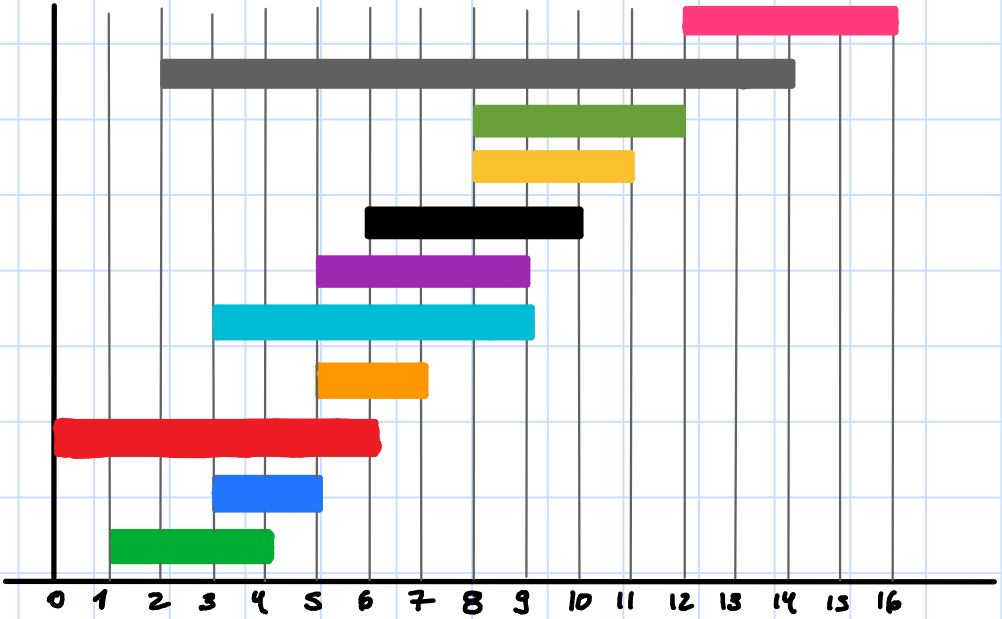
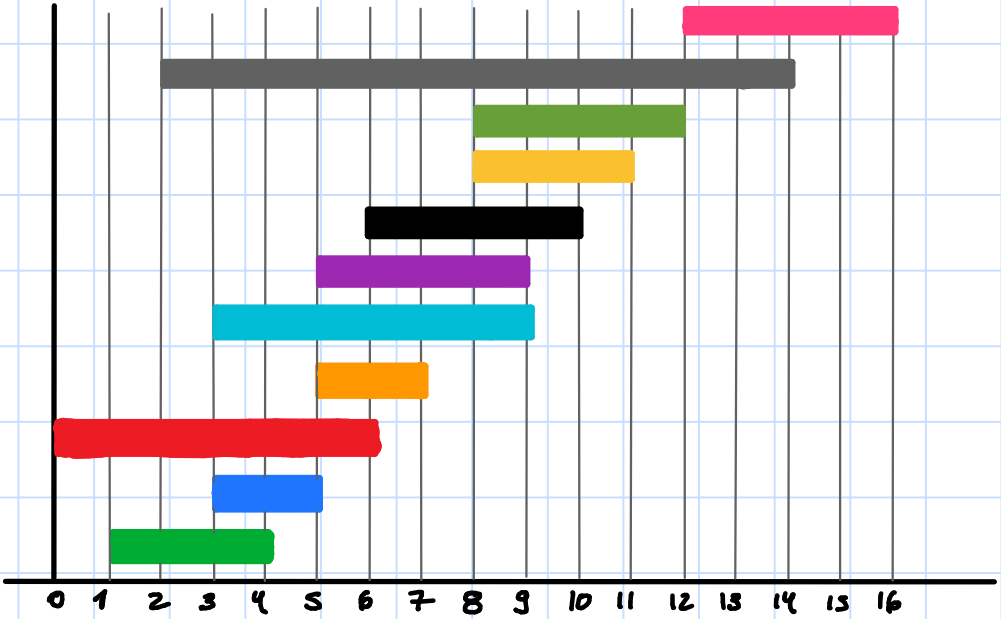
Algoritmos Gulosos

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 $[s_i, f_i)$ $\in \mathbb{N}^2$

Problema da Seleção de Atividades

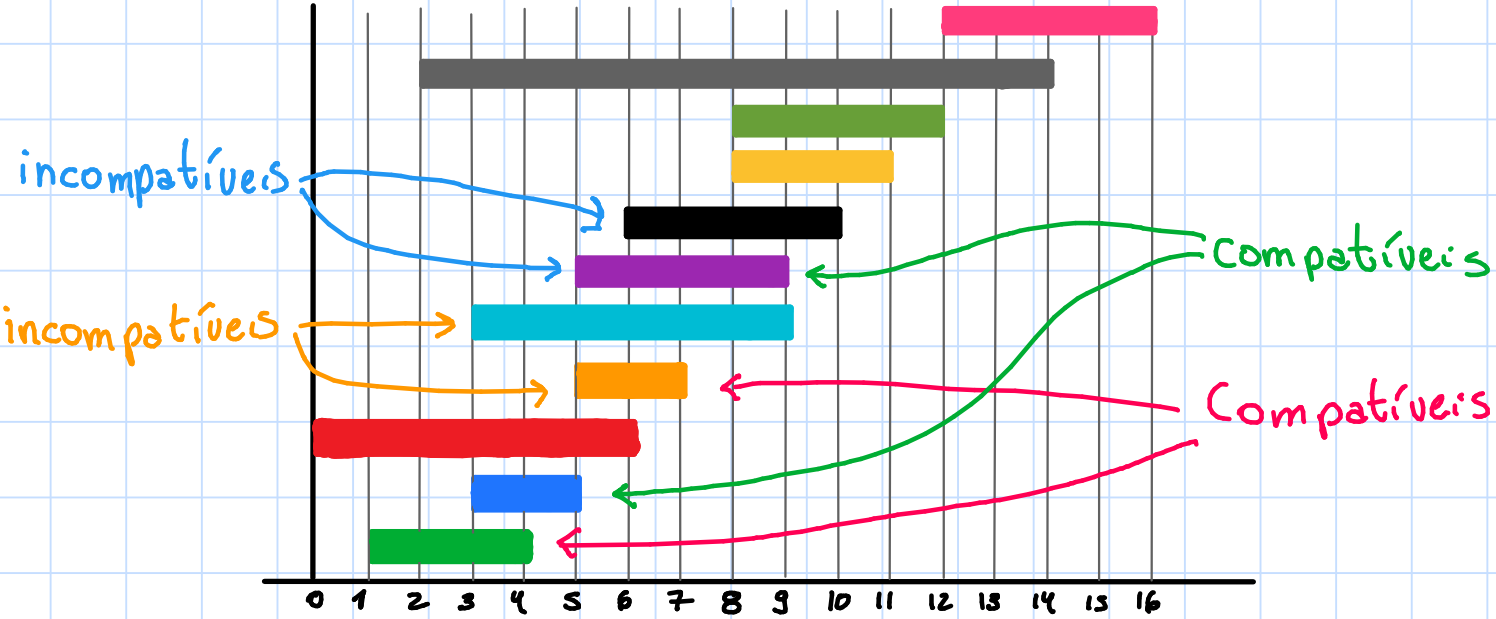


- As atividades competem por um recurso exclusivo
- Dizemos que as atividades a_i e a_j são <u>compatíveis</u> se

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

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

Problema da Seleção de Atividades

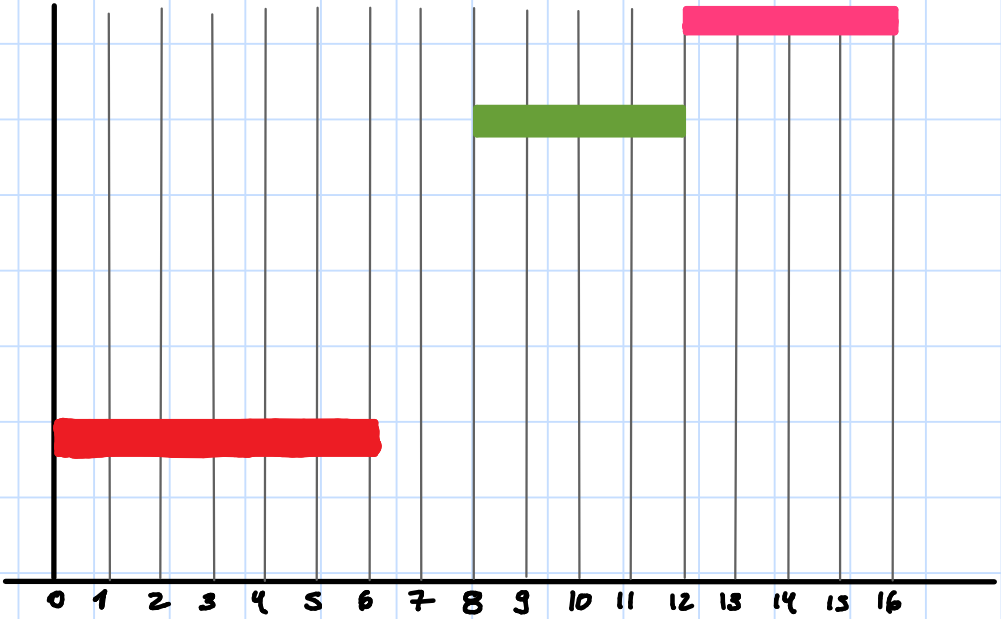


- As atividades competem por um recurso exclusivo
- Dizemos que as atividades a_i e a_j são <u>compatíveis</u> se

