

Árvore geradora mínima

CLRS Cap 23

Árvore geradora mínima

Seja G um grafo.

Um subgrafo que contém todos os vértices de G é dito **gerador**.

Uma **árvore geradora** é um subgrafo gerador que é uma árvore.

Árvore geradora mínima

Seja G um grafo.

Um subgrafo que contém todos os vértices de G é dito **gerador**.

Uma **árvore geradora** é um subgrafo gerador que é uma árvore.

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.

Um subgrafo que contém todos os vértices de G é dito **gerador**.

Uma **árvore geradora** é um subgrafo gerador que é uma árvore.

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 dos pesos de suas arestas.

Árvore geradora mínima

Seja G um grafo.

Um subgrafo que contém todos os vértices de G é dito **gerador**.

Uma **árvore geradora** é um subgrafo gerador que é uma árvore.

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 dos pesos 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 e w uma função que atribui um peso $w(e)$ para cada aresta $e \in E$.

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

Arestas seguras

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

Suponha que $A \subseteq E$ está 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) .

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

Arestas seguras

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

Suponha que $A \subseteq E$ está 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) .

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

GULOSO-GENÉRICO (G, w)

- 1 $A \leftarrow \emptyset$
- 2 **enquanto** A não é geradora **faça**
- 3 encontre aresta segura e para A
- 4 $A \leftarrow A \cup \{e\}$
- 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 toda 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 toda 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 . Considere uma tal e e suponha que e não está em T .

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 toda 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 . Considere uma tal e e suponha que e não está em T .

Então $T + e$ contém um circuito, que tem pelo menos uma outra aresta f de T que cruza o corte.

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 toda 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 . Considere uma tal e e suponha que e não está em T .

Então $T + e$ contém um circuito, que tem pelo menos uma outra aresta f de T que cruza o corte.

Portanto $T' := T - f + e$ 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 **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 **Prim**, que mantém uma árvore T que contém um vértice s , acrescentando em cada iteração uma aresta segura a T .

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

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

O algoritmo de Kruskal vai aumentando uma **floresta**.

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

Lembre-se que $n := |V(G)|$ e $m := |E(G)|$.

Algoritmo de Prim

v .key = peso da aresta mais leve ligando v à árvore corrente

Algoritmo de Prim

$v.key$ = peso da aresta mais leve ligando v à árvore corrente

PRIM (G, w)

- 1 seja s um vértice arbitrário de G
- 2 **para** $v \in V(G) \setminus \{s\}$ **faça** $v.key \leftarrow \infty$
- 3 $s.key \leftarrow 0$ $s.\pi \leftarrow \text{nil}$
- 4 $Q \leftarrow V(G)$
- 5 **enquanto** $Q \neq \emptyset$ **faça**
- 6 $u \leftarrow \text{EXTRACT-MIN}(Q)$
- 7 **para cada** $v \in \text{adj}(u)$ **faça**
- 8 **se** $v \in Q$ e $w(uv) < v.key$
- 9 **então** $v.\pi \leftarrow u$ $v.key \leftarrow w(uv)$
- 10 π descreve a MST

Qual é a diferença para o algoritmo de Dijkstra?

Algoritmo de Prim

v .key = peso da aresta mais leve ligando v à árvore corrente

PRIM (G, w)

- 1 seja s um vértice arbitrário de G
- 2 **para** $v \in V(G) \setminus \{s\}$ **faça** v .key $\leftarrow \infty$
- 3 s .key $\leftarrow 0$ s . π \leftarrow nil
- 4 $Q \leftarrow V(G)$
- 5 **enquanto** $Q \neq \emptyset$ **faça**
- 6 $u \leftarrow$ EXTRACT-MIN(Q)
- 7 **para cada** $v \in \text{adj}(u)$ **faça**
- 8 **se** $v \in Q$ e $w(uv) < v$.key
- 9 **então** v . π $\leftarrow u$ v .key $\leftarrow w(uv)$
- 10 π descreve a MST

Algoritmo de Prim

v .key = peso da aresta mais leve ligando v à árvore corrente

PRIM (G, w)

