de um *software* que analisa bases de DNA, representadas pelas letras A, C, G, T, utilizou-se as estruturas de dados: pilha e fila. Considere que, se uma sequência representa uma pilha, o topo é o elemento mais à esquerda; e se uma sequência representa uma fila, a sua frente é o elemento mais à esquerda.

Analise o seguinte cenário: “a sequência inicial ficou armazenada na primeira estrutura de dados na seguinte ordem: (A,G,T,C,A,G,T,T). Cada elemento foi retirado da primeira estrutura de dados e inserido na segunda estrutura de dados, e a sequência ficou armazenada na seguinte ordem: (T,T,G,A,C,T,G,A). Finalmente, cada elemento foi retirado da segunda estrutura de dados e inserido na terceira estrutura de dados e a sequência ficou armazenada na seguinte ordem: (T,T,G,A,C,T,G,A)”.

Qual a única sequência de estruturas de dados apresentadas a seguir pode ter sido usada no cenário descrito acima?

- A) Fila – Pilha – Fila.
- B) Fila – Fila – Pilha.
- C) Fila – Pilha – Pilha.
- D) Pilha – Fila – Pilha.
- E) Pilha – Pilha – Pilha.

Gabarito oficial: A

1^a → 2^a: os elementos saem da 1^a estrutura na ordem A, G, T, C, A, G, T, T (seja ela pilha com topo à esquerda, seja fila com frente à esquerda) e a 2^a passa a mostrar a sequência *invertida*. Inserir e o último inserido aparecer à esquerda (no topo) é o comportamento de uma **pilha**: a 2^a estrutura é pilha.

2^a → 3^a: os elementos saem da pilha na ordem T, T, G, A, C, T, G, A e a 3^a estrutura mostra exatamente essa ordem, sem inverter: o primeiro inserido fica na frente, ou seja, a 3^a é uma **fila**.

Das alternativas, a única com 2^a = pilha e 3^a = fila é **Fila – Pilha – Fila**.

Questão 10 ENADE 2011 – Tecnologia em Análise e Desenvolvimento de Sistemas – Questão 29

A pilha é uma estrutura de dados que permite a inserção/remoção de itens dinamicamente seguindo a norma de último a entrar, primeiro a sair. Suponha que para uma estrutura de dados, tipo pilha, são definidos os comandos:

- **PUSH (p, n)**: Empilha um número “n” em uma estrutura de dados do tipo pilha “p”;
- **POP (p)**: Desempilha o elemento no topo da pilha.

Considere que, em uma estrutura de dados tipo pilha “p”, inicialmente vazia, sejam executados os seguintes comandos:

```
PUSH (p, 10)
PUSH (p, 5)
PUSH (p, 3)
PUSH (p, 40)
POP (p)
PUSH (p, 11)
PUSH (p, 4)
PUSH (p, 7)
POP (p)
POP (p)
```

Após a execução dos comandos, o elemento no topo da pilha “p” e a soma dos elementos armazenados na pilha “p” são, respectivamente,

- A)** 11 e 29. **B)** 11 e 80. **C)** 4 e 80. **D)** 7 e 29. **E)** 7 e 40.

Gabarito oficial: A

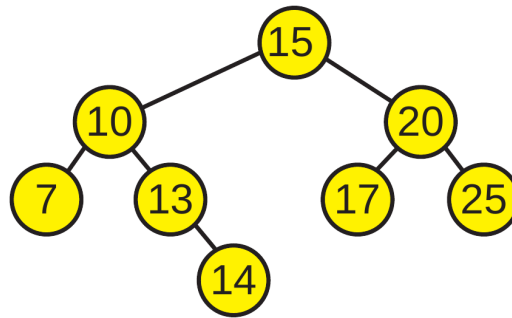
Estado da pilha (topo à direita): $[10] \rightarrow [10, 5] \rightarrow [10, 5, 3] \rightarrow [10, 5, 3, 40] \rightarrow \text{POP } [10, 5, 3] \rightarrow [10, 5, 3, 11] \rightarrow [10, 5, 3, 11, 4] \rightarrow [10, 5, 3, 11, 4, 7] \rightarrow \text{POP } [10, 5, 3, 11, 4] \rightarrow \text{POP } [10, 5, 3, 11]$.
Topo: **11**; soma: $10 + 5 + 3 + 11 = \mathbf{29}$.

2 Árvores e *heaps*

Questão 11

ENADE 2014 – Ciência da Computação (Bacharelado) – Questão 13

A figura a seguir apresenta uma árvore binária de pesquisa, que mantém a seguinte propriedade fundamental: o valor associado à raiz é sempre menor do que o valor de todos os nós da subárvore à direita e sempre maior do que o valor de todos os nós da subárvore à esquerda.



Em relação à árvore apresentada na figura, avalie as afirmações a seguir.

- I. A árvore possui a vantagem de realizar a busca de elementos de forma eficiente, como a busca binária em um vetor.
- II. A árvore está desbalanceada, pois a subárvore da esquerda possui um número de nós maior do que a subárvore da direita.
- III. Quando a árvore é percorrida utilizando o método de caminhamento pós-ordem, os valores são encontrados em ordem decrescente.
- IV. O número de comparações realizadas em função do número n de elementos na árvore em uma busca binária realizada com sucesso é $O(\log n)$.

É correto apenas o que se afirma em

- A) I e III.
- B) I e IV.
- C) II e III.
- D) I, II e IV.
- E) II, III e IV.

Gabarito oficial: B

I – correta. Em cada nó descarta-se uma subárvore inteira, como na busca binária em vetor ordenado.

II – incorreta. Balanceamento se mede pela *altura*, não pelo número de nós. Alturas: subárvore esquerda $(10-13-14) = 3$; direita $(20-17/25) = 2$; diferença 1 na raiz e em todos os demais nós ≤ 1 , logo a árvore está balanceada (no sentido AVL).

III – incorreta. Pós-ordem: 7, 14, 13, 10, 17, 25, 20, 15 — não é decrescente. A ordem decrescente seria obtida por um percurso em-ordem invertido (direita, raiz, esquerda).

IV – correta. Como a árvore é balanceada, sua altura é $O(\log n)$, e uma busca com sucesso faz no máximo altura comparações.

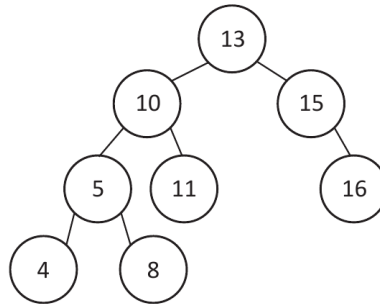
Questão 12

ENADE 2017 – Ciência da Computação (Bacharelado e Licenciatura) – Questão 9

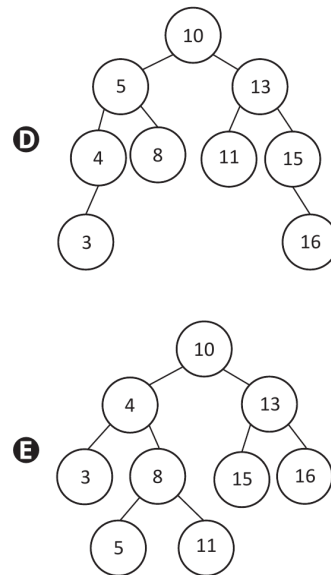
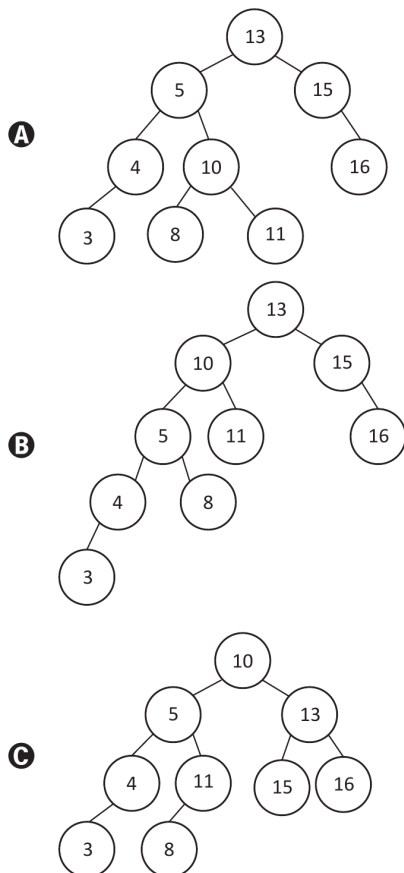
Uma árvore AVL é um tipo de árvore binária balanceada na qual a diferença entre as alturas de suas subárvores da esquerda e da direita não pode ser maior do que 1 para qualquer nó. Após a inserção de um nó em uma AVL, a raiz da subárvore de nível mais baixo no qual o novo nó foi inserido é marcada. Se a altura de seus filhos diferir em mais de uma unidade, é realizada uma rotação simples ou uma rotação dupla para igualar suas alturas.

LAFORÉ, R. **Data structures & algorithms in Java**. Indianópolis: Sams Publishing, 2003 (adaptado).

A seguir, é apresentado um exemplo de árvore AVL.



Pelo exposto no texto acima, após a inserção de um nó com valor 3 na árvore AVL exemplificada, é correto afirmar que ela ficará com a seguinte configuração:



Gabarito oficial: A

O 3 é inserido à esquerda do 4. Recalculando os fatores de balanceamento no caminho de volta: o nó 5 fica com alturas 2 (esq.) e 1 (dir.) — ok; o nó 10 fica com alturas 3 (esq., via 5–4–3) e 1 (dir., nó 11): desbalanceado. O desequilíbrio é do tipo *esquerda-esquerda* (inserção na subárvore esquerda do filho esquerdo), corrigido por uma **rotação simples à direita** em torno do 10: o 5 sobe, o 10 vira seu filho direito e o 8 (antigo filho direito do 5) passa a ser filho esquerdo do 10. Resultado: $13 \rightarrow \{5 \rightarrow \{4 \rightarrow 3, 10 \rightarrow \{8, 11\}\}, 15 \rightarrow 16\}$, que é a alternativa A. A alternativa B é a árvore sem rebalanceamento (não é AVL).

Questão 13

ENADE 2021 – Ciência da Computação (Bacharelado e Licenciatura) – Questão 23

O uso da estrutura de dados tipo Árvore Binária de Busca é uma técnica fundamental de programação. Uma árvore binária é um conjunto finito de elementos que está vazio ou é particionado em três subconjuntos, a saber: 1) raiz da árvore – elemento inicial (único), 2) subárvore da esquerda – se vista isoladamente compõe outra árvore e 3) subárvore da direita – se vista isoladamente compõe outra árvore. A árvore pode não ter qualquer elemento (árvore vazia). A definição de árvore é recursiva e, devido a isso, muitas operações sobre árvores binárias utilizam recursão. Sendo “A” a raiz de uma árvore binária e “B” a raiz de sua subárvore esquerda ou direita, é dito que “A” é pai de “B” e que “B” é filho de “A”. Um elemento sem filhos é chamado de folha. A altura da árvore é o número de elementos encontrados no caminho descendente mais longo que liga a sua raiz até uma folha.

Uma Árvore de Busca Binária é uma árvore binária especializada, na qual a informação que o elemento filho esquerdo possui é numericamente menor que a informação do elemento pai. De forma análoga, a informação que o elemento filho direito possui é numericamente maior ou igual à informação do elemento pai. O objetivo de organizar dados em Árvores Binárias de Busca é facilitar a tarefa de encontrar um determinado elemento. O percurso completo de uma árvore binária consiste em visitar todos os elementos desta árvore, segundo algum critério, a fim de processá-los. Três formas são bem conhecidas para a realização deste percurso: 1) pré-ordem, 2) em-ordem e 3) pós-ordem. A figura a seguir mostra um exemplo de árvore binária.

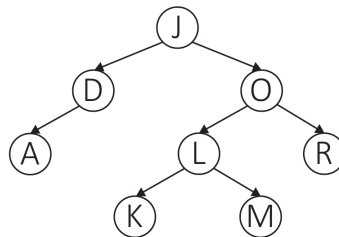


Figura – Exemplo de Árvore Binária

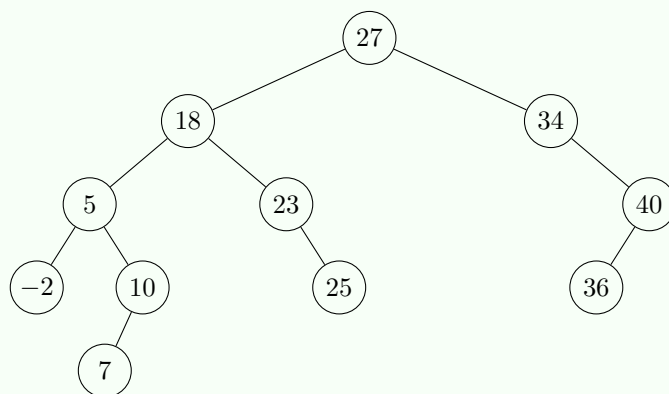
LAUREANO, M. A. P. **Estrutura de Dados com Algoritmos**. São Paulo: Brasport, 2008. p. 126, 129, 136 (adaptado).

Considerando o texto e a figura apresentados e que a seguinte lista de elementos numéricos: (27, 34, 40, 18, 23, 5, 25, 36, 10, 7, -2) seja totalmente transferida para uma estrutura de Árvore Binária de Busca, inicialmente vazia, elemento a elemento, da esquerda para a direita, assinale a alternativa correta.

- A) A árvore resultante terá 5 níveis de altura, com 6 elementos à esquerda da raiz principal (inicial) e 4 elementos à direita.
- B) O percurso da árvore em Pré-ordem irá processar os elementos na seguinte ordem (do primeiro ao último): -2, 7, 10, 5, 25, 23, 18, 36, 40, 34, 27.
- C) O percurso da árvore em Em-ordem irá processar os elementos na seguinte ordem (do primeiro ao último): -2, 5, 7, 10, 18, 23, 25, 27, 34, 36, 40.
- D) O percurso da árvore em Pós-ordem irá processar os elementos na seguinte ordem (do primeiro ao último): 27, 18, 5, -2, 10, 7, 23, 25, 34, 40, 36.
- E) O número máximo de elementos que essa árvore poderá ter com 10 níveis será de 1 024 elementos.

Gabarito oficial: C

Árvore construída:



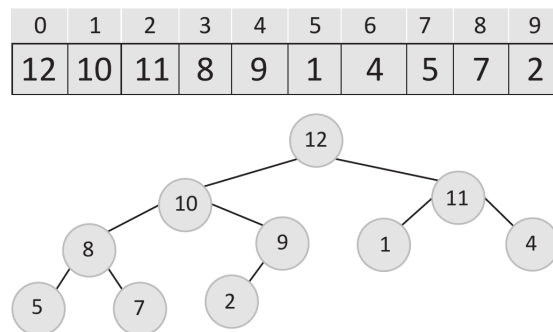
C – correta. O percurso em-ordem de uma ABB sempre produz os elementos em ordem crescente.

A – incorreta: a altura é 5, mas há 7 elementos à esquerda da raiz (18, 5, 23, -2, 10, 25, 7) e 3 à direita (34, 40, 36). B – incorreta: a sequência dada é a *pós*-ordem. A pré-ordem é 27, 18, 5, -2, 10, 7, 23, 25, 34, 40, 36. D – incorreta: a sequência dada é a *pré*-ordem (B e D estão trocadas). E – incorreta: uma árvore binária com 10 níveis tem no máximo $2^{10} - 1 = 1\,023$ nós.

Questão 14

ENADE 2021 – Ciência da Computação (Bacharelado) – Discursiva 5

Um *heap* binário é um arranjo que pode ser visualizado como uma árvore binária, sendo que cada nó da árvore corresponde a um elemento do arranjo, como pode ser observado na figura a seguir.



Percebe-se que existem dois tipos de *heaps*: *heaps* máximo e *heaps* mínimo. O *heap* máximo é uma estrutura de dados que possibilita a consulta ou extração de forma eficiente do maior elemento de uma coleção. A propriedade de *heap* máximo especifica que um nó filho (no código calculado pelas funções *left* e *right*) tem sempre armazenado um valor menor ou igual ao seu pai.

CORMEN, T. H.; LEISERSON, C. E.; RIVEST, R. L.; STEIN, C. **Introduction to Algorithms**. 3. ed. MIT Press and McGraw-Hill. p. 131-161, 2009 (adaptado).

Considerando a implementação a seguir, o *heapify* é uma função auxiliar para reorganizar o arranjo (garantindo a propriedade de *heap* máximo em uma determinada posição do arranjo) e *buildHeap* é uma função que usa *heapify* para reorganizar todas as posições do arranjo (garantindo a propriedade de *heap* máximo para todos os elementos).

```

int left(int i) { return (2 * i + 1); }
int right(int i) { return (2 * i + 2); }
/* a - arranjo, n - número de elementos,
   i - posição do elemento que deve ser
   colocado em propriedade de heap */
void heapify (int *a, int n, int i)
{
    int e, d, max, aux;

    e = left(i);
    d = right(i);
    if (e < n && a[e] > a[i])
        max = e;
    else
        max = i;
    if (d < n && a[d] > a[max])
        max = d;
    if (max != i)
    {
        aux = a[i];
        a[i] = a[max];
        a[max] = aux;
        heapify(a, n, max);
    }
}
/* a - arranjo, n - número de elementos */
void buildHeap(int *a, int n)
{
    int i;

```

```
for (i = (n-1)/2; i >= 0; i--)  
    heapify(a, n, i);  
}
```

De acordo com as informações apresentadas, faça o que se pede nos itens a seguir.

- a) Como ficará o arranjo `int a[ ] = {2, 5, 8, 13, 21, 1, 3, 34}` após a execução da função `buildHeap(a, 8)`. (valor: 5,0 pontos)
- b) Apresente a complexidade de tempo no pior caso para a função `heapify`, use a notação O ou Θ . (valor: 5,0 pontos)

Padrão de resposta oficial (INEP)

a) {34, 21, 8, 13, 2, 1, 3, 5}.

b) $O(\log n)$, sendo n o número de elementos do *heap*.

Justificativa (rastreamento). $n = 8$, o laço começa em $i = (8 - 1)/2 = 3$.

- $i = 3$ (13): filhos $a[7] = 34 \rightarrow$ troca: {2, 5, 8, 34, 21, 1, 3, 13}.
- $i = 2$ (8): filhos 1 e 3 $\rightarrow$ nada muda.
- $i = 1$ (5): filhos 34 e 21 $\rightarrow$ troca com 34: {2, 34, 8, 5, 21, 1, 3, 13}; desce em $i = 3$ (5): filho 13 $\rightarrow$ troca: {2, 34, 8, 13, 21, 1, 3, 5}.
- $i = 0$ (2): filhos 34 e 8 $\rightarrow$ troca com 34: {34, 2, 8, 13, 21, 1, 3, 5}; desce em $i = 1$ (2): filhos 13 e 21 $\rightarrow$ troca com 21: {34, 21, 8, 13, 2, 1, 3, 5}; em $i = 4$ não há filhos.

Em (b), cada chamada recursiva de `heapify` desce um nível da árvore, que é completa e tem altura $\lfloor \log_2 n \rfloor$; logo, no pior caso, $O(\log n)$.

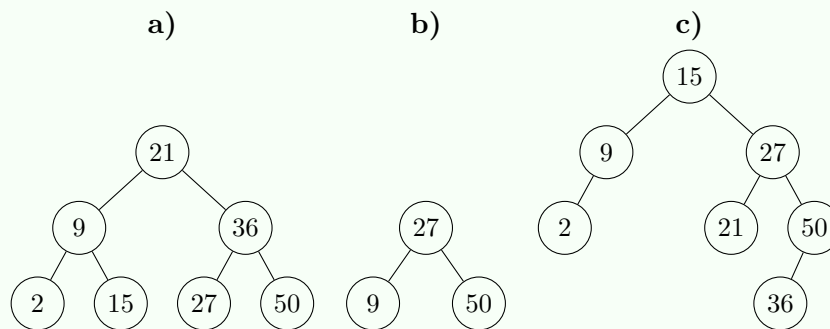
Questão 15

ENADE 2019 – Engenharia de Computação – Discursiva 3

Uma árvore binária de busca é uma árvore ordenada que pode apresentar prejuízos no desempenho de determinados algoritmos em função do desbalanceamento causado pela ordem de inserção dos elementos na estrutura. Uma árvore AVL é uma árvore binária de busca balanceada em que a diferença em módulo entre a altura da subárvore esquerda e a altura da subárvore direita de cada nó é, no máximo, de uma unidade.

Nesse contexto, faça o que se pede nos itens a seguir.

- Apresente uma árvore binária de busca balanceada com os elementos 2, 9, 15, 21, 27, 36 e 50 em que o nó raiz principal contém o elemento 21 e o balanceamento de cada nó seja no máximo uma unidade. (valor: 3,0 pontos)
- Considerando as inserções dos elementos 9, 27 e 50, nesta ordem, em uma árvore AVL inicialmente vazia, apresente a árvore resultante. (valor: 3,0 pontos)
- Considerando as inserções dos elementos 9, 27, 50, 15, 2, 21 e 36, nesta ordem, em uma árvore AVL inicialmente vazia, apresente a árvore resultante. (valor: 4,0 pontos)

Padrão de resposta oficial (INEP)

Justificativa (elaborada). a) Com 21 na raiz, os menores (2, 9, 15) ficam à esquerda com 9 no meio, e os maiores (27, 36, 50) à direita com 36 no meio: árvore completa, todos os fatores iguais a 0.

b) Inserindo 9 e 27 (27 à direita de 9) e depois 50 (à direita de 27), o nó 9 fica com fator -2 (caso direita-direita): **rotação simples à esquerda** em 9, e 27 vira a raiz.

c) Partindo de (b): 15 entra à direita de 9 e 2 à esquerda de 9, sem desbalancear. O 21 entra à direita de 15; agora o nó 27 tem altura 3 à esquerda (9-15-21) e 1 à direita (50): desbalanceamento do tipo *esquerda-direita*, corrigido por **rotação dupla** (à esquerda em 9 e à direita em 27), e 15 sobe para a raiz, com 9 (filho 2) à esquerda e 27 (filhos 21 e 50) à direita. Por fim, 36 entra à esquerda de 50 sem violar a propriedade AVL.