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

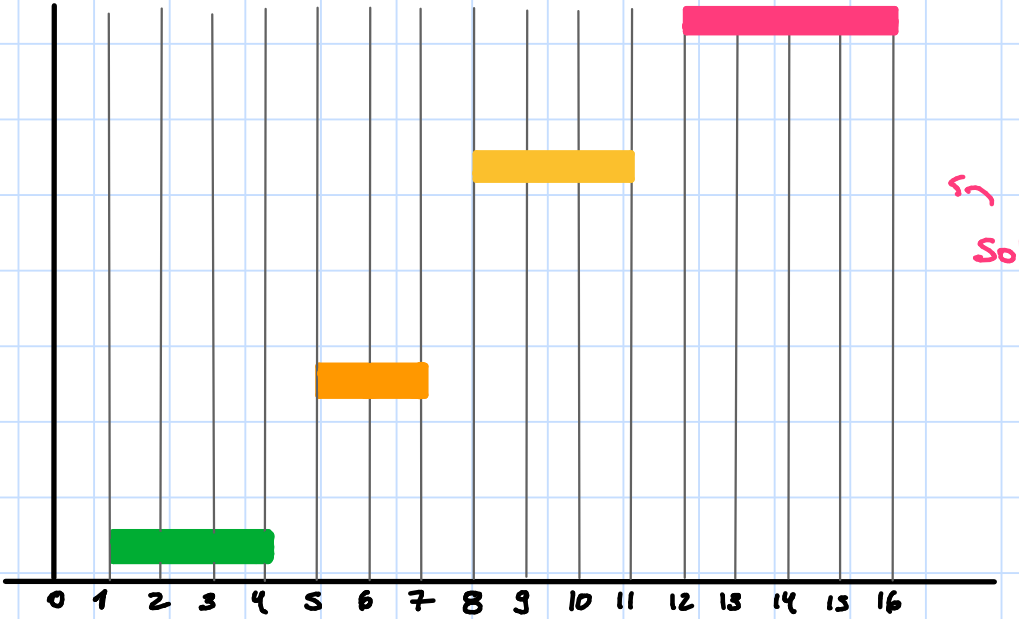
Caso contrário, dizemos que são <u>incompatíveis</u>

Problema da Seleção de Atividades



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

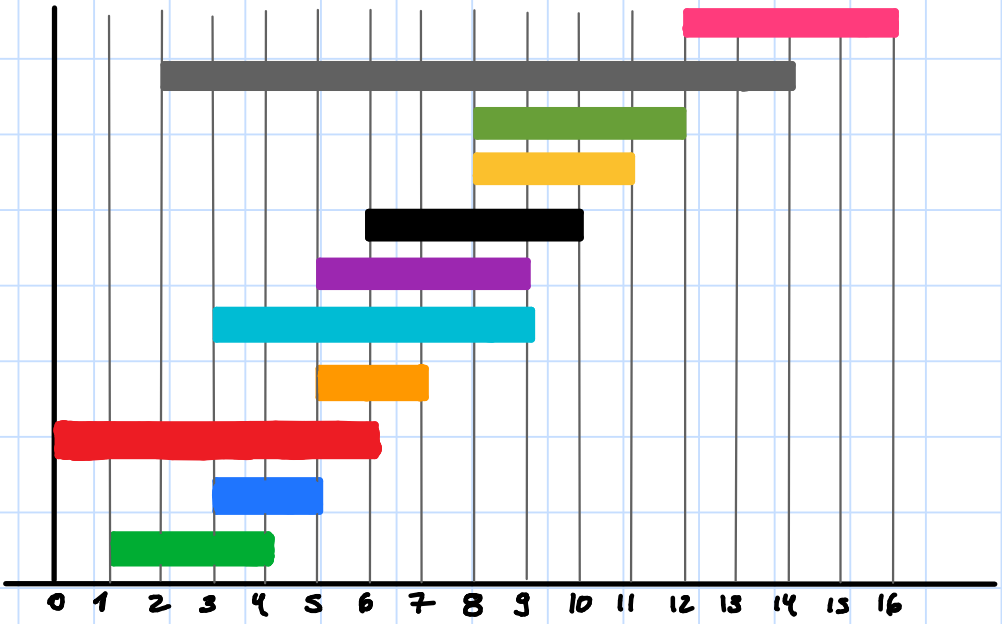
Problema da Seleção de Atividades



↳ 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 $[s_i, t_i)$, onde $s_i, t_i \in \mathbb{N} \cup \{0\}$

Saída: Um subconjunto $S^* \subseteq S$ de atividades 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 $[s_i, t_i)$

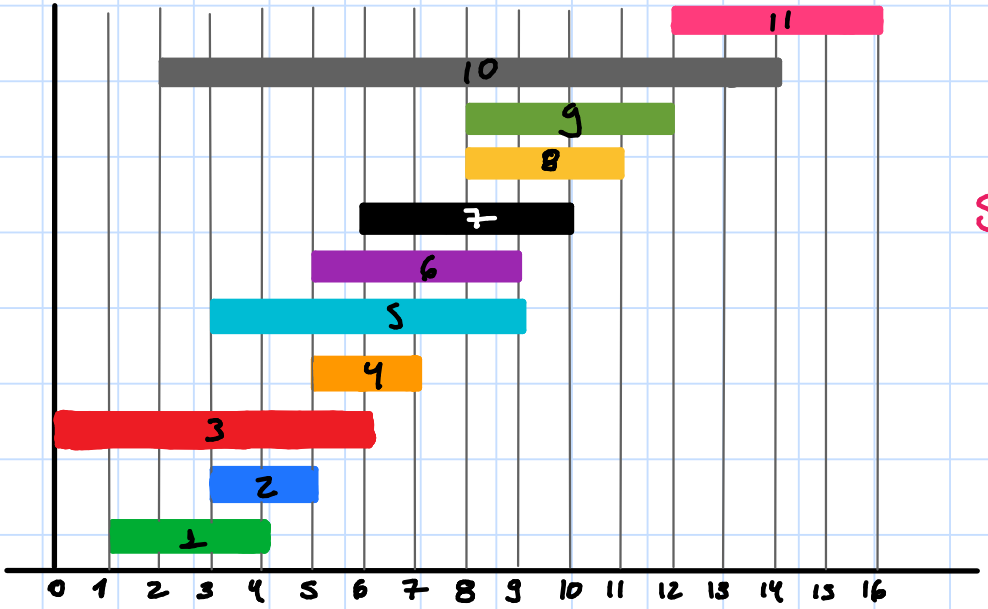
Saída: Um subconjunto $S^* \subseteq S$ de atividades compatíveis de cardinalidade máxima