- 1 seja s um vértice arbitrário de G
- 2 **para** $v \in V(G) \setminus \{s\}$ **faça** v .key $\leftarrow \infty$
- 3 s .key $\leftarrow 0$ s . π \leftarrow nil
- 4 $Q \leftarrow V(G)$
- 5 **enquanto** $Q \neq \emptyset$ **faça**
- 6 $u \leftarrow$ EXTRACT-MIN(Q)
- 7 **para cada** $v \in \text{adj}(u)$ **faça**
- 8 **se** $v \in Q$ e $w(uv) < v$.key
- 9 **então** v . π $\leftarrow u$ v .key $\leftarrow w(uv)$
- 10 π descreve a MST

Consumo de tempo das operações na fila de prioridade:

Linha 4: $\Theta(n)$

EXTRACT-MIN e DECREASE-KEY (linha 9): $O(\lg n)$

Algoritmo de Prim

v .key = peso da aresta mais leve ligando v à árvore corrente

PRIM (G, w)

- 1 seja s um vértice arbitrário de G
- 2 **para** $v \in V(G) \setminus \{s\}$ **faça** v .key $\leftarrow \infty$
- 3 s .key $\leftarrow 0$ s . π \leftarrow nil
- 4 $Q \leftarrow V(G)$
- 5 **enquanto** $Q \neq \emptyset$ **faça**
- 6 $u \leftarrow$ EXTRACT-MIN(Q)
- 7 **para cada** $v \in \text{adj}(u)$ **faça**
- 8 **se** $v \in Q$ e $w(uv) < v$.key
- 9 **então** v . π $\leftarrow u$ v .key $\leftarrow w(uv)$
- 10 π descreve a MST

Consumo de tempo do Prim: $O(m \lg n)$

Consumo de tempo com Fibonacci heap: $O(m + n \lg n)$

Algoritmo de Kruskal - a idéia

Algoritmo de Kruskal - a idéia

1. Mantem um conjunto A de arestas tal que o subgrafo gerador $G[A]$ é uma floresta.

Algoritmo de Kruskal - a idéia

1. Mantem um conjunto A de arestas tal que o subgrafo gerador $G[A]$ é uma floresta.
2. Processa as arestas em ordem crescente de peso.

Algoritmo de Kruskal - a idéia

1. Mantem um conjunto A de arestas tal que o subgrafo gerador $G[A]$ é uma floresta.
2. Processa as arestas em ordem crescente de peso.
3. Sempre que a aresta for segura para A , incorpore a A .

Algoritmo de Kruskal - a idéia

1. Mantem um conjunto A de arestas tal que o subgrafo gerador $G[A]$ é uma floresta.
2. Processa as arestas em ordem crescente de peso.
3. Sempre que a aresta for segura para A , incorpore a A .

PROBLEMA: Como verificar se uma aresta é segura?

Algoritmo de Kruskal - a idéia

1. Mantem um conjunto A de arestas tal que o subgrafo gerador $G[A]$ é uma floresta.
2. Processa as arestas em ordem crescente de peso.
3. Sempre que a aresta for segura para A , incorpore a A .

PROBLEMA: Como verificar se uma aresta é segura?

Basta verificar se suas pontas estão em componentes distintas de $G[A]$.

Algoritmo de Kruskal - a idéia

1. Mantem um conjunto A de arestas tal que o subgrafo gerador $G[A]$ é uma floresta.
2. Processa as arestas em ordem crescente de peso.
3. Sempre que a aresta for segura para A , incorpore a A .

PROBLEMA: Como verificar se uma aresta é segura?

Basta verificar se suas pontas estão em componentes distintas de $G[A]$.

Como fazer isso eficientemente?

A estrutura de dados

É preciso manter

A estrutura de dados

É preciso manter uma **partição** de V ,

A estrutura de dados

É preciso manter uma **partição** de V ,
e dar suporte a duas operações:

A estrutura de dados

É preciso manter uma **partição** de V ,
e dar suporte a duas operações:

1. Dados dois vértices, decidir se estão no mesmo bloco.

A estrutura de dados

É preciso manter uma **partição** de V ,
e dar suporte a duas operações:

1. Dados dois vértices, decidir se estão no mesmo bloco.
2. Dados dois blocos, substituí-los por sua união.

A estrutura de dados

É preciso manter uma **partição** de V ,
e dar suporte a duas operações:

1. Dados dois vértices, decidir se estão no mesmo bloco.
2. Dados dois blocos, substituí-los por sua união.

Hmmm, é preciso um jeito de se referir aos blocos...

