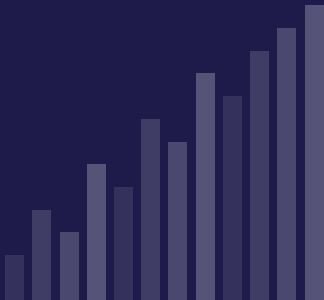

AULA 31 · BLOCO 4: GRAFOS

Caminhos mínimos: Bellman-Ford e Dijkstra

Seções 24.1 e 24.3

Estruturas de dados e seus algoritmos · José Dihego · IFBA



Roteiro da aula

Ao final, você será capaz de

- ▶ Definir caminhos mínimos com pesos e o papel dos ciclos negativos
- ▶ Usar o relaxamento como operação básica
- ▶ Executar Bellman-Ford ($O(VE)$) e Dijkstra ($O((V + E) \lg V)$ com heap)
- ▶ Medir vetor contra heap no Dijkstra e o custo do Bellman-Ford



Teoria

Relaxamento, Bellman-Ford, Dijkstra, a escolha da fila de prioridade



Prática em C

Dijkstra com heap na Figura 24.6



Experimento

Três implementações em grafos esparsos



Conexões

GPS, roteamento na internet, logística e jogos

Relaxamento

- ▶ $\delta(s, v)$: peso do caminho mínimo de s a v ; $v.d$ é a estimativa atual, começando em ∞ .
- ▶ $\text{RELAX}(u, v, w)$: **se** $v.d > u.d + w(u, v)$ **então** $v.d \leftarrow u.d + w(u, v)$ e $v.\pi \leftarrow u$.
- ▶ Todos os algoritmos do capítulo 24 diferem só na **ordem** e no **número** de relaxamentos.
- ▶ Subcaminhos de caminhos mínimos são mínimos (lema 24.1): a base da programação dinâmica e da gulosa.
- ▶ Ciclo de peso negativo alcançável: δ não está definido ($-\infty$).

Bellman-Ford

- ▶ Aceita pesos **negativos**.
- ▶ Após i passadas, acerta todo caminho mínimo com até i arestas.
- ▶ Detecta ciclos negativos.
- ▶ Custo $\Theta(VE)$.

BELLMAN-FORD(G, w, s)

```
1 INITIALIZE-SINGLE-SOURCE( $G, s$ )
2 para  $i \leftarrow 1$  até  $|G.V| - 1$  faça
3   para cada aresta  $(u, v) \in G.E$  faça
4     RELAX( $u, v, w$ )
5 para cada aresta  $(u, v) \in G.E$  faça
6   se  $v.d > u.d + w(u, v)$  então
7     retorne FALSO                                // ciclo negativo
8 retorne VERDADEIRO
```

Bellman-Ford (exemplo)

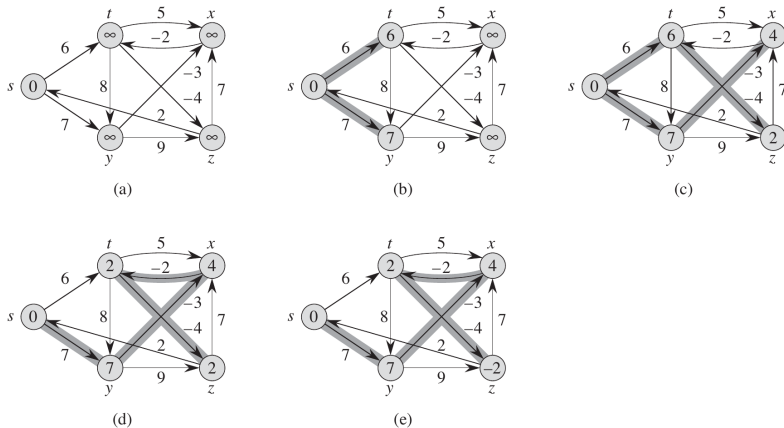


Figura 24.4 de Cormen et al. (2009) — Bellman-Ford com pesos negativos: cada passada relaxa todas as arestas

Dijkstra

- ▶ Exige pesos **não negativos**.
- ▶ Guloso: o vértice de menor d em Q já tem $d = \delta$ (teorema 24.6).
- ▶ Vetor: $O(V^2)$. Heap: $O((V + E) \lg V)$. Heap de Fibonacci: $O(V \lg V + E)$.

