

Projeto e Análise de Algoritmos

Aula 8:

Algoritmos Gulosos (DPV 5; CLRS 4)

DECOM/UFOP

2020 – 5º. Período

Anderson Almeida Ferreira

Adaptado do material de Andréa

Iabrudi Tavares



Objetivos

- Entender os conceitos e aplicabilidade de Algoritmos Gulosos
- Entender o papel de estrutura de dados na eficiência de algoritmos
- Conhecer abordagens gulosas ótimas para alguns problemas
- Saber utilizar abordagem gulosa para solução sub-ótima
- Bibliografia
 - DPV 5.1-5.3, CLRS 4.

Algoritmos Gulosos

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

Um problema de seleção de atividades

- Seja $S = \{a_1, a_2, \dots, a_n\}$ 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 início s_i e um tempo de término 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 antes da atividade a_j começar.
- Acrescentamos duas atividades fictícias a_0 e a_{n+1} , onde $f_0 = 0$ e $s_{n+1} = \infty$.
- Logo, $S = S_{0\ n+1}$
- 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

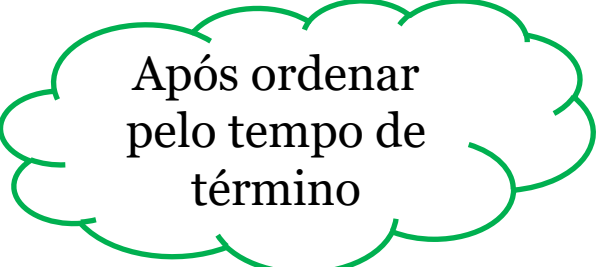
- Uma solução ótima para S_{ij} seria

$$A_{ij} = A_{ik} \cup \{a_k\} \cup A_{kj}$$
- $A_{0 \ n+1}$ 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 que 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 atividades 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 $A'_{ij} = A_{ij} - \{a_k\} \cup \{a_m\}$.

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



Após ordenar
pelo tempo de
término

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 a 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.

Comparação dos Problemas

Seja um grafo $G=(V,E)$ e uma função de custo w , pede-se

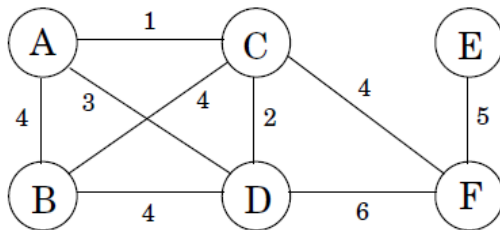
- Caixeiro Viajante
Ciclo que passe uma única vez por todos os vértices e tenha peso mínimo.
- Árvore Geradora Mínima
Árvore que passe por todos os vértices e tenha peso mínimo.

Revisão de Conceitos de Grafo

- O que é ciclo.
- O que é árvore.
- Como determinar se é uma árvore
 - Número de nós
 - Conexos
- Número de nós n e arestas m .
 - Em grafo conexo, $n-1 \leq m < n^2$

Árvores Geradoras Mínimas (AGM)

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



Propriedades

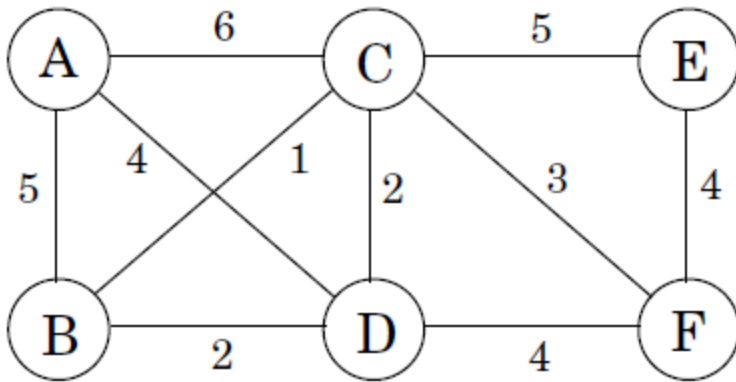
Grafo $G=(V,E,w)$ conexo $n=|V|$
Solução é uma árvore ($n-1$ arestas)

Formulação

Dado um grafo $G=(V,E,w)$, encontrar árvore $A=(V,E_A,w)$ $E_A \subseteq E$ tal que $w(E_A)$ é mínimo.

Guloso AGM: Algoritmo de Kruskal

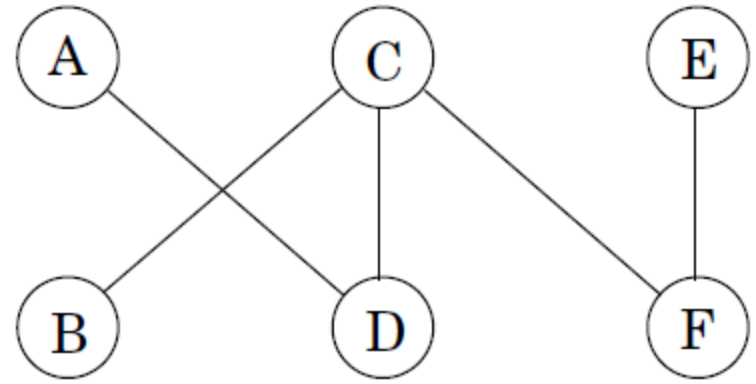
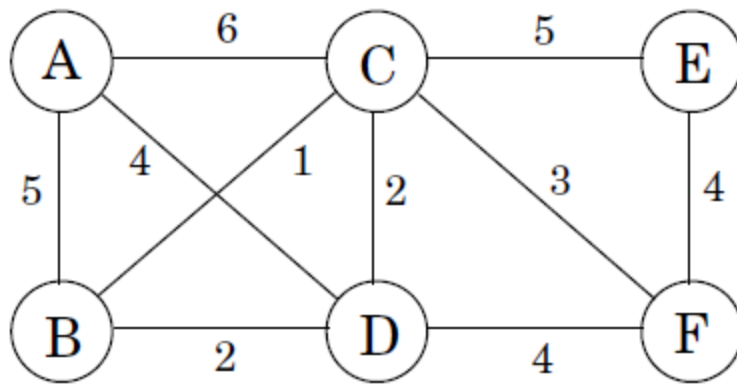
- Adicione sempre a aresta de menor peso que não forma ciclo (1956).



