Estruturas de Dados

Aula 14 — Caminhos Mínimos em Grafos

Prof. Ana Carolina Sokolonski

Bacharelado em Sistemas de Informação
Instituto Federal da Bahia — Campus Feira de Santana

2026

Agenda

Motivação e Definição

Dijkstra

Bellman-Ford

Floyd-Warshall

Comparação

Exercícios

Caminhos Mínimos

Dijkstra, Bellman-Ford e Floyd-Warshall

Motivação e Problema Formal

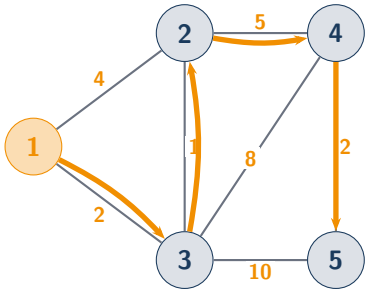
Aplicações reais

GPS/Mapas: menor rota entre duas cidades
Roteamento de rede: menor latência
Logística: minimizar distância/custo
Jogos: pathfinding de personagens
Finanças: arbitragem de câmbio (Bellman-Ford)

Variantes do problema

- **SSSP** (*Single-Source*): menor caminho de uma fonte s a todos os outros
- **APSP** (*All-Pairs*): menor caminho entre todo par (u, v)
- **STSP** (*Single-Target*): de todos até um destino

Grafo ponderado — qual a rota mais curta de 1 a 5?



custo = 10

Dijkstra

Algoritmo guloso para pesos não-negativos

Algoritmo de Dijkstra

Ideia gulosa

Mantém um conjunto S de vértices com distância **finalizada**. A cada passo, adiciona a S o vértice $u \notin S$ com **menor estimativa** $d[u]$ e **relaxa** todas as arestas que partem de u.

Relaxamento de (u, v, w)

se $d[u] + w(u, v) < d[v]$ então $d[v] \leftarrow d[u] + w(u, v)$

Só melhora a estimativa, nunca piora.

Restrição importante

Funciona **somente** com pesos $w(u, v) \geq 0$. Com arestas negativas, a garantia gulosa falha.

Pseudocódigo

Dijkstra(G, s):

$d[v] \leftarrow \infty$ para todo $v \in V$
$d[s] \leftarrow 0$; $S \leftarrow \emptyset$
$Q \leftarrow V$ *(fila de prioridade)*

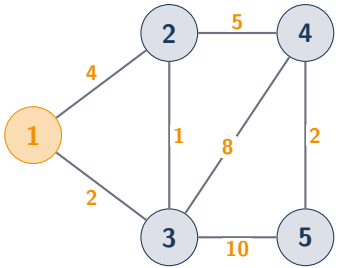
enquanto $Q \neq \emptyset$:
  $u \leftarrow \textsc{ExtractMin}(Q)$
  $S \leftarrow S \cup \{u\}$
  para cada $(u, v) \in E$:
    $\textsc{Relaxar}(u, v, w)$

Complexidade

Vetor simples:	$O(V^2)$
Heap binário:	$O((V + E) \log V)$
Heap de Fibonacci:	$O(E + V \log V)$

Dijkstra — Trace Passo a Passo

Grafo — fonte: vértice 1



Evolução de $d[\,]$ a cada passo (fonte = 1)

Passo	Escolhe	$d[1]$	$d[2]$	$d[3]$	$d[4]$	$d[5]$
Início	—	**0**	∞	∞	∞	∞
1	**1**	0	4	2	∞	∞
2	**3**	0	3	2	10	12
3	**2**	0	3	2	8	12
4	**4**	0	3	2	8	10
5	**5**	0	3	2	8	10

Caminhos mínimos a partir de 1:

$1 \to 2$: $1 \to 3 \to 2 = 3$

$1 \to 3$: $1 \to 3 = 2$

$1 \to 4$: $1 \to 3 \to 2 \to 4 = 8$

$1 \to 5$: $1 \to 3 \to 2 \to 4 \to 5 = 10$

Dijkstra — Implementação em C

dijkstra.c — $O(V^2)$ com matriz

```c
#define MAXV 100
#define INF  1e9
void dijkstra(double w[MAXV][MAXV],
              int n, int s, double d[]){
  bool finalizado[MAXV] = {false};
  for(int i=0; i<n; i++) d[i]=INF;
  d[s] = 0;
  for(int iter=0; iter<n-1; iter++){
//escolhe não-finalizado com menor d
    int u=-1;
    for(int v=0; v<n; v++)
      if(!finalizado[v]&&(u==-1||d[v]<d[
          u])) u=v;
      if(d[u]==INF) break;
      finalizado[u]=true;
//relaxa vizinhos de u
      for(int v=0; v<n; v++)
        if(w[u][v]>0 && !finalizado[v])
          if(d[u]+w[u][v]<d[v]) d[v]=d[u
              ]+w[u][v];}}
```

Observações

`w[u][v] = 0` indica ausência de aresta

Usa **vetor simples** para a fila de prioridade: $O(V^2)$

Para grafos esparsos, preferir **heap binário** com lista de adjacência: $O((V+E)\log V)$

Reconstruindo o caminho

Mantém vetor `pai[v]`: quando $d[v]$ é atualizado via u, faz `pai[v] = u`.

