Análise de Algoritmos

Parte destes slides são adaptações de slides

do Prof. Paulo Feofiloff e do Prof. José Coelho de Pina.

Árvore geradora mínima

CLRS Cap 23

Árvore geradora mínima

Seja G um grafo conexo.

Uma árvore T em G é geradora
se contém todos os vértices de G.

Árvore geradora mínima

Seja G um grafo conexo.

Uma árvore T em G é geradora
se contém todos os vértices de G.

Problema: Dado G conexo com peso w_e para cada aresta e,
encontrar árvore geradora em G de peso mínimo.

Árvore geradora mínima

Seja G um grafo conexo.

Uma árvore T em G é geradora
se contém todos os vértices de G.

Problema: Dado G conexo com peso w_e para cada aresta e,
encontrar árvore geradora em G de peso mínimo.

O peso de uma árvore é a soma do peso de suas arestas.

Árvore geradora mínima

Seja G um grafo conexo.

Uma árvore T em G é geradora
se contém todos os vértices de G.

Problema: Dado G conexo com peso w_e para cada aresta e,
encontrar árvore geradora em G de peso mínimo.

O peso de uma árvore é a soma do peso de suas arestas.

Tal árvore é chamada de árvore geradora mínima em G.

Árvore geradora mínima

Seja G um grafo conexo.

Uma árvore T em G é geradora
se contém todos os vértices de G.

Problema: Dado G conexo com peso w_e para cada aresta e,
encontrar árvore geradora em G de peso mínimo.

O peso de uma árvore é a soma do peso de suas arestas.

Tal árvore é chamada de árvore geradora mínima em G.

MST: minimum spanning tree

Arestas seguras

Seja $G = (V, E)$ um grafo conexo.
Função w que atribui um peso w_e para cada aresta $e \in E$.

$A \subseteq E$ contido em alguma MST de (G, w).

Arestas seguras

Seja $G = (V, E)$ um grafo conexo.
Função w que atribui um peso w_e para cada aresta $e \in E$.

$A \subseteq E$ contido em alguma MST de (G, w).

Aresta $e \in E$ é segura para A se
$A \cup \{e\}$ está contido em alguma MST de (G, w).

Se A não é uma MST, então existe aresta segura para A.

Arestas seguras

Seja $G = (V, E)$ um grafo conexo.
Função w que atribui um peso w_e para cada aresta $e \in E$.

$A \subseteq E$ contido em alguma MST de (G, w).

Aresta $e \in E$ é segura para A se
$A \cup \{e\}$ está contido em alguma MST de (G, w).

Se A não é uma MST, então existe aresta segura para A.

GENÉRICO (G, w)

```
1  A ← ∅
2  enquanto A não é geradora faça
3      encontre aresta segura e para A
4      A ← A ∪ {e}
5  devolva A
```

Arestas seguras

Corte em G: partição $(S, V \setminus S)$.

Aresta e cruza o corte $(S, V \setminus S)$ se
exatamente um de seus extremos está em S.

Arestas seguras

Corte em G: partição $(S, V \setminus S)$.

Aresta e cruza o corte $(S, V \setminus S)$ se
exatamente um de seus extremos está em S.

Corte respeita $A \subseteq E$: nenhuma aresta de A o cruza.

Arestas seguras

Corte em G: partição $(S, V \setminus S)$.

Aresta e cruza o corte $(S, V \setminus S)$ se
exatamente um de seus extremos está em S.

Corte respeita $A \subseteq E$: nenhuma aresta de A o cruza.

Teorema: Se A está contida em MST de (G, w), então e com $w(e)$ mínimo em corte que respeita A é segura para A.

Arestas seguras

Corte em G: partição $(S, V \setminus S)$.

Aresta e cruza o corte $(S, V \setminus S)$ se
exatamente um de seus extremos está em S.

Corte respeita $A \subseteq E$: nenhuma aresta de A o cruza.

Teorema: Se A está contida em MST de (G, w), então e com $w(e)$ mínimo em corte que respeita A é segura para A.

Prova: Seja T uma MST em (G, w) que contém A.
Se e está em T, não há nada mais a provar.

Arestas seguras

Corte em G: partição $(S, V \setminus S)$.

Aresta e cruza o corte $(S, V \setminus S)$ se
exatamente um de seus extremos está em S.

Corte respeita $A \subseteq E$: nenhuma aresta de A o cruza.

Teorema: Se A está contida em MST de (G, w), então e com $w(e)$ mínimo em corte que respeita A é segura para A.

Prova: Seja T uma MST em (G, w) que contém A.
Se e está em T, não há nada mais a provar.

