

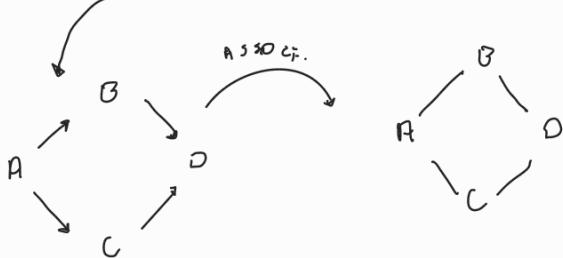
1) $G = (V, E)$ direcionado e não ponderado

A) fracamente $\rightarrow \exists u, v \in V | \nexists \text{path}(u, v)$
conexo

semi-fortemente $\rightarrow \forall u, v \in V | \begin{cases} \exists \text{path}(u, v) \\ \text{ou} \\ \exists \text{path}(v, u) \end{cases}$
conexo

fortemente
conexo $\rightarrow \forall u, v \in V | \begin{cases} \exists \text{path}(u, v) \\ \text{e} \\ \exists \text{path}(v, u) \end{cases}$

simplesmente
conexo \rightarrow o associado é conexo.



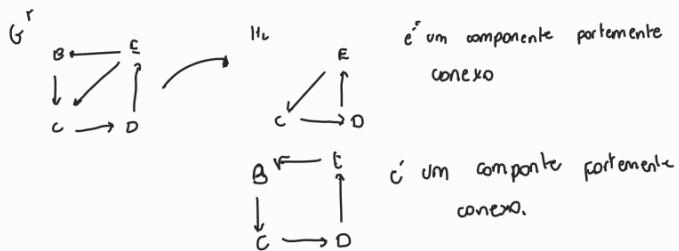
Fracamente \rightarrow existe par (u, v) pertencente a V tal que não existe caminho entre (u, v)

Fortemente \rightarrow para todo par (u, v) pertencente a V existe um caminho de ida e de volta entre (u, v)

Semi \rightarrow para todo par (u, v) pertencente a V existe um caminho de ida ou de volta entre (u, v)

Simplesmente \rightarrow o associado G' de G direcionado é conexa

B) Saber-se que um componente fortemente conexo é um subgrafo de G há um caminho de ida e de volta entre qualquer par de vértices.



Para identificar os componentes conexos de um grafo direcionado e conexo G .

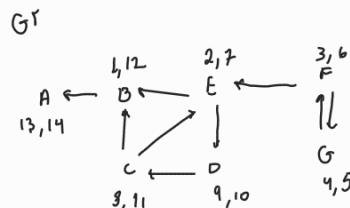
1) Escolher um vértice $u \in V$ e efetuar uma busca salvando os tempos de inicio

e fim.

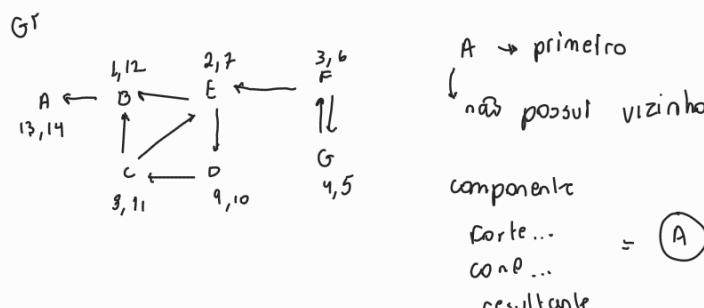
$A \rightarrow$ $1, 12$ $13, 14$	$B \rightarrow$ $2, 7$ $9, 10$	$E \rightarrow$ $3, 6$ 11	$F \rightarrow$ $4, 5$ 1
\downarrow $C \rightarrow$ $8, 11$		\downarrow $D \rightarrow$ $9, 10$	

Para a busca usaremos a DFS que tem como princípio buscar o vizinho mais profundo de um vértice não repetindo vértices já marcados como "processado".

