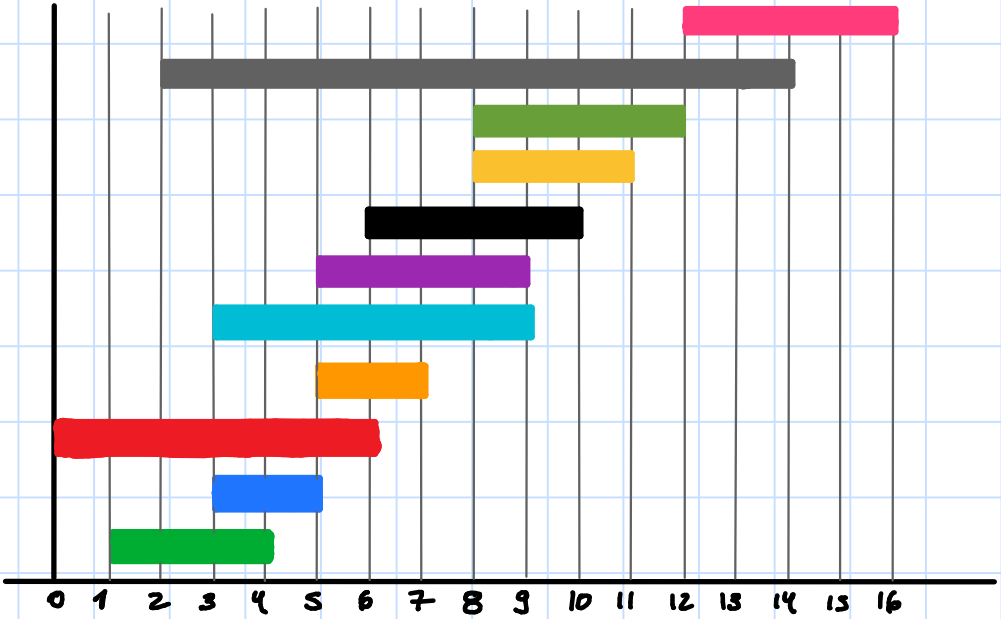
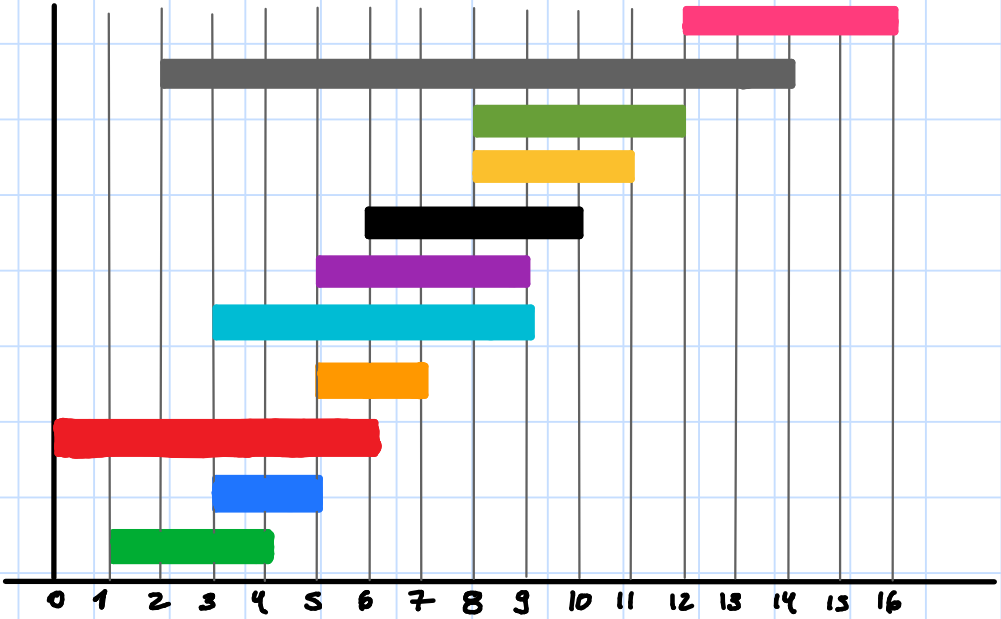
Algoritmos Gulosos

Problema da Seleção de Atividades



Dado um conjunto de atividades $S = \{a_1, a_2, \ldots, a_n\}$ tal que a atividade a_i acontece no intervalo de tempo $[s_i, f_i)$

Problema da Seleção de Atividades

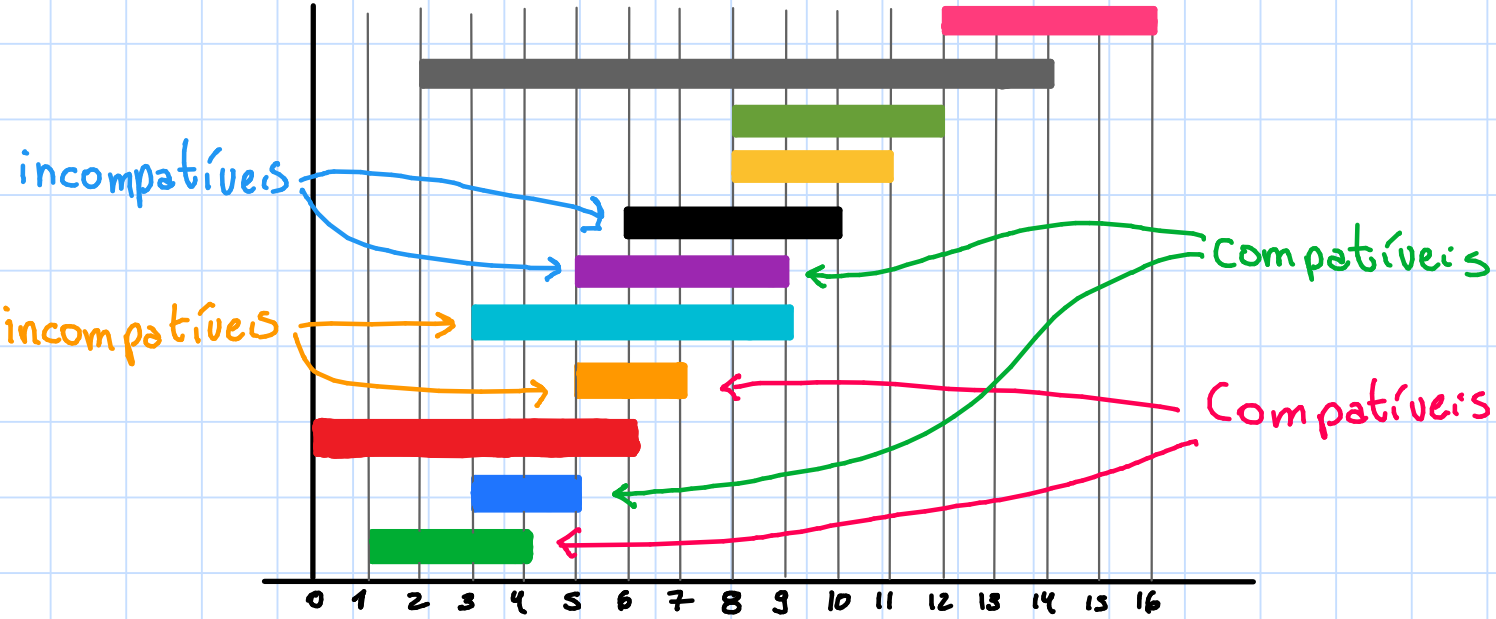


- As atividades competem por um recurso exclusivo
- Dizemos que as atividades a_i e a_j são <u>compatíveis</u> se

$$f_i \leq s_j \quad \text{ou} \quad f_j \leq s_i$$

Caso contrário, dizemos que são <u>incompatíveis</u>

Problema da Seleção de Atividades

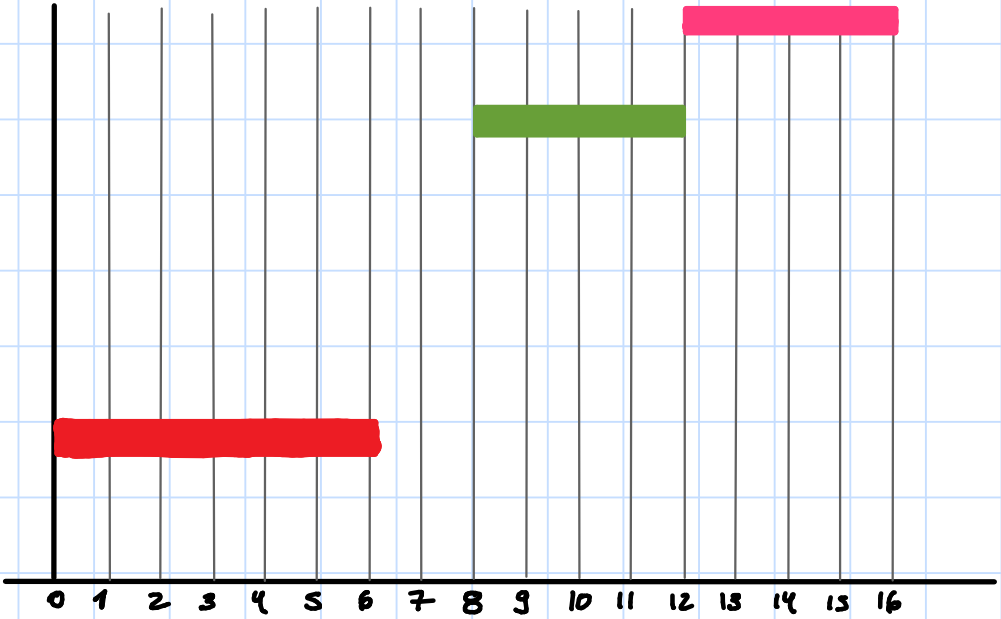


- As atividades competem por um recurso exclusivo
- Dizemos que as atividades a_i e a_j são <u>compatíveis</u> se

