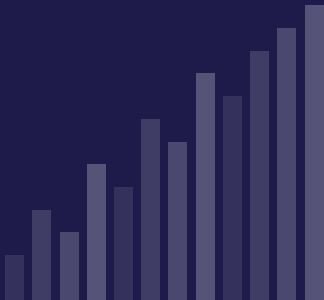

AULA 30 · BLOCO 4: GRAFOS

Conjuntos disjuntos e árvores geradoras mínimas

Capítulo 21 e capítulo 23

Estruturas de dados e seus algoritmos · José Dihego · IFBA



Roteiro da aula

Ao final, você será capaz de

- ▶ Implementar conjuntos disjuntos com união por posto e compressão de caminho
- ▶ Enunciar a propriedade de corte que justifica os algoritmos gulosos de AGM
- ▶ Executar Kruskal e Prim
- ▶ Medir o custo quase constante das operações de conjuntos disjuntos



Teoria

Floresta de conjuntos disjuntos; corte e aresta leve;
Kruskal e Prim



Prática em C

Kruskal com conjuntos disjuntos no grafo da Figura 23.1



Experimento

Ingênuo contra heurísticas: $\Theta(n)$ contra constante



Conexões

Redes elétricas, telecomunicações, agrupamento e visão computacional

Conjuntos disjuntos

Operações: $\text{Make-Set}(x)$, $\text{Find-Set}(x)$ (o representante) e $\text{UNION}(x, y)$.

- ▶ Cada conjunto é uma árvore; a raiz é o representante.
- ▶ **União por posto:** a árvore mais baixa vira filha da mais alta.
- ▶ **Compressão de caminho:** na busca, todos os nós do caminho passam a apontar para a raiz.
- ▶ Com as duas: $O(m \alpha(n))$ para m operações, com $\alpha(n) \leq 4$ para qualquer n do universo.

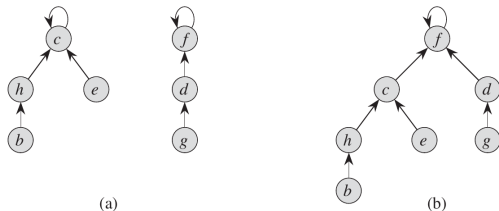


Figura 21.4 de Cormen et al. (2009) — floresta de conjuntos disjuntos: (a) dois conjuntos; (b) depois de UNION.

Compressão de caminho

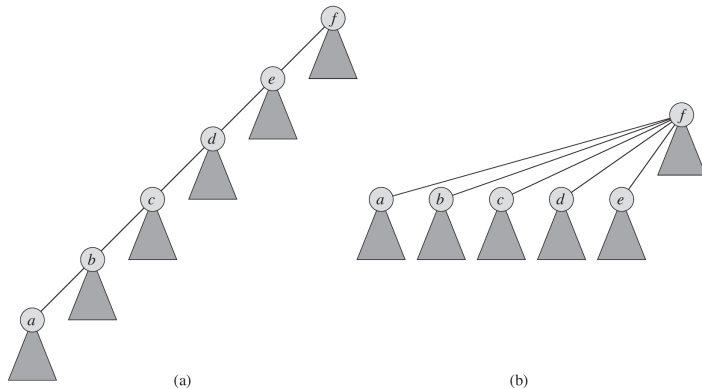


Figura 21.5 de Cormen et al. (2009) — $\text{Find-Set}(a)$ antes (a) e depois (b): todo o caminho aponta para a raiz.

Árvore geradora mínima

Num grafo conexo com pesos, escolher $V - 1$ arestas que ligam todos os vértices com **peso total mínimo**.

Teorema 23.1 (corte): para qualquer corte que respeite as arestas já escolhidas, a aresta **leve** (mais barata) que cruza o corte é segura.

Os dois algoritmos são **gulosos**: sempre pegam a aresta segura mais barata.

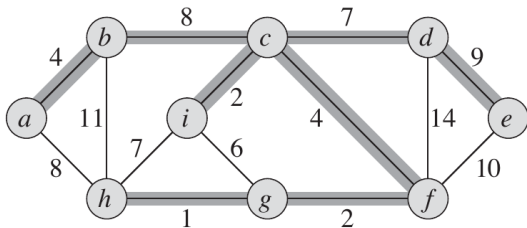


Figura 23.1 de Cormen et al. (2009) — uma AGM de peso 37 (arestas sombreadas). Trocar (b, c) por (a, h) dá outra AGM de mesmo peso.