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

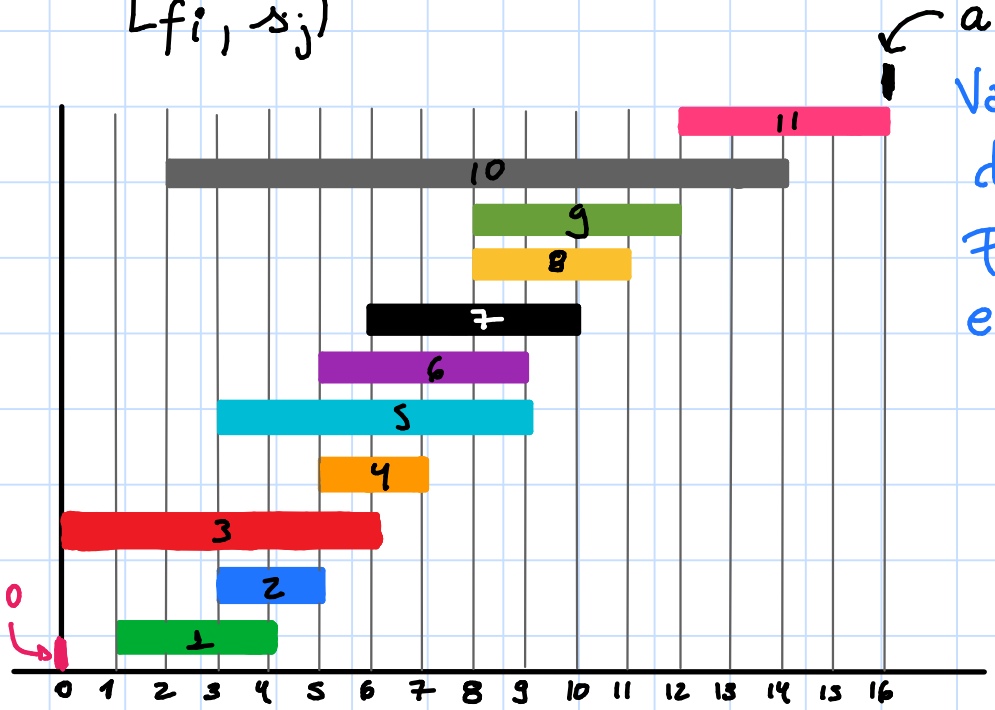
- Vamos denotar por S_{ij} o conjunto das atividades cujo tempo de realização se encontra no intervalo $[f_i, s_j)$



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

Notação

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



Vamos assumir que duas tarefas postiças foram adicionadas à entrada:

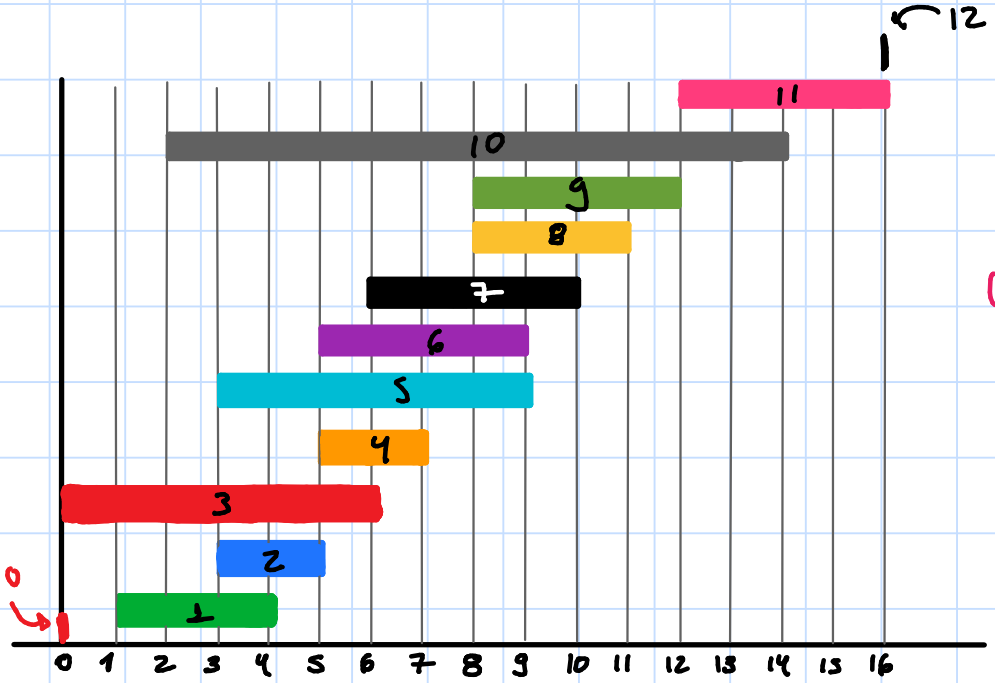
$a_0 = [0, 0)$ — s_0 f_0

$a_{n+1} = [f_n, f_n)$ — s_{n+1} f_{n+1}

Ex: $S_{0,n+1}$ = tarefas originais

Notação

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



$\psi_{0,12} = 4$

Problema da Seleção de Atividades

Entrada: Uma conjunto $S = \{a_1, a_2, \dots, a_n\}$ de atividades tal que a atividade a_i acontece no intervalo de tempo $[s_i, t_i)$ e $1 \leq i, j \leq n$

Saída: Um subconjunto $S^* \subseteq S_{i,j}$ de atividades compatíveis de cardinalidade máxima

↑ maior # de atividades compatíveis que começam depois da atividade i e antes j

$S = \{a_2, a_3, \dots, a_{n-1}\}$

$S' = S \cup \{a_1, a_n\} = \{a_1, a_2, \dots, a_n\}$

↑ FAKE

Solução pl $S'_{1,n}$ é sol. pl o prob. original com S.

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

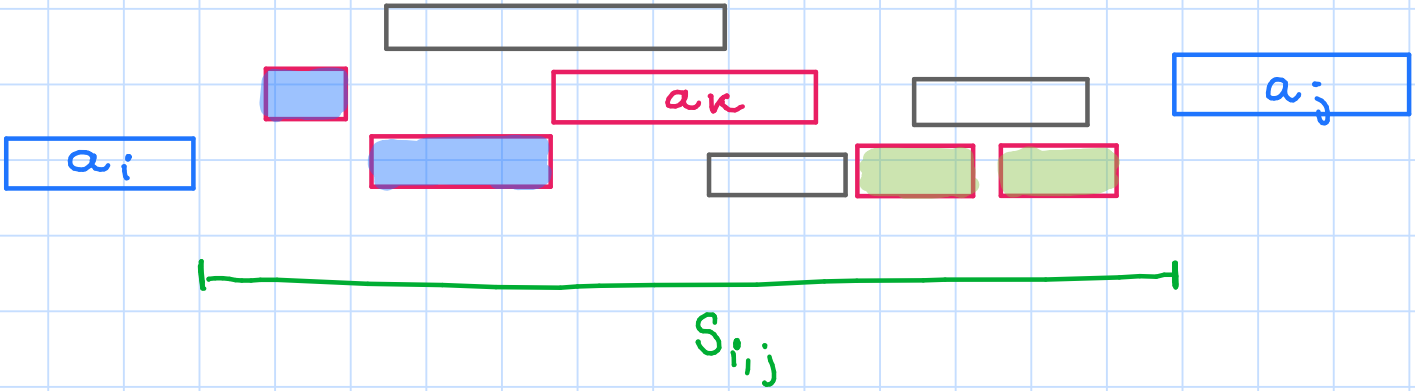
$$t_1 \leq t_2 \leq t_3 \leq \dots \leq t_n$$

Subestrutura Ótima

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

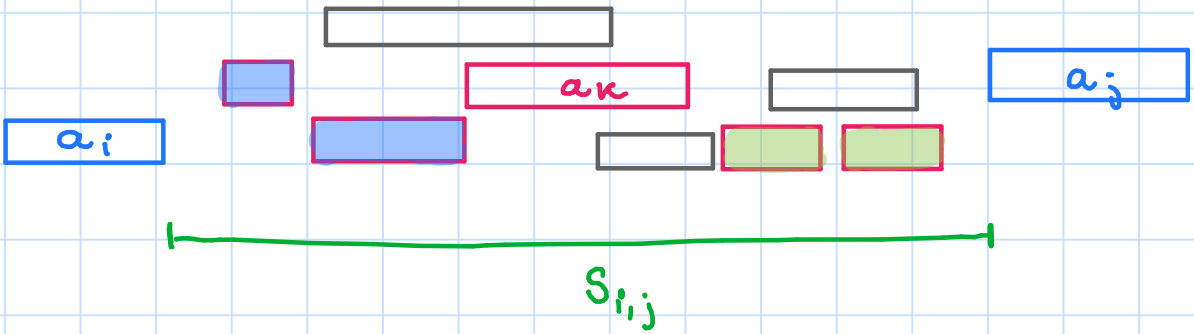
$$|A| = \psi_{i,j}$$

- 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 a_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