A estrutura de dados

É preciso manter uma **partição** de V ,
e dar suporte a duas operações:

1. Dados dois vértices, decidir se estão no mesmo bloco.
2. Dados dois blocos, substituí-los por sua união.

Hmmm, é preciso um jeito de se referir aos blocos... um **nome!**

A estrutura de dados

É preciso manter uma **partição** de V ,
e dar suporte a duas operações:

1. Dados dois vértices, decidir se estão no mesmo bloco.
2. Dados dois blocos, substituí-los por sua união.

Hmmm, é preciso um jeito de se referir aos blocos... um **nome!**

Melhor:

A estrutura de dados

É preciso manter uma **partição** de V ,
e dar suporte a duas operações:

1. Dados dois vértices, decidir se estão no mesmo bloco.
2. Dados dois blocos, substituí-los por sua união.

Hmmm, é preciso um jeito de se referir aos blocos... um **nome!**

Melhor:

1. Dado um vértice,
devolver o nome do seu bloco.

A estrutura de dados

É preciso manter uma **partição** de V ,
e dar suporte a duas operações:

1. Dados dois vértices, decidir se estão no mesmo bloco.
2. Dados dois blocos, substituí-los por sua união.

Hmmm, é preciso um jeito de se referir aos blocos... um **nome!**

Melhor:

1. Dado um vértice,
devolver o nome do seu bloco.
2. Dados os nomes de dois blocos,
substituí-los por sua união.

A estrutura de dados

É preciso manter uma **partição** de V ,
e dar suporte a duas operações:

1. Dados dois vértices, decidir se estão no mesmo bloco.
2. Dados dois blocos, substituí-los por sua união.

Hmmm, é preciso um jeito de se referir aos blocos... um **nome!**

Melhor:

1. Dado um vértice,
devolver o nome do seu bloco. **FINDSET**
2. Dados os nomes de dois blocos,
substituí-los por sua união. **UNION**

O problema **UNION-FIND**

O algoritmo de Kruskal

KRUSKAL (G, w)

- 1 $A \leftarrow \emptyset$
- 2 **para cada** $v \in V(G)$ **faça** MAKESET(u)
- 3 **para** $e \in E$ em ordem crescente de $w(e)$ **faça**
- 4 sejam u e v as pontas de e
- 5 **se** FINDSET(u) \neq FINDSET(v)
- 6 **então** $A \leftarrow A \cup \{e\}$
- 7 UNION(FINDSET(u), FINDSET(v))
- 8 **devolva** A

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 bloco da partição que contém x .

UNION (x, y): substitui os blocos 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 bloco da partição que contém x .

UNION (x, y): substitui os blocos da partição que contêm x e y pela união deles.

IDÉIA:

Usar um elemento do conjunto (o **representante**)
como **identificador**.

Como analisar?

Uso típico:

$$n - 1 \text{ UNION, } m \text{ FINDSET, } m \gg n$$

em sequência arbitrária.

Como analisar?

Uso típico:

$$n - 1 \text{ UNION, } m \text{ FINDSET, } m \gg n$$

em sequência arbitrária.

O custo de uma operação pode variar muito

- é **muito** pessimista supor o pior caso em cada uma.

Como analisar?

Uso típico:

$$n - 1 \text{ UNION, } m \text{ FINDSET, } m \gg n$$

em sequência arbitrária.

O custo de uma operação pode variar muito

- é **muito** pessimista supor o pior caso em cada uma.

Vale a pena considerar uma sequência completa de operações, em vez de analisar uma a uma.

Como analisar?

Uso típico:

$$n - 1 \text{ UNION, } m \text{ FINDSET, } m \gg n$$

em sequência arbitrária.

O custo de uma operação pode variar muito
- é **muito** pessimista supor o pior caso em cada uma.