Se e não está em T, seja C o único circuito em $T + e$,
e seja $f \in C$ com $w(f)$ máximo que cruza o mesmo corte
(distinta de e em caso de empate).

Arestas seguras

Corte em G: partição $(S, V \setminus S)$.

Aresta e cruza o corte $(S, V \setminus S)$ se
exatamente um de seus extremos está em S.

Corte respeita $A \subseteq E$: nenhuma aresta de A o cruza.

Teorema: Se A está contida em MST de (G, w), então e com $w(e)$ mínimo em corte que respeita A é segura para A.

Prova: Seja T uma MST em (G, w) que contém A.
Se e está em T, não há nada mais a provar.

Se e não está em T, seja C o único circuito em $T + e$,
e seja $f \in C$ com $w(f)$ máximo que cruza o mesmo corte
(distinta de e em caso de empate).

Então $T' := T + e - f$ é MST e contém $A \cup \{e\}$. ■

Árvore geradora mínima

Os dois próximos algoritmos se enquadram no genérico.

Árvore geradora mínima

Os dois próximos algoritmos se enquadram no genérico.

O primeiro é o algoritmo de Kruskal que, em cada iteração, escolhe uma aresta segura mais leve possível.

O algoritmo de Kruskal vai aumentando uma floresta.

Árvore geradora mínima

Os dois próximos algoritmos se enquadram no genérico.

O primeiro é o algoritmo de Kruskal que, em cada iteração, escolhe uma aresta segura mais leve possível.

O algoritmo de Kruskal vai aumentando uma floresta.

O segundo é o algoritmo de Prim,
que mantém uma árvore T que contém um vértice s,
acrescentando em cada iteração uma aresta segura a T.

Árvore geradora mínima

Os dois próximos algoritmos se enquadram no genérico.

O primeiro é o algoritmo de Kruskal que, em cada iteração, escolhe uma aresta segura mais leve possível.

O algoritmo de Kruskal vai aumentando uma floresta.

O segundo é o algoritmo de Prim,
que mantém uma árvore T que contém um vértice s,
acrescentando em cada iteração uma aresta segura a T.

Os dois produzem uma MST de (G, w).

$n := |V(G)|$ e $m := |E(G)|$.

Algoritmo de Kruskal

```
KRUSKAL (G, w)
 1  A ← ∅
 2  sejam e_1, ..., e_m as arestas de G ordenadas por w
 3  para cada u ∈ V(G) faça MAKESET(u)
 5  para i ← 1 até m faça
 6      sejam u e v as pontas de e_i
 7      se FINDSET(u) ≠ FINDSET(v)
 8          então A ← A ∪ {e_i}
 9                UNION(u, v)
10  devolva A
```

Algoritmo de Kruskal

```
KRUSKAL (G, w)
 1  A ← ∅
 2  sejam e_1, ..., e_m as arestas de G ordenadas por w
 3  para cada u ∈ V(G) faça MAKESET(u)
 5  para i ← 1 até m faça
 6      sejam u e v as pontas de e_i
 7      se FINDSET(u) ≠ FINDSET(v)
 8          então A ← A ∪ {e_i}
 9                UNION(u, v)
10  devolva A
```

Correção:

Note que e_i na linha 8 é uma aresta segura para A.

Algoritmo de Kruskal

```
KRUSKAL (G, w)
 1  A ← ∅
 2  sejam e_1, ..., e_m as arestas de G ordenadas por w
 3  para cada u ∈ V(G) faça MAKESET(u)
 5  para i ← 1 até m faça
 6      sejam u e v as pontas de e_i
 7      se FINDSET(u) ≠ FINDSET(v)
 8          então A ← A ∪ {e_i}
 9                UNION(u, v)
10  devolva A
```

Algoritmo de Kruskal

```
KRUSKAL (G, w)
 1  A ← ∅
 2  sejam e_1, ..., e_m as arestas de G ordenadas por w
 3  para cada u ∈ V(G) faça MAKESET(u)
 5  para i ← 1 até m faça
 6      sejam u e v as pontas de e_i
 7      se FINDSET(u) ≠ FINDSET(v)
 8          então A ← A ∪ {e_i}
 9                UNION(u, v)
10  devolva A
```

Consumo de tempo do union-find:

MAKESET : $O(1)$ FINDSET e UNION : $O(\lg n)$

Algoritmo de Kruskal

```
KRUSKAL (G, w)
 1  A ← ∅
 2  sejam e_1, ..., e_m as arestas de G ordenadas por w
 3  para cada u ∈ V(G) faça MAKESET(u)
 5  para i ← 1 até m faça
 6      sejam u e v as pontas de e_i
 7      se FINDSET(u) ≠ FINDSET(v)
 8          então A ← A ∪ {e_i}
 9                UNION(u, v)
10  devolva A
```