Questão 16

ENADE 2008 – Computação (CC, EC e SI) – Questão 14

Um programador propôs um algoritmo não-recursivo para o percurso em preordem de uma árvore binária com as seguintes características.

- Cada nó da árvore binária é representado por um registro com três campos: **chave**, que armazena seu identificador; **esq** e **dir**, ponteiros para os filhos esquerdo e direito, respectivamente.
- O algoritmo deve ser invocado inicialmente tomando o ponteiro para o nó raiz da árvore binária como argumento.
- O algoritmo utiliza **push()** e **pop()** como funções auxiliares de empilhamento e desempilhamento de ponteiros para nós de árvore binária, respectivamente.

A seguir, está apresentado o algoritmo proposto, em que λ representa o ponteiro nulo.

```
Procedimento preordem (ptraiiz : PtrNoArvBin)
  Var ptr : PtrNoArvBin;
  ptr := ptraiiz;
  Enquanto (ptr  $\neq$   $\lambda$ ) Faça
    escreva (ptr↑.chave);
    Se (ptr↑.dir  $\neq$   $\lambda$ ) Então
      push(ptr↑.dir);
    Se (ptr↑.esq  $\neq$   $\lambda$ ) Então
      push(ptr↑.esq);
    ptr := pop();
  Fim_Enquanto
Fim_Procedimento
```

Com base nessas informações e supondo que a raiz de uma árvore binária com n nós seja passada ao procedimento **preordem()**, julgue os itens seguintes.

- I. O algoritmo visita cada nó da árvore binária exatamente uma vez ao longo do percurso.
- II. O algoritmo só funcionará corretamente se o procedimento **pop()** for projetado de forma a retornar λ caso a pilha esteja vazia.
- III. Empilhar e desempilhar ponteiros para nós da árvore são operações que podem ser implementadas com custo constante.
- IV. A complexidade do pior caso para o procedimento **preordem()** é $O(n)$.

Assinale a opção correta.

- A) Apenas um item está certo.
- B) Apenas os itens I e IV estão certos.
- C) Apenas os itens I, II e III estão certos.
- D) Apenas os itens II, III e IV estão certos.
- E) Todos os itens estão certos.

Gabarito oficial: E

I – correta. Cada nó é empilhado uma única vez (pelo pai) e, ao ser desempilhado, é impresso; como a árvore não tem ciclos, nenhum nó volta à pilha.

II – correta. O laço só termina quando **ptr** vale λ , o que acontece quando a pilha se esvazia; se **pop()** em pilha vazia não devolver λ , o laço não termina ou acessa memória inválida.

III – correta. Numa pilha em vetor ou lista encadeada, **push/pop** são $O(1)$.

IV – correta. São n iterações, cada uma com trabalho constante: $O(n)$.

Note a ordem dos **push**: o filho *direito* é empilhado antes do esquerdo, para que o esquerdo seja desempilhado primeiro — garantindo a ordem raiz, esquerda, direita.

Questão 17

ENADE 2011 – Computação (todas as áreas) – Questão 24

As filas de prioridades (*heaps*) são estruturas de dados importantes no projeto de algoritmos. Em especial, *heaps* podem ser utilizados na recuperação de informação em grandes bases de dados constituídos por textos. Basicamente, para se exibir o resultado de uma consulta, os documentos recuperados são ordenados de acordo com a relevância presumida para o usuário. Uma consulta pode recuperar milhões de documentos que certamente não serão todos examinados. Na verdade, o usuário examina os primeiros m documentos dos n recuperados, em que m é da ordem de algumas dezenas.

Considerando as características dos *heaps* e sua aplicação no problema descrito acima, avalie as seguintes afirmações.

- I. Uma vez que o *heap* é implementado como uma árvore binária de pesquisa essencialmente completa, o custo computacional para sua construção é $O(n \log n)$.
- II. A implementação de *heaps* utilizando-se vetores é eficiente em tempo de execução e em espaço de armazenamento, pois o pai de um elemento armazenado na posição i se encontra armazenado na posição $2i + 1$.
- III. O custo computacional para se recuperar de forma ordenada os m documentos mais relevantes armazenados em um *heap* de tamanho n é $O(m \log n)$.
- IV. Determinar o documento com maior valor de relevância armazenado em um *heap* tem custo computacional $O(1)$.

Está correto apenas o que se afirma em

- A) I e II. B) II e III. C) III e IV. D) I, II e IV. E) I, III e IV.

Gabarito oficial: C

I – incorreta. *Heap* não é árvore binária de *pesquisa* (a ordem é só entre pai e filhos, não esquerda < raiz < direita). Além disso, construir um *heap* com *buildHeap* custa $O(n)$, não $O(n \log n)$.

II – incorreta. Na representação em vetor (índices a partir de 1), os *filhos* de i ficam em $2i$ e $2i + 1$, e o *pai* em $\lfloor i/2 \rfloor$ (a partir de 0: filhos $2i + 1$, $2i + 2$; pai $\lfloor (i - 1)/2 \rfloor$).

III – correta. Cada uma das m extrações do máximo custa $O(\log n)$.

IV – correta. Num *heap* máximo, o maior elemento está na raiz (posição inicial do vetor): $O(1)$.

Assim, para $m \ll n$, construir o *heap* ($O(n)$) e extrair m elementos ($O(m \log n)$) é bem mais barato do que ordenar tudo ($O(n \log n)$).

Questão 18

ENADE 2011 – Computação (todas as áreas) – Questão 30

Suponha que se queira pesquisar a chave 287 em uma árvore binária de pesquisa com chaves entre 1 e 1 000. Durante uma pesquisa como essa, uma sequência de chaves é examinada. Cada sequência abaixo é uma suposta sequência de chaves examinadas em uma busca da chave 287.

I. 7, 342, 199, 201, 310, 258, 287

II. 110, 132, 133, 156, 289, 288, 287

III. 252, 266, 271, 294, 295, 289, 287

IV. 715, 112, 530, 249, 406, 234, 287

É válido apenas o que se apresenta em

A) I.

B) III.

C) I e II.

D) II e IV.

E) III e IV.

Gabarito oficial: C

Numa busca em ABB, cada chave examinada estreita o intervalo (inf,sup) em que 287 pode estar: se a chave é menor que 287, ela vira o novo limite inferior; se é maior, o novo limite superior. Toda chave seguinte tem de cair dentro do intervalo atual.

- **I – válida:** 7 (inf=7), 342 (sup=342), 199 (inf), 201 (inf), 310 (sup), 258 (inf), 287. Todas dentro dos limites.
- **II – válida:** 110, 132, 133, 156 (inf sobe), 289 (sup), 288 (sup), 287.
- **III – inválida:** após 294 o limite superior é 294, e a chave seguinte, 295, está fora dele.
- **IV – inválida:** após 249 o limite inferior é 249 (e o superior, 406); a chave 234 é menor que 249.

Questão 19

ENADE 2011 – Computação (todas as áreas) – Discursiva 4

Listas ordenadas implementadas com vetores são estruturas de dados adequadas para a busca binária, mas possuem o inconveniente de exigirem custo computacional de ordem linear para a inserção de novos elementos. Se as operações de inserção ou remoção de elementos forem frequentes, uma alternativa é transformar a lista em uma árvore binária de pesquisa balanceada, que permitirá a execução dessas operações com custo logarítmico.

Considerando essas informações, escreva um algoritmo recursivo que construa uma árvore binária de pesquisa completa, implementada por estruturas auto-referenciadas ou apontadores, a partir de um vetor ordenado, v , de n inteiros, em que $n = 2^m - 1$, $m > 0$. O algoritmo deve construir a árvore em tempo linear, sem precisar fazer qualquer comparação entre os elementos do vetor, uma vez que este já está ordenado. Para isso,

- descreva a estrutura de dados utilizada para a implementação da árvore (valor: 2,0 pontos);
- escreva o algoritmo para a construção da árvore. A chamada principal à função recursiva deve passar, como parâmetros, o vetor, índice do primeiro e último elementos, retornando a referência ou apontador para a raiz da árvore criada (valor: 8,0 pontos).

Observação: Qualquer notação em português estruturado, de forma imperativa ou orientada a objetos deve ser considerada, assim como em uma linguagem de alto nível, como o Pascal, C e Java.

Padrão de resposta oficial (INEP)

- a) Registro **nodo** com campos: **chave** do tipo inteiro; **esq** e **dir** do tipo apontadores para registro **nodo**. O importante é a presença desses campos e do uso de apontadores ou autorreferências.
- b)

```
função CriaABP(v: vetor de inteiros; i, j: inteiros) retorna apontador para registro
    nodo
// i, j são os índices inicial e final do vetor
| Cria novo nodo apontado por p, o qual deve ser uma variável local
| pos = (i + j) / 2           // determina a posição central do vetor
| p->chave = v[pos]          // guarda o elemento v[pos] no novo nodo criado
| se i < j então              // ainda não se está no nível das folhas
| | p->esq = CriaABP(v, i, pos-1) // subárvore da esquerda
| | p->dir = CriaABP(v, pos+1, j) // subárvore da direita
| senão p->esq = p->dir = NULL // nível das folhas
| retorna p
```

Chamada principal: raiz = CriaABP(v, 1, n)

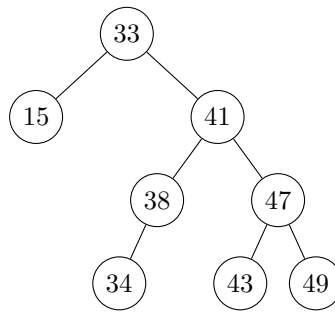
A função deve ser recursiva e não pode usar comparações para encontrar o elemento a ser inserido, nem utilizar operações de inserção que façam comparações implicitamente. A condição de parada da recursão (nível das folhas) deve estar clara, os parâmetros das chamadas recursivas devem estar corretos e os apontadores dos nodos-folhas devem ser aterrados.

Observação

Cada elemento do vetor gera exatamente um nó, com trabalho constante por chamada: $T(n) = 2T(n/2) + O(1) = O(n)$. Como $n = 2^m - 1$, todo intervalo $[i, j]$ tem tamanho ímpar e a árvore resultante é completa; para n qualquer, basta trocar o caso base por “se $i > j$, retorna NULL”.

Questão 20

ENADE 2005 – Computação (Sistemas de Informação) – Discursiva 39



Tendo como base a árvore acima, faça o que se pede nos itens a seguir.

- Descreva uma ordem de visita dos nós para uma busca em profundidade a partir do nó de valor 41. (valor: 3,0 pontos)
- Considerando que o nó de valor 33 seja a raiz da árvore, descreva a ordem de visita para uma varredura em pré-ordem (r-e-d, ou pré-fixado à esquerda) na árvore. (valor: 3,0 pontos)
- Considerando que a árvore cuja raiz é o nó de valor 33 represente uma árvore de busca binária, desenhe a nova árvore que será obtida após a realização das seguintes operações: inserir um nó de valor 21; remover o nó de valor 47; inserir um nó de valor 48. (valor: 4,0 pontos)

Padrão de resposta oficial (INEP)

a) Foram aceitas respostas que consideram a estrutura uma árvore ou um grafo.

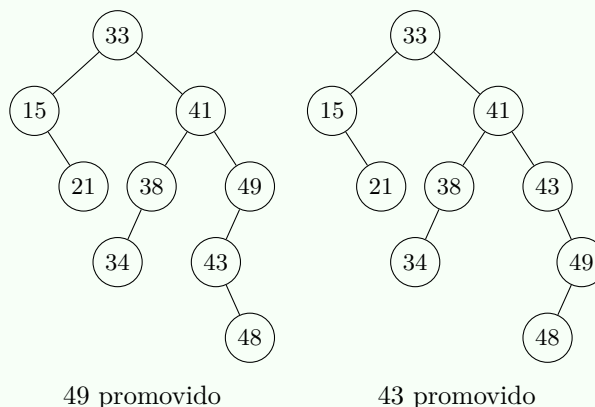
- Considerando um grafo (o 41 tem como vizinho também o 33): 41, 38, 34, 47, 49, 43, 33, 15 ou 15, 33, 34, 38, 49, 43, 47, 41.
- Considerando uma árvore (só a subárvore do 41): 41, 38, 34, 47, 43, 49 ou 41, 47, 49, 43, 38, 34 ou 34, 38, 43, 49, 47, 41 ou 49, 43, 47, 34, 38, 41.

Qualquer percurso que obedeça a uma das duas formas da busca em profundidade (visitar o nó antes ou depois de explorar seus vizinhos) está correto.

b) 33, 15, 41, 38, 34, 47, 43, 49.

c)

- insere(21):** única possibilidade — à direita do 15.
- remove(47):** o 49 (sucessor) ou o 43 (antecessor) é promovido para o lugar do 47.
- insere(48):** depende da escolha anterior — se o 49 subiu, o 48 entra à direita do 43; se o 43 subiu, entra à esquerda do 49.



O padrão também aceitou uma árvore rebalanceada (por exemplo, com 48 no lugar do 47, tendo 43 e 49 como filhos).

3 Tabelas de dispersão (*hash*)

Questão 21

ENADE 2021 – Sistemas de Informação – Questão 33

Uma abordagem calcula a posição da chave na tabela com base no valor da chave. Este valor é a única indicação da posição. Quando a chave é conhecida, a posição na tabela pode ser acessada diretamente, sem fazer qualquer outro teste preliminar, conforme exigido na busca binária ou durante a pesquisa em uma árvore. Precisamos encontrar uma função h que possa transformar uma chave particular K — seja ela uma cadeia de caracteres, um número, um registro etc. — em um índice na tabela usada para armazenar itens do mesmo tipo que K . A função h é chamada função de escrutínio (*hash*) e não retorna valores únicos. Por exemplo, $h(\text{"abc"}) = h(\text{"acb"})$. Este problema é chamado colisão.

DROZDEK, A. **Estrutura de Dados e Algoritmos em C++**. São Paulo: Cengage do Brasil, p.473, 2016 (adaptado).

Uma tabela na qual os dados são inseridos em entradas determinadas por uma função h , conforme descrito no texto, é chamada de tabela de dispersão ou tabela de espalhamento. A tabela de dispersão é representada por meio do vetor `tabelaH` de T posições, no qual cada posição é uma estrutura `ItemDado` e o procedimento de inserção em `tabelaH` está definido a seguir.

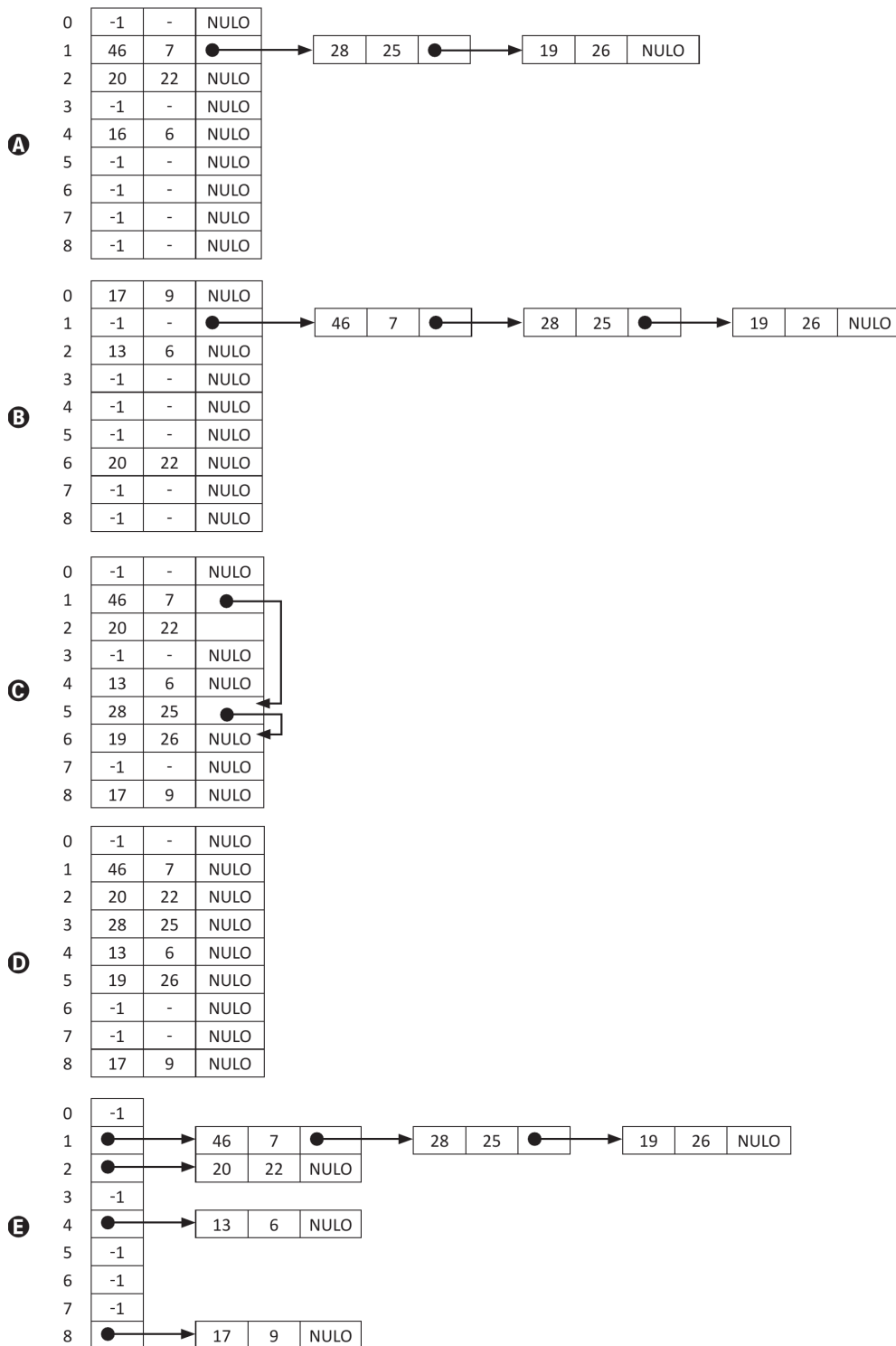
```
ItemDado:
  Chave: inteiro
  Dado: inteiro
  Prox: ItemDado

procedimento insere(chave, dado: inteiro)
início
  item, temp: ItemDado
  índice: inteiro
  índice <- h(chave)
  se tabelaH[índice].chave = -1 então
    tabelaH[índice].chave <- chave
    tabelaH[índice].dado <- dado
    tabelaH[índice].prox <- NULO
  senão
    item.chave <- chave
    item.dado <- dado
    item.prox <- NULO
    se tabelaH[índice].prox = NULO então
      tabelaH[índice].prox <- item
    senão
      temp <- tabelaH[índice].prox
      enquanto temp.prox ≠ NULO faça
        temp <- temp.prox
      fimenquanto
      temp.prox <- item
    fimse
  fimse
fimprocedimento
```

O valor -1 no campo chave de uma entrada na tabela significa que aquela entrada está livre; o campo `Prox` com o valor `NULO` significa que ele não aponta para nenhum item de dado. A função $h(\text{chave})$ simplesmente retorna o valor do resto da divisão inteira de chave por T .

Considerando as informações apresentadas e supondo que $T = 9$, assinale a opção que corresponde ao estado de `tabelaH` após a seguinte sequência de chamadas: `insere(13, 6)`; `insere(46, 7)`;

insere(20, 22); insere(28, 25); insere(19, 26); insere(17, 9).



Gabarito oficial: ANULADA

Calculando $h(k) = k \bmod 9$: $13 \rightarrow 4$; $46 \rightarrow 1$; $20 \rightarrow 2$; $28 \rightarrow 1$ (colisão); $19 \rightarrow 1$ (colisão); $17 \rightarrow 8$. Pelo procedimento, a primeira chave de cada índice fica na própria tabela e as colisões são encadeadas no fim da lista daquele índice. O estado correto seria:

índice	conteúdo
1	$(46, 7) \rightarrow (28, 25) \rightarrow (19, 26) \rightarrow \text{NULO}$
2	$(20, 22) \rightarrow \text{NULO}$
4	$(13, 6) \rightarrow \text{NULO}$
8	$(17, 9) \rightarrow \text{NULO}$
0, 3, 5, 6, 7	livres (−1)

Nenhuma alternativa corresponde exatamente a esse estado: a mais próxima, A, traz a chave 16 (e não 13) no índice 4 e omite (17, 9) no índice 8; B usa índices errados; C e D tratam as colisões por endereçamento aberto; E representa a tabela como vetor de ponteiros, o que não corresponde à estrutura `ItemDado` armazenada diretamente em `tabelaH`. Por isso a questão foi anulada. Ela continua útil em sala com a pergunta “qual é o estado correto da tabela?”.

Questão 22

ENADE 2008 – Computação (CC, EC e SI) – Discursiva 20

Tabelas de dispersão (tabelas *hash*) armazenam elementos com base no valor absoluto de suas chaves e em técnicas de tratamento de colisões. As funções de dispersão transformam chaves em endereços-base da tabela, ao passo que o tratamento de colisões resolve conflitos em casos em que mais de uma chave é mapeada para um mesmo endereço-base da tabela.

Suponha que uma aplicação utilize uma tabela de dispersão com 23 endereços-base (índices de 0 a 22) e empregue $h(x) = x \bmod 23$ como função de dispersão, em que x representa a chave do elemento cujo endereço-base deseja-se computar. Inicialmente, essa tabela de dispersão encontra-se vazia. Em seguida, a aplicação solicita uma sequência de inserções de elementos cujas chaves aparecem na seguinte ordem: 44, 46, 49, 70, 27, 71, 90, 97, 95.

Com relação à aplicação descrita, faça o que se pede a seguir.

- A) Escreva, no espaço reservado, o conjunto das chaves envolvidas em colisões. (valor: 4,0 pontos)
- B) Assuma que a tabela de dispersão trate colisões por meio de encadeamento exterior. Esboce a tabela de dispersão para mostrar seu conteúdo após a sequência de inserções referida. (valor: 6,0 pontos)

Padrão de resposta oficial (INEP)

A) Colisões ocorrerão entre elementos cujas chaves forem mapeadas pela função de dispersão $h(x) = x \bmod 23$ no mesmo endereço-base. Portanto, o conjunto de chaves envolvidas em colisões é $\{44, 49, 95, 90\}$.