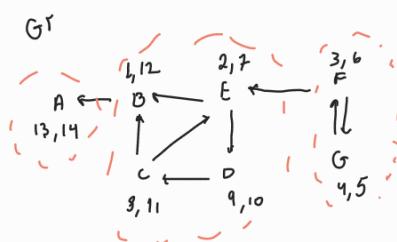
2) transpor $G \rightarrow$ um grafo G^T é um grafo resultante de G
que possui as direções de arestas contrárias de G
tal que $G^T = (V, E')$
 $G = (V, E)$



3) Executar uma nova busca em G transportado seguindo
a ordem de busca de maior tempo de fim



4. Cada chamada de busca
no passo 3 resulta num componente
fortemente conexo.



5) Sabese que uma árvore geradora mínima de G é um grafo T
resultante que é conexo e que possui $|V'| = |V|$ e $|E'| = |V'| - 1$ de
forma que a remoção de uma aresta de T desconecta o grafo e que
a soma dos pesos das arestas de T é mínima ou seja, não existe
outra árvore geradora com soma dos pesos das arestas < T .

$$|E'| = |V| - 1$$

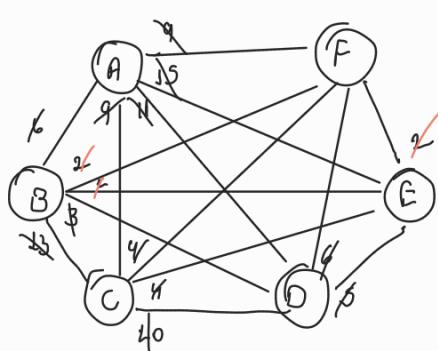
$$|V'| \leq |V|$$

$$T = |E'| = V - 1$$

$$|V'| = |V|$$

Kruskall X Prim
verifica todos os vértices
rodos rodos
arestas

A) Kruskall



1) Ordenar as arestas de forma
não decrescente.

ordem	1	2	3	4	5
(B,E)					
(B,F)					
(F,E)					
(B,D)					
(C,E)					
(C,F)					
(D,F)					
(A,C)					
(A,F)					
(C,D)					
(A,D)					
(B,C)					
(A,E)					
	6	7	8	9	10
	12	13	14		11
	(A,D)	(B,C)	(A,C)	(A,F)	(C,D)
	(B,L)				

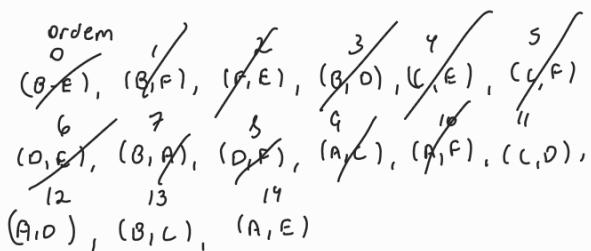
2) Enquanto $|E'| < (|V| - 1)$

3) selecionar menor aresta ordenada (B, E)

S) Se a aresta não formar um ciclo \rightarrow se as arestas de inicio e destino não estiverem no mesmo subconjunto de "vértices conhecidos"

SI Adicionar aresta a $|E'|$.

A funcionamento no grafo dado



Execução 5

$ E' < V - 1$?	Aresta	Peso	Vértices Conhecidos	Fechou Ciclo	Soma
True	(B, E)	1	{B, E}	Não	1
True	(B, F)	2	{B, E, F}	Não	3
True	(F, E)	2	{B, E, F}	Sim	3
True	(B, D)	3	{B, E, F, D}	Não	6
True	(C, E)	4	{B, E, F, D, C}	Não	10
True	(C, F)	4	{B, E, F, D, C}	Sim	10
True	(D, E)	5	{B, E, F, D, C}	Sim	10
True	(B, A)	6	{B, E, F, D, C, A}	Não	16
True	(D, F)			Sim	16
True	(A, C)			Sim	16
True	(A, F)			Sim	16