Consumo de tempo do union-find:
MAKESET : $\mathrm{O}(1)$ FINDSET e UNION : $\mathrm{O}(\lg n)$

Consumo de tempo do Kruskal: $\mathrm{O}(m \lg n)$

Análise do Union-Find

CLRS cap 21

Coleção de conjuntos disjuntos

Queremos uma ED boa para representar uma partição de um conjunto, e as seguintes operações sobre a partição:

Coleção de conjuntos disjuntos

Queremos uma ED boa para representar uma partição de um conjunto, e as seguintes operações sobre a partição:

- MAKESET(x): cria um conjunto unitário com o elemento x;
- FINDSET(x): devolve o identificador do conjunto da partição que contém x;
- UNION(x, y): substitui os conjuntos da partição que contêm x e y pela união deles.

Coleção de conjuntos disjuntos

Queremos uma ED boa para representar uma partição de um conjunto, e as seguintes operações sobre a partição:

- MAKESET(x): cria um conjunto unitário com o elemento x;
- FINDSET(x): devolve o identificador do conjunto da partição que contém x;
- UNION(x, y): substitui os conjuntos da partição que contêm x e y pela união deles.

O identificador de um conjunto é um elemento do conjunto: o seu representante.

Coleção de conjuntos disjuntos

Queremos uma ED boa para representar uma partição de um conjunto, e as seguintes operações sobre a partição:

- MAKESET(x): cria um conjunto unitário com o elemento x;
- FINDSET(x): devolve o identificador do conjunto da partição que contém x;
- UNION(x, y): substitui os conjuntos da partição que contêm x e y pela união deles.

O identificador de um conjunto é um elemento do conjunto: o seu representante.

Como podemos armazenar cada conjunto da partição?

Implementação 1 do union-find

Make-Set (x)
1 pai$[x] \leftarrow x$

Implementação 1 do union-find

Make-Set (x)
1 pai$[x] \leftarrow x$

Find (x)
1 $r \leftarrow x$
2 **enquanto** pai$[r] \neq r$ **faça**
3 $\quad r \leftarrow$ pai$[r]$
4 **devolva** r

Implementação 1 do union-find

Make-Set (x)
1 pai$[x] \leftarrow x$

Find (x)
1 $r \leftarrow x$
2 **enquanto** pai$[r] \neq r$ **faça**
3 $\quad r \leftarrow$ pai$[r]$
4 **devolva** r

Union (x, y) $\quad \triangleright$ x e y representantes distintos
1 pai$[y] \leftarrow x$

Implementação 1 do union-find

Make-Set (x)
1 pai$[x] \leftarrow x$

Find (x)
1 $r \leftarrow x$
2 **enquanto** pai$[r] \neq r$ **faça**
3 $\quad r \leftarrow$ pai$[r]$
4 **devolva** r

Union (x, y) $\quad \triangleright$ x e y representantes distintos
1 pai$[y] \leftarrow x$

Consumo de tempo: do Find pode ser muito ruim... $\Theta(n)$.
Temos que fazer melhor...

Implementação 2

Heurística dos tamanhos

Make-Set (x)

1 pai$[x] \leftarrow x$

2 rank$[x] \leftarrow 0$

Implementação 2

Heurística dos tamanhos

Make-Set (x)

1 pai$[x] \leftarrow x$

2 rank$[x] \leftarrow 0$

Find (x): o mesmo de antes

Implementação 2

Heurística dos tamanhos

Make-Set (x)

```
1  pai[x] ← x
2  rank[x] ← 0
```

Find (x): o mesmo de antes

Union (x, y) ▷ x e y representantes distintos

```
1  se rank[x] ≥ rank[y]
2      então pai[y] ← x
3            se rank[x] = rank[y]
4                então rank[x] ← rank[x] + 1
5      senão pai[x] ← y
```

Implementação 2

Heurística dos tamanhos

Make-Set (x)

```
1  pai[x] ← x
2  rank[x] ← 0
```

Find (x): o mesmo de antes

Union (x, y) ▷ x e y representantes distintos

```
1  se rank[x] ≥ rank[y]
2      então pai[y] ← x
3            se rank[x] = rank[y]
4                então rank[x] ← rank[x] + 1
5      senão pai[x] ← y
```

Consumo de tempo: melhor... $\Theta(\lg n)$.
Dá para fazer melhor ainda!

