GRAFOS

Problemas envolvendo Redes

Redes

Conceitos gerais

- Quando a um grafo G se associam valores aos seus vértices e/ou às arestas (grafo pesado), este grafo designa-se geralmente por **rede**
- Numa rede
  - os vértices designam-se por **nós/nodos**
  - arestas designam-se por **arcos**
- Uma rede pode ser representada por G = (N, A, C), em que
  - N é o conjunto de nós
  - A é o conjunto de arcos
  - C é o conjunto de valores associados aos arcos (*comprimentos*, *custos*, ...):

    ao arco (i, j) está associado o valor C_{ij}
- De uma maneira geral, os conceitos utilizados em grafos são extensíveis a redes

Conceitos gerais

- Seja G = (N, A, C) uma rede não-dirigida conexa
  - o *comprimento* do caminho **p** entre dois nós da rede G, é a soma dos *comprimentos* dos arcos que pertencem àquele caminho:

$$C(p) = \sum_{(i,j)\in p} C_{ij}$$

  - designa-se por **árvore abrangente mínima** ("Minimum Spanning Tree") a árvore abrangente com o menor *comprimento* entre todas as árvores abrangentes de G
    - um subgrafo que seja uma árvore e contenha todos os vértices do grafo é designado por **árvore abrangente** ou **árvore total** ("Spanning Tree")
  - designa-se por **árvore mínima** (**árvore de caminhos mais curtos**) com raiz em S, a árvore que contém todos os nós de N acessíveis a partir de S, em que para cada nó **k** o único caminho de **S** para **k** é o caminho mais curto (de comprimento mínimo) na rede G, que liga **S** a **k**
  - em geral, a *árvore abrangente mínima* é diferente da *árvore mínima* entre um nó origem e todos os restantes nós da rede

Problema da Árvore Abrangente Mínima

Conceitos gerais

- Dado uma rede não-dirigida G, pretende-se determinar/encontrar uma árvore abrangente mínima de G
- Os valores (custos, comprimentos, ...) dos arcos da rede G são valores inteiros (positivos e negativos)
- O problema tem solução se e só se a rede G é conexa
- Portanto, com a resolução deste problema, pretende-se determinar uma árvore abrangente mínima de uma rede não-dirigida conexa

Algoritmo de PRIM

Seja G = (N, A, C) uma rede não-dirigida conexa

Passo 1.

Tome-se arbitrariamente um nó S e atribui-se-lhe um rótulo permanente nulo: $R_S = 0$

Aos restantes nós da rede atribuem-se rótulos temporários:

$R_j \leftarrow C_{Sj}$, se $(S, j) \in A$

$R_j \leftarrow \infty$, se $(S, j) \notin A$

Permanentes $\leftarrow \{ S \}$

Temporários $\leftarrow N - \{ S \}$

Algoritmo de PRIM

Passo 2.

k $\leftarrow$ nó com rótulo temporário contendo o menor valor (que é vizinho de um nó **i**)

Permanentes $\leftarrow$ Permanentes $\cup$ { k }

Temporários $\leftarrow$ Temporários - { k }

(o arco com o mínimo valor $C_{ik} = R_k$ passa a pertencer à árvore abrangente mínima)

se (Temporários = $\emptyset$) **então**

PARAR (*foi determinada uma árvore abrangente mínima*)

fim_se

Passo 3.

para (todo o j $\in$ N tal que (k, j) $\in$ A **e** j $\in$ Temporários) **fazer**

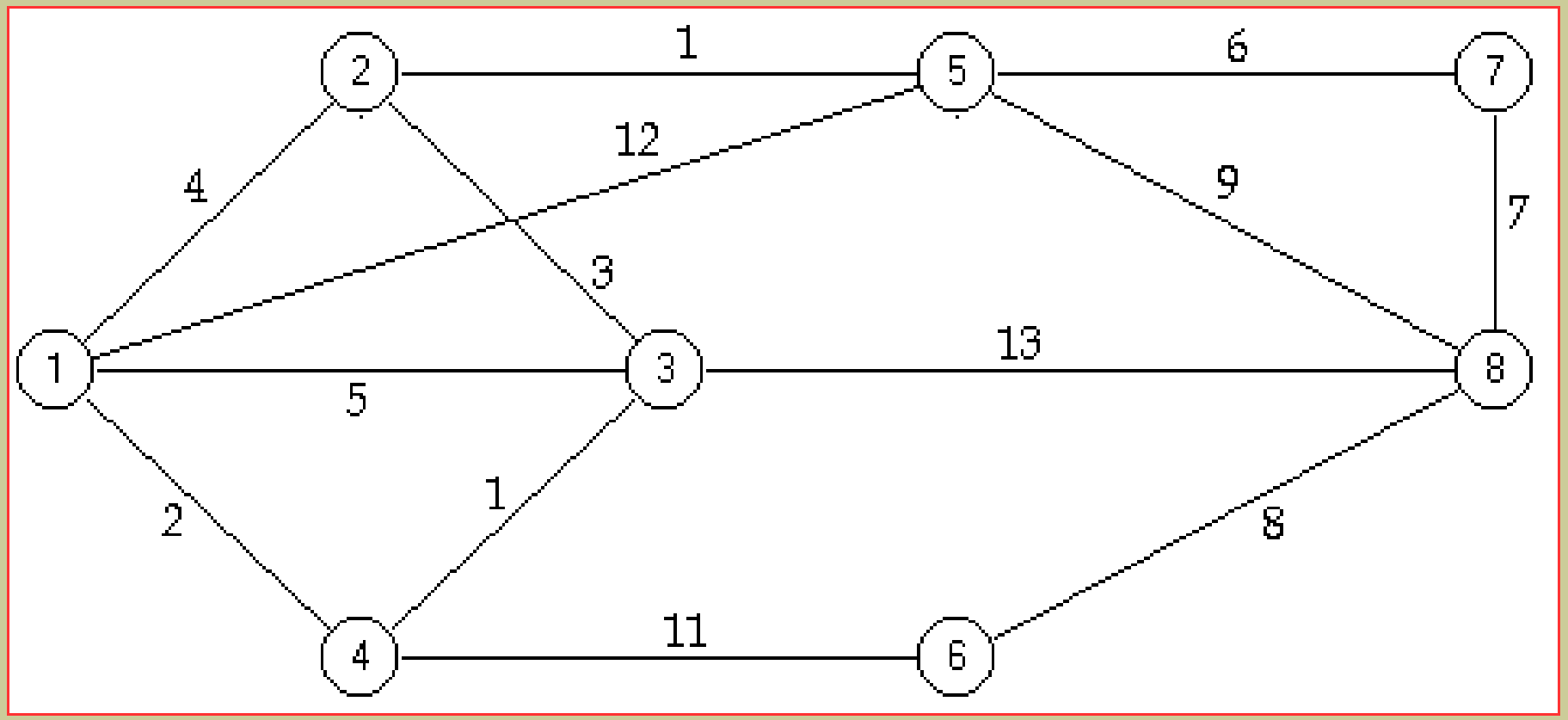
$R_j \leftarrow \min\{ R_j, C_{kj} \}$

fim_para

regressar ao Passo 2

Algoritmo de PRIM – exemplo

- Determinar a árvore abrangente mínima da seguinte rede



Algoritmo de PRIM – exemplo

Passo 1.

Tome-se o nó S = 1 e coloque-se um rótulo permanente nulo: $R_1 = 0$

Colocar rótulos temporários nos restantes nós:

$R_2 = 4$

$R_3 = 5$

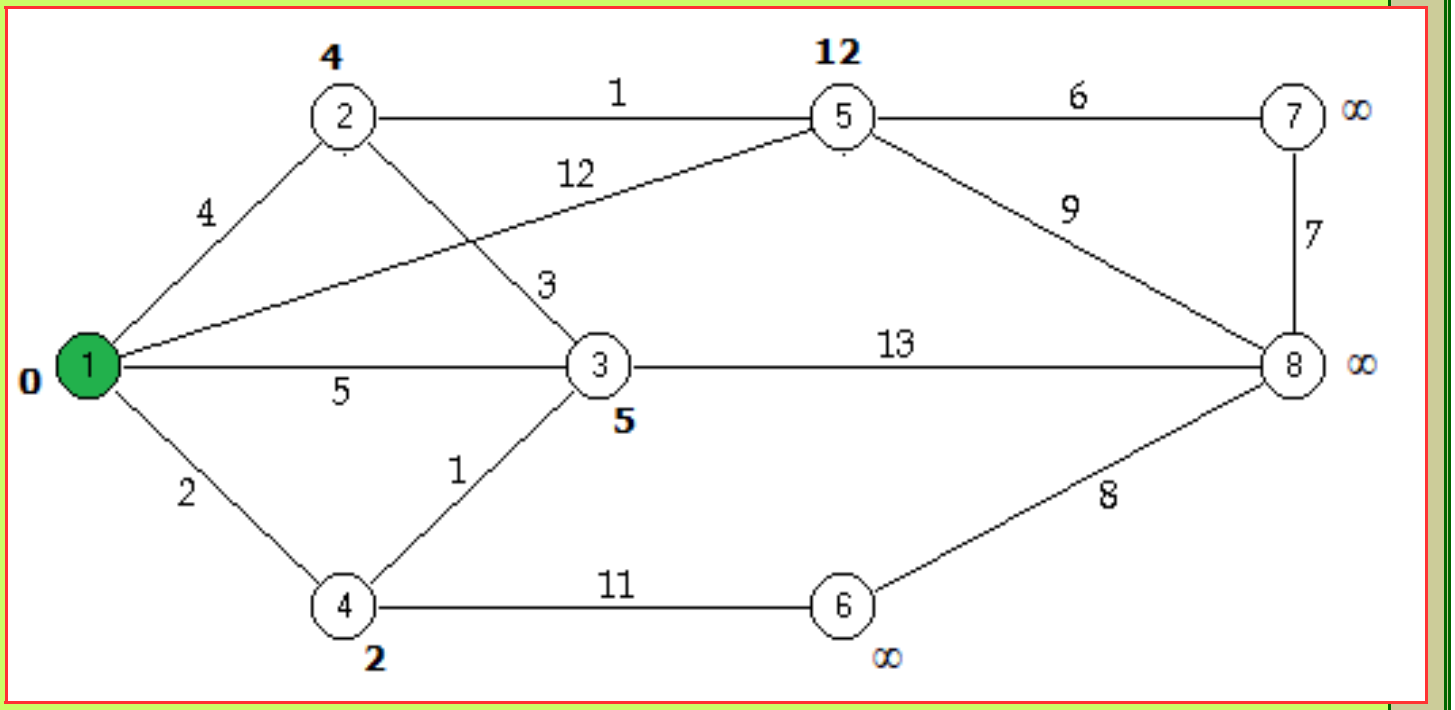
$R_4 = 2$

$R_5 = 12$

$R_6 = \infty$

$R_7 = \infty$

$R_8 = \infty$



Permanentes = { 1 }

Temporários = { 2, 3, 4, 5, 6, 7, 8 }

Nota: os nós com rótulo permanente são pintados a **verde**

Algoritmo de PRIM – exemplo

Passo 2.

k = 4 (nó com menor rótulo entre os temporários)

Permanentes = Permanentes ∪ { 4 } = { 1, 4 }

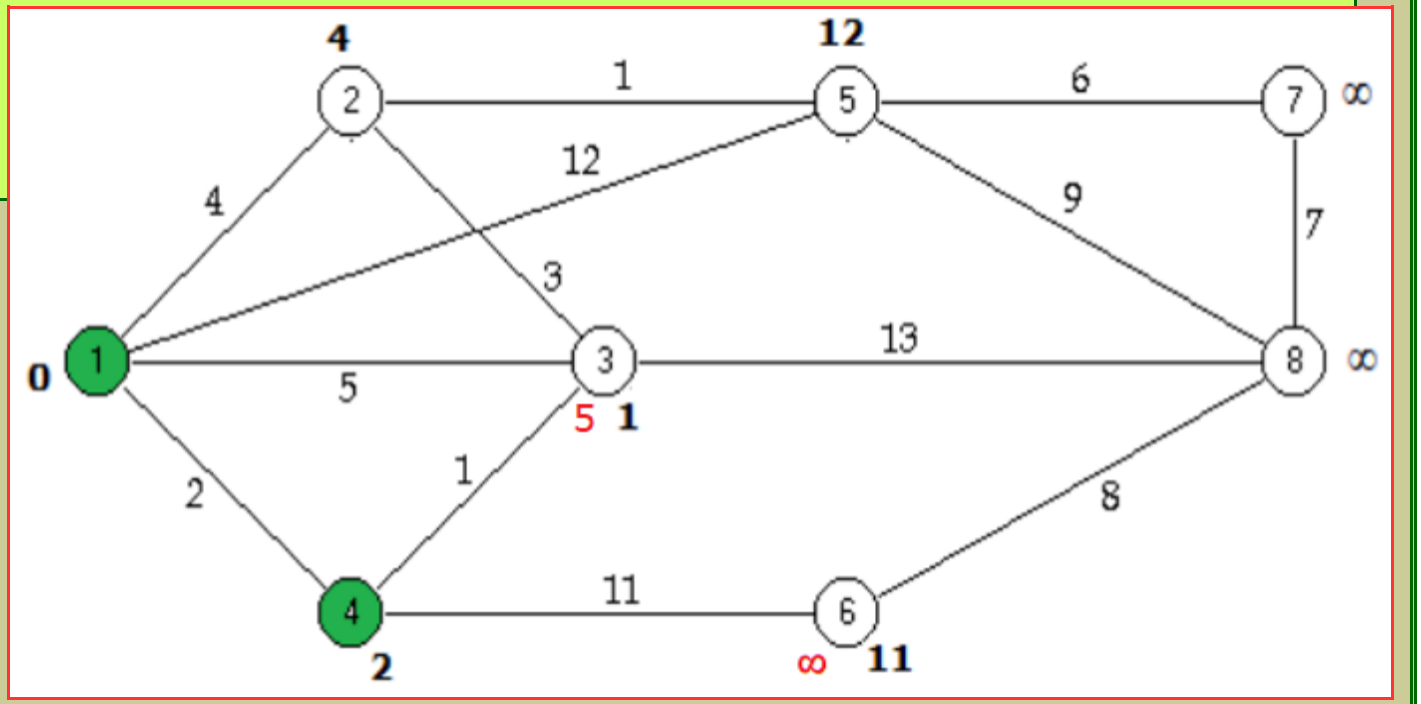
Temporários = Temporários - { 4 } = { 2, 3, 5, 6, 7, 8 }

O arco (1, 4) passa a fazer parte da árvore abrangente mínima, pois $C_{14} = R_4 = 2$

Passo 3.

$R_3 = \min \{ 5, 1 \} = 1$

$R_6 = \min \{ \infty, 11 \} = 11$



Algoritmo de PRIM – exemplo

Passo 2.

k = 3 (nó com menor rótulo entre os temporários)

Permanentes = Permanentes ∪ { 3 } = { 1, 4, 3 }

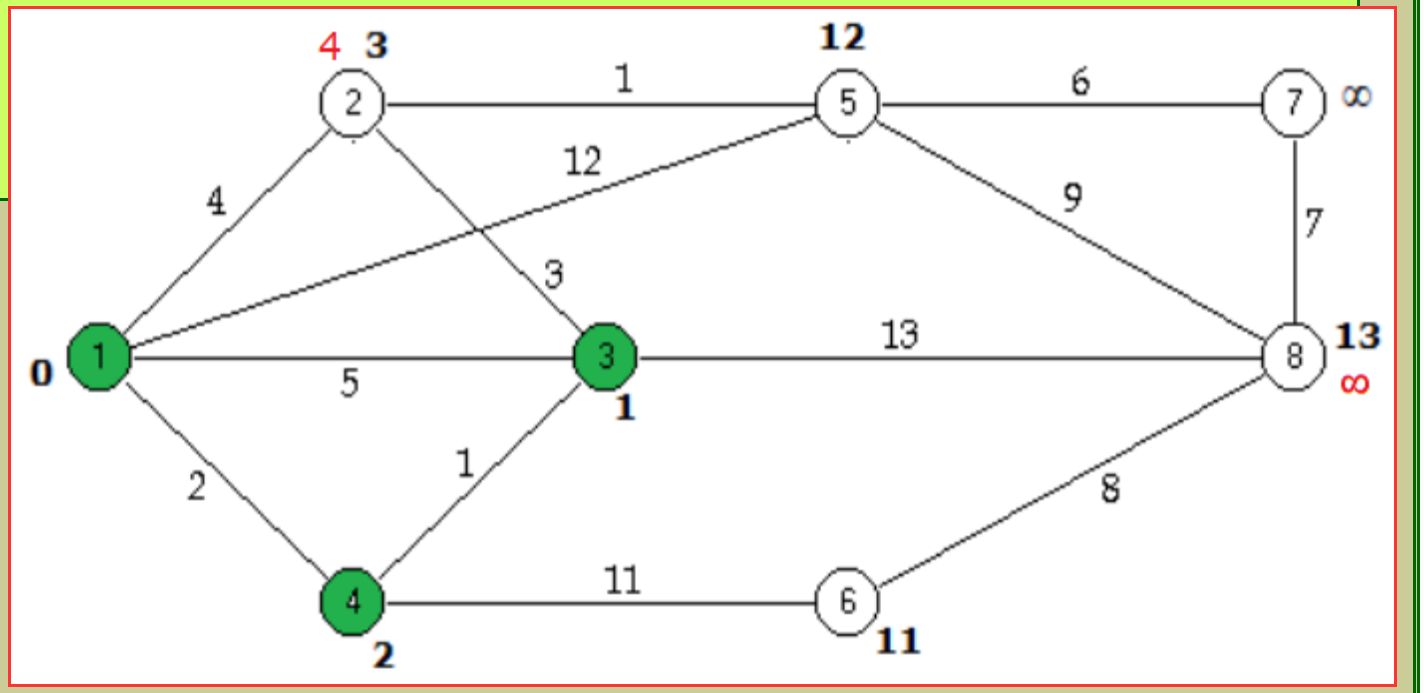
Temporários = Temporários - { 3 } = { 2, 5, 6, 7, 8 }

O arco (4, 3) passa a fazer parte da árvore abrangente mínima, pois $C_{43} = R_3 = 1$

Passo 3.