$$f_i \leq s_j \quad \text{ou} \quad f_j \leq s_i$$

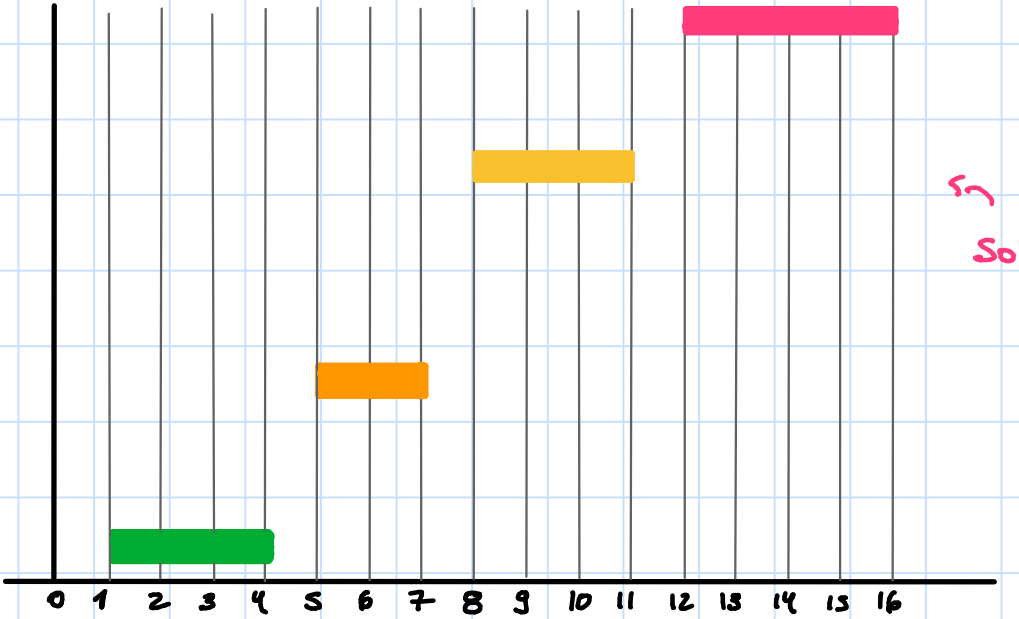
Caso contrário, dizemos que são <u>incompatíveis</u>

Problema da Seleção de Atividades



Queremos encontrar um subconjunto $S^* \subseteq S$ de atividades compatíveis ...

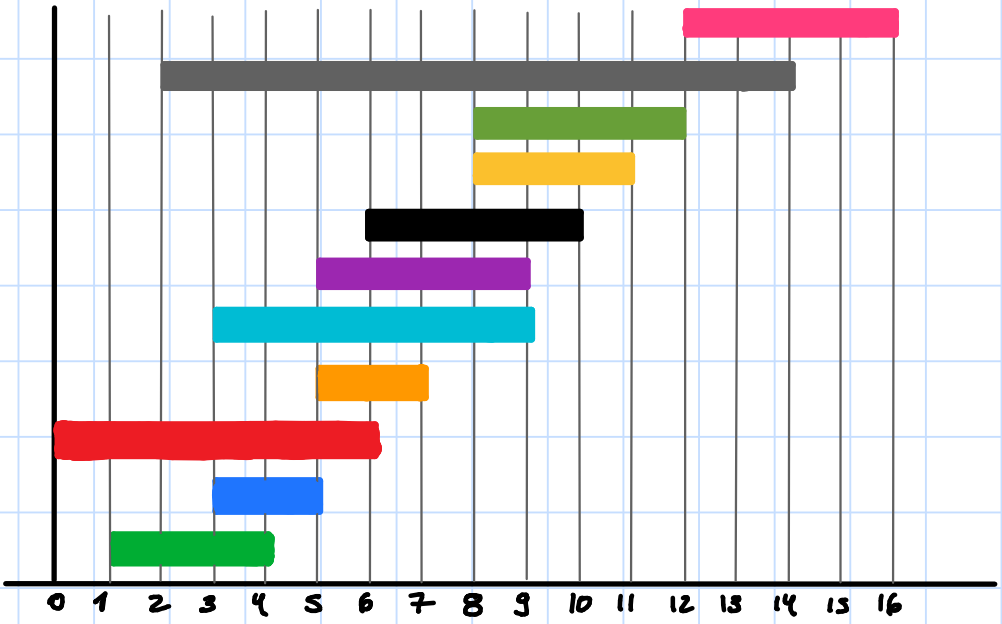
Problema da Seleção de Atividades



↳ solução ótima

Queremos encontrar um subconjunto $S^* \subseteq S$ de atividades compatíveis de cardinalidade máxima

Problema da Seleção de Atividades



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	9	10	10	11	12	14	16

Problema da Seleção de Atividades

Entrada: Uma conjunto $S = \{a_1, a_2, \ldots, a_n\}$ de atividades tal que a atividade a_i acontece no intervalo de tempo $[s_i, t_i)$

Saída: Um subconjunto $S^* \subseteq S$ de tarefas compatíveis de cardinalidade máxima

Problema da Seleção de Atividades

Entrada: Uma conjunto $S = \{a_1, a_2, \ldots, a_n\}$ de atividades tal que a atividade a_i acontece no intervalo de tempo $[s_i, t_i)$

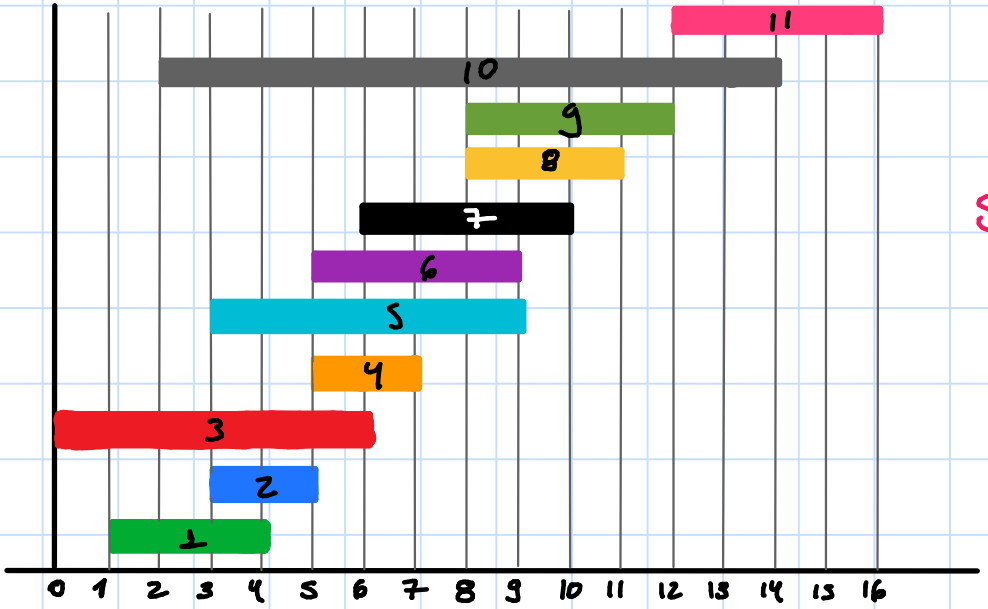
Saída: Um subconjunto $S^* \subseteq S$ de tarefas compatíveis de cardinalidade máxima

Vamos assomir que as atividades estão ordenadas em ordem não decrescente de término, i.e.,

$$t_1 \leq t_2 \leq t_3 \leq \cdots \leq t_n$$

Notação

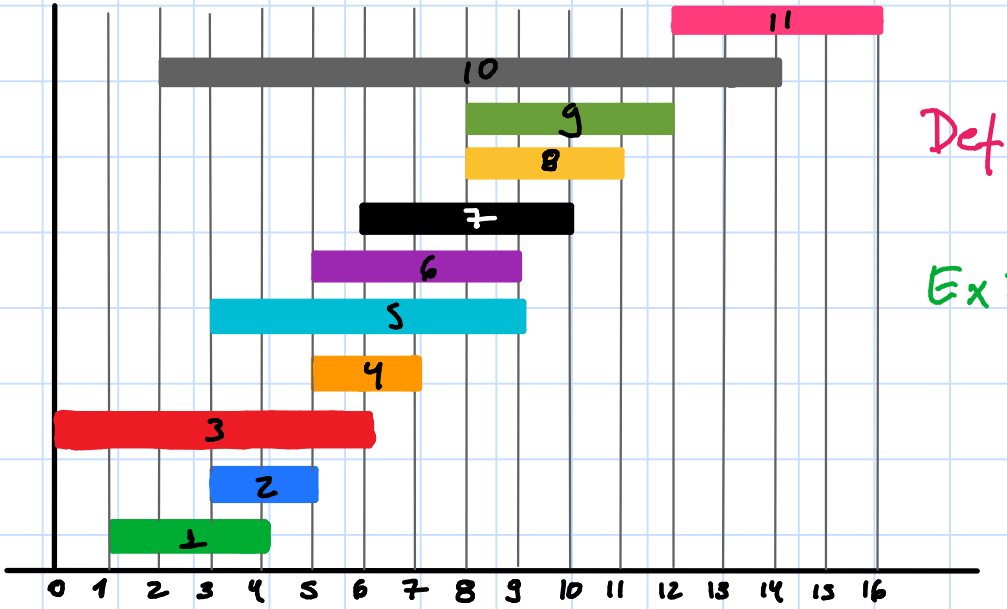
- Vamos denotar por S_{ij} o conjunto das atividades cujo tempo de realização se encontra no intervalo $[f_i, s_j)$



$S_{2,11} = \{4, 6, 7, 8, 9\}$

Notação

- Vemos denotar por S_{ij} o conjunto das atividades cujo tempo de realização se encontra no intervalo $[f_i, s_j)$