Para imprimir o caminho de s a t, imprime o caminho de s a `pai[t]` e, depois, o de t a t. (Recursão simples.)

Quando falha

Qualquer aresta com $w < 0$ pode levar Dijkstra a produzir resultados incorretos. Nesses casos:

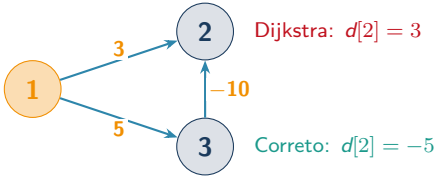
Bellman-Ford

Caminhos mínimos com pesos negativos

Por quê Dijkstra falha?

Dijkstra finaliza $d[v]$ de forma irreversível. Com aresta negativa, um caminho via vértice **ainda não finalizado** pode ser mais curto.

Dijkstra erra neste dígrafo:



Algoritmo Bellman-Ford

Repete o relaxamento de **todas** as arestas $|V| - 1$ vezes. Garante que após k iterações, $d[v]$ é o custo do menor caminho usando no máximo k arestas.

Pseudocódigo

$d[v] \leftarrow \infty$ para todo v; $d[s] \leftarrow 0$

repita $|V| - 1$ vezes:
  para cada aresta $(u, v, w) \in E$:
    RELAXAR(u, v, w)

 # detectar ciclo negativo:
para cada (u, v, w):
  se $d[u] + w < d[v]$: **ciclo negativo!**

Complexidade

$O(V \cdot E)$ — mais lento que Dijkstra, mas correto para pesos negativos (sem ciclos negativos).

Bellman-Ford — Implementação em C

bellman_ford.c

```c
typedef struct {
    int u, v, w;//aresta u->v, peso w
} Aresta;
//Retorna false se há ciclo negativo
bool bellmanFord(Aresta *E,int na,int nv,
    int s,int d[]){
  for(int i=0; i<nv; i++) d[i]=INF;
  d[s] = 0;
  for (int iter=0; iter<nv-1; iter++)
    for (int e=0; e<na; e++){
      int u=E[e].u, v=E[e].v, w=E[e].w;
      if (d[u]!=INF && d[u]+w < d[v])
        d[v] = d[u]+w;
    }
  for (int e=0; e<na; e++)
    if (d[E[e].u]!=INF && d[E[e].u]+E[e].w
        < d[E[e].v])
        return false; //ciclo negativo
  return true;
}
```

▪ Trace para o exemplo

Grafo: $1 \to 2{:}3$, $1 \to 3{:}5$, $3 \to 2{:}-10$

Iter.	$d[1]$	$d[2]$	$d[3]$
Início	0	∞	∞
1	0	3	5
2	0	-5	5

Correto: $d[2] = -5$ ✓

▪ Ciclo negativo

Se, após $|V| - 1$ iterações, ainda for possível relaxar alguma aresta, então existe um **ciclo de peso negativo** no grafo. Nesse caso, o caminho mínimo é $-\infty$ (indefinido). **Aplicação:** detectar **oportunidades de arbitragem** no mercado de câmbio.

Floyd-Warshall

Caminhos mínimos entre todos os pares

Floyd-Warshall — Todos os Pares

Problema

Encontrar o menor caminho entre **todos os pares** de vértices (i, j). Executar Dijkstra $|V|$ vezes daria $O(V^3)$ (ou $O(V(V+E)\log V)$ com heap). **Floyd-Warshall** também é $O(V^3)$, mas com uma constante menor e um código mais simples.

Ideia: programação dinâmica

Define $d^{(k)}[i][j]$ = menor distância de i a j usando apenas vértices intermediários $\{1, \ldots, k\}$.

Recorrência:

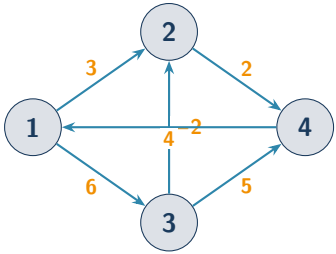
$$d^{(k)}[i][j] = \min\Big(d^{(k-1)}[i][j],\ d^{(k-1)}[i][k] + d^{(k-1)}[k][j]\Big)$$

Percorre cada vértice k como intermediário potencial.

