4. Grafos – Problemas envolvendo grafos/redes

1. Conceitos gerais de Redes

- Quando se associam valores aos vértices e/ou às arestas de um grafo,
  - o grafo designa-se geralmente por **rede**,
  - os vértices e arestas designam-se por **nós/nodos** e **arcos**, respetivamente
- Uma rede pode ser representada por G = (V, A, C), em que
  - (V, A) é um grafo e
  - C é o conjunto de valores associados aos arcos (“comprimentos“): ao arco (i, j) está associado o valor c_{ij}
- De uma maneira geral, os conceitos utilizados em grafos são extensíveis às redes.
- O “comprimento“ do caminho **p** de S para T na rede G é a soma dos “comprimentos“ dos arcos que pertencem àquele caminho: $C(p) = \sum_{(i,j) \in p} c_{ij}$
- Define-se **árvore mínima** (**árvore de caminhos mais curtos**) com raiz em S, como a árvore que contém todos os vértices de V acessíveis a partir de S, em que para cada nó $\mathbf{n_2}$ o único caminho de **S** para $\mathbf{n_2}$ é o caminho mais curto (de comprimento mínimo) na rede G que liga **S** a $\mathbf{n_2}$

2. O Problema da Árvore Abrangente Mínima

- Um subgrafo que seja uma árvore e contenha todos os vértices do grafo é designado por **árvore abrangente** (árvore total — ”Spanning Tree”)
- A **árvore abrangente mínima** (“Minimum Spanning Tree”) é a árvore abrangente com o menor comprimento entre todas as árvores abrangentes.
- O comprimento de uma árvore abrangente é o somatório dos comprimentos associados aos respectivos arcos.
- Note-se que, em geral, a árvore abrangente mínima é diferente da árvore de caminho mais curto entre um nó origem e todos os outros nós da rede (calculada pelo algoritmo de Dijkstra).

2. O Problema da Árvore Abrangente Mínima - Algoritmo de PRIM (versão 1)

Passo 1.

- Tome-se arbitrariamente um nó S e atribui-se-lhe um rótulo permanente nulo: $\pi_S = 0$
- Aos restantes nós da rede atribuem-se rótulos temporários:

  $\pi_j = C_{Sj}$ se $(S, j) \in A$

  $\pi_j = \infty$ se $(S, j) \notin A$

- Permanentes = { S }
- Temporários = N - { S }

2. O Problema da Árvore Abrangente Mínima - Algoritmo de PRIM (versão 1)

Passo 2.

- k = nó com rótulo temporário que possua menor valor (que é vizinho de um nó **i**)
- Permanentes = Permanentes $\cup$ { k }
- Temporários = Temporários - { k }
- O arco com o mínimo valor $C_{ik} = \pi_k$ passa a fazer parte da árvore abrangente mínima
- <u>Se</u> Temporários = $\varnothing$
- <u>Então</u> STOP (*foi determinada a árvore abrangente mínima*)

Passo 3.

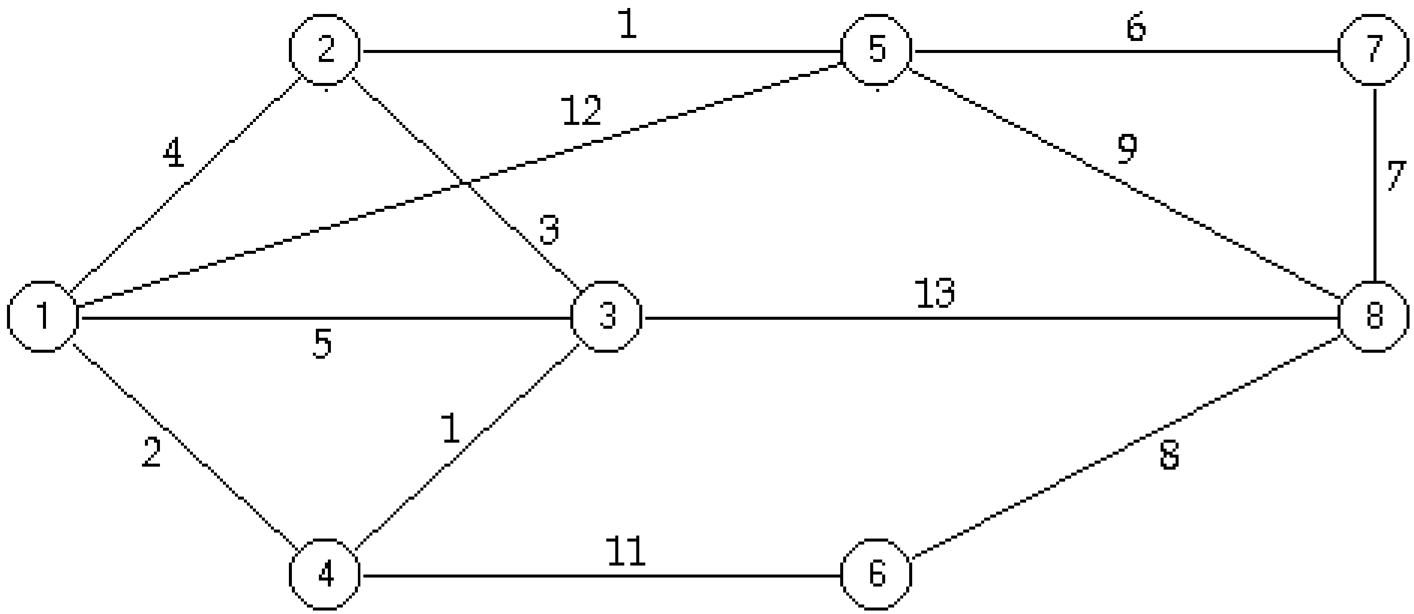
- <u>Para</u> todo o $j \in N$ tal que $(k, j) \in A$ e $j \in$ Temporários <u>Fazer</u>

  $\pi_j = \min \{ \pi_j, C_{kj} \}$

- Voltar ao Passo 2

2. O Problema da Árvore Abrangente Mínima - Algoritmo de PRIM (versão 1)

Exemplo: Determinar a árvore abrangente mínima da seguinte rede

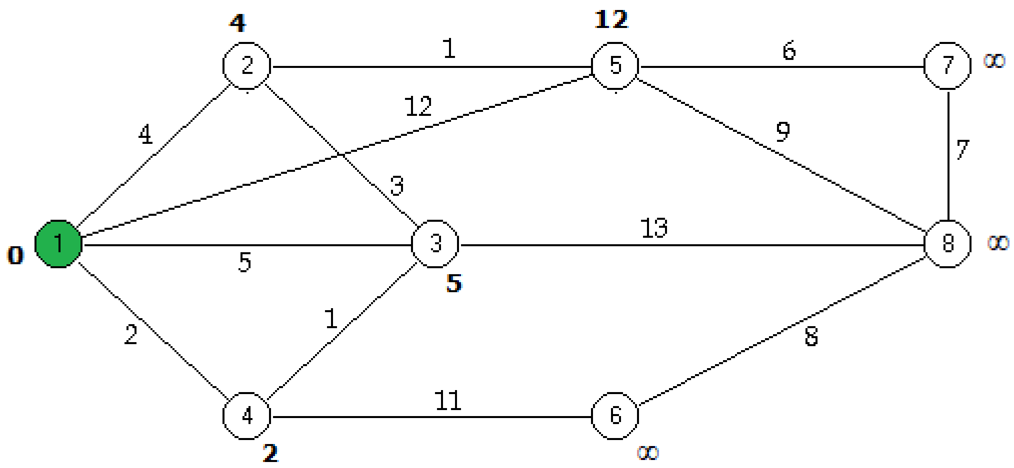


2. O Problema da Árvore Abrangente Mínima - Algoritmo de PRIM (versão 1)

Exemplo: Determinar a árvore abrangente mínima da seguinte rede

Passo 1.