B) Prim

1) Considerar um vértice x de V tal que $V^r = \{x\}$ e

$|E'| = 0$ ou seja

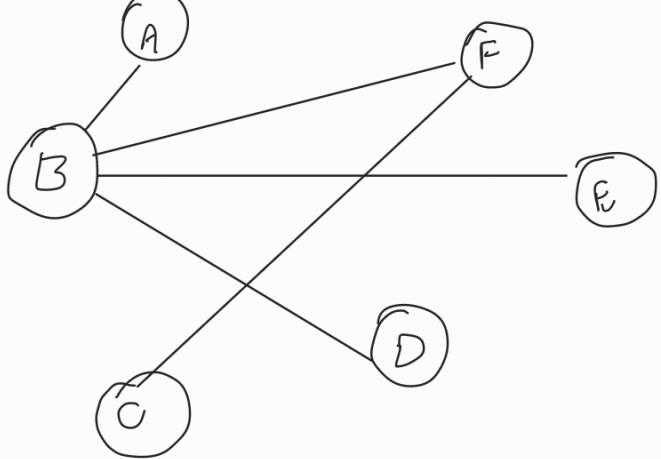
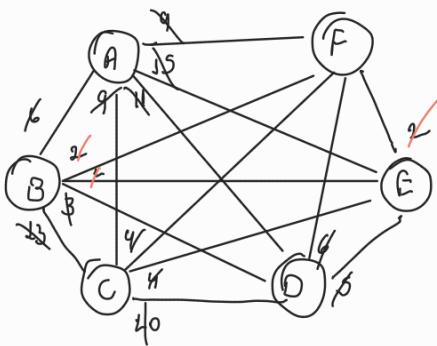
2) Enquanto $|V^r| \neq |V|$

3) Selecionar a aresta de menor peso (y, w) tal que $y \in V^r$ e $w \in V$

4) se a aresta não fechar ciclo

SI Adicionar aresta em $|E'|$ e adicionar v em V^r .

* Para fechar um ciclo x e w não podem fazer parte do mesmo subconjunto.



$$V^r = \{O\}$$

$$V^T = \{O, B\}$$

$$V^r = \{O, B, E\}$$

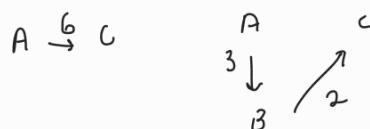
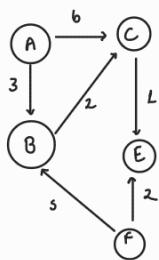
$$V^r = \{O, B, E, F\}$$

$$V^r = \{O, B, E, F, C\}$$

$$V^r = \{O, B, E, F, C, A\}$$

soma = 16

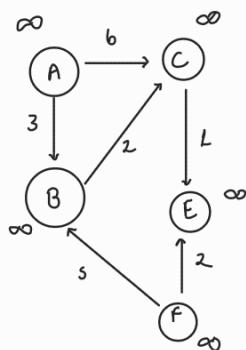
c) O algoritmo de Dijkstra é usado para encontrar o caminho mínimo (soma dos pesos de arestas) entre um vértice x e outro vértice w . Por exemplo, no grafo abaixo a menor distância entre A-C em número de arestas é 1, porém se levarmos em consideração os pesos, o número de arestas será 2 pois $6 > 3 + 2$



Para executar o algoritmo:

1) Marcar todos os vértices como "não processados" $\rightarrow [A, B, C, D, E, F]$

2) $\forall v \in V \rightarrow \text{dist}[v] = \infty$ e predecessores $[v] = -1$



dist
pred

A	B	C	D	E	F
dist	∞	∞	∞	∞	∞
pred	-1	-1	-1	-1	-1

3) Escolher um vértice de inicio.