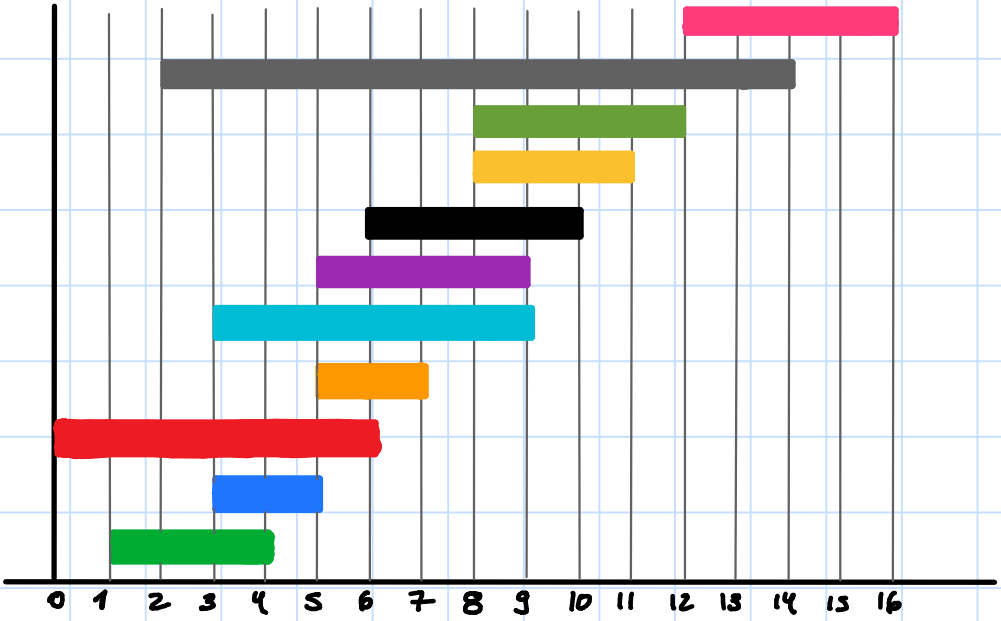
Descobrindo um Algoritmo

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

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

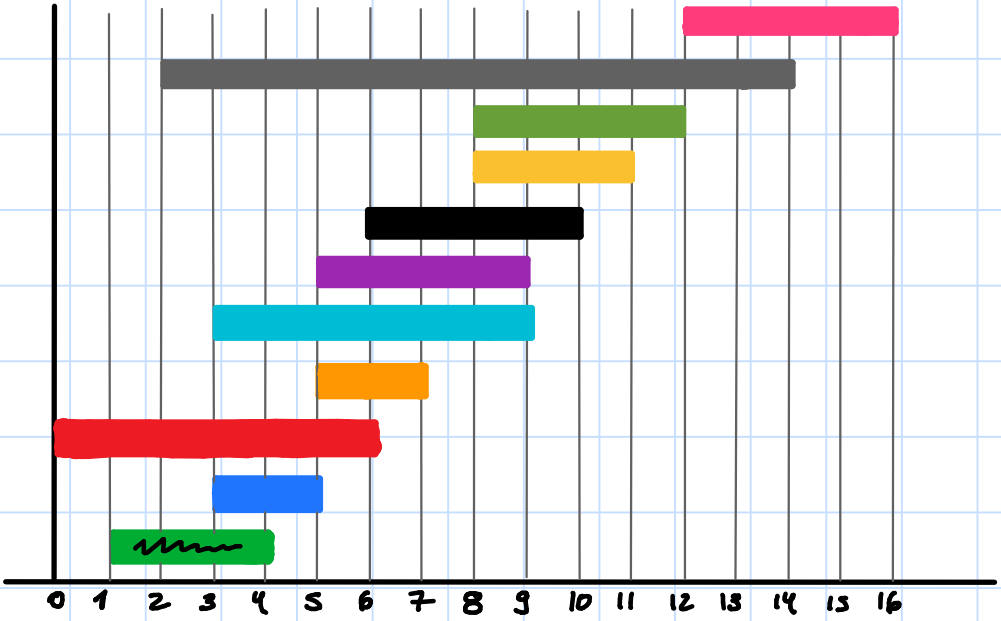
  - Usar PD nesse problema é e não gera o algoritmo mais eficiente
    - E vai desperdiçar memória

Descobrindo um Algoritmo



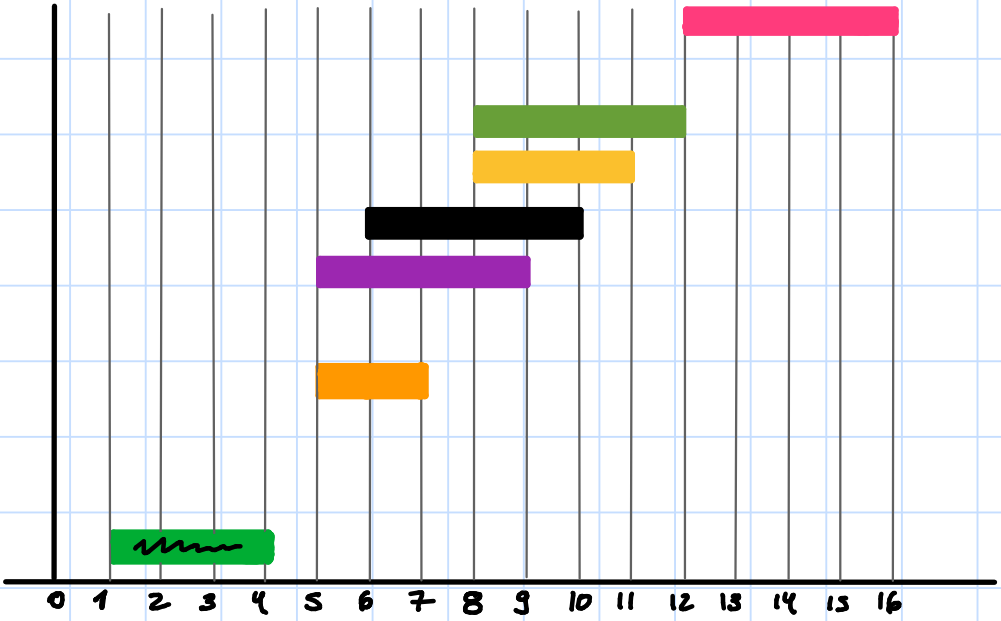
intuitivamente, para maximizar o número de tarefas escolhemos uma tarefa que termine o mais rápido possível, assim sobra mais espaço para colocar as outras.

Descobrindo um Algoritmo



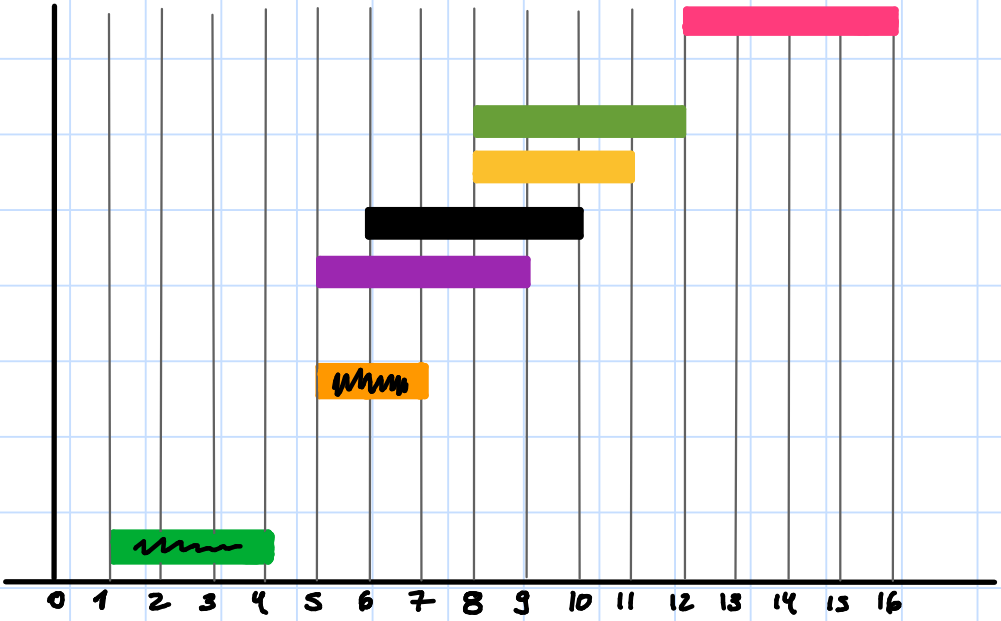
Pegamos "gulosamente" o "melhor" atividade entre as atividades restantes

