Projeto e Análise de Algoritmos
Aula 8:
Algoritmos Gulosos (5)

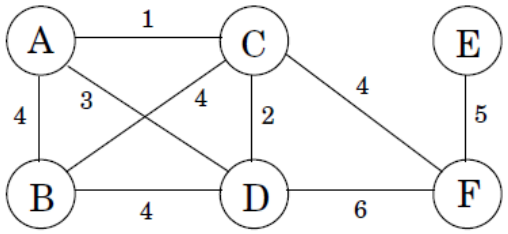
DECOM/UFOP
2012/2 – 5º. Período
Anderson Almeida Ferreira
Adaptado do material de Andréa Iabrudi Tavares

Algoritmos Gulosos

- Algoritmos míopes
  - Escolha “óbvia” a cada passo
  - Escolhe e depois resolve subproblema recursivamente.
  - Solução construtiva.
- Quando é ótimo
  - Melhor decisão local é melhor decisão global.
  - Mostrar que a solução é ótima normalmente ajuda a formular o problema.
  - Sub-estrutura ótima (princípio da otimalidade)

Árvores Geradoras Mínimas (AGM)

- Conectar computadores por rede, cada link com um custo de manutenção

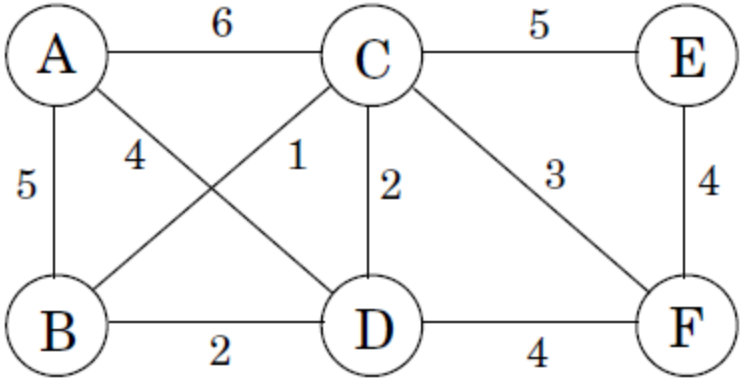


Propriedades

Formulação

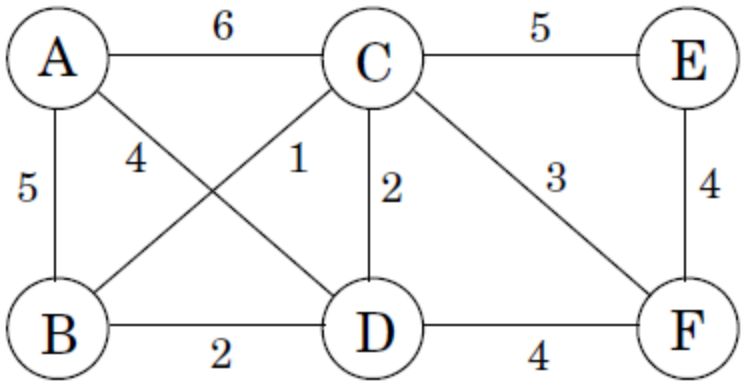
Guloso AGM: Algoritmo de Kruskal

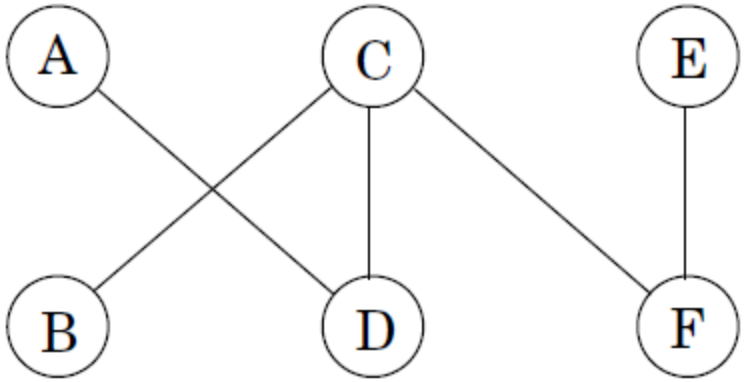
- Adicione sempre a aresta de menor peso que não forma ciclo (1956).



Guloso AGM: Algoritmo de Kruskal

- Adicione sempre a aresta de menor peso que não forma ciclo.