4) $\text{dist}[\text{inicio}] = 0$

A	B	C	D	E	F
0	∞	∞	∞	∞	∞
-1	-1	-1	-1	-1	-1

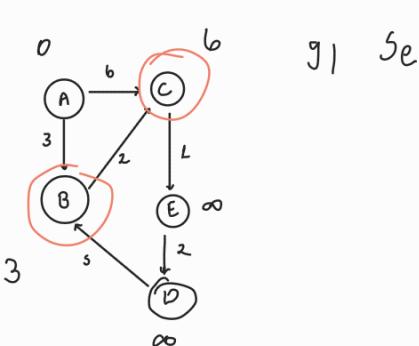
5) Enquanto existirem vértices para ser processado

→ 6) Selecionar $v = \arg \min (\text{distâncias}[v])$ | $v \in$ vértices não processados)

7) marcar v como processado

vértices
não processados

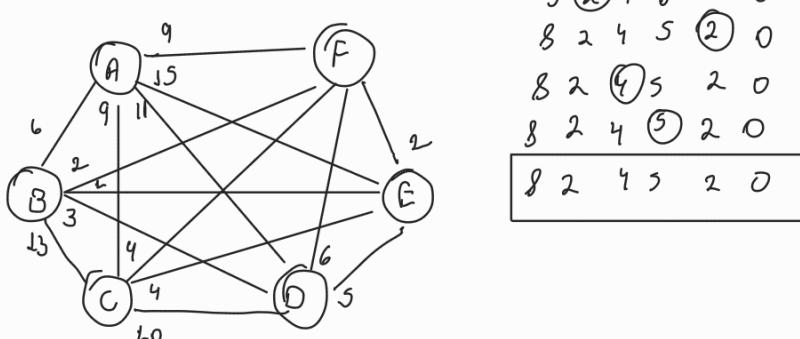
8) Para todo vértice w vizinho de v ainda não processado



g) Se $\text{dist}[w] > (\text{dist}[v] + \text{peso}(v, w))$

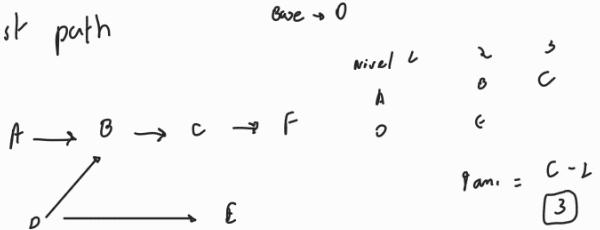
L0) $\text{dist}[w] = \text{dist}[v] + \text{peso}(v, w)$

L1) $\text{pred}[w] = v$



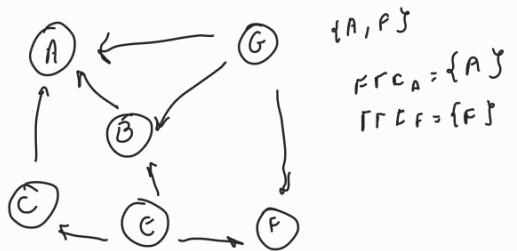
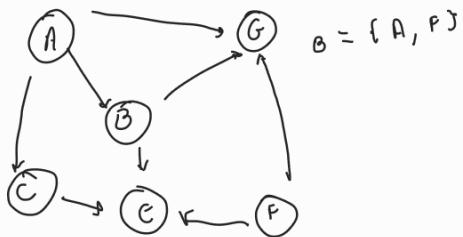
A	X	X	X	X	X
∞	∞	∞	∞	∞	0
9	(2)	4	6	2	0
8	2	4	5	(2)	0
8	2	(4)	5	2	0
8	2	4	(5)	2	0
8	2	4	5	2	0