Descobrindo um Algoritmo



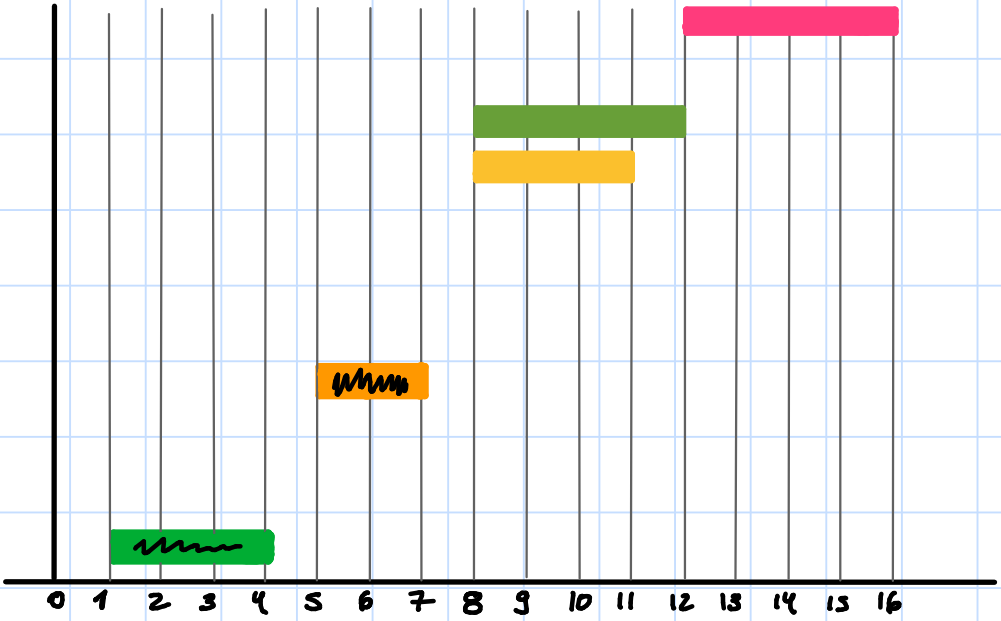
Pegamos "gulosamente" o "melhor" atividade entre as atividades restantes

Descobrindo um Algoritmo



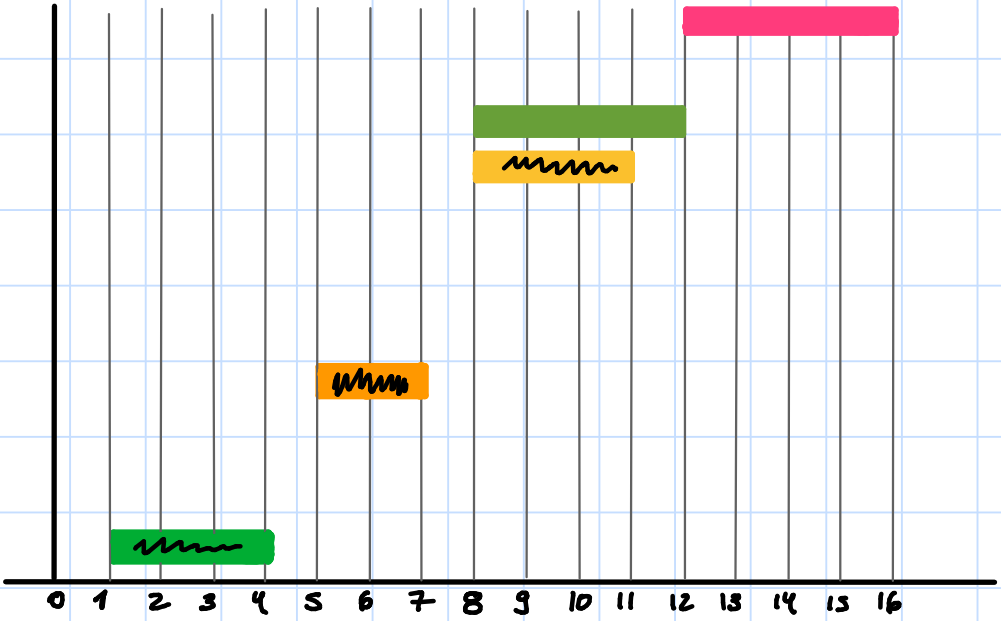
Pegamos "gulosamente" o "melhor" atividade entre as atividades restantes

Descobrindo um Algoritmo



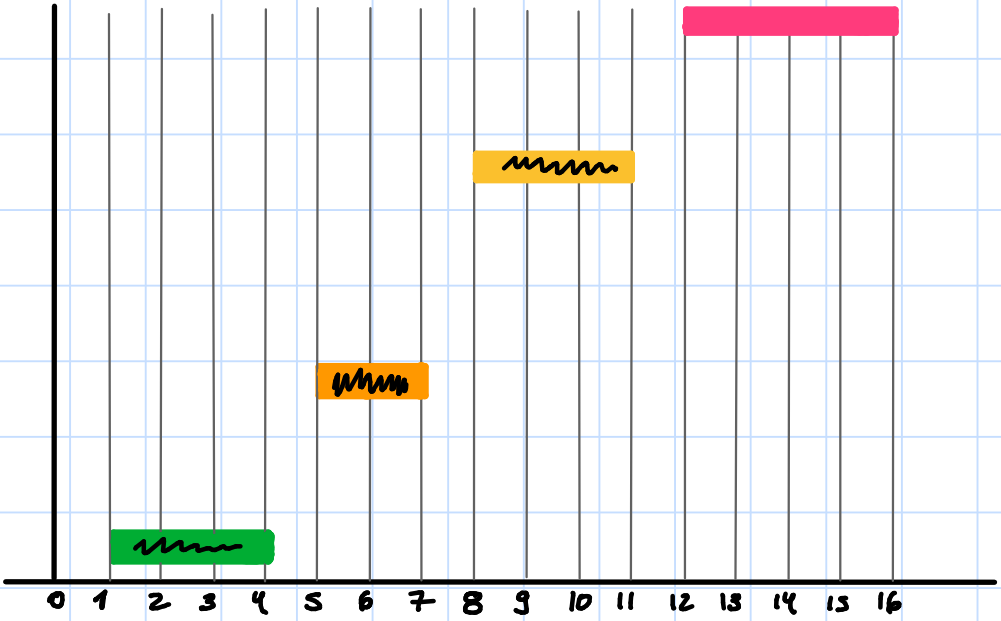
Pegamos "gulosamente" o "melhor" atividade entre as atividades restantes

Descobrindo um Algoritmo



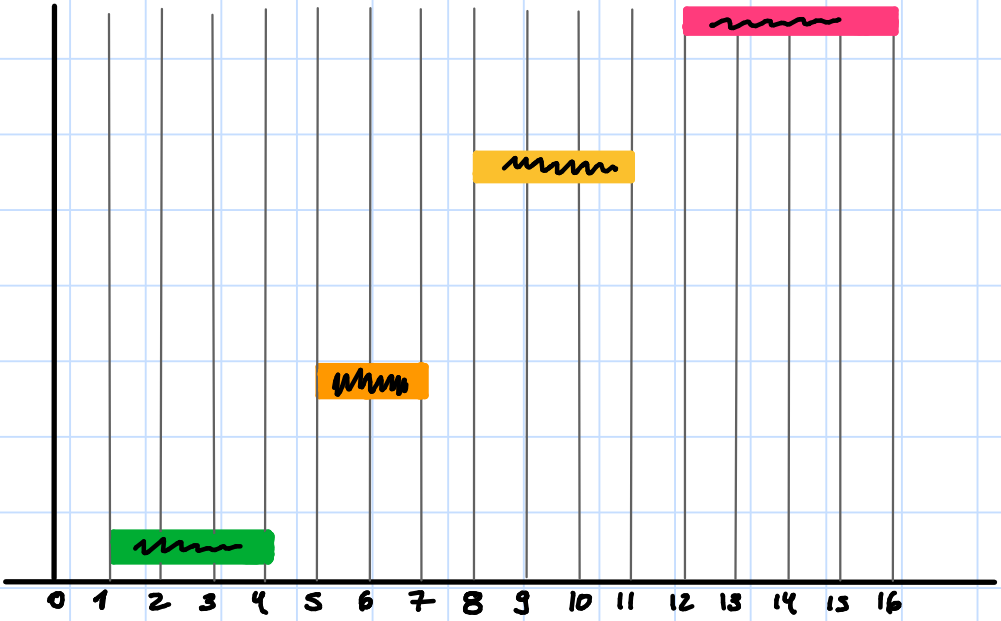
Pegamos "gulosamente" o "melhor" atividade entre as atividades restantes