- Colocar rótulo permanente no nó 1 e rótulos temporários nos restantes nós :
  - $\pi_1 = 0$
  - $\pi_2 = 4$; $\pi_3 = 5$; $\pi_4 = 2$; $\pi_5 = 12$
  - $\pi_6 = \pi_7 = \pi_8 = \infty$
- Permanentes = { 1 }
- Temporários = { 2, 3, 4, 5, 6, 7, 8 }



2. O Problema da Árvore Abrangente Mínima - Algoritmo de PRIM (versão 1)

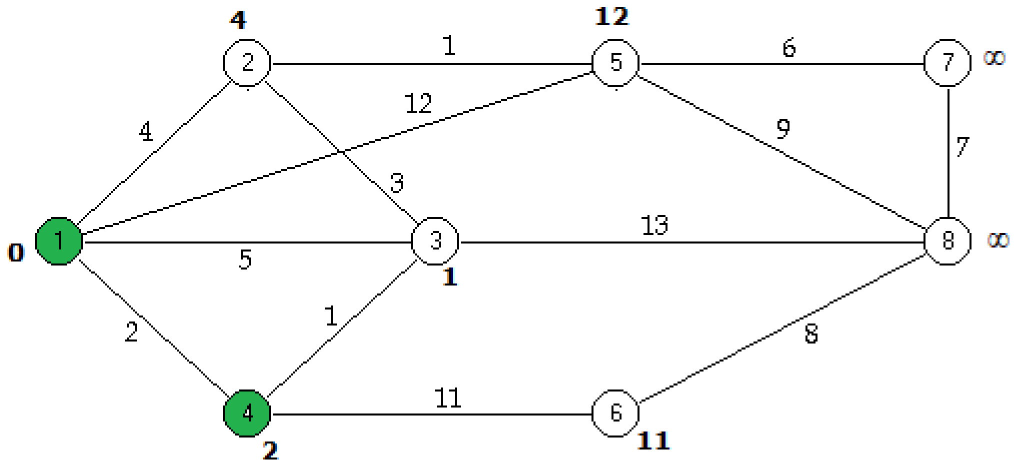
Exemplo: Determinar a árvore abrangente mínima da seguinte rede

Passo 2.

- $k = 4$
- Permanentes = Permanentes $\cup$ { 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 $C_{14} = \pi_4 = 2$

Passo 3.

- $\pi_3 = \min \{ 5, 1 \} = 1$
- $\pi_6 = \min \{ \infty, 11 \} = 11$



2. O Problema da Árvore Abrangente Mínima - Algoritmo de PRIM (versão 1)

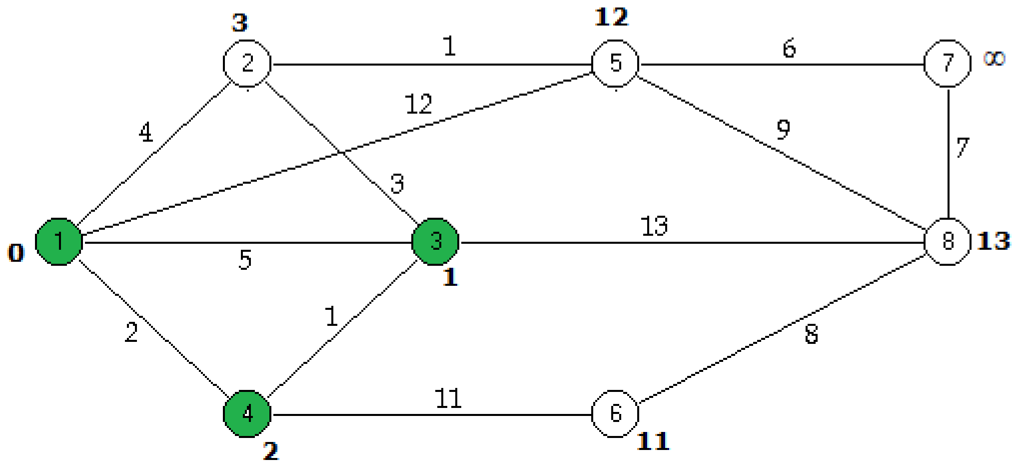
Exemplo: Determinar a árvore abrangente mínima da seguinte rede

Passo 2.

- $k = 3$
- Permanentes = Permanentes $\cup$ { 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 $C_{43} = \pi_3 = 1$

Passo 3.

- $\pi_2 = \min \{ 4, 3 \} = 3$
- $\pi_8 = \min \{ \infty, 13 \} = 13$



2. O Problema da Árvore Abrangente Mínima - Algoritmo de PRIM (versão 1)

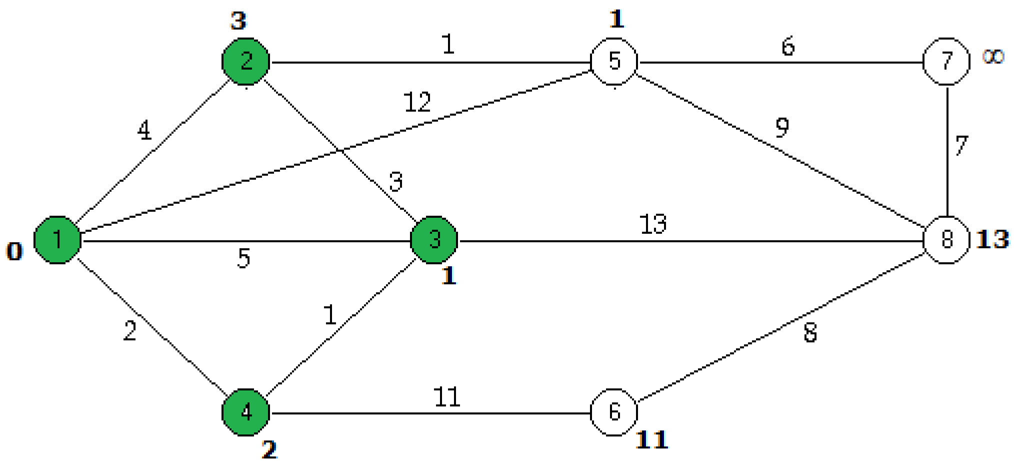
Exemplo: Determinar a árvore abrangente mínima da seguinte rede

Passo 2.

- $k = 2$
- Permanentes = Permanentes $\cup$ { 2 } = { 1, 4, 3, 2 }
- Temporários = Temporários – { 3 } = { 5, 6, 7, 8 }
- O arco (3, 2) passa a fazer parte da árvore abrangente mínima, pois $C_{32} = \pi_2 = 3$

Passo 3.

- $\pi_5 = \min \{ 12, 1 \} = 1$



2. O Problema da Árvore Abrangente Mínima - Algoritmo de PRIM (versão 1)

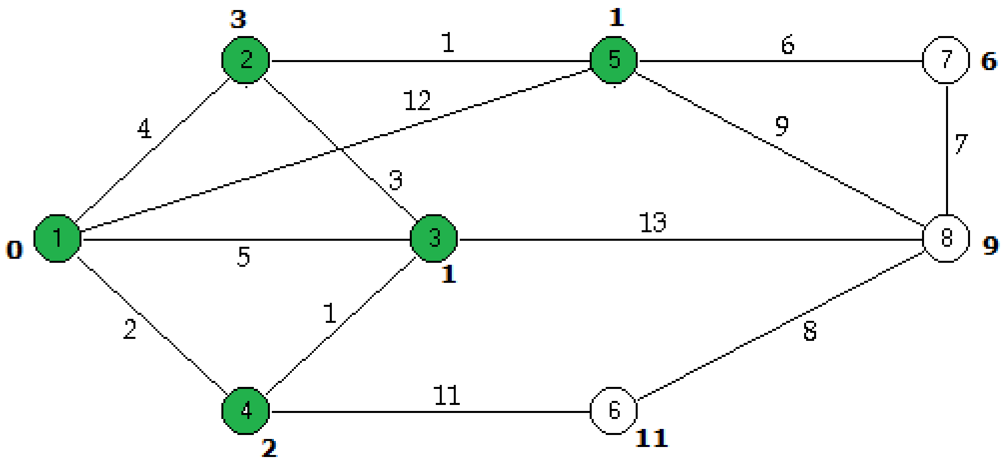
Exemplo: Determinar a árvore abrangente mínima da seguinte rede

Passo 2.

- $k = 5$
- Permanentes = Permanentes $\cup$ { 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 $C_{25} = \pi_5 = 1$

Passo 3.