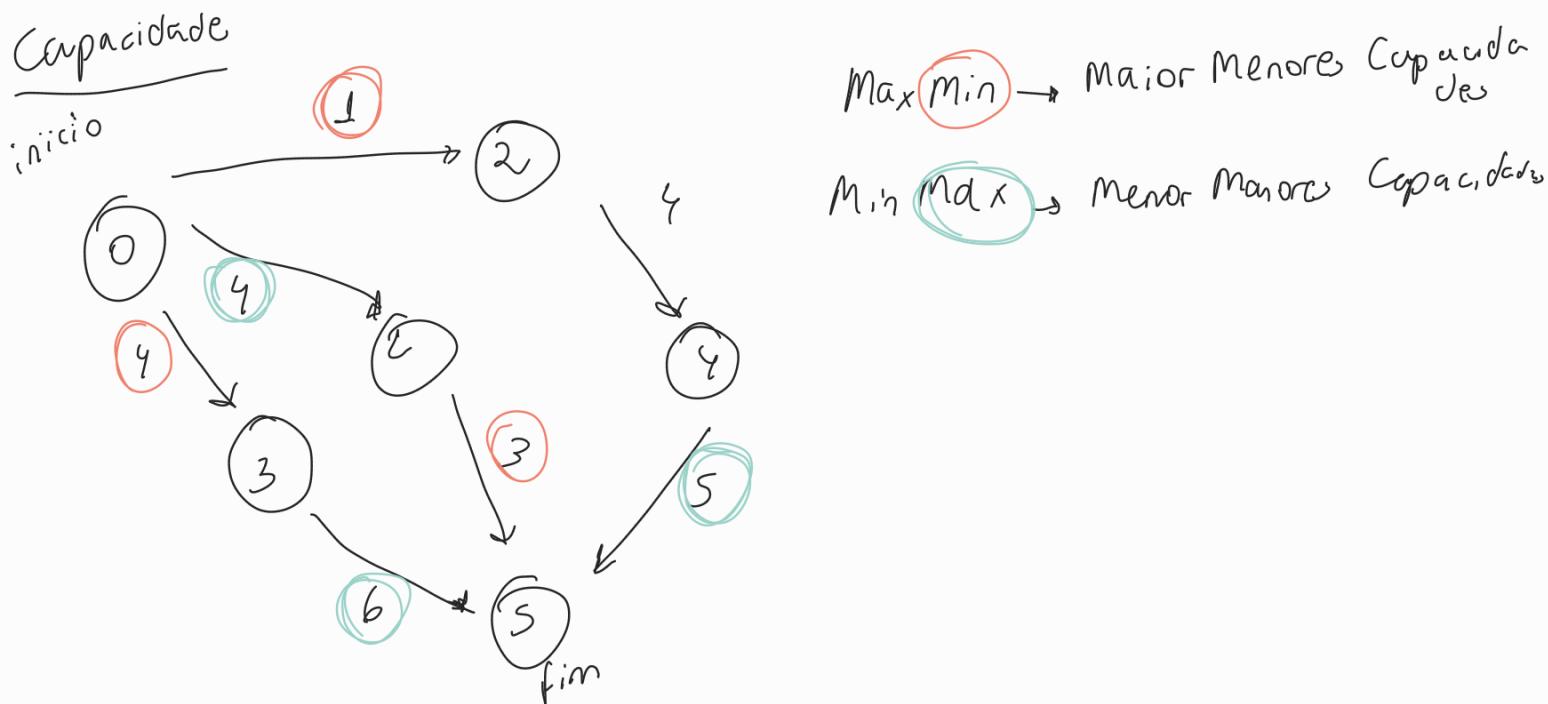
Longest path



Sabe-se que uma base é o menor subconjunto $B \subseteq V$, tal que não existe caminho entre os elementos de B e todos os elementos de V saídos acomodados por um elemento de V . Para encontrar uma base, podemos efetuar uma busca em profundidade em G para encontrar o fecho transitivo inverso de cada vértice, para isso:

- 1) Transpor G
- 2) Para todo os vértices de v , visitar todos os vértices adjacentes de v adicionando-os ao FRC de v
- 3) Todos os vértices que tiverem FRC apenas com 1 elemento (ele próprio) será uma base.





$$\text{Maior } [4, 1, 3] = 4 \quad \text{Max Min} = 4$$

$$\text{Menor } [6, 4, 5] = 4 \quad \text{Min max} = 4$$

1) $\forall v \in V, \text{cap}[v] = \infty$ e $\text{pred}[v] = -1$

2) $\text{cap}[\text{origem}] = 0$

3) FP. adicionar ($\text{cap}[\text{origem}]$, origem) \rightarrow fila de prioridade