B) Configuração da tabela (as entradas de 6 a 20 e a 22 têm ponteiros nulos):

índice	lista encadeada
0	46
1	70
2	71
3	49 → 95
4	27
5	97
6 a 20	(vazias)
21	44 → 90
22	(vazia)

Cálculos: $44 \bmod 23 = 21$; $46 \rightarrow 0$; $49 \rightarrow 3$; $70 \rightarrow 1$; $27 \rightarrow 4$; $71 \rightarrow 2$; $90 \rightarrow 21$; $97 \rightarrow 5$; $95 \rightarrow 3$.

4 Busca e ordenação

Questão 23

ENADE 2014 – Sistemas de Informação – Discursiva 3

Uma aplicação mantém um grande conjunto de dados ordenados em uma lista linear em memória, implementada na forma de *array*. Os dados são estáveis, ou seja, são realizadas poucas operações de inserção e remoção, e a aplicação exige grande quantidade de pesquisas. Sabe-se que vários algoritmos permitem realizar pesquisas em um conjunto de dados, sendo alguns mais eficientes que outros. Nesse sentido, a pesquisa binária apresenta características bem definidas para se encontrar um dado procurado.

Considerando o contexto apresentado, desenvolva um algoritmo de pesquisa binária, de forma imperativa ou orientada a objetos, utilizando qualquer notação em português estruturado ou em uma linguagem de alto nível, como o Pascal, C ou Java. (valor: 10,0 pontos)

Padrão de resposta oficial (INEP)

O estudante deve apresentar um algoritmo com pesquisa binária. A seguir são apresentados dois exemplos de resposta escritos em linguagem C: um utilizando a abordagem iterativa e outro utilizando a abordagem recursiva.

Solução iterativa:

```
// x: valor a ser pesquisado, n: tamanho do vetor, v: vetor de valores
// Se encontrar retorna o índice do vetor, senão retorna -1
int buscaBinaria(int x, int n, int v[]) {
    int e, m, d;
    e = 0; d = n - 1;
    while (e <= d) {
        m = (e + d) / 2;
        if (v[m] == x)
            return m;
        if (v[m] < x)
            e = m + 1;
        else
            d = m - 1;
    }
    return -1;
}
```

Solução recursiva:

```
// x: valor a ser pesquisado, e: índice do início da pesquisa,
// d: índice do fim da pesquisa, v: vetor de valores
// Se encontrar retorna o índice do vetor, senão retorna -1
int buscaBinaria(int x, int e, int d, int v[]) {
    int m;
    if (e <= d) {
        m = (e + d) / 2;
        if (v[m] == x)
            return m;
        if (v[m] < x)
            return buscaBinaria(x, m + 1, d, v);
        else
            return buscaBinaria(x, e, m - 1, v);
    }
    else return -1;
}
```

Complexidade: $O(\log n)$ comparações, contra $O(n)$ da busca sequencial — vantajoso justamente porque os dados estão ordenados e mudam pouco.

Questão 24

ENADE 2014 – Sistemas de Informação – Questão 10

A área de complexidade de algoritmos abrange a medição da eficiência de um algoritmo em função da quantidade de operações realizadas até que ele encontre seu resultado final.

A respeito desse contexto, suponha que um arquivo de texto contenha os nomes de N cidades de determinado estado, em que cada nome de cidade esteja separado do seguinte por um caractere especial de fim de linha e classificado em ordem alfabética crescente. Considere um programa que leia linha a linha esse arquivo, em busca do nome de uma cidade.

Com base nessa descrição, verifica-se que a complexidade desse programa é

- A) $O(1)$, em caso de busca sequencial.
- B) $O(N)$, em caso de busca sequencial.
- C) $O(\log_2 N)$, em caso de busca binária.
- D) $O(N)$, em caso de transferência dos nomes para uma árvore binária e, então, realizar a busca.
- E) $O(\log_2 N)$, em caso de transferência dos nomes para uma árvore binária e, então, realizar a busca.

Gabarito oficial: B

O programa lê o arquivo *linha a linha*: não há acesso direto à k -ésima linha (as linhas têm tamanhos variáveis), então é uma busca sequencial, que, no pior caso, examina as N linhas: $O(N)$. A busca binária (C) exigiria acesso aleatório. Em D e E, apenas carregar os N nomes numa árvore já custa pelo menos $O(N)$ (e, com os nomes já ordenados, uma ABB simples degenera em lista, levando a $O(N^2)$ para construí-la), de modo que o custo total não seria $O(\log N)$.

Questão 25

ENADE 2021 – Ciência da Computação (Bacharelado e Licenciatura) – Questão 20

Observe o código abaixo escrito na linguagem C.

```
1 #include <stdio.h>
2 #define TAM 10
3 int funcao1(int vetor[], int v){
4     int i;
5     for (i = 0; i < TAM; i++){
6         if (vetor[i] == v)
7             return i;
8     }
9     return -1;
10 }
11 int funcao2(int vetor[], int v, int i, int f){
12     int m = (i + f) / 2;
13     if (v == vetor[m])
14         return m;
15     if (i >= f)
16         return -1;
17     if (v > vetor[m])
18         return funcao2(vetor, v, m+1, f);
19     else
20         return funcao2(vetor, v, i, m-1);
21 }
22 int main(){
23     int vetor[TAM] = {1, 3, 5, 7, 9, 11, 13, 15, 17, 19};
24     printf("%d - %d", funcao1(vetor, 15), funcao2(vetor, 15, 0, TAM-1));
25     return 0;
26 }
```

A respeito das funções implementadas, avalie as afirmações a seguir.

- I. O resultado da impressão na linha 24 é: 7 - 7.
- II. A função `funcao1`, no pior caso, é uma estratégia mais rápida do que a `funcao2`.
- III. A função `funcao2` implementa uma estratégia iterativa na concepção do algoritmo.

É correto o que se afirma em

- A) I, apenas.
- B) III, apenas.
- C) I e II, apenas.
- D) II e III, apenas.
- E) I, II e III.

Gabarito oficial: A

I – correta. `funcao1` (busca sequencial) encontra 15 no índice 7. `funcao2` (busca binária recursiva): $(0, 9) \rightarrow m = 4$ ($9 < 15$) $\rightarrow (5, 9) \rightarrow m = 7$, `vetor[7] = 15` $\rightarrow$ retorna 7. Impressão: 7 - 7.

II – incorreta. No pior caso a busca sequencial é $O(n)$ e a binária, $O(\log n)$.

III – incorreta. `funcao2` é *recursiva* (chama a si mesma), não iterativa.

Observação

No caderno original, a fonte tipográfica torna ambíguos os caracteres “1” e “l” em `funcao1` (linha 3) e em `m+1` (linha 18). Foram transcritos aqui como `funcao1` e `m+1`, que é a leitura coerente com a chamada da linha 24 e com o gabarito.

Questão 26

ENADE 2017 – Ciência da Computação (Bacharelado e Licenciatura) – Questão 18

O algoritmo a seguir recebe um vetor *v* de números inteiros e rearranja esse vetor de tal forma que seus elementos, ao final, estejam ordenados de forma crescente.

```
01 void ordena(int *v, int n)
02 {
03     int i, j, chave;
04     for(i = 1; i < n; i++)
05     {
06         chave = v[i];
07         j = i - 1;
08         while(j >= 0 && v[j] < chave)
09         {
10             v[j-1] = v[j];
11             j = j - 1;
12         }
13         v[j+1] = chave;
14     }
15 }
```

Considerando que nesse algoritmo há erros de lógica que devem ser corrigidos para que os elementos sejam ordenados de forma crescente, assinale a opção correta no que se refere às correções adequadas.

- A) A linha 04 deve ser corrigida da seguinte forma: `for(i = 1; i < n - 1; i++)` e a linha 13, do seguinte modo: `v[j - 1] = chave;`.
- B) A linha 04 deve ser corrigida da seguinte forma: `for(i = 1; i < n - 1; i++)` e a linha 07, do seguinte modo: `j = i + 1;`.
- C) A linha 07 deve ser corrigida da seguinte forma: `j = i + 1` e a linha 08, do seguinte modo: `while(j >= 0 && v[j] > chave)`.
- D) A linha 08 deve ser corrigida da seguinte forma: `while(j >= 0 && v[j] > chave)` e a linha 10, do seguinte modo: `v[j + 1] = v[j];`.
- E) A linha 10 deve ser corrigida da seguinte forma: `v[j + 1] = v[j];` e a linha 13, do seguinte modo: `v[j - 1] = chave;`.

Gabarito oficial: D

O código é o *insertion sort* com dois erros:

- Linha 08: para ordem *crescente*, os elementos maiores que a chave é que devem ser deslocados; a condição correta é `v[j] > chave` (com `<` a ordenação seria decrescente).
- Linha 10: o deslocamento abre espaço à direita, `v[j+1] = v[j];` `v[j-1]` sobrescreve o elemento errado e acessa `v[-1]` quando `j = 0`.

As linhas 04, 07 e 13 já estão corretas: o laço externo deve ir até *n-1* (inclusive), *j* começa em *i-1* e a chave é colocada em `v[j+1]`.

Questão 27

ENADE 2021 – Ciência da Computação (Bacharelado) – Questão 32

Existe um grande número de implementações para algoritmos de ordenação. Um dos fatores a serem considerados, por exemplo, é o número máximo e médio de comparações que são necessárias para ordenar um vetor com n elementos. Diz-se também que um algoritmo de ordenação é estável se ele preserva a ordem de elementos que são iguais. Isto é, se tais elementos aparecem na sequência ordenada na mesma ordem em que estão na sequência inicial. Analise o algoritmo abaixo, onde A é um vetor e “ i , j , lo e hi ” são índices do vetor:

```
algoritmo ordena(A, lo, hi)
    se lo < hi então
        p := particao(A, lo, hi)
        ordena(A, lo, p - 1)
        ordena(A, p + 1, hi)

algoritmo particao(A, lo, hi)
    pivot := A[hi]
    i := lo
    repita para j := lo até hi
        se A[j] < pivot então
            troca A[i] com A[j]
            i := i + 1
    troca A[i] com A[hi]
    return i
```

Com relação ao algoritmo apresentado, avalie as afirmações a seguir.

- I. O algoritmo precisa de um espaço adicional $O(n)$ para a pilha de recursão.
- II. O algoritmo apresentado é um algoritmo de ordenação recursivo e estável.
- III. O algoritmo precisa, em média, de $O(n \log n)$ comparações para ordenar n itens.
- IV. O uso do primeiro elemento do vetor como “pivot” é mais eficiente que usar o último.

É correto apenas o que se afirma em

- A) I e III.
- B) II e IV.
- C) III e IV.
- D) I, II e III.
- E) I, II e IV.

Gabarito oficial: A

Trata-se do *quicksort* com partição de Lomuto (pivô = último elemento).

I – correta. No pior caso (por exemplo, vetor já ordenado), cada partição separa apenas um elemento e a recursão atinge profundidade n : pilha $O(n)$.

II – incorreta. É recursivo, mas *não* estável: as trocas de longa distância podem inverter a ordem relativa de elementos iguais.

III – correta. No caso médio, o *quicksort* faz $\Theta(n \log n)$ comparações.

IV – incorreta. Escolher o primeiro ou o último elemento é equivalente em eficiência: ambos degeneram para $O(n^2)$ em entradas ordenadas. Melhorias reais vêm de pivô aleatório ou mediana de três.

Questão 28

ENADE 2019 – Engenharia de Computação – Questão 25

A linguagem Python não permite alguns tipos de otimização como, por exemplo, a recursão em cauda e, devido à sua natureza dinâmica, é impossível realizar esse tipo de otimização em tempo de compilação tal como em linguagens funcionais como Haskell ou ML.

Disponível em: <<http://www.python-history.blogspot.com/2009/04/origins-of-pythons-functional-features.html>>. Acesso em: 15 jun. 2019 (adaptado).

O trecho de código a seguir, escrito em Python, realiza a busca binária de um elemento x em uma lista `lst` e a função `binary_search` tem código recursivo em cauda.

```
1 def binary_search(x, lst, low=None, high=None):
2     if low == None : low = 0
3     if high == None : high = len(lst)-1
4     mid = low + (high - low) // 2
5     if low > high :
6         return None
7     elif lst[mid] == x :
8         return mid
9     elif lst[mid] > x :
10        return binary_search(x, lst, low, mid-1)
11    else :
12        return binary_search(x, lst, mid+1, high)
```

Disponível em: <<https://www.kylem.net/programming/tailcall.html>>. Acesso em: 15 jun. 2019 (adaptado).

Considerando esse trecho de código, avalie as afirmações a seguir.

- I. Substituindo-se o conteúdo da linha 10 por `high = mid - 1` e substituindo-se o conteúdo da linha 12 por `low = mid + 1`, não se altera o resultado de uma busca.
- II. Envolvendo-se o código das linhas 4 a 12 em um laço `while True`, substituindo-se o conteúdo da linha 10 por `high = mid - 1` e substituindo-se o conteúdo da linha 12 por `low = mid + 1` remove-se a recursão de cauda e o resultado da busca não é alterado.
- III. Substituindo-se o código da linha 10 por
`newhigh = mid-1`
`return binary_search(x, lst, low, newhigh)`
e substituindo-se o código da linha 12 por
`newlow = mid+1`
`return binary_search(x, lst, newlow, high)`
remove-se a recursão de cauda.
- IV. Substituindo-se o conteúdo das linhas 9 a 12 por
`if lst[mid] > x :`
 `newlow = low`
 `newhigh = mid-1`
`else:`
 `newlow = mid+1`
 `newhigh = high`
`return binary_search(x, lst, newlow, newhigh)`
mantém-se o resultado da busca.

É correto o que se afirma em

- A) I, apenas.
- B) II e III, apenas.
- C) II e IV, apenas.

D) I, III e IV, apenas.

E) I, II, III e IV.

Gabarito oficial: C

I – incorreta. Sem **return** e sem laço, a função apenas altera **high/low** e termina, devolvendo **None** sempre que o elemento não estiver exatamente no primeiro **mid**.

II – correta. É a transformação clássica de recursão de cauda em iteração: os parâmetros da chamada recursiva viram atribuições, e o **while True** repete o corpo até um dos **return** (elemento achado ou **low > high**).

III – incorreta. Usar variáveis auxiliares não muda nada: continua havendo uma chamada recursiva como última ação, ou seja, recursão de cauda.

IV – correta. Só reorganiza o cálculo dos novos limites e faz uma única chamada recursiva com os mesmos valores das linhas 10 e 12 originais; o resultado é o mesmo.

Questão 29**ENADE 2019 – Engenharia de Computação – Questão 9**

O MergeSort é um método de ordenação que combina dois vetores ordenados e cria um terceiro vetor maior também ordenado. O algoritmo abaixo apresenta essa ideia e combina os vetores `a[lo..mid]` e `a[mid+1..hi]` no vetor `a[lo..hi]`.

```
public class MergeSort {
    private static Comparable[] aux;
    public static void merge(Comparable[] a, int lo, int mid, int hi) {
        int i = lo, j = mid+1;
        for (int k = lo; k <= hi; k++)
            aux[k] = a[k];
        for (int k = lo; k <= hi; k++) {
            if (i > mid)
                a[k] = aux[j++];
            else if (j > hi)
                a[k] = aux[i++];
            else if (aux[j].compareTo(aux[i]) < 0)
                a[k] = aux[j++];
            else
                a[k] = aux[i++];
        }
    }
    public static void sort(Comparable[] a) {
        aux = new Comparable[a.length];
        sort(a, 0, a.length - 1);
    }
    private static void sort(Comparable[] a, int lo, int hi) {
        //implementação
    }
}
```

SEDGEWICK, R.; WAYNE, K. Algorithms. 4. ed. Boston: Addison-Wesley, 2011 (adaptado).

Considerando o código apresentado, a implementação do protótipo do método `sort` da classe `MergeSort` é

A)

```
if (hi == lo)
    return;
int mid = lo + (hi - lo)/2;
sort(a, lo, mid);
sort(a, mid, hi);
merge(a, lo, mid, hi);
```

B)

```
if (hi > lo)
    return;
int mid = lo + (hi - lo)/2;
sort(a, lo, mid);
sort(a, mid, hi);
merge(a, lo, mid, hi);
```

C)

```
if (hi <= lo)
    return;
int mid = lo + (hi - lo)/2;
sort(a, lo, mid);
sort(a, mid, hi);
merge(a, lo, mid, hi);
```

D)

```
if (hi > lo)
    return;
int mid = lo + (hi - lo)/2;
sort(a, lo, mid);
sort(a, mid+1, hi);
merge(a, lo, mid, hi);
```

E)

```
if (hi <= lo)
    return;
int mid = lo + (hi - lo)/2;
sort(a, lo, mid);
sort(a, mid+1, hi);
merge(a, lo, mid, hi);
```

Gabarito oficial: E

Dois detalhes decidem a questão:

- **Caso base:** o trecho tem 0 ou 1 elemento quando $hi \leq lo$; aí não há nada a ordenar e a função retorna. Com $hi > lo$ (B, D) a função retornaria justamente quando há trabalho a fazer. (A usa $hi == lo$, que serve para trechos de um elemento, mas erra na divisão.)
- **Divisão:** as metades são $[lo..mid]$ e $[mid+1..hi]$, exatamente as que **merge** espera. Com `sort(a, mid, hi)` (A, B, C) o elemento `mid` entra nas duas metades e, quando $hi = lo + 1$, a chamada `sort(a, mid, hi)` repete o mesmo intervalo indefinidamente (estouro de pilha).

Só E combina os dois corretamente.

Observação

No código da prova, `aux[j].compareTo(aux[i])` devolve um `int`, e não um `boolean`; em Java isso não compila. No original de Sedgewick e Wayne a condição é `less(aux[j], aux[i])`, isto é, `aux[j].compareTo(aux[i]) < 0`. O erro não afeta a resposta.

Questão 30

ENADE 2023 – Engenharia de Computação – Discursiva 2

Considere que uma rede de varejo resolva consolidar a base de dados de suas M lojas, o que resulta em uma tabela X não ordenada com N registros de clientes, possivelmente repetidos. Devido à necessidade de se criar uma tabela Y contendo os clientes da tabela X com as repetições eliminadas, cogita-se utilizar dois possíveis algoritmos, os quais são apresentados a seguir.

Algoritmo A, que executa as seguintes ações:

1. Cria uma tabela Y inicialmente vazia;
2. Percorre a tabela X , cliente por cliente, verificando se cada um deles já está na tabela Y . Caso não esteja, insere na tabela Y o cliente que está faltando.

Algoritmo B, que executa as seguintes ações:

1. Cria uma tabela Y inicialmente vazia;
2. Ordena a tabela X usando o algoritmo *quicksort*;
3. Insere, na tabela Y , o cliente da primeira posição da tabela X ;
4. Percorre a tabela X , cliente por cliente, a partir do segundo cliente, verificando se cada cliente é igual ao anterior. Caso não seja, insere o cliente na tabela Y .

Como resultado das ações tanto do algoritmo A quanto do algoritmo B, a tabela Y gerada ao final conterá os clientes da tabela X com as repetições eliminadas.

A partir dessas informações, observe o código em linguagem C apresentado a seguir.

```
void quicksort(int *v, int ini, int fim) {
    // v é o vetor a ser ordenado
    // ini é o índice do primeiro elemento a ser ordenado
    // fim é o índice do último elemento a ser ordenado
    if(ini < fim) {
        x = particiona(v, ini, fim, (ini+fim)/2);
        // atribuir os valores para a, b, c e d
        quicksort(v,a,b);
        quicksort(v,c,d);
    }
}

int particiona (int *vetor, int ini, int fim, int pivot){
    // implementar uma função que reorganiza o vetor e retorna a posição
    // final x do pivot. Ao final do processo, os elementos menores ou iguais a
    // vetor[pivot] devem ter índice menor do que x; os elementos maiores que
    // vetor[pivot] devem ter índice maior do que x.
}

void troca (int *vetor, int i, int j) {
    // função auxiliar que permuta os conteúdos das posições i e j do vetor
    aux = vetor[i];
    vetor[i] = vetor[j];
    vetor[j] = aux;
}
```

Com base nos dados apresentados, faça o que se pede nos itens a seguir.

- a) Determine o número máximo de comparações executadas no passo 2 do algoritmo A. (valor: 2,0 pontos)

- b) Determine o número máximo de comparações executadas no passo 4 do algoritmo B. (valor: 2,0 pontos)
- c) Na implementação recursiva do programa apresentado, quais são os valores dos parâmetros **a**, **b**, **c** e **d**? (valor: 2,0 pontos)
- d) Escreva o corpo da função **particiona** em linguagem C utilizando a função auxiliar **troca**, a qual foi definida anteriormente. (valor: 4,0 pontos)

Padrão de resposta oficial (INEP)

- a) O máximo ocorre quando todos os clientes são diferentes (não há repetição): o k -ésimo cliente é comparado com os $k - 1$ já inseridos em Y , totalizando $0 + 1 + 2 + \dots + (N - 1) = \frac{N(N - 1)}{2}$ comparações. Resposta alternativa aceita: $O(N^2)$ comparações.
- b) $N - 1$ comparações (cada cliente, a partir do segundo, é comparado com o anterior). Resposta alternativa aceita: $O(N)$.
- c) **a** = ini; **b** = **x** - 1 ou **b** = **x**; **c** = **x** + 1 ou **c** = **x**; **d** = fim.
- d)

```
troca(v, ini, pivot);
pivot = ini;
ini = ini + 1;
while (ini <= fim)
    if (vetor[ini] <= vetor[pivot])
        ini = ini + 1;
    else {
        troca(v, ini, fim);
        fim = fim - 1;
    }
troca(v, pivot, fim);
return fim;
```

Observação

Dois detalhes do padrão oficial merecem atenção em sala:

- O código usa **v** nas chamadas a **troca**, mas o parâmetro de **particiona** se chama **vetor**; para compilar, use **troca(vetor, ...)**.
- Em (c), as variantes **b** = **x** ou **c** = **x** incluem o pivô (já na posição final) em uma das chamadas. Isso continua ordenando, mas pode não reduzir o tamanho do subproblema: se o pivô for o maior elemento e terminar em **fim**, **quicksort(v, ini, x)** repete o mesmo intervalo e a recursão não termina. A escolha segura é **a** = ini, **b** = **x** - 1, **c** = **x** + 1, **d** = fim.