Dijkstra(G, w, s)

```
1 INITIALIZE-SINGLE-SOURCE( $G, s$ )  
2  $S \leftarrow \emptyset$ ;  $Q \leftarrow G.V$  // fila de mínimo por  $d$   
3 enquanto  $Q \neq \emptyset$  faça  
4    $u \leftarrow \text{EXTRACT-MIN}(Q)$   
5    $S \leftarrow S \cup \{u\}$   
6   para cada  $v \in G.Adj[u]$  faça  
7     RELAX( $u, v, w$ ) // DECREASE-KEY
```

Dijkstra (exemplo)

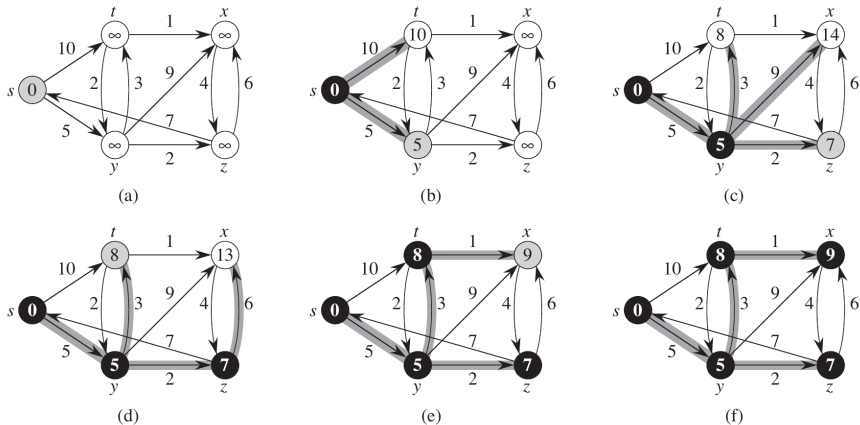


Figura 24.6 de Cormen et al. (2009) — Dijkstra a partir de s : em preto, os vértices já em S ; sombreadas, as arestas



Prática em C

- ▶ Heap de mínimo com `pos[]`: Decrease-Key em $O(\lg V)$
- ▶ O grafo da Figura 24.6
- ▶ Imprime d e o caminho até x pelos predecessores π

Dijkstra em C (1/4)

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  /* 0 grafo da Figura 24.6 (vértices s, t, x, y, z) com pesos não negativos. */
5  #define V 5
6  #define INF 1000000000
7  const char nome[V] = {'s', 't', 'x', 'y', 'z'};
8  int w[V][V]; // 0 = sem aresta (pesos positivos)
9  int d[V], pi[V], heap[V], pos[V], tam;
10
11 void troca(int i, int j) {
12     int t = heap[i]; heap[i] = heap[j]; heap[j] = t;
13     pos[heap[i]] = i; pos[heap[j]] = j;
14 }
```

Dijkstra em C (2/4)

```
15 void sobe(int i) { // DECREASE-KEY: o vértice sobe no heap
16     while (i > 0 && d[heap[(i - 1) / 2]] > d[heap[i]]) {
17         troca(i, (i - 1) / 2);
18         i = (i - 1) / 2;
19     }
20 }
21 int extrai_min(void) { // EXTRACT-MIN
22     int u = heap[0];
23     troca(0, --tam);
24     for (int i = 0;;) {
25         int e = 2 * i + 1, dd = e + 1, m = i;
26         if (e < tam && d[heap[e]] < d[heap[m]]) m = e;
27         if (dd < tam && d[heap[dd]] < d[heap[m]]) m = dd;
28         if (m == i) break;
29         troca(i, m);
30         i = m;
```

Dijkstra em C (3/4)

```
31     }
32     return u;
33 }
34
35 int main(void) {
36     int s;
37     if (scanf("%d", &s) != 1 || s < 0 || s >= V) return 1;
38     int arestas[][3] = {{0, 1, 10}, {0, 3, 5}, {1, 2, 1}, {1, 3, 2}, {2, 4, 4},
39                         {3, 1, 3}, {3, 2, 9}, {3, 4, 2}, {4, 0, 7}, {4, 2, 6}};
40     for (int k = 0; k < 10; k++) w[arestas[k][0]][arestas[k][1]] = arestas[k][2];
41     for (int u = 0; u < V; u++) { d[u] = INF; pi[u] = -1; heap[u] = u; pos[u] = u; }
42     d[s] = 0; // INITIALIZE-SINGLE-SOURCE
43     tam = V;
44     sobe(pos[s]);
45     while (tam > 0) {
46         int u = extrai_min(); // u entra em S: d[u] é definitivo
```

Dijkstra em C (4/4)