Desempates: ordem lexicográfica

Guloso AGM: Algoritmo de Kruskal

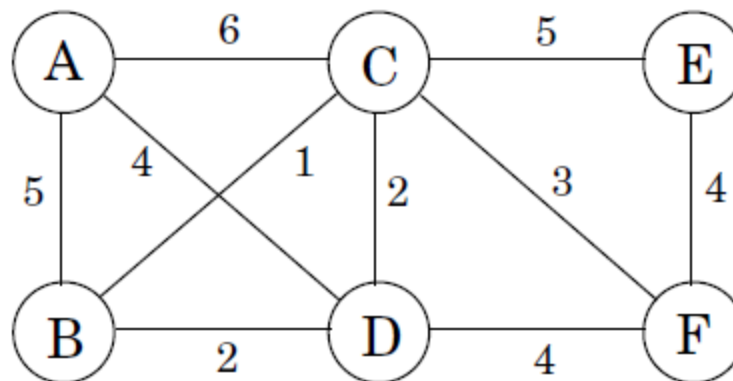
- Adicione sempre a aresta de menor peso que não forma ciclo.



Está correto, ou seja, é ótimo?????

Kruskal está correto: cortes

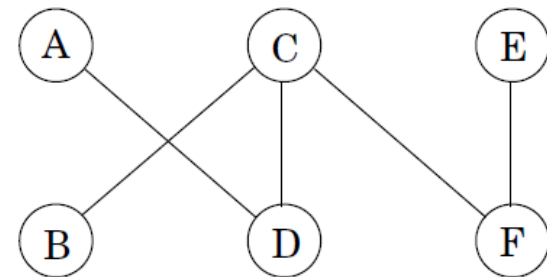
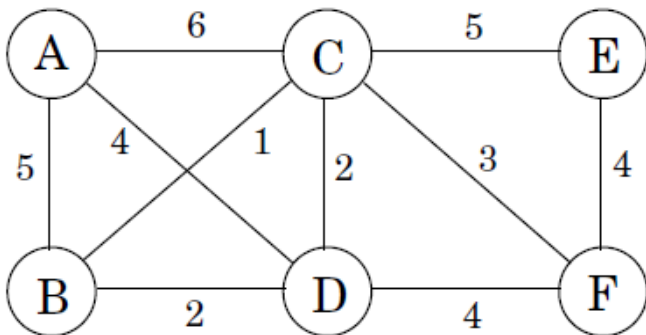
- Partição $(S, V-S)$ ou de forma equivalente conjunto de arestas.

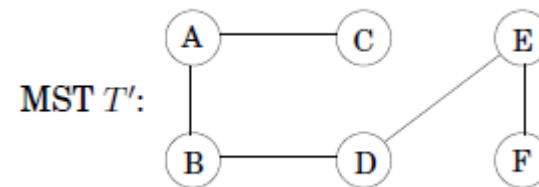
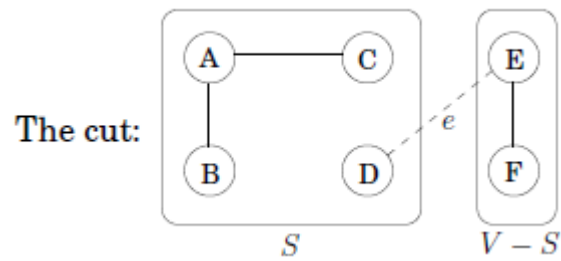
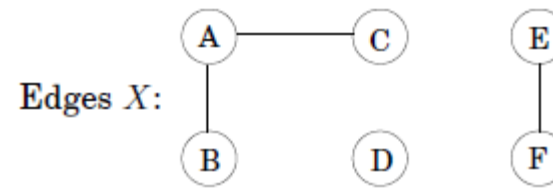
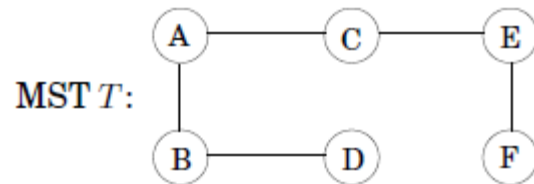
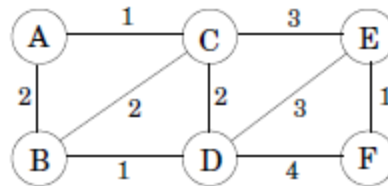


Kruskal está correto

Seja X um subconjunto das arestas E_A de uma 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 $X \cup \{e\}$ faz parte de alguma AGM.