- $\pi_7 = \min \{ \infty, 6 \} = 6$
- $\pi_8 = \min \{ 13, 9 \} = 9$



2. O Problema da Árvore Abrangente Mínima - Algoritmo de PRIM (versão 1)

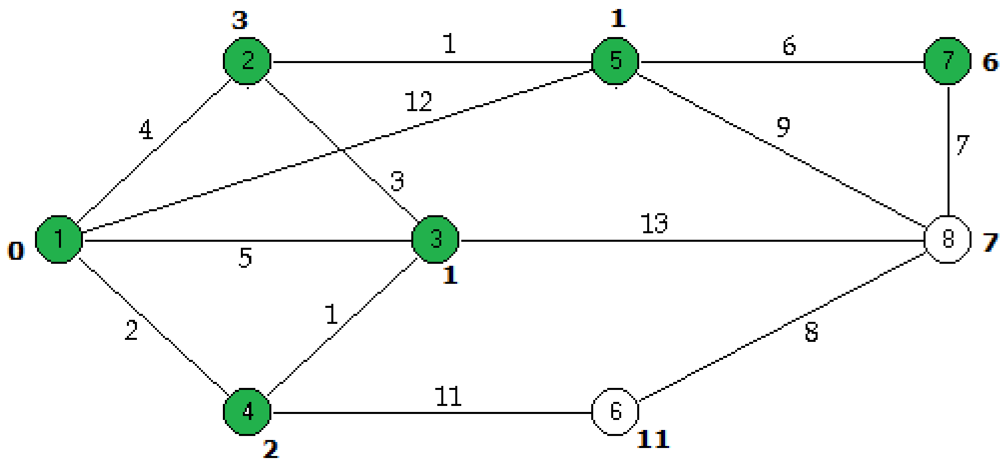
Exemplo: Determinar a árvore abrangente mínima da seguinte rede

Passo 2.

- $k = 7$
- Permanentes = Permanentes $\cup$ { 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 $C_{57} = \pi_7 = 6$

Passo 3.

- $\pi_8 = \min \{ 9, 7 \} = 7$



2. O Problema da Árvore Abrangente Mínima - Algoritmo de PRIM (versão 1)

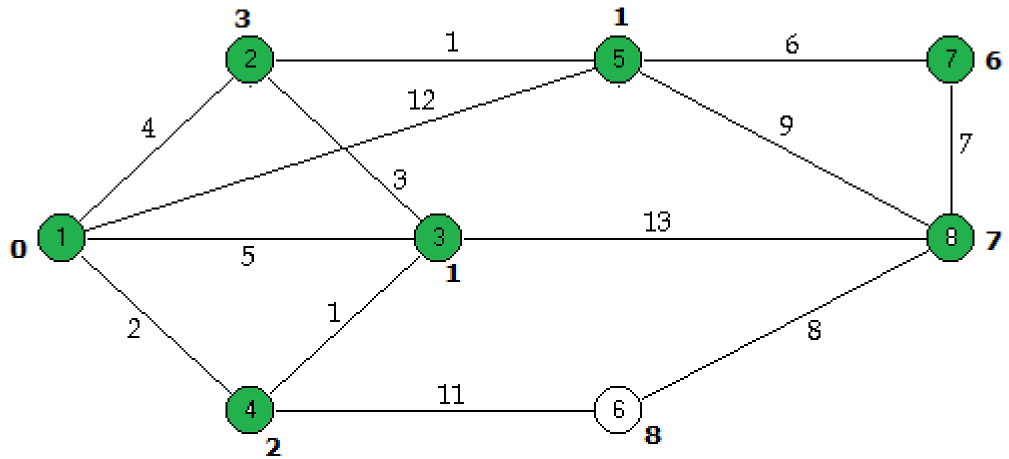
Exemplo: Determinar a árvore abrangente mínima da seguinte rede

Passo 2.

- $k = 8$
- Permanentes = Permanentes $\cup$ { 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 $C_{78} = \pi_8 = 7$

Passo 3.

- $\pi_6 = \min \{ 11, 8 \} = 8$

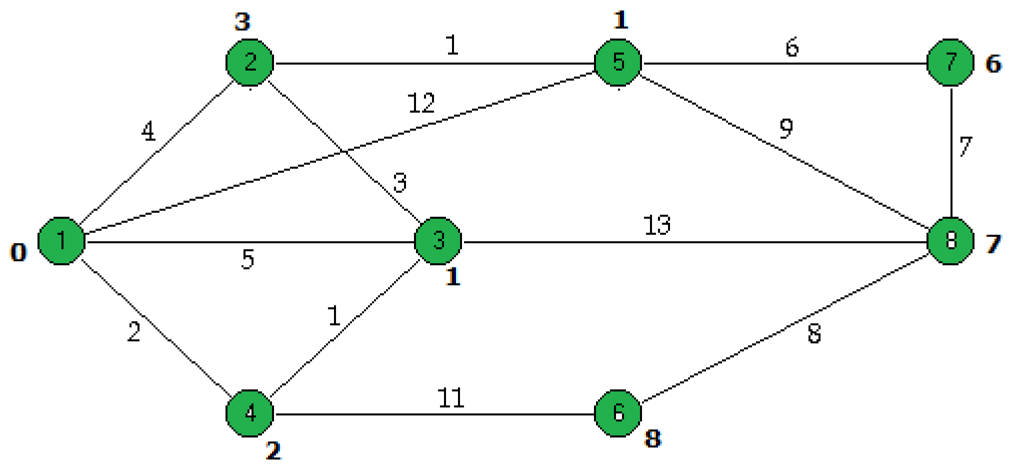


2. O Problema da Árvore Abrangente Mínima - Algoritmo de PRIM (versão 1)

Exemplo: Determinar a árvore abrangente mínima da seguinte rede

Passo 2.

- $k = 6$
- Permanentes = Permanentes $\cup$ { 6 } = { 1, 4, 3, 2, 5, 7, 8, 6 }
- Temporários = Temporários – { 6 } = $\varnothing$
- O arco (8, 6) passa a fazer parte da árvore abrangente mínima, pois $C_{86} = \pi_6 = 8$
- Como Temporários = $\varnothing$ STOP (foi determinada a árvore abrangente mínima)

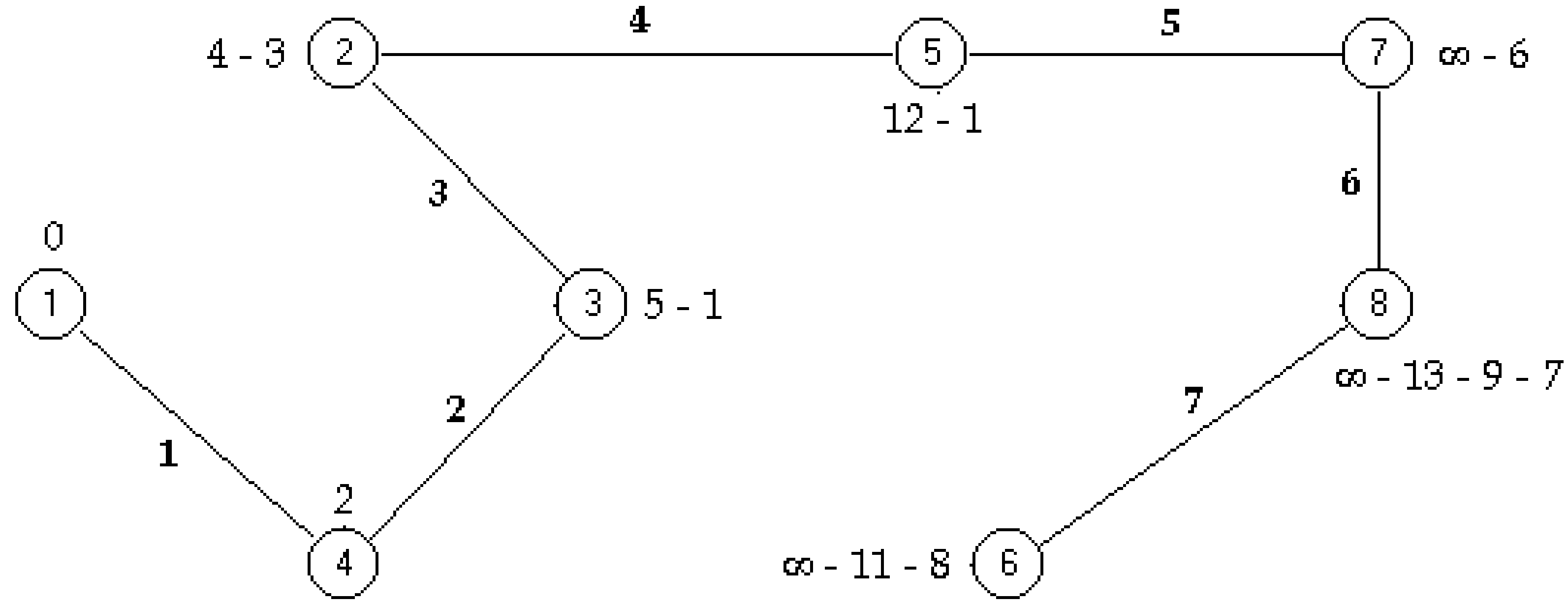


2. O Problema da Árvore Abrangente Mínima - Algoritmo de PRIM (versão 1)

Exemplo: Determinar a árvore abrangente mínima da seguinte rede

Resultados após o final do algoritmo:

- a árvore abrangente mínima (árvore que visita todos os nós) é a seguinte:



- a árvore abrangente mínima tem um comprimento total igual a 28 (2 + 1 + 3 + 1 + 6 + 7 + 8)

2. O Problema da Árvore Abrangente Mínima - Algoritmo de PRIM (versão 2)

- Existe uma outra versão do algoritmo de PRIM, que opera sobre a matriz das distâncias (custos) da rede

Passo 1.

- *Riscar* a 1ª coluna e *marcar* a 1ª linha

Passo 2.

- Seleccionar o menor elemento (C_{ij}) das linhas *marcadas* (não considerar as colunas *riscadas*)
- Se tiverem todas as colunas *riscadas*, STOP — foi determinada a árvore abrangente mínima

Passo 3.

- *Riscar* a coluna **j** e *marcar* a linha **j**
- Voltar ao Passo 2

- Os arcos que constituem a árvore abrangente mínima, são os correspondentes aos seleccionados, e o seu valor é o somatório daqueles C_{ij}

2. O Problema da Árvore Abrangente Mínima - Algoritmo de PRIM (versão 2)

Exemplo: Determinar a árvore abrangente mínima da rede do exemplo anterior

Passo 1.

- Riscar a 1ª coluna e marcar a 1ª linha:

	1	2	3	4	5	6	7	8	
1	0	4	5	2	12	–	–	–	✓
2	4	0	3	–	1	–	–	–	
3	5	3	0	1	–	–	–	–	
4	2	–	1	0	–	–	–	13	
5	12	1	–	–	0	–	6	9	
6	–	–	–	11	–	0	–	8	
7	–	–	–	–	6	–	0	7	
8	–	–	13	–	9	8	7	0	

Passo 2.

- O menor elemento *não riscado* da linha 1 (a única seleccionada) é: $C_{14} = 2$

2. O Problema da Árvore Abrangente Mínima - Algoritmo de PRIM (versão 2)

Exemplo: Determinar a árvore abrangente mínima da rede do exemplo anterior

Passo 3.

- *Riscar* a 4ª coluna e *marcar* a 4ª linha:

	1	2	3	4	5	6	7	8	
1	0	4	5	**<u>2</u>**	12	–	–	–	✓
2	4	0	3	–	1	–	–	–	
3	5	3	0	1	–	–	–	–	
4	2	–	1	0	–	–	–	13	✓
5	12	1	–	–	0	–	6	9	
6	–	–	–	11	–	0	–	8	
7	–	–	–	–	6	–	0	7	
8	–	–	13	–	9	8	7	0	

Passo 2.

O menor elemento *não riscado* das linhas 1 e 4 (as seleccionadas) é: $C_{43} = 1$

2. O Problema da Árvore Abrangente Mínima - Algoritmo de PRIM (versão 2)

Exemplo: Determinar a árvore abrangente mínima da rede do exemplo anterior

Passo 3.

- *Riscar* a 3ª coluna e *marcar* a 3ª linha :

	1	2	3	4	5	6	7	8	
1	0	4	5	<u>2</u>	12	–	–	–	✓
2	4	0	3	–	1	–	–	–	
3	5	3	0	1	–	–	–	–	✓
4	2	–	<u>1</u>	0	–	–	–	13	✓
5	12	1	–	–	0	–	6	9	
6	–	–	–	11	–	0	–	8	
7	–	–	–	–	6	–	0	7	
8	–	–	13	–	9	8	7	0	

Passo 2.

- O menor elemento *não riscado* das linhas 1, 3 e 4 (as seleccionadas) é: $C_{32} = 3$

2. O Problema da Árvore Abrangente Mínima - Algoritmo de PRIM (versão 2)

Exemplo: Determinar a árvore abrangente mínima da rede do exemplo anterior

Passo 3.

- *Riscar* a 2ª coluna e *marcar* a 2ª linha:

	1	2	3	4	5	6	7	8	
1	0	4	5	<u>2</u>	12	–	–	–	✓
2	4	0	3	–	1	–	–	–	✓
3	5	<u>3</u>	0	1	–	–	–	–	✓
4	2	–	<u>1</u>	0	–	–	–	13	✓
5	12	1	–	–	0	–	6	9	
6	–	–	–	11	–	0	–	8	
7	–	–	–	–	6	–	0	7	
8	–	–	13	–	9	8	7	0	

Passo 2.

- O menor elemento *não riscado* das linhas 1, 2, 3 e 4 (as seleccionadas) é: $C_{25} = 1$

2. O Problema da Árvore Abrangente Mínima - Algoritmo de PRIM (versão 2)

Exemplo: Determinar a árvore abrangente mínima da rede do exemplo anterior

Passo 3.

- *Riscar* a 5ª coluna e *marcar* a 5ª linha:

	1	2	3	4	5	6	7	8	
1	0	4	5	<u>2</u>	12	–	–	–	✓
2	4	0	3	–	<u>1</u>	–	–	–	✓
3	5	<u>3</u>	0	1	–	–	–	–	✓
4	2	–	<u>1</u>	0	–	–	–	13	✓
5	12	1	–	–	0	–	6	9	✓
6	–	–	–	11	–	0	–	8	
7	–	–	–	–	6	–	0	7	
8	–	–	13	–	9	8	7	0	

Passo 2.

- O menor elemento *não riscado* das linhas 1, 2, 3, 4 e 5 (as seleccionadas) é: $C_{57} = 6$

2. O Problema da Árvore Abrangente Mínima - Algoritmo de PRIM (versão 2)

Exemplo: Determinar a árvore abrangente mínima da rede do exemplo anterior

Passo 3.

- *Riscar* a 7ª coluna e *marcar* a 7ª linha:

	1	2	3	4	5	6	7	8	
1	0	4	5	<u>2</u>	12	–	–	–	✓
2	4	0	3	–	<u>1</u>	–	–	–	✓
3	5	<u>3</u>	0	1	–	–	–	–	✓
4	2	–	<u>1</u>	0	–	–	–	13	✓
5	12	1	–	–	0	–	<u>6</u>	9	✓
6	–	–	–	11	–	0	–	8	
7	–	–	–	–	6	–	0	7	✓
8	–	–	13	–	9	8	7	0	

Passo 2.

- O menor elemento *não riscado* das linhas 1, 2, 3, 4, 5 e 7 (as seleccionadas) é: $C_{78} = 7$

2. O Problema da Árvore Abrangente Mínima - Algoritmo de PRIM (versão 2)

Exemplo: Determinar a árvore abrangente mínima da rede do exemplo anterior

Passo 3.

- *Riscar* a 8ª coluna e *marcar* a 8ª linha:

	1	2	3	4	5	6	7	8	
1	0	4	5	<u>2</u>	12	–	–	–	✓
2	4	0	3	–	<u>1</u>	–	–	–	✓
3	5	<u>3</u>	0	1	–	–	–	–	✓
4	2	–	<u>1</u>	0	–	–	–	13	✓
5	12	1	–	–	0	–	<u>6</u>	9	✓
6	–	–	–	11	–	0	–	8	
7	–	–	–	–	6	–	0	<u>7</u>	✓
8	–	–	13	–	9	8	7	0	✓

Passo 2.

- O menor elemento *não riscado* das linhas 1, 2, 3, 4, 5, 7 e 8 (as seleccionadas) é : $C_{86} = 8$

2. O Problema da Árvore Abrangente Mínima - Algoritmo de PRIM (versão 2)

Exemplo: Determinar a árvore abrangente mínima da rede do exemplo anterior

Passo 3.