Vale a pena considerar uma sequência completa
de operações, em vez de analisar uma a uma.

Isso se chama **Análise Amortizada**.

Como analisar?

Uso típico:

$$n - 1 \text{ UNION, } m \text{ FINDSET, } m \gg n$$

em sequência arbitrária.

O custo de uma operação pode variar muito

- é **muito** pessimista supor o pior caso em cada uma.

Vale a pena considerar uma sequência completa de operações, em vez de analisar uma a uma.

Isso se chama **Análise Amortizada**.

É um método especialmente útil para analisar estruturas adaptativas, em que uma operação pode preparar caminho para a execução de operações futuras.

Não é tão novidade assim

Numa busca em grafos:

- 1 para cada $u \in G.V$
- 2 para cada $v \in u.Estrela$ faça

Não é tão novidade assim

Numa busca em grafos:

- 1 para cada $u \in G.V$
- 2 para cada $v \in u.$ Estrela faça

Análise ingênua:

*Como cada estrela tem tamanho $O(n)$,
os dois laços combinados levam $O(n^2)$.*

Não é tão novidade assim

Numa busca em grafos:

- 1 para cada $u \in G.V$
- 2 para cada $v \in u.Estrela$ faça

Análise ingênua:

*Como cada estrela tem tamanho $O(n)$,
os dois laços combinados levam $O(n^2)$.*

Análise amortizada:

*Como na linha 2 cada arco é examinado uma única vez,
os dois laços combinados levam $O(m)$.*

Implementação 1 do union-find

- ▶ Cada conjunto é uma lista ligada, com cabeçalho apontando início e fim - facilita UNION.

Implementação 1 do union-find

- ▶ Cada conjunto é uma lista ligada, com cabeçalho apontando início e fim - facilita **UNION**.
- ▶ Cada elemento da lista tem um apontador **nome** para o cabeçalho - facilita **FINDSET**.

Implementação 1 do union-find

- ▶ Cada conjunto é uma lista ligada, com cabeçalho apontando início e fim - facilita **UNION**.
- ▶ Cada elemento da lista tem um apontador **nome** para o cabeçalho - facilita **FINDSET**.

MAKESET (x) cria uma lista contendo só x .

Implementação 1 do union-find

- ▶ Cada conjunto é uma lista ligada, com cabeçalho apontando início e fim - facilita **UNION**.
- ▶ Cada elemento da lista tem um apontador **nome** para o cabeçalho - facilita **FINDSET**.

MAKESET (x) cria uma lista contendo só x .

FINDSET respondido direto pelo **nome**.

Implementação 1 do union-find

- ▶ Cada conjunto é uma lista ligada, com cabeçalho apontando início e fim - facilita **UNION**.
- ▶ Cada elemento da lista tem um apontador **nome** para o cabeçalho - facilita **FINDSET**.

MAKESET (x) cria uma lista contendo só x .

FINDSET respondido direto pelo **nome**.

UNION: juntar as duas listas é fácil, mas é preciso atualizar o **nome** dos membros de um dos conjuntos.

Implementação 1 do union-find

- ▶ Cada conjunto é uma lista ligada, com cabeçalho apontando início e fim - facilita **UNION**.
- ▶ Cada elemento da lista tem um apontador **nome** para o cabeçalho - facilita **FINDSET**.

MAKESET (x) cria uma lista contendo só x .

FINDSET respondido direto pelo **nome**.

UNION: juntar as duas listas é fácil, mas é preciso atualizar o **nome** dos membros de um dos conjuntos.

MAKESET e **FINDSET** levam $O(1)$, mas vários **UNION** podem gastar $\Omega(n^2)$ mudanças de **nome**.

Melhoria

Melhoria

Na **UNION**, pendure a lista menor na maior
(cabeçalho precisa guardar o tamanho da lista).

Melhoria

Na **UNION**, pendure a lista menor na maior
(cabeçalho precisa guardar o tamanho da lista).

Agora, cada vez que um elemento muda de **nome**,
o conjunto em que ele está pelo menos dobra de tamanho.

Assim, seu **nome** muda no máximo

Melhoria

Na **UNION**, pendure a lista menor na maior
(cabeçalho precisa guardar o tamanho da lista).