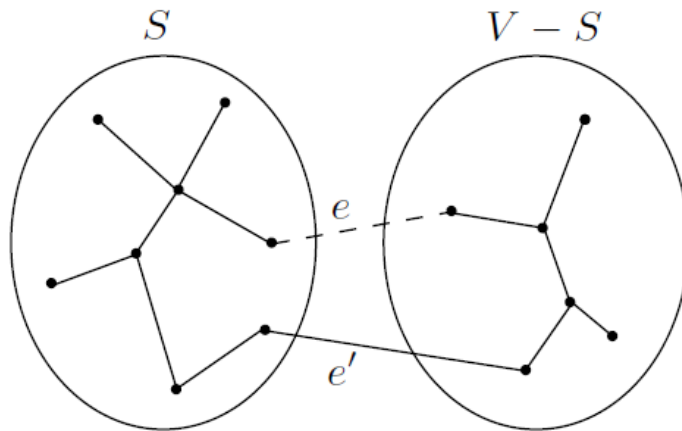




Kruskal está correto

- Prova

Seja X um subconjunto das arestas E_A de uma 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 $X \cup \{e\}$ faz parte de alguma AGM.



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  $\{u, v\} \in E$ , in increasing order of weight:
```

```
    if  $\{u, v\} \cup X$  sem ciclo
```

```
        add edge  $\{u, v\}$  to  $X$ 
```

Algoritmo está correto como visto anteriormente.

Depende de ordenação de arestas ($m \log m$ ou m - linear).

Faz $O(m)$ vezes a operação de verificação de ciclo.

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  $\{u, v\} \in E$ , in increasing order of weight:
```

```
    if  $\{u, v\} \cup X$  sem ciclo
```

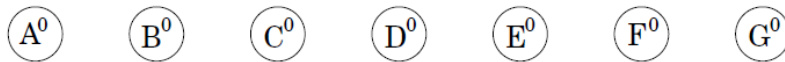
```
        add edge  $\{u, v\}$  to  $X$ 
```

Complexidade depende de qual operação? De saber se há ciclo...

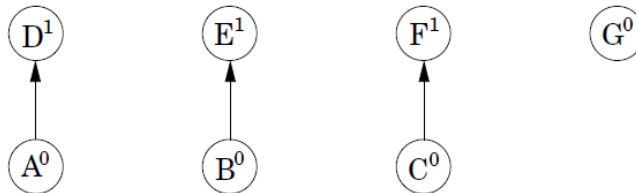
Conjuntos Disjuntos

Mesmo componente em $\log(n)$

$\text{makeset}(A), \text{makeset}(B), \dots, \text{makeset}(G):$

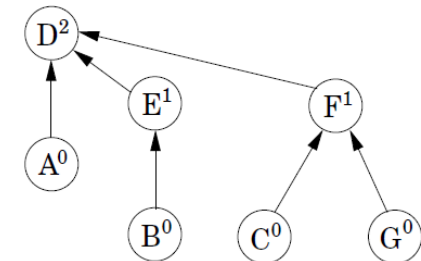
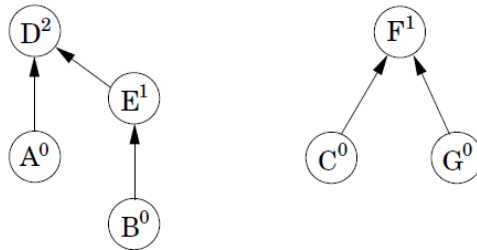


$\text{union}(A, D), \text{union}(B, E), \text{union}(C, F):$



$\text{union}(B, G):$

$\text{union}(C, G), \text{union}(E, A):$



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  $u \in V$ :
```

```
    makeset( $u$ )
```

```
 $X = \{\}$ 
```

```
Sort the edges  $E$  by weight
```

```
for all edges  $\{u, v\} \in E$ , in increasing order of weight:
```

```
    if find( $u$ )  $\neq$  find( $v$ ):
```

```
        add edge  $\{u, v\}$  to  $X$ 
```

```
        union( $u, v$ )
```

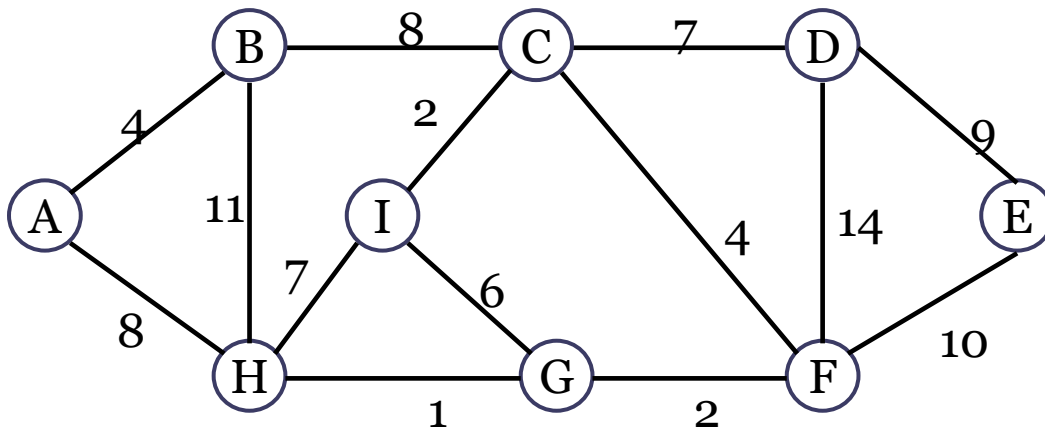

Algoritmo para componentes disjuntos

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

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