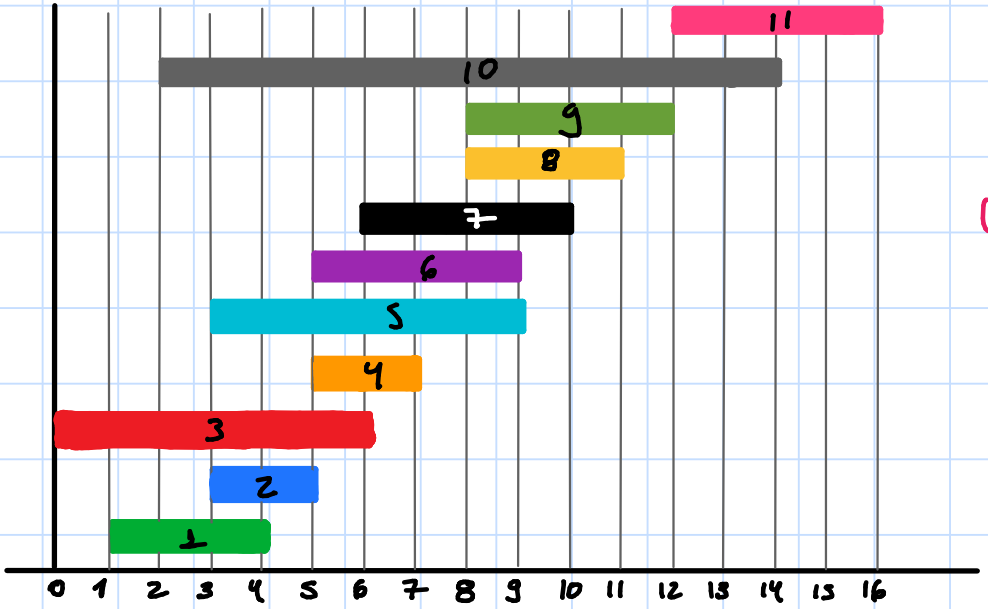
Def. $f_0 = 0$ e $s_{n+1} = f_n$

Ex:

$S_{0,n+1} = S$

Notação

- Vamos denotar por $\varphi_{i,j}$ o tamanho de uma solução ótima para $S_{i,j}$



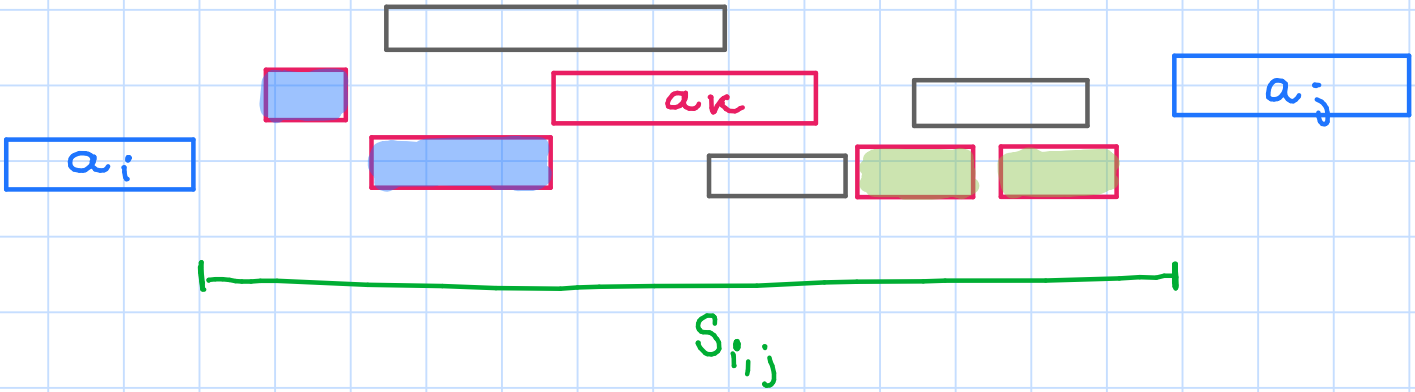
$\varphi_{0,12} = 4$

Subestrutura Ótima

- Seja $A \subseteq S_{i,j}$ uma solução ótima, portanto

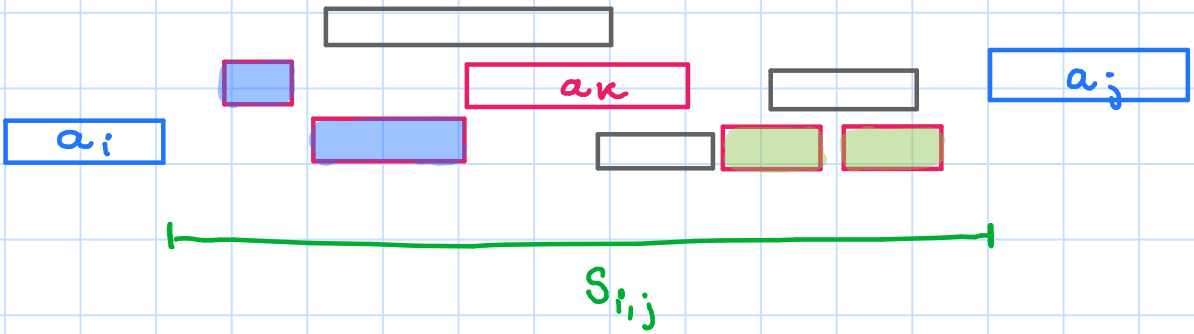
$$|A| = \psi_{i,j}$$

- Seja $a_k \in A$
- Seja $A_{i,k} = A \cap S_{i,k}$ e $A_{k,j} = A \cap S_{k,j}$



- $|A| = |A_{i,k}| + 1 + |A_{k,j}|$

Subestrutura Ótima



- $|A| = |A_{i,k}| + 1 + |A_{k,j}|$

  $\|$ $\|$ $\|$

  $\psi_{i,j}$ $\psi_{i,k}$ $\psi_{k,j}$

← por um argumento "corta e cola" (contradição)

Temos subestrutura ótima!

Descobrindo um Algoritmo

$$\psi_{i,j} = \psi_{i,k} + \psi_{k,j} + 1$$

- não sabemos quem é k
- sabemos que $a_k \in S_{i,j}$
  - Podemos testar todos!

$$\psi_{i,j} = \max_{a_\ell \in S_{i,j}} \{ \psi_{i,\ell} + \psi_{\ell,j} + 1 \}$$

Descobrindo um Algoritmo

$$\psi_{i,j} = \max_{a_\ell \in S_{i,j}} \{ \psi_{i,\ell} + \psi_{\ell,j} + 1 \}$$

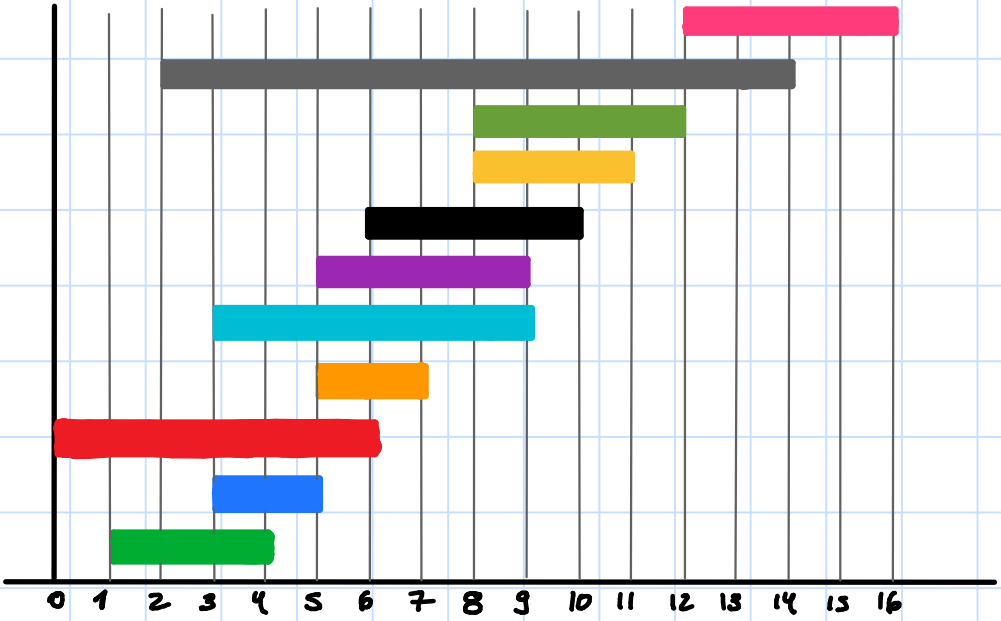
- Agora podemos fazer um algoritmo recursivo, uma Programação Dinâmica e etc

Descobrindo um Algoritmo

$$\psi_{i,j} = \max_{a_\ell \in S_{i,j}} \{ \psi_{i,\ell} + \psi_{\ell,j} + 1 \}$$

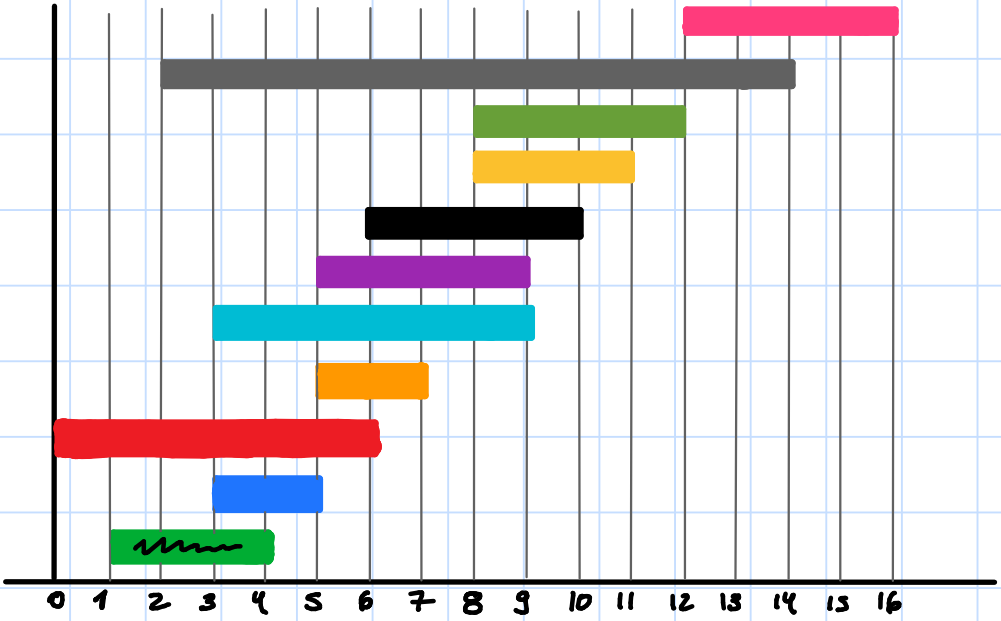
- Agora podemos fazer um algoritmo recursivo, uma Programação Dinâmica e etc
  - Usar PD nesse problema é e não gera o algoritmo mais eficiente