Complexidade e uso

Tempo: $O(v^3)$
Espaço: $O(v^2)$
Aceita **pesos negativos** (sem ciclos negativos)
Ideal para grafos **densos** e pequenos (ex.: $V \leq 500$)

Exemplo: dígrafo de 4 vértices



Floyd-Warshall — Implementação em C

floyd_warshall.c

```c
#define MAXV 100
#define INF  1e9
//d[][] inicializado com pesos diretos: d[i][j]
    = w(i,j) ou INF se não existe, d[i][i] = 0
void floydWarshall(double d[MAXV][MAXV],int n){
  for (int k = 0; k < n; k++)
    for (int i = 0; i < n; i++)
      for (int j = 0; j < n; j++)
        if (d[i][k]+d[k][j] < d[i][j])
            d[i][j] = d[i][k]+d[k][j];}
/* Detectar ciclo negativo: d[i][i] < 0 */
bool temCicloNeg(double d[][MAXV], int n) {
  for(int i=0; i<n; i++)
    if(d[i][i] < 0) return true;
  return false;}
```

Matriz d antes e depois

Antes (arestas diretas)

	1	2	3	4
1	0	3	6	∞
2	∞	0	∞	2
3	∞	-2	0	5
4	4	∞	∞	0

Depois (caminhos mínimos)

	1	2	3	4
1	0	3	6	5
2	6	0	∞	2
3	2	-2	0	0
4	4	7	10	0

Comparação

Qual algoritmo usar?

Comparação dos Algoritmos

Critério	Dijkstra	Bellman-Ford	Floyd-Warshall
Tipo do problema	SSSP	SSSP	APSP
Pesos negativos	**Não**	**Sim**	**Sim**
Ciclos negativos	N/A	Detecta	Detecta
Complexidade	$O(V^2)$ ou $O((V+E)\log V)$	$O(V \cdot E)$	$O(V^3)$
Implementação	Moderada	Simples	Muito simples
Grafos densos	OK	Lento	Bom
Grafos esparsos	Ótimo (heap)	OK	Lento
Uso típico	GPS, redes	Câmbio, redes	Grafos pequenos

Comparação dos Algoritmos

Use Dijkstra quando

Pesos ≥ 0 e grafo esparso ou médio. É o mais rápido na prática.

Use Bellman-Ford quando

Há pesos negativos ou precisa detectar ciclos negativos.

Use Floyd-Warshall quando

Precisa de **todos os pares** ou o grafo é pequeno e denso.

Exercícios

Pratique!

Exercícios Propostos

▪ Exercício 1 — Trace Dijkstra

Execute Dijkstra a partir do vértice **A** no grafo não-dirigido ponderado:

A–*B*:1, *A*–*C*:4, *B*–*C*:2, *B*–*D*:6, *C*–*D*:3, *D*–*E*:2, *C*–*E*:5

Monte a tabela de $d[\,]$ a cada passo e indique o caminho mínimo de A até E.

▪ Exercício 2 — Bellman-Ford

No dígrafo: $1 \to 2 : 4$, $1 \to 3 : 5$, $2 \to 3 : -3$, $3 \to 4 : 2$, $4 \to 2 : 1$.

Execute Bellman-Ford a partir de 1. Existe ciclo negativo? Justifique.

▪ Exercício 3 — Floyd-Warshall

Dado o dígrafo de 3 vértices: $1 \to 2 : 3$, $2 \to 3 : -2$, $3 \to 1 : 1$.

Preencha a matriz d após cada iteração $k = 1, 2, 3$ do Floyd-Warshall. Há ciclo negativo?

▪ Exercício 4 — Implementação

Implemente em C o Dijkstra com **lista de adjacência** e **heap binário** (usando array como min-heap). Compare o tempo de execução com a versão $O(V^2)$ para grafos com $V = 1000$ e diferentes densidades.

Fim da Aula 14

Dúvidas e próximos passos

Próxima Aula e Referências

Próxima aula — Árvore Geradora Mínima

- **Kruskal**: ordena arestas, usa Union-Find
- **Prim**: cresce árvore vértice a vértice
- Aplicações: redes, infraestrutura, clustering
- Diferença entre MST e caminho mínimo

Referências

- CORMEN et al. *Introduction to Algorithms*, 4ª ed. Cap. 22–23.

Resumo da Aula 14

- Relaxamento: $d[v] \leftarrow \min(d[v], d[u] + w)$
- Dijkstra: guloso, $O(V^2)$, pesos ≥ 0
- Bellman-Ford: $O(VE)$, pesos negativos, detecta ciclos
- Floyd-Warshall: $O(V^3)$, todos os pares, DP

Dúvidas?

carolsoko@ifba.edu.br

IFBA – Campus Feira de Santana