Agora, cada vez que um elemento muda de **nome**,
o conjunto em que ele está pelo menos dobra de tamanho.

Assim, seu **nome** muda no máximo $\lg n$ vezes.

Melhoria

Na **UNION**, pendure a lista menor na maior
(cabeçalho precisa guardar o tamanho da lista).

Agora, cada vez que um elemento muda de **nome**,
o conjunto em que ele está pelo menos dobra de tamanho.

Assim, seu **nome** muda no máximo $\lg n$ vezes.

Tempo total de n **UNION**: $O(n \lg n)$.

Melhoria

Na **UNION**, pendure a lista menor na maior
(cabeçalho precisa guardar o tamanho da lista).

Agora, cada vez que um elemento muda de **nome**,
o conjunto em que ele está pelo menos dobra de tamanho.

Assim, seu **nome** muda no máximo $\lg n$ vezes.

Tempo total de n **UNION**: $O(n \lg n)$.

Tempo total: $O(m + n \lg n)$

Implementação 2 do union-find

Cada conjunto é uma árvore apontando para a raiz.

MAKESET (x)

1 $x.pai \leftarrow x$

Implementação 2 do union-find

Cada conjunto é uma árvore apontando para a raiz.

MAKESET (x)

1 $x.pai \leftarrow x$

FINDSET (x)

1 **se** $x = x.pai$

2 **devolva** x

3 **devolva** FINDSET ($x.pai$)

Implementação 2 do union-find

Cada conjunto é uma árvore apontando para a raiz.

MAKESET (x)

1 $x.pai \leftarrow x$

FINDSET (x)

1 se $x = x.pai$

2 devolva x

3 devolva FINDSET ($x.pai$)

FINDSET (x)

1 enquanto $x.pai \neq x$ faça

2 $x \leftarrow x.pai$

3 devolva x

Implementação 2 do union-find

Cada conjunto é uma árvore apontando para a raiz.

MAKESET (x)

1 $x.pai \leftarrow x$

FINDSET (x)

1 se $x = x.pai$

2 devolva x

3 devolva FINDSET ($x.pai$)

FINDSET (x)

1 enquanto $x.pai \neq x$ faça

2 $x \leftarrow x.pai$

3 devolva x

UNION (x, y)

▷ x e y representantes distintos

1 $y.pai \leftarrow x$

Consumo de tempo: do FINDSET pode ser muito ruim... $\Theta(n)$.

Implementação 2 do union-find

Cada conjunto é uma árvore apontando para a raiz.

MAKESET (x)

1 $x.pai \leftarrow x$

FINDSET (x)

1 se $x = x.pai$

2 devolva x

3 devolva FINDSET ($x.pai$)

FINDSET (x)

1 enquanto $x.pai \neq x$ faça

2 $x \leftarrow x.pai$

3 devolva x

UNION (x, y)

▷ x e y representantes distintos

1 $y.pai \leftarrow x$

Consumo de tempo: do FINDSET pode ser muito ruim... $\Theta(n)$.

Temos que fazer melhor...

Melhoria 1: Heurística das alturas

MAKESET (x)

1 $x.pai \leftarrow x$

2 $x.rank \leftarrow 0$

Melhoria 1: Heurística das alturas

MAKESET (x)

1 $x.pai \leftarrow x$

2 $x.rank \leftarrow 0$

FINDSET (x): o mesmo de antes

Melhoria 1: Heurística das alturas

MAKESET (x)

- 1 $x.pai \leftarrow x$
- 2 $x.rank \leftarrow 0$

FINDSET (x): o mesmo de antes

UNION (x, y) $\triangleright x$ e y representantes distintos

- 1 se $x.rank \geq y.rank$
- 2 então $y.pai \leftarrow x$
- 3 se $x.rank = y.rank$
- 4 então $x.rank \leftarrow x.rank + 1$
- 5 senão $x.pai \leftarrow y$

Melhoria 1: Heurística das alturas

MAKESET (x)

1 $x.pai \leftarrow x$

2 $x.rank \leftarrow 0$

FINDSET (x): o mesmo de antes

UNION (x, y) $\triangleright x$ e y representantes distintos

1 se $x.rank \geq y.rank$

2 então $y.pai \leftarrow x$

3 se $x.rank = y.rank$

4 então $x.rank \leftarrow x.rank + 1$

5 senão $x.pai \leftarrow y$

INVARIANTE: $x.rank \geq$ altura da árvore pendurada em x .