```
procedure makeset( $x$ )  
   $\pi(x) = x$   
   $\text{rank}(x) = 0$ 
```

```
procedure union( $x, y$ )  
   $r_x = \text{find}(x)$   
   $r_y = \text{find}(y)$   
  if  $r_x = r_y$ : return  
  if  $\text{rank}(r_x) > \text{rank}(r_y)$ :  
     $\pi(r_y) = r_x$   
  else:  
     $\pi(r_x) = r_y$   
    if  $\text{rank}(r_x) = \text{rank}(r_y)$ :  $\text{rank}(r_y) = \text{rank}(r_y) + 1$ 
```



procedure union(x, y)

$r_x = \text{find}(x)$

$r_y = \text{find}(y)$

if $r_x = r_y$: return

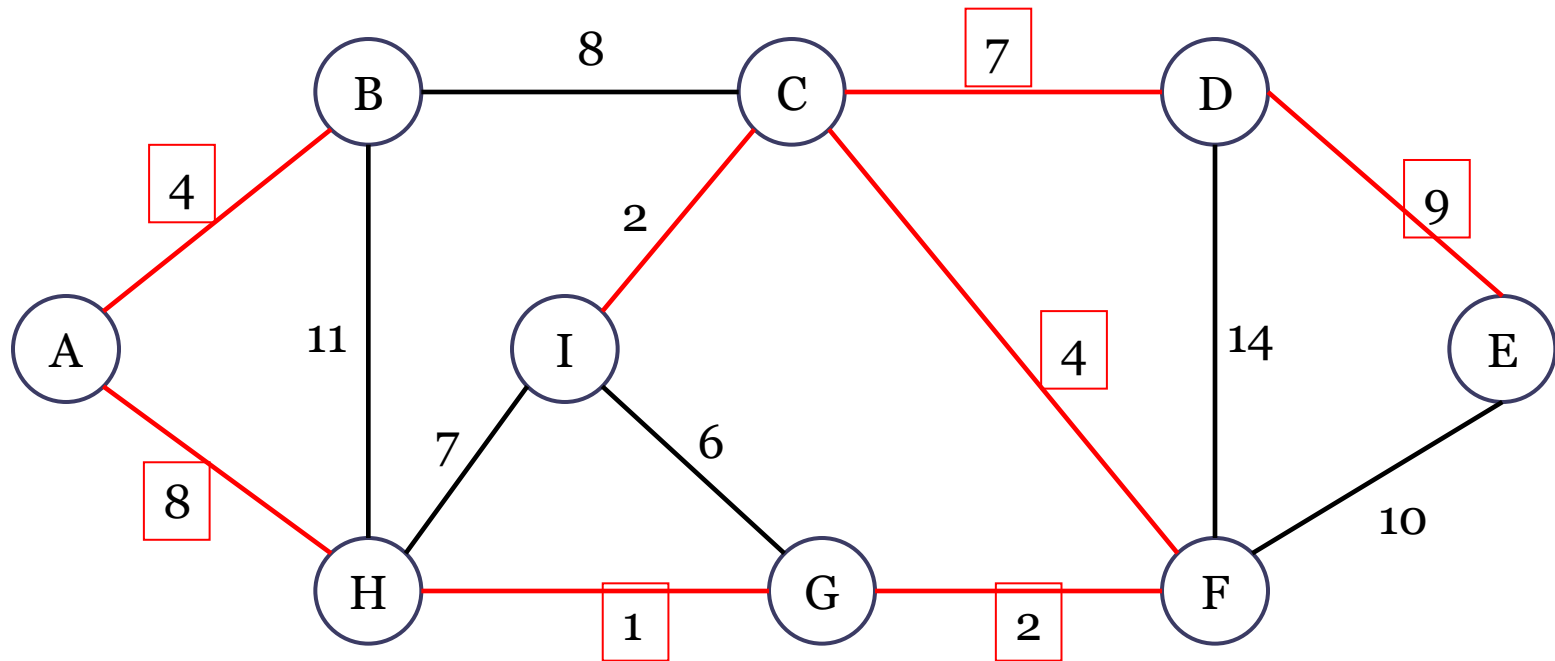
if $\text{rank}(r_x) > \text{rank}(r_y)$:

$\pi(r_y) = r_x$

else:

$\pi(r_x) = r_y$

if $\text{rank}(r_x) = \text{rank}(r_y)$: $\text{rank}(r_y) = \text{rank}(r_y) + 1$



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  $u \in V$ :
```

```
    makeset( $u$ )
```

```
 $X = \{\}$ 
```

```
Sort the edges  $E$  by weight
```

```
for all edges  $\{u, v\} \in E$ , in increasing order of weight:
```

```
    if find( $u$ )  $\neq$  find( $v$ ):
```

```
        add edge  $\{u, v\}$  to  $X$ 
```

```
        union( $u, v$ )
```

Complexidade?