Descobrindo um Algoritmo



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

Descobrindo um Algoritmo

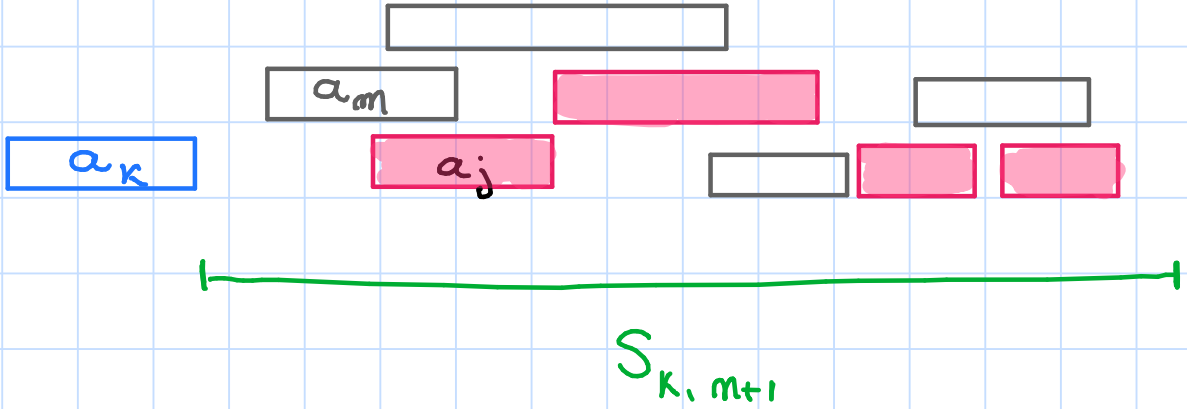


Pegamos "gulosamente" o "melhor" atividade entre as atividades restantes

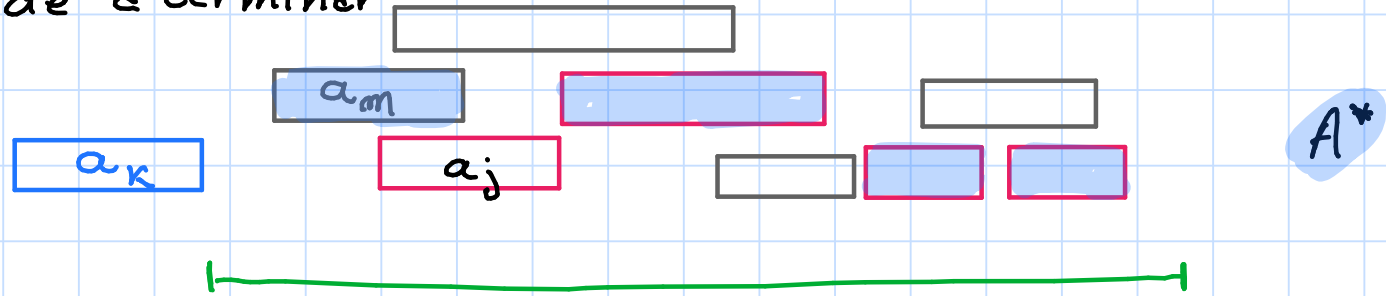
Lema Considere um subproblema $S_{k,m}$ não vazio, e seja $a_m \in S_{k,n}$ uma atividade com o menor tempo final. Então, existe uma solução ótima A de $S_{k,m}$ tal que $a_m \in A$.

Demonstração

- Seja A uma solução ótima de $S_{k,m}$, i.e.,
$$|A| = \psi_{k,m}$$
- 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



- Ademais $|A^*| = |A| = \psi_{k,m}$ □

Descobrindo um Algoritmo

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

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

Sabemos*, sim!!!

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

* ao menos para uma solução ótima específica!

escolha atividades entre $S_{k,n}$

Seleção-Atividade (s, f, k, n)

$m \leftarrow k+1$

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

$m \leftarrow m+1$

se $m < n$ e $f[m] \leq s[m]$

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

Retorna $\emptyset$

Next slide ⟹

Seleção-Atividade (s, f, k, n)

$m \leftarrow k+1$

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

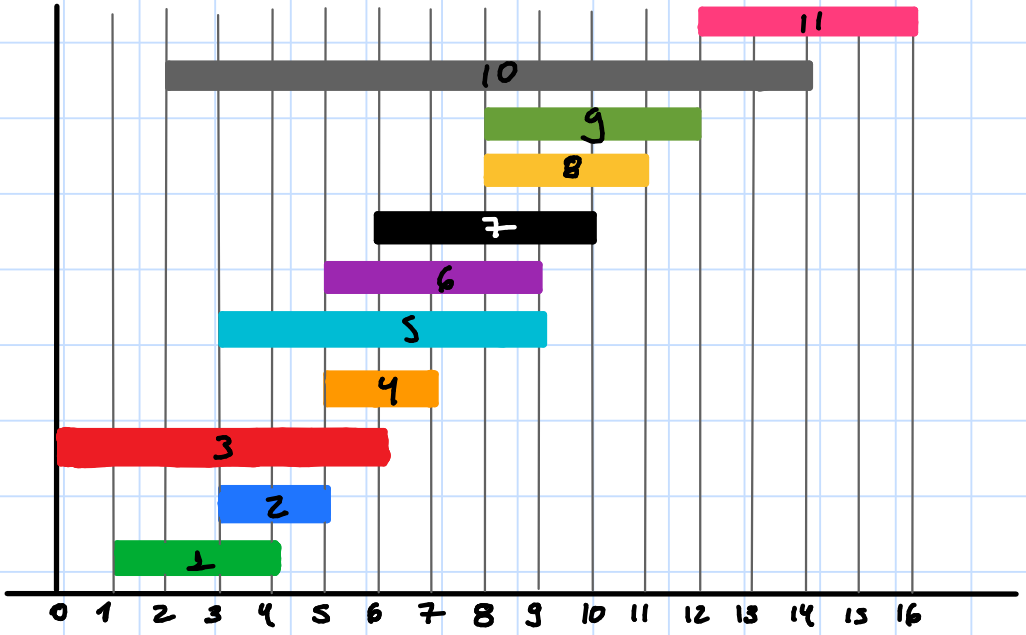
$m \leftarrow m+1$

se $m < n$ e $f[m] \leq s[n]$

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

Retorna $\emptyset$

Exemplo



escolha atividades entre $S_{k,n}$

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

Este algoritmo é o que chamamos de algoritmo guloso

- Defronte de um subproblema ($S_{k,n}$), 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 ($S_{m,n}$) 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

Receita para um algoritmo Guloso

Ingredientes

- Existe uma solução ótima que contém a escolha gulosa
- Subestrutura ótima

- Seja $S = \{a_1, a_2, \ldots, a_m\}$ (atividades ordenadas pelo tempo final)
  - Se a atividade a_k pertence a uma solução ótima do problema, vimos que

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

  - Pelo lema anterior, sabemos que existe uma solução ótima $A \subseteq S_{i,j}$ tal que $a_m \in A$, onde a_m é a atividade com o menor final em $S_{i,j}$
    - Então

$$\psi_{i,j} = \cancel{\psi_{i,m}}^{0} + \psi_{m,j} + 1$$

$$= \psi_{m,j} + 1$$

Pois $S_{i,m} = \emptyset$

Apenas um subprob.

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

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

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

③ Ademais

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

Então B é uma solução ótima.

escolha atividades entre $S_{k,n}$

Seleção-Atividade (s, f, k, n)

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

Next slide ⟹

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

Demonstração

A prova segue por indução em $l = |S_{k,n}|$

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

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

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

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

Pela subestrutura ótima do problema, sabemos que

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

Logo, $|B| = \psi_{k,n}$ □

Complexidade

Seleção-Atividade (s, f, k, n)

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

Complexidade

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

Podemos eliminar a recursão do algoritmo anterior, resultando no seguinte algoritmo iterativo.

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

```
Seleção-Atividade2 (s, f, k, n)

    A ← {a_{k+1}}
    t ← k+1
    Para m ← k+2 até n-1
        Se s[m] ≥ f[t] e f[m] ≤ s[m]
            A ← A ∪ {a_m}
            t ← m
    Retorna A
```

E observando o comportamento do algoritmo anterior sobre a entrada com as atividades postiças e pensando apenas no problema original, vemos que podemos simplificar o algoritmo para