R_2 = min { 4, 3 } = 3

R_8 = min { ∞, 13 } = 13



Algoritmo de PRIM – exemplo

Passo 2.

k = 2 (nó com menor rótulo entre os temporários)

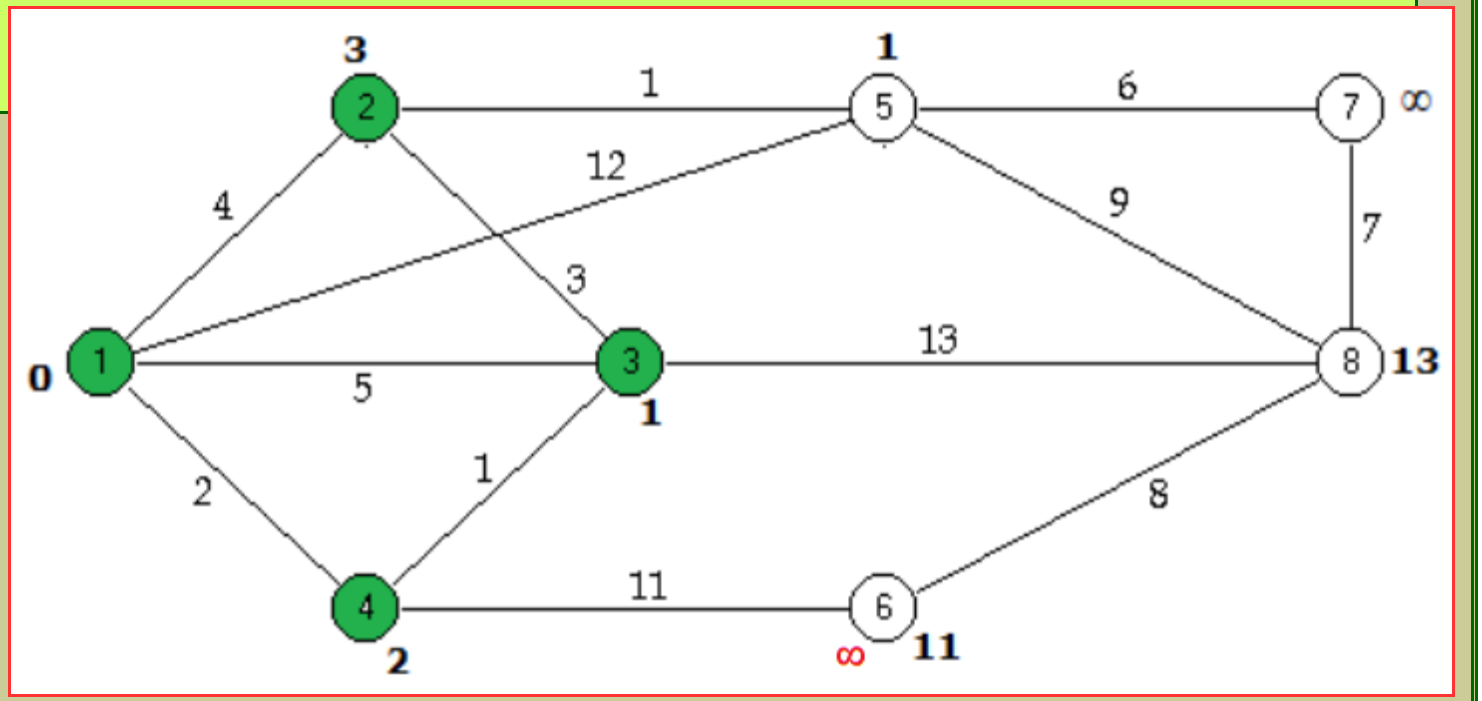
Permanentes = Permanentes ∪ { 2 } = { 1, 4, 3, 2 }

Temporários = Temporários - { 2 } = { 5, 6, 7, 8 }

O arco (3, 2) passa a fazer parte da árvore abrangente mínima, pois $C_{32} = R_2 = 3$

Passo 3.

$R_5 = \min \{ 12, 1 \} = 1$



Algoritmo de PRIM – exemplo

Passo 2.

k = 5 (nó com menor rótulo entre os temporários)

Permanentes = Permanentes ∪ { 5 } = { 1, 4, 3, 2, 5 }

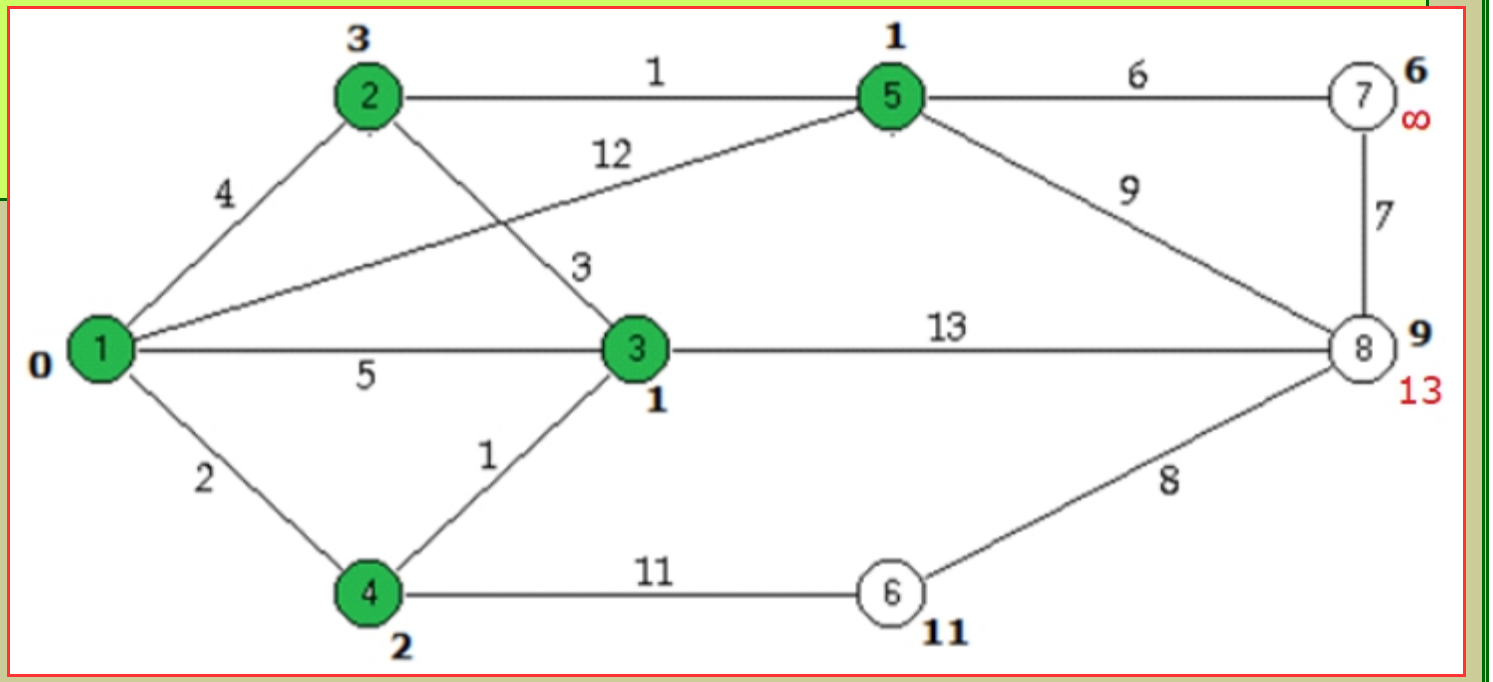
Temporários = Temporários - { 5 } = { 6, 7, 8 }

O arco (2, 5) passa a fazer parte da árvore abrangente mínima, pois $C_{25} = R_5 = 1$

Passo 3.

$R_7 = \min \{ \infty, 6 \} = 6$

$R_8 = \min \{ 13, 9 \} = 9$



Algoritmo de PRIM – exemplo

Passo 2.

k = 7 (nó com menor rótulo entre os temporários)

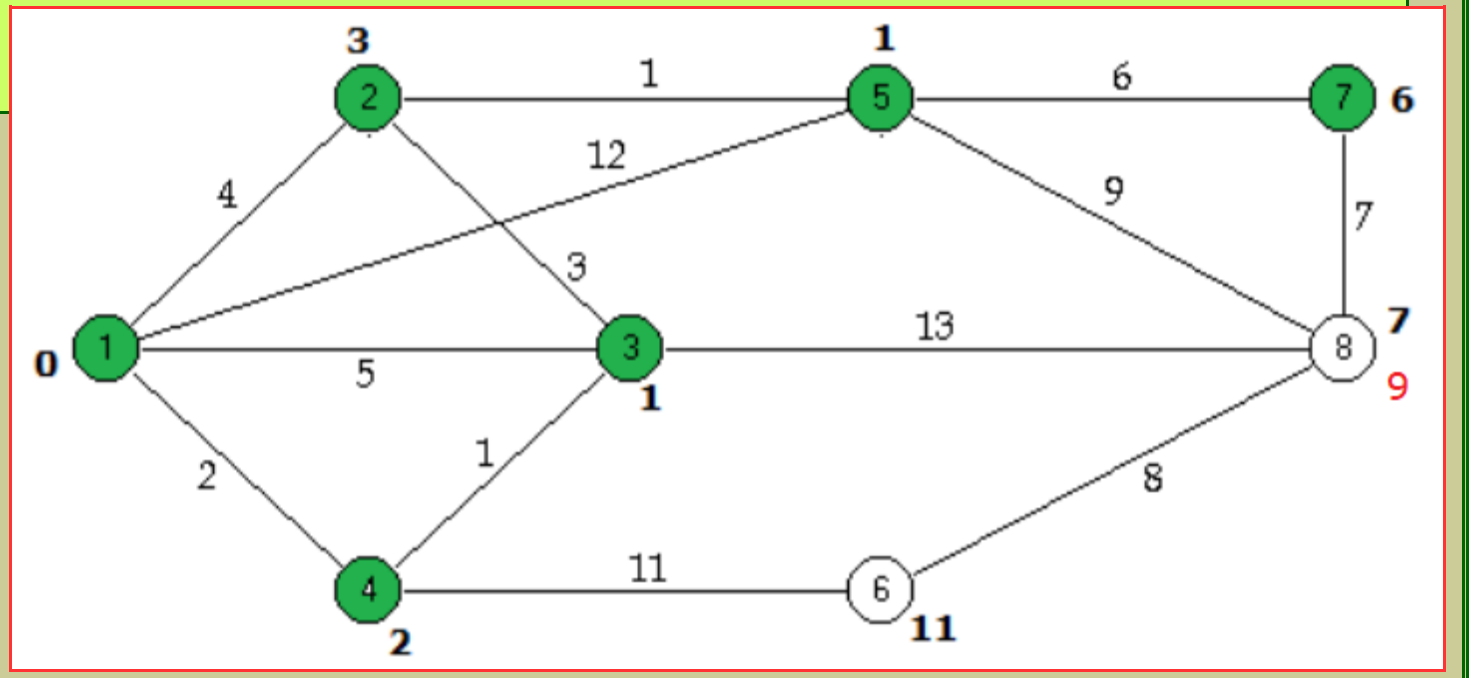
Permanentes = Permanentes ∪ { 6 } = { 1, 4, 3, 2, 5, 7 }

Temporários = Temporários - { 7 } = { 6, 8 }

O arco (5, 7) passa a fazer parte da árvore abrangente mínima, pois $C_{57} = R_7 = 6$

Passo 3.

$R_8 = \min \{ 9, 7 \} = 7$



Algoritmo de PRIM – exemplo

Passo 2.

k = 8 (nó com menor rótulo entre os temporários)

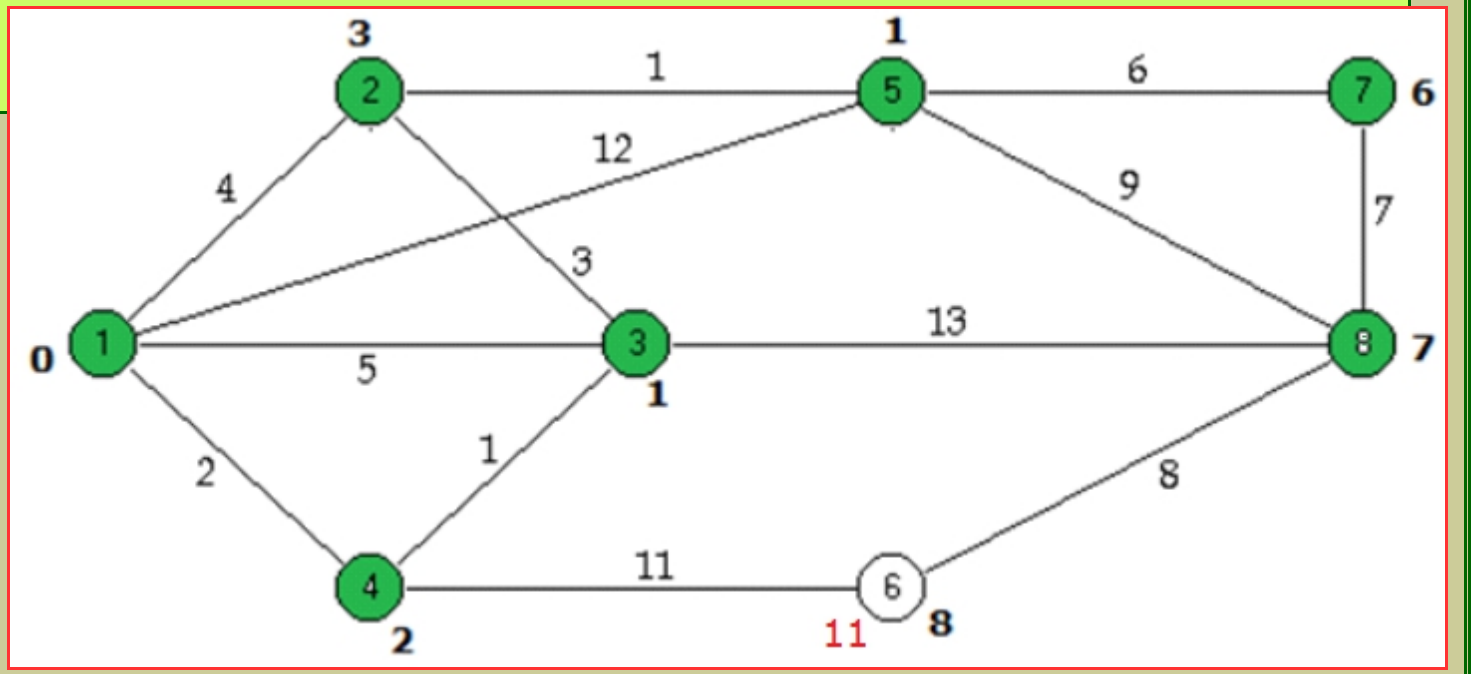
Permanentes = Permanentes ∪ { 8 } = { 1, 4, 3, 2, 5, 7, 8 }

Temporários = Temporários - { 8 } = { 6 }

O arco (7, 8) passa a fazer parte da árvore abrangente mínima, pois $C_{78} = R_8 = 7$

Passo 3.

R_6 = min { 11, 8 } = 8



Algoritmo de PRIM – exemplo

Passo 2.

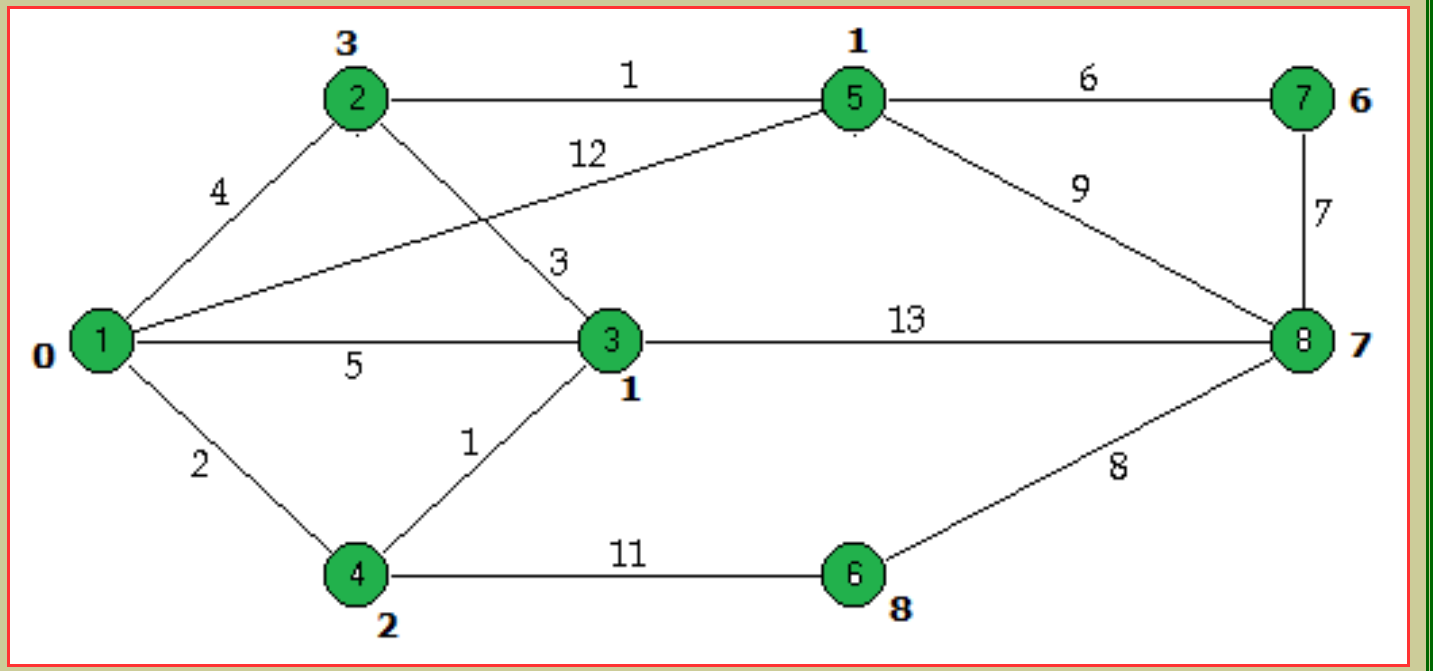
k = 6 (nó com menor rótulo entre os temporários)