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 $u \in V$:

 makeset(u)

$X = \{\}$

Sort the edges E by weight

for all edges $\{u, v\} \in E$, in increasing order of weight:

 if find(u) \neq find(v):

 add edge $\{u, v\}$ to X

 union(u, v)

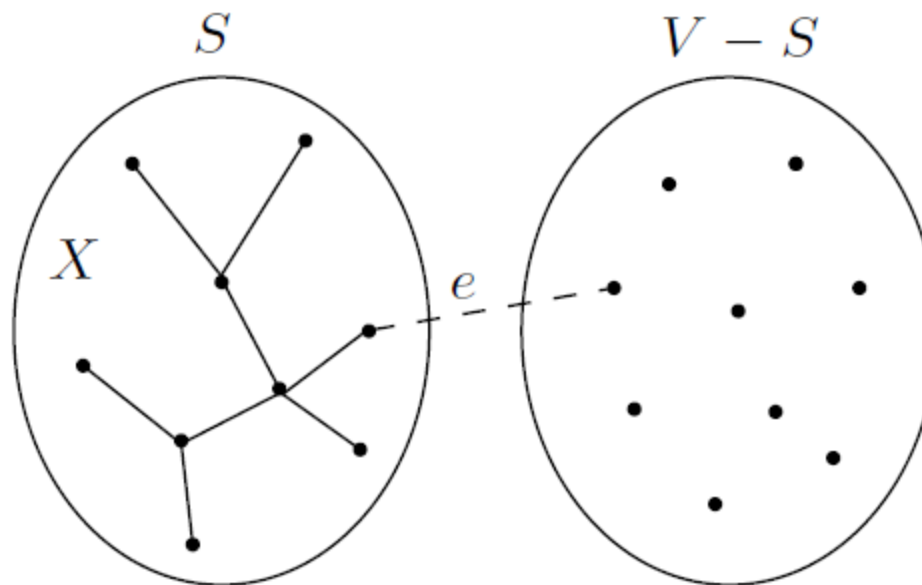
Complexidade?

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

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

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)



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  $u \in V$ :
```

```
     $\text{cost}(u) = \infty$ 
```

```
     $\text{prev}(u) = \text{nil}$ 
```

```
Pick any initial node  $u_0$ 
```

```
 $\text{cost}(u_0) = 0$ 
```

```
 $H = \text{makequeue}(V)$      (priority queue, using cost-values as keys)
```

```
while  $H$  is not empty:
```

```
     $v = \text{deletemin}(H)$ 
```

```
    for each  $\{v, z\} \in E$ :
```

```
        if  $\text{cost}(z) > w(v, z)$ :
```

```
             $\text{cost}(z) = w(v, z)$ 
```

```
             $\text{prev}(z) = v$ 
```

```
             $\text{decreasekey}(H, z)$ 
```

Guloso Menor caminho: Dijkstra

```
procedure dijkstra( $G, l, s$ )
```

```
Input:      Graph  $G = (V, E)$ , directed or undirected;  
           positive edge lengths  $\{l_e : e \in E\}$ ; vertex  $s \in V$ 
```

```
Output:     For all vertices  $u$  reachable from  $s$ ,  $\text{dist}(u)$  is set  
           to the distance from  $s$  to  $u$ .
```

```
for all  $u \in V$ :
```

```
     $\text{dist}(u) = \infty$ 
```

```
     $\text{prev}(u) = \text{nil}$ 
```

```
 $\text{dist}(s) = 0$ 
```

```
 $H = \text{makequeue}(V)$  (using  $\text{dist}$ -values as keys)
```

```
while  $H$  is not empty:
```

```
     $u = \text{deletemin}(H)$ 
```

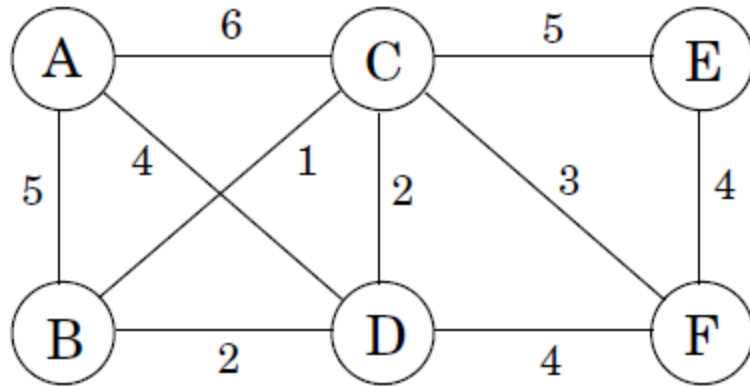
```
    for all edges  $(u, v) \in E$ :
```

```
        if  $\text{dist}(v) > \text{dist}(u) + l(u, v)$ :
```

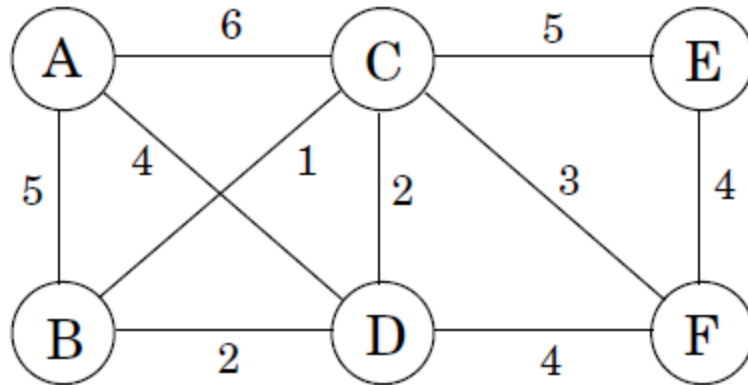
```
             $\text{dist}(v) = \text{dist}(u) + l(u, v)$ 
```

