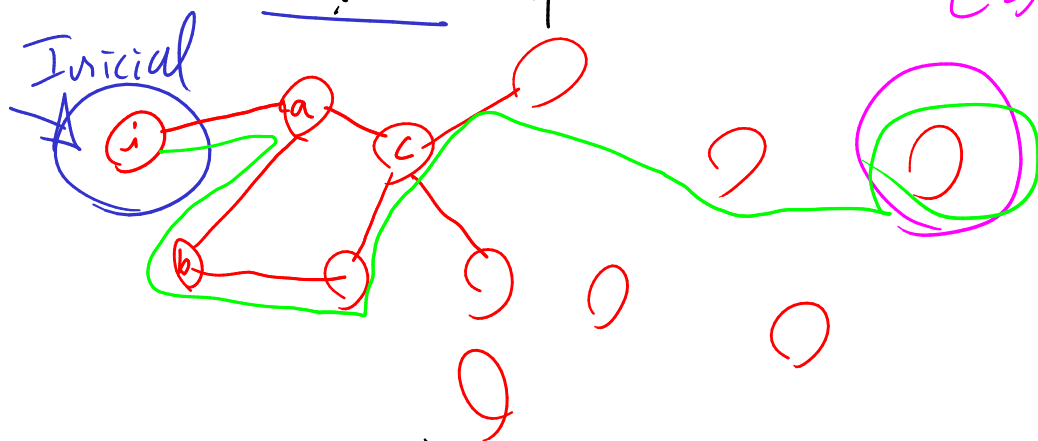


## 6, Problemas de Travessia.

→ Vértices: estados / marcam importantes "acontecimentos" para o problema

→ Transições marcam mudanças de estado respeitando as restrições do problema.

→ Uma forma de representar "a resolução do problema" na qual a travessia/percurso nos leva a solução requerida.



→ Conhece o estado inicial e vai conhecendo o restante do grafo a medida que a travessia vai acontecendo.



→ Buscas em largura e profundidade

↳ Best First Search: função de avaliação (distância)  
↳ A\*: distância + heurística

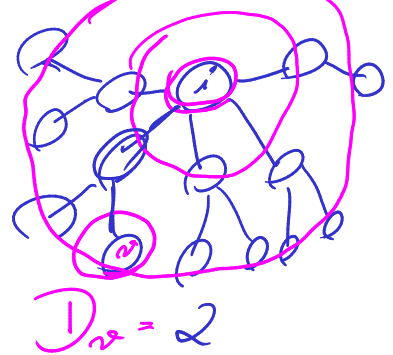
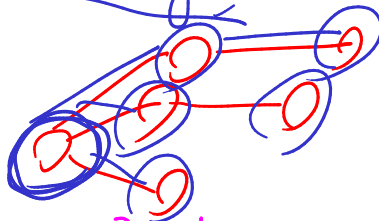
→ Exemplo:  $\begin{pmatrix} (0,0,0) \\ (0,0,1) \\ (1,0,0) \end{pmatrix}$   $\begin{pmatrix} (0,0,0) & (0,0,1) \\ & (1,0,0) \end{pmatrix}$

# Prova de Queimante

- 1) 
 → Identificação de cada indivíduo  
 → Listagem de ligações e a duração Lados  
 → ligações importantes: mais de 30 seg devem ser considerados  
 → Ind. próximos: até 2 ligações

↳ 1 ind. chamado

→ cada ind. será um vértice  
 → Uma aresta representa uma ligação entre dois indiv.: +30 seg




- 1a) Para o problema será utilizado um grafo não-dirigido e não-ponderado, no qual cada vértice representa um indivíduo e cada aresta representa uma ligação telefônica entre dois indivíduos que durou mais de 30 segundos. O grafo é não-dirigido porque a informação de origem/destino da ligação não é importante para responder o problema, apenas a informação dos indivíduos que participam da ligação é importante. O grafo é não-ponderado porque considera-se apenas ligações com mais de 30 segundos, então entende-se que a representação de arestas apenas para ligações com essa característica é suficiente para responder o problema.

- 1b) Dados a listagem de indivíduos, ligações e informação da duração de cada chamada, obtém-se o grafo a partir do seguinte pseudo-código:
- entrada: lista de indivíduos  $I$ , lista de ligações  $L$  [ $E(c_1, c_2)$ ]  
 uma função  $D: L \rightarrow \mathbb{R}^+$  da duração em minutos segundos

1.  $V = \emptyset$
2.  $E = \emptyset$
3. foreach (nome, cpf) in  $I$ ;
4.  $V \leftarrow V \cup \{\text{cpf}\}$
5. foreach ( $c_1, c_2$ ) in  $L$ ;
6.   if  $D(c_1, c_2) > 30$ ;
7.     $E = E \cup \{\{c_1, c_2\}\}$
8.  $G = (V, E)$
9. return  $G$ .