Permanentes = Permanentes ∪ { 6 } = { 1, 4, 3, 2, 5, 7, 8, 6 }

Temporários = Temporários - { 6 } = ∅

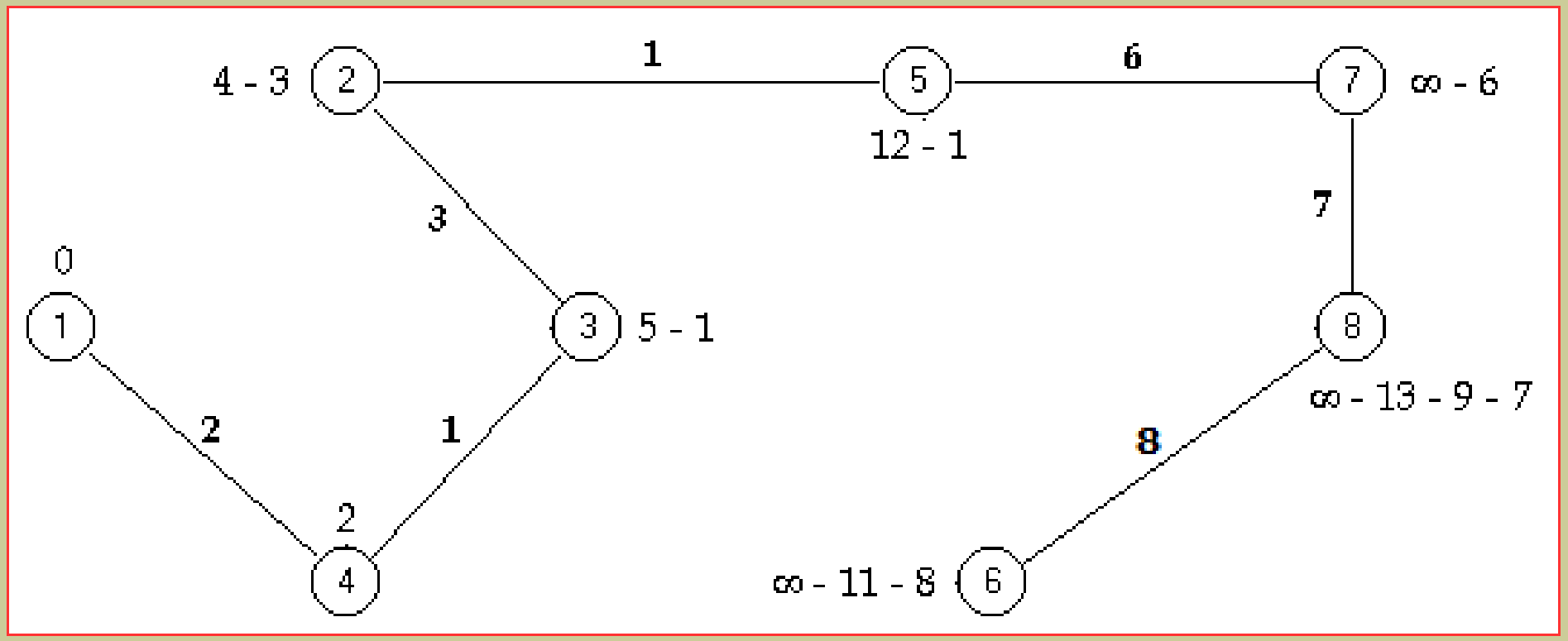
O arco (8, 6) passa a fazer parte da árvore abrangente mínima, pois $C_{86} = R_6 = 8$

Como Temporários = ∅, então **PARAR** (determinada *uma árvore abrangente mínima*)



Algoritmo de PRIM – exemplo

- Resultados após o final do algoritmo:
  - a árvore abrangente mínima (árvore que visita todos os nós) é a seguinte:



- comprimento total da árvore abrangente mínima:

$C_{12} + C_{23} + C_{32} + C_{25} + C_{57} + C_{78} + C_{86} = 2 + 1 + 3 + 1 + 6 + 7 + 8 = 28$

O problema do Caminho Mais Curto

Conceitos gerais

- O problema do Caminho Mais Curto ("Shortest Path Problem") é um modelo matemático fundamental e frequentemente usado quando se pretende estudar redes de transportes e de comunicação
- Este problema surge quando se pretende determinar o caminho mais curto, mais barato ou mais fiável, entre dois ou vários pares de nós de uma rede
- Existem três tipos de problemas de caminho mais curto:
  (1) de um nó para outro,
  (2) de um nó para todos os outros,
  (3) entre todos os pares de nós
- No entanto, os dois primeiros são essencialmente o mesmo problema:
  - o problema (1) é um caso particular do problema (2)

Conceitos gerais

- Sejam S e T dois nós de uma rede G = (N, A, C) e P o conjunto de todos os caminhos de S para T em G
- O comprimento de um caminho de S para T é a soma dos comprimentos dos arcos que o compõem
- O problema do caminho mais curto de S para T tem por objetivo determinar o caminho de valor mínimo existente em P:
  - determinar o caminho $p \in P$ tal que $C(p) \leq C(q), \forall\ q \in P$
- Considere-se algumas observações relacionadas com este tipo de problemas:
  - o comprimento de um caminho é maior do que o de qualquer dos seus subcaminhos
  - qualquer subcaminho de um caminho mais curto é ele próprio um caminho mais curto (princípio da otimalidade)
  - para uma rede com **n** nós, qualquer caminho mais curto tem no máximo **n-1** arcos (no caminho mais curto entre dois nós, não existem nós repetidos)

Conceitos gerais

- Matematicamente, este problema pode ser formulado da seguinte forma:

$$\text{minimizar } Z = \sum_{i \in N} \sum_{j \in N} C_{ij} X_{ij}$$

sujeito a

$$\sum_{j \in N} X_{Sj} = 1$$

$$\sum_{i \in N} X_{ij} - \sum_{k \in N} X_{jk} = 0, \quad \forall j \in N - \{S, T\}$$

$$\sum_{i \in N} X_{iT} = 1$$

em que,

$$X_{ij} = \begin{cases} 1, & \text{se } (i,j) \text{ pertence ao caminho} \\ 0, & \text{se } (i,j) \text{ não pertence ao caminho} \end{cases}$$

- Vários algoritmos eficientes para resolver este problema, sendo os mais conhecidos: Dijkstra (1 e 2) e Floyd (3)

Algoritmo de Dijkstra

- Só pode ser aplicada a redes cujos arcos têm comprimentos não negativos
- Baseia-se num processo de
  - rotulação dos nós da rede e
  - classificação dos respetivos rótulos
- A cada nó **i** é atribuído um rótulo $[Q_i, R_i]$ permanente ou temporário:
  - $[Q_i, R_i]$ permanente, representa o caminho mais curto de S para **i**

    $Q_i \leftarrow$ nó que antecede **i** no caminho mais curto de S para **i**

    $R_i \leftarrow$ valor do caminho mais curto de S para **i**
  - $[Q_i, R_i]$ temporário, representa um caminho mais curto de **S** para **i**

    $Q_i \leftarrow$ nó que antecede **i** no melhor caminho, até ao momento, de **S** para **i**

    $R_i \leftarrow$ valor do melhor caminho, até ao momento, de **S** para **i**
  - O rótulo temporário de um nó representa um limite superior da distância mais curta de **S** a esse nó, pois o caminho que lhe está associado pode ser ou não o mais curto

Algoritmo de Dijkstra

- O algoritmo consiste em rotular os nós da rede, começando pelo S, de uma forma ordenada, segundo as distâncias de cada nó a S:
  - escolher o nó com rótulo temporário com menor valor de R (que se torna permanente),
  - depois, varrer todos os nós seus adjacentes que sejam temporários, de forma a atualizar os rótulos destes nós
- O algoritmo termina quando não existirem nós com rótulos temporários
- Inicialmente apenas o nó S é permanente, sendo os restantes temporários

Algoritmo de Dijkstra

Seja G = (N, A, C) uma rede

Passo 1.

$[Q_S, R_S] \leftarrow [S, 0]$ (caminho mais curto para S custa 0 e não tem nós intermédios)

$[Q_i, R_i] \leftarrow [S, C_{Si}]$, $\forall\ i \in N - \{ S \}$ e $(S, i) \in A$

$[Q_i, R_i] \leftarrow [S, \infty]$, $\forall\ i \in N - \{ S \}$ e $(S, i) \notin A$

Temporários $\leftarrow N - \{ S \}$ (Temporários = conjunto de nós com rótulos temporários)

Permanentes $\leftarrow \{ S \}$ (Permanentes = conjunto de nós com rótulos permanentes)

Algoritmo de Dijkstra

Passo 2.

se (Temporários = $\emptyset$) **então**

PARAR (todos os nós têm rótulos permanentes)

fim_se

k $\leftarrow$ nó de Temporários tal que R_k é mínimo ($k : R_k = \min \{ R_x, x \in \text{Temporários} \}$)

Temporários $\leftarrow$ Temporários - { k }

Permanentes $\leftarrow$ Permanentes $\cup$ { k } (k passou a permanente)

Passo 3.

para (todo o **j** $\in$ N tal que (k, j) $\in$ A **e** j $\in$ Temporários) **fazer**

se ($R_k + C_{kj} < R_j$) **então**

$R_j \leftarrow R_k + C_{kj}$

$Q_j \leftarrow k$

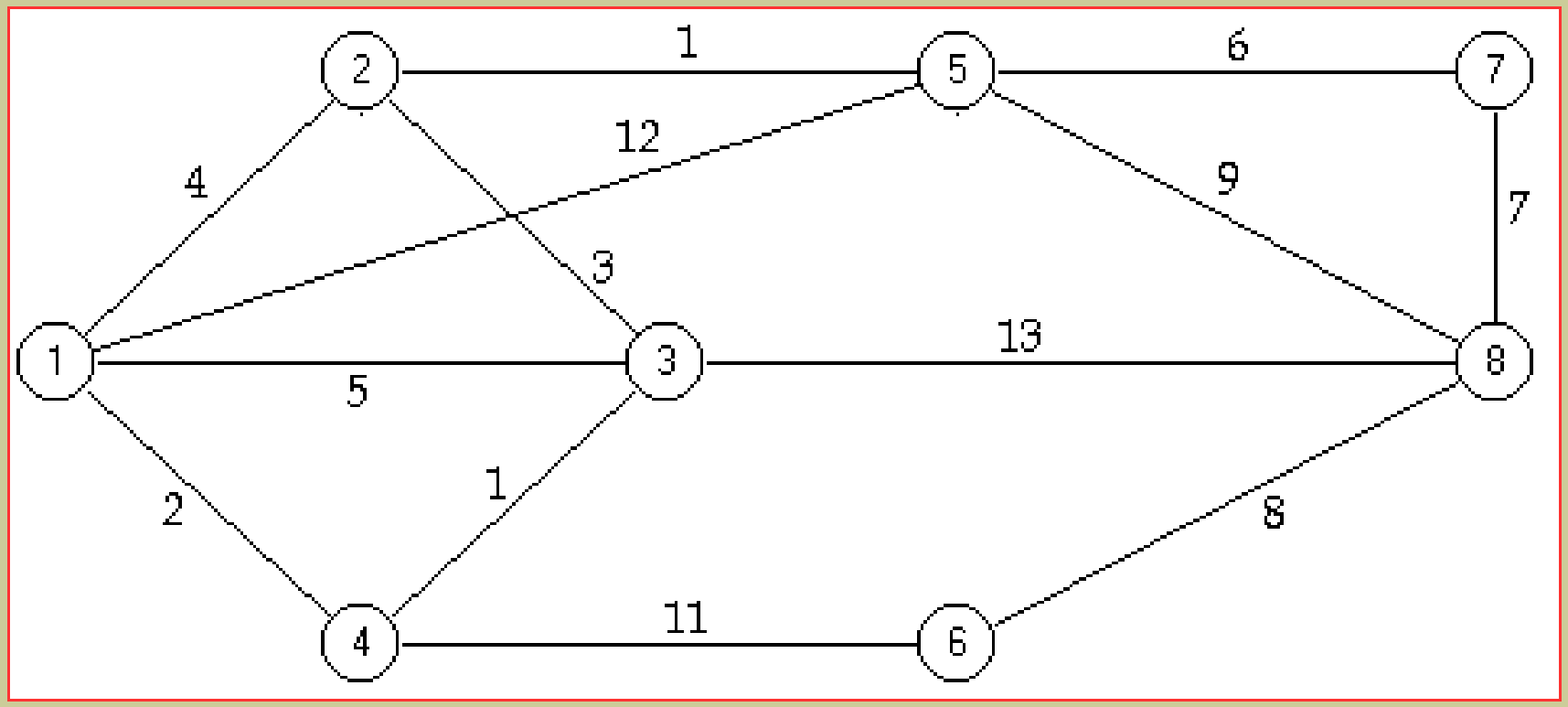
fim_se

fim_para

regressar ao **Passo 2**

Algoritmo de Dijkstra - exemplo

- Determinar o caminho mais curto entre o nó S = 1 e todos os outros nós da seguinte rede:



Algoritmo de Dijkstra - exemplo

Passo 1.

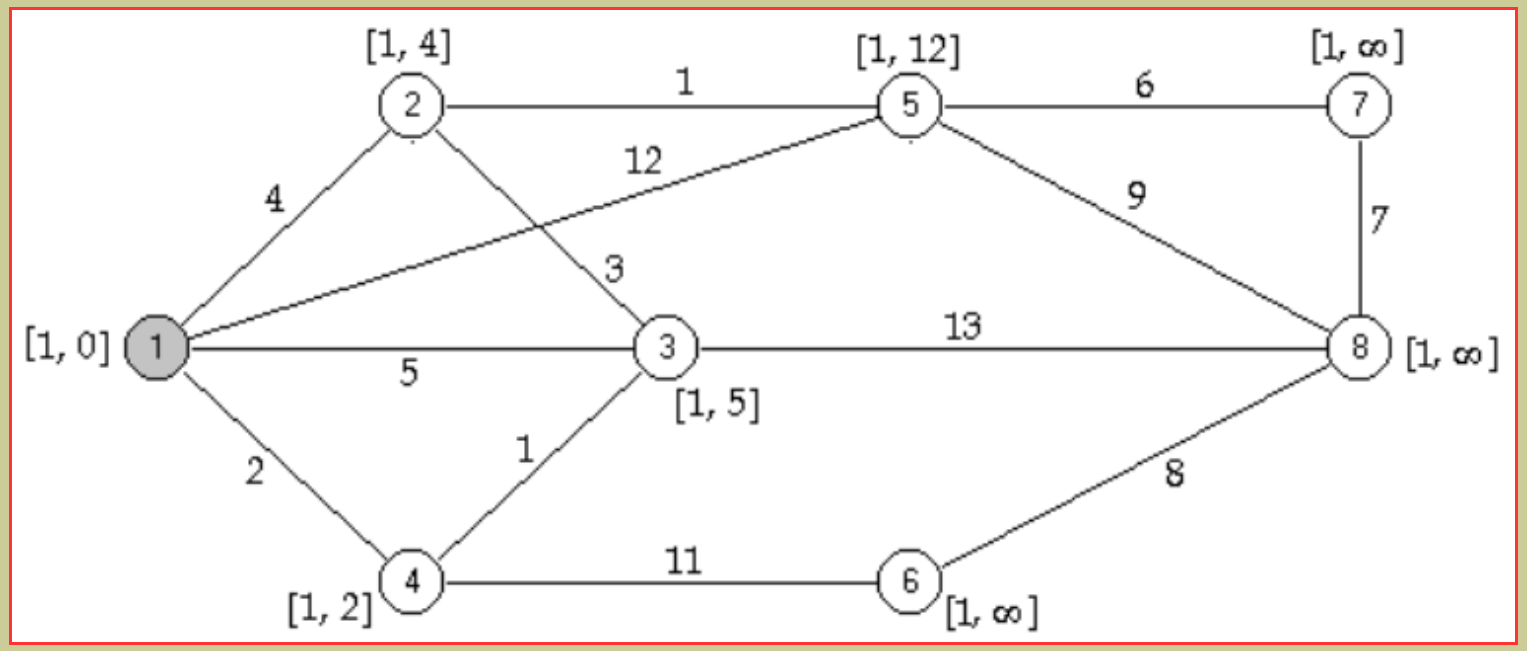
$[Q_1, R_1] = [1, 0]$

$[Q_i, R_i] = [1, C_{Si}], \forall\ i \in \{ 2, 3, 4, 5 \}\ (N - \{ 1 \}\ e\ (1, i) \in A)$

$[Q_i, R_i] = [1, \infty], \forall\ i \in \{ 6, 7, 8\}\ (N - \{ 1 \}\ e\ (1, i) \notin A)$

Permanentes = { 1 }

Temporários = { 2, 3, 4, 5, 6, 7, 8 }



Algoritmo de Dijkstra - exemplo

Passo 2.

k = 4, pois R_4 = min { R_2, R_3, R_4, R_5, R_6, R_7, R_8 } = min { 4, 5, 2, 12, ∞, ∞, ∞ }

Permanentes = Permanentes ∪ { 4 } = { 1, 4 }

Temporários = Temporários - { 4 } = { 2, 3, 5, 6, 7, 8 }

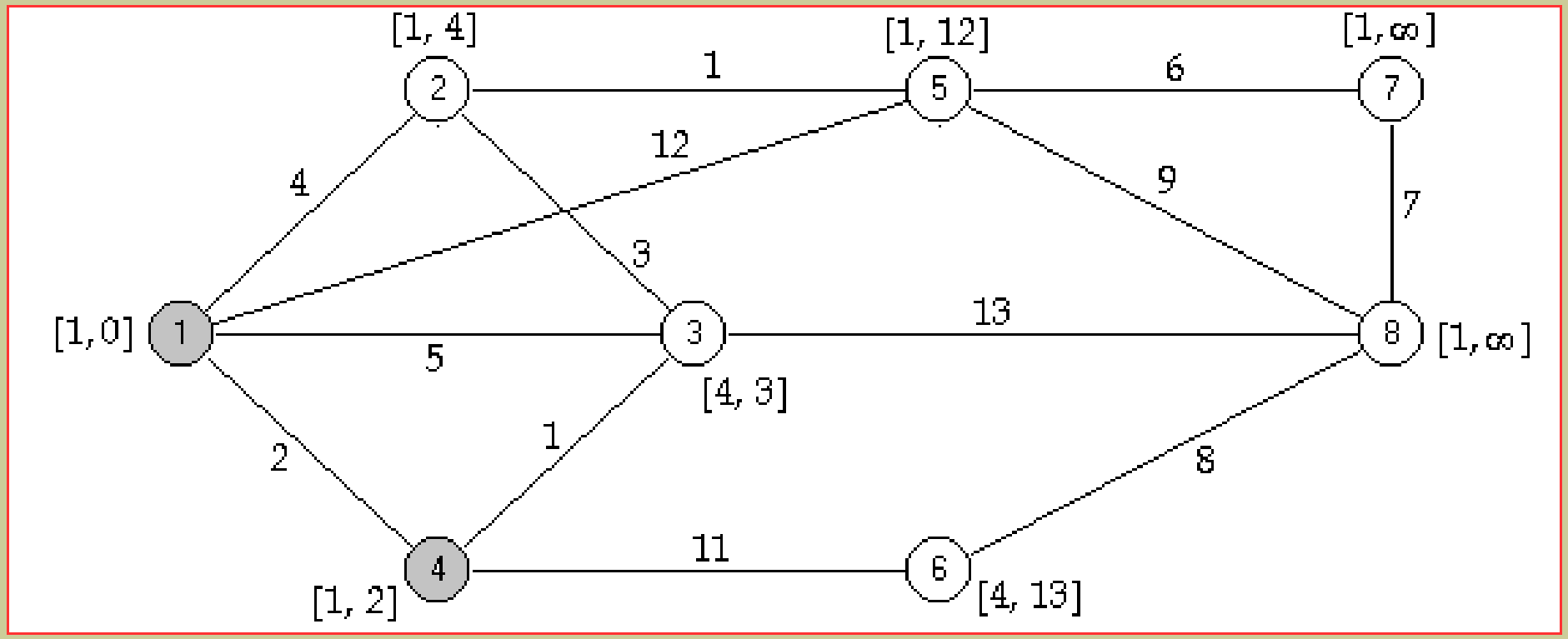
Algoritmo de Dijkstra - exemplo

Passo 3.