4) Enquanto ! FP vazia

5) $v = \text{FP.pop}()$

6) $\forall w$ vizinho de v

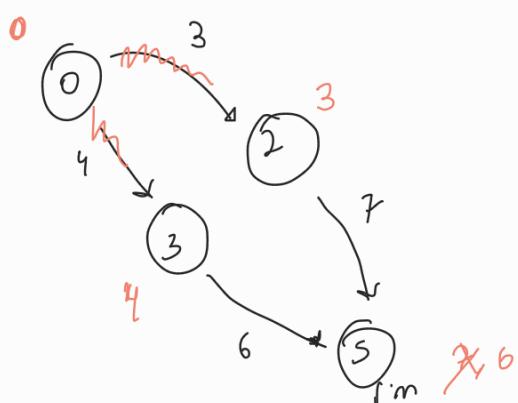
7) $\text{Cap Atual} \leftarrow \text{arg max}(\text{cap}[v], \text{peso}(v, w))$

8) if ($\text{cap Atual} < \text{cap}[w]$)

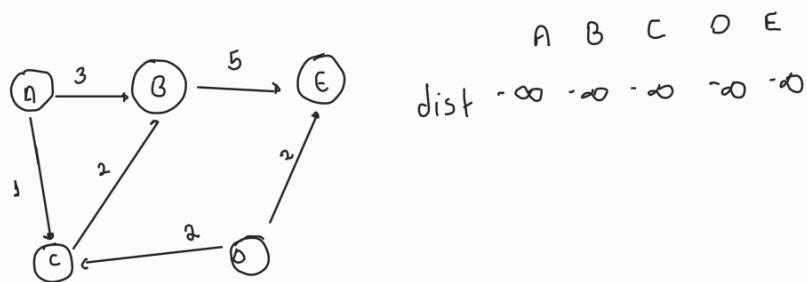
$\text{cap}[w] = \text{cap Atual}$

$\text{pred}[w] = v$

FP.adicionar (cap Atual , w).



1) para todos os vértices v de V iremos inicializar a capacidade $\text{cap}[v]$ como $-\infty$, de forma que:



2) definiremos a capacidade do vértice inicial como ∞ e iniciaremos uma fila de prioridade (FP). O critério será a maior capacidade.) e adicionaremos o vértice inicial e sua capacidade à FP.

início	A	B	C	D	E	FP
	∞	$-\infty$	$-\infty$	$-\infty$	$-\infty$	(∞, A)

3) Enquanto existirem elementos na fila iremos remover o primeiro elemento da fila e atribuir a v .

$$\text{FP} (\infty, A) \quad v = A \quad \text{cap}[v] = \infty$$

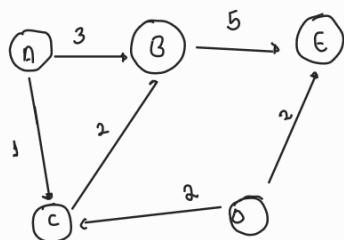
4) Para todos os vizinhos w não visitados de v iremos definir uma capacidade atual como o argumento mínimo entre $\text{capacidade}[w]$ e o peso da aresta (v, w) .

$$\text{capacidade}_{\text{atual}} = \arg \min(\infty, 3)$$

Se a capacidade atual for maior que a capacidade $[w]$ iremos atualizar a capacidade de w para capacidade atual e adicionaremos o par $(\text{capacidade}[w], w)$ na fila de prioridade $\text{cap}[B] = -\infty \quad 3 > (-\infty) ? \text{Sim} \quad \text{cap}[B] = 3 \quad \text{FP. adicionar } (3, B)$

5) A maior capacidade mínima de Início para destino será a capacidade final em destino.