Comparando (a) e (b): o algoritmo B custa $O(N \log N)$ em média (dominado pelo *quicksort*) mais $O(N)$, contra $O(N^2)$ do algoritmo A.

Questão 31

ENADE 2005 – Computação (Sistemas de Informação) – Questão 13

Julgue os itens a seguir, acerca de algoritmos para ordenação.

- I. O algoritmo de ordenação por inserção tem complexidade $O(n \times \log n)$.
- II. Um algoritmo de ordenação é dito estável caso ele não altere a posição relativa de elementos de mesmo valor.
- III. No algoritmo *quicksort*, a escolha do elemento pivô influencia o desempenho do algoritmo.
- IV. O *bubble-sort* e o algoritmo de ordenação por inserção fazem, em média, o mesmo número de comparações.

Estão certos apenas os itens

- A) I e II. B) I e III. C) II e IV. D) I, III e IV. E) II, III e IV.

Gabarito oficial: E

I – incorreta. A inserção é $O(n^2)$ no pior caso e no caso médio (é $O(n)$ apenas no melhor caso, vetor já ordenado).

II – correta. É a definição de estabilidade.

III – correta. Pivôs que dividem o vetor de forma equilibrada levam a $O(n \log n)$; pivôs extremos (por exemplo, sempre o menor) levam a $O(n^2)$.

IV – correta (em ordem de grandeza). Ambos fazem $\Theta(n^2)$ comparações em média. (Em número exato, a inserção faz cerca de $n^2/4$ comparações no caso médio, contra $n^2/2$ do *bubble sort* sem otimizações; a banca considerou a ordem assintótica.)

Questão 32

ENADE 2005 – Computação (Sistemas de Informação) – Questão 15

Considere o algoritmo que implementa o seguinte processo: uma coleção desordenada de elementos é dividida em duas metades e cada metade é utilizada como argumento para a reaplicação recursiva do procedimento. Os resultados das duas reaplicações são, então, combinados pela intercalação dos elementos de ambas, resultando em uma coleção ordenada. Qual é a complexidade desse algoritmo?

- A) $O(n^2)$ B) $O(n^{2n})$ C) $O(2^n)$ D) $O(\log n \times \log n)$ E) $O(n \times \log n)$

Gabarito oficial: E

É o *mergesort*: $T(n) = 2T(n/2) + \Theta(n)$ (duas chamadas recursivas sobre metades e intercalação linear). Pelo Teorema Mestre (caso 2), $T(n) = \Theta(n \log n)$. Intuitivamente: são $\log_2 n$ níveis de recursão, e em cada nível a intercalação percorre os n elementos.

Questão 33

ENADE 2005 – Computação (CC e EC) – Questão 43

No processo de pesquisa binária em um vetor ordenado, os números máximos de comparações necessárias para se determinar se um elemento faz parte de vetores com tamanhos 50, 1.000 e 300 são, respectivamente, iguais a

- A) 5, 100 e 30. B) 6, 10 e 9. C) 8, 31 e 18. D) 10, 100 e 30. E) 25, 500 e 150.

Gabarito oficial: B

Na busca binária, cada comparação divide o intervalo ao meio. O número máximo de comparações (com o elemento do meio) para um vetor de n posições é $\lceil \log_2 n \rceil + 1$:

- $n = 50$: $2^5 = 32 \leq 50 < 64 = 2^6 \Rightarrow 5 + 1 = 6$;
- $n = 1\,000$: $2^9 = 512 \leq 1\,000 < 1\,024 \Rightarrow 9 + 1 = 10$;
- $n = 300$: $2^8 = 256 \leq 300 < 512 \Rightarrow 8 + 1 = 9$.

(Equivalentemente, $\lceil \log_2(n + 1) \rceil$.)

Questão 34

ENADE 2008 – Tecnologia em Análise e Desenvolvimento de Sistemas – Questão 20

```

1 Algoritmo ENADE2008
2 variaveis
3   V[0..4] ← {2,0,4,3,1}:inteiro
4   I,J,A : inteiro
5 inicio
6   para I ← 0 ate 3 passo 1 faca
7     para J ← 0 ate 3-I passo 1 faca
8       se (V[J] > V[J+1] ) entao
9         A ← V[J]
10        V[J] ← V[J+1]
11        V[J+1] ← A
12      fim se
13    escreva V[0],V[1],V[2],V[3],V[4]
14  fim para
15  fim para
16 fim algoritmo

```

Com relação ao algoritmo acima, que manipula um vetor de inteiros, julgue os itens a seguir.

- I. Quando as variáveis I e J valerem, respectivamente, 0 e 1, a linha 13 apresentará a sequência de valores 0,2,4,3,1.
- II. Quando as variáveis I e J valerem, respectivamente, 1 e 0, a linha 13 apresentará a sequência de valores 0,2,3,1,4.
- III. Quando as variáveis I e J valerem, respectivamente, 1 e 2, a linha 13 apresentará a sequência de valores 0,2,1,3,4.

Assinale a opção correta.

- A) Apenas um item está certo.
- B) Apenas os itens I e II estão certos.
- C) Apenas os itens I e III estão certos.
- D) Apenas os itens II e III estão certos.
- E) Todos os itens estão certos.

Gabarito oficial: E

É o *bubble sort*. Rastreando (vetor após cada comparação):

I	J	V	I	J	V
0	0	0,2,4,3,1	1	0	0,2,3,1,4 (item II)
0	1	0,2,4,3,1 (item I)	1	1	0,2,3,1,4
0	2	0,2,3,4,1	1	2	0,2,1,3,4 (item III)
0	3	0,2,3,1,4	2	1	0,1,2,3,4

Ao fim de cada passada externa, o maior elemento ainda não posicionado “borbulha” para o fim do vetor; por isso o laço interno vai só até 3-I.

Questão 35

ENADE 2008 – Tecnologia em Análise e Desenvolvimento de Sistemas – Questão 22

```

1 funcao busca(V[0..9] : inteiro, K : inteiro):
    inteiro
2 variaveis
3   C, F, K, M : inteiro
4 inicio
5   F ← 9
6   [_____]
7   enquanto((V[M] <> K) ou (F > C))
8     [_____]
9     se(K < V[M]) entao
10      F ← M - 1;
11    senao
12      [_____]
13  fim enquanto
14  se(V[M] <> K) entao
15    retorne(0)
16  senao
17    retorne(M)
18  fim se
19 fim

```

O algoritmo representado pelo pseudocódigo acima está incompleto, pois faltam 3 linhas de código. A função busca desse algoritmo recebe um vetor ordenado de forma crescente e um valor a ser pesquisado. A partir disso, essa função verificará se o número armazenado no ponto mediano do vetor é o número procurado. Se for o número procurado, retornará o índice da posição do elemento no vetor e encerrará a busca. Se não for, a função segmentará o vetor em duas partes a partir do ponto mediano, escolherá o segmento no qual o valor procurado está inserido, e o processo se repetirá.

A partir dessas informações, assinale a opção que contém os comandos que completam, respectivamente, as linhas 6, 8 e 12 do algoritmo.

A) $C \leftarrow 0$
 $M \leftarrow (C + F)/2$
 $C \leftarrow M + 1$

C) $C \leftarrow 0$
 $C \leftarrow M + 1$
 $M \leftarrow (C + F)/2$

E) $C \leftarrow 1$
 $M \leftarrow (C + F)/2$
 $C \leftarrow M + 1$

B) $C \leftarrow 1$
 $M \leftarrow (C + F)/2$
 $C \leftarrow M - 1$

D) $C \leftarrow 1$
 $C \leftarrow M + 1$
 $M \leftarrow (C + F)/2$

Gabarito oficial: A

Linha 6: o início do intervalo é o índice 0 do vetor $V[0..9]$ ($C \leftarrow 0$; com $C \leftarrow 1$ a posição 0 nunca seria examinada). **Linha 8:** a cada iteração calcula-se o ponto médio, $M \leftarrow (C + F)/2$. **Linha 12:** se K não é menor que $V[M]$, ele está à direita do meio: $C \leftarrow M + 1$ (simétrico a $F \leftarrow M - 1$ da linha 10).

Observação

O pseudocódigo da prova tem falhas que valem ser discutidas em sala: (i) na linha 7, $V[M]$ é usado antes de M receber valor; (ii) K aparece como parâmetro e também como variável local; (iii) a condição usa ou: o laço continua mesmo após achar K, e pode não parar quando o intervalo se esgota. A versão correta seria algo como $M \leftarrow (C+F)/2$ antes do laço e **enquanto** ($V[M] <> K$) e ($C \leq F$), recalculando M no fim de cada iteração.

5 Recursão e análise de complexidade

Questão 36

ENADE 2014 – Ciência da Computação (Bacharelado) – Questão 12

Análise o custo computacional dos algoritmos a seguir, que calculam o valor de um polinômio de grau n , da forma: $P(x) = a_n x^n + a_{n-1} x^{n-1} + \dots + a_1 x + a_0$, onde os coeficientes são números de ponto flutuante armazenados no vetor $a[0..n]$, e o valor de n é maior que zero. Todos os coeficientes podem assumir qualquer valor, exceto o coeficiente a_n , que é diferente de zero.

Algoritmo 1:

```
soma = a[0]
Repita para i = 1 até n
  Se a[i] ≠ 0.0 então
    potência = x
    Repita para j = 2 até i
      potência = potência * x
    Fim repita
    soma = soma + a[i] * potência
  Fim se
Fim repita
Imprima(soma)
```

Algoritmo 2:

```
soma = a[n]
Repita para i = n-1 até 0 passo -1
  soma = soma * x + a[i]
Fim repita
Imprima(soma)
```

Com base nos algoritmos 1 e 2, avalie as asserções a seguir e a relação proposta entre elas.

- I. Os algoritmos possuem a mesma complexidade assintótica.

PORQUE

- II. Para o melhor caso, ambos os algoritmos possuem complexidade $O(n)$.

A respeito dessas asserções, assinale a opção correta.

- A) As asserções I e II são proposições verdadeiras, e a II é uma justificativa correta da I.
 B) As asserções I e II são proposições verdadeiras, mas a II não é uma justificativa correta da I.
 C) A asserção I é uma proposição verdadeira, e a II é uma proposição falsa.
 D) A asserção I é uma proposição falsa, e a II é uma proposição verdadeira.
 E) As asserções I e II são proposições falsas.

Gabarito oficial: D

Algoritmo 1: no pior caso (todos os $a_i \neq 0$), o laço interno executa $i - 1$ multiplicações para cada i : $\sum_{i=1}^n (i - 1) = \frac{n(n-1)}{2}$, ou seja, $\Theta(n^2)$. **Algoritmo 2** (regra de Horner): sempre n iterações, $\Theta(n)$.

I – falsa: as complexidades (pior caso) são $\Theta(n^2)$ e $\Theta(n)$.

II – verdadeira: no melhor caso do Algoritmo 1, só $a_n \neq 0$ (é obrigatório); o laço externo roda n vezes e o interno só é executado para $i = n$, com $n - 1$ passos: total $O(n)$. O Algoritmo 2 é $O(n)$ em qualquer caso.

Questão 37

ENADE 2014 – Ciência da Computação (Bacharelado) – Questão 23

Qual o valor de retorno da função a seguir, caso $n = 27$?

```
int recursao (int n) {  
    if (n <= 10) {  
        return n * 2;  
    }  
    else {  
        return recursao(recursao(n/3));  
    }  
}
```

A) 8.

B) 9.

C) 12.

D) 16.

E) 18.

Gabarito oficial: D

Divisão inteira em C:

$$r(27) = r(r(9)) = r(18)$$

$$r(18) = r(r(6)) = r(12)$$

$$r(12) = r(r(4)) = r(8) = 16.$$

Logo $r(27) = 16$.

Questão 38

ENADE 2017 – Ciência da Computação (Bacharelado) – Questão 25

A sequência de Fibonacci é uma sequência de números inteiros que começa em 1, a que se segue 1, e na qual cada elemento subsequente é a soma dos dois elementos anteriores. A função `fib` a seguir calcula o n -ésimo elemento da sequência de Fibonacci:

```
unsigned int fib (unsigned int n)
{
    if (n < 2)
        return 1;
    return fib(n - 2) + fib (n - 1);
}
```

Considerando a implementação acima, avalie as afirmações a seguir.

- I. A complexidade de tempo da função `fib` é exponencial no valor de n .
- II. A complexidade de espaço da função `fib` é exponencial no valor de n .
- III. É possível implementar uma versão iterativa da função `fib` com complexidade de tempo linear no valor de n e complexidade de espaço constante.

É correto o que se afirma em

- A) I, apenas.
- B) II, apenas.
- C) I e III, apenas.
- D) II e III, apenas.
- E) I, II e III.

Gabarito oficial: C

I – correta. $T(n) = T(n - 1) + T(n - 2) + \Theta(1)$, cuja solução é $\Theta(\varphi^n)$, com $\varphi \approx 1,618$: exponencial.

II – incorreta. O espaço é determinado pela profundidade máxima da pilha de chamadas, que é n (a cadeia $\text{fib}(n) \rightarrow \text{fib}(n-1) \rightarrow \dots$); logo o espaço é $\Theta(n)$, linear.

III – correta. Guardando apenas os dois últimos valores:

```
unsigned int fib(unsigned int n) {
    unsigned int a = 1, b = 1, t;
    for (unsigned int i = 2; i <= n; i++) { t = a + b; a = b; b = t; }
    return b;
}
```

Tempo $\Theta(n)$, espaço $\Theta(1)$.

Questão 39

ENADE 2017 – Ciência da Computação (Bacharelado) – Questão 33

Considere a função recursiva **F** a seguir, que em sua execução chama a função **G**:

```
1 void F(int n) {  
2     if(n > 0) {  
3         for(int i = 0; i < n; i++) {  
4             G(i);  
5         }  
6         F(n/2);  
7     }  
8 }
```

Com base nos conceitos de teoria da complexidade, avalie as afirmações a seguir.

- I. A equação de recorrência que define a complexidade da função **F** é a mesma do algoritmo clássico de ordenação *mergesort*.
- II. O número de chamadas recursivas da função **F** é $\Theta(\log n)$.
- III. O número de vezes que a função **G** da linha 4 é chamada é $O(n \log n)$.

É correto o que se afirma em

- A) I, apenas.
- B) II, apenas.
- C) I e III, apenas.
- D) II e III, apenas.
- E) I, II e III.

Gabarito oficial: B

I – incorreta. Para **F**: $T(n) = T(n/2) + \Theta(n)$ (uma única chamada recursiva). O *mergesort* tem $T(n) = 2T(n/2) + \Theta(n)$ (duas chamadas).

II – correta. O argumento é dividido por 2 a cada chamada até chegar a 0: $\lfloor \log_2 n \rfloor + 1$ chamadas, isto é, $\Theta(\log n)$.

III – Pelo gabarito, considerada incorreta: **G** é chamada $n + n/2 + n/4 + \dots < 2n$ vezes, ou seja, $\Theta(n)$.

Observação

Do ponto de vista estritamente formal, a afirmativa III é verdadeira: se o número de chamadas é $\Theta(n)$, ele também é $O(n \log n)$, pois O é apenas um limite superior. A banca evidentemente usou O no sentido de “limite justo” (como Θ). Vale usar a questão em sala justamente para discutir a diferença entre O e Θ .

Questão 40

ENADE 2014 – Ciência da Computação (Bacharelado) – Questão 28

Um cientista afirma ter encontrado uma redução polinomial de um problema NP-Completo para um problema pertencente à classe P. Considerando que esta afirmação tem implicações importantes no que diz respeito à complexidade computacional, avalie as seguintes asserções e a relação proposta entre elas.

I. A descoberta do cientista implica $P = NP$.

PORQUE

II. A descoberta do cientista implica na existência de algoritmos polinomiais para todos os problemas NP-Completo.

A respeito dessas asserções, assinale a opção correta.

- A) As asserções I e II são proposições verdadeiras, e a II é uma justificativa correta da I.
- B) As asserções I e II são proposições verdadeiras, mas a II não é uma justificativa correta da I.
- C) A asserção I é uma proposição verdadeira, e a II é uma proposição falsa.
- D) A asserção I é uma proposição falsa, e a II é uma proposição verdadeira.
- E) As asserções I e II são proposições falsas.

Gabarito oficial: A

Se um problema NP-completo X se reduz polinomialmente a um problema $Y \in P$, então X pode ser resolvido em tempo polinomial (reduz-se e resolve-se Y). Como todo problema de NP se reduz polinomialmente a X (definição de NP-completude), todo problema de NP — em particular todos os NP-completos — passa a ter algoritmo polinomial (II). Daí $NP \subseteq P$ e, como $P \subseteq NP$, $P = NP$ (I). A II é exatamente o argumento que sustenta a I.

Questão 41**ENADE 2019 – Engenharia de Computação – Discursiva 4**

Na matemática, um produtório é definido como:

$$\prod_{i=m}^n x_i = x_m \times x_{m+1} \times x_{m+2} \times \cdots \times x_{n-1} \times x_n.$$

Com base nessa equação, e considerando que $x_i = i + \frac{1}{i}$, com $i > 0$, faça o que se pede nos itens a seguir.

- Escreva uma **função iterativa**, em linguagem C, que receba os parâmetros de limite inferior m e de limite superior n , calcule e retorne o resultado do produtório. (valor: 5,0 pontos)
- Escreva uma **função recursiva**, em linguagem C, que receba os parâmetros de limite inferior m e de limite superior n , calcule e retorne o resultado do produtório. (valor: 5,0 pontos)

Padrão de resposta oficial (INEP)

O estudante deve elaborar algoritmos tais como os seguintes (float pode ser trocado por double).

a) Iterativa

```
/* Solução 1 */
float produtorio(int m, int n)
{
    float i = 1;      // contador como ponto flutuante
    float r = 1.0;    // necessário resultado como ponto flutuante
    for (i = m; i <= n; i += 1)    // incremento com soma
        r *= i + 1/i;
    return r;
}

/* Solução 2 */
float produtorio(int m, int n)
{
    int i = 1;        // contador como inteiro
    float r = 1.0;    // necessário resultado como ponto flutuante
    for (i = m; i <= n; i++)
        r *= i + 1.0/i; // divisão em ponto flutuante; com 1/i o resultado estaria incorreto
    return r;
}
```

b) Recursiva

```
/* Solução 1 */
float produtorio(int m, int n)
{
    if (m == n)
        return (n + 1.0/n); // pode ser (m + 1.0/m)
    else
        return (m + 1.0/m) * produtorio(m+1, n);
}

/* Solução 2 */
float produtorio(int m, int n)
{
    if (m == n)
```

```
    return (n + 1.0/n);    // pode ser (m + 1.0/m)
else
    return (n + 1.0/n) * produtorio(m, n-1);
}
```

Observação

O ponto que mais derruba respostas é a **divisão inteira**: com `int i`, a expressão `1/i` vale 0 para todo $i > 1$, e o produtório sai errado. Por isso o padrão usa `1.0/i` ou um contador `float`. Vale observar também que as versões recursivas supõem $m \leq n$; com $m > n$, a recursão não termina (por convenção, o produtório vazio vale 1).

Questão 42

ENADE 2023 – Engenharia de Computação – Questão 14

Uma lista pode ser dividida em duas partes: o primeiro elemento (a cabeça da lista) e os demais elementos (sua cauda). Por exemplo, em uma lista de inteiros [1, 2, 3, 4], a cabeça dessa lista é o valor inteiro 1, enquanto sua cauda é a lista de inteiros [2, 3, 4]. Uma lista vazia é representada por [].

O código a seguir define duas funções descritas em uma linguagem de programação funcional que manipulam listas de inteiros. A função **enade** recebe uma lista de inteiros e produz uma nova lista de inteiros. A função **auxiliar** é chamada pela função **enade** e possui dois parâmetros: um número inteiro e uma lista de inteiros. Essa função produz uma lista de inteiros.

```
enade :: [Int] -> [Int]
enade [] = []
enade (cabeca:cauda) = auxiliar cabeca (enade cauda)
auxiliar :: Int -> [Int] -> [Int]
auxiliar x [] = [x]
auxiliar x (cabeca:cauda)
  | (x `mod` 2 == 0) = x:cabeca:cauda
  | otherwise = cabeca:auxiliar x cauda
```

Considerando o código apresentado, é correto afirmar que se a função **enade** for executada recebendo como parâmetro de entrada a lista [1, 2, 3, 4, 5, 6, 7, 8], o resultado será

- A) [].
- B) [2, 4, 6, 8].
- C) [1, 2, 3, 4, 5, 6, 7, 8].
- D) [2, 4, 6, 8, 1, 3, 5, 7].
- E) [2, 4, 6, 8, 7, 5, 3, 1].

Gabarito oficial: E

enade processa a lista de trás para a frente (primeiro resolve a cauda, depois chama **auxiliar** com a cabeça). **auxiliar x 1** coloca um número **par** no início de 1 e um **ímpar** no final. Rastreando:

enade [8]	[8]
enade [7,8]	7 é ímpar, vai para o fim: [8,7]
enade [6..8]	6 é par, vai para o início: [6,8,7]
enade [5..8]	[6,8,7,5]
enade [4..8]	[4,6,8,7,5]
enade [3..8]	[4,6,8,7,5,3]
enade [2..8]	[2,4,6,8,7,5,3]
enade [1..8]	[2,4,6,8,7,5,3,1]

Os pares saem em ordem crescente e os ímpares em ordem inversa, pois cada ímpar novo vai para o final, depois dos que já estavam lá. Custo: $O(n^2)$ no pior caso (cada ímpar percorre a lista até o fim).