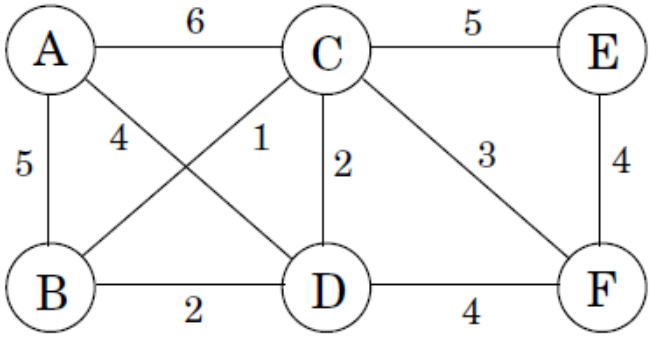




Está correto, ou seja, é ótimo?????

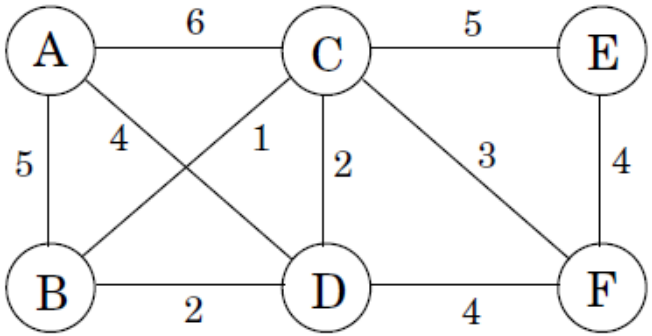
Kruskal está correto: cortes

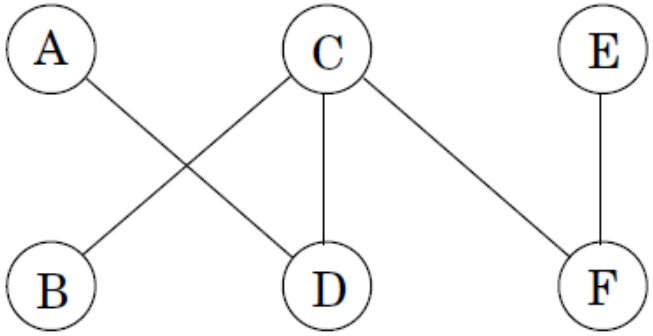
- Partição (S,V-S) ou
  de forma equivalente conjunto de vertices.



Kruskal está correto

Sejam X arestas de alguma AGM. Seja qualquer subconjunto S de vértices tal que X não cruze S e V-S e seja e a aresta mais leve do corte. Então X U {e} faz parte de alguma AGM.

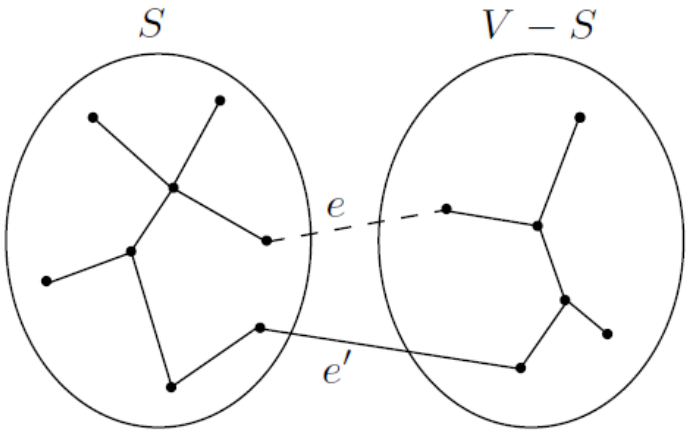


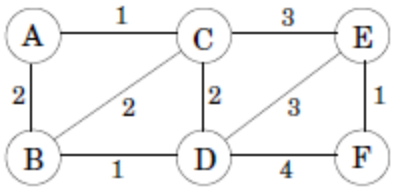


Kruskal está correto

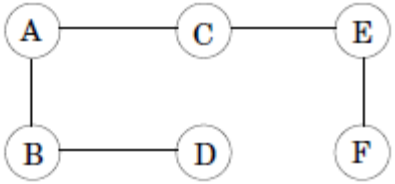
- Prova

  Sejam X arestas de alguma AGM. Seja qualquer subconjunto S de vértices tal que X não cruze S e V-S e seja e a aresta mais leve do corte. Então X U {e} faz parte de alguma AGM.