- *Riscar* a 6ª coluna e *marcar* a 6ª linha:

	1	2	3	4	5	6	7	8	
1	0	4	5	<u>**2**</u>	12	–	–	–	✓
2	4	0	3	–	<u>**1**</u>	–	–	–	✓
3	5	<u>**3**</u>	0	1	–	–	–	–	✓
4	2	–	<u>**1**</u>	0	–	–	–	13	✓
5	12	1	–	–	0	–	<u>**6**</u>	9	✓
6	–	–	–	11	–	0	–	8	
7	–	–	–	–	6	–	0	<u>**7**</u>	✓
8	–	–	13	–	9	<u>**8**</u>	7	0	✓

Passo 2.

- Todas as colunas se encontram riscadas, então foi determinada a Árvore Abrangente Mínima

2. O Problema da Árvore Abrangente Mínima - Algoritmo de PRIM (versão 2)

Exemplo: Determinar a árvore abrangente mínima da rede do exemplo anterior

Resultados no final do algoritmo

A árvore abrangente mínima é constituída pelos seguintes arcos:

(1, 4), (2, 5), (3, 2), (4, 3), (5, 7), (7, 8) e (8, 6)

O valor da árvore abrangente mínima é:

2 + 1 + 3 + 1 + 6 + 7 + 8 = 28

Como se pode verificar, a árvore é a mesma que foi determinada pela versão anterior

3. O problema do Caminho Mais Curto

3.1. 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 um 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)

3. O problema do Caminho Mais Curto

3.1. Conceitos gerais

- Sejam S e T dois nós de uma rede G = (V, A, C), em que a cada arco é associado apenas um valor (comprimento do arco)
- 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 entre os nós S e T tem por objetivo determinar o caminho de valor mínimo existente em P
- Ou seja, 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)

3. O problema do Caminho Mais Curto

3.1. 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,

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

- Existem vários algoritmos eficientes para resolver problemas de caminho mais curto, sendo os mais conhecidos os algoritmos
  - de Dijkstra (1 e 2) e
  - de Floyd (3).

3. O problema do Caminho Mais Curto

3.2. Algoritmo de Dijkstra

- Este algoritmo, que só pode ser aplicada a redes cujos arcos têm associados 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 $[\xi_i, \pi_i]$, o qual pode ser permanente ou temporário. Isto quer dizer o seguinte:

  $[\xi_i, \pi_i]$ permanente, representa <u>o caminho</u> mais curto de S para i

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

  $\pi_i \leftarrow$ valor do caminho mais curto de S para i

  $[\xi_i, \pi_i]$ temporário, representa <u>um caminho</u> mais curto de S para i

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

  $\pi_i \leftarrow$ 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

3. O problema do Caminho Mais Curto

3.2. 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 π, que se torna permanente,
  - depois varrer todos os seus adjacentes, de forma a atualizar os rótulos destes (temporários)
- O algoritmo termina quando não existirem nós com rótulos temporários
- Inicialmente apenas o nó S é permanente, sendo os restantes temporários

3. O problema do Caminho Mais Curto

3.2. Algoritmo de Dijkstra

Passo 1.

- $[\xi_S, \pi_S] = [S, 0]$ (caminho mais curto para S custa 0 e não tem nós intermédios)
- $[\xi_i, \pi_i] = [S, C_{Si}]$, $\forall\, i \in V - \{ S \}$ e $(S, i) \in A$
- $[\xi_i, \pi_i] = [-, \infty]$, $\forall\, i \in V - \{ S \}$ e $(S, i) \notin A$
- Temporários = V - { S } (Temporários = conjunto de nós com rótulos temporários)
- Permanentes = { S } (Permanentes = conjunto de nós com rótulos permanentes)

Passo 2.

- **Se** Temporários = $\emptyset$ **Então** STOP (todos os nós têm rótulos permanentes)
- k = nó de Temporários tal que π_k é mínimo ($k : \pi_k = \min \{ \pi_x ,\ x \in$ Temporários })
- Temporários = Temporários - { k }
- Permanentes = Permanentes $\cup$ { k } (k passou a permanente)

Passo 3.

- **Para** todo o $j \in V$ tal que $(k, j) \in A$ e $j \in$ Temporários **Fazer**