1) Para todos os vértices de U , iremos iniciá-las com $\text{cap}[v] = \infty$.



vértice	A	B	C	D	E
cap	∞	∞	∞	∞	∞

2) Escolher um vértice inicial e atualizar $\text{cap}[\text{inicial}]$ como $-\infty$ e iremos uma fila de prioridade (o critério será a menor capacidade) e adicionaremos o vértice inicial e sua capacidade à fila.

vértice	A	B	C	D	E
cap	$-\infty$	∞	∞	∞	∞

$$\text{inicial} : A \quad \text{cap}[\text{inicial}] = -\infty$$

3) Enquanto existirem elementos na fila de prioridades, iremos remover o primeiro elemento e atribuiremos a U .

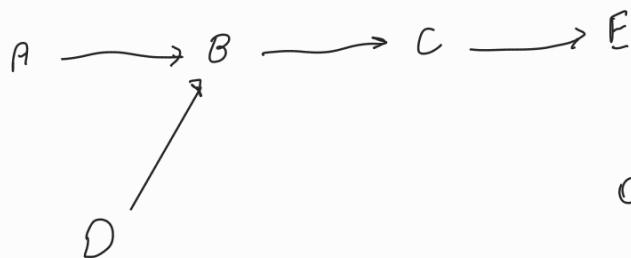
$$U = \text{FP. remove}() \quad U = A \\ \text{cap}[U] = -\infty$$

4) Para todos os vizinhos w não visitados de U , definiremos uma capacidade atual como argumento máximo entre ($\text{cap}[w]$, peso(U, w)).

$$-\infty \xrightarrow[3]{A} B \quad \text{capacidade} = \underset{\text{atual}}{\arg \max} (-\infty, 3)$$

Se a capacidade atual for menor que $\text{cap}[w]$ atribuiremos a $\text{cap}[w]$ o valor da capacidade atual e adicionaremos o par $(\text{cap}[w], w)$ a fila de prioridades

$$\text{cap}[B] = \infty \quad 3 < \infty ? \text{ sim} \quad \text{cap}[B] = 3 \quad \text{F.P. adicionar } (3, B)$$