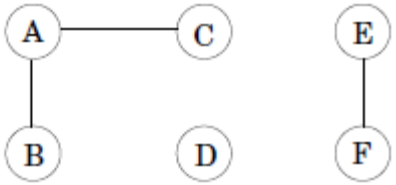




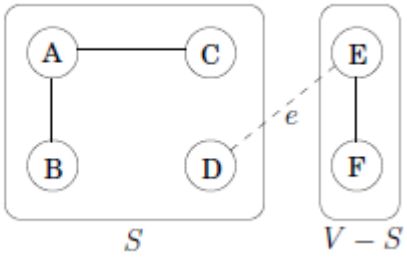
MST T:



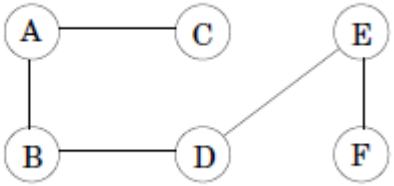
Edges X:



The cut:



MST T':



Guloso AGM: Algoritmo de Kruskal

```
procedure kruskal (G, w)
Input:    A connected undirected graph G = (V, E) with edge weights w_e
Output:   A minimum spanning tree defined by the edges X

X = {}
Sort the edges E by weight
for all edges {u, v} ∈ E, in increasing order of weight:
   if {u, v} ∪ X sem ciclo
      add edge {u, v} to X
```

Complexidade depende de qual operação?

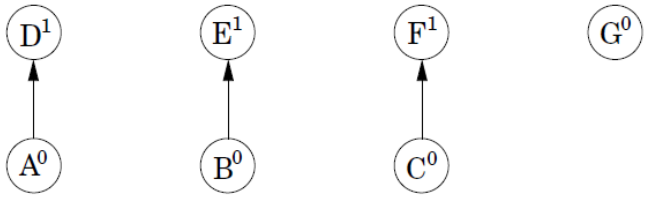
Guloso AGM: Algoritmo de Kruskal

```
procedure kruskal(G, w)
Input:    A connected undirected graph G = (V, E) with edge weights w_e
Output:   A minimum spanning tree defined by the edges X

for all u ∈ V:
   makeset(u)

X = {}
Sort the edges E by weight
for all edges {u, v} ∈ E, in increasing order of weight:
   if find(u) ≠ find(v):
      add edge {u, v} to X
      union(u, v)
```

Conjuntos Disjuntos
Mesmo componente em log(n)

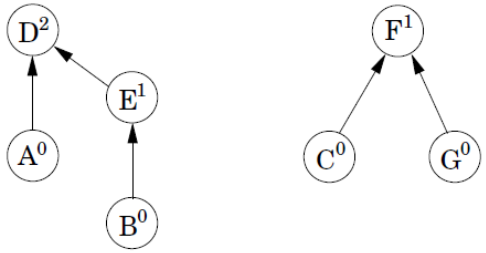
makeset(A), makeset(B), ..., makeset(G):

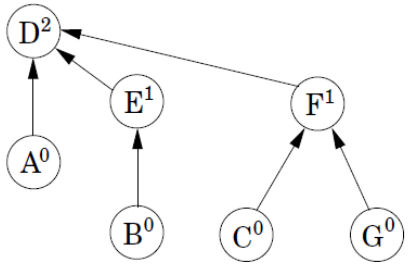
union(A, D), union(B, E), union(C, F):



union(B, G):

union(C, G), union(E, A):