  Se $(\pi_k + C_{kj} < \pi_j)$

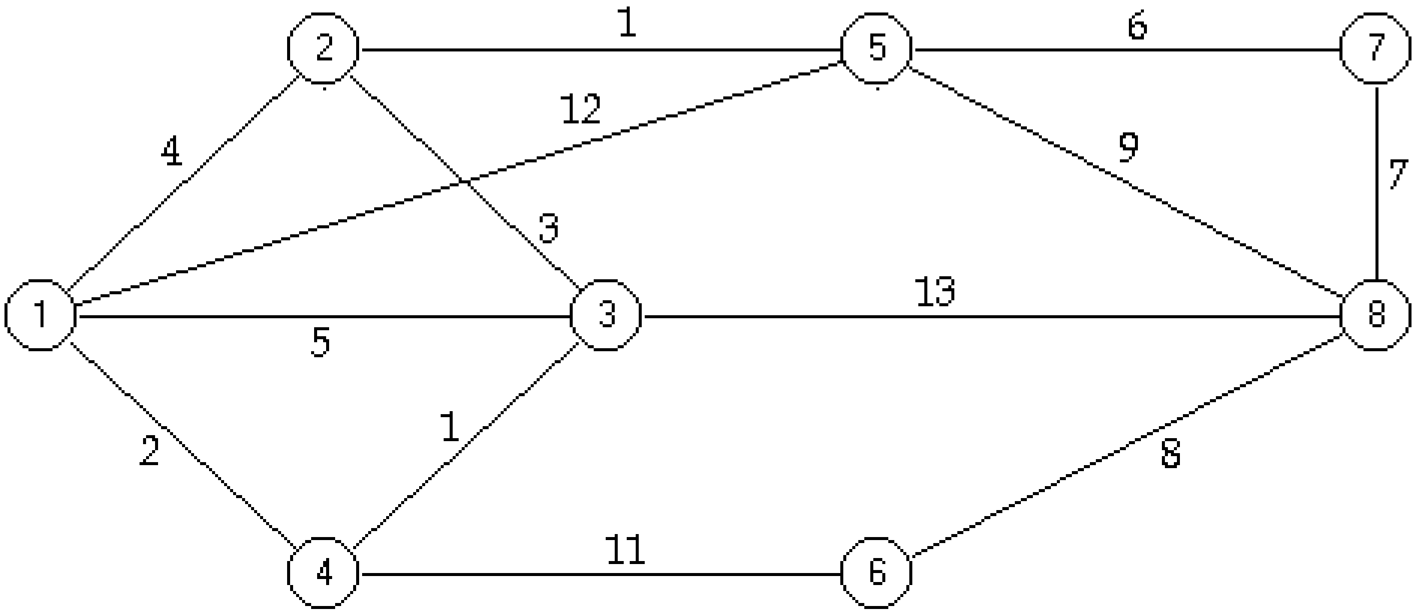
  Então $\pi_j = \pi_k + C_{kj}$

  $\xi_j = k$
- Regressar ao Passo 2

3. O problema do Caminho Mais Curto

3.2. Algoritmo de Dijkstra (exemplo)

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

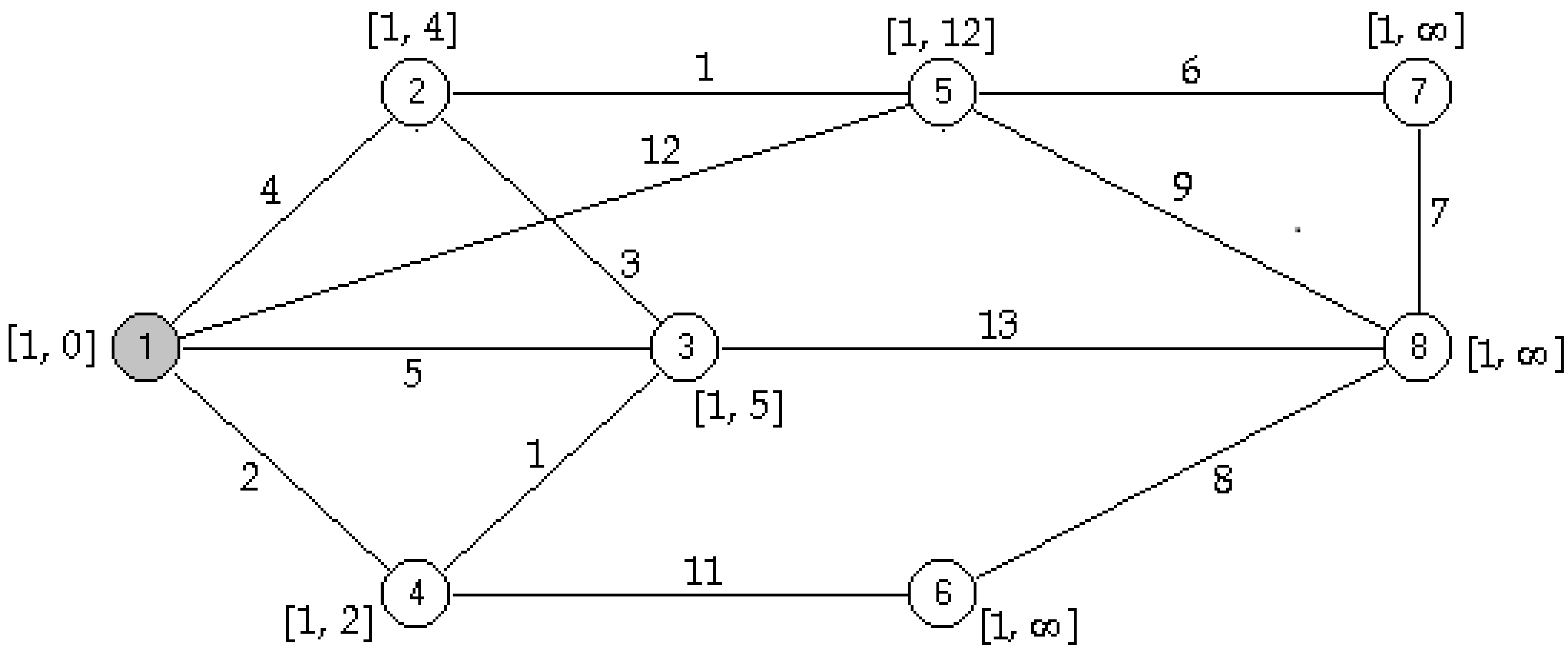


3. O problema do Caminho Mais Curto

3.2. Algoritmo de Dijkstra (exemplo)

Passo 1.

- Colocar rótulo permanente no nó 1 e rótulos temporários nos restantes nós



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

3. O problema do Caminho Mais Curto

3.2. Algoritmo de Dijkstra (exemplo)

Passo 2.

- k = 4, pois $\pi_4 = \min \{ \pi_2, \pi_3, \pi_4, \pi_5, \pi_6, \pi_7, \pi_8 \} = \min \{ 4, 5, 2, 12, \infty, \infty, \infty \}$
- Permanentes = Permanentes $\cup$ { 4 } = { 1, 4 }
- Temporários = Temporários - { 4 } = { 2, 3, 5, 6, 7, 8 }

3. O problema do Caminho Mais Curto

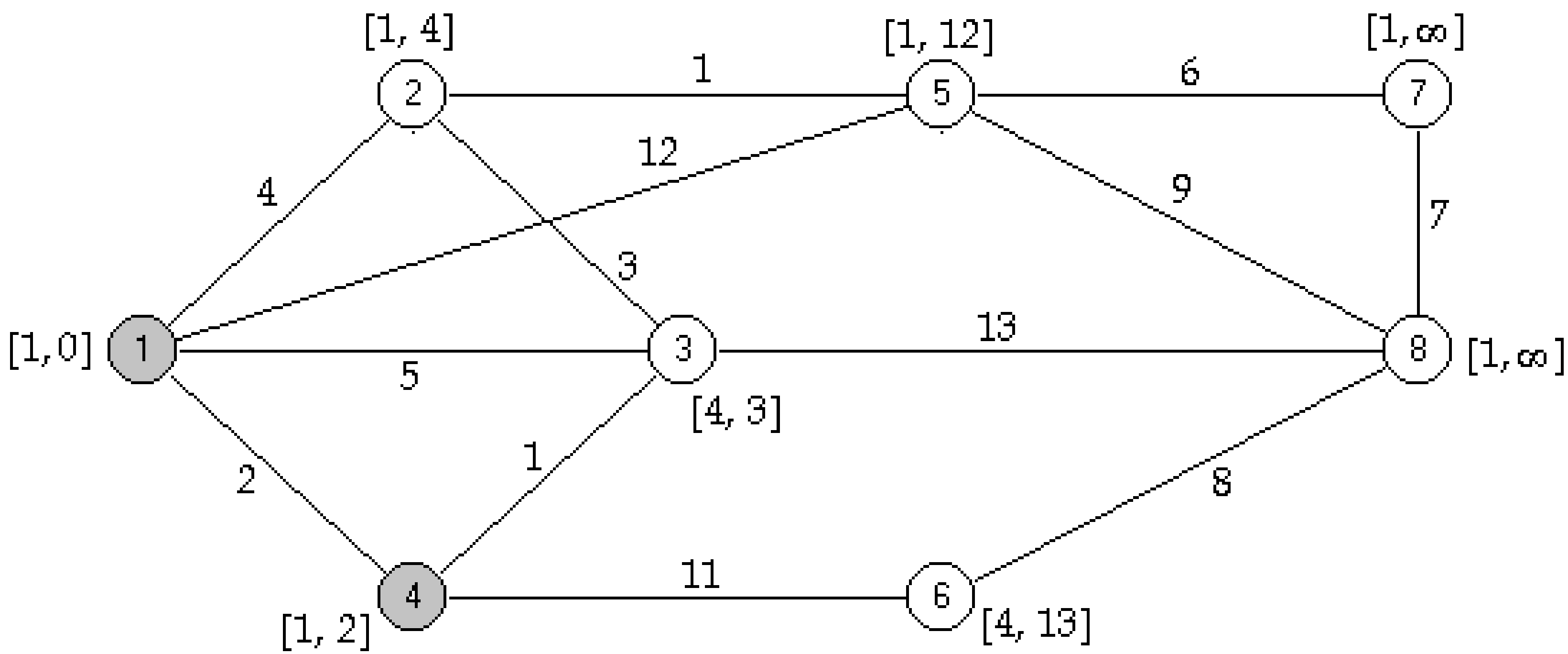
3.2. Algoritmo de Dijkstra (exemplo)

Passo 3.

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

$\pi_3 = \min \{ \pi_3, \pi_4 + C_{43} \} = \min \{ 5, 2 + 1 \} = 3 \quad \Rightarrow \quad [\xi_3, \pi_3] = [4, 3]$

$\pi_6 = \min \{ \pi_6, \pi_4 + C_{46} \} = \min \{ \infty, 2 + 11 \} = 13 \quad \Rightarrow \quad [\xi_6, \pi_6] = [4, 13]$



3. O problema do Caminho Mais Curto

3.2. Algoritmo de Dijkstra (exemplo)

Passo 2.