Descobrindo um Algoritmo



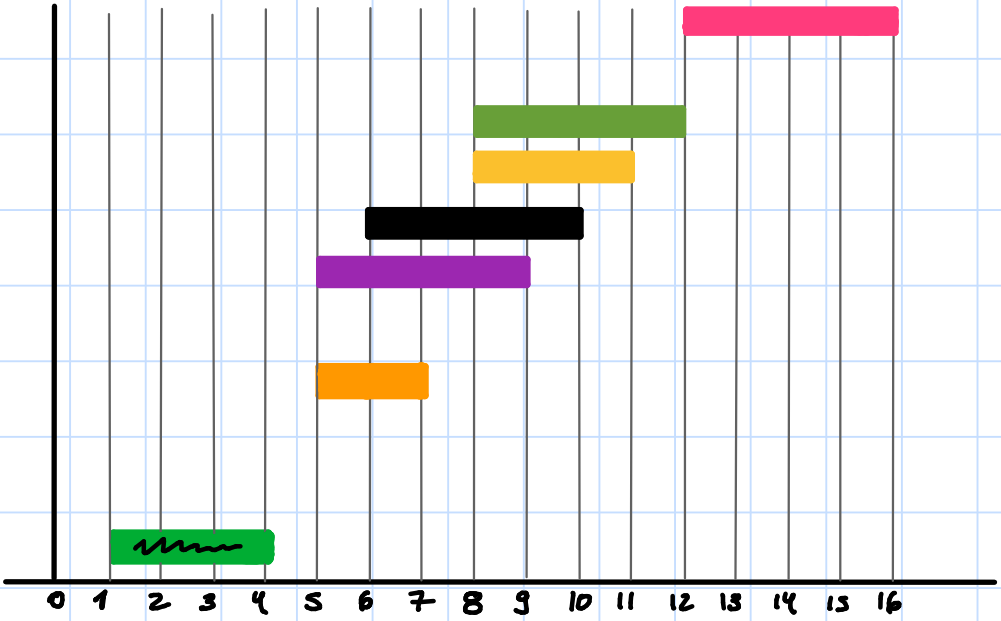
intuitivamente, para maximizar o número de tarefas escolhemos uma tarefa que termine o mais rápido possível, assim sobra mais espaço para colocar as outras.

Descobrindo um Algoritmo



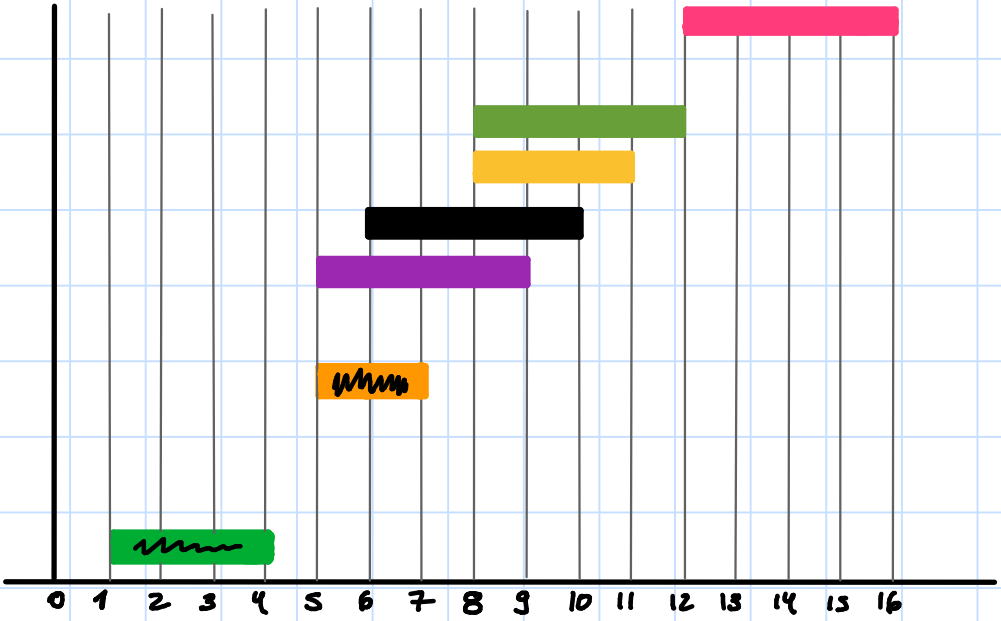
Pegamos "gulosamente" o "melhor" atividade entre as atividades restantes

Descobrindo um Algoritmo



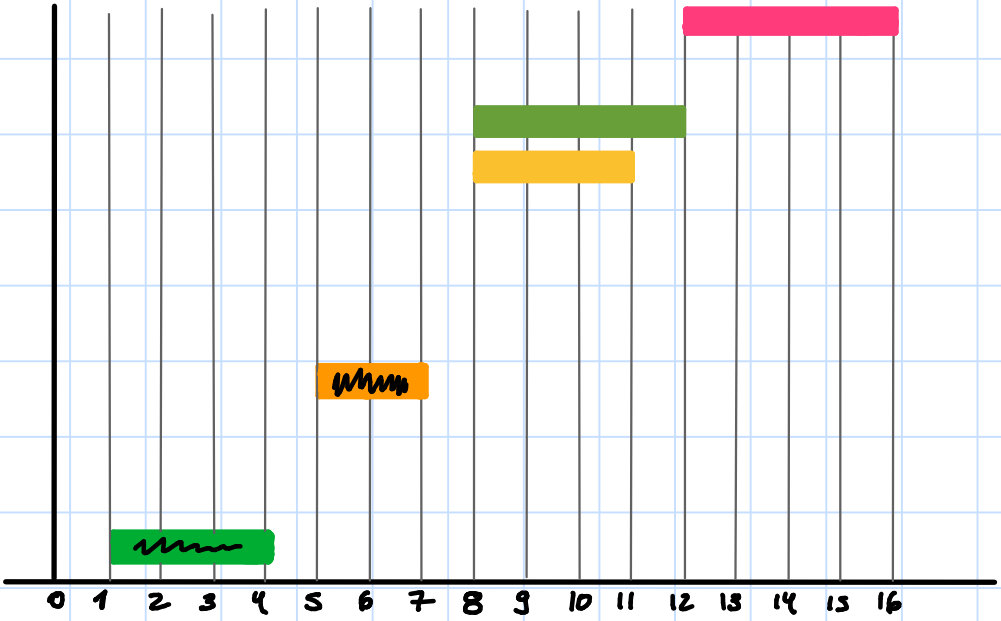
Pegamos "gulosamente" o "melhor" atividade entre as atividades restantes

Descobrindo um Algoritmo



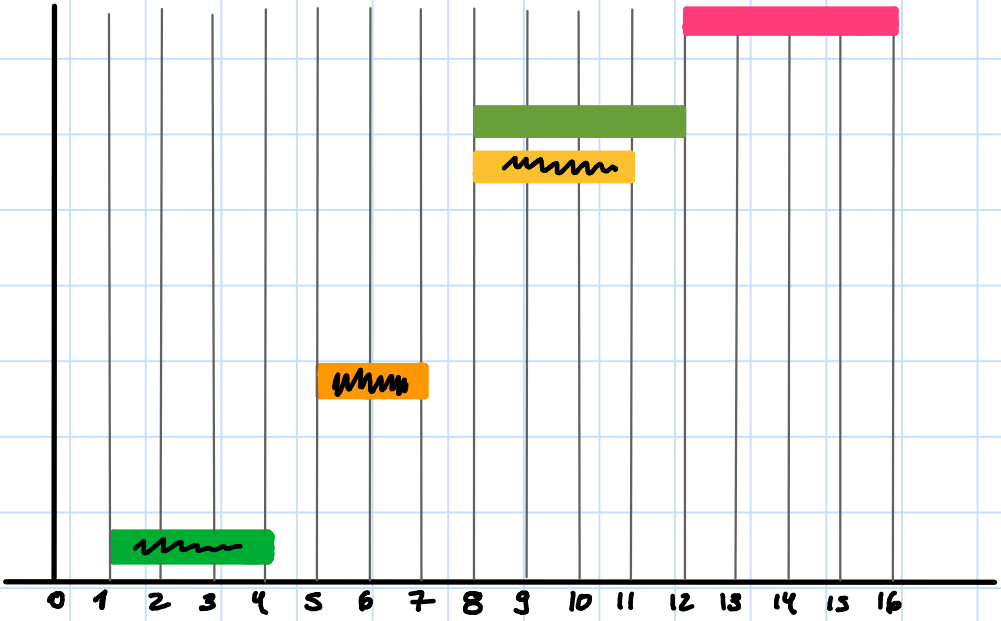
Pegamos "gulosamente" o "melhor" atividade entre as atividades restantes