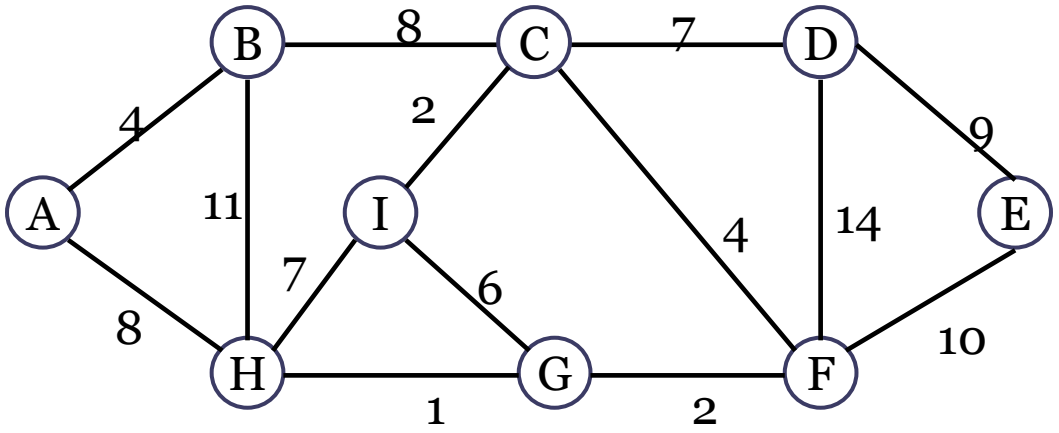
Algoritmo para componentes disjuntos

Propriedade 1: $\text{rank}(x) < \text{rank}(\pi(x))$

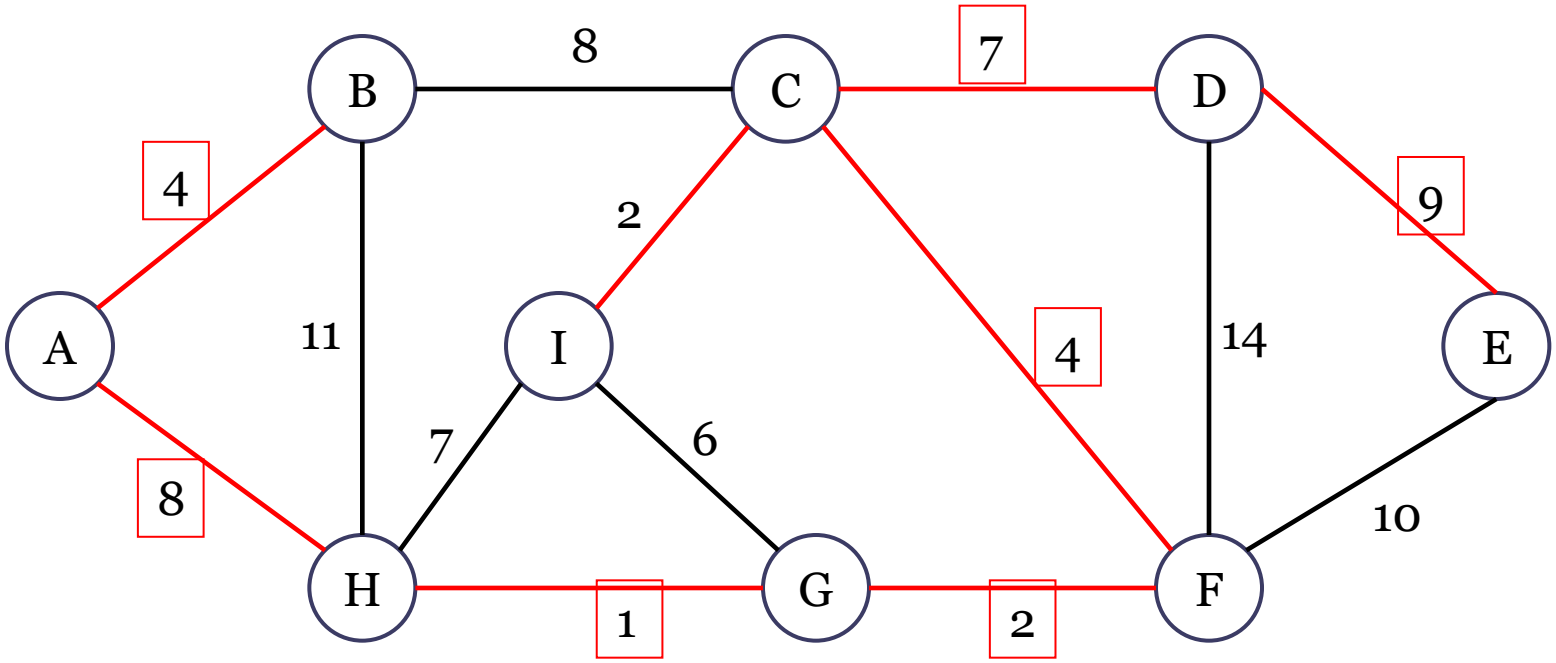
Propriedade 2: rank da raiz é *k*, pelo menos 2^k

```
procedure makeset(x)
π(x) = x
rank(x) = 0
```

```
procedure union(x, y)
r_x = find(x)
r_y = find(y)
if r_x = r_y:   return
if rank(r_x) > rank(r_y):
    π(r_y) = r_x
else:
    π(r_x) = r_y
    if rank(r_x) = rank(r_y):   rank(r_y) = rank(r_y) + 1
```




```
procedure union(x, y)
r_x = find(x)
r_y = find(y)
if r_x = r_y:   return
if rank(r_x) > rank(r_y):
    π(r_y) = r_x
else:
    π(r_x) = r_y
    if rank(r_x) = rank(r_y) :   rank(r_y) = rank(r_y) + 1
```


A
B
C
D
E
F
G
H
I
4
8
11
8
2
7
4
6
7
14
9
10
1
2

Guloso AGM: Algoritmo de Kruskal

```
procedure kruskal(G, w)
Input:    A connected undirected graph G = (V, E) with edge weights w_e
Output:   A minimum spanning tree defined by the edges X

for all u ∈ V:
   makeset(u)

X = {}
Sort the edges E by weight
for all edges {u, v} ∈ E, in increasing order of weight:
   if find(u) ≠ find(v):
      add edge {u, v} to X
      union(u, v)
```