```
47     for (int v = 0; v < V; v++)
48         if (w[u][v] && d[u] + w[u][v] < d[v]) {    // RELAX(u, v, w)
49             d[v] = d[u] + w[u][v];
50             pi[v] = u;
51             sobe(pos[v]);
52         }
53     }
54     for (int u = 0; u < V; u++) printf("%c:%d ", nome[u], d[u]);
55     printf("\ncaminho até x:");
56     int caminho[V], k = 0;
57     for (int u = 2; u != -1; u = pi[u]) caminho[k++] = u;
58     while (k) printf(" %c", nome[caminho[--k]]);
59     printf("\n");
60     return 0;
61 }
```



Experimento

- ▶ Grafo com $E = 8V$ e pesos de 1 a 100
- ▶ Vetor: $V^2 + E$; heap: $\Theta((V + E) \lg V)$; Bellman-Ford: $(V - 1)E$
- ▶ V de 250 a 4.000

A estrutura de dados decide

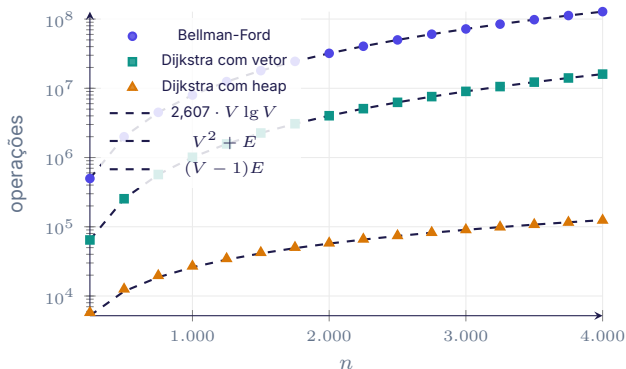
Hipótese. Com vetor, cada EXTRACT-MIN varre V posições: $V^2 + E = V^2 + 8V$ operações. Com heap, $\Theta((V + E) \lg V)$. Bellman-Ford relaxa as E arestas $V - 1$ vezes: exatamente $8V(V - 1)$.

Método

1. Mesmo grafo para os três algoritmos
2. Vetor: varreduras + relaxamentos
3. Heap: comparações do heap + relaxamentos
4. Bellman-Ford sem parada antecipada

🧪 EXPERIMENTO

Operações das três implementações



n	medido	razão
250,0	5.759	2,892
1.000	26.930	2,702
1.750	50.204	2,663
2.500	74.193	2,629
3.250	98.911	2,609
4.000	123.972	2,590

razão = medido $\div V \lg V$

✓ desvio máx. 1,1%

Para $V = 4.000$: 124 mil operações com heap, 16 milhões com vetor, 128 milhões no Bellman-Ford.

🧪 EXPERIMENTO

O que o experimento mostrou

As contagens batem: $V^2 + E$ e $(V - 1)E$ exatos, e $\Theta(V \lg V)$ com heap. Em grafos esparsos, o heap é **130 vezes** mais barato que o vetor para $V = 4.000$. Em grafos densos ($E \approx V^2$), o vetor volta a ser competitivo: a melhor estrutura depende da **densidade** do grafo.

➤ Reproduza: `gcc -O2 aula31_experimento.c -o exp -lm && ./exp`



Conexões

- ▶ Navegação (GPS)
- ▶ Roteamento na internet
- ▶ Finanças
- ▶ Logística

Onde isso aparece na computação

Navegação (GPS)

Google Maps e Waze usam Dijkstra com acelerações (A^* , hierarquias de contração) em grafos de bilhões de arestas, com pesos que mudam com o trânsito.

Roteamento na internet

OSPF e IS-IS rodam Dijkstra em cada roteador; RIP e BGP são variações distribuídas do Bellman-Ford (vetor de distâncias).

Finanças

Com pesos $-\log(\text{câmbio})$, um ciclo negativo é uma oportunidade de arbitragem entre moedas: Bellman-Ford a detecta.

Logística

Roteirização de entregas e planejamento de rotas de transporte público começam com caminhos mínimos entre todos os pontos de interesse.

Para casa

Exercícios

1. (24.1-1) Execute Bellman-Ford na Figura 24.4 a partir de z .
2. (24.3-1) Execute Dijkstra na Figura 24.2 a partir de s e depois de z .
3. (24.3-2) Dê um grafo com aresta negativa em que Dijkstra erra.
4. (24.3-4) Verifique em $O(V + E)$ se valores d e π dados são caminhos mínimos.
5. Repita o experimento com grafos densos ($E = V^2/4$). Onde as curvas do vetor e do heap se cruzam?

Leitura

Cormen et al., seções 24.1 a 24.3.

Código

josedihego.net/estruturas