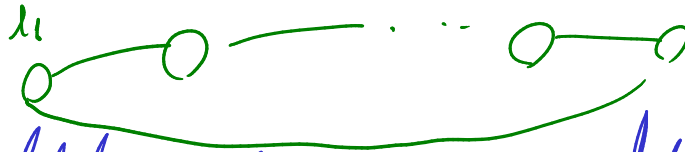
- 1c) Será utilizado uma busca em largura, pois ela permite encontrar um caminho mínimo em número de arestas a partir de um vértice inicial  $s$  para todo vértice do grafo. A alteração requerida deve ser realizada na linha 11 do Algoritmo 3 das anotações de aula. Deve-se substituir a instrução da linha pela seguinte:

$D_u = 1$   11.  $\text{if } C_v = \text{false} \text{ and } D_u < 2 \text{ then}$   
 $D_v = 2$

2)  $\rightarrow l_1, l_2, \dots, l_{1000}$  : localidades  
 $l_{\text{início}}$  e  $l_{\text{fim}}$

$\rightarrow P = \{(l_i, l_j), \dots\}$  onde  $l_i$  e  $l_j$  não poderiam ser conectados.

$\rightarrow$  Não é informado o custo de cada conexão.



$\rightarrow$  todas as localidades devem ser conectadas, nesse trajeto

$\rightarrow$  Ciclo Hamiltoniano  $\rightarrow$  Bellman-Held-Karp.

- 2a) Será criado um grafo não-dirigido e não-ponderado no qual cada localidade é um vértice. O conjunto de arestas será definido de acordo com o pseudo-código abaixo assumindo o conjunto  $P$  como os das conexões que devem ser evitadas e que  $V$  é o conjunto de vértices:

1.  $E = V \times V$

2. For each  $(l_i, l_j) \in P$ :

3.  $E = E \setminus \{(l_i, l_j)\}$ .

- 2b) Será utilizado o algoritmo de Bellman-Held-Karp para descobrir o ciclo desejado. Não será necessária nenhuma alteração no algoritmo, desde que o grafo produzido na questão 2a seja adaptado da seguinte forma:

Entrada: um grafo  $G = (V, E)$  não-dirigido e não-ponderado

1.  $E' = V \times V$

2. Criar função de peso  $w: E' \rightarrow \mathbb{R}^+$

3. ...

3. For each  $\{u, v\}$  in  $E'$  do  
 4.   if  $\{u, v\} \in E$  then  
 5.     L define  $w(\{u, v\}) \rightarrow 1$   
 6.   else  
 7.     L define  $w(\{u, v\}) \rightarrow \infty$   
 8. return  $(V, E', w)$ .

O grafo resultante desse procedimento deve ser o grafo de entrada para o algoritmo de Bellman-Held-Karp. Caso a resposta obtida pelo algoritmo tenha custo infinito, então deve-se informar que não é possível encontrar um ciclo para o problema. Caso contrário, pode-se apresentar o ciclo obtido como resposta para a questão.

3)  $|E| \approx |V| \rightarrow |E| \leq c \cdot |V|$   $m \leq c \cdot n$   
 FW  $\rightarrow \text{Tempo} \in O(n^3)$   $n = |V|$   $\forall s \forall t$   
 BF  $\rightarrow O(n, m) \approx O(n^2)$   $\forall t$   
 Dijkstra  $O((n+m) \log_2 n)$   $\forall t$

$\forall v \in V: B(v)$   
 $n \times n^2 = n^3$   
 $\forall v \in V: \text{Dijkstra}(v)$   
 $n \cdot (n + n) \log_2 n$   
 $n \cdot 2n \log_2 n$   
 $2n^2 \log_2 n \leq n^3$

3a) Entrada:  $G = (V, E)$

1. Criar um vetor  $D^{(u)} \forall u \in V$   
 2. Criar um vetor  $A^{(u)} \forall u \in V$   
 3. For each  $u \in V$  do  
 4.   L  $(D^{(u)}, A^{(u)}) \leftarrow \text{Dijkstra}(G, u)$   
 5. return  $(D, A)$

D	1	2	...
u=1			
u=2			

3b) A complexidade do algoritmo sugerido será de  $O(2 \cdot n^2 \log_2 n)$  para o pior caso, considerando que a quantidade de arestas/arcs é aproximada ao número de vértices, e considerando também que  $n = |V|$ .

3c) Dijkstra usado: s/ resposta.

4) O seguinte algoritmo resolve o problema:  
Entrada:  $G = (V, A, w)$ ,  $t \in V$ .

1.  $A' \leftarrow \{ \}$

2. criar função  $w': A' \rightarrow \mathbb{R}$

3. foreach  $(u, v) \in A$  do

4.  $A' \leftarrow A' \cup \{(v, u)\}$

5.  $\left\{ \begin{array}{l} \text{definir } w'((v, u)) \rightarrow w((u, v)) \end{array} \right.$

6.  $(D, A) = \text{Bellman-Ford}((V, A', w'), t)$

7. return  $(D, A)$

O algoritmo inverteu a direção dos arcos e manteve seus custos, desse modo, utilizando Bellman-Ford, a origem requerida por esse algoritmo se torna o destino desejado pelo enunciado. Como informações sobre custos negativos não foram dadas pelo enunciado, Dijkstra não foi utilizado, apesar de ser mais eficiente.