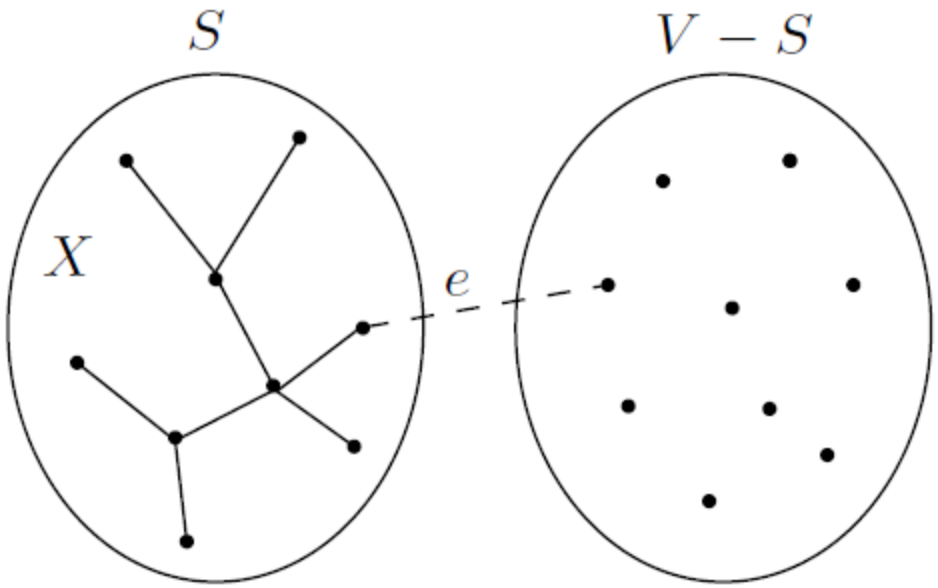
Complexidade?

$$T(n) = \Theta(m \log n)$$

$$\Theta(n \log n) \leq T(n) \leq \Theta(n^2 \log n)$$

Guloso AGM: Algoritmo de Prim

- Cresça a árvore, colocando sempre a menor aresta que liga X aos vértices que faltam (1930 Jarnik, 1957 Prim, 1959 Dijkstra)



decom
departamento de computação

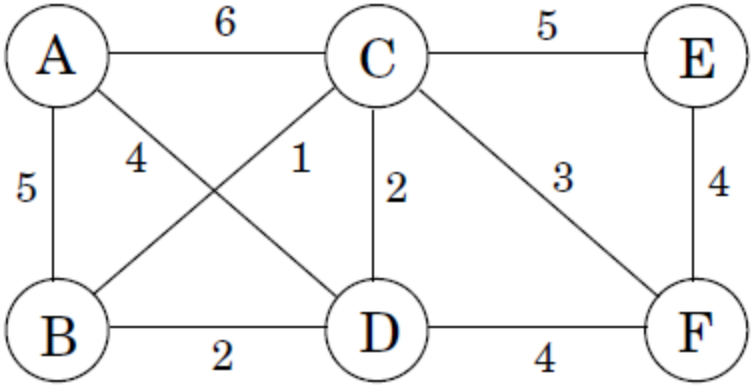
Guloso AGM: Algoritmo de Prim

```
procedure prim(G, w)
Input:    A connected undirected graph G = (V, E) with edge weights w_e
Output:   A minimum spanning tree defined by the array prev

for all u ∈ V:
    cost(u) = ∞
    prev(u) = nil
Pick any initial node u_0
cost(u_0) = 0

H = makequeue(V)      (priority queue, using cost-values as keys)
while H is not empty:
    v = deletemin(H)
    for each {v, z} ∈ E:
        if cost(z) > w(v, z):
            cost(z) = w(v, z)
            prev(z) = v
            decreasekey(H, z)
```