Descobrindo um Algoritmo



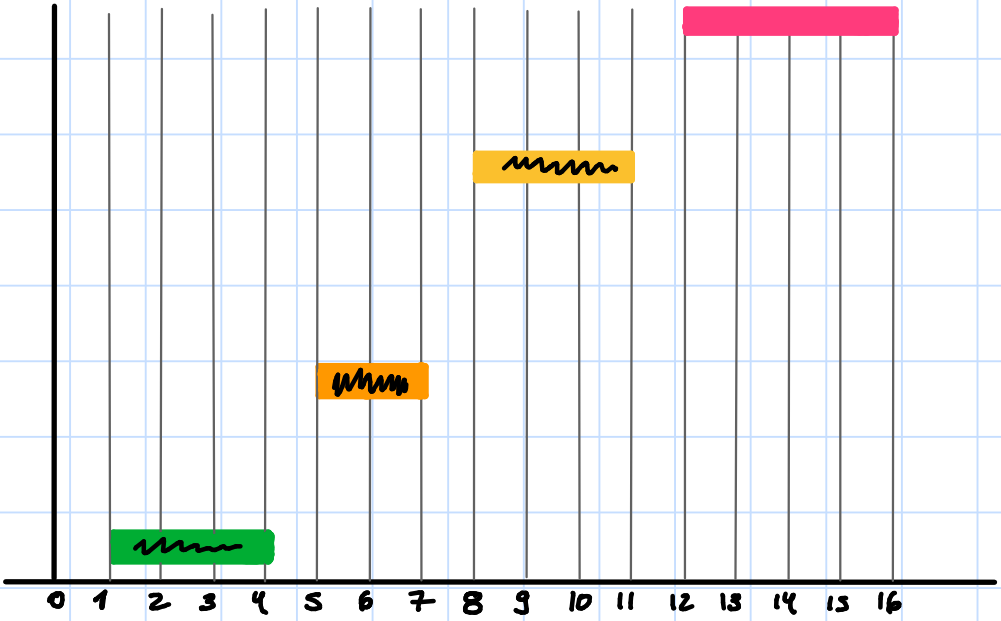
Pegamos "gulosamente" o "melhor" atividade entre as atividades restantes

Descobrindo um Algoritmo



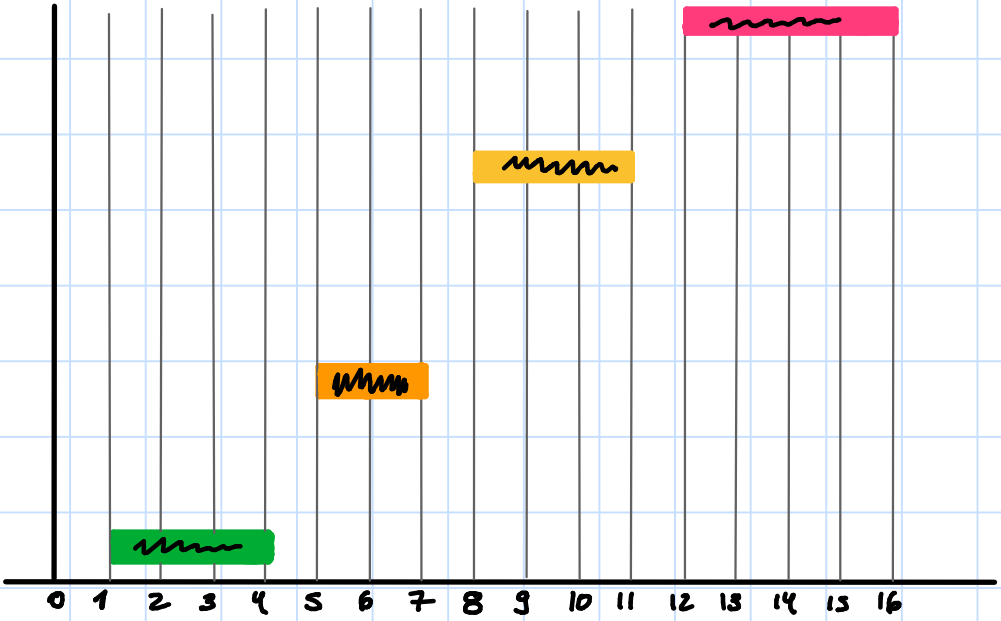
Pegamos "gulosamente" o "melhor" atividade entre as atividades restantes

Descobrindo um Algoritmo



Pegamos "gulosamente" o "melhor" atividade entre as atividades restantes

Descobrindo um Algoritmo



Pegamos "gulosamente" o "melhor" atividade entre as atividades restantes

escolha atividades entre $S_{k,n+1}$

Seleção-Atividade (s, f, k, n)

$m \leftarrow k+1$

Enquanto $m \leq n$ e $s[m] < f[k]$

$m \leftarrow m+1$

se $m \leq n$

retorna $\{a_m\} \cup$ Seleção-Atividade(s, f, m, n)

Retorna $\emptyset$

escolha atividades entre $S_{k,n+1}$

```
Seleção-Atividade (s, f, k, n)
    m ← k+1
    Enquanto  m ≤ n  e  s[m] < f[k]
        m ← m+1
    se m ≤ n
        retorna {a_m} ∪ Seleção-Atividade(s, f, m, n)
    Retorna ∅
```

Este algoritmo é o que chamamos de algoritmo guloso

- Defronte de um subproblema ($S_{k,n+1}$), ele toma a melhor decisão segundo um dado critério escolhendo um elemento para a solução (escolher a primeira tarefa válida a terminar), gerando um novo subproblema ($S_{m,n+1}$) que também é resolvido gulosamente

- Algoritmos gulosos nem sempre garantem encontrar a solução ótima, nesses casos são chamados de Heurísticas

↳ Heurística são úteis principalmente para lidar com problemas NP-difíceis

- Alguns algoritmos gulosos sempre encontram a solução ótima, o algoritmo anterior é um desses casos

— Vamos prover isso

- Vamos prover isso:
    - Existe uma solução ótima que contém a escolha gulosa
    - Subestrutura ótima / correção do algoritmo

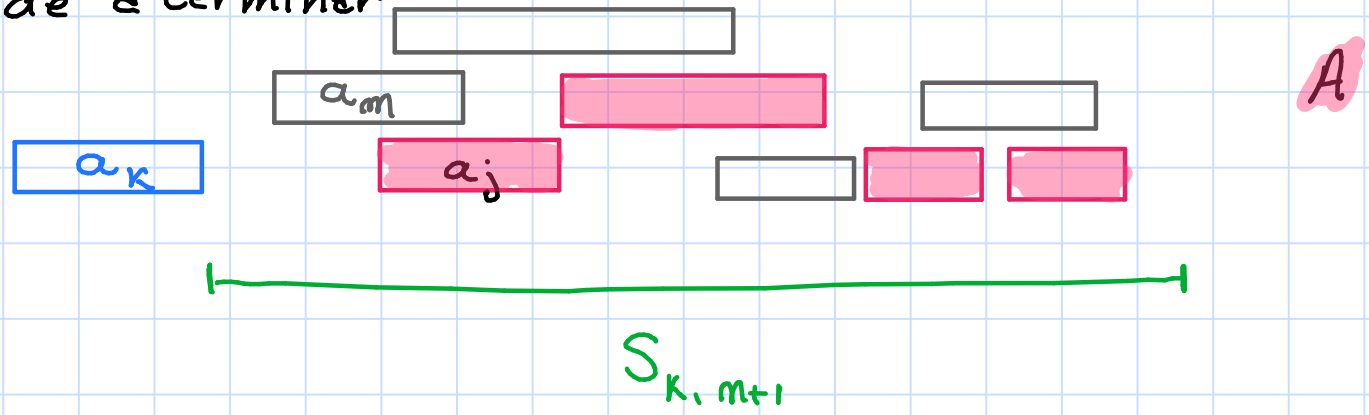
Lema Considere um subproblema $S_{k,m+1}$ não vazio, e seja $a_m \in S_{k,m+1}$ uma atividade com o menor tempo final. Então, existe uma solução ótima A de $S_{k,m+1}$ tal que $a_m \in A$.

Demonstração

- Seja A uma solução ótima de $S_{k,m+1}$, i.e.,
$$|A| = \psi_{k,m+1}$$
- Se $a_m \in A$, então o resultado segue.
- Então, suponha que $a_m \notin A$ e seja $a_j \in A$ com o menor tempo de fim

Demonstração

- Seja A uma solução ótima de $S_{k,m+1}$, i.e.,
$$|A| = \psi_{k,m+1}$$
- Se $a_m \in A$, então o resultado segue.
- Então, suponha que $a_m \notin A$ e seja $a_j \in A$ com o menor tempo de fim.
- Seja $A^* = (A \setminus \{a_j\}) \cup \{a_m\}$ e note que A^* é uma solução viável, pois A era viável e a_j é a primeira atividade a terminar



Demonstração

- Seja A uma solução ótima de $S_{k,m+1}$, i.e.,
$$|A| = \psi_{k,m+1}$$
- Se $a_m \in A$, então o resultado segue.
- Então, suponha que $a_m \notin A$ e seja $a_j \in A$ com o menor tempo de fim.
- seja $A^* = (A \setminus \{a_j\}) \cup \{a_m\}$ e note que A^* é uma solução viável, pois A era viável e a_j é a primeira atividade a terminar
- Portanto
$$|A^*| = |A| = \psi_{k,m+1}$$

□

- Seja $S = \{a_1, a_2, \ldots, a_n\}$ (tarefas ordenadas por tempo final)
  - Pela subestrutura ótima do problema, sabemos que

$$\psi_{0,n+1} = \psi_{0,k} + \psi_{k,n+1} + 1$$

  - Pelo lema anterior, sabemos que existe uma solução ótima $A \subseteq S$ tal que $a_1 \in A$ ($k = 1$)
    - Então

$$\psi_{0,n+1} = \cancel{\psi_{0,1}}^{\emptyset} + \psi_{1,n+1} + 1$$

$$= \psi_{1,n+1} + 1$$

← Apenas um subprob.

$$\psi_{0,n+1} = \psi_{1,n+1} + 1$$

① por indução o nosso algoritmo retorna uma solução A' do prob. $S_{1,n+1}$ tal que $|A'| = \psi_{1,n+1}$

② Então nosso algoritmo faz $B = A' \cup \{a_1\}$, que é viável

③ Ademais

$$|B| = |A'| + 1 = \psi_{1,n+1} + 1 = \psi_{0,n+1}$$

Então B é uma solução ótima.