Questão 43

ENADE 2023 – Engenharia de Computação – Questão 18

Vetores de tamanho dinâmico são muito utilizados como estruturas de dados para armazenar listas e tabelas de dispersão (tabelas *hash*). Essa representação permite que o número máximo de elementos a ser inserido não precise ser pré-determinado. Uma técnica para implementar vetores de tamanho dinâmico é aquela que dobra o tamanho do vetor quando o número de itens a ser armazenado ultrapassa o tamanho atual do vetor. Essa operação requer uma alocação de memória para conter o vetor com o dobro do tamanho atual e a realização de cópia dos elementos para o novo vetor. Com base nessas informações, considere uma lista linear implementada com um vetor dinâmico. Assuma que todos os itens inseridos na lista tenham o mesmo tamanho e que o vetor tenha tamanho inicial para comportar apenas 1 item.

Considerando uma sequência de n inserções no final dessa lista, o tempo médio por inserção e o tempo total gasto para executar todas as n operações são, respectivamente, da ordem de

- A) $\Theta(\log n)$ e $\Theta(n \log n)$.
- B) $\Theta(1)$ e $\Theta(n)$.
- C) $\Theta(1)$ e $\Theta(n^2)$.
- D) $\Theta(n)$ e $\Theta(n^2)$.
- E) $\Theta(\sqrt{n})$ e $\Theta(n^{3/2})$.

Gabarito oficial: B

Análise amortizada: as cópias acontecem quando o vetor passa de 1 para 2, 2 para 4, 4 para 8, ..., e copiam $1 + 2 + 4 + \dots + 2^{\lfloor \log_2 n \rfloor} < 2n$ elementos no total. Somando as n escritas, o custo total é menor que $3n$, isto é, $\Theta(n)$, e o custo médio (amortizado) por inserção é $\Theta(1)$, embora uma inserção isolada que provoque redimensionamento custe $\Theta(n)$. Se o vetor crescesse de tamanho fixo (por exemplo, $+1$ ou $+k$) a cada vez, o total seria $\Theta(n^2)$; é a duplicação que garante o custo linear.

Questão 44

ENADE 2005 – Computação (Sistemas de Informação) – Questão 21

No modo recursivo de representação, a descrição de um conceito faz referência ao próprio conceito. Julgue os itens abaixo, com relação à recursividade como paradigma de programação.

- I. São elementos fundamentais de uma definição recursiva: o caso-base (base da recursão) e a reaplicação da definição.
- II. O uso da recursão não é possível em linguagens com estruturas para orientação a objetos.
- III. As linguagens de programação funcionais têm, na recursão, seu principal elemento de repetição.
- IV. No que diz respeito ao poder computacional, as estruturas iterativas e recursivas são equivalentes.
- V. Estruturas iterativas e recursivas não podem ser misturadas em um mesmo programa.

Estão certos apenas os itens

- A) I e IV. B) II e III. C) I, III e IV. D) I, III e V. E) II, IV e V.

Gabarito oficial: C

I – correta: toda definição recursiva precisa de um caso-base e do passo recursivo. **II – incorreta:** Java, C++, Python etc. permitem recursão normalmente. **III – correta:** em linguagens funcionais puras (Haskell, Lisp, ML) não há laços com variáveis mutáveis; repete-se por recursão. **IV – correta:** todo algoritmo iterativo pode ser escrito recursivamente e vice-versa (usando, se preciso, uma pilha explícita). **V – incorreta:** é comum misturar as duas (por exemplo, um laço dentro de uma função recursiva, como no Sudoku ou no *quicksort*).

Questão 45

ENADE 2005 – Computação (CC e EC) – Questão 51

No famoso jogo da Torre de Hanoi, é dada uma torre com discos de raios diferentes, empilhados por tamanho decrescente em um dos três pinos dados. O objetivo do jogo é transportar-se toda a torre para um dos outros pinos, de acordo com as seguintes regras: apenas um disco pode ser deslocado por vez, e, em todo instante, todos os discos precisam estar em um dos três pinos; além disso, em nenhum momento, um disco pode ser colocado sobre um disco de raio menor que o dele; é claro que o terceiro pino pode ser usado como local temporário para os discos.

Imaginando que se tenha uma situação em que a torre inicial tenha um conjunto de 5 discos, qual o número mínimo de movimentações de discos que deverão ser realizadas para se atingir o objetivo do jogo?

A) 25

B) 28

C) 31

D) 34

E) 38

Gabarito oficial: C

Para mover n discos: move-se $n - 1$ discos para o pino auxiliar, o maior para o destino e os $n - 1$ de volta sobre ele. Logo $T(n) = 2T(n - 1) + 1$, com $T(1) = 1$, cuja solução é $T(n) = 2^n - 1$. Para $n = 5$: $2^5 - 1 = 31$.

```
void hanoi(int n, char origem, char destino, char aux) {  
    if (n == 0) return;  
    hanoi(n - 1, origem, aux, destino);  
    printf("disco %d: %c -> %c\n", n, origem, destino);  
    hanoi(n - 1, aux, destino, origem);  
}
```

Questão 46

ENADE 2005 – Computação (CC e EC) – Questão 54

Considere que, durante a análise de um problema de programação, tenha sido obtida a seguinte fórmula recursiva que descreve a solução para o problema.

$$C(i, j) = \begin{cases} 0 & \text{se } i = 0 \text{ ou } j = 0 \\ 1 + C(i - 1, j - 1) & \text{se } 0 < i \leq M, 0 < j \leq N \text{ e } i = j \\ \max\{C(i, j - 1), C(i - 1, j)\} & \text{se } 0 < i \leq M, 0 < j \leq N \text{ e } i \neq j \end{cases}$$

Qual a complexidade da solução encontrada?

- A) $O(n \times \log n)$ B) $O(n^2)$ C) $O(n^2 \times \log n)$ D) $O(2^n)$ E) $O(n^3)$

Gabarito oficial: B

A fórmula é a recorrência da **subsequência comum mais longa** (LCS), em que “ $i = j$ ” deve ser lido como “o i -ésimo elemento da 1ª sequência é igual ao j -ésimo da 2ª”. Resolvida por **programação dinâmica**, preenche-se uma tabela de $(M + 1) \times (N + 1)$ posições, cada uma em tempo constante: $\Theta(M \cdot N)$, ou seja, $O(n^2)$ quando as duas sequências têm tamanho n . (Implementada como recursão pura, sem memorização, ela seria exponencial; o enunciado pede a complexidade “da solução”, isto é, da programação dinâmica.) Compare com a questão de LCS de 2023 (Questão 71).

Questão 47

ENADE 2005 – Computação (Ciência da Computação) – Questão 65

A análise de complexidade provê critérios para a classificação de problemas com base na computabilidade de suas soluções, utilizando-se a máquina de Turing como modelo referencial e possibilitando o agrupamento de problemas em classes. Nesse contexto, julgue os itens a seguir.

- I. É possível demonstrar que $P \subseteq NP$ e $NP \subseteq P$.
- II. É possível demonstrar que se $P \neq NP$, então $P \cap \text{NP-Completo} = \emptyset$.
- III. Se um problema Q é NP-difícil e $Q \in NP$, então Q é NP-completo.
- IV. O problema da satisfatibilidade de uma fórmula booleana F (uma fórmula é satisfatível, se é verdadeira em algum modelo) foi provado ser NP-difícil e NP-Completo.
- V. Encontrar o caminho mais curto entre dois vértices dados em um grafo de N vértices e M arestas não é um problema da classe P.

Estão certos apenas os itens

- A) I, III e IV. B) II, III, e IV. C) III, IV e V. D) I, II, III, e IV. E) II, III, IV e V.

Gabarito oficial: B

I – incorreta: $P \subseteq NP$ é demonstrável, mas $NP \subseteq P$ é justamente o problema em aberto $P \times NP$. **II – correta:** se algum problema NP-completo estivesse em P, todo NP se reduziria a ele e teríamos $P = NP$; logo, se $P \neq NP$, nenhum NP-completo está em P. **III – correta:** é a definição de NP-completo (NP-difícil e pertencente a NP). **IV – correta:** é o Teorema de Cook–Levin (1971). **V – incorreta:** caminho mínimo é resolvido em tempo polinomial (Dijkstra, Bellman-Ford), portanto está em P.

Questão 48

ENADE 2008 – Computação (CC, EC e SI) – Questão 18

Os números de Fibonacci constituem uma sequência de números na qual os dois primeiros elementos são 0 e 1 e os demais, a soma dos dois elementos imediatamente anteriores na sequência. Como exemplo, a sequência formada pelos 10 primeiros números de Fibonacci é: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34. Mais precisamente, é possível definir os números de Fibonacci pela seguinte relação de recorrência:

$$\text{fib}(n) = 0, \text{ se } n = 0; \quad \text{fib}(n) = 1, \text{ se } n = 1; \quad \text{fib}(n) = \text{fib}(n - 1) + \text{fib}(n - 2), \text{ se } n > 1.$$

Abaixo, apresenta-se uma implementação em linguagem funcional para essa relação de recorrência:

```
fib :: Integer -> Integer
fib 0 = 0
fib 1 = 1
fib n = fib (n - 1) + fib (n - 2)
```

Considerando que o programa acima não reutilize resultados previamente computados, quantas chamadas são feitas à função `fib` para computar `fib 5`?

A) 11**B)** 12**C)** 15**D)** 24**E)** 25**Gabarito oficial: C**

Seja $c(n)$ o número de chamadas para calcular `fib n` (contando a própria): $c(0) = c(1) = 1$ e $c(n) = 1 + c(n - 1) + c(n - 2)$. Então $c(2) = 3$, $c(3) = 5$, $c(4) = 9$ e $c(5) = 1 + 9 + 5 = 15$. (Em geral, $c(n) = 2 \text{fib}(n + 1) - 1$, que cresce exponencialmente.)

Questão 49 ENADE 2008 – Tecnologia em Análise e Desenvolvimento de Sistemas – Questão 27

Os termos da sequência de Fibonacci são definidos por:

```
Fibonacci(0) = 0
Fibonacci(1) = 1
Fibonacci(n) = Fibonacci(n-1) + Fibonacci(n-2)
```

Uma solução recursiva para o cálculo do i -ésimo termo da sequência é dada pela função a seguir.

```
1 funcao fibonacci(inteiro longo n)
2   se((n=0) OU (n=1)) entao
3     retorne n
4   senao
5     retorne fibonacci(n-1) + fibonacci(n-2)
6   fim se
7 fim
```

Acerca da execução recursiva dessa função, assinale a opção **incorreta**.

- A) À medida que o valor de n cresce, há um aumento no número de chamadas recursivas.
- B) Na linha 4, a ordem de execução é calcular o valor para `fibonacci(n-1)` e somente depois calcular o valor para `fibonacci(n-2)`.
- C) O método recursivo é o mais eficiente para o cálculo do i -ésimo termo da sequência de Fibonacci, pois realiza duas chamadas por passo da recursão, cada uma mais simples do que a chamada original.
- D) As condições de parada da recursão são: o valor de n é 0 ou o valor de n é 1.
- E) O uso da recursão para o problema da série de Fibonacci não é indicado, pois ele gera rapidamente uma explosão de chamadas do método.

Gabarito oficial: C

A alternativa C é a incorreta: a versão recursiva é a *menos* eficiente, justamente porque as duas chamadas recalculam os mesmos valores repetidas vezes, levando a tempo exponencial ($\Theta(\varphi^n)$). Uma versão iterativa (ou com memorização) é $O(n)$. As demais afirmativas são verdadeiras. (A rigor, as chamadas recursivas estão na linha 5; a linha 4 é o **senao** — imprecisão do enunciado que não altera a resposta.)

Questão 50

ENADE 2011 – Computação (todas as áreas) – Discursiva 3

Os números de Fibonacci correspondem a uma sequência infinita na qual os dois primeiros termos são 0 e 1. Cada termo da sequência, à exceção dos dois primeiros, é igual à soma dos dois anteriores, conforme a relação de recorrência abaixo.

$$f_n = f_{n-1} + f_{n-2}$$

Desenvolva dois algoritmos, um iterativo e outro recursivo, que, dado um número natural $n > 0$, retorna o n -ésimo termo da sequência de Fibonacci. Apresente as vantagens e desvantagens de cada algoritmo. (valor: 10,0 pontos)

Padrão de resposta oficial (INEP)**Algoritmo iterativo**

```
int fibonacci(n) {
    prevFib ← 0
    currFib ← 1
    if n == 1
        return 0
    if n == 2
        return 1
    for i ← 1 to n - 2 { /* repetir n-2
        vezes */
        temp ← prevFib + currFib
        prevFib ← currFib
        currFib ← temp
    }
    return currFib
}
```

Algoritmo recursivo

```
int fibonacci(n) {
    if n == 1
        return 0
    if n == 2
        return 1
    else
        return fibonacci(n-1)
            + fibonacci(n-2)
}
```

Discussão: a solução recursiva clássica possui a vantagem de ser implementada diretamente a partir da definição do problema, mas tem a grande desvantagem de possuir uma ordem de complexidade exponencial. A versão iterativa tem complexidade linear, o que a torna mais vantajosa em termos de eficiência, mas exige mais atenção na implementação.

Observação

O padrão considera o 1º termo como 0 ($\text{fibonacci}(1) = 0$, $\text{fibonacci}(2) = 1$). Também seria aceitável indexar a partir de $f_0 = 0$; o importante é ser coerente. Outra desvantagem da versão recursiva que vale citar: o espaço da pilha de chamadas é $O(n)$, contra $O(1)$ da iterativa.

Questão 51

ENADE 2011 – Ciência da Computação – Questão 36

O problema **P versus NP** é um problema ainda não resolvido e um dos mais estudados em Computação. Em linhas gerais, deseja-se saber se todo problema cuja solução pode ser eficientemente verificada por um computador, também pode ser eficientemente obtida por um computador. Por “eficientemente” ou “eficiente” significa “em tempo polinomial”.

A classe dos problemas cujas soluções podem ser eficientemente obtidas por um computador é chamada de **classe P**. Os algoritmos que solucionam os problemas dessa classe têm complexidade de pior caso polinomial no tamanho das suas entradas.

Para alguns problemas computacionais, não se conhece solução eficiente, isto é, não se conhece algoritmo eficiente para resolvê-los. No entanto, se para uma dada solução de um problema é possível verificá-la eficientemente, então o problema é dito estar em NP. Dessa forma, a classe de problemas para os quais suas soluções podem ser eficientemente verificadas é chamada de **classe NP**.

Um problema é dito ser **NP-completo** se pertence à classe NP e, além disso, se qualquer outro problema na classe NP pode ser eficientemente transformado nesse problema. Essa transformação eficiente envolve as entradas e saídas dos problemas.

Considerando as noções de complexidade computacional apresentadas acima, analise as afirmações que se seguem.

- I. Existem problemas na classe P que não estão na classe NP.
- II. Se o problema A pode ser eficientemente transformado no problema B e B está na classe P, então A está na classe P.
- III. Se $P = NP$, então um problema NP-completo pode ser solucionado eficientemente.
- IV. Se P é diferente de NP, então existem problemas na classe P que são NP-completos.

É correto apenas o que se afirma em

- A) I. B) IV. C) I e III. D) II e III. E) II e IV.

Gabarito oficial: D

I – incorreta: $P \subseteq NP$ (se a solução pode ser obtida em tempo polinomial, ela também pode ser verificada em tempo polinomial). **II – correta:** para resolver A, transforma-se a entrada em uma instância de B (tempo polinomial) e resolve-se B (tempo polinomial); a composição continua polinomial. **III – correta:** se $P = NP$, todo problema de NP, inclusive os NP-completos, está em P. **IV – incorreta:** é o contrário — se $P \neq NP$, *nenhum* problema NP-completo está em P (ver também a Questão 47).

6 Grafos

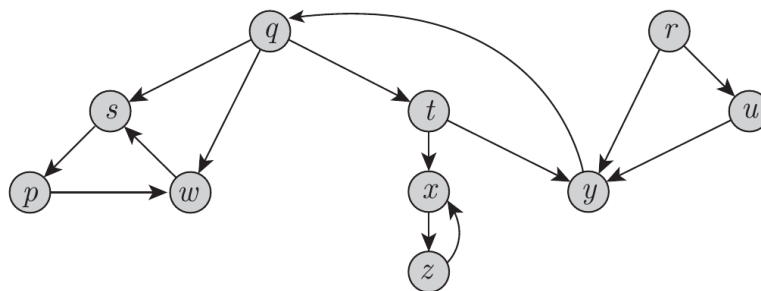
Questão 52

ENADE 2017 – Ciência da Computação (Bacharelado) – Discursiva 5

A busca primeiro em profundidade é um algoritmo de exploração sistemática em grafos, em que as arestas são exploradas a partir do vértice v mais recentemente descoberto que ainda tem arestas inexploradas saindo dele. Quando todas as arestas de v são exploradas, a busca regressa para explorar as arestas que deixam o vértice a partir do qual v foi descoberto. Esse processo continua até que todos os vértices acessíveis a partir do vértice inicial sejam explorados.

CORMEN, T. H.; LEISERSON, C. E.; RIVEST, R. L.; STEIN, C. **Introduction to algorithms**. 3. ed. Cambridge, Massachusetts: The MIT Press, 2009 (adaptado).

Considere o grafo a seguir.



Com base nas informações apresentadas, faça o que se pede nos itens a seguir.

- Mostre a sequência de vértices descobertos no grafo durante a execução de uma busca em profundidade com controle de estados repetidos. Para isso, utilize o vértice r como inicial. No caso de um vértice explorado ter mais de um vértice adjacente, utilize a ordem alfabética crescente como critério para priorizar a exploração. (valor: 7,0 pontos)
- Faça uma representação da matriz de adjacências desse grafo, podendo os zeros ser omitidos nessa matriz. (valor: 3,0 pontos)

Padrão de resposta oficial (INEP)

a) O estudante deve apresentar uma das quatro soluções a seguir:

- $r - u - y - q - s - p - w - t - x - z$
- $r - y - q - w - t - x - z - s - p - u$
- $r - y - q - w - s - p - t - x - z - u$
- $r - u - y - q - s - p - t - x - z - w$

Essas soluções foram encontradas utilizando quatro versões diferentes do algoritmo DFS (Cormen, p. 604): a resposta 1 segue exatamente o algoritmo recursivo; a 2 usa uma pilha e a ordem lexicográfica; a 3 permite empilhar um nó mais de uma vez (adiando a decisão de visitá-lo); a 4 empilha na ordem inversa para imprimir na ordem mais próxima da lexicográfica.

b) O estudante deve apresentar a matriz de adjacências a seguir (linha = origem, coluna = destino; o padrão aceita também a representação transposta):

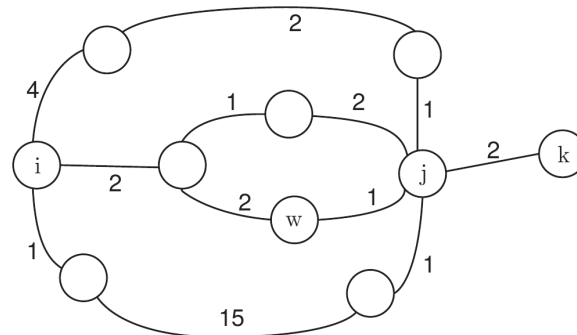
	P	Q	R	S	T	U	W	X	Y	Z
P							1			
Q				1	1		1			
R						1			1	
S	1									
T								1	1	
U									1	
W				1						
X										1
Y		1								
Z								1		

Rastreamento da solução 1 (DFS recursiva). Arcos lidos da figura: $q \rightarrow s$, $q \rightarrow t$, $q \rightarrow w$, $s \rightarrow p$, $p \rightarrow w$, $w \rightarrow s$, $t \rightarrow x$, $t \rightarrow y$, $x \rightarrow z$, $z \rightarrow x$, $y \rightarrow q$, $r \rightarrow u$, $r \rightarrow y$, $u \rightarrow y$. A partir de r (vizinhos em ordem alfabética): $r \rightarrow u \rightarrow y \rightarrow q \rightarrow s \rightarrow p \rightarrow w$; volta a $q \rightarrow t \rightarrow x \rightarrow z$. Os arcos $w \rightarrow s$, $t \rightarrow y$ e $q \rightarrow w$ levam a vértices já visitados.

Questão 53

ENADE 2017 – Ciência da Computação (Bacharelado) – Questão 24

A figura a seguir exibe um grafo que representa um mapa rodoviário, no qual os vértices representam cidades e as arestas representam vias. Os pesos indicam o tempo atual de deslocamento entre duas cidades.



Considerando que os tempos de ida e volta são iguais para qualquer via, avalie as afirmações a seguir acerca desse grafo.

- I. Dado o vértice de origem i , o algoritmo de Dijkstra encontra o menor tempo de deslocamento entre a cidade i e todas as demais cidades do grafo.
- II. Uma árvore geradora de custo mínimo gerada pelo algoritmo de Kruskal contém um caminho de custo mínimo cuja origem é i e cujo destino é k .
- III. Se um caminho de custo mínimo entre os vértices i e k contém o vértice w , então o subcaminho de origem w e destino k deve também ser mínimo.

É correto o que se afirma em

- A) I, apenas.
- B) II, apenas.
- C) I e III, apenas.
- D) II e III, apenas.
- E) I, II e III.

Gabarito oficial: E

I – correta. Todos os pesos são não negativos, condição suficiente para que o Dijkstra calcule os caminhos mínimos de uma origem para todos os vértices.

II – correta (para este grafo). Os caminhos mínimos de i a k custam 7 ($i \rightarrow \cdot \rightarrow \cdot \rightarrow j \rightarrow k$ pelo vértice de cima, $2 + 1 + 2 + 2$, ou por w , $2 + 2 + 1 + 2$). No Kruskal, entram primeiro todas as arestas de peso 1 e depois as de peso 2; a aresta i –(vértice central) de peso 2 e a aresta j – k de peso 2 entram na árvore, e a ligação até j se completa por um dos dois caminhos de custo 5. O caminho de i a k na árvore custa, portanto, 7, que é mínimo. (Em geral, uma árvore geradora mínima *não* garante caminhos mínimos; a afirmação vale para este grafo.)