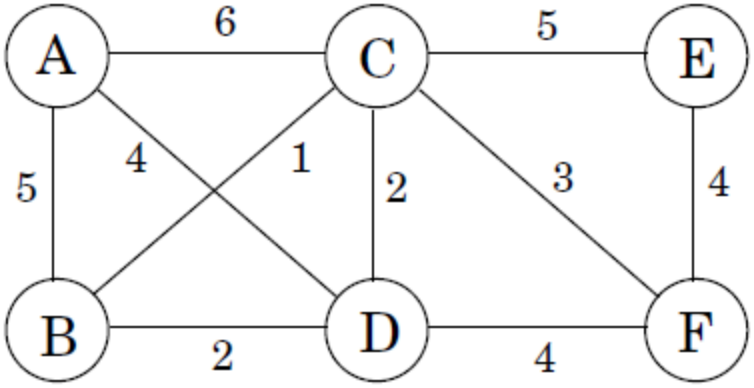
decom
departamento de computação

Guloso Menor caminho: Dijkstra

```
procedure dijkstra(G, l, s)
Input:    Graph G = (V, E), directed or undirected;
          positive edge lengths {l_e : e ∈ E}; vertex s ∈ V
Output:   For all vertices u reachable from s, dist(u) is set
          to the distance from s to u.

for all u ∈ V:
    dist(u) = ∞
    prev(u) = nil
dist(s) = 0

H = makequeue(V)   (using dist-values as keys)
while H is not empty:
    u = deletemin(H)
    for all edges (u, v) ∈ E:
        if dist(v) > dist(u) + l(u, v):
            dist(v) = dist(u) + l(u, v)
            prev(v) = u
            decreasekey(H, v)
```



Set S	A	B	C	D	E	F
$\{\}$	0/nil	∞/nil	∞/nil	∞/nil	∞/nil	∞/nil
A		$5/A$	$6/A$	$4/A$	∞/nil	∞/nil
A, D		$2/D$	$2/D$		∞/nil	$4/D$
A, D, B			$1/B$		∞/nil	$4/D$
A, D, B, C					$5/C$	$3/C$
A, D, B, C, F					$4/F$	



Atenção: apesar de ser linear nesse exemplo, representa qualquer árvore.

prev: permite recuperar árvore

Set S	A	B	C	D	E	F
$\{\}$	$0/\text{nil}$	∞/nil	∞/nil	∞/nil	∞/nil	∞/nil
A		$5/A$	$6/A$	$4/A$	∞/nil	∞/nil
A, D		$2/D$	$2/D$		∞/nil	$4/D$
A, D, B			$1/B$		∞/nil	$4/D$
A, D, B, C					$5/C$	$3/C$
A, D, B, C, F					$4/F$	

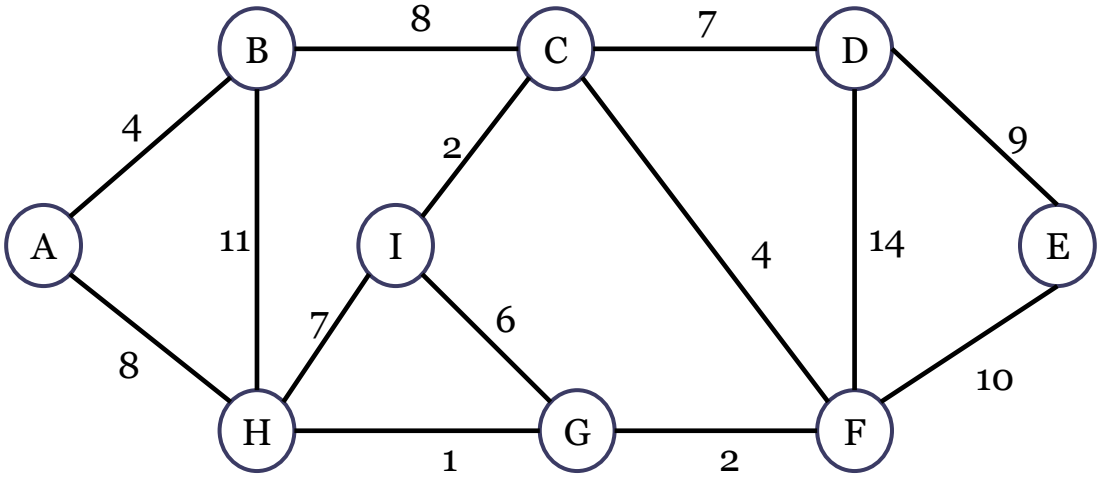
Outro exemplo:

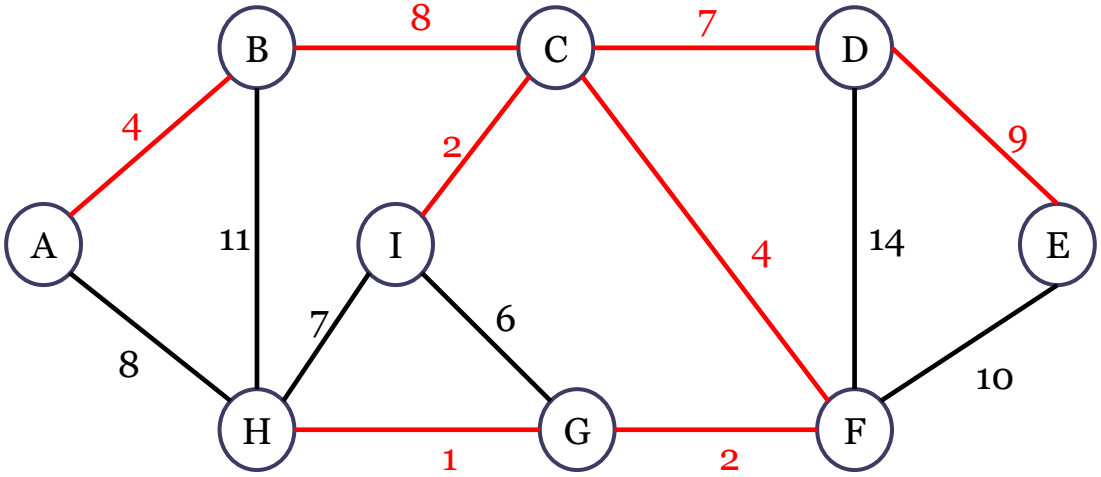
- http://en.wikipedia.org/wiki/Prim%27s_algorithm
- http://en.wikipedia.org/wiki/Kruskal%27s_algorithm

decom
departamento de computação

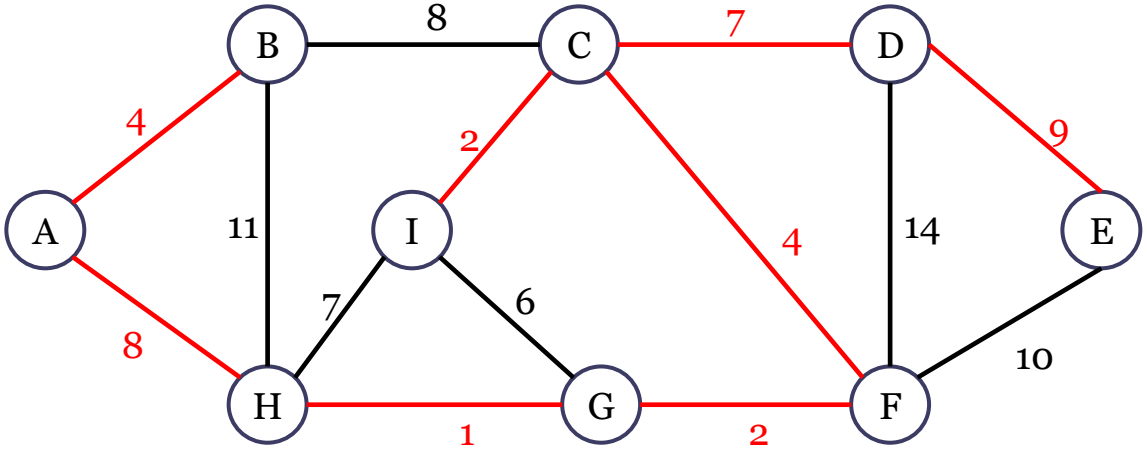
Fila Prioridades x Complexidade

Implementation	`deletemin`	`insert`/ `decreasekey`	$\|V\| \times$ `deletemin` + $(\|V\| + \|E\|) \times$ `insert`
Array	$O(\|V\|)$	$O(1)$	$O(\|V\|^2)$
Binary heap	$O(\log \|V\|)$	$O(\log \|V\|)$	$O((\|V\| + \|E\|) \log \|V\|)$
d-ary heap	$O(\frac{d \log \|V\|}{\log d})$	$O(\frac{\log \|V\|}{\log d})$	$O((\|V\| \cdot d + \|E\|) \frac{\log \|V\|}{\log d})$
Fibonacci heap	$O(\log \|V\|)$	$O(1)$ (amortized)	$O(\|V\| \log \|V\| + \|E\|)$


A
B
C
D
E
F
G
H
I
4
8
11
8
7
2
6
4
7
14
9
10
1
2



Árvores podem ser diferentes, mas têm o mesmo peso!



Um problema de seleção de atividades

- Seja $S=\{a_1, a_2, \ldots, a_n\}$ um conjunto de atividades propostas que desejam usar um recurso, que só pode ser usado por uma atividade de cada vez.
- Cada atividade a_i tem um tempo de inicio si e um tempo de termino f_i, onde $0 \leq s_i < f_i < \infty$.
- As atividades a_i e a_j são compatíveis se $s_i \geq f_j$ ou $s_j \geq f_i$.
- Problema: Selecionar um subconjunto de tamanho máximo de atividades mutuamente compatíveis.