Teo Seleção-Atividade (s, f, k, n) retorna um conj. de atividades $A \subseteq S_{k,n+1}$ tal que $|A| = \psi_{k,n+1}$.

Demonstração

A prova segue por indução em $\ell = |S_{k,n+1}|$

Base ($\ell = 0$): neste caso o laço da linha 2 não encontra nenhuma atividade m e, consequentemente, encerra com $m = n+1$. O teste da linha 4 falha e o algoritmo retorna $\emptyset$, corretamente

Passo ($\ell > 0$): neste caso o laço da linha 2 termina com $m \leq n$. Assim o teste da linha 4 dá verdadeiro e a linha 5 executa. Note que $|S_{m,n+1}| < |S_{k,n+1}|$ e, por hipótese de indução, o Seleção-Atividade (s, f, m, n) retorna uma solução A' tal que $|A'| = \psi_{m,n+1}$

Passo ($\ell > 0$): neste caso o laço da linha 2 termina com $m \leq n$. Assim o teste da linha 4 da verdadeiro e a linha 5 executa. Note que $|S_{m,n+1}| < |S_{k,n+1}|$ e, por hipótese de indução, o Seleção-Atividade(s, f, m, n) retorna uma solução A' tal que $|A'| = \psi_{m,n+1}$.

Então o algoritmo faz $B = \{a_m\} \cup A'$. É fácil perceber que B é uma solução viável. Assim

$$|B| = |A'| + 1 = \psi_{m,n+1} + 1$$

Pela subestrutura ótima do problema, sabemos que

$$\psi_{k,n+1} = \psi_{m,n+1} + 1$$

Assim, $|B| = \psi_{k,n+1}$ □

Complexidade

- Ao longo de todas as chamadas recursivas, temos que cada atividade é examinada apenas uma vez. Portanto o algoritmo é $\Theta(n)$

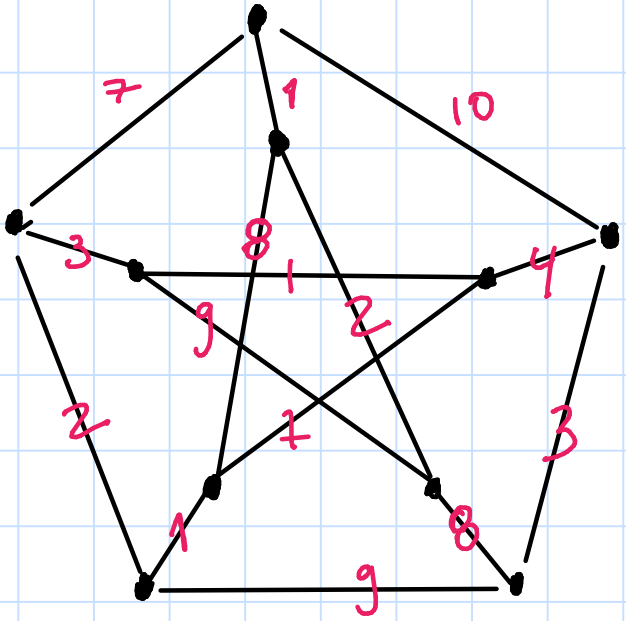
assumindo que $f_1 \leq f_2 \leq \cdots \leq f_n$

```
Seleção-Atividade2 (s, f)
    n ← s.comprimento
    A ← {a_1}
    k ← 1
    Para m ← 2 até n
        Se s[m] ≥ f[k]
            A ← A ∪ {a_m}
            k ← m
    Retorna A
```

Árvore Geradora Mínima

Grafo Ponderado

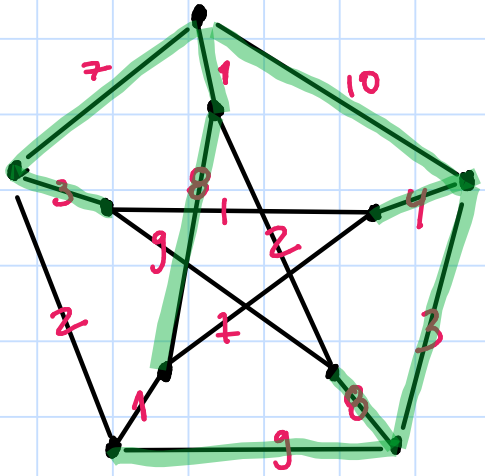
Um grafo ponderado é um grafo G e um função de custo $w : E(G) \to \mathbb{R}$.



Notação

- Dado um grafo G ponderado por uma função $w: E(G) \to \mathbb{R}$ e um subgrafo $H \subseteq G$, o peso (custo) de H, denotado por $w(H)$, é definido como

$$w(H) = \sum_{e \in E(H)} w(e)$$



$H \subseteq G$

$w(H) = 53$

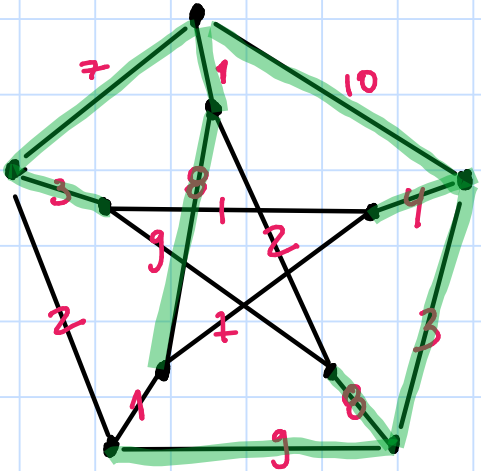
Árvore Geradora Mínima (AGM)

Problema da Árvore Geradora Mínima

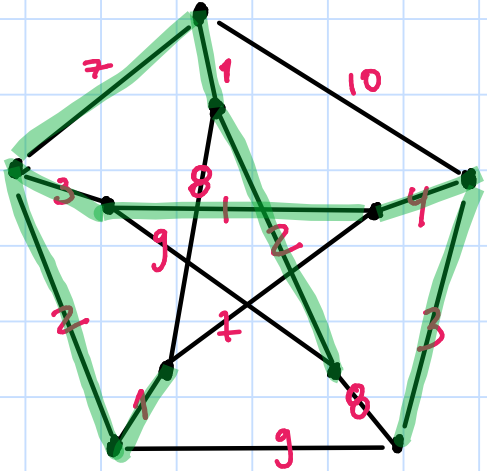
Entrada: Um grafo conexo G ponderado por uma função $w: E(G) \to \mathbb{R}^+$.

Saída: uma árvore geradora $T \subseteq G$ tal que $w(T) = \min \{ w(T') : T' \subseteq G \text{ é um AGM} \}$.