- k = 3, pois $\pi_3 = \{ \pi_2, \pi_3, \pi_5, \pi_6, \pi_7, \pi_8 \} = \min \{ 4, 3, 12, 13, \infty, \infty \}$
- Permanentes = Permanentes $\cup$ { 3 } = { 1, 3, 4 }
- Temporários = Temporários - { 3 } = { 2, 5, 6, 7, 8 }

3. O problema do Caminho Mais Curto

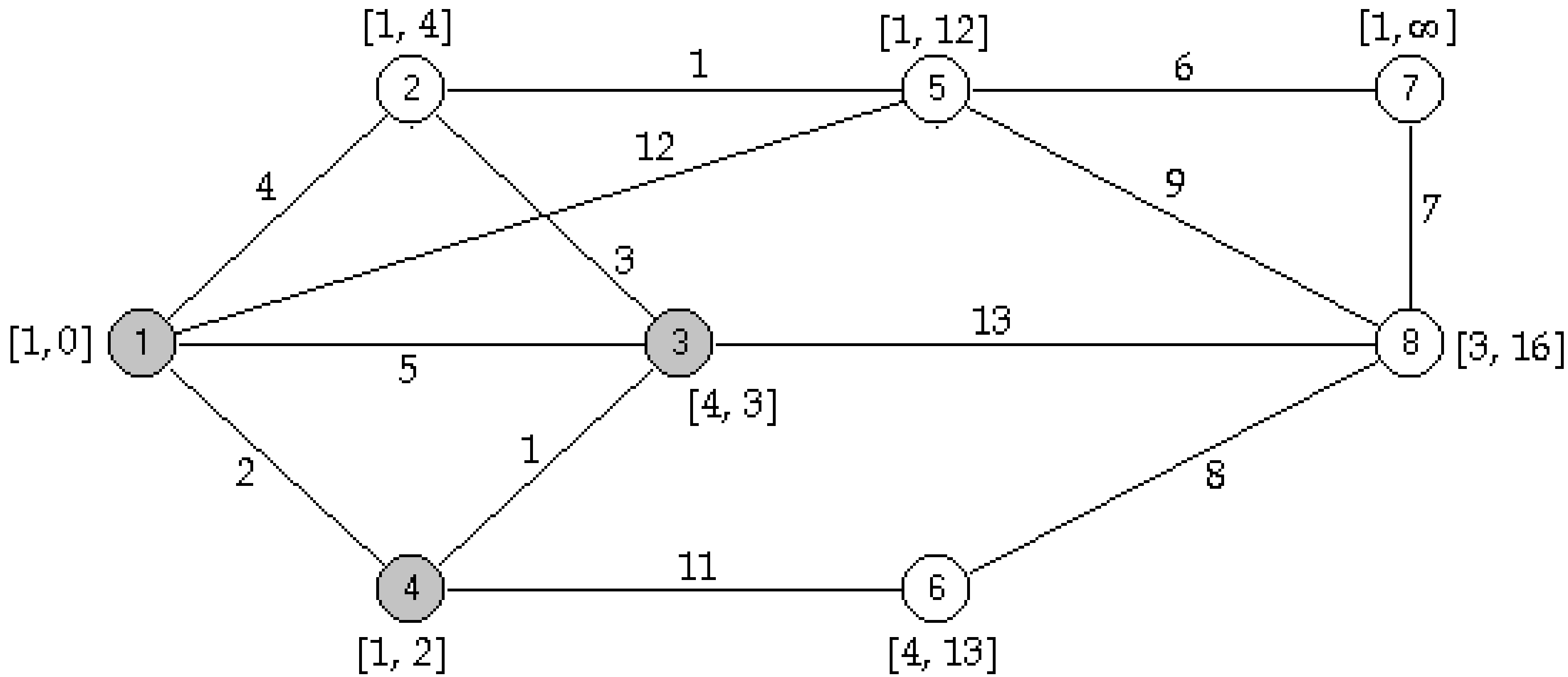
3.2. Algoritmo de Dijkstra (exemplo)

Passo 3.

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

$\pi_2 = \min\{\pi_2, \pi_3 + C_{32}\} = \min\{4, 3 + 3\} = 4 \quad \Rightarrow \quad [\xi_3, \pi_3] = [1, 4]$ (sem alteração)

$\pi_8 = \min\{\pi_8, \pi_3 + C_{38}\} = \min\{\infty, 3 + 13\} = 16 \quad \Rightarrow \quad [\xi_8, \pi_8] = [3, 16]$



3. O problema do Caminho Mais Curto

3.2. Algoritmo de Dijkstra (exemplo)

Passo 2.

- k = 2, pois π_2 = { π_2, π_5, π_6, π_7, π_8 } = min { 4, 12, 13, ∞, 16 }
- Permanentes = Permanentes $\cup$ { 2 } = { 1, 2, 3, 4 }
- Temporários = Temporários - { 2 } = { 5, 6, 7, 8 }

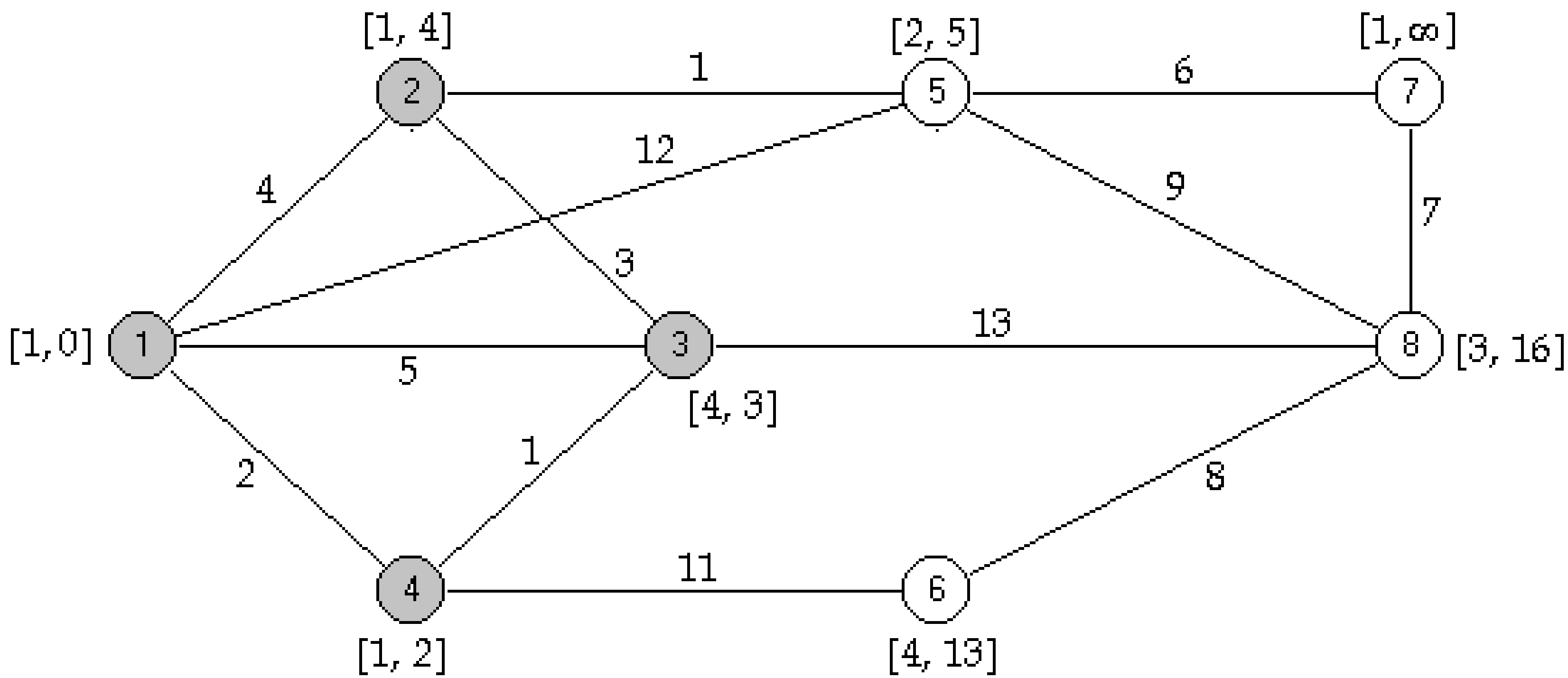
3. O problema do Caminho Mais Curto

3.2. Algoritmo de Dijkstra (exemplo)

Passo 3.

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

π_5 = min { π_5, π_2 + C_{25} } = min { 12, 4 + 1 } = 5 $\Rightarrow$ [ξ_5, π_5] = [2, 5]



3. O problema do Caminho Mais Curto

3.2. Algoritmo de Dijkstra (exemplo)

Passo 2.