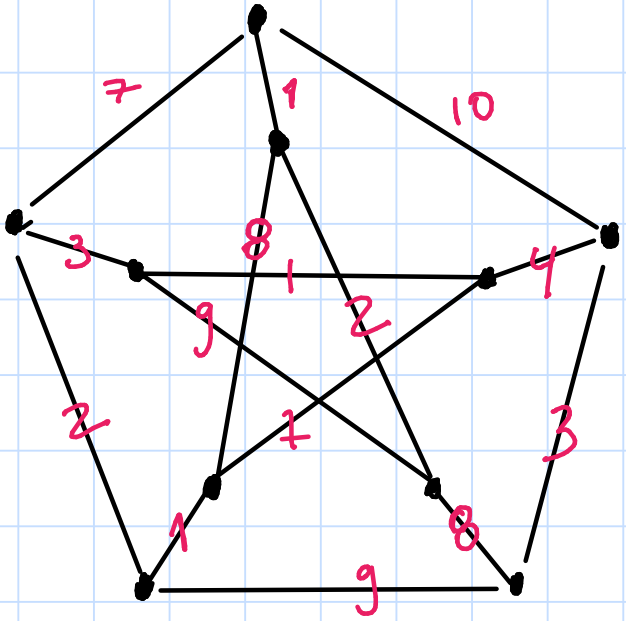
assumindo que $f_1 \leq f_2 \leq \cdots \leq f_n$

```
Seleção-Atividade3(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

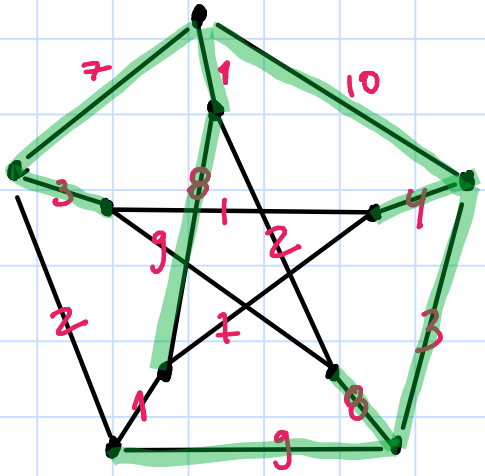
Um grafo ponderado é um grafo G e um função de custo $w : E(G) \to \mathbb{R}$.



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

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



$H \subseteq G$

$w(H) = 53$

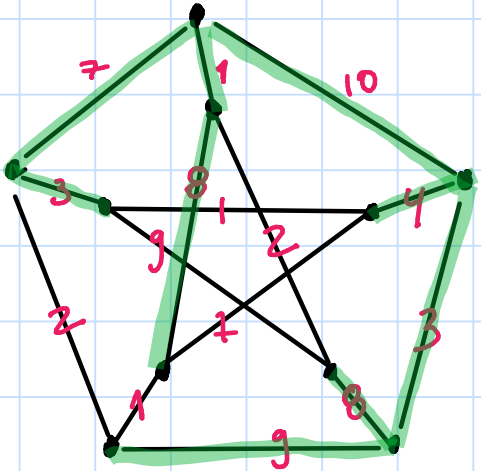
Árvore Geradora Mínima (AGM)

Problema da Árvore Geradora Mínima

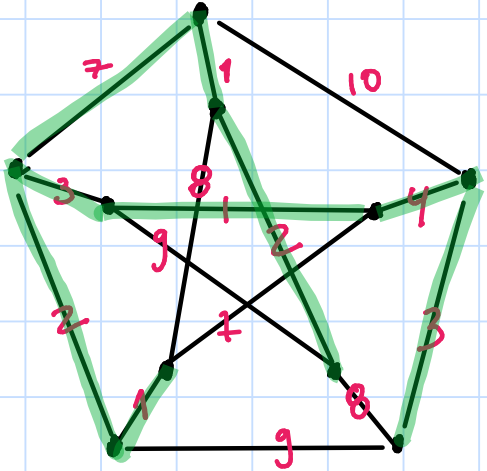
Entrada: Um grafo conexo G ponderado por uma função $w: E(G) \to \mathbb{R}^+$.

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

Exemplo

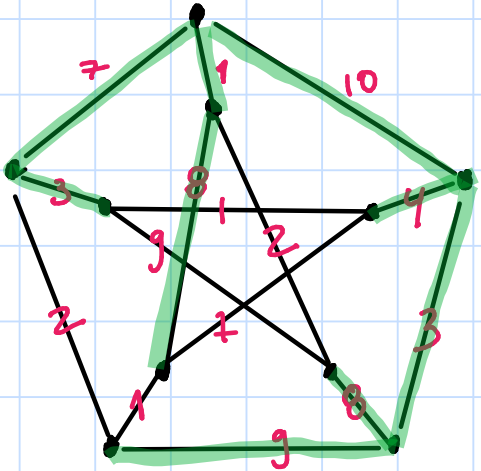


$w(H) = 53$



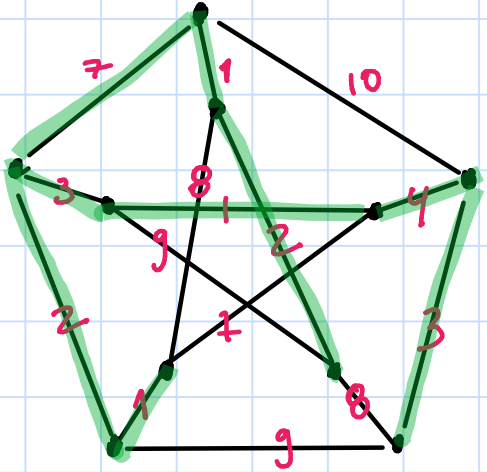
$w(H) = 24$

Exemplo



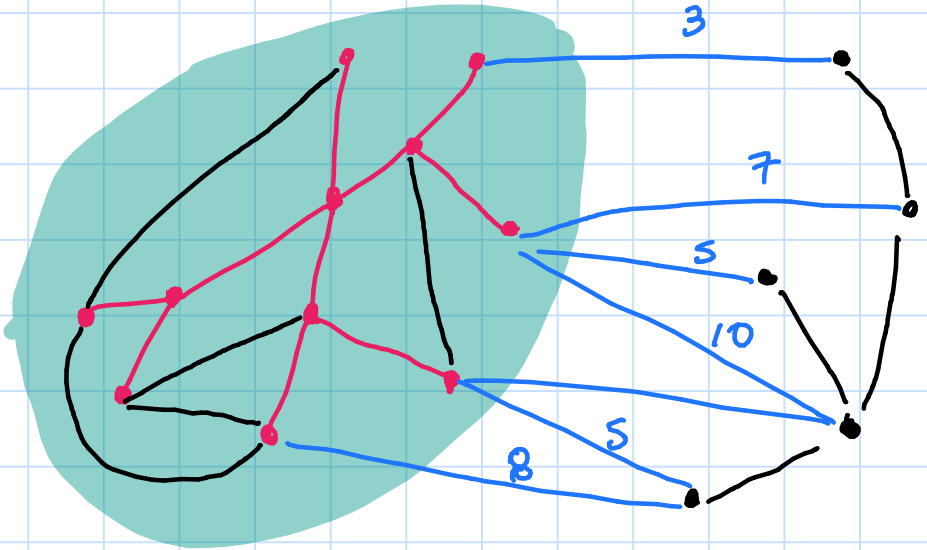
$w(H) = 53$

solução ótima

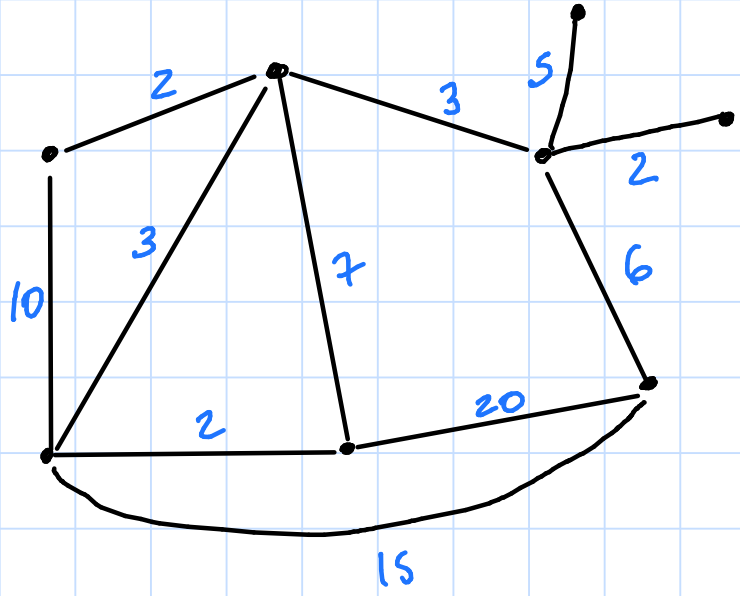


$w(H) = 24$

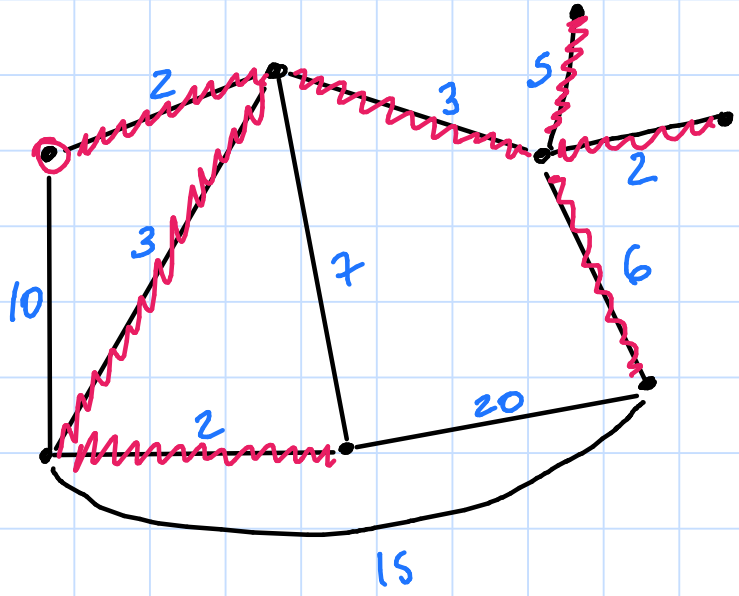
Intuição



Exemplo



Exemplo



$w(T) = 23$

- Dado um grafo G, seja ψ_G o peso de uma 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 $xy \in E(T^*)$!

Dado $X \subseteq V(G)$, dizemos que uma aresta $e \in E(X)$ é <u>Leve</u> se $w(e) = \min\{w(f) : f \in E(X)\}$.

Lema Sejam $X, Y \subseteq V(G)$ tais que $X \cap Y = \emptyset$ e $X \cup Y = V(G)$. Se $e \in E(X)$ é uma aresta leve, então $\exists$ uma AGM T^* de G tal que $e \in E(G)$.

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 $E(T_1)$

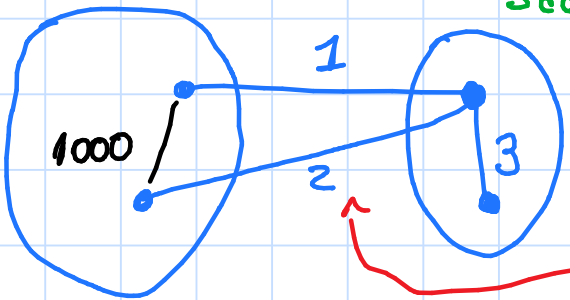
Subestrutura Ótima

- 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~~



cuidado!

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

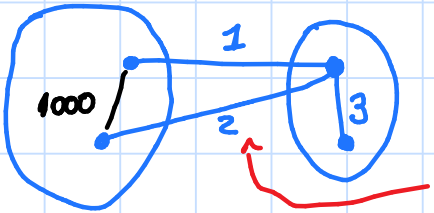
Subestrutura Ótima

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

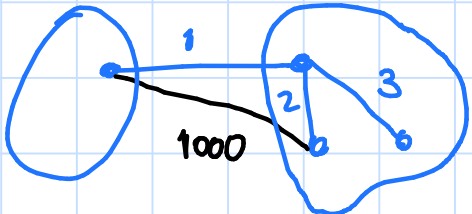
$$\psi_G = \psi_{G[V(T_1)]} + \psi_{G[V(T_2)]} + c(xy)$$

não sabemos quem são T_1 e T_2

não sabemos quem é xy



cuidado!



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

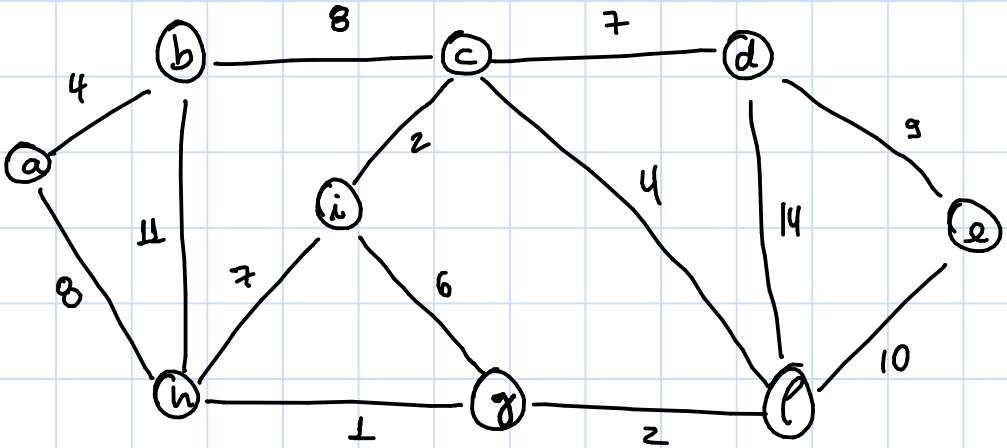
Subestrutura Ótima

Lema Seja G um grafo, seja $T \subseteq G$ uma árvore e seja T_1^* uma AGM de G. Se $T \subseteq T_1^*$ e $uv \in E(V(T))$ é uma aresta leve, então $\exists$ uma AGM T_2^* de G tal que

$$T + uv \subseteq T_2^*.$$

Demonstração (Exercício)

Exemplo



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

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 e T tem $\ell - 1$ arestas"

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 e tem $\ell - 1$ arestas. 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!

→ vai terminar pq o grafo é finito e na iteração k tem $k-1$ arestas

- 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 □

Implementação do Algoritmo de Prim

- Vamos manter as seguintes informações durante a execução do algoritmo
  - * todos os vértices que <u>não</u> estão na árvore estão em uma fila de prioridades (de mínima) Q
  - * a prioridade de $u \in V(G)$ na fila Q é $\kappa < \infty$ sse existe uma aresta $uv \in E(G)$ tal que $w(uv) = \kappa$ e $\kappa = \min \{ w(uz) : uz \in E(V(T)) \}$

    ↑ árvore que está sendo construída
  - * se a prioridade de u na fila Q for ∞, então não é possível conectar u a árvore sendo construída (até agora).

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   CriaFilaPrioridades (Q, V(G), ∞)   ▷ criando a fila com todos os vértices de G, dando prioridade ∞ para todos eles.
2   pred[s] = s                        ▷ escolhendo o vértice s como raiz
3   AtualizaPrioridade (Q, s, 0)
4   Enquanto não FilaEhVazia(Q) Faça
5       u ← Extract-Min (Q)
6       Para cada v ∈ N_G(u)
7           Se v ∈ Q e w(uv) < FilaGetPrioridade (Q, v)
8               pred[v] ← u
9               AtualizaPrioridade (Q, v, w(uv))
```