Exemplo

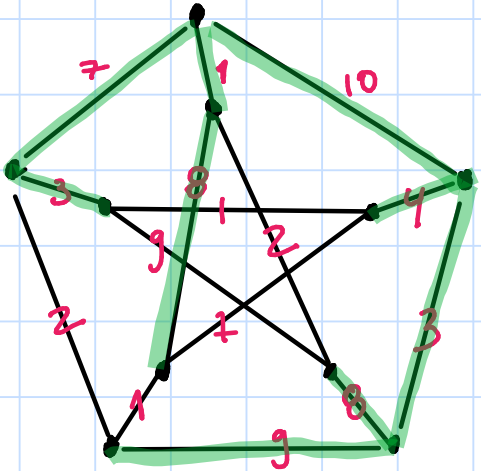


$w(H) = 53$



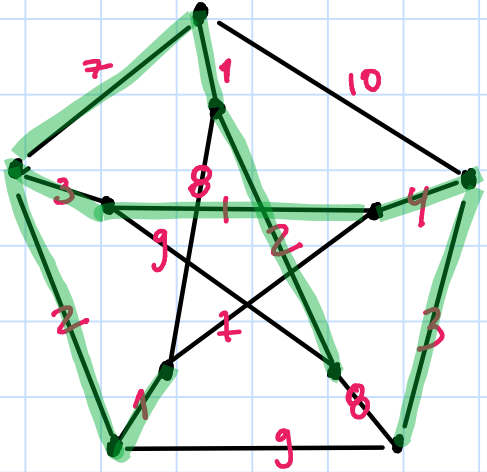
$w(H) = 24$

Exemplo



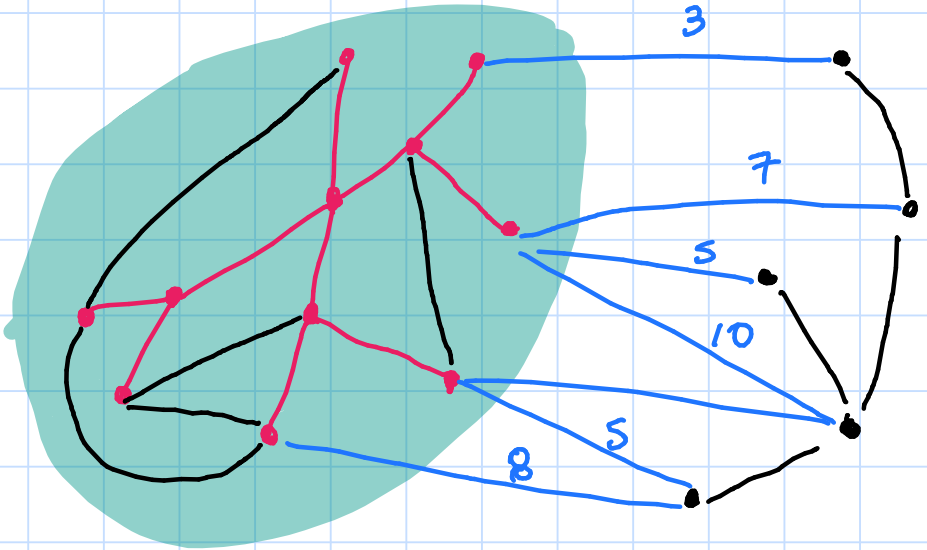
$w(H) = 53$

solução ótima

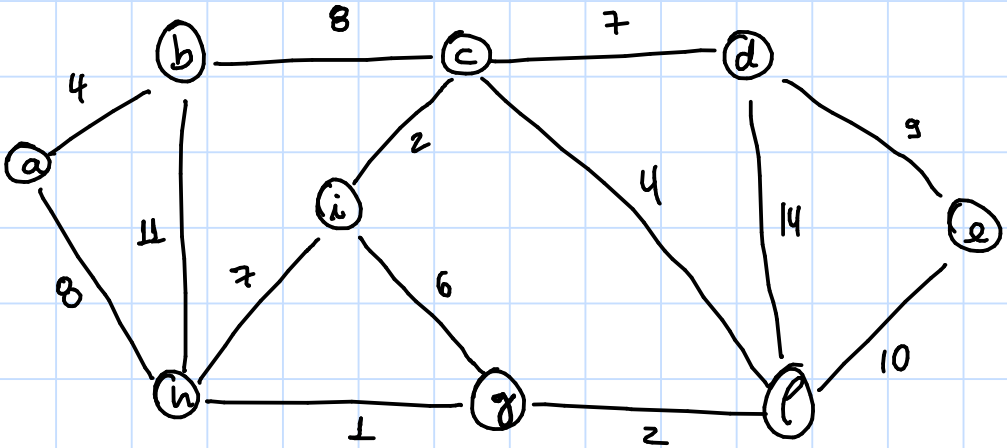


$w(H) = 24$

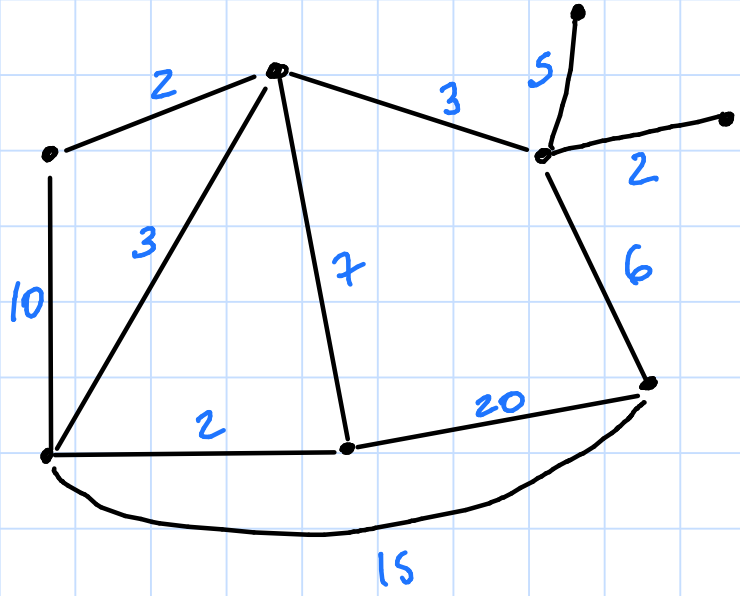
Intuição



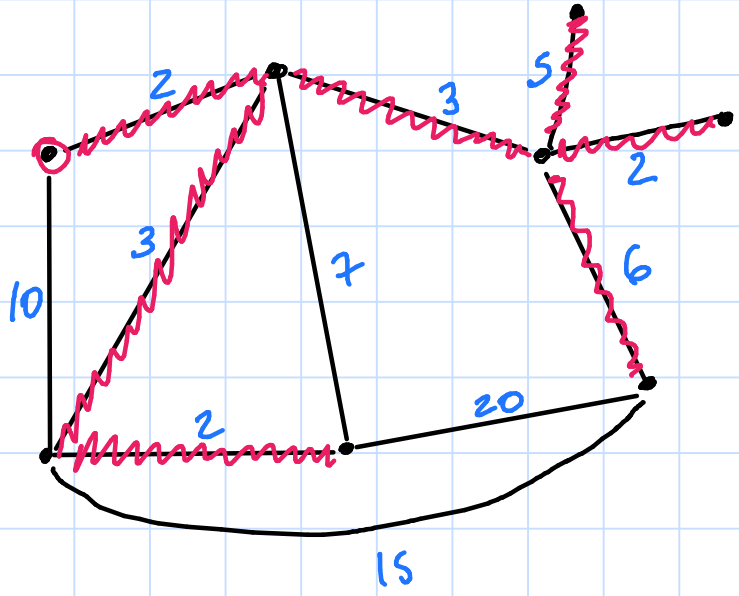
Exemplo



Exemplo



Exemplo



$w(T) = 23$

Algoritmo Prim (G, w)

1 Seja $s \in V(G)$

2 Seja $T \leftarrow (\{s\}, \emptyset)$

3 Enquanto $E(T) \neq \emptyset$

4 seja $uv \in E(T)$ tal que

$$w(uv) = \min \{ w(xy) : xy \in E(T) \}$$

5 $T \leftarrow T \cup \{uv\}$

6 Retorna $w(T)$, T

- Dado um grafo G, seja ψ_G o peso de uma AGM

Teorema O Algoritmo de Prim resolve o prob. da AGM.

Demonstração

O laço da linha 3 mantém a seguinte invariante

$p(\ell)$ = "antes da ℓ-ésima iteração começar, T está contido dentro de uma AGM de G"

Base ($\ell = 1$): Quando $\ell = 1$, temos que $T = (\{1\}, \emptyset)$ e o resultado segue

Passo ($\ell > 1$): Por H.I. $T \subseteq T^*$ está contida em uma AGM T^* de G. Seja uv a aresta escolhida pelo algoritmo na linha 4. Seja $H = T \cup uv$. Na linha 5, o Algoritmo faz $T \leftarrow H$. Assim, se $H \subseteq T^*$, o resultado segue. Portanto, suponha que $H \not\subseteq T^*$, o que implica que $uv \notin E(T^*)$. Seja P o único caminho de u à v em T^*. Seja $W = T^* \cup uv$ e note que W contém exatamente um ciclo, o ciclo $P \cup uv$. Note que $E(T) \cap E(P) \neq \emptyset$.

Seja $xy \in E(T) \cap E(P)$ e seja $W^* = W - xy$. Como xy pertence ao ciclo $Puxy$ de W, temos que W^* é uma árvore geradora. Ademais

$$\psi_G \leq w(W^*) = w(W) - c(xy) = w(T^*) + w(uv) - w(xy)$$
$$= \psi_G + w(uv) - w(xy)$$

Pela escolha de uv, sabemos que $w(uv) \leq w(xy)$, portanto

$$w(uv) - w(xy) \leq 0$$

Assim, concluímos que $w(W^*) = \psi_G$.
Como $H \subseteq W^*$, temos que o resultado segue.
Isso finaliza a prova da invariante!

- Quando o laço da linha 3 terminar, temos que

$E(T) = \emptyset$ e como o grafo é conexo, temos que T é uma árvore geradora. Pela invariante, T está contido dentro de uma AGM de G, neste caso T é a própria AGM □

Subestrutura Ótima

Lema Seja $T \subseteq G$ uma AGM de G e seja $uv \in E(T)$. Seja T_1 e T_2 as duas árvores de $T - uv$. Então $w(T_1) = \psi_{G[V(T_1)]}$ e $w(T_2) = \psi_{G[V(T_2)]}$.

Demonstração

- Note que $\psi_G = w(T) = w(T_1) + w(T_2) + w(uv)$
- Suponha para uma contradição que $w(T_1) > \psi_{G[V(T_1)]}$
- Seja $H \subseteq G[V(T_1)]$ uma AGM. Logo $w(H) < w(T_1)$
- Seja $H^* = H \cup T_2 \cup uv$ e note que H^* é uma árvore geradora de G.
- Note também que

$$\psi_G \leq w(H^*) = w(H) + w(T_2) + w(uv) < w(T_1) + w(T_2) + w(uv) = w(T) = \psi_G,$$

uma contradição. □

Subestrutura Ótima

- Seja T uma AGM de G e $xy \in E(T)$
- Sejam T_1 e T_2 as árvores de $T - xy$

$$\psi_G = \psi_{G[V(T_1)]} + \psi_{G[V(T_2)]} + c(xy)$$

não sabemos quem são T_1 e T_2

não sabemos quem é xy

Escolha gulosa pertence Solução Ótima

- Seja $X, Y \subseteq V(G)$ tal que $X \cap Y = \emptyset$ e $X \cup Y = V(G)$
- Seja $xy \in E(X)$ tal que
$$w(xy) = \min\{ w(uv) : uv \in E(X) \}$$
- Então, existe uma AGM T tal que $xy \in E(T)$
  - Seja T uma AGM de G
  - Se $xy \in T$, acabou
  - Então, suponha que $xy \notin T$. Seja P o caminho de x a y em T. Existe uma aresta $uv \in E(P) \cap E(X)$
  - Seja $T^+ = T \cup xy$ e note que $P \cup xy$ é um ciclo
  - $T^* = T^+ - uv$ é uma árvore geradora e

Escolha gulosa pertence Solução Ótima

$$\psi_G \leq w(T^*) = w(T^+) - w(uv) = w(T) + w(xy) - w(uv)$$
$$= \psi_G + \underbrace{w(xy) - w(uv)}_{\leq 0}$$

- Então $w(xy) = w(uv) \Rightarrow T^*$ é AGM e $xy \in E(T^*)$!