Kruskal e Prim

Kruskal: ordene as arestas; para cada uma, se liga componentes diferentes (FIND-Set diferentes), escolha-a e una. $O(E \lg V)$, dominado pela ordenação.

Prim: cresça uma árvore a partir de uma raiz, sempre pela aresta mais leve que sai dela, com fila de prioridade. $O(E \lg V)$ com heap binário.

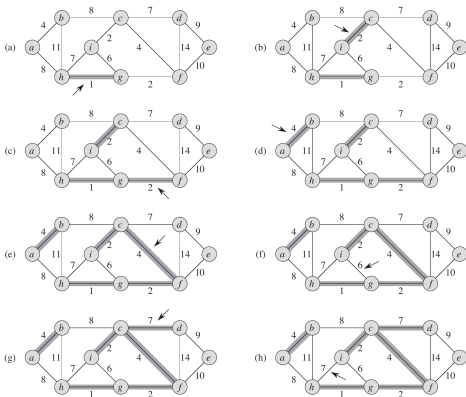


Figura 23.4 de Cormen et al. (2009) — Kruskal na Figura 23.1:



Prática em C

- ▶ acha com compressão; une por posto
- ▶ Arestas da Figura 23.1 ordenadas com qsort
- ▶ Imprime as arestas escolhidas e o peso total (37)

Kruskal com conjuntos disjuntos (1/4)

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  /* Floresta de conjuntos disjuntos com união por posto e compressão de caminho. */
5  int pai[16], posto[16];
6
7  void cria(int x) { pai[x] = x; posto[x] = 0; }      // MAKE-SET
8
9  // FIND-SET com compressão de caminho
10 int acha(int x) {
11     if (pai[x] != x) pai[x] = acha(pai[x]);
12     return pai[x];
13 }
```

Kruskal com conjuntos disjuntos (2/4)

```
14
15 void une(int x, int y) {                               // UNION por posto
16     x = acha(x); y = acha(y);
17     if (posto[x] > posto[y]) pai[y] = x;
18     else {
19         pai[x] = y;
20         if (posto[x] == posto[y]) posto[y]++;
21     }
22 }
23
24 typedef struct { int u, v, w; } Aresta;
```

Kruskal com conjuntos disjuntos (3/4)

```
26 int compara(const void *a, const void *b) {
27     return ((const Aresta *)a)->w - ((const Aresta *)b)->w;
28 }
29
30 int main(void) {
31     int qualquer;
32     if (scanf("%d", &qualquer) != 1) return 1;
33     /* 0 grafo da Figura 23.1: vértices a..i. */
34     Aresta E[] = {{0, 1, 4}, {0, 7, 8}, {1, 2, 8}, {1, 7, 11}, {2, 3, 7}, {2, 8, 2},
35                  {2, 5, 4}, {3, 4, 9}, {3, 5, 14}, {4, 5, 10}, {5, 6, 2}, {6, 7, 1},
36                  {6, 8, 6}, {7, 8, 7}};
37     int m = sizeof E / sizeof E[0], total = 0;
```

Kruskal com conjuntos disjuntos (4/4)