Varrer todos os nós adjacentes a 4, com rótulos temporários e atualizar os seus rótulos:

$R_3 = \min \{ R_3, R_4 + C_{43} \} = \min \{ 5, 2 + 1 \} = 3 \quad \Rightarrow \quad [Q_3, R_3] = [4, 3]$

$R_6 = \min \{ R_6, R_4 + C_{46} \} = \min \{ \infty, 2 + 11 \} = 13 \quad \Rightarrow \quad [Q_6, R_6] = [4, 13]$



Algoritmo de Dijkstra - exemplo

Passo 2.

k = 3, pois $R_3 = \min \{ R_2, R_3, R_5, R_6, R_7, R_8 \} = \min \{ 4, 3, 12, 13, \infty, \infty \}$

Permanentes = Permanentes $\cup$ { 3 } = { 1, 3, 4 }

Temporários = Temporários - { 4 } = { 2, 5, 6, 7, 8 }

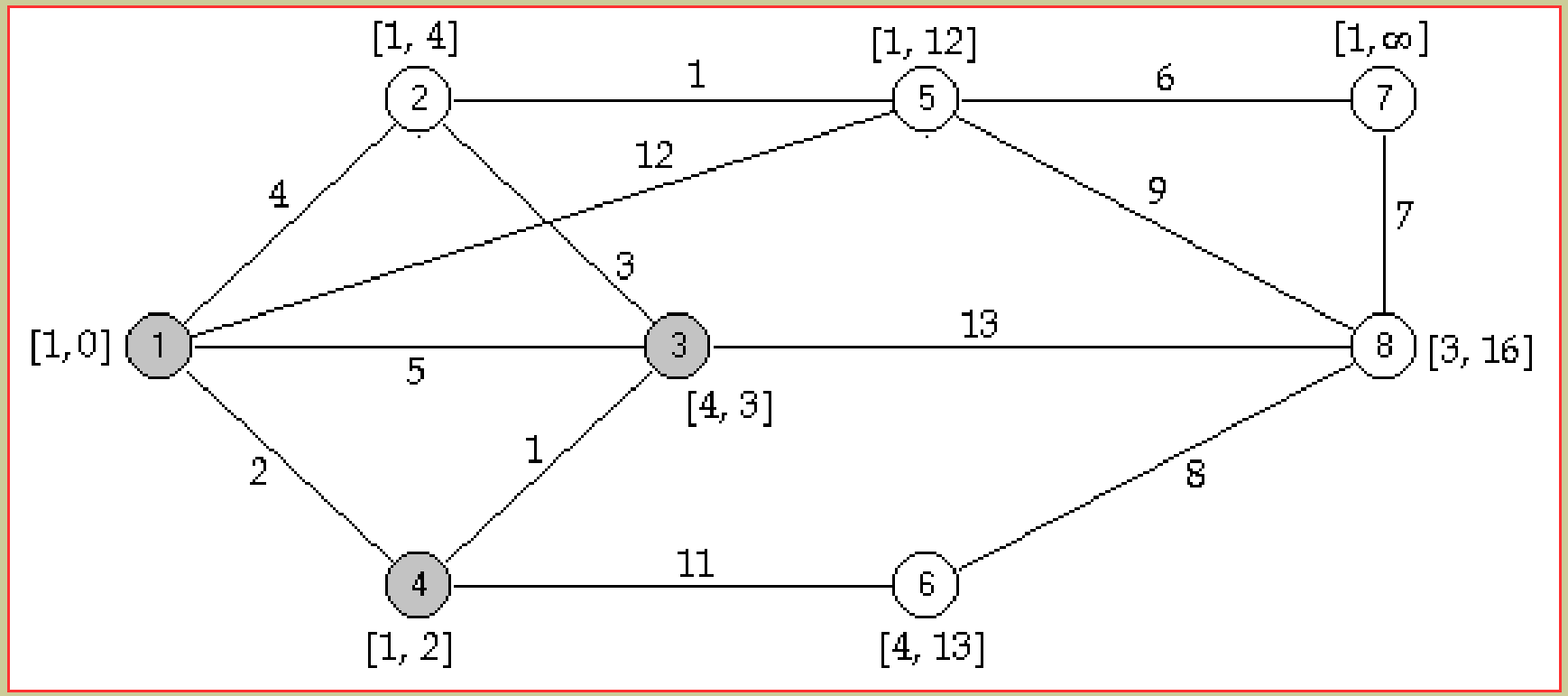
Algoritmo de Dijkstra - exemplo

Passo 3.

Varrer todos os nós adjacentes a 3, com rótulos temporários e atualizar os seus rótulos:

$R_2 = \min \{ R_2, R_3 + C_{32} \} = \min \{ 4, 3 + 3 \} = 4 \quad \Rightarrow \quad [Q_2, R_2] = [1, 4]$ (*mantém*)

$R_8 = \min \{ R_8, R_3 + C_{38} \} = \min \{ \infty, 3 + 13 \} = 16 \quad \Rightarrow \quad [Q_8, R_8] = [3, 16]$



Algoritmo de Dijkstra - exemplo

Passo 2.

k = 2, pois R_2 = min { R_2, R_5, R_6, R_7, R_8 } = min { 4, 12, 13, ∞, 16 }

Permanentes = Permanentes $\cup$ { 2 } = { 1, 2, 3, 4 }

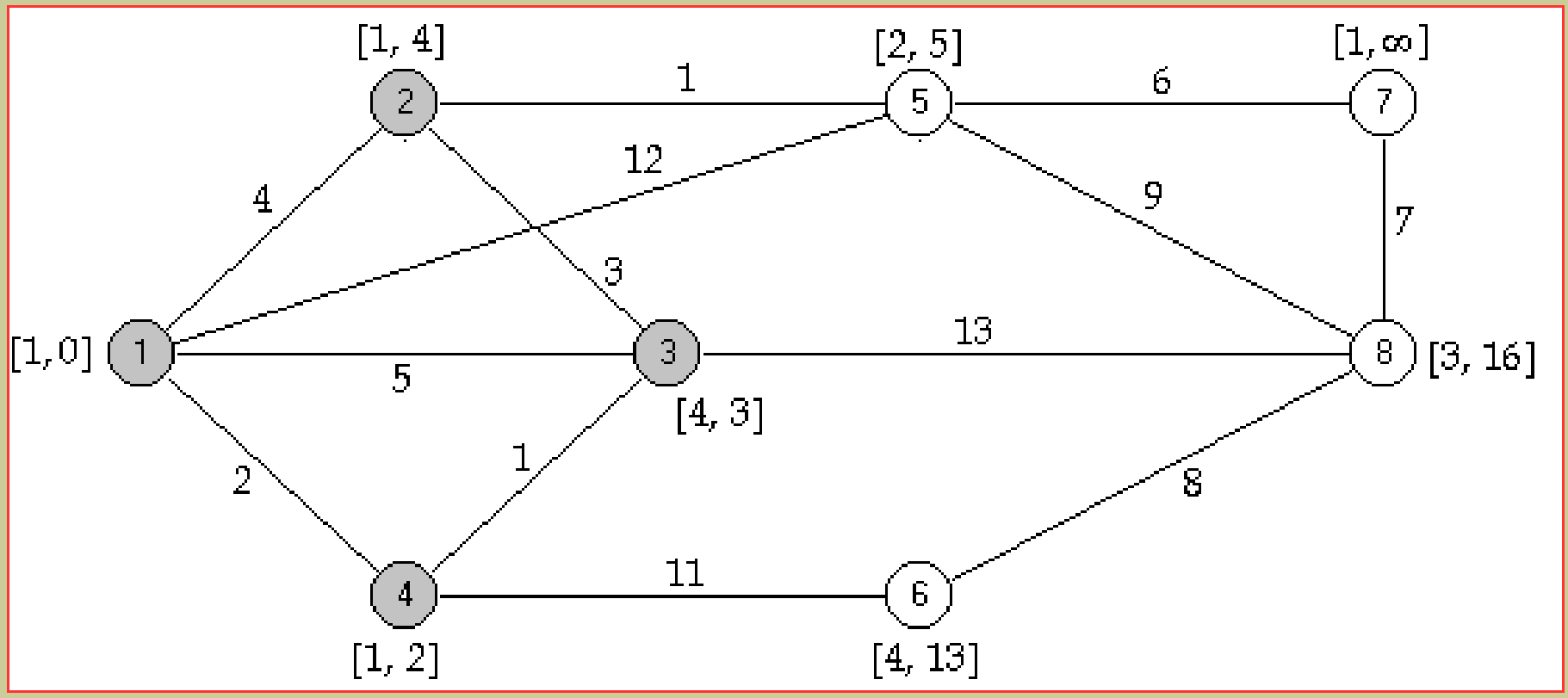
Temporários = Temporários - { 2 } = { 5, 6, 7, 8 }

Algoritmo de Dijkstra - exemplo

Passo 3.

Varrer todos os nós adjacentes a 2, com rótulos temporários e atualizar os seus rótulos:

$R_5 = \min \{ R_5, R_2 + C_{25} \} = \min \{ 12, 4 + 1 \} = 5 \Rightarrow [Q_5, R_5] = [2, 5]$



Algoritmo de Dijkstra - exemplo

Passo 2.

k = 5, pois $R_5 = \min \{ R_5, R_6, R_7, R_8 \} = \min \{ 5, 13, \infty, 16 \}$

Permanentes = Permanentes $\cup$ { 5 } = { 1, 2, 3, 4, 5 }

Temporários = Temporários - { 5 } = { 6, 7, 8 }

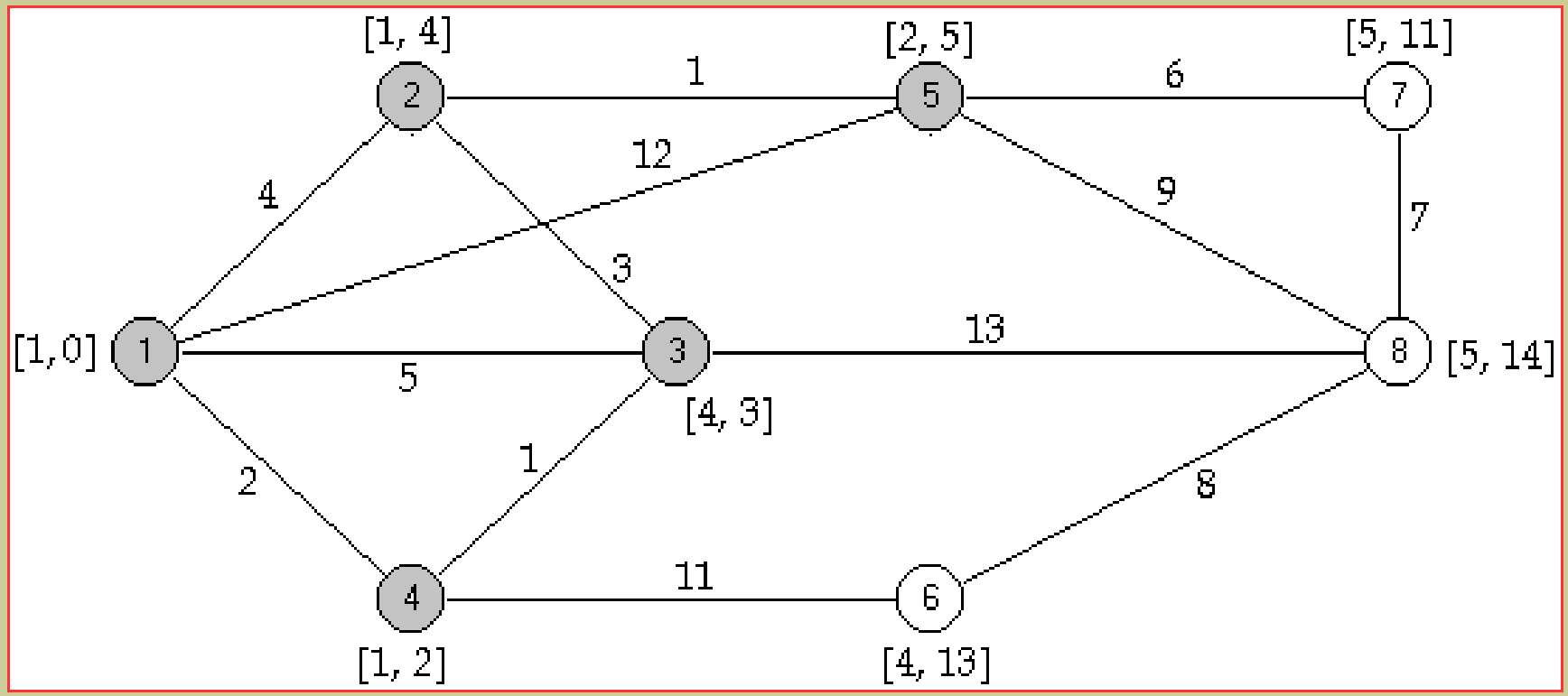
Algoritmo de Dijkstra - exemplo

Passo 3.

Varrer todos os nós adjacentes a 5, com rótulos temporários e atualizar os seus rótulos:

$R_7 = \min \{ R_7, R_5 + C_{57} \} = \min \{ \infty, 5 + 6 \} = 11 \quad \Rightarrow \quad [Q_7, R_7] = [5, 11]$

$R_8 = \min \{ R_8, R_5 + C_{58} \} = \min \{ 16, 5 + 9 \} = 14 \quad \Rightarrow \quad [Q_8, R_8] = [5, 14]$



Algoritmo de Dijkstra - exemplo

Passo 2.

k = 7, pois R_7 = min { R_6, R_7, R_8 } = min { 13, 11, 14 }

Permanentes = Permanentes $\cup$ { 7 } = { 1, 2, 3, 4, 5, 7 }

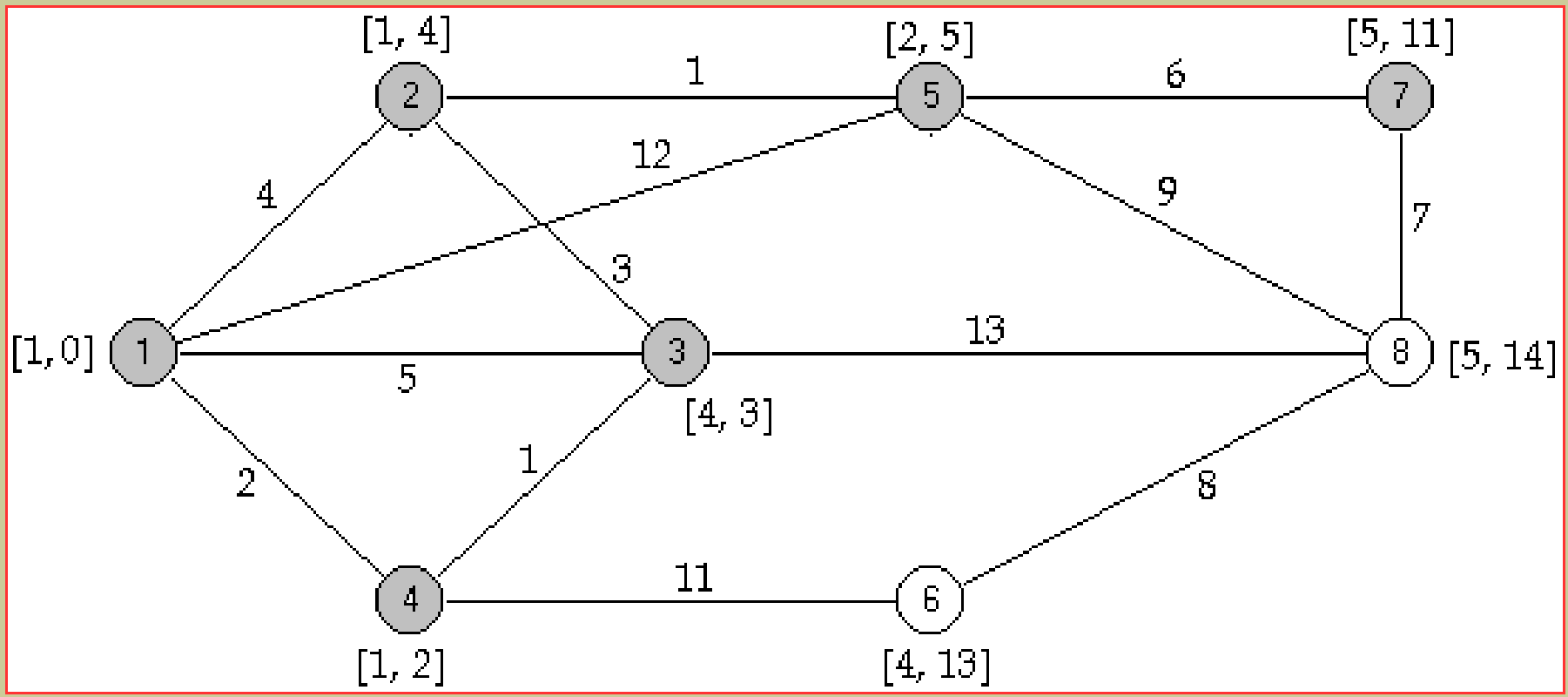
Temporários = Temporários - { 7 } = { 6, 8 }

Algoritmo de Dijkstra - exemplo

Passo 3.