Subestrutura Ótima

- Seja T uma AGM de G e $xy \in E(T)$
- Sejam T_1 e T_2 as árvores de $T - xy$

$$\psi_G = \psi_{G[V(T_1)]} + \psi_{G[V(T_2)]} + c(xy)$$

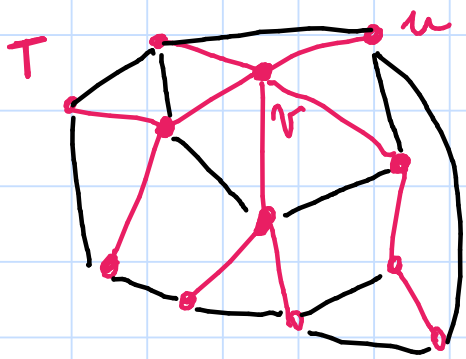
não sabemos quem são T_1 e T_2

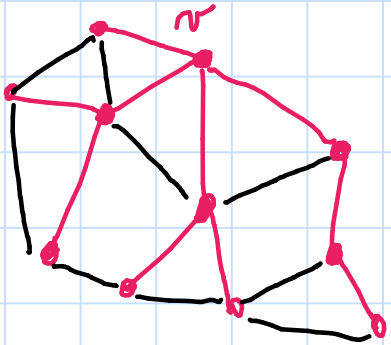
não sabemos quem é xy

podemos escolher aresta de menor custo em $E(T_1)$

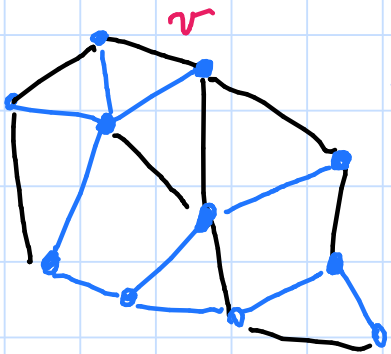
Subestrutura Ótima

Lema Seja $T \subseteq G$ uma AGM e seja u uma folha de T. Então, $T - u$ é uma AGM de $G - u$.



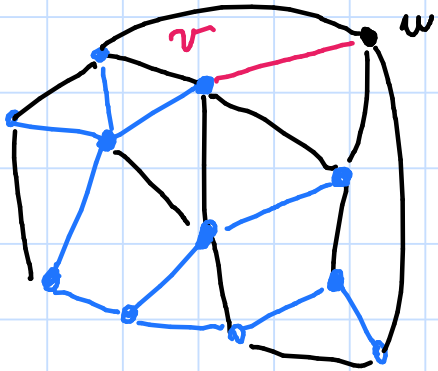


$w(T - u)$



Suponha p/ contradição

$w(T') < w(T - u)$

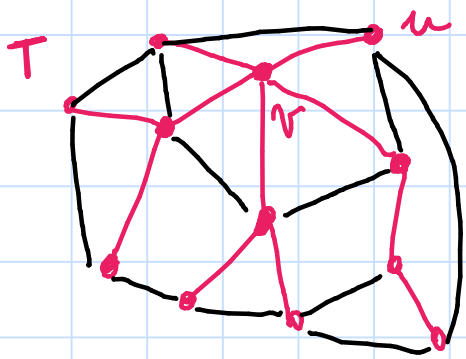


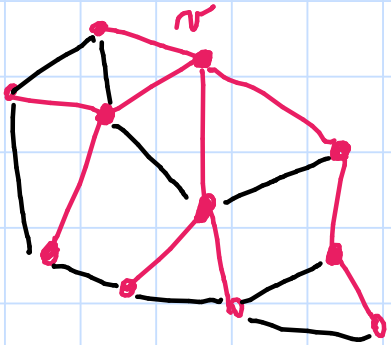
$T^* = T' \cup \{uv\}$

$$w(T^*) = w(T') + c(uv) < c(T - u) + c(uv) = c(T)$$

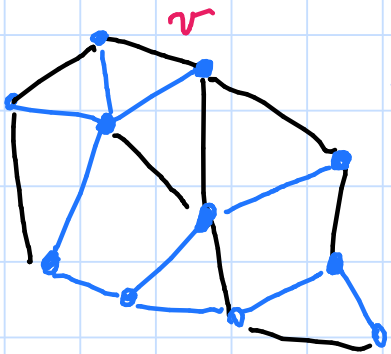
Subestrutura Ótima

Lema Seja $T \subseteq G$ uma AGM e seja u uma folha de T. Então, $T - u$ é uma AGM de $G - u$.



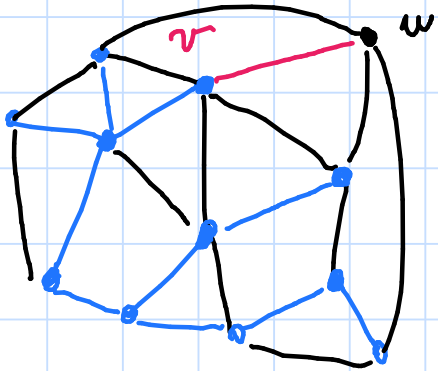


$w(T - u)$



Suponha p/ contradição

$w(T') < w(T - u)$



$T^* = T' \cup \{uv\}$

$$w(T^*) = w(T') + c(uv) < c(T - u) + c(uv) = c(T)$$

Subestrutura Ótima

Lema Seja $T \subseteq G$ uma AGM e seja u uma folha de T. Então, $T - u$ é uma AGM de $G - u$.

Demonstração

- Suponha para uma contradição que $T-u$ não é uma AGM de $G-u$ e seja $uv \in E(T)$.
- Note que $w(T) = w(T-u) + w(uv)$
- Seja T' um AGM de $G-u$. Portanto $w(T') < w(T)$
- Seja $T^* = T' \cup uv$ e note que T^* é uma árvore.
- Note tbm que

$$w(T^*) = w(T') + w(uv) < w(T-u) + w(uv) = w(T),$$

o que é um absurdo.

□

Seja T uma AGM de G, seja u uma folha de T e seja $uv \in E(T)$

$$\psi_G = \psi_{G-u} + w(uv)$$

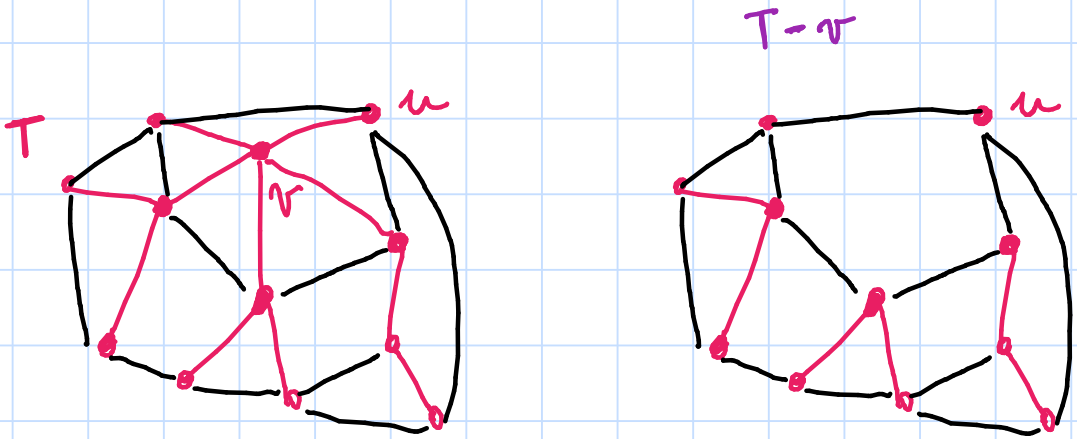
Seja T uma AGM de G, seja u uma folha de T e seja $uv \in E(T)$

$$\psi_G = \psi_{G-u} + w(uv)$$

↑

conseguimos garantir pela escolha gulosa

- Note que foi muito importante escolher uma folha para demonstrar a subestrutura ótima



Implementação do Algoritmo de Prim

- Vamos manter as seguintes informações durante a execução do algoritmo
  - * todos os vértices que não estão na árvore estão em uma fila de prioridades (de mínima) Q
  - * um vetor Key[ ], indexado pelos vértices, armazena o menor peso de conexão de um vértice u a árvore que está sendo construída pelo algoritmo
    - Key[u] = k ⇔ existe uma aresta de peso k que conecta u a árvore em construção
    - Se nenhuma aresta conecta u a árvore que está sendo construída, então Key[u] = ∞

Implementação do Algoritmo de Prim

- O vetor pred[·], indexado pelos vértices, armazena a árvore, enraizada em um vértice arbitrário, que está sendo construída pelo algoritmo.

$$E(T) = \{ \{u, pred[u]\} : u \in V(T) \text{ e } pred[u] \neq u \}$$

Algoritmo de Prim

```
PRIM (G, w)
1     para cada u ∈ V(G) faça
2           Key[u] = ∞
3           pred[u] = NULL
4     pred[0] = 0          ▷ escolhendo o vértice 0
5     Key[0] = 0             como raiz
6     Q = V(G)             ▷ criando a fila com todos
                             os vértices de G.
7     Enquanto Q ≠ ∅ faça
8           u ← Extract-Min (Q)
9           para cada v ∈ N_G(u)
10                Se v ∈ Q e w(uv) < Key[v] então
11                      pred[v] = u
12                      Key[v] = w(uv)
```

Complexidade

- O corpo do laço gasta $\Theta(1)$ e é executado $|V(G)|$ vezes. Portanto, as linhas 1-3 tomam tempo $\Theta(V)$
- As linhas 4-5 levam tempo $\Theta(1)$
- Podemos implementar a fila como uma Heap Mínima, portanto, a construção da fila de prioridades da linha 6 pode ser feita em $\Theta(V)$.
- O laço da linha 7 é executado $|V(G)|$ vezes, já que a fila começa com $|V(G)|$ elementos, cada iteração remove um elemento da fila, e nenhum elemento é inserido.
- A linha 7 leva $\Theta(1)$ para executar e é executada $|V(G)|$ vezes. Portanto, gastamos $\Theta(V)$ com a linha 7

Complexidade

- Em uma Heap, a operação Extract-Min leva tempo $O(\lg n)$ em uma Heap com n elementos. Assim, gastamos $O(V \lg V)$ com a linha 8
- Ao longo de toda a execução do Algoritmo, o laço da linha 9 executa $O(E)$ vezes, assumindo que o grafo está implementado como uma lista de adjacências
- Assim, as linhas 9-11 vão gastar $O(E)$ ao longo da execução
- A linha 12 esconde uma chamada implícita à HeapCorrigeSubindo, que leva $O(\lg n)$ em uma Heap com n elementos. Portanto, a linha 12 leva $O(E \lg V)$
- Portanto, o algoritmo de Prim leva $O(V \lg V + E \lg V) = O(E \cdot \lg V)$