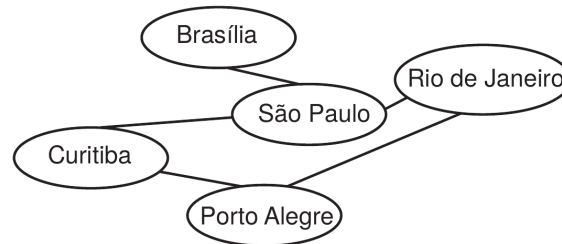
III – correta. Subestrutura ótima: se o trecho $w \rightsquigarrow k$ não fosse mínimo, bastaria substituí-lo por um mais curto para obter um caminho $i \rightsquigarrow k$ mais barato, contradizendo a hipótese.

Questão 54

ENADE 2017 – Sistemas de Informação – Questão 19

A teoria dos grafos estuda as relações entre elementos em um determinado conjunto. Um grafo $G(V, E)$ é determinado por um conjunto não vazio de vértices (V) e um subconjunto de pares de vértices não ordenados, chamados arestas (E).

A figura a seguir representa a possibilidade de voos de uma companhia aérea tal que $V = \{\text{Porto Alegre, Curitiba, São Paulo, Rio de Janeiro e Brasília}\}$, representa o conjunto das capitais (vértices), e $E = \{\{\text{Porto Alegre, Curitiba}\}, \{\text{Curitiba, São Paulo}\}, \{\text{São Paulo, Rio de Janeiro}\}, \{\text{Porto Alegre, Rio de Janeiro}\}, \{\text{São Paulo, Brasília}\}\}$, representa essas possíveis conexões (arestas) entre as capitais.



Com base nas informações apresentadas, avalie as afirmações a seguir.

- I. São Paulo é adjacente a Porto Alegre.
- II. Porto Alegre possui valência igual a 2.
- III. Rio de Janeiro é incidente em São Paulo e Porto Alegre.
- IV. Curitiba possui comprimento igual a 2.

É correto apenas o que se afirma em

- A) I e II.
- B) II e III.
- C) III e IV.
- D) I, II e IV.
- E) I, III e IV.

Gabarito oficial: B

I – incorreta: não existe a aresta $\{\text{São Paulo, Porto Alegre}\}$. **II – correta:** Porto Alegre tem grau (valência) 2 (Curitiba e Rio de Janeiro). **III – correta (no sentido usado pela banca):** as arestas $\{\text{São Paulo, Rio}\}$ e $\{\text{Porto Alegre, Rio}\}$ incidem em Rio de Janeiro, que assim se liga a São Paulo e a Porto Alegre. **IV – incorreta:** “comprimento” é propriedade de caminhos (número de arestas), não de vértices; para vértices fala-se em grau.

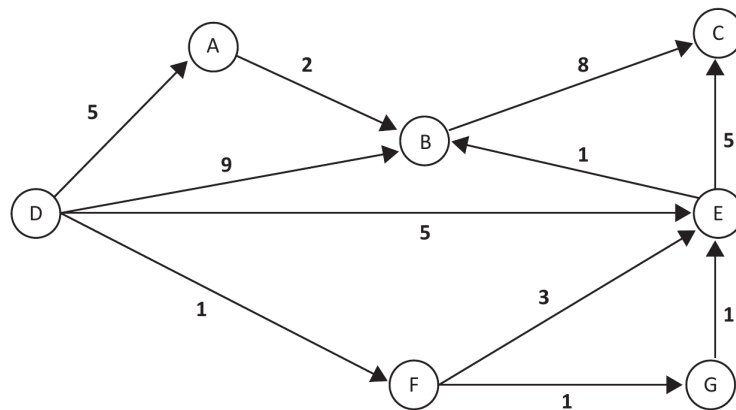
Observação

Rigorosamente, incidência é uma relação entre *aresta* e *vértice* (a aresta incide no vértice). A redação de III é imprecisa; a banca a considerou correta.

Questão 55

ENADE 2021 – Ciência da Computação (Bacharelado) – Questão 34

O algoritmo de Dijkstra para o problema do caminho mínimo em dígrafos com pesos utiliza uma fila de prioridades de vértices, na qual as prioridades são uma estimativa do custo final. A cada iteração, um vértice é retirado da fila, e os arcos que começam nesse vértice são analisados. Considere o seguinte grafo, no qual deseja-se conhecer o custo de um caminho mínimo para cada vértice, a partir do vértice D. Considere que -1 representa um custo “infinito”, ou seja, nenhum caminho até o vértice foi até o momento descoberto.



Com base nas informações e no grafo apresentados, assinale a alternativa que representa a estimativa de custo após duas iterações do algoritmo.

- A) A: 5 B: 6 C: 10 D: 0 E: 4 F: 1 G: -1
 B) A: 5 B: 9 C: -1 D: 0 E: 5 F: 1 G: -1
 C) A: 5 B: 9 C: -1 D: 0 E: 4 F: 1 G: 2
 D) A: 5 B: 7 C: 8 D: 0 E: 4 F: 1 G: 2
 E) A: 5 B: 6 C: 8 D: 0 E: 3 F: 1 G: 2

Gabarito oficial: C

Arcos: $D \rightarrow A(5)$, $D \rightarrow B(9)$, $D \rightarrow E(5)$, $D \rightarrow F(1)$, $A \rightarrow B(2)$, $B \rightarrow C(8)$, $E \rightarrow B(1)$, $E \rightarrow C(5)$, $F \rightarrow E(3)$, $F \rightarrow G(1)$, $G \rightarrow E(1)$.

- **Iteração 1:** retira D (custo 0) e relaxa seus arcos: $A=5$, $B=9$, $E=5$, $F=1$; C e G continuam -1 .
- **Iteração 2:** retira o menor da fila, F (1): $E = \min(5, 1+3) = 4$; $G = 1+1 = 2$.

Estimativas: A 5, B 9, C -1 , D 0, E 4, F 1, G 2. As alternativas D e E mostram estados de iterações posteriores.

Questão 56

ENADE 2014 – Ciência da Computação (Bacharelado) – Questão 30

Uma fazenda possui um único poço artesiano que deve abastecer n bebedouros para o gado. Deseja-se determinar um projeto de ligação entre esses $n + 1$ pontos através de encanamentos com a menor extensão total. Um algoritmo proposto para a solução do problema executa os seguintes passos:

1. Crie $n + 1$ conjuntos unitários, cada um contendo um dos pontos a serem ligados entre si e insira esses conjuntos em um conjunto C .
2. Crie um conjunto D contendo um registro para cada combinação possível de dois pontos distintos a serem ligados. Cada registro deve conter os campos c_i , c_j e d , em que c_i e c_j são os dois pontos a serem ligados e d é a distância entre eles.
3. Enquanto D não estiver vazio faça:
 - 3.1. Remova o registro de D com o menor valor de distância d .
 - 3.2. Se os valores de c_i e c_j do registro removido pertencerem a conjuntos distintos de C , então:
 - 3.2.1. Substitua estes dois conjuntos pela união entre eles.
 - 3.2.2. Guarde o registro removido em um conjunto-solução.

Com base na descrição do problema e do algoritmo proposto, conclui-se que

- A) o problema exemplifica a obtenção de uma árvore geradora mínima, portanto está no conjunto P.
- B) o algoritmo é uma heurística para o Problema do Caixeiro Viajante, logo apresenta complexidade polinomial.
- C) o problema descrito é de otimização, logo pertence ao conjunto NP-difícil, mas não ao conjunto NP-completo.
- D) uma alternativa para a solução do problema é usar o algoritmo de Dijkstra para obtenção do caminho mínimo entre dois pontos.
- E) o passo de maior custo do algoritmo é a criação do conjunto D com as combinações de pontos, apresentando complexidade computacional $O(n!)$.

Gabarito oficial: A

O algoritmo é o de **Kruskal**: ordena as arestas por peso e as aceita sempre que unem componentes distintos (estrutura *union-find*). Ligar todos os pontos com extensão total mínima é o problema da árvore geradora mínima, resolvido em tempo polinomial ($O(m \log m)$), com $m = \binom{n+1}{2}$ arestas), logo está em P.

B: não é heurística para o caixeiro viajante (não forma ciclo hamiltoniano). C: ser de otimização não torna um problema NP-difícil. D: Dijkstra resolve caminho mínimo, não árvore geradora mínima. E: criar D custa $O(n^2)$ (número de pares), não $O(n!)$; o passo dominante é a ordenação/remoção das arestas.

Questão 57

ENADE 2014 – Ciência da Computação (Bacharelado) – Questão 22

Considere o processo de fabricação de um produto siderúrgico que necessita passar por n tratamentos térmicos e químicos para ficar pronto. Cada uma das n etapas de tratamento é realizada uma única vez na mesma caldeira. Além do custo próprio de cada etapa do tratamento, existe o custo de se passar de uma etapa para outra, uma vez que, dependendo da sequência escolhida, pode ser necessário alterar a temperatura da caldeira e limpá-la para evitar a reação entre os produtos químicos utilizados. Assuma que o processo de fabricação inicia e termina com a caldeira limpa. Deseja-se projetar um algoritmo para indicar a sequência de tratamentos que possibilite fabricar o produto com o menor custo total. Nessa situação, conclui-se que

- A) a solução do problema é obtida em tempo de ordem $O(n \log n)$, utilizando-se um algoritmo ótimo de ordenação.
- B) uma heurística para a solução do problema de coloração de grafos solucionará o problema em tempo polinomial.
- C) o problema se reduz a encontrar a árvore geradora mínima para o conjunto de etapas do processo, requerendo tempo de ordem polinomial para ser solucionado.
- D) a utilização do algoritmo de Dijkstra para se determinar o caminho de custo mínimo entre o estado inicial e o final soluciona o problema em tempo polinomial.
- E) qualquer algoritmo conhecido para a solução do problema descrito possui ordem de complexidade de tempo não-polinomial, uma vez que o problema do caixeiro viajante se reduz a ele.

Gabarito oficial: E

Modelando cada etapa como um vértice (mais o estado “caldeira limpa”) e o custo de transição como peso da aresta, deseja-se uma ordem que passe por *todas* as etapas exatamente uma vez, começando e terminando no estado limpo, com custo mínimo: é um ciclo hamiltoniano de custo mínimo, ou seja, o **Problema do Caixeiro Viajante** (PCV), que é NP-difícil. Como o PCV se reduz a este problema, não se conhece algoritmo polinomial para ele.

A: ordenar não captura custos que dependem do par consecutivo. B: coloração é outro problema (e heurística não garante ótimo). C: a árvore geradora mínima não impõe uma sequência única passando por todos os vértices. D: Dijkstra encontra caminho mínimo, mas não obriga a visitar todas as etapas.

Questão 58

ENADE 2019 – Engenharia de Computação – Questão 11

O gestor de um hotel precisa satisfazer a demanda de vários grupos de eventos pelo uso exclusivo de auditórios por uma semana utilizando o menor número possível de auditórios. A tabela apresenta os conflitos de demandas dos grupos de eventos para todos os dias na semana, em que o valor 1 em uma célula indica que um determinado grupo precisará usar um auditório no mesmo horário que outro grupo e o valor 0 indica que não há conflito entre os dois grupos.

	G1	G2	G3	G4	G5
G1	–	1	1	1	1
G2	1	–	1	1	0
G3	1	1	–	0	1
G4	1	1	0	–	0
G5	1	0	1	0	–

Nessa situação, o número mínimo de auditórios que satisfaz todas as demandas é de

- A) um. B) dois. C) três. D) quatro. E) cinco.

Gabarito oficial: C

Modela-se como **coloração de grafos**: cada grupo é um vértice, cada conflito (1) é uma aresta, e cada cor é um auditório. O mínimo de auditórios é o número cromático $\chi(G)$.

- **Limite inferior:** G1, G2 e G3 são mutuamente conflitantes (triângulo), logo $\chi(G) \geq 3$.
- **Coloração com 3 cores:** G1 $\rightarrow$ auditório 1 (conflita com todos); G2 e G5 $\rightarrow$ auditório 2 (não conflitam entre si); G3 e G4 $\rightarrow$ auditório 3 (não conflitam entre si).

Portanto, $\chi(G) = 3$.

Questão 59

ENADE 2019 – Engenharia de Computação – Questão 24

Protocolos de roteamento de estado de enlace utilizam difusão para propagar informações de estado de enlace que são usadas para calcular rotas individuais. Entretanto, algumas técnicas provocam a transmissão de pacotes redundantes na rede. Idealmente, cada nó deveria receber apenas uma cópia do pacote de difusão.

Uma técnica utilizada para resolver o problema da redundância de pacotes, é a difusão por *spanning tree*. Uma *spanning tree* de um grafo $G = (N, E)$ é um grafo $G' = (N, E')$ tal que E' é um subconjunto de E , G' é conexo, não possui ciclos e contém todos os nós originais em G . Se cada enlace tiver um custo associado e o custo de uma árvore for a soma dos custos dos enlaces, então uma árvore cujo custo seja o mínimo entre todas as *spanning trees* do grafo é denominada uma *spanning tree* mínima.

KUROSE, J. F.; ROSS, K. W. **Redes de computadores e a Internet**: uma abordagem top-down. 6. ed. São Paulo: Pearson Education do Brasil, 2013 (adaptado).

Considere uma rede composta por 6 roteadores, designados pelas letras A, B, C, D, E e F, conectados conforme a seguinte tabela de custos de seus enlaces:

Conexão	Custo do enlace
A – B	2
A – C	2
B – C	2
B – D	3
C – D	3
C – E	1
C – F	1
D – F	2
E – F	1

Neste cenário, o custo da *spanning tree* mínima correspondente é, exatamente:

- A) 5. B) 7. C) 8. D) 9. E) 11.

Gabarito oficial: C

Aplicando Kruskal (arestas em ordem crescente de custo, descartando as que fecham ciclo):

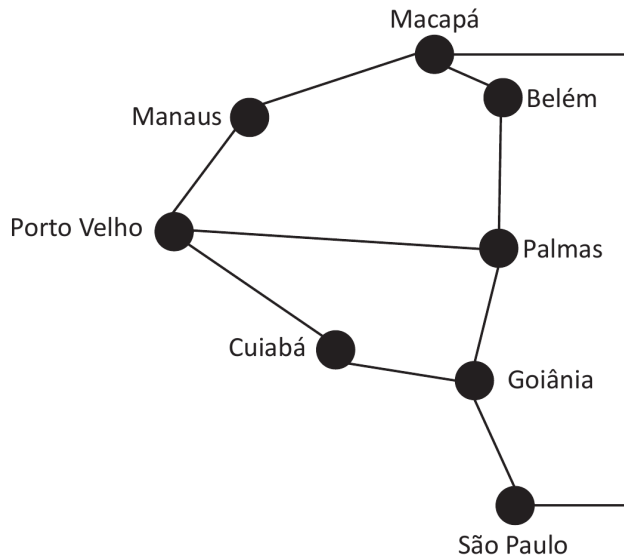
Aresta	Custo	Decisão
C-E	1	aceita
C-F	1	aceita
E-F	1	descartada (ciclo C-E-F)
A-B	2	aceita
A-C	2	aceita (une {A,B} a {C,E,F})
B-C	2	descartada (ciclo)
D-F	2	aceita (D entra; 5 arestas para 6 nós: fim)

Custo: $1 + 1 + 2 + 2 + 2 = 8$. A árvore tem $|N| - 1 = 5$ arestas, como esperado.

Questão 60

ENADE 2023 – Engenharia de Computação – Questão 32

Suponha que uma empresa de transportes deseje calcular uma rota de Manaus para São Paulo utilizando a estratégia da busca gulosa, técnica de busca local que seleciona a melhor alternativa disponível a cada passo. Observe, a seguir, um grafo de conexão de cidades até o destino final e o quadro com estimativas de distância de São Paulo até as demais cidades do grafo.



Cidade	Distância
Manaus	2 693 km
Porto Velho	2 464 km
Macapá	2 665 km
Belém	2 463 km
Palmas	1 489 km
Cuiabá	1 326 km
Goiânia	809 km
São Paulo	0 km

Com base nessas informações, é correto afirmar que a solução encontrada a partir da utilização do algoritmo será

- A) Manaus → Macapá → São Paulo
- B) Manaus → Porto Velho → Cuiabá → Goiânia → São Paulo
- C) Manaus → Porto Velho → Palmas → Goiânia → São Paulo
- D) Manaus → Macapá → Belém → Palmas → Goiânia → São Paulo
- E) Manaus → Macapá → Belém → Palmas → Porto Velho → Cuiabá → Goiânia → São Paulo

Gabarito oficial: B

A busca gulosa (*greedy best-first*) escolhe, a cada passo, o vizinho com a menor estimativa $h(n)$ até o destino:

- Manaus: vizinhos Macapá (2 665) e Porto Velho (2 464) → **Porto Velho**;
- Porto Velho: vizinhos Palmas (1 489) e Cuiabá (1 326) → **Cuiabá**;
- Cuiabá: vizinho Goiânia (809) → **Goiânia**;
- Goiânia: vizinho São Paulo (0) → **São Paulo**.

Note que a estratégia gulosa olha apenas a estimativa $h(n)$, sem somar o custo já percorrido. Por isso não escolhe Macapá, embora exista uma ligação direta Macapá–São Paulo (alternativa A, com menos trechos): a estimativa de Macapá (2 665) é pior que a de Porto Velho. A busca gulosa não garante a rota ótima; o A* ($f = g + h$) garantiria, com heurística admissível.

Questão 61

ENADE 2005 – Computação (CC e EC) – Questão 44

Deseja-se supervisionar as redes de comunicação de dados de um conjunto de empresas. Cada empresa tem a sua própria rede, que é independente das redes das outras empresas e é constituída de ramos de fibra óptica. Cada ramo conecta duas filiais distintas (ponto-a-ponto) da empresa. Há, no máximo, um ramo de fibra interligando diretamente um mesmo par de filiais. A comunicação entre duas filiais pode ser feita diretamente por um ramo de fibra que as interliga, se este existir, ou, indiretamente, por meio de uma sequência de ramos e filiais. A rede de cada empresa permite a comunicação entre todas as suas filiais. A tabela abaixo apresenta algumas informações acerca das redes dessas empresas.

empresa	n.º de filiais	número de ramos de fibra entre filiais
E1	9	18
E2	10	45
E3	14	13
E4	8	24

Com relação à situação apresentada acima, é correto deduzir que,

- I. no caso da empresa E1, a falha de um ramo de rede certamente fará que, ao menos, uma filial não possa mais comunicar-se diretamente com todas as outras filiais da empresa.
- II. na rede da empresa E2, a introdução de um novo ramo de rede certamente violará a informação de que há somente um par de fibras entre duas filiais.
- III. no caso da empresa E3, a falha de um único ramo de rede certamente fará que, ao menos, uma filial não possa mais comunicar-se, direta ou indiretamente, com todas as outras filiais da empresa.
- IV. na rede da empresa E4, todas as filiais da empresa comunicam-se entre si diretamente.

Estão certos apenas os itens

- A) I e II. B) I e IV. C) II e III. D) II e IV. E) III e IV.

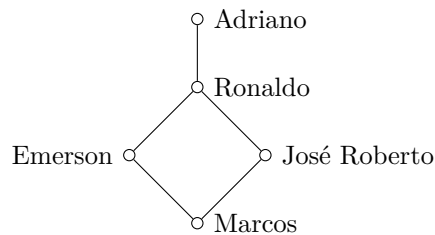
Gabarito oficial: C

Modelando cada rede como um grafo simples e conexo (filiais = vértices, ramos = arestas):

- **I – incorreta:** com 9 vértices e 18 arestas a rede não é completa ($\binom{9}{2} = 36$), então já há filiais que não se comunicam *diretamente* com todas; a falha de um ramo não é o que “faz” isso. E a rede tem arestas de sobra para continuar conexa.
- **II – correta:** $\binom{10}{2} = 45$: E2 já é um grafo completo K_{10} ; qualquer novo ramo duplicaria uma ligação existente.
- **III – correta:** um grafo conexo com n vértices e $n - 1$ arestas (14 e 13) é uma **árvore**; remover qualquer aresta o desconecta.
- **IV – incorreta:** para todas se ligarem diretamente seriam necessários $\binom{8}{2} = 28$ ramos; há apenas 24.

Questão 62

ENADE 2005 – Computação (CC e EC) – Questão 48



Considerando o diagrama de Hasse apresentado acima, assinale a opção que apresenta uma lista ordenada, da esquerda para a direita, que preserve a ordem do diagrama.

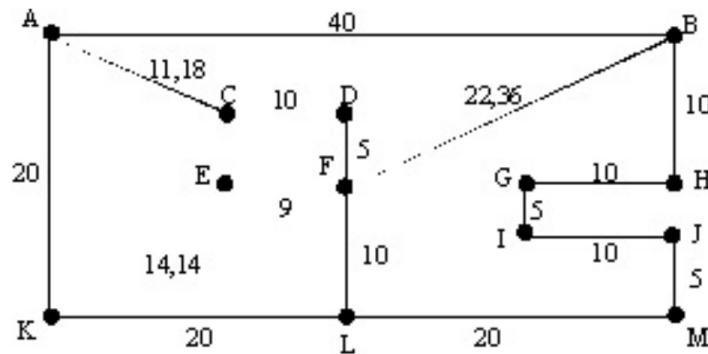
- A) Marcos, José Roberto, Emerson, Ronaldo, Adriano
- B) Emerson, Marcos, Ronaldo, Adriano, José Roberto
- C) Adriano, Ronaldo, José Roberto, Marcos, Emerson
- D) Ronaldo, Marcos, Emerson, Adriano, José Roberto
- E) Marcos, Adriano, Emerson, José Roberto, Ronaldo

Gabarito oficial: A

O diagrama de Hasse define uma ordem parcial: $\text{Marcos} < \text{Emerson}$, $\text{Marcos} < \text{José Roberto}$, $\text{Emerson} < \text{Ronaldo}$, $\text{José Roberto} < \text{Ronaldo}$, $\text{Ronaldo} < \text{Adriano}$. Uma lista que “preserva a ordem” é uma **ordenação topológica** desse grafo: cada elemento aparece depois de todos os que estão abaixo dele. Marcos vem primeiro, depois Emerson e José Roberto (em qualquer ordem entre si), depois Ronaldo e por fim Adriano. Só a alternativa A respeita isso.