```
             $\text{prev}(v) = u$ 
```

```
             $\text{decreasekey}(H, v)$ 
```

Set S	A	B	C	D	E	F
$\{\}$	0/nil	∞ /nil	∞ /nil	∞ /nil	∞ /nil	∞ /nil



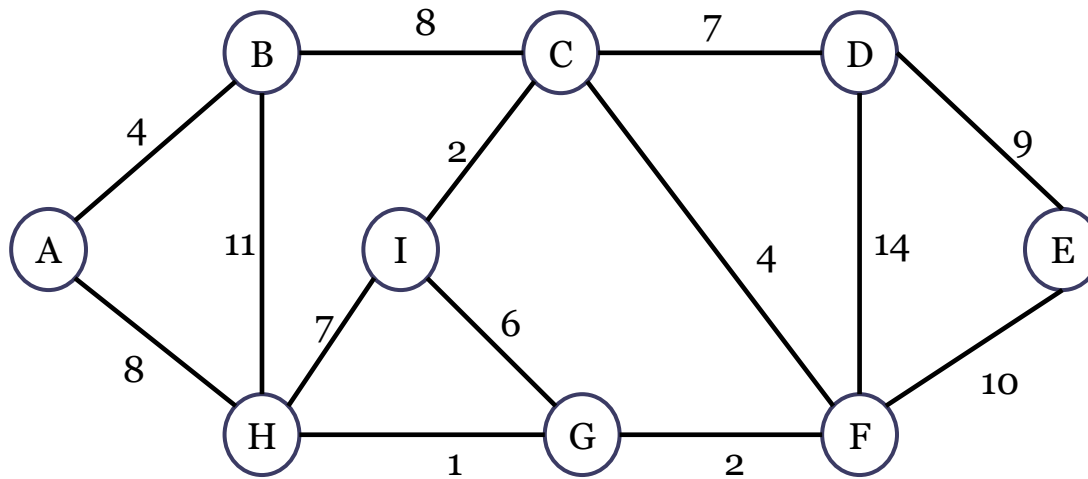
Atenção: apesar de ser linear nesse exemplo, representa qualquer árvore.

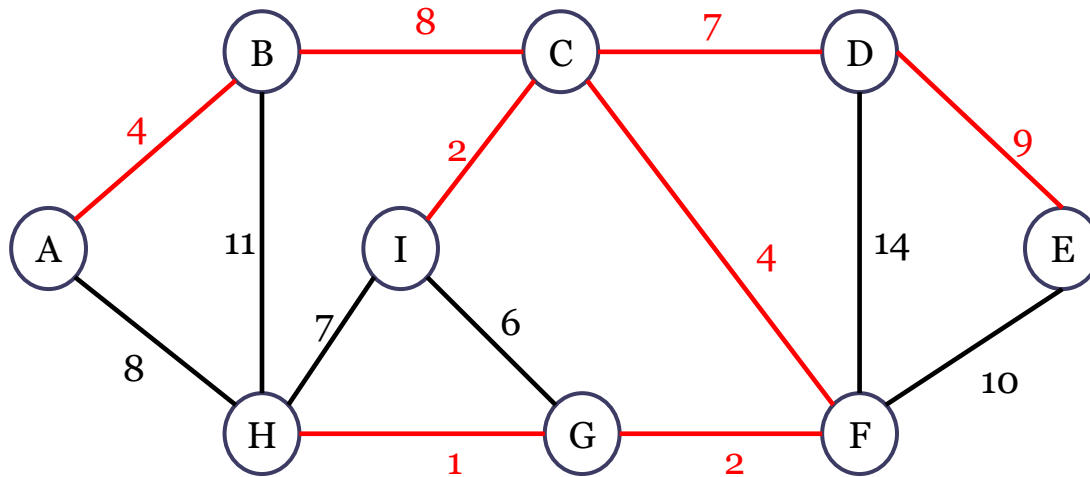
prev:
permite
recuperar
árvore

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	

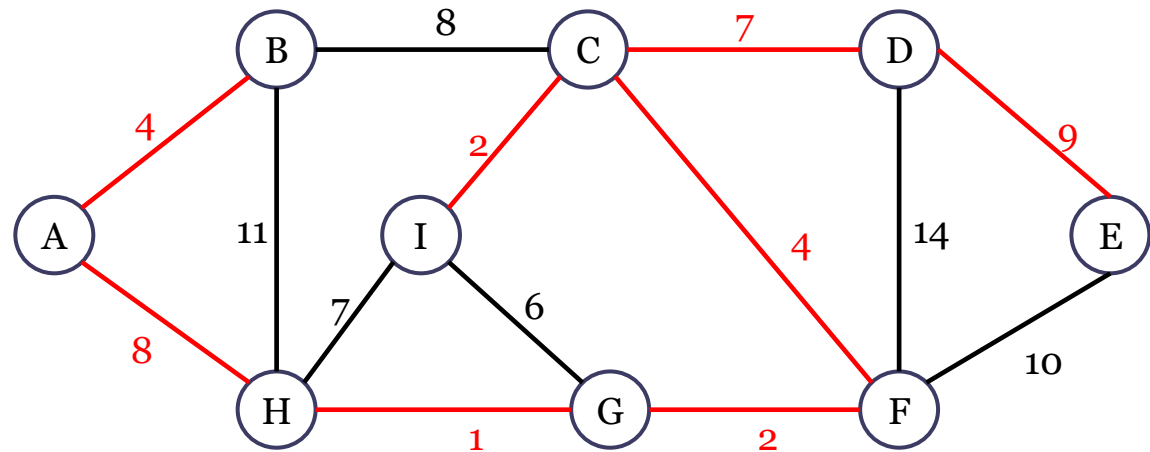
Outro exemplo:

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





Árvores podem
ser diferentes,
mas têm o
mesmo peso!

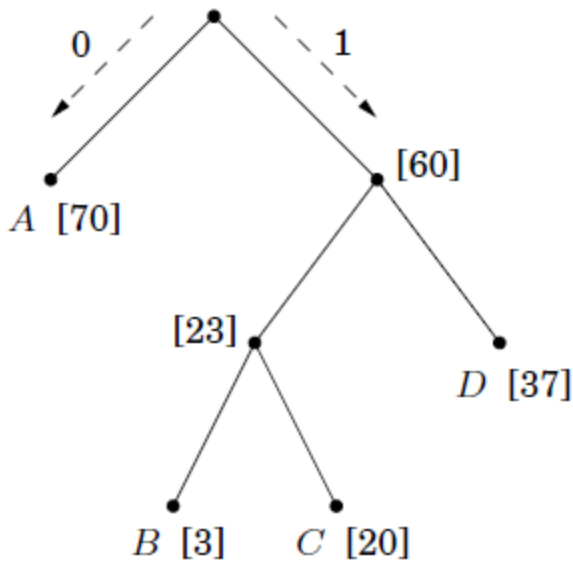
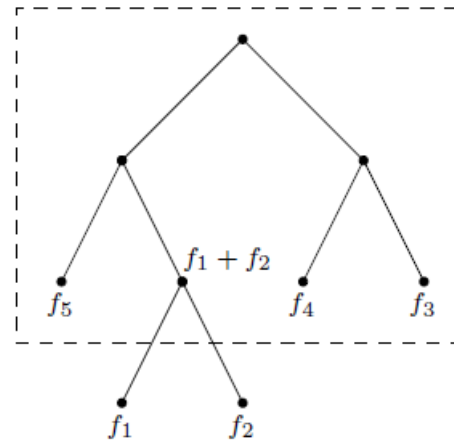


Código de Huffman

Symbol	Frequency
<i>A</i>	70 million
<i>B</i>	3 million
<i>C</i>	20 million
<i>D</i>	37 million

Código de Huffman

Symbol	Frequency
<i>A</i>	70 million
<i>B</i>	3 million
<i>C</i>	20 million
<i>D</i>	37 million



Symbol	Codeword
<i>A</i>	0
<i>B</i>	100
<i>C</i>	101
<i>D</i>	11

Código de Huffman - Fazendo as árvores

Symbol	Frequency
<i>A</i>	70 million
<i>B</i>	3 million
<i>C</i>	20 million
<i>D</i>	37 million

Código de Huffman

- 1952, David Huffman
- Compressão (MP3, por exemplo)
- Propriedades
 - Número variável de bits
 - Não-ambiguidade (árvore binária completa)

Algoritmo Guloso: Huffman Ótimo