```
38     for (int x = 0; x < 9; x++) cria(x);
39     qsort(E, m, sizeof(Aresta), compara);           // arestas em ordem de peso
40     printf("arestas escolhidas:");
41     for (int k = 0; k < m; k++)
42         if (acha(E[k].u) != acha(E[k].v)) {         // não forma ciclo: entra na árvore
43             une(E[k].u, E[k].v);
44             total += E[k].w;
45             printf(" %c-%c", 'a' + E[k].u, 'a' + E[k].v);
46         }
47     printf("\n peso total %d\n", total);
48     return 0;
49 }
```



Experimento

- ▶ Uniões em corrente (0-1, 1-2, ...): o pior caso do ingênuo
- ▶ 4 buscas aleatórias para cada união
- ▶ Ingênuo até $n = 16.000$; heurísticas até 2 milhões

Quase constante por operação

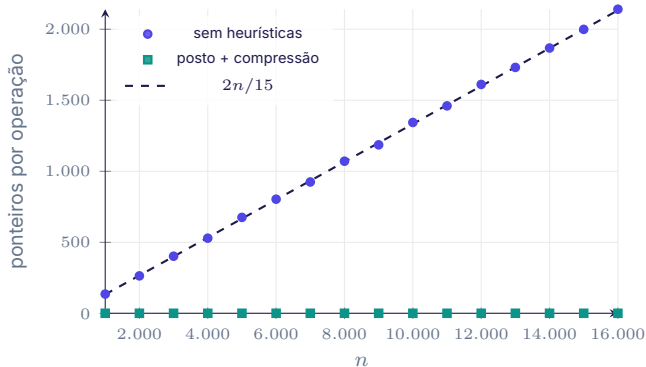
Hipótese. Sem heurísticas, as uniões em corrente formam uma lista; uma busca aleatória sobe em média $u^2/(2n)$ ponteiros depois de u uniões, o que dá $\approx \frac{2n}{15}$ por operação: **linear**. Com união por posto e compressão de caminho, o custo por operação é $O(\alpha(n))$: na prática, **constante**.

Método

1. $m = 5n$ operações: uniões $x-(x+1)$ e 4 buscas sorteadas
2. Contador de ponteiros seguidos até a raiz
3. Ingênuo: n de 1.000 a 16.000
4. Heurísticas: n de 2^{12} a 2^{21}

🧪 EXPERIMENTO

Ingênuo: linear por operação



n	medido	razão
1.000	135,8	1,019
4.000	529,2	0,992
7.000	924,8	0,991
10.000	1.344	1,008
13.000	1.731	0,999
16.000	2.141	1,004

razão = medido $\div$ $2n/15$

✓ desvio máx. 1,1%

Com $n = 16.000$: 2.141 ponteiros por operação sem heurísticas; 0,6 com elas.

EXPERIMENTO

Com heurísticas: constante



n	medido	razão
4.096	0,597	0,597
16.384	0,598	0,598
65.536	0,600	0,600
262.144	0,600	0,600
1.048.576	0,600	0,600
2.097.152	0,600	0,600

razão = medido $\div$ constante

✓ desvio máx. 0,0%

0,60 ponteiro por operação de quatro mil a dois milhões de elementos: $\alpha(n)$ na prática.

EXPERIMENTO

O que o experimento mostrou

Sem heurísticas, cada operação custa $\approx 2n/15$: com 16 mil elementos, mais de 2 mil passos. Com união por posto e compressão de caminho, o custo cai para **0,6** e não muda até dois milhões de elementos. É o que torna o Kruskal $O(E \lg V)$: o gargalo passa a ser só a ordenação das arestas.

>_ Reproduza: `gcc -O2 aula30_experimento.c -o exp -lm && ./exp`



Conexões

- ▶ Redes de infraestrutura
- ▶ Agrupamento de dados
- ▶ Visão computacional
- ▶ Compiladores e redes

Onde isso aparece na computação

Redes de infraestrutura

Projetar redes elétricas, de fibra óptica ou de saneamento ligando todos os pontos com o menor custo de cabo é uma AGM (Borůvka, 1926, eletrificação da Morávia).

Agrupamento de dados

Remover as $k - 1$ arestas mais pesadas da AGM separa os dados em k grupos: o agrupamento por ligação simples (*single linkage*).

Visão computacional

A segmentação de imagens de Felzenszwalb e Huttenlocher une pixels parecidos com conjuntos disjuntos, no estilo do Kruskal.

Compiladores e redes

Conjuntos disjuntos resolvem unificação de tipos (inferência de tipos) e acompanham componentes conexas de redes que mudam em tempo real.

Para casa

Exercícios

1. (21.3-1) Refaça o exercício 21.2-2 com união por posto e compressão de caminho.
2. (23.1-1) Mostre que uma aresta de peso mínimo pertence a alguma AGM.
3. (23.2-1) Kruskal pode dar AGMs diferentes conforme a ordem dos empates. Mostre que toda AGM pode ser obtida assim.
4. (23.2-8) O algoritmo “divida os vértices ao meio, ache as AGMs e ligue pela aresta mais leve funciona? Prove ou dê contraexemplo.
5. Implemente Prim com heap e confira que dá o mesmo peso que o Kruskal em grafos aleatórios.

Leitura

Cormen et al., seções 21.1 a 21.3 e capítulo 23.

Código

josedihego.net/estruturas