Varrer todos os nós adjacentes a 7, com rótulos temporários e atualizar os seus rótulos:

$R_8 = \min \{ R_8, R_7 + C_{78} \} = \min \{ 14, 11 + 7 \} = 14 \quad \Rightarrow \quad [Q_8, R_8] = [5, 14]$ (*mantém*)



Algoritmo de Dijkstra - exemplo

Passo 2.

k = 6, pois R_6 = min { R_6, R_8 } = min { 13, 14 }

Permanentes = Permanentes $\cup$ { 6 } = { 1, 2, 3, 4, 5, 6, 7 }

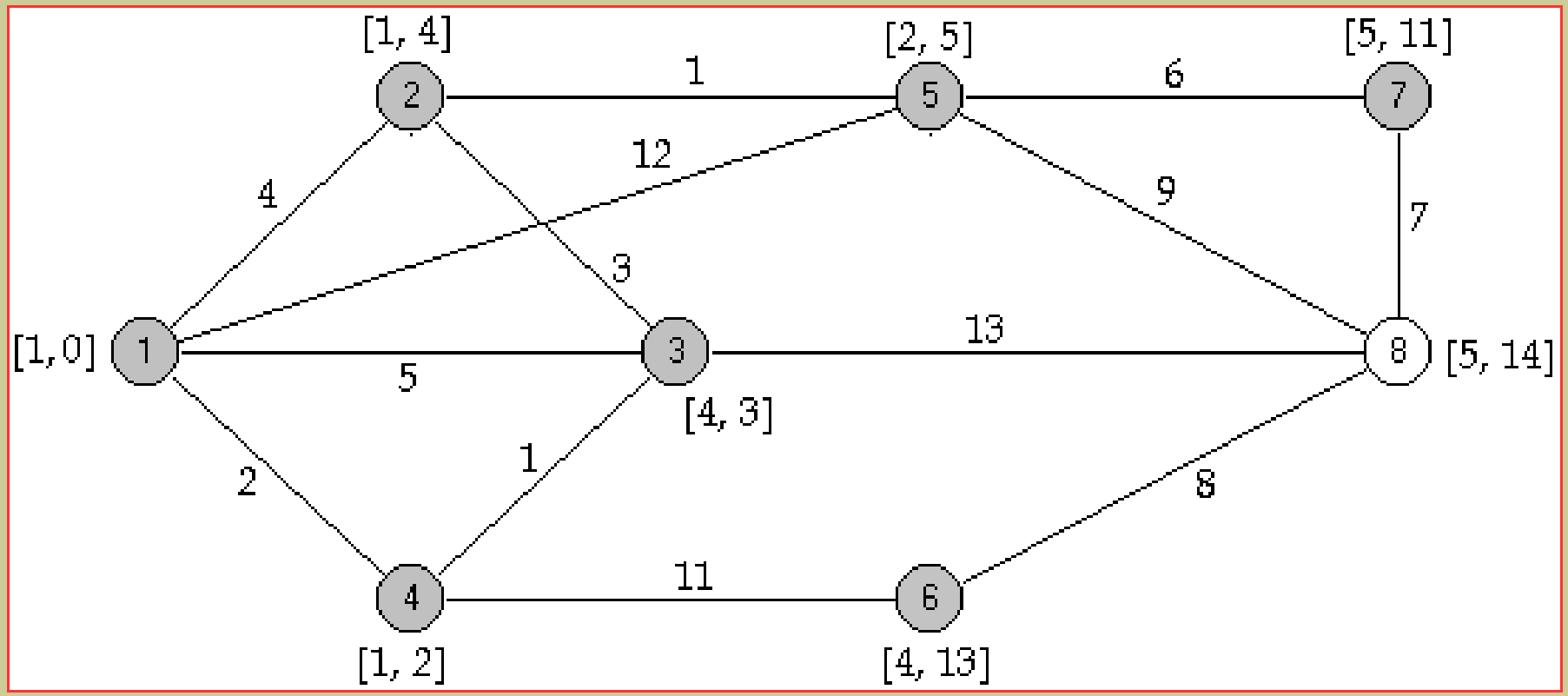
Temporários = Temporários - { 6 } = { 8 }

Algoritmo de Dijkstra - exemplo

Passo 3.

Varrer todos os nós adjacentes a 6, com rótulos temporários e atualizar os seus rótulos:

$R_8 = \min \{ R_8, R_6 + C_{68} \} = \min \{ 14, 13 + 8 \} = 14 \quad \Rightarrow \quad [Q_8, R_8] = [5, 14]$ (*mantém*)



Algoritmo de Dijkstra - exemplo

Passo 2.

k = 8, pois $R_8 = \min \{ R_8 \} = \min \{ 14 \}$

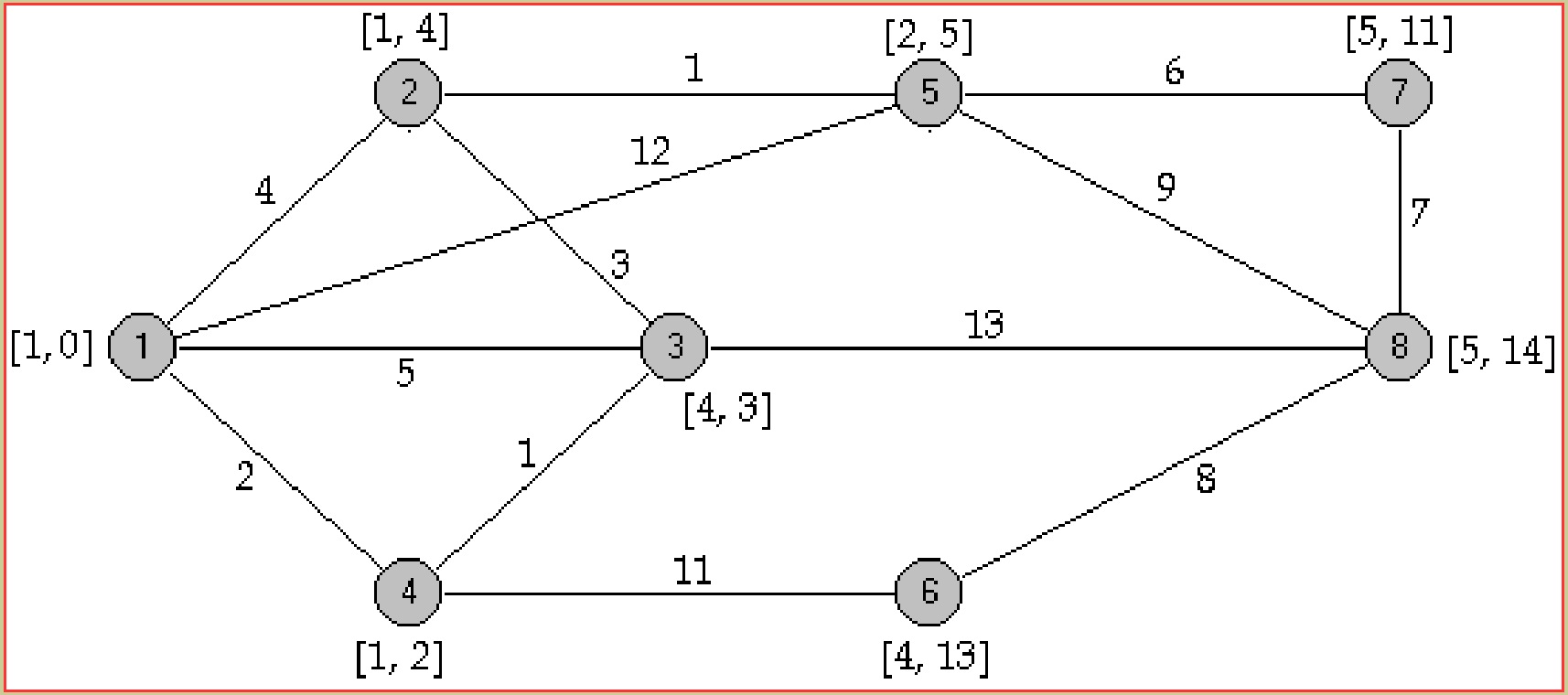
Permanentes = Permanentes $\cup$ { 8 } = { 1, 2, 3, 4, 5, 6, 7, 8 }

Temporários = Temporários - { 8 } = Ø

Algoritmo de Dijkstra - exemplo

Passo 3.

Não existem nós adjacentes a 8 com rótulos temporários.



Passo 2.

Como Temporários = Ø **então PARAR**

Algoritmo de Dijkstra - exemplo

- Resultados após o término do algoritmo:
  - caminho mais curto entre os nós 1 e 8 calcula-se da seguinte forma:

    $i = 8;\ \ p = \{ 8 \}$
  - como $i \neq 1$ então

    $i = Q_i = Q_8 = 5;\ \ p = p \cup \{ i \} = \{ 5, 8 \}$
  - como $i \neq 1$ então

    $i = Q_i = Q_5 = 2;\ \ p = p \cup \{ i \} = \{ 2, 5, 8 \}$
  - como $i \neq 1$ então

    $i = Q_i = Q_2 = 1;\ \ p = p \cup \{ i \} = \{ 1, 2, 5, 8 \}$
  - como $i = 1$ então

    termina o processo;
  - assim:
    - o caminho mais curto entre 1 e 8 é $p = \{ 1, 2, 5, 8 \}$ e
    - o comprimento é $C(p) = 14\ (= R_8)$

Algoritmo de Floyd

- Para determinar o caminho mais curto **entre todos os pares** de nós,
  - aplicar o algoritmo de Dijkstra **n** vezes, usando cada nó sucessivamente, como origem
  - aplicar outros algoritmos para resolver este problema:
    - o **algoritmo de Floyd**, desde que não haja circuitos negativos (os arcos podem ter comprimentos negativos)
- Considere-se uma rede G = (N, A, C),
  - um arco (i, j) designa-se de **arco básico** se formar o caminho mais curto entre os nós **i** e **j**
  - um caminho mais curto entre quaisquer dois nós da rede será totalmente constituído por arcos básicos (embora existam arcos básicos não pertencentes a este caminho)
- Algoritmo de Floyd
  - utiliza uma matriz D, de ordem **n**, das distâncias diretas mais curtas entre os nós
    - os nós não adjacentes têm associado uma distância direta infinita
    - como os nós são (por convenção) adjacentes com eles próprios, têm distância direta **0**

Algoritmo de Floyd

- Entre cada par de nós não ligados por um arco básico é criado um arco, através de um processo identificado por "tripla ligação":

  $d_{ij} \leftarrow \min \{ d_{ij} , d_{ik} + d_{kj} \}$

- Fixando um **k**, a "tripla ligação" é efetuada para todos os nós i, $j \neq k$
- Após efetuar a "tripla ligação" para cada $k \in N$ com $i, j \in N - \{ k \}$,
  - a rede (matriz D alterada ao longo do processo) é apenas constituída por arcos básicos,
  - logo, o comprimento associado a cada arco dirigido do nó **i** ao nó **k** é o caminho mais curto entre aqueles dois nós
- Para conhecer todos os nós intermédios num dado caminho, usa-se paralelamente uma matriz **P**, de ordem **n**, onde o elemento $\mathbf{P_{ij}}$ representa o primeiro nó intermédio entre os nós **i** e **j**
- Resultado da execução do algoritmo:
  - os elementos da matriz **D** são as distâncias mais curtas entre qualquer par de nós

Algoritmo de Floyd

Passo 1.

D = matriz das distâncias diretas

$P_{ij} = j, \quad \forall\ i, j \in N$

$k \leftarrow 0$

Passo 2.

se $(k \geq n)$ **então**

PARAR

fim_se

$k \leftarrow k + 1$

Passo 3.

se $(D_{ij} > D_{ik} + D_{kj})$ **então** $(i, j \neq k$ – não se considera a linha k e a coluna k)

$D_{ij} \leftarrow D_{ik} + D_{kj}$

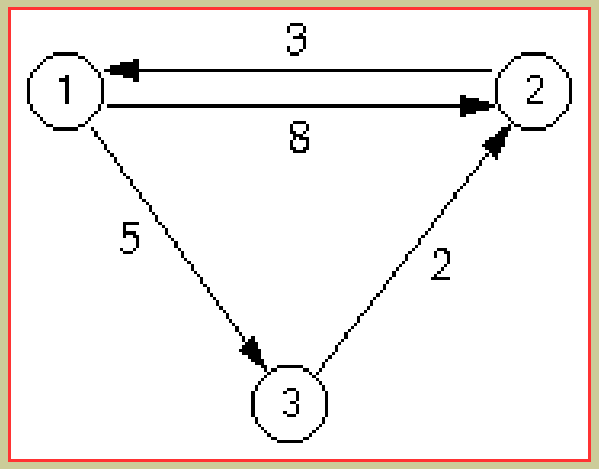
$P_{ij} \leftarrow P_{ik}$

fim_se

regressar ao **Passo 2**

Algoritmo de Floyd - exemplo

- Determinar o caminho mais curto entre todos os pares de nós da seguinte rede:



Passo1.

$$D = \begin{bmatrix} 0 & 8 & 5 \\ 3 & 0 & \infty \\ \infty & 2 & 0 \end{bmatrix}$$

$$P = \begin{bmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \\ 1 & 2 & 3 \end{bmatrix}$$

$k \leftarrow 0$

Passo 1.

D = matriz das distâncias diretas

$P_{ij} = j, \ \forall\ i, j \in N$

$k \leftarrow 0$

Algoritmo de Floyd - exemplo

Passo 2.

$0 \geq 3$ (n = 3) Falso

$k \leftarrow k + 1 = 0 + 1 = 1$

Passo 3.

i = 2; k = 1; j = 2: $D_{22} < D_{21} + D_{12}$ $(0 < 3 + 8)$

i = 2; k = 1; j = 3: $\mathbf{D_{23} > D_{21} + D_{13}}$ $(\infty \mathbf{> 3 + 5})$

$\Rightarrow$ $D_{23} = D_{21} + D_{13} = 8$; $P_{23} = P_{21} = 1$

i = 3; k = 1; j = 2: $D_{32} < D_{31} + D_{12}$ $(2 < \infty + 8)$

i = 3; k = 1; j = 3: $D_{33} < D_{31} + D_{13}$ $(0 < \infty + 5)$

$$D = \begin{bmatrix} 0 & 8 & 5 \\ 3 & 0 & \infty \\ \infty & 2 & 0 \end{bmatrix} \Rightarrow D = \begin{bmatrix} 0 & 8 & 5 \\ 3 & 0 & 8 \\ \infty & 2 & 0 \end{bmatrix}$$

$$P = \begin{bmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \\ 1 & 2 & 3 \end{bmatrix} \Rightarrow P = \begin{bmatrix} 1 & 2 & 3 \\ 1 & 2 & 1 \\ 1 & 2 & 3 \end{bmatrix}$$

Passo 2.
  se $(k \geq n)$ **então** PARAR
  $k \leftarrow k + 1$
Passo 3.
  se $(D_{ij} > D_{ik} + D_{kj}$ $(i, j \neq k))$ **então**
    $D_{ij} \leftarrow D_{ik} + D_{kj}$
    $P_{ij} \leftarrow P_{ik}$
  fim_se

Algoritmo de Floyd - exemplo

Passo 2.

$1 \geq 3$ (n = 3) Falso

$k \leftarrow k + 1 = 1 + 1 = 2$

Passo 3.

i = 1; k = 2; j = 1: $D_{11} < D_{12} + D_{21}$ $(0 < 8 + 3)$

i = 1; k = 2; j = 3: $D_{13} < D_{12} + D_{23}$ $(5 < 8 + 8)$

i = 3; k = 2; j = 1: $\mathbf{D_{31} > D_{32} + D_{21}}$ $(\infty \mathbf{> 2 + 3})$

$\Rightarrow$ $D_{31} = D_{32} + D_{21} = 5$; $P_{31} = P_{32} = 2$

i = 3; k = 2; j = 3: $D_{33} < D_{32} + D_{23}$ $(0 < 2 + 8)$

$$D = \begin{bmatrix} 0 & 8 & 5 \\ 3 & 0 & 8 \\ \infty & 2 & 0 \end{bmatrix} \Rightarrow D = \begin{bmatrix} 0 & 8 & 5 \\ 3 & 0 & 8 \\ 5 & 2 & 0 \end{bmatrix}$$

$$P = \begin{bmatrix} 1 & 2 & 3 \\ 1 & 2 & 1 \\ 1 & 2 & 3 \end{bmatrix} \Rightarrow P = \begin{bmatrix} 1 & 2 & 3 \\ 1 & 2 & 1 \\ 2 & 2 & 3 \end{bmatrix}$$

Passo 2.
  se $(k \geq n)$ **então** PARAR
  $k \leftarrow k + 1$
Passo 3.
  se $(D_{ij} > D_{ik} + D_{kj}$ $(i, j \neq k))$ **então**
    $D_{ij} \leftarrow D_{ik} + D_{kj}$
    $P_{ij} \leftarrow P_{ik}$
  fim_se

Algoritmo de Floyd - exemplo

Passo 2.