```
procedure Huffman( $f$ )
```

```
Input:  An array  $f[1 \dots n]$  of frequencies
```

```
Output: An encoding tree with  $n$  leaves
```

```
let  $H$  be a priority queue of integers, ordered by  $f$ 
```

```
for  $i = 1$  to  $n$ : insert( $H, i$ )
```

```
for  $k = n + 1$  to  $2n - 1$ :
```

```
     $i = \text{deletemin}(H)$ ,  $j = \text{deletemin}(H)$ 
```

```
    create a node numbered  $k$  with children  $i, j$ 
```

```
     $f[k] = f[i] + f[j]$ 
```

```
    insert( $H, k$ )
```

Huffman: Complexidade

```
procedure Huffman( $f$ )
```

```
Input:  An array  $f[1 \dots n]$  of frequencies
```

```
Output: An encoding tree with  $n$  leaves
```

```
let  $H$  be a priority queue of integers, ordered by  $f$ 
```

```
for  $i = 1$  to  $n$ : insert( $H, i$ )
```

```
for  $k = n + 1$  to  $2n - 1$ :
```

```
     $i = \text{deletemin}(H)$ ,  $j = \text{deletemin}(H)$ 
```

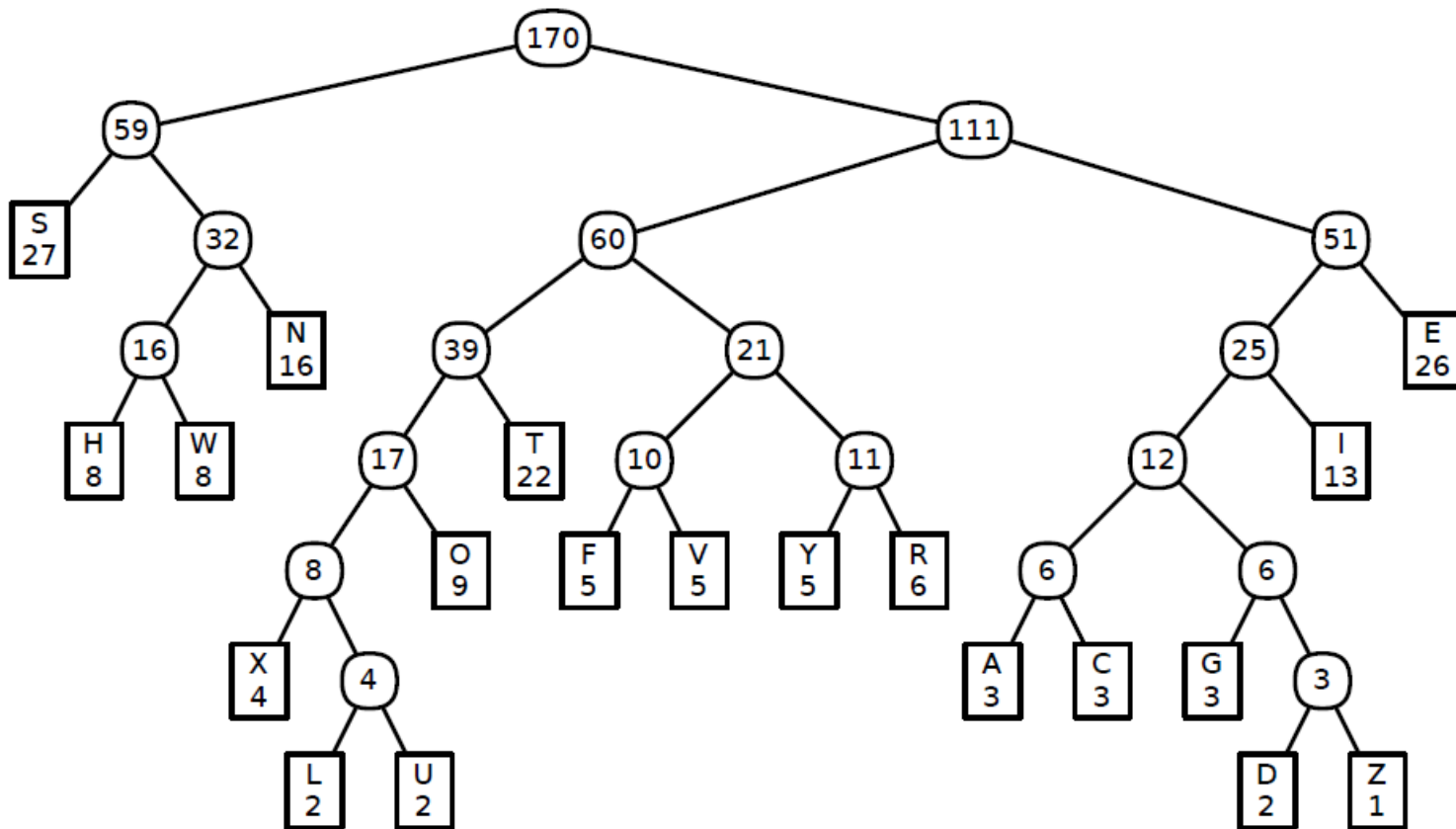
```
    create a node numbered  $k$  with children  $i, j$ 
```

```
     $f[k] = f[i] + f[j]$ 
```

```
    insert( $H, k$ )
```

Complexidade: $O(n \log n)$

A	C	D	E	F	G	H	I	L	N	O	R	S	T	U	V	W	X	Y	Z
3	3	2	26	5	3	8	13	2	16	9	6	27	22	2	5	8	4	5	1



Huffman: Exercício

Suponha que os símbolos a, b, c, d, e ocorram com frequências $\frac{1}{2}$, $\frac{1}{4}$, $\frac{1}{8}$, $\frac{1}{16}$, $\frac{1}{16}$, respectivamente.

- a) Qual a codificação de Huffman do alfabeto?
- b) Se esta codificação é aplicada a um arquivo consistindo em 1.000.000 caracteres com as dadas frequências, qual é o tamanho do arquivo codificado em bits?

Um problema de lógica

A mulher do coronel foi assassinada e existem três suspeitos: o coronel, o açougueiro e o amante.

Definir quem é o assassino, dado que:

o assassinato aconteceu na cozinha

o assassinato aconteceu às 8 da noite

o coronel estava dormindo às 8 da noite

Problema de satisfabilidade

A mulher do coronel foi assassinada e existem três suspeitos: o coronel, o açougueiro e o amante. Definir quem é o assassino, dado que:

o assassinato aconteceu na cozinha

o assassinato aconteceu às 8 da noite

o coronel estava dormindo às 8 da noite

U – o coronel é inocente

X – o assassinato aconteceu na cozinha

W – o assassinato aconteceu às 8 da noite

Y – o açougueiro é inocente

Z – o coronel estava dormindo às 8 da noite

Cláusulas de Horn

- 1951, Alfred Horn
- É possível deduzir um fato a partir de um conjunto de implicações?
 - Fatos: variáveis lógicas
 - Implicações : com antecedente de conjunto de fatos e um conseqüente unitário
- Conjunção de cláusulas de disjunção com no máximo uma variável não-negada
 - Subconjunto de problemas de satisfabilidade

Cláusulas de Horn

A mulher do coronel foi assassinada e existem três suspeitos: o coronel, o açougueiro e o amante. Definir quem é o assassino, dado que:

o assassinato aconteceu na cozinha

o assassinato aconteceu às 8 da noite

o coronel estava dormindo às 8 da noite

U – o coronel é inocente

X – o assassinato aconteceu na cozinha

W – o assassinato aconteceu às 8 da noite

Y – o açougueiro é inocente

Z – o coronel estava dormindo às 8 da noite

Cláusulas de Horn

$$(w \wedge y \wedge z) \Rightarrow x, \quad (x \wedge z) \Rightarrow w, \quad x \Rightarrow y, \quad \Rightarrow x, \quad (x \wedge y) \Rightarrow w, \quad (\overline{w} \vee \overline{x} \vee \overline{y}), \quad (\overline{z}).$$

Horn - Algoritmo Guloso Exato

Input: a Horn formula

Output: a satisfying assignment, if one exists

```
set all variables to false
```

```
while there is an implication that is not satisfied:  
    set the right-hand variable of the implication to true
```

```
if all pure negative clauses are satisfied: return the assignment  
else: return ``formula is not satisfiable``
```

Implementação em tempo linear: Norvig and Russel,
Lógica Proposicional Forward Chaining

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