Exemplo

i	1	2	3	4	5	6	7	8	9	10	11
s_i	1	3	0	5	3	5	6	8	8	2	12
f_i	4	5	6	7	8	9	10	11	12	13	14

Subestrutura ótima

- Seja $S_{ij}=\{a_k \in S \mid f_i \leq s_k < f_k \leq s_j\}$, ou seja, o subconjunto de atividades que podem começar após a_i terminar e que terminam após a atividade a_j começar.
- Acrescentamos duas atividades fictícias a_o e a_{n+1}, onde $f_o=0$ e $s_{n+1}=\infty$.
- Logo, $S = S_{o\ n+1}$
- Se uma solução para s_{ij} inclui a_k, então a_k gera dois subproblemas s_{ik} e $s_{kj} \subset s_{ij}$.
- Logo, se há uma solução ótima para s_{ij} que inclui a_k, as soluções para s_{ik} e s_{kj} usadas dentro da solução ótima de s_{ij} também devem ser ótimas.

Teorema

- Uma solução ótima para S_{ij} seria
  $A_{ij} = A_{ik} \cup \{a_k\} \cup A_{kj}$
- $A_{0\ n+1}$ seria a solução para o problema inteiro.
- Teorema: Considere qualquer subproblema não vazio S_{ij} e seja a_m a atividade em S_{ij} com término mais antigo. Então,
  1. A atividade a_m é usada em algum subconjunto de tamanho máximo de atividades mutuamente compatíveis de S_{ij}.
  2. O subproblema S_{im} é vazio, de forma e a escolha de a_m deixa o subproblema S_{mj} como único que pode ser não vazio.

Teorema

2. Suponha que exista algum $a_k \in S_{im}$. Então $f_i \leq s_k < f_k \leq s_m < f_m \Rightarrow f_k < f_m$.
   Então $a_k \in S_{ij}$ e ele tem $f_k < f_m$. Contradição.
1. Seja A_{ij} um subconjunto de tamanho máximo de atividades mutuamente compatíveis em S_{ij}.
   Ordene as atividade em A_{ij} em ordem crescente de tempo de término.
   Seja a_k a primeira atividade em A_{ij}.
   Se $a_k = a_m$, pronto.
   Caso contrário, construa $A'_{ij} = A_{ij} - \{a_k\} \cup \{a_m\}$.

Algoritmo

- Recursive-Active-Selector(s,f,i,j)

  $m \leftarrow i + 1$

  while $m<j$ and $s_m < f_i$

  $m \leftarrow m + 1$

  if $m < j$

  Return $\{am\} \cup$ Recursive-Active-Selector(s,f,m,j)

  return $\emptyset$

- Iteractive-Active-Selector(s,f)
  - n = comprimento(s)
  - A = {1}
  - i = 1
  - for m=2..n:
    - If $s_m \geq f_i$
      - $A = A \cup \{a_m\}$
      - i = m

Exemplo

i	1	2	3	4	5	6	7	8	9	10	11
s_i	1	3	0	5	3	5	6	8	8	2	12
f_i	4	5	6	7	8	9	10	11	12	13	14

Estratégia gulosa

- Moldar o problema de otimização como um problema no qual fazemos uma escolha e ficamos com um único subproblema para resolver.
- Provar que sempre existe uma solução ótima para o problema original que contém a escolha gulosa.
- Demonstrar que, tendo feita escolha gulosa, o que resta é um subproblema com a propriedade de que, se combinarmos uma solução ótima para o subproblema com a escolha gulosa que fizemos, chegamos a uma solução ótima para o problema original.

Problema da mochila
Knapsack problem

- Há n itens, onde o i-ésimo item vale v_i e pesa w_i quilos.
- Deve-se colocar uma carga tão valiosa quanto possível em uma mochila, mas ela comporta no máximo w quilos.

Problema da mochila

- Problema da mochila 0-1
  - Subestrutura ótima: Para a carga mais valiosa que pese no máximo w quilos, se removermos o item j, a carga restante deve ser a mais valiosa que pese w – w_j.
- Problema da mochila fracionada
  - Se removermos um peso w de um item j da carga ótima, a carga restante deve ser mais valiosa que pese no máximo W-w que o ladrão pode levar dos n-1 itens originais, mais w_j-w do item j.

Problema da mochila

- Fracionada
  - Estratégia gulosa
    - Divida v_i/w_i para cada item
    - Pega o máximo do item de maior valor por quilo
    - Se o suprimento deste item esgotar e puder levar mais, pega o máximo possível do próximo item com maior valor por quilo.
- Exemplo:
  - Mochila – 50 quilos
  - Item 1 – 10 quilos, $60
  - Item 2 – 20 quilos, $100
  - Item 3 – 30 quilos, $120