Melhoria 1: Heurística das alturas

MAKESET (x)

1 $x.pai \leftarrow x$

2 $x.rank \leftarrow 0$

FINDSET (x): o mesmo de antes

UNION (x, y) $\triangleright x$ e y representantes distintos

1 se $x.rank \geq y.rank$

2 então $y.pai \leftarrow x$

3 se $x.rank = y.rank$

4 então $x.rank \leftarrow x.rank + 1$

5 senão $x.pai \leftarrow y$

INVARIANTE: $x.rank \geq$ altura da árvore pendurada em x .

Melhorou: FINDSET é $\Theta(\lg n)$. Total: $O(n + m \lg n)$

Melhoria 1: Heurística das alturas

MAKESET (x)

1 $x.pai \leftarrow x$

2 $x.rank \leftarrow 0$

FINDSET (x): o mesmo de antes

UNION (x, y) $\triangleright x$ e y representantes distintos

1 se $x.rank \geq y.rank$

2 então $y.pai \leftarrow x$

3 se $x.rank = y.rank$

4 então $x.rank \leftarrow x.rank + 1$

5 senão $x.pai \leftarrow y$

INVARIANTE: $x.rank \geq$ altura da árvore pendurada em x .

Melhorou: FINDSET é $\Theta(\lg n)$. Total: $O(n + m \lg n)$

Um pouco melhor que antes, mas dá para fazer melhor ainda!

Implementação 3

Heurística da compressão dos caminhos

FINDSET (x)

1 if $x.pai \neq x$

2 então $x.pai \leftarrow$ FINDSET ($x.pai$)

3 devolva $x.pai$

Implementação 3

Heurística da compressão dos caminhos

```
FINDSET ( $x$ )  
1  if  $x.pai \neq x$   
2      então  $x.pai \leftarrow$  FINDSET ( $x.pai$ )  
3  devolva  $x.pai$ 
```

Consumo *amortizado* de tempo de cada operação:

$$O(\lg^* n),$$

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

```
FINDSET ( $x$ )  
1  if  $x.pai \neq x$   
2      então  $x.pai \leftarrow \text{FINDSET}(x.pai)$   
3  devolva  $x.pai$ 
```

Consumo *amortizado* de tempo de cada operação:

$$O(\lg^* n),$$

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.

A análise desta ED é vista na disciplina MAC6711.

O que isso significa

Duas maneiras de entender o \lg^* :

$$\lg^{(0)}(n) = n$$

$$\lg^{(k)}(n) = \lg(\lg^{(k-1)}(n)), \quad k > 0$$

$$\lg^*(n) = \min\{k \mid \lg^{(k)}(n) \leq 1\}$$

O que isso significa

Duas maneiras de entender o \lg^* :

$$\lg^{(0)}(n) = n$$

$$\lg^{(k)}(n) = \lg(\lg^{(k-1)}(n)), \quad k > 0$$

$$\lg^*(n) = \min\{k \mid \lg^{(k)}(n) \leq 1\}$$

$$b_0 = 1$$

$$b_k = 2^{b_{k-1}}, \quad k > 0$$

$$\lg^*(n) = \min\{k \mid n < b_k\}.$$

O que isso significa

O que isso significa

$$b_n = 2^{\underbrace{2^{2^{\dots^2}}}_{n \text{ times}}}$$

O que isso significa

$$b_n = 2^{\underbrace{2^{2^{\dots^2}}}_{n \text{ times}}}$$

k	b_k
0	1
1	2
2	4
3	16
4	65536
5	$> 10^{19000}$

O que isso significa

$$b_n = \underbrace{2^{2^{2^{\dots}}}}_{n \text{ times}}^2$$

k	b_k
0	1
1	2
2	4
3	16
4	65536
5	$> 10^{19000}$

No mundo real, $\lg^* n \leq 5$.