Questão 63

ENADE 2005 – Computação (Ciência da Computação) – Questão 58



Uma forma de analisar e comparar o desempenho de algoritmos de busca heurística é utilizar um problema bem conhecido como referência. Um exemplo desse tipo de problema é o cálculo de rotas entre diferentes cidades. No grafo ilustrado acima, cada nó representa uma cidade distinta, e cada ramo, uma rodovia que interliga as cidades representadas pelos nós que ele une, cujo peso indica a distância, em km, entre essas cidades pela rodovia. Suponha que se deseje encontrar a melhor rota entre as cidades A e M, indicadas nesse grafo. Considere, ainda, os valores indicados na tabela abaixo como distância em linha reta, em km, de cada cidade para a cidade M.

A	44,72	E	30,67	I	11,18
B	20,00	F	22,36	J	5,00
C	33,54	G	14,14	K	40,00
D	25,00	H	10,00	L	20,00

A partir dessas informações, julgue os itens seguintes, relativos a algoritmos de busca.

- I. Utilizando-se o algoritmo A^* , a rota entre A e M encontrada no problema acima é ACDFLM e o custo do caminho é 56,18.
- II. Utilizando-se a busca gulosa, a rota encontrada no problema acima é ACDFLM.
- III. Para utilizar algoritmos de busca heurística, deve-se definir uma heurística que superestime o custo da solução.
- IV. O A^* é um algoritmo ótimo e completo quando heurísticas admissíveis são utilizadas.
- V. No *simulated annealing*, é possível haver movimentos para um estado com avaliação pior do que a do estado corrente, dependendo da temperatura do processo e da probabilidade de escolha.

Estão certos apenas os itens

- A) I, II e III. B) I, IV e V. C) I, III e V. D) II, III, e IV. E) II, IV e V.

Gabarito oficial: B

I – correta. A^* expande sempre o nó de menor $f = g + h$:

- de A: C ($11,18 + 33,54 = 44,72$), K ($20 + 40 = 60$), B ($40 + 20 = 60$) → expande C;
- de C: D ($21,18 + 25 = 46,18$) → expande D; de D: F ($26,18 + 22,36 = 48,54$) → expande F;
- de F: L ($36,18 + 20 = 56,18$) → expande L; de L: M ($56,18 + 0$) → meta.

Rota ACDFLM, custo $11,18 + 10 + 5 + 10 + 20 = 56,18$.

II – incorreta. A busca gulosa usa só h : de A escolhe B ($h = 20$, menor que C e K); de B, H (10); de H, G; de G, I; de I, J; de J, M. Rota ABHGIJM (custo 80), não ACDFLM — e nem é a rota ótima.

III – incorreta. Para A^* ser ótimo a heurística deve ser *admissível*, isto é, nunca *superestimar* o custo real.

IV – correta. Com heurística admissível (e grafo finito com custos positivos), A^* é completo e ótimo.

V – correta. O *simulated annealing* aceita movimentos piores com probabilidade $e^{-\Delta/T}$, que diminui conforme a temperatura cai.

Observação

Na figura original, as ligações do vértice E (rótulos 9 e 14,14) não aparecem desenhadas. Considerando-as (E–F e E–K), E seria alcançado com $f > 56,18$ e não altera o resultado.

Questão 64

ENADE 2008 – Ciência da Computação – Questão 31

Julgue os itens a seguir, relativos a métodos de busca com informação (busca heurística) e sem informação (busca cega), aplicados a problemas em que todas as ações têm o mesmo custo, o grafo de busca tem fator de ramificação finito e as ações não retornam a estados já visitados.

- I. A primeira solução encontrada pela estratégia de busca em largura é a solução ótima.
- II. A primeira solução encontrada pela estratégia de busca em profundidade é a solução ótima.
- III. As estratégias de busca com informação usam funções heurísticas que, quando bem definidas, permitem melhorar a eficiência da busca.
- IV. A estratégia de busca gulosa é eficiente porque expande apenas os nós que estão no caminho da solução.

Estão certos apenas os itens

- A) I e II. B) I e III. C) I e IV. D) II e IV. E) III e IV.

Gabarito oficial: B

I – correta: com custos iguais, a busca em largura encontra primeiro a solução com menos ações, que é a de menor custo. **II – incorreta:** a busca em profundidade pode achar primeiro uma solução muito longa. **III – correta:** heurísticas bem definidas guiam a busca e reduzem o número de nós expandidos. **IV – incorreta:** a gulosa pode expandir nós fora do caminho da solução (e pode nem achar a ótima), como na Questão 63.

Questão 65

ENADE 2008 – Engenharia de Computação – Questão 51

Considere um jogo do tipo *8-puzzle*, cujo objetivo é conduzir o tabuleiro esquematizado na figura abaixo para o seguinte estado final.

1	2	3
8		4
7	6	5

Considere, ainda, que, em determinado instante do jogo, se tenha o estado E0 a seguir.

3	4	6
5	8	
2	1	7

Pelas regras desse jogo, sabe-se que os próximos estados possíveis são os estados E1, E2 e E3 mostrados abaixo.

3	4	6	3	4	6	3	4	
5		8	5	8	7	5	8	6
2	1	7	2	1		2	1	7
E1			E2			E3		

Considere uma função heurística h embasada na soma das distâncias das peças em relação ao estado final desejado, em que a distância d a que uma peça p está da posição final é dada pela soma do número de linhas com o número de colunas que a separam da posição final desejada. Por exemplo, em E1, $d(1) = 2 + 1 = 3$. A partir dessas informações analise as asserções a seguir.

Utilizando-se um algoritmo de busca gulosa pela melhor escolha que utiliza a função h , o próximo estado no desenvolvimento do jogo a partir do estado E0 tem de ser E3

porque,

dos três estados E1, E2 e E3 possíveis, o estado com menor soma das distâncias entre a posição atual das peças e a posição final é o estado E3.

Assinale a opção correta a respeito dessas asserções.

- A) As duas asserções são proposições verdadeiras, e a segunda é uma justificativa correta da primeira.
- B) As duas asserções são proposições verdadeiras, e a segunda não é uma justificativa correta da primeira.
- C) A primeira asserção é uma proposição verdadeira, e a segunda é uma proposição falsa.
- D) A primeira asserção é uma proposição falsa, e a segunda é uma proposição verdadeira.
- E) As duas asserções são proposições falsas.

Gabarito oficial: A

h é a soma das distâncias de Manhattan. Calculando:

Estado	E0	E1	E2	E3
h	19	20	20	18

A busca gulosa escolhe o sucessor de menor h , que é E3 ($h = 18$). As duas asserções são verdadeiras, e a segunda justifica a primeira.

Questão 66

ENADE 2011 – Computação (todas as áreas) – Questão 20

Considere que G é um grafo qualquer e que V e E são os conjuntos de vértices e de arestas de G , respectivamente. Considere também que $\text{grau}(v)$ é o grau de um vértice v pertencente ao conjunto V . Nesse contexto, analise as seguintes asserções.

Em G , a quantidade de vértices com grau ímpar é ímpar.

PORQUE

Para G , vale a identidade dada pela expressão

$$\sum_{v \in V} \text{grau}(v) = 2|E|.$$

Acerca dessas asserções, assinale a opção correta.

- A) As duas asserções são proposições verdadeiras, e a segunda é uma justificativa correta da primeira.
- B) As duas asserções são proposições verdadeiras, mas a segunda não é uma justificativa correta da primeira.
- C) A primeira asserção é uma proposição verdadeira, e a segunda, uma proposição falsa.
- D) A primeira asserção é uma proposição falsa, e a segunda, uma proposição verdadeira.
- E) Tanto a primeira quanto a segunda asserções são proposições falsas.

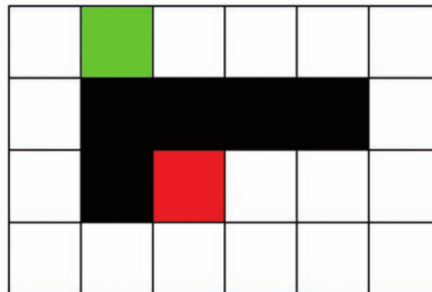
Gabarito oficial: D

A segunda asserção é o **Lema do Aperto de Mãos**: cada aresta contribui com 2 para a soma dos graus. Como essa soma é par, a soma dos graus ímpares também tem de ser par, o que só é possível se a quantidade de vértices de grau ímpar for **par** — a primeira asserção é falsa (o lema prova exatamente o contrário dela).

Questão 67

ENADE 2011 – Ciência da Computação – Questão 40

Considere que a figura abaixo corresponde ao cenário de um jogo de computador. Esse cenário é dividido em 24 quadrados e a movimentação de um personagem entre cada quadrado tem custo 1, sendo permitida apenas na horizontal ou na vertical. Os quadrados marcados em preto correspondem a regiões para as quais os personagens não podem se mover.



Nesse cenário, o algoritmo A^* vai ser usado para determinar o caminho de custo mínimo pelo qual um personagem deve se mover desde o quadrado verde até o quadrado vermelho. Considere que, no A^* , o custo $f(x) = g(x) + h(x)$ de determinado nó x é computado somando-se o custo real $g(x)$ ao custo da função heurística $h(x)$ e que a heurística utilizada é a distância de Manhattan (soma das distâncias horizontal e vertical de x até o objetivo). Desse modo, o custo $f(x)$ do quadrado verde é igual a

- A) 2. B) 3. C) 5. D) 7. E) 9.

Gabarito oficial: B

No nó inicial, $g = 0$. A heurística de Manhattan ignora os obstáculos: o verde está na linha 1, coluna 2, e o vermelho na linha 3, coluna 3 (contando de cima para baixo e da esquerda para a direita), logo $h = |1 - 3| + |2 - 3| = 3$ e $f = 0 + 3 = 3$. (O custo real do caminho, contornando a barreira preta, é bem maior — o que mostra que Manhattan é admissível: nunca superestima.)

7 Técnicas de projeto de algoritmos

Questão 68

ENADE 2014 – Ciência da Computação (Bacharelado) – Discursiva 3

O jogo Sudoku consiste em uma matriz 9×9 dividida em 9 sub-matrizes 3×3 , como mostrado na figura a seguir.

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

Disponível em <<http://www.en-wikipedia.org>>. Acesso em: 26 jul 2014 (adaptado).

A matriz está parcialmente preenchida com números de 1 a 9, e o objetivo do jogo é completar a matriz, de forma que cada linha, coluna e sub-matriz contenham todos os números de 1 a 9.

A partir dessas informações, escreva um algoritmo recursivo baseado em retrocesso (*backtracking*) para resolver o jogo. A matriz foi transformada em um vetor V de 81 posições, contendo zeros nas posições que faltam para serem preenchidas.

Considere que existem duas funções implementadas. A primeira função, $\text{NaoHaViolacao}(x, i, V)$, retorna verdadeiro se a inserção do número x na posição i do vetor V não causa violação das restrições do jogo (número repetido em linha, coluna ou sub-matriz). A segunda função, $\text{Imprime}(V)$, realiza a impressão do vetor V .

Considere, ainda, que o algoritmo deve imprimir o vetor V com a solução encontrada, se esta existir. (valor: 10,0 pontos)

Observação: Qualquer notação em português estruturado, de forma imperativa ou orientada a objetos pode ser utilizada, assim como em uma linguagem de alto nível, como Pascal, C ou Java.

Resposta esperada (elaborada)

Ideia: percorrer as posições em ordem; posições já preenchidas são puladas; numa posição vazia, tenta-se cada valor de 1 a 9 que não viole as regras e avança-se recursivamente. Se nenhum valor levar a uma solução, desfaz-se a escolha ($V[i] = 0$) e retorna-se falso, provocando o retrocesso na chamada anterior. Ao passar da última posição, a solução está completa e é impressa.

```

/* chamada inicial: if (!Resolve(V, 0)) printf("Sem solução"); */
int Resolve(int V[], int i) {
    if (i == 81) {                                /* todas as posições preenchidas */
        Imprime(V);
        return 1;
    }
    if (V[i] != 0)                                /* posição já dada: segue adiante */
        return Resolve(V, i + 1);
    for (int x = 1; x <= 9; x++) {
        if (NaoHaViolacao(x, i, V)) {
            V[i] = x;                               /* escolhe */
            if (Resolve(V, i + 1)) /* explora */
                return 1;
        }
    }
    return 0;
}

```

```
        V[i] = 0;           /* desfaz (retrocesso) */
    }
}
return 0;                  /* nenhum x serve: volta */
}
```

Elementos avaliados: caso base (vetor completo $\Rightarrow$ imprime), tratamento das posições fixas, laço de tentativas com `NaoHaViolacao`, chamada recursiva para a próxima posição e restauração do valor ao retroceder.

Questão 69

ENADE 2014 – Ciência da Computação (Bacharelado) – Discursiva 5

As técnicas de projeto de algoritmos são essenciais para que os desenvolvedores possam implementar software de qualidade. Essas técnicas descrevem os princípios que devem ser adotados para se projetar soluções algorítmicas para um dado problema. Entre as principais técnicas, destacam-se os projetos de algoritmos por tentativa e erro, divisão e conquista, programação dinâmica e algoritmos gulosos.

Nesse contexto, faça o que se pede nos itens a seguir.

- a) Descreva o que caracteriza o projeto de algoritmos por divisão e conquista. (valor: 6,0 pontos)
- b) Apresente uma situação de uso da técnica de projeto de algoritmos por divisão e conquista. (valor: 4,0 pontos)

Resposta esperada (elaborada)

a) A divisão e conquista resolve um problema em três etapas, em geral de forma recursiva:

- (i) **Dividir** a instância em subproblemas menores, do mesmo tipo e independentes entre si;
- (ii) **Conquistar** cada subproblema recursivamente, até chegar a um caso base pequeno o suficiente para ser resolvido diretamente;
- (iii) **Combinar** as soluções dos subproblemas para obter a solução do problema original.

O custo costuma ser descrito por uma recorrência do tipo $T(n) = aT(n/b) + f(n)$, que pode ser resolvida, por exemplo, pelo Teorema Mestre. Difere da programação dinâmica porque os subproblemas não se sobrepõem.

b) Exemplos de uso (basta um): ordenação com *mergesort* (divide o vetor ao meio, ordena cada metade e intercala — $O(n \log n)$); *quicksort*; busca binária; multiplicação rápida de inteiros (Karatsuba) ou de matrizes (Strassen); par de pontos mais próximos; cálculo de potência x^n por exponenciação rápida.

Questão 70

ENADE 2017 – Ciência da Computação (Bacharelado) – Questão 22

Um país utiliza moedas de 1, 5, 10, 25 e 50 centavos. Um programador desenvolveu o método a seguir, que implementa a estratégia gulosa para o problema do troco mínimo. Esse método recebe como parâmetro um valor inteiro, em centavos, e retorna um *array* no qual cada posição indica a quantidade de moedas de cada valor.

```
public static int[] troco(int valor){
    int[] moedas = new int[5];

    moedas[4] = valor / 50;
    valor = valor % 50;
    moedas[3] = valor / 25;
    valor = valor % 25;
    moedas[2] = valor / 10;
    valor = valor % 10;
    moedas[1] = valor / 5;
    valor = valor % 5;
    moedas[0] = valor;
    return(moedas);
}
```

Considerando o método apresentado, avalie as asserções a seguir e a relação proposta entre elas.

- I. O método guloso encontra o menor número de moedas para o valor de entrada, considerando as moedas do país.

PORQUE

- II. Métodos gulosos sempre encontram a solução global ótima.

A respeito dessas asserções, assinale a opção correta.

- A) As asserções I e II são proposições verdadeiras, e a II é uma justificativa correta da I.
B) As asserções I e II são proposições verdadeiras, mas a II não é uma justificativa correta da I.
C) A asserção I é uma proposição verdadeira, e a II é uma proposição falsa.
D) A asserção I é uma proposição falsa, e a II é uma proposição verdadeira.
E) As asserções I e II são proposições falsas.

Gabarito oficial: C

I – verdadeira. O sistema {1, 5, 10, 25, 50} é *canônico*: para ele, escolher sempre a maior moeda possível produz o número mínimo de moedas.

II – falsa. Algoritmos gulosos fazem escolhas localmente ótimas e nem sempre chegam ao ótimo global. Contraexemplo: moedas {1, 3, 4} e troco 6 — o guloso dá 4 + 1 + 1 (3 moedas), mas o ótimo é 3 + 3 (2 moedas). A correção do guloso depende da estrutura do problema (propriedade da escolha gulosa e subestrutura ótima).

Questão 71

ENADE 2023 – Engenharia de Computação – Questão 17

Dada uma sequência de caracteres $S = s_1, s_2, \dots, s_n$, uma subsequência de S é dada pela remoção de zero ou mais caracteres de S . Formalmente, a sequência $X = x_1, x_2, \dots, x_k$ é subsequência de S se existe uma sequência crescente de índices $i_1, \dots, i_k$ de S , tal que $x_j = s_{i_j}$ para todo $j = 1, 2, \dots, k$. Define-se, também, um prefixo da sequência S com comprimento i para $i = 0, \dots, n$, como $S_i = s_1, s_2, \dots, s_i$.

O problema da Subsequência Comum Mais Longa (LCS) e suas variações, definido a seguir, pode ser utilizado em análise de textos e em bioinformática. O problema da LCS pressupõe duas sequências de caracteres $A = a_1, a_2, \dots, a_n$ e $B = b_1, b_2, \dots, b_m$, de comprimentos n e m , respectivamente. Dadas essas duas sequências, é preciso encontrar uma sequência de maior comprimento possível que seja, simultaneamente, subsequência de A e de B .

A partir dessas informações, considere que $Z = z_1, z_2, \dots, z_k$ seja uma LCS das sequências A e B . Em relação ao problema da LCS, avalie as afirmações a seguir.

- I. Se $a_n = b_m$, então $z_k = a_n$, $z_k = b_m$ e Z_{k-1} é uma LCS de A_{n-1} e B_{m-1} .
- II. Se $a_n \neq b_m$ e $z_k \neq a_n$, então Z é uma LCS de A_{n-1} e B .
- III. Seja $lcs(i, j)$ o comprimento da LCS entre A_i e B_j . Para $i, j > 0$, se $a_i \neq b_j$, então $lcs(i, j) = \min\{lcs(i, j-1), lcs(i-1, j)\}$.
- IV. Seja $lcs(i, j)$ o comprimento da LCS entre A_i e B_j . Para $i, j > 0$, se $a_i = b_j$, então $lcs(i, j) = lcs(i-1, j-1) + 1$.

É correto apenas o que se afirma em

- A) I e III.
- B) I e IV.
- C) II e III.
- D) I, II e IV.
- E) II, III e IV.

Gabarito oficial: D

As afirmativas I e II são a **subestrutura ótima** da LCS (Cormen, Teorema 15.1), base do algoritmo de **programação dinâmica**.

I – correta. Se os últimos caracteres coincidem, eles fazem parte de toda LCS, e o restante é uma LCS dos prefixos sem eles.

II – correta. Se $a_n \neq b_m$ e a LCS não termina em a_n , então a_n pode ser descartado: Z é LCS de A_{n-1} e B .

III – incorreta. Quando $a_i \neq b_j$, toma-se o *máximo*, não o mínimo: $lcs(i, j) = \max\{lcs(i, j-1), lcs(i-1, j)\}$.

IV – correta. É o caso de coincidência da recorrência.

Recorrência completa: $lcs(i, 0) = lcs(0, j) = 0$; $lcs(i, j) = lcs(i-1, j-1) + 1$ se $a_i = b_j$; caso contrário, $\max\{lcs(i, j-1), lcs(i-1, j)\}$. Preenchendo a tabela $(n+1) \times (m+1)$, o custo é $\Theta(nm)$ de tempo e espaço.

Questão 72 ENADE 2008 – Tecnologia em Análise e Desenvolvimento de Sistemas – Questão 26

Os métodos de projeto e análise de algoritmos são necessários para o desenvolvimento de algoritmos eficientes, pois eles permitem que se resolva problemas computacionais, reduzindo complexidade e tempo de execução. Acerca desses métodos, assinale a opção **incorreta**.

- A) A abordagem de Divisão e Conquista propõe dividir o problema em vários subproblemas, resolvendo-os e combinando suas soluções para criar a solução final do problema original.
- B) A Programação Dinâmica é uma técnica tipicamente aplicada a problemas de otimização em que pode haver várias soluções possíveis.
- C) O método Guloso nem sempre encontra a solução ótima, mas faz sempre a melhor escolha momentânea.
- D) A Programação Dinâmica e o Método Guloso têm em comum o fato de que se aplicam a problemas em que se observa a existência de sobreposição de subproblemas, ou seja, subproblemas que se repetem.
- E) Os métodos de Divisão e Conquista e Programação Dinâmica apresentam a mesma eficiência quando resolvem problemas combinando soluções de subproblemas dependentes uns dos outros.

Gabarito oficial: ANULADA

A questão pede *uma* opção incorreta, mas há duas:

- **D – incorreta:** sobreposição de subproblemas é característica da programação dinâmica. O que PD e guloso têm em comum é a *subestrutura ótima*; o guloso faz uma escolha e resolve um único subproblema, sem reaproveitar subproblemas repetidos.
- **E – incorreta:** quando os subproblemas são dependentes (se repetem), a divisão e conquista os recalcula e pode ficar exponencial, enquanto a programação dinâmica resolve cada um uma vez (compare Fibonacci recursivo $\times$ iterativo).

A, B e C estão corretas. Com duas respostas possíveis, a questão foi anulada.

Questão 73

ENADE 2011 – Computação (todas as áreas) – Questão 13

O problema do escalonamento de intervalos tem como entrada um conjunto de intervalos numéricos (usualmente interpretados como início e fim de atividades), e o objetivo é escolher, desse conjunto, o maior número possível de intervalos disjuntos dois a dois. Há vários problemas práticos que podem ser modelados dessa forma, como, por exemplo, a seleção de tarefas com horário marcado.

O problema do escalonamento de intervalos pode ser resolvido com o algoritmo descrito a seguir. O conjunto de intervalos dados inicialmente é R e o conjunto de intervalos escolhidos, A , começa vazio.

```
enquanto  $R$  não estiver vazio,  
    seja  $x$  o intervalo de  $R$  com menor tempo de término, e que não tenha interseção com  
    algum intervalo em  $A$   
    retire  $x$  de  $R$  e adicione ao conjunto  $A$   
retorne  $A$ 
```

A respeito desse algoritmo, analise as seguintes asserções.