Complexidade

- Na análise desse código, vamos assumir que o grafo G é representado por uma lista de adjacências e que a Fila de Prioridades foi representada por um Heap mínimo.

- Na linha 1, chamamos uma rotina que cria uma heap mínima de V elementos, onde cada elemento tem prioridade ∞. Tal rotina pode ser implementada em $O(V)$, pois basta inicializar o vetor da heap colocando todos os elementos junto de suas prioridades. Afinal, como todos as prioridades são iguais, temos uma heap trivial.

Complexidade

- A linha 2 leva tempo constante p/ executar.
- Na linha 3 chamamos uma rotina para atualizar a prioridade do elemento s na Fila de Prioridades (Heap). Como visto em aula, tal procedimento pode ser feito em $O(\lg V)$.
- O laço da linha 4 executa enquanto a Fila $\bar{n}$ for vazia. Inicialmente a fila começa com V elementos e, em cada iteração do laço da linha 4, exatamente um elemento é removido da Fila Q. Assim, o laço da linha 4 executa $O(V)$ vezes.
- A Função EhVazia(Q) pode ser implementado em tempo constante, logo o custo da linha 4 é $O(V)$.

Complexidade

- A operação Extract Min (Q) da linha 5 pode ser implementada em $O(\lg V)$. Portanto o custo com a linha 5 é $O(V \lg V)$.

- Para um u fixo, a linha 6 do algoritmo executa em $O(d(u))$. Ao longo de toda a execução do algoritm, temos que esse laço executa

$$\sum_{u \in V(G)} d(u) = 2e(G) = O(E)$$

- Assim, o custo com a linha 6 é $O(E)$.

Complexidade

- Na linha 7 fazemos

  Se $v \in Q$ e $w(uv) <$ FilaGetPrioridade(Q, v) então

- Verificar se $v \in Q$ pode ser feito em $O(1)$, usando-se de um vetor de flags.
- Acessar o peso $w(uv)$ da aresta uv tbm pode ser feito em $O(1)$.
- Recuperar a prioridade de um elemento na fila tbm é uma operação que pode ser implementado em $O(1)$
- Portanto o custo com as linha 7-8 é $O(E) \cdot O(1) = O(E)$

Complexidade

- Como visto em sala de aula, e comentado anteriormente, o custo pl atualizar a prioridade de um elemento na heap é $O(\lg V)$. Portanto o custo com a linha 9 é $O(E \lg V)$.

- Assim, o custo total do algoritmo e

$$O(V \lg V + E \lg V)$$

- Como o grafo de entrada é conexo, sabemos que

$$e(G) \geq v(G) - 1$$

- Portanto $O(V \lg V + E \lg V) = O(E \lg V)$