$2 \geq 3$ (n = 3) Falso

$k \leftarrow k + 1 = 2 + 1 = 3$

Passo 3.

i = 1; k = 3; j = 1: $D_{11} < D_{13} + D_{31}$ (0 < 5 + 5)

i = 1; k = 3; j = 2: **$D_{12} > D_{13} + D_{32}$** (**8 > 5 + 2**)

$\Rightarrow$ $D_{12} = D_{13} + D_{32} = 7$; $P_{12} = P_{13} = 3$

i = 2; k = 3; j = 1: $D_{21} < D_{23} + D_{31}$ (3 < 8 + 5)

i = 2; k = 3; j = 2: $D_{22} < D_{23} + D_{32}$ (0 < 8 + 2)

$$D = \begin{bmatrix} 0 & 8 & 5 \\ 3 & 0 & 8 \\ 5 & 2 & 0 \end{bmatrix} \Rightarrow D = \begin{bmatrix} 0 & 7 & 5 \\ 3 & 0 & 8 \\ 5 & 2 & 0 \end{bmatrix}$$

$$P = \begin{bmatrix} 1 & 2 & 3 \\ 1 & 2 & 1 \\ 2 & 2 & 3 \end{bmatrix} \Rightarrow P = \begin{bmatrix} 1 & 3 & 3 \\ 1 & 2 & 1 \\ 2 & 2 & 3 \end{bmatrix}$$

Passo 2.
  se ($k \geq n$) **então** PARAR
  $k \leftarrow k + 1$
Passo 3.
  se ($D_{ij} > D_{ik} + D_{kj}$ (i, j $\neq$ k)) **então**
    $D_{ij} \leftarrow D_{ik} + D_{kj}$
    $P_{ij} \leftarrow P_{ik}$
  fim_se

Algoritmo de Floyd - exemplo

Passo 2.

$3 \geq 3$ (n = 3) Verdadeiro

PARAR (termina algoritmo)

Passo 2.

se ($k \geq n$) **então** PARAR

$k \leftarrow k + 1$

- Resultados após o final do algoritmo:

$$D = \begin{bmatrix} 0 & 7 & 5 \\ 3 & 0 & 8 \\ 5 & 2 & 0 \end{bmatrix} \qquad P = \begin{bmatrix} 1 & 3 & 3 \\ 1 & 2 & 1 \\ 2 & 2 & 3 \end{bmatrix}$$

  - o comprimento do caminho mais curto entre os nós 1 e 2 é:

    7 ($D_{12} = 7$)

  - o caminho mais curto entre os nós 1 e 2 é:

    [1, 3 ($P_{12} = 3$), 2 ($P_{32} = 2$)]

O problema do Fluxo Máximo

Conceitos gerais

- Considere uma rede G, em que os valores associados aos seus arcos, $\mathbf{b_{ij}}$, representam as respetivas capacidades
  - correspondem à quantidade máxima de fluxo que pode ser enviada pelos arcos
  - os valores associados terão que ser positivos ($b_{ij} \geq 0$)
- Pode-se definir a rede da seguinte forma: G = (N, A, B), em que B = $[b_{ij}]$
- Em problemas de fluxo máximo existem 2 nós especiais:
  - nó origem e
  - nó terminal
- Com a resolução do problema de fluxo máximo, pretende-se determinar a quantidade máxima de unidades de fluxo que pode ser enviada de um nó origem (S) para um nó terminal (T)
- O fluxo no arco (i, j) é designado por $\mathbf{x_{ij}}$

Conceitos gerais

- Devido às restrições de capacidade nos arcos, tem-se o conjunto de restrições

  $0 \leq x_{ij} \leq b_{ij}$, para todo $(i, j) \in A$ **[restrição 1]**

- Além disso, em cada nó (exceto S e T) deve haver conservação de fluxo
  - a quantidade de fluxo que chega a um nó é igual à quantidade de fluxo que sai desse nó:

  $$\sum_{i \in N} x_{ij} = \sum_{k \in N} x_{jk} \ , \quad \forall j \in N - \{S, T\}$$ **[restrição 2]**

- Como existe conservação de fluxo em todos os nós, o fluxo que sai do nó S é igual ao fluxo que chega ao nó T

  $$\sum_{i \in N} x_{Si} = f = \sum_{j \in N} x_{jT}$$ (onde f é o valor de fluxo) **[restrição 3]**

- Portanto, com a resolução do problema de fluxo máximo, pretende-se determinar o valor do fluxo nos arcos $\mathbf{x_{ij}}$ **[restrição 1]**, que maximize o valor do fluxo **f**, sujeito às restrições de capacidade **[restrição 2]** e de conservação de fluxo **[restrição 3]**

Conceitos gerais

- Matematicamente, este problema pode ser formulado da seguinte forma:

maximizar $f = \sum_{j \in N} x_{Sj}$

sujeito a

$\sum_{i \in N} x_{ij} = \sum_{k \in N} x_{jk}$, $\forall j \in N - \{S, T\}$ **[restrição 2]**

$\sum_{i \in N} x_{Si} = \sum_{j \in N} x_{jT}$ **[restrição 3]**

$0 \leq x_{ij} \leq b_{ij}$, para todo $(i, j) \in A$ **[restrição 1]**

- Dado um caminho qualquer de S para T numa rede,

$p = [\ S=n_1, n_2, n_3, \ldots, n_{k-1}, n_k=T\ ]$,

a quantidade máxima de fluxo que pode ser enviada de S para T, por aquele caminho, satisfazendo as restrições [1], [2] e [3] é a seguinte:

$\min \{\ b_{ij} : (i, j) \in p\ \}$

- O arco com a menor capacidade fica "saturado", não passando mais fluxo nele

Algoritmo de Ford-Fulkerson

- É um modo sistemático de pesquisar todos os possíveis caminhos de aumento de fluxo (c.a.f.) de S para T, atribuindo rótulos aos nós para indicar a direção em que o fluxo pode ser aumentado
- Cada nó pode estar num dos 3 estados seguintes:
  - rotulado e varrido $\Rightarrow$ tem um rótulo e todos os seus vizinhos estão rotulados
  - rotulado e não varrido $\Rightarrow$ tem um rótulo e nem todos os seus vizinhos estão rotulados
  - não rotulado $\Rightarrow$ não tem rótulo
- O rótulo do nó **j**, R(j), é composto por duas partes: $[i^+, Q(j)]$ (ou $[i^-, Q(j)]$)
  - $\mathbf{i^+}$: índice de um nó **i**, indicando que pode-se enviar fluxo de **i** para **j**
  - $\mathbf{i^-}$: índice de um nó **i**, indicando que pode-se enviar fluxo de **j** para **i**
  - **Q(j)**: fluxo máximo adicional que se pode enviar de S para **j**

Algoritmo de Ford-Fulkerson

Passo 1. (iniciação)

R(S) $\leftarrow [S^+, \infty]$ (S é rotulado com S^+ e ∞)

S fica rotulado e não varrido

Algoritmo de Ford-Fulkerson

Passo 2. (processo de rotulação)

j (rotulado e não varrido) $\leftarrow [i^+, Q(j)]$ **ou** $[i^-, Q(j)]$

para (todo o **k** $\in$ N não rotulado, tal que $(j, k) \in A$ **e** $x_{jk} < b_{jk}$) **fazer**

R(k) $\leftarrow [j^+, Q(k)]$ com $Q(k) = \min \{ Q(j), b_{jk} - x_{jk} \}$

fim_para

para (todo o **k** $\in$ N não rotulado, tal que $(k, j) \in A$ **e** $x_{kj} > 0$) **fazer**

R(k) $\leftarrow [j^-, Q(k)]$ com $Q(k) = \min \{ Q(j), x_{kj} \}$

fim_para

j fica **rotulado** e **varrido**

todos os nós **k** ficam **rotulados** e **não varridos**

se (T está rotulado) **ou** (não é possível rotular T) **então**

(foi determinado um c.a.f. $\Rightarrow$ **passar** ao **Passo 3**) **ou**

(não existe c.a.f. — o fluxo atual é máximo $\Rightarrow$ **PARAR**)

senão

regressar ao **Passo 2** (início)

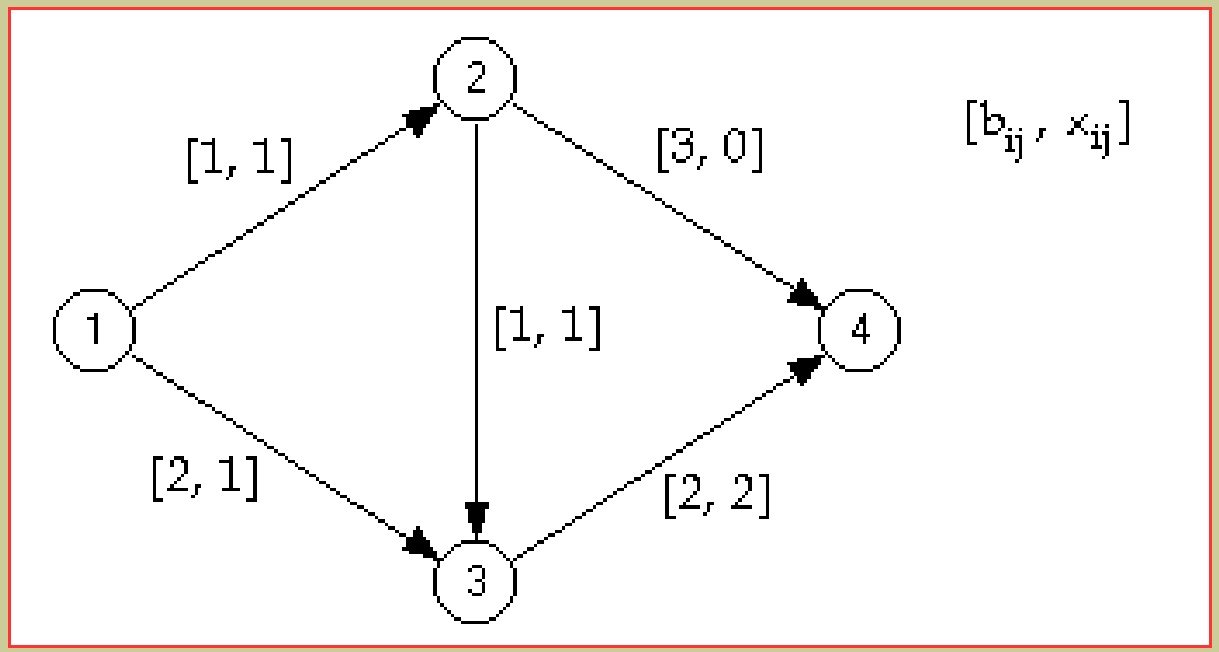
fim_se

Algoritmo de Ford-Fulkerson

Passo 3. (mudança de fluxo)
  como ($R(T) = [k^+, Q(T)]$) **então**
    $x_{kT} \leftarrow x_{kT} + Q(T)$
  enquanto ($k \neq S$) **fazer**
    se ($R(k) = [j^+, Q(k)]$) **então**
      $x_{jk} \leftarrow x_{jk} + Q(T)$
    fim_se
    se ($R(k) = [j^-, Q(k)]$) **então**
      $x_{kj} \leftarrow x_{kj} - Q(T)$
    fim_se
    $k \leftarrow j$
  fim_enquanto
  apagar os rótulos
  regressar ao **Passo 1**

Algoritmo de Ford-Fulkerson - exemplo

- Considere a seguinte rede onde os valores correspondem à capacidade e ao fluxo atual nos arcos:



- Pretende-se determinar o fluxo máximo a enviar do nó **1** para o nó **4**, atendendo a que o fluxo atual enviado entre aqueles dois nós é de 2 unidades

Algoritmo de Ford-Fulkerson - exemplo

Fluxo atual = 2

Passo 1.

R(1) $\leftarrow [1^+, \infty]$

RotuladosVarridos = $\varnothing$

RotuladosNãoVarridos = { 1 }

NãoRotulados = { 2, 3, 4 }

Passo 1.

R(S) $\leftarrow [S^+, \infty]$ (S é rotulado com S^+ e ∞)

S fica rotulado e não varrido

Algoritmo de Ford-Fulkerson - exemplo

Passo 2.

j ← **1** (rotulado e não varrido)

Vizinhos (não rotulados) do nó 1: 2 e 3

k ← **2**

não pode ser rotulado, pois $b_{12} = x_{12}$

k ← **3**

R(3) ← $[1^+, \min \{ Q(1), b_{13} - x_{13} \}] =$

$[1^+, \min \{ \infty, 1 \}] =$ **$[1^+, 1]$**

RotuladosVarridos = { 1 }

RotuladosNãoVarridos = { 3 }

NãoRotulados = { 2, 4 }

Passo 2. (processo de rotulação)
j (rotulado e não varrido) ← $[i^+, Q(j)]$ ou $[i^-, Q(j)]$
para todo o **k** ∈ N : (j, k) ∈ A **e** $x_{jk} < b_{jk}$ **fazer**
R(k) ← $[j^+, Q(k)]$, $Q(k) = \min\{ Q(j), b_{jk} - x_{jk} \}$
fim_para
para todo o **k** ∈ N : (k, j) ∈ A **e** $x_{kj} > 0$ **fazer**
R(k) ← $[j^-, Q(k)]$, $Q(k) = \min\{ Q(j), x_{kj} \}$
fim_para
j fica rotulado e varrido
todos os nós **k** ficam rotulados e não varridos
se (T está rotulado **ou** não é possível rotular) **então**
foi determinado um c.a.f. **ou**
não existe c.a.f. ⇒ o fluxo atual é máximo
senão
regressar ao **Passo 2** (início)
fim_se

Algoritmo de Ford-Fulkerson - exemplo

Passo 2.

j ← **3** (rotulado e não varrido)

Vizinhos (não rotulados) do nó 3: 2 e 4

k ← **2**

R(2) ← $[3^-, \min \{ Q(3), x_{23} \}]$ =

$[3^-, \min \{ 1, 1 \}]$ = **$[3^-, 1]$**

k ← **4**

não pode ser rotulado, pois $b_{34} = x_{34}$

RotuladosVarridos = { 1, 3 }

RotuladosNãoVarridos = { 2 }

NãoRotulados = { 4 }

Passo 2. (processo de rotulação)
j (rotulado e não varrido) ← $[i^+, Q(j)]$ ou $[i^-, Q(j)]$
para todo o **k** ∈ N : (j, k) ∈ A **e** $x_{jk} < b_{jk}$ **fazer**
R(k) ← $[j^+, Q(k)]$, $Q(k) = \min\{ Q(j), b_{jk} - x_{jk} \}$
fim_para
para todo o **k** ∈ N : (k, j) ∈ A **e** $x_{kj} > 0$ **fazer**
R(k) ← $[j^-, Q(k)]$, $Q(k) = \min\{ Q(j), x_{kj} \}$
fim_para
j fica rotulado e varrido
todos os nós **k** ficam rotulados e não varridos
se (T está rotulado **ou** não é possível rotular) **então**
foi determinado um c.a.f. **ou**
não existe c.a.f. ⇒ o fluxo atual é máximo
senão
regressar ao **Passo 2** (início)
fim_se

Algoritmo de Ford-Fulkerson - exemplo

Passo 2.

j ← **2** (rotulado e não varrido)

Vizinhos (não rotulados) do nó 2: 4

k ← **4**

R(4) ← $[2^+, \min \{ Q(2), b_{24} - x_{24} \}] =$

$[2^+, \min \{ 1, 3 \}] =$ **$[2^+, 1]$**

RotuladosVarridos = { 1, 3, 2 }

RotuladosNãoVarridos = { 4 }

NãoRotulados = ∅

como (o nó 4=T foi rotulado) **então**

foi determinado um c.a.f.

Passo 2. (processo de rotulação)

j (rotulado e não varrido) ← $[i^+, Q(j)]$ ou $[i^-, Q(j)]$

para todo o **k** ∈ N : (j, k) ∈ A **e** $x_{jk} < b_{jk}$ **fazer**

R(k) ← $[j^+, Q(k)]$, $Q(k) = \min\{ Q(j), b_{jk} - x_{jk} \}$

fim_para

para todo o **k** ∈ N : (k, j) ∈ A **e** $x_{kj} > 0$ **fazer**

R(k) ← $[j^-, Q(k)]$, $Q(k) = \min\{ Q(j), x_{kj} \}$

fim_para

j fica rotulado e varrido

todos os nós **k** ficam rotulados e não varridos

se (T está rotulado **ou** não é possível rotular) **então**

foi determinado um c.a.f. **ou**

não existe c.a.f. ⇒ o fluxo atual é máximo

senão

regressar ao **Passo 2** (início)

fim_se

Algoritmo de Ford-Fulkerson - exemplo

Passo 3.