Para checar se o algoritmo está correto, basta verificar que o primeiro intervalo adicionado ao conjunto A necessariamente faz parte de uma solução ótima.

PORQUE

Pode-se mostrar, por indução no número máximo de intervalos calculados (ou seja, o número de vezes que o laço “enquanto” é executado), que embora possa haver soluções tão boas quanto A , nenhuma delas é estritamente melhor que A . O conjunto com um único intervalo é a base de indução.

Acerca dessas asserções, assinale a opção correta.

- A) As duas asserções são proposições verdadeiras, e a segunda é uma justificativa correta da primeira.
- B) As duas asserções são proposições verdadeiras, mas a segunda não é uma justificativa correta da primeira.
- C) A primeira asserção é uma proposição verdadeira, e a segunda, uma proposição falsa.
- D) A segunda asserção é uma proposição falsa e a segunda, uma proposição verdadeira.
- E) Tanto a primeira quanto a segunda asserções são proposições falsas.

Gabarito oficial: ANULADA

O algoritmo é o clássico **guloso por menor tempo de término** (*activity selection*, Cormen cap. 16), que é ótimo. A prova usa duas peças: (1) a *propriedade da escolha gulosa* — o intervalo que termina primeiro faz parte de alguma solução ótima (argumento de troca); (2) a *subestrutura ótima* — retirado esse intervalo e os que o interceptam, o restante é um subproblema do mesmo tipo, tratado por indução.

Por isso a questão é ambígua: a primeira asserção diz que “basta” verificar a escolha inicial (sem a subestrutura ótima não basta, mas com a indução da segunda asserção a verificação se estende a todas as escolhas), e a redação da opção D tem erro de digitação (“a segunda asserção ... e a segunda”). A questão foi anulada.

Questão 74

ENADE 2011 – Computação (todas as áreas) – Questão 28

Algoritmos criados para resolver um mesmo problema podem diferir de forma drástica quanto a sua eficiência. Para evitar este fato, são utilizadas técnicas algorítmicas, isto é, conjunto de técnicas que compreendem os métodos de codificação de algoritmos de forma a salientar sua complexidade, levando-se em conta a forma pela qual determinado algoritmo chega à solução desejada.

Considerando os diferentes paradigmas e técnicas de projeto de algoritmos, analise as afirmações abaixo.

- I. A técnica de tentativa e erro (*backtracking*) efetua uma escolha ótima local, na esperança de obter uma solução ótima global.
- II. A técnica de divisão e conquista pode ser dividida em três etapas: dividir a instância do problema em duas ou mais instâncias menores; resolver as instâncias menores recursivamente; obter a solução para as instâncias originais (maiores) por meio da combinação dessas soluções.
- III. A técnica de programação dinâmica decompõe o processo em um número finito de subtarefas parciais que devem ser exploradas exaustivamente.
- IV. O uso de heurísticas (ou algoritmos aproximados) é caracterizado pela ação de um procedimento chamar a si próprio, direta ou indiretamente.

É correto apenas o que se afirma em

- A) I. B) II. C) I e IV. D) II e III. E) III e IV.

Gabarito oficial: B

I – incorreta: “escolha ótima local na esperança do ótimo global” é a definição do método *gulosos*. O *backtracking* explora as alternativas e retrocede quando uma escolha não leva a solução (ver Sudoku, Questão 68). **II – correta:** são as três etapas da divisão e conquista: dividir, conquistar, combinar. **III – incorreta:** a exploração exaustiva de subtarefas descreve a força bruta ou o *backtracking*. A programação dinâmica resolve cada subproblema uma única vez e reaproveita o resultado. **IV – incorreta:** “procedimento que chama a si próprio” é a definição de recursão, não de heurística.

8 Lógica de programação com vetores e matrizes

Questão 75

ENADE 2014 – Sistemas de Informação – Questão 9

Considere uma situação em que um professor que queira saber se existem alunos cursando, ao mesmo tempo, as disciplinas A e B, tenha implementado um programa que:

- 1) inicializa um *array* **a** de 30 posições que contém as matrículas dos alunos da disciplina A;
- 2) inicializa outro *array* **b** de 40 posições, que contém as matrículas dos alunos da disciplina B;
- 3) imprime a matrícula dos alunos que estão cursando as disciplinas A e B ao mesmo tempo.

Considere, ainda, que os *arrays* foram declarados e inicializados, não estão necessariamente ordenados, e seus índices variam entre 0 e $n - 1$, sendo n o tamanho do *array*.

```
1. for ( i = 0 to 29 ) {  
2.     for ( j = 0 to 39 ) {  
3.  
4.  
5.  
6.     }  
7. }
```

Com base nessas informações, conclui-se que o trecho a ser incluído nas linhas 3, 4 e 5 do código acima, para que o programa funcione corretamente, é

- | | | |
|---|---|---|
| A) 3. if (a[i] == b[j]) {
4. print(a[i]);
5. } | C) 3. if (a[i] == b[j]) {
4. print(a[j]);
5. } | E) 3. if (a[j] == b[j]) {
4. print(a[j]);
5. } |
| B) 3. if (a[j] == b[i]) {
4. print(a[j]);
5. } | D) 3. if (a[i] == b[i]) {
4. print(a[i]);
5. } | |

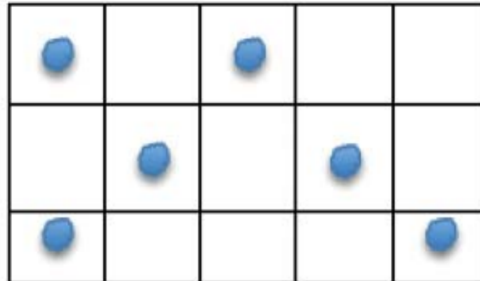
Gabarito oficial: A

O índice **i** percorre **a** (0 a 29) e **j** percorre **b** (0 a 39). Como os vetores não estão ordenados, é preciso comparar cada **a[i]** com cada **b[j]** e imprimir o valor comum, **a[i]** (ou **b[j]**, que é igual). B e E usam **a[j]** com **j** até 39, acessando fora dos limites de **a**; C imprime **a[j]**, que é outro aluno (e pode estar fora dos limites); D compara posições iguais dos dois vetores, o que só funcionaria se estivessem alinhados. Custo: 30×40 comparações, $O(n \cdot m)$.

Questão 76

ENADE 2017 – Sistemas de Informação – Discursiva 4

A figura a seguir representa uma plantação de café com x colunas e y linhas. Nessa figura, as regiões com os cafeeiros plantados são as que estão com os espaços preenchidos, e as regiões com falhas de plantio são as que apresentam seus espaços vazios.



Considerando uma solução algorítmica para contar a quantidade de falhas de plantio do cafezal representado na figura, faça o que se pede nos itens a seguir.

- Indique o tipo de estrutura de dados utilizada na solução, justificando a sua resposta. (valor: 4,0 pontos)
- Escreva o algoritmo da solução em pseudocódigo ou em linguagem de programação. (valor: 6,0 pontos)

Padrão de resposta oficial (INEP)

a) O estudante pode apresentar, dentre outras soluções devidamente justificadas, as seguintes:

- Matriz**, visto que já está sendo informado que a estrutura é de natureza bidimensional e o problema que se deseja resolver é a contagem das regiões com falhas de plantio.
- Lista**, visto que essa estrutura permite contar as regiões com falhas de plantio, embora não considere diretamente a bidimensionalidade.

b) O estudante deve apresentar um pseudocódigo ou código em linguagem de programação que percorre a estrutura de dados, verificando se cada posição da estrutura possui falha no plantio de cafeeiros, incrementando um contador de ocorrência de falhas.

Exemplo de solução (elaborado), com a matriz guardando 1 (cafeeiro) ou 0 (falha):

```
int contaFalhas(int y, int x, int cafezal[y][x]) {  
    int falhas = 0;  
    for (int i = 0; i < y; i++)  
        for (int j = 0; j < x; j++)  
            if (cafezal[i][j] == 0)  
                falhas++;  
    return falhas;  
}
```

Para a figura ($y = 3$, $x = 5$, 6 cafeeiros), o algoritmo retorna $15 - 6 = 9$ falhas. A complexidade é $\Theta(x \cdot y)$.

Questão 77

ENADE 2017 – Sistemas de Informação – Questão 26

Algumas linguagens de programação obedecem a uma ordem específica no que diz respeito à avaliação de operadores. A precedência dos operadores aritméticos, relacionais e booleanos é demonstrada na tabela a seguir, na qual os operadores ++ e --, pré ou pós-fixados, respectivamente, incrementam ou decrementam a variável, como no exemplo: variável `x` com valor 0, resultado de ++`x` é igual a 1 e o de --`x` é igual a -1.

Ordem de precedência (do maior para o menor)	Operadores
1	++ e -- pós-fixados
2	+ e - unários, ++ e -- pré-fixados, !
3	*, /, %
4	+ e - binários
5	<, >, <=, >=
6	=, !=
7	&&
8	

SEBESTA, R. W. **Conceitos de linguagens de programação**. Porto Alegre: Bookman, 2011 (adaptado).

Considerando essas informações e sabendo que as variáveis **a**, **b** e **c** têm, em determinado momento, os valores 1, 2 e 0, respectivamente, assinale a opção em que o resultado da expressão apresentada corresponde ao da expressão **-a * ++b - --c**.

- A) $a * b++ - --c$
 B) $--b > a * c$
 C) $--c/a * --b$
 D) $++c/--b * ++a$
 E) $-a * b++ - c--$

Gabarito oficial: E

Expressão de referência (cada expressão avaliada a partir de $a = 1$, $b = 2$, $c = 0$): $-a \cdot (++b) - (--c) = (-1) \cdot 3 - (-1) = -2$.

- A: $1 \cdot 2 - (-1) = 3$ (pós-incremento usa o valor antigo de b).
- B: $(--b = 1) > (1 \cdot 0 = 0) \Rightarrow$ verdadeiro (1).
- C: $(-1)/1 \cdot 1 = -1$.
- D: $1/1 \cdot 2 = 2$.
- E: $(-1) \cdot 2 - 0 = -2$ (pós-fixados usam os valores antigos: $b = 2, c = 0$). ✓

Questão 78 ENADE 2021 – Tecnologia em Análise e Desenvolvimento de Sistemas – Discursiva 3

Considere a realização de uma pesquisa com 1 000 pessoas para obtenção das seguintes informações: o valor da maior altura; o valor da menor altura; a média das alturas; quantas pessoas têm altura inferior à média das alturas.

Considere, ainda, que um programador foi selecionado para desenvolver um modelo de código que soluciona o problema automatizando a coleta das alturas e a geração das informações.

Com base nas informações apresentadas, desenvolva o código adequado para resolver o problema usando pseudocódigo ou uma linguagem de programação. (valor: 10,0 pontos)

Padrão de resposta oficial (INEP)

```
algoritmo "pesquisa"
var altura[1000], menor, maior, total=0, media: real
var qtdmenor=0, ctpessoas: inteiro
inicio
leia altura[1]
menor = altura[1]
maior = altura[1]
total = altura[1]
para ctpessoas=2 até 1000 faça
    leia (altura[ctpessoas])
    total = total + altura[ctpessoas]
fimpara
media = total/1000
para ctpessoas=1 até 1000 faça
    se menor > altura[ctpessoas] então menor = altura[ctpessoas]
    se maior < altura[ctpessoas] então maior = altura[ctpessoas]
    se media > altura[ctpessoas] então qtdmenor = qtdmenor + 1
fimpara
escreva ("Maior = ", maior, " Menor= ", menor, " Média= ", media,
        " Quantidade de alturas menores que a média= ", qtdmenor)
```

Observação

O ponto central da questão é perceber que a **quantidade abaixo da média** só pode ser contada *depois* de conhecida a média; por isso as alturas precisam ser guardadas num vetor e percorridas duas vezes. Maior e menor poderiam ser atualizados já no primeiro laço (economizando comparações), mas a complexidade continua $\Theta(n)$ nas duas versões.

Questão 79

ENADE 2005 – Computação (Sistemas de Informação) – Questão 14

Os proprietários de um teatro necessitam de uma ferramenta de *software* para reserva de lugares. O desenvolvedor contratado verificou que as poltronas disponíveis para reserva são referenciadas pelo número da fila (a partir do n.º 1) e pelo número da cadeira (a partir do n.º 1) em cada fila, em uma representação matricial em que as linhas e colunas da matriz correspondem, respectivamente, às filas e às colunas de cadeiras. Embora o contexto seja o da organização matricial — N filas de cadeiras (linhas), cada uma contendo M cadeiras (colunas) —, a solução a ser implementada utilizará uma estrutura linear unidimensional (vetor), sendo, portanto, necessária uma conversão entre o lugar referenciado (número f da fila, número c da cadeira) e a posição real na estrutura de armazenamento (posição p no vetor).

Na situação apresentada, considere que a referida matriz seja armazenada no vetor segundo sua sequência de linhas, da primeira para a última, e, em cada linha, da primeira coluna para a última, e que a primeira posição no vetor tenha índice 0. Nessa situação, a posição p da poltrona do teatro localizada à fila de número f e à coluna de número c , é igual a

- A) $c + f \times M$. C) $M \times (f - 1) + (c - 1)$. E) $M \times (c - 1) + M \times f$.
B) $f + c \times M$. D) $M \times (c - 1) + (f - 1)$.

Gabarito oficial: C

No armazenamento por linhas (*row-major*, como em C), antes da fila f vêm $f - 1$ filas completas, cada uma com M cadeiras: $M \times (f - 1)$ posições. Dentro da fila, a cadeira c está deslocada $c - 1$ posições (porque a numeração começa em 1 e o vetor em 0). Logo $p = M(f - 1) + (c - 1)$.
Teste: fila 1, cadeira 1 $\rightarrow p = 0$; fila 2, cadeira 1 $\rightarrow p = M$.

Questão 80

ENADE 2008 – Tecnologia em Análise e Desenvolvimento de Sistemas – Questão 21

```
1 Algoritmo ENADE2008
2 variaveis
3   varA, varB, varC: inteiro
4   varF : real
5   varS : literal
6   varL : logico
7 inicio
8   varS ← "1000"
9   varA ← 4
10  varF ← 3.5
11  varC ← 0
12  varL ← VERDADEIRO
13  se((varC < varA) E varL OU (varS > varC)) entao
14    varB ← varF/varA
15  senao
16    varB ← varA/varC
17  fim se
18 fim algoritmo
```

O código acima

- A) não apresenta erros de nenhum tipo.
- B) apresenta erros de atribuição de tipo inválido, divisão por zero e expressão relacional inválida.
- C) apresenta erros de atribuição de tipo inválido, divisão por zero e estrutura condicional.
- D) apresenta erros de estrutura condicional e expressão relacional inválida.
- E) apresenta somente erro de divisão por zero.

Gabarito oficial: B

- **Atribuição de tipo inválido** (linha 14): `varF/varA` é real, mas `varB` é inteiro.
- **Divisão por zero** (linha 16): `varA/varC` com `varC = 0`.
- **Expressão relacional inválida** (linha 13): `varS > varC` compara um literal (texto "1000") com um inteiro.

A estrutura `se ... entao ... senao ... fim se` está correta. Em tempo de execução, como `(varC < varA) E varL` é verdadeiro, o desvio seguiria pelo `entao`; mas a banca considerou os erros presentes no código, independentemente do caminho executado.

Questão 81 ENADE 2008 – Tecnologia em Análise e Desenvolvimento de Sistemas – Discursiva 40

```

1 Algoritmo MatrizENADE2008
2 variaveis
3   M[0..2][0..3], I, J, C : inteiro
4 inicio
5   C ← 0
6   para I ← 0 ate 2 passo 1 faca
7     inicio
8       para J ← 0 ate 3 passo 1 faca
9         inicio
10          C ← C + 1
11          M[I][J] ← C
12        fim para
13      fim para
14    para I ← 0 ate 2 passo 1 faca
15      inicio
16        para J ← 0 até 3 passo 1 faca
17          inicio
18            C ← M[2-I][3-J]
19            M[I][J] ← C
20          fim para
21        fim para
22 fim algoritmo

```

Considerando a execução do algoritmo apresentada acima, faça o que se pede a seguir.

- A) Apresente os dados da matriz M ao término da execução da linha 12. (valor: 4,0 pontos)
- B) Apresente os dados da matriz M ao término da execução da linha 19. (valor: 6,0 pontos)

Gabarito oficial: ANULADA (o padrão de resposta traz a solução)

A)

1	2	3	4
5	6	7	8
9	10	11	12

B)

12	11	10	9
8	7	7	8
9	10	11	12

Por que B não é a matriz “invertida”? A cópia é feita na *própria* matriz, sem matriz auxiliar. A linha 0 recebe os valores da linha 2 (12, 11, 10, 9). Na linha 1, M[1][0] e M[1][1] recebem 8 e 7 de M[1][3] e M[1][2], mas M[1][2] e M[1][3] copiam M[1][1] e M[1][0], que *já* foram sobrescritos (7 e 8). Na linha 2, os valores vêm da linha 0, já alterada: 9, 10, 11, 12. É um bom exemplo do cuidado necessário ao modificar uma estrutura enquanto se lê dela.

Observação

O gabarito definitivo de 2008 registra a Discursiva 40 como **anulada**. O padrão de resposta publicado traz a solução acima sob o rótulo “Discursiva 39” (a 39 da prova é sobre diagrama de classes), e depois indica “Questão 40 – Discursiva (Anulada)”. Os valores conferem com a execução do algoritmo; por isso a questão foi mantida aqui.

Questão 82 ENADE 2011 – Tecnologia em Análise e Desenvolvimento de Sistemas – Discursiva 4

Considerando a execução do algoritmo abaixo, responda ao que se pede nos itens a e b.

```

01 algoritmo Vetores
02 variaveis
03   vetA[1..10], vetB[1..10], i: inteiro
04 inicio
05   para i <- 1 ate 10 passo 1 faca
06     vetB[i] <- 0
07     se resto(i,2) = 0 entao
08       vetA[i] <- i
09     senão
10       vetA[i] <- 2 * i
11     fimse
12   fimpara
13   para i <- 1 ate 10 passo 1 faca
14     enquanto(vetA[i] > i)
15       vetB[i] <- vetA[i]
16       vetA[i] <- vetA[i] - 1
17     fimenquanto
18   fimpara
19 fimalgoritmo

```

- a) Apresente os dados dos vetores vetA e vetB ao término da execução da linha 12. (valor: 5,0 pontos)
- b) Apresente os dados dos vetores vetA e vetB ao término da execução da linha 19. (valor: 5,0 pontos)

Padrão de resposta oficial (INEP)

Posição	1	2	3	4	5	6	7	8	9	10
a) após a linha 12										
vetA	2	2	6	4	10	6	14	8	18	10
vetB	0	0	0	0	0	0	0	0	0	0
b) após a linha 19										
vetA	1	2	3	4	5	6	7	8	9	10
vetB	2	0	4	0	6	0	8	0	10	0

Justificativa. No primeiro laço, posições pares recebem i e ímpares recebem $2i$. No segundo, o **enquanto** decrementa **vetA[i]** até chegar a i ; só as posições ímpares entram no laço, e **vetB[i]** guarda o último valor antes do decremento final, isto é, $i + 1$.

Questão 83 ENADE 2011 – Tecnologia em Análise e Desenvolvimento de Sistemas – Discursiva 5

Jogos de tabuleiro são atividades comuns de entretenimento na vida cotidiana das pessoas. Uma de suas características é a necessidade do uso de um tabuleiro com localizações bem definidas para o posicionamento de peças, podendo indicar também as fases do jogo. No livro **O Homem que Calculava**, de Malba Tahan (São Paulo: Record, 2002, p. 120), há uma história na qual um rei deveria efetuar o pagamento pelos serviços de um de seus conselheiros, dando-lhe uma certa quantidade de grãos de trigo a ser calculada da seguinte forma: coloca-se 1 grão de trigo na primeira casa do tabuleiro, 2 na segunda casa, 4 na terceira casa, 8 na quarta casa e assim dobrando-se sucessivamente até a última casa.

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

Considerando o tabuleiro 4×4 ilustrado acima, contendo a indicação da ordem das casas, construa um único algoritmo que:

- calcule, armazene em uma estrutura e escreva em ordem a quantidade de grãos de trigo em cada casa do tabuleiro; (valor: 8,0 pontos)
- calcule, armazene em uma variável e escreva a quantidade total de grãos de trigo presentes no tabuleiro. (valor: 2,0 pontos)

Padrão de resposta oficial (INEP)

A resposta do item (a) está no cálculo, armazenamento e escrita da variável **Trigo**; a do item (b), no bom funcionamento da variável **TotalTrigo**. (Os comentários não são necessários na resposta.)

```
início
// declara variável do tipo vetor
tipo Trigo = vetor [1..16] de inteiros;
// declaração de variáveis simples
inteiro: TotalTrigo, Indice1;
// inicializa acumulador de total de trigo
TotalTrigo <- 0;
// determinando a qtde de trigo em cada casa
Para Indice1 de 0 até 15 passo 1 faça
    Trigo[Indice1 + 1] <- pot(2, Indice1);
    TotalTrigo <- TotalTrigo + Trigo[Indice1 + 1];
Fim para;
// escrevendo o resultado
Para Indice1 de 1 até 16 passo 1 faça
    Escreva("Quantidade de grãos de trigo na casa ", Indice1, " = ", Trigo[Indice1
]);
Fim para;
Escreva("Total de grãos de trigo no tabuleiro = ", TotalTrigo);
Fim.
```

Observação

O total é a soma de uma progressão geométrica: $\sum_{k=0}^{15} 2^k = 2^{16} - 1 = 65\,535$ grãos. No tabuleiro de xadrez da lenda (8×8), seriam $2^{64} - 1 \approx 1,8 \times 10^{19}$ grãos, que não cabem num inteiro de 32

bits (e mal cabem num inteiro sem sinal de 64 bits) — bom gancho para discutir *overflow*. Em vez de `pot(2, i)`, pode-se dobrar o valor da casa anterior (`Trigo[i] <- 2 * Trigo[i-1]`), evitando a função de potência.