ordem

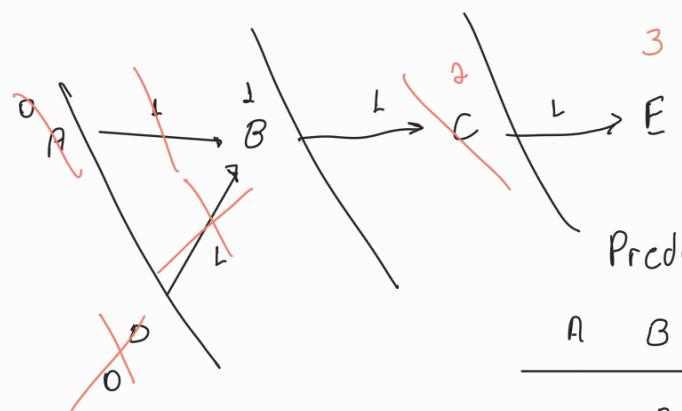
A D B C E

Ramalho maior Caminho ≈ 3

Te amo! A gente
te ajudaria a

ordem

A D B C E



$$\text{dist}[vizinho] < \text{dist}[atual] + \text{peso}$$

$-\infty$

0

$\tau = r$

1

0

$\tau = v$

$-\infty$

1

$\tau = t$

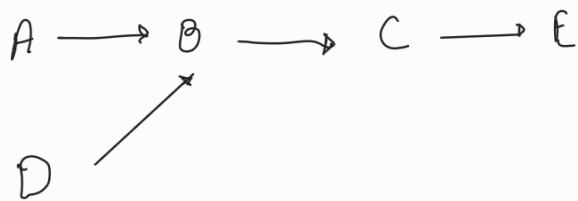
$-\infty$

2

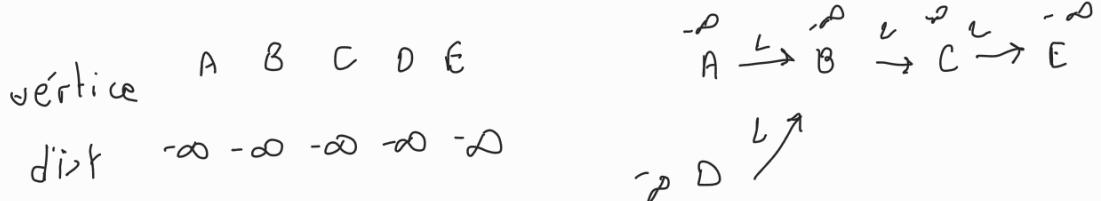
$\tau = r$

Predecessores

A	B	C	D	E
A	B	C		



- 1) Sabendo a ordem topológica, para todos os vértices iniciar $\text{dist}[v] = -\infty$ e para todas as arestas atribuir peso 1.



- 2) Para as bases de G, atribuir o valor 0 a $\text{dist}[\text{base}]$

Bases {A, D}

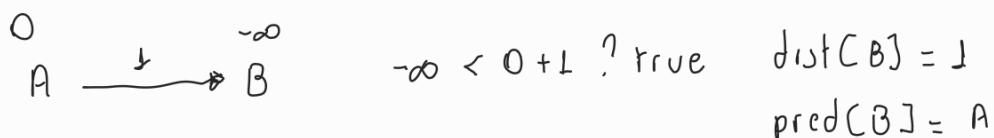
vértice	A	B	C	D	E
dist	0	$-\infty$	$-\infty$	0	$-\infty$

- 3) Enquanto $V \neq \{\}$

- 4) Seguindo a ordem topológica, atribuir primeiro valor a v .

ordem	$v = A$
X A B C E	$\text{dist}[v] = 0$

5) Para todos os vizinhos w de v ainda não visitados, se $\text{dist}[w] < (\text{dist}[v] + \text{peso}(v, w))$, atribuir a $\text{dist}[w]$ o valor de $(\text{dist}[v] + \text{peso}(v, w))$ e atualizar o $\text{pred}[w]$ para v .



6) remover v de \cup .

7) O caminho é obtido após retroceder no array de pred começando pelo vértice v de maior dist[v] até encontrar o vértice sem predecessor, ou seja, o anel central de todos.

$A \rightarrow B \rightarrow C \rightarrow E$

D



caminho

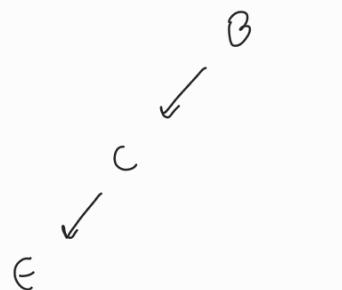
A B C D E

pred

A B C

dist

0 1 2 0 3



Sabe-se que uma árvore geradora mínima é um subgrafo T resultante de G que possui $|V'| = |V|$ e $|E'| = (|V|-1)$, também que a remoção de uma aresta de T o desconecta e que a soma dos pesos das arestas de T é mínimo, ou seja, não existe outra AGM com soma dos pesos menor que T.

Suponha que T_1 e T_2 sejam árvores geradoras mínimas de G :

$$|E^1| = \{e_1^{r_1}, e_2^{r_1}, e_3^{r_1} \dots e_{v-1}^{r_1}\}$$

$$|E^2| = \{e_1^{r_2}, e_2^{r_2}, e_3^{r_2} \dots e_{v-1}^{r_2}\}$$

Como r_1 e r_2 são AGMs, em teoria ambas possuem a mesma soma dos pesos, mesma cardinalidade de vértices mas estruturas diferentes, ou seja, as arestas de r_1 não estão presentes em r_2 .

Seja e_x^* uma aresta mínima de T_1 que não está presente em T_2 . Se adicionarmos essa aresta em T_2 formaremos um ciclo, já que T_2 já conecta todos os vértices, logo se substituirmos a aresta e_y que fecha ciclo em T_2 por e_x e $e_y < e_x$, significa que T_1 não tem peso mínimo e se $e_y > e_x$, significa que e_y não possui peso mínimo, logo, o peso de e_x e e_y deve ser o mesmo, impedindo a existência de duas AGMs simultâneas em G.