O aumento de fluxo é de Q(4) = 1 unidade

como (R(4) = [2^+, 1]) **então**

$x_{24} = x_{24} + Q(4) = 0 + 1 = 1$

k ← **2**

como (2 ≠ 1 **e** R(2) = [3^-, 1]) **então**

$x_{23} = x_{23} - Q(4) = 1 - 1 = 0$

k ← **3**

como (3 ≠ 1 **e** R(3) = [1^+, 1]) **então**

$x_{13} = x_{13} + Q(4) = 1 + 1 = 2$

k ← **1**

como 1 = 1 **então** terminar

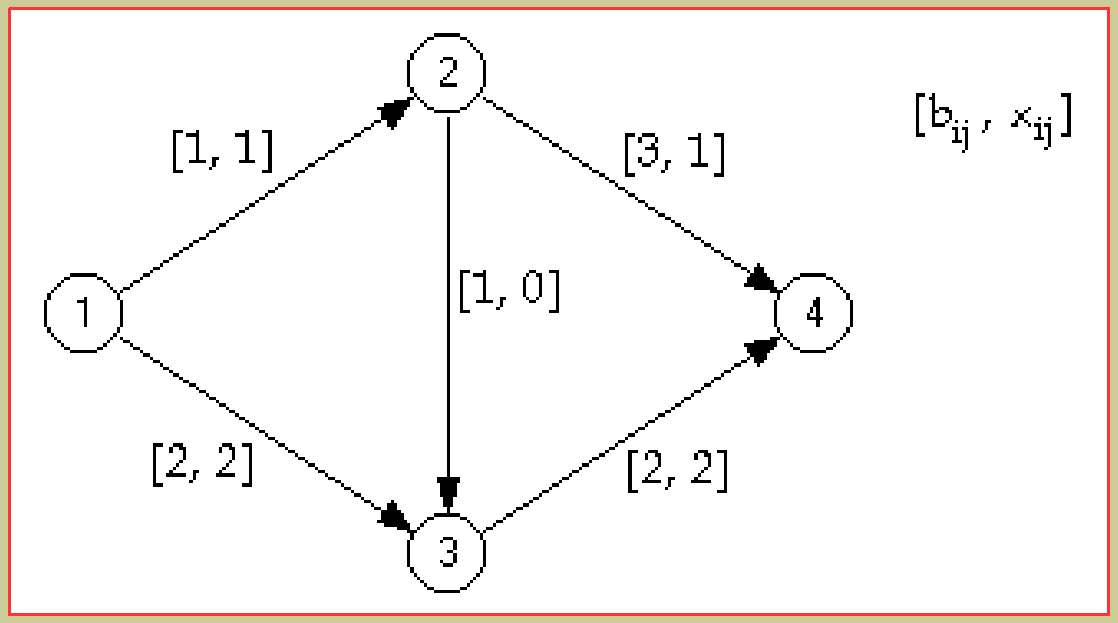
apagar os rótulos dos nós da rede

regressar ao **Passo 1**

Passo 2. (processo de rotulação)
j (rotulado e não varrido) ← [i^+, Q(j)] ou [i^-, Q(j)]
para todo o **k** ∈ N : (j, k) ∈ A **e** $x_{jk} < b_{jk}$ **fazer**
R(k) ← [j^+, Q(k)], Q(k) = min{ Q(j), $b_{jk} - x_{jk}$ }
fim_para
para todo o **k** ∈ N : (k, j) ∈ A **e** $x_{kj} > 0$ **fazer**
R(k) ← [j^-, Q(k)], Q(k) = min{ Q(j), x_{kj} }
fim_para
j fica rotulado e varrido
todos os nós **k** ficam rotulados e não varridos
se (T está rotulado **ou** não é possível rotular) **então**
foi determinado um c.a.f. **ou**
não existe c.a.f. ⇒ o fluxo atual é máximo
senão
regressar ao **Passo 2** (início)
fim_se

Algoritmo de Ford-Fulkerson - exemplo

- Fluxo atual = Fluxo anterior + Q(4) = 2 + 1 = 3



- Aplicar novamente o algoritmo à nova configuração de rede

Algoritmo de Ford-Fulkerson - exemplo

Fluxo atual = 3

Passo 1.

R(1) $\leftarrow [1^+, \infty]$

RotuladosVarridos = $\varnothing$

RotuladosNãoVarridos = { 1 }

NãoRotulados = { 2, 3, 4 }

Passo 1.

R(S) $\leftarrow [S^+, \infty]$ (S é rotulado com S^+ e ∞)

S fica rotulado e não varrido

Algoritmo de Ford-Fulkerson - exemplo

Passo 2.

j ← **1** (rotulado e não varrido)

Vizinhos (não rotulados) do nó 1: 2 e 3

k ← **2**

não pode ser rotulado: $b_{12} = x_{12}$

k ← **3**

não pode ser rotulado: $b_{13} = x_{13}$

como (não é possível rotular mais nós **e**

o nó 4=T não foi rotulado) **então**

não existe c.a.f.

(fluxo atual é máximo)

Passo 2. (processo de rotulação)

j (rotulado e não varrido) ← $[i^+, Q(j)]$ ou $[i^-, Q(j)]$

para todo o **k** ∈ N : (j, k) ∈ A **e** $x_{jk} < b_{jk}$ **fazer**

R(k) ← $[j^+, Q(k)]$, $Q(k) = \min\{ Q(j), b_{jk} - x_{jk} \}$

fim_para

para todo o **k** ∈ N : (k, j) ∈ A **e** $x_{kj} > 0$ **fazer**

R(k) ← $[j^-, Q(k)]$, $Q(k) = \min\{ Q(j), x_{kj} \}$

fim_para

j fica rotulado e varrido

todos os nós **k** ficam rotulados e não varridos

se (T está rotulado **ou** não é possível rotular) **então**

foi determinado um c.a.f. **ou**

não existe c.a.f. ⇒ o fluxo atual é máximo

senão

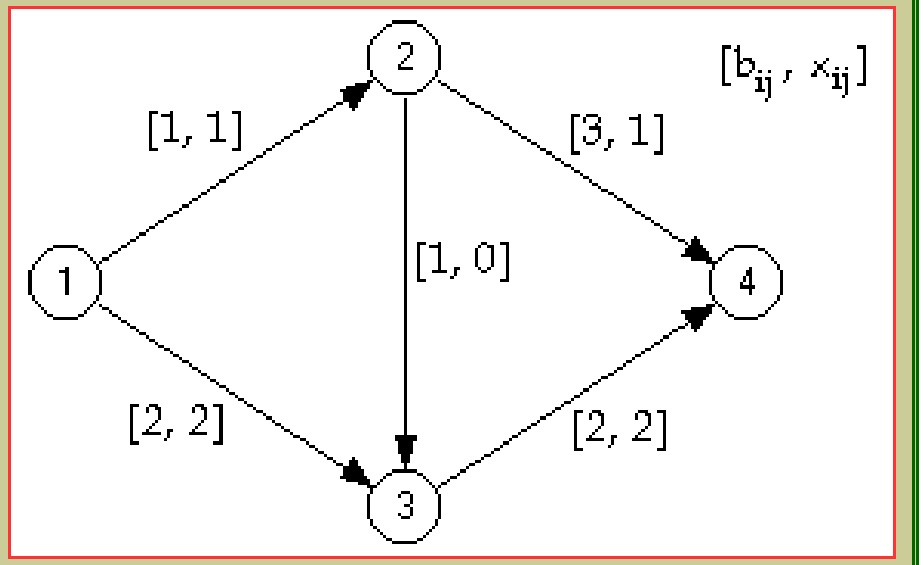
regressar ao **Passo 2** (início)

fim_se

Algoritmo de Ford-Fulkerson - exemplo

Resultados a retirar após o final do algoritmo (S = 1 e T = 4):

- Fluxo máximo = 3
- Fluxo que sai do nó 1 = Fluxo que chega ao nó 4
  - fluxo que sai de 1: $x_{12} + x_{13} = 1 + 2 = 3$
  - fluxo que chega a 4: $x_{24} + x_{34} = 1 + 2 = 3$
- Fluxo que sai do nó 2 = Fluxo que chega ao nó 2
  - fluxo que sai de 2: $x_{23} + x_{24} = 0 + 1 = 1$
  - fluxo que chega a 2: $x_{12} = 1$
- Fluxo que sai do nó 4 = Fluxo que chega ao nó 4
  - fluxo que sai de 3: $x_{34} = 2$
  - fluxo que chega a 3: $x_{13} + x_{23} = 2 + 0 = 2$



O problema dos Casamentos Estáveis

Definição do problema

- Do ponto de vista de teoria dos Grafos, podem ser vistos como problemas para se obter emparelhamentos com preferência em grafos bipartidos, o qual é denominada de "emparelhamento estável"
- Os problemas de emparelhamento estável ("Stable Marriage Problem")
  - consistem em dividir um ou mais grupos de agentes em pares, onde cada agente possui uma lista ordenada de preferências,
  - pretendem encontrar um emparelhamento entre eles que respeite um critério de estabilidade baseado nas suas preferências

Definição do problema

- A versão clássica do problema traduz-se da forma que se segue

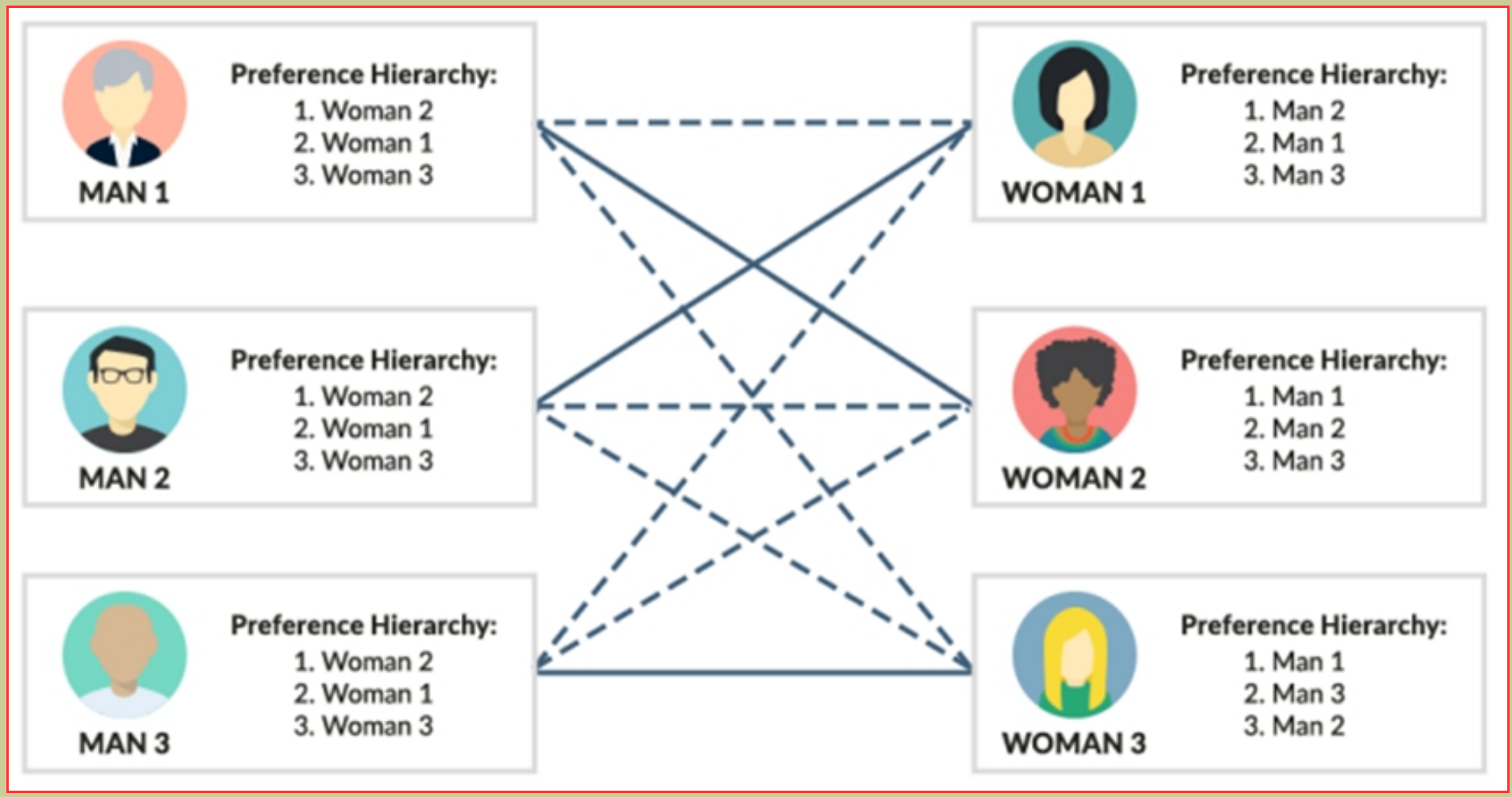
  StableMarriage:

  Supondo que cada elemento de um grupo de N homens e N mulheres ordenou todos os de sexo oposto por ordem de preferência estrita, pretende-se determinar um emparelhamento estável

- Sendo H = { h_1, ..., h_N } e M = { m_1, ..., m_N } os conjuntos de homens e mulheres, um emparelhamento E é uma qualquer função injectiva de H em M
  - informalmente, neste caso, um emparelhamento é um conjunto de N casais (monogâmicos e heterosexuais)
- Um emparelhamento **E** diz-se **instável** se e só se existir um par (**h**, **m**) $\notin$ **E** tal que **h** prefere **m** à sua parceira em **E** e **m** também prefere **h** ao seu parceiro em **E**; caso contrário, diz-se **estável**

Definição do problema

- Este problema pode ser modelado através do grafo que se segue



Exemplo

- Considere-se a seguinte instância (com n = 4), assim como as listas de preferências ordenadas por ordem (estritamente) decrescente da esquerda para a direita,

  h_1: m_4, m_2, m_3, m_1 $\quad$ m_1: h_4, h_2, h_1, h_3

  h_2: m_2, m_3, m_4, m_1 $\quad$ m_2: h_3, h_1, h_4, h_2

  h_3: m_2, m_3, m_1, m_4 $\quad$ m_3: h_2, h_3, h_1, h_4

  h_4: m_1, m_3, m_2, m_4 $\quad$ m_4: h_3, h_4, h_2, h_1

  - pode-se verificar, através duma simples análise de casos, que

    $E = \{ (h_1, m_4), (h_2, m_3), (h_3, m_2), (h_4, m_1) \}$

    é um **emparelhamento estável**

  - pode-se também verificar, através duma simples análise de casos, que

    $E' = \{ (h_1, m_4), (h_2, m_2), (h_3, m_1), (h_4, m_3) \}$

    não é um **emparelhamento estável**

    - $(h_4, m_1) \notin E'$, h_4 prefere m_1 a m_3, m_1 prefere h_4 a h_3

Algoritmo de Gale-Shapley

- Gale e Shapley (1962) mostraram que qualquer instância de *StableMarriage* admite pelo menos uma solução (ou seja, um emparelhamento estável) e que um tal emparelhamento poderia ser obtido por aplicação do algoritmo Gale-Shapley

Algoritmo de Gale-Shapley

```
Algoritmo Gale-Shapley
  considerar inicialmente que todas as pessoas estão livres
  enquanto (houver algum homem h livre) fazer
     seja m a primeira mulher na lista de h a quem este ainda não se propôs
     se (m está livre) então
        emparelhar h e m (ficam noivos)
     senão
        se (m preferir h ao seu atual noivo h') então
           emparelhar h e m (ficam noivos), voltando h' a estar livre
        senão
           m rejeita h e h continua livre
        fim_se
     fim_se
  fim_enquanto
fim_algoritmo
```

Algoritmo de Gale-Shapley - exemplo

- Sejam H e M os conjuntos de homens e mulheres seguinte:

  H = { Vítor (V), Wilson (W), Xavier (X), Yuri (Y), Zeca (Z) }

  M = { Ana (A), Beatriz (B), Carolina (C), Débora (D), Erica (E) }

- Sejam as listas de preferências dos elementos de H e M seguintes:

V - Vitor:	A > C > D > E > B	A - Ana:	Z > W > V > Y > X
W - Wilson:	A > D > E > B > C	B - Beatriz:	V > X > Y > Z > W
X - Xavier:	D > C > B > A > E	C - Carolina:	Z > W > V > X > Y
Y - Yuri:	A > D > B > E > C	D - Débora:	V > W > Y > X > Z
Z - Zeca:	C > D > A > B > E	E - Erica:	W > Z > X > Y > V

- Pretende-se determinar um emparelhamento estável, em que:
  - os homens são os agentes proponentes (os homens propõem às mulheres de que mais gostam, de forma a escolher a primeira em cada lista de preferência),
  - as mulheres os agentes seletores (somente poderão escolher dentre as propostas recebidas)

Algoritmo de Gale-Shapley - exemplo (iteração 1)

	preferências de H					emparelhamentos				
	V	**W**	**X**	**Y**	**Z**	**A**	**B**	**C**	**D**	**E**
1ª	A	A	D	A	C					
2ª	C	D	C	D	D					
3ª	D	E	B	B	A					
4ª	E	B	A	E	B					

h = V (V está livre)

m = A; A está livre, então emparelhar V a A

Algoritmo de Gale-Shapley - exemplo (iteração 1)

preferências de H						emparelhamentos				
	V	**W**	**X**	**Y**	**Z**	**A**	**B**	**C**	**D**	**E**
1ª	A	A	D	A	C	V				
2ª	C	D	C	D	D					
3ª	D	E	B	B	A					
4ª	E	B	A	E	B					

h = V (V está livre)

m = A; A está livre, então emparelhar V a A

h = W (W está livre)

m = A; A não está livre, então A prefere W a V? **Sim**: emparelhar W a A e V fica livre