Implementação 3

Heurística da compressão dos caminhos

```
Find (x)
1  if pai[x] ≠ x
2       então pai[x] ← Find (pai[x])
3  devolva pai[x]
```

Implementação 3

Heurística da compressão dos caminhos

```
Find (x)
1  if pai[x] ≠ x
2        então pai[x] ← Find (pai[x])
3  devolva pai[x]
```

Consumo *amortizado* de tempo de cada operação:

$$\mathrm{O}(\lg^* n),$$

onde $\lg^* n$ é o número de vezes que temos que aplicar o $\lg$ até atingir um número menor ou igual a 1.

Implementação 3

Heurística da compressão dos caminhos

```
Find (x)
1  if pai[x] ≠ x
2        então pai[x] ← Find (pai[x])
3  devolva pai[x]
```

Consumo *amortizado* de tempo de cada operação:

$$\mathrm{O}(\lg^* n),$$

onde $\lg^* n$ é o número de vezes que temos que aplicar o $\lg$ até atingir um número menor ou igual a 1.

Na verdade, é melhor do que isso,
e há uma análise justa, conforme discutido em aula.

Union-Find

```
Make-Set (x)
1   pai[x] ← x
2   rank[x] ← 0

Find (x)
1   if pai[x] ≠ x
2       então pai[x] ← Find (pai[x])
3   devolva pai[x]

Union (x, y)          ▷ x e y representantes distintos
1   se rank[x] ≥ rank[y]
2       então  pai[y] ← x
3              se rank[x] = rank[y]
4                  então rank[x] ← rank[x] + 1
5       senão pai[x] ← y
```

Union-Find

```
Union (x, y)
1   x' ← Find (x)
2   y' ← Find (y)
3   se x' ≠ y'
4       então Link (x', y')
```

```
Link (x, y)          ▷ x e y representantes distintos
1   se rank[x] ≥ rank[y]
2       então  pai[y] ← x
3              se rank[x] = rank[y]
4                  então rank[x] ← rank[x] + 1
5       senão pai[x] ← y
```

Consumo de tempo

Dada sequência de MAKESET, FINDSET e UNION,
converta-a em uma sequência de MAKESET, FINDSET e LINK.

Consumo de tempo

Dada sequência de MAKESET, FINDSET e UNION,
converta-a em uma sequência de MAKESET, FINDSET e LINK.

Sequência de m operações MAKESET, FINDSET e LINK
das quais n são MAKESET.

Consumo de tempo

Dada sequência de MAKESET, FINDSET e UNION,
converta-a em uma sequência de MAKESET, FINDSET e LINK.

Sequência de m operações MAKESET, FINDSET e LINK
das quais n são MAKESET.

Custo de pior caso de cada operação: $\mathrm{O}(\lg n)$.
Custo amortizado de cada operação: $\mathrm{O}(\lg^* n)$.

Consumo de tempo

Dada sequência de MAKESET, FINDSET e UNION,
converta-a em uma sequência de MAKESET, FINDSET e LINK.

Sequência de m operações MAKESET, FINDSET e LINK
das quais n são MAKESET.

Custo de pior caso de cada operação: $\mathrm{O}(\lg n)$.
Custo amortizado de cada operação: $\mathrm{O}(\lg^* n)$.

Para definir $\lg^* n$, seja $\lg^{(1)} x = \lg x$.
Para $i \geq 2$, seja $\lg^{(i)} x = \lg(\lg^{(i-1)} x)$.

Consumo de tempo

Dada sequência de MAKESET, FINDSET e UNION,
converta-a em uma sequência de MAKESET, FINDSET e LINK.

Sequência de m operações MAKESET, FINDSET e LINK
das quais n são MAKESET.

Custo de pior caso de cada operação: $\mathrm{O}(\lg n)$.
Custo amortizado de cada operação: $\mathrm{O}(\lg^* n)$.

Para definir $\lg^* n$, seja $\lg^{(1)} x = \lg x$.
Para $i \geq 2$, seja $\lg^{(i)} x = \lg(\lg^{(i-1)} x)$.

Então $\lg^* n \;=\; \min\{i : \lg^{(i)} n \leq 1\}$.

Consumo de tempo

Dada sequência de MAKESET, FINDSET e UNION,
converta-a em uma sequência de MAKESET, FINDSET e LINK.

Sequência de m operações MAKESET, FINDSET e LINK
das quais n são MAKESET.

Custo de pior caso de cada operação: $\mathrm{O}(\lg n)$.
Custo amortizado de cada operação: $\mathrm{O}(\lg^* n)$.

Para definir $\lg^* n$, seja $\lg^{(1)} x = \lg x$.
Para $i \geq 2$, seja $\lg^{(i)} x = \lg(\lg^{(i-1)} x)$.

Então $\lg^* n \;=\; \min\{i : \lg^{(i)} n \leq 1\}$.

A análise desta ED é vista na disciplina MAC6711.