Algoritmo de Gale-Shapley - exemplo (iteração 1)

preferências de H						emparelhamentos				
	V	**W**	**X**	**Y**	**Z**	**A**	**B**	**C**	**D**	**E**
1ª	A	A	D	A	C	V				
2ª	C	D	C	D	D	W				
3ª	D	E	B	B	A					
4ª	E	B	A	E	B					

h = V (V está livre)

m = A; A está livre, então emparelhar V a A

h = W (W está livre)

m = A; A não está livre, então A prefere W a V? **Sim**: emparelhar W a A e V fica livre

h = X (X está livre)

m = D; D está livre, então emparelhar X a D

Algoritmo de Gale-Shapley - exemplo (iteração 1)

preferências de H						emparelhamentos				
	V	**W**	**X**	**Y**	**Z**	**A**	**B**	**C**	**D**	**E**
1ª	A	A	D	A	C	V			X	
2ª	C	D	C	D	D	W				
3ª	D	E	B	B	A					
4ª	E	B	A	E	B					

h = V (V está livre)

m = A; A está livre, então emparelhar V a A

h = W (W está livre)

m = A; A não está livre, então A prefere W a V? **Sim**: emparelhar W a A e V fica livre

h = X (X está livre)

m = D; D está livre, então emparelhar X a D

h = Y (Y está livre)

m = A; A não está livre, então A prefere Y a W? **Não**: Y continua livre

Algoritmo de Gale-Shapley - exemplo (iteração 1)

preferências de H						emparelhamentos				
	V	**W**	**X**	**Y**	**Z**	**A**	**B**	**C**	**D**	**E**
1ª	A	A	D	A	C	V			X	
2ª	C	D	C	D	D	W				
3ª	D	E	B	B	A					
4ª	E	B	A	E	B					

h = V (V está livre)

m = A; A está livre, então emparelhar V a A

h = W (W está livre)

m = A; A não está livre, então A prefere W a V? **Sim**: emparelhar W a A e V fica livre

h = X (X está livre)

m = D; D está livre, então emparelhar X a D

h = Y (Y está livre)

m = A; A não está livre, então A prefere Y a W? **Não**: Y continua livre

h = Z (Z está livre)

m = C; C está livre, então emparelhar Z a C

Algoritmo de Gale-Shapley - exemplo (iteração 1)

preferências de H						emparelhamentos				
	V	**W**	**X**	**Y**	**Z**	**A**	**B**	**C**	**D**	**E**
1ª	A	A	D	A	C	V		Z	X	
2ª	C	D	C	D	D	W				
3ª	D	E	B	B	A					
4ª	E	B	A	E	B					

h = V (V está livre)

m = A; A está livre, então emparelhar V a A

h = W (W está livre)

m = A; A não está livre, então A prefere W a V? **Sim**: emparelhar W a A e V fica livre

h = X (X está livre)

m = D; D está livre, então emparelhar X a D

h = Y (Y está livre)

m = A; A não está livre, então A prefere Y a W? **Não**: Y continua livre

h = Z (Z está livre)

m = C; C está livre, então emparelhar Z a C

Algoritmo de Gale-Shapley - exemplo (iteração 2)

preferências de H						emparelhamentos				
	V	**W**	**X**	**Y**	**Z**	**A**	**B**	**C**	**D**	**E**
1ª	A	A	D	A	C	V		Z	X	
2ª	C	D	C	D	D	W				
3ª	D	E	B	B	A					
4ª	E	B	A	E	B					

h = V (V está livre)

m = C; C não está livre, então C prefere V a Z? **Não**: V continua livre

Algoritmo de Gale-Shapley - exemplo (iteração 2)

preferências de H						emparelhamentos				
	V	**W**	**X**	**Y**	**Z**	**A**	**B**	**C**	**D**	**E**
1ª	A	A	D	A	C	V		Z	X	
2ª	C	D	C	D	D	W				
3ª	D	E	B	B	A					
4ª	E	B	A	E	B					

h = V (V está livre)

m = C; C não está livre, então C prefere V a Z? **Não**: V continua livre

h = W (W não está livre)

h = X (X não está livre)

h = Y (Y está livre)

m = D; D não está livre, então D prefere Y a X? **Sim**: emparelhar Y a D e X fica livre

Algoritmo de Gale-Shapley - exemplo (iteração 2)

preferências de H						emparelhamentos				
	V	**W**	**X**	**Y**	**Z**	**A**	**B**	**C**	**D**	**E**
1ª	A	A	D	A	C	V		Z	X	
2ª	C	D	C	D	D	W			Y	
3ª	D	E	B	B	A					
4ª	E	B	A	E	B					

h = V (V está livre)

m = C; C não está livre, então C prefere V a Z? **Não**: V continua livre

h = W (W não está livre)

h = X (X não está livre)

h = Y (Y está livre)

m = D; D não está livre, então D prefere Y a X? **Sim**: emparelhar Y a D e X fica livre

h = Z (Z não está livre)

Algoritmo de Gale-Shapley - exemplo (iteração 2)

preferências de H						emparelhamentos				
	V	**W**	**X**	**Y**	**Z**	**A**	**B**	**C**	**D**	**E**
1ª	A	A	D	A	C	V		Z	X	
2ª	C	D	C	D	D	W			Y	
3ª	D	E	B	B	A					
4ª	E	B	A	E	B					

h = V (V está livre)

m = C; C não está livre, então C prefere V a Z? **Não**: V continua livre

h = W (W não está livre)

h = X (X não está livre)

h = Y (Y está livre)

m = D; D não está livre, então D prefere Y a X? **Sim**: emparelhar Y a D e X fica livre

h = Z (Z não está livre)

Algoritmo de Gale-Shapley - exemplo (iteração 3)

preferências de H						emparelhamentos				
	V	**W**	**X**	**Y**	**Z**	**A**	**B**	**C**	**D**	**E**
1ª	A	A	D	A	C	V		Z	X	
2ª	C	D	C	D	D	W			Y	
3ª	D	E	B	B	A					
4ª	E	B	A	E	B					

h = V (V está livre)

m = D; D não está livre, então D prefere V a Y? **Sim**: emparelhar V a D e Y fica livre

Algoritmo de Gale-Shapley - exemplo (iteração 3)

preferências de H						emparelhamentos				
	V	**W**	**X**	**Y**	**Z**	**A**	**B**	**C**	**D**	**E**
1ª	A	A	D	A	C	V		Z	X	
2ª	C	D	C	D	D	W			Y	
3ª	D	E	B	B	A				V	
4ª	E	B	A	E	B					

h = V (V está livre)

m = D; D não está livre, então D prefere V a Y? **Sim**: emparelhar V a D e Y fica livre

h = W (W não está livre)

h = X (X está livre)

m = C; C não está livre, então C prefere X a Z? **Não**, então X continua livre

Algoritmo de Gale-Shapley - exemplo (iteração 3)

preferências de H						emparelhamentos				
	V	**W**	**X**	**Y**	**Z**	**A**	**B**	**C**	**D**	**E**
1ª	A	A	D	A	C	V		Z	X	
2ª	C	D	C	D	D	W			Y	
3ª	D	E	B	B	A				V	
4ª	E	B	A	E	B					

h = V (V está livre)

m = D; D não está livre, então D prefere V a Y? **Sim**: emparelhar V a D e Y fica livre

h = W (W não está livre)

h = X (X está livre)

m = C; C não está livre, então C prefere X a Z? **Não**, então X continua livre

h = Y (Y está livre)

m = B; B está livre, então emparelhar Y a B

Algoritmo de Gale-Shapley - exemplo (iteração 3)

	preferências de H					emparelhamentos				
	V	**W**	**X**	**Y**	**Z**	**A**	**B**	**C**	**D**	**E**
1ª	A	A	D	A	C	V	Y	Z	X	
2ª	C	D	C	D	D	W			Y	
3ª	D	E	B	B	A				V	
4ª	E	B	A	E	B					

h = V (V está livre)

m = D; D não está livre, então D prefere V a Y? **Sim**: emparelhar V a D e Y fica livre

h = W (W não está livre)

h = X (X está livre)

m = C; C não está livre, então C prefere X a Z? **Não**, então X continua livre

h = Y (Y está livre)

m = B; B está livre, então emparelhar Y a B

h = Z (Z não está livre)

Algoritmo de Gale-Shapley - exemplo (iteração 3)

	preferências de H					emparelhamentos				
	V	**W**	**X**	**Y**	**Z**	**A**	**B**	**C**	**D**	**E**
1ª	A	A	D	A	C	V	Y	Z	X	
2ª	C	D	C	D	D	W			Y	
3ª	D	E	B	B	A				V	
4ª	E	B	A	E	B					

h = V (V está livre)

m = D; D não está livre, então D prefere V a Y? **Sim**: emparelhar V a D e Y fica livre

h = W (W não está livre)

h = X (X está livre)

m = C; C não está livre, então C prefere X a Z? **Não**, então X continua livre

h = Y (Y está livre)

m = B; B está livre, então emparelhar Y a B

h = Z (Z não está livre)

Algoritmo de Gale-Shapley - exemplo (iteração 4)

preferências de H						emparelhamentos				
	V	**W**	**X**	**Y**	**Z**	**A**	**B**	**C**	**D**	**E**
1ª	A	A	D	A	C	V	Y	Z	X	
2ª	C	D	C	D	D	W			Y	
3ª	D	E	B	B	A				V	
4ª	E	B	A	E	B					

h = V (V não está livre)

h = W (W não está livre)

h = X (X está livre)

m = B; B não está livre, então B prefere X a Y? **Sim**: emparelhar X a B e Y fica livre

Algoritmo de Gale-Shapley - exemplo (iteração 4)

preferências de H						emparelhamentos				
	V	**W**	**X**	**Y**	**Z**	**A**	**B**	**C**	**D**	**E**
1ª	A	A	D	A	C	V	Y	Z	X	
2ª	C	D	C	D	D	W	X		Y	
3ª	D	E	B	B	A				V	
4ª	E	B	A	E	B					

h = V (V não está livre)

h = W (W não está livre)

h = X (X está livre)

m = B; B não está livre, então B prefere X a Y? **Sim**: emparelhar X a B e Y fica livre

h = Y (Y não está livre)

m = E; E está livre, então emparelhar Y a E

Algoritmo de Gale-Shapley - exemplo (iteração 4)

	preferências de H					emparelhamentos				
	V	**W**	**X**	**Y**	**Z**	**A**	**B**	**C**	**D**	**E**
1ª	A	A	D	A	C	V	Y	Z	X	Y
2ª	C	D	C	D	D	W	X		Y	
3ª	D	E	B	B	A				V	
4ª	E	B	A	E	B					

h = V (V não está livre)

h = W (W não está livre)

h = X (X está livre)

m = B; B não está livre, então B prefere X a Y? **Sim**: emparelhar X a B e Y fica livre

h = Y (Y não está livre)

m = E; E está livre, então emparelhar Y a E

h = Z (Z não está livre)

Algoritmo de Gale-Shapley - exemplo (iteração 4)

preferências de H						emparelhamentos				
	V	**W**	**X**	**Y**	**Z**	**A**	**B**	**C**	**D**	**E**
1ª	A	A	D	A	C	V	Y	Z	X	Y
2ª	C	D	C	D	D	W	X		Y	
3ª	D	E	B	B	A				V	
4ª	E	B	A	E	B					

h = V (V não está livre)

h = W (W não está livre)

h = X (X está livre)

m = B; B não está livre, então B prefere X a Y? **Sim**: emparelhar X a B e Y fica livre

h = Y (Y não está livre)

m = E; E está livre, então emparelhar Y a E

h = Z (Z não está livre)

Algoritmo de Gale-Shapley - exemplo (iteração 5)

preferências de H						emparelhamentos				
	V	**W**	**X**	**Y**	**Z**	**A**	**B**	**C**	**D**	**E**
1ª	A	A	D	A	C	V	Y	Z	X	Y
2ª	C	D	C	D	D	W	X		Y	
3ª	D	E	B	B	A				V	
4ª	E	B	A	E	B					

h = V (V não está livre)

h = W (W não está livre)

h = X (X não está livre)

h = Y (Y não está livre)

h = Z (Z não está livre)

Como todos os elementos do conjunto H estão emparelhados (não estão livres),

então **terminar o algoritmo**

Algoritmo de Gale-Shapley - exemplo (resultado final)

preferências de H						emparelhamentos				
	V	**W**	**X**	**Y**	**Z**	**A**	**B**	**C**	**D**	**E**
1ª	A	A	D	A	C	V	Y	Z	X	Y
2ª	C	D	C	D	D	W	X		Y	
3ª	D	E	B	B	A				V	
4ª	E	B	A	E	B					

A solução é a seguinte:

(V, D) = (Vítor, Débora)

(W, A) = (Wilson, Ana)

(X, B) = (Xavier, Beatriz)

(Y, E) = (Yuri, Erica)

(Z, C) = (Zeca, Carolina)

Algoritmo de Gale-Shapley

- O emparelhamento obtido pelo Algoritmo de Gale-Shapley é ótimo para os homens e péssimo para as mulheres
  - qualquer homem fica com a melhor parceira que pode ter em qualquer emparelhamento estável
  - cada mulher fica com o pior parceiro
- A situação inverte-se se passarem a ser as mulheres que se propõem
- Foi criada uma extensão do algoritmo de Gale-Shapley, ainda da autoria dos mesmos autores, a qual permitiu reconhecer estas e outras propriedades estruturais das soluções do problema

Extensão do algoritmo de Gale-Shapley

```
Algoritmo Gale-Shapley
  considerar inicialmente que todas as pessoas estão livres
  enquanto (houver algum homem h livre) fazer
     seja m a primeira mulher na lista atual de h
     se (algum homem p estiver noivo de m) então
        p passará a estar livre
     fim_se
     h e m ficam noivos
     para (cada sucessor h' de h na lista de m) fazer
        retirar o par (h', m) das listas de h' e m
     fim_para
  fim_enquanto
fim_algoritmo
```

Variantes do Problema dos Casamentos Estáveis

- O trabalho de Gale e Shapley teve como principal motivação a resolução do problema de colocação de alunos em cursos universitários nos EUA
  - cada aluno candidata-se a algumas universidades e define uma lista de preferências (pode ser incompleta) ordenadas estritamente
  - cada universidade tem um certo número de vagas e ordena estritamente os seus candidatos, podendo não aceitar alguns (as universidades podem ter listas diferentes)
  - pretende-se colocar os alunos de acordo com as preferências mútuas: os alunos propõem-se às universidades, resultando um emparelhamento ótimo para os alunos
  - nesta variante, designada por StableMarriageWithIncompleteLists,
    - as vagas correspondem às mulheres e os alunos aos homens
  - um **emparelhamento** será um conjunto **E** de pares (**h**, m) com **h** $\in$ **H** e **m** $\in$ **M**, tq
    - **h** e **m** se consideram mutuamente aceitáveis,
    - não existem pares em **E** que partilhem elementos, e
    - não existem pares de **H×M** que possam ser acrescentados a **E**

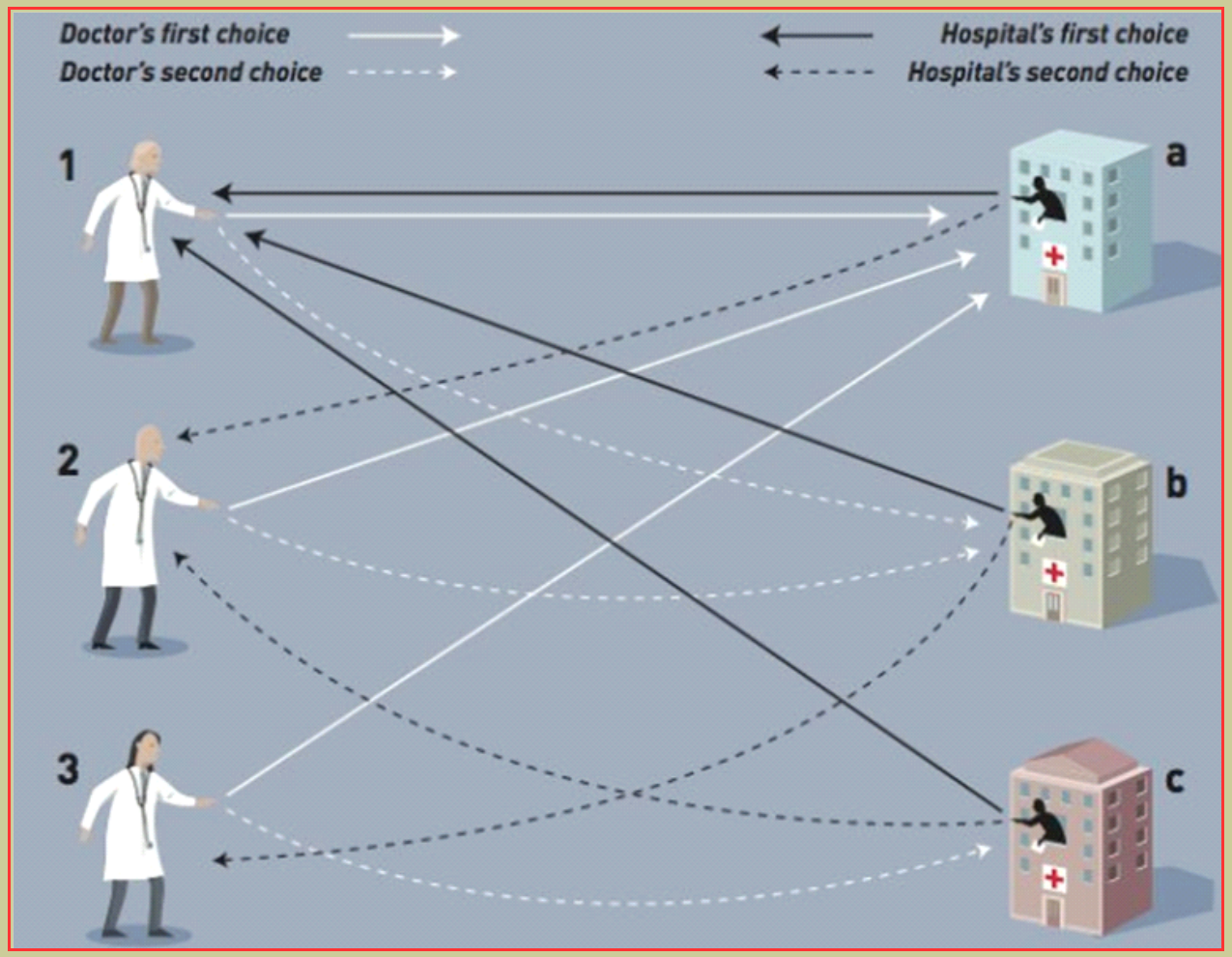
Variantes do Problema dos Casamentos Estáveis

- Alguns anos depois de 1962, criou-se um algoritmo, no essencial análogo que estava já a ser usado desde 1952 nos EUA (no National Intern Matching), para colocação de estudantes de medicina nos hospitais para realizarem o internato
  - também aqui, cada hospital tem uma lista de preferências própria
  - no entanto, são os hospitais que se propõem aos candidatos, resultando num emparelhamento ótimo para os hospitais
  - o critério de estabilidade é reformulado do seguinte modo: um **emparelhamento** é **instável** se e só se existir um candidato **r** e um hospital **h** tais que:
    - **h** é aceitável para **r** e **r** é aceitável para **h**,
    - o candidato **r** não ficou colocado ou prefere **h** ao seu atual hospital, e
    - **h** ficou com vagas por preencher ou **h** prefere **r** a pelo menos um dos candidatos com que ficou;

    caso contrário, diz-se **estável**

Variantes do Problema dos Casamentos Estáveis

- Representação gráfica do problema de colocação de estudantes de medicina nos hospitais para realizarem o internato



Variantes do Problema dos Casamentos Estáveis

- Concurso de Colocação de Professores em Portugal
  - as escolas (ou vagas) correspondem aos hospitais (ou mulheres) e
  - os opositores ao concurso correspondem aos internos (ou homens)
  - A diferença essencial está na existência de uma lista de graduação dos opositores (com prioridades), o que de certo modo, faz com que todas as escolas (i.e., mulheres) tenham exactamente as mesmas preferências pelos candidatos (i.e., homens)