- k = 5, pois $\pi_5 = \min \{ \pi_5, \pi_6, \pi_8 \} = \min \{ 5, 13, 16 \}$
- Permanentes = Permanentes $\cup$ { 5 } = { 1, 2, 3, 4, 5 }
- Temporários = Temporários - { 5 } = { 6, 7, 8 }

3. O problema do Caminho Mais Curto

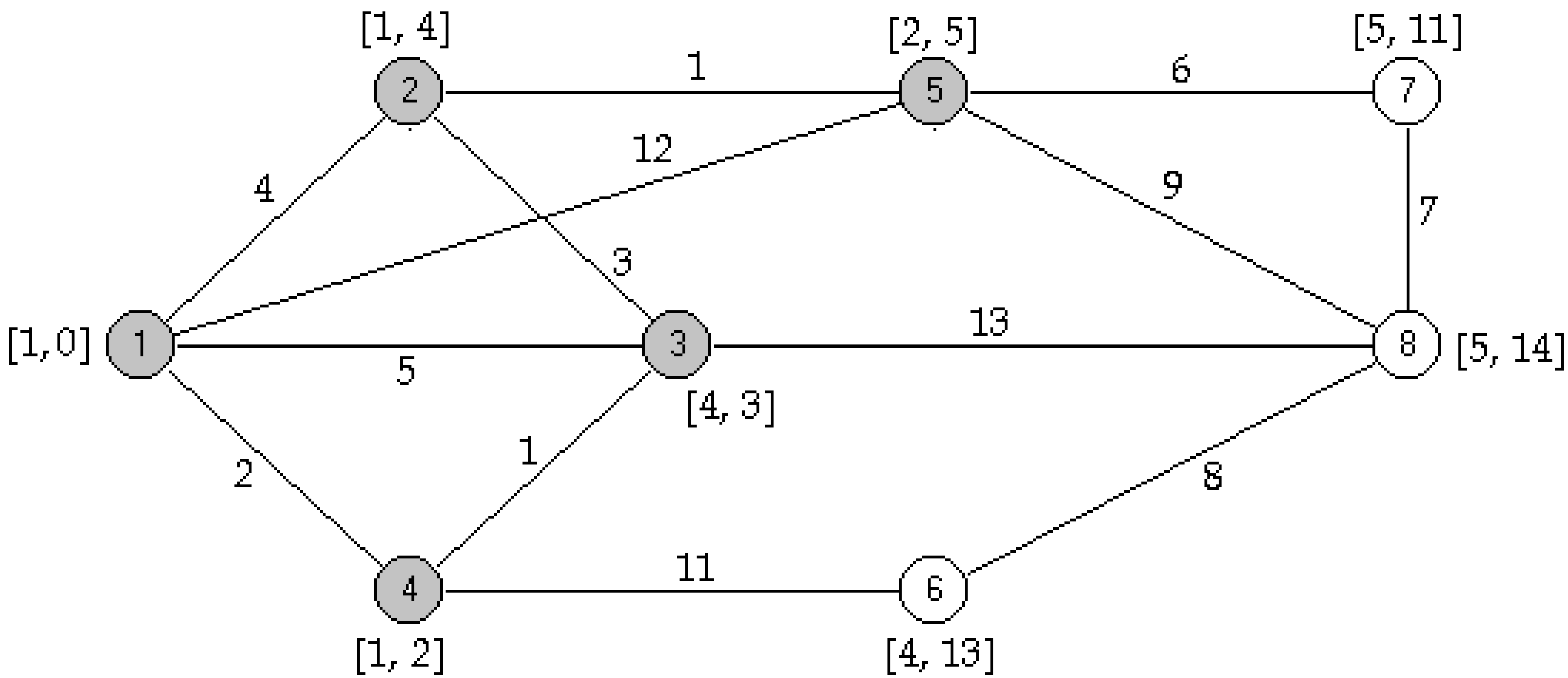
3.2. Algoritmo de Dijkstra (exemplo)

Passo 3.

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

π_7 = min { π_7, π_5 + C_{57} } = min { ∞, 5 + 6 } = 11 $\Rightarrow$ [ξ_7, π_7] = [5, 11]

π_8 = min { π_8, π_5 + C_{58} } = min { 16, 5 + 9 } = 14 $\Rightarrow$ [ξ_8, π_8] = [5, 14]



3. O problema do Caminho Mais Curto

3.2. Algoritmo de Dijkstra (exemplo)

Passo 2.

- $k = 7$, pois $\pi_7 = \min \{ \pi_6, \pi_7, \pi_8 \} = \min \{ 13, 11, 14 \}$
- Permanentes = Permanentes $\cup \{ 7 \} = \{ 1, 2, 3, 4, 5, 7 \}$
- Temporários = Temporários - $\{ 7 \} = \{ 6, 8 \}$

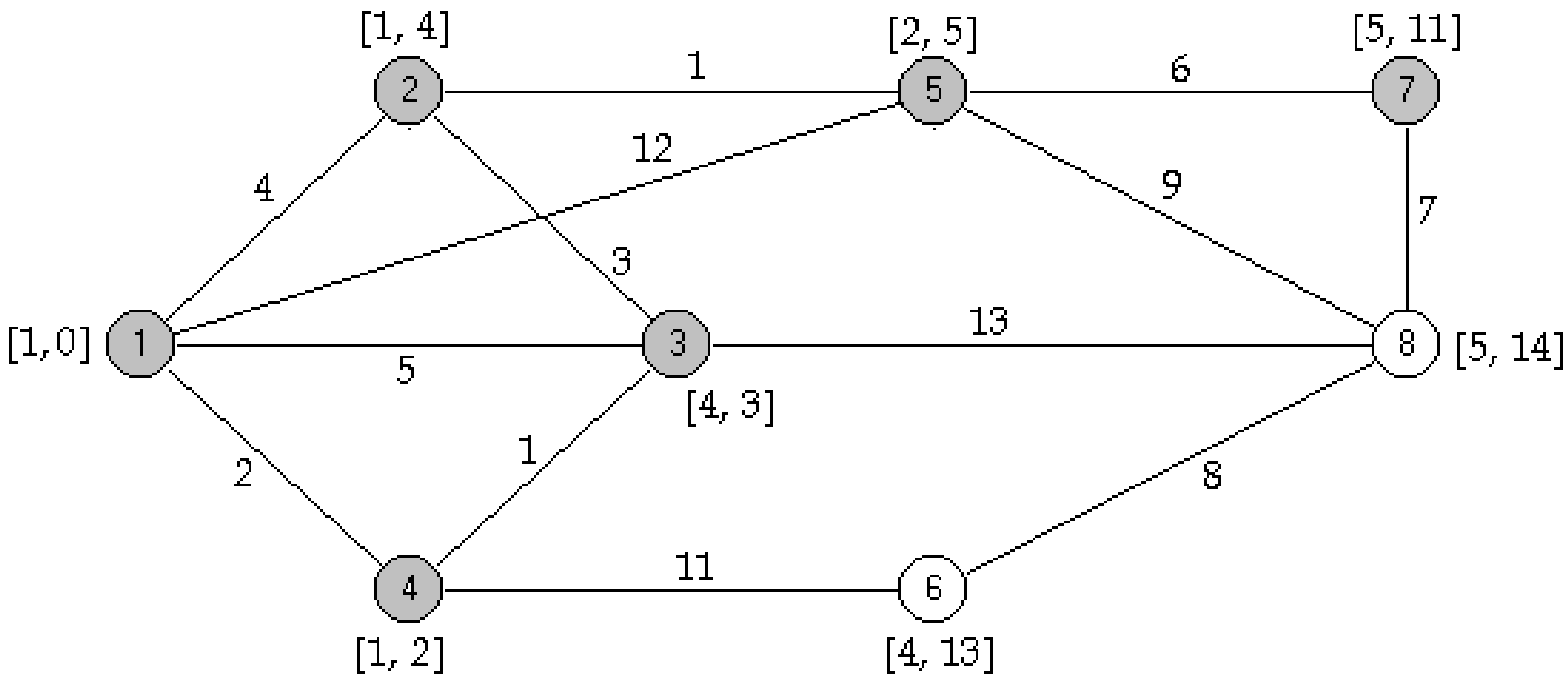
3. O problema do Caminho Mais Curto

3.2. Algoritmo de Dijkstra (exemplo)

Passo 3.

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

π_8 = min { π_8, π_7 + C_{78} } = min { 14, 11 + 7 } = 14 $\Rightarrow$ [ξ_8, π_8] = [5, 14] (sem alteração)



3. O problema do Caminho Mais Curto

3.2. Algoritmo de Dijkstra (exemplo)

Passo 2.

- k = 6, pois $\pi_6 = \min\{ \pi_6, \pi_8 \} = \min\{ 13, 14 \}$
- Permanentes = Permanentes $\cup$ { 6 } = { 1, 2, 3, 4, 5, 6, 7 }
- Temporários = Temporários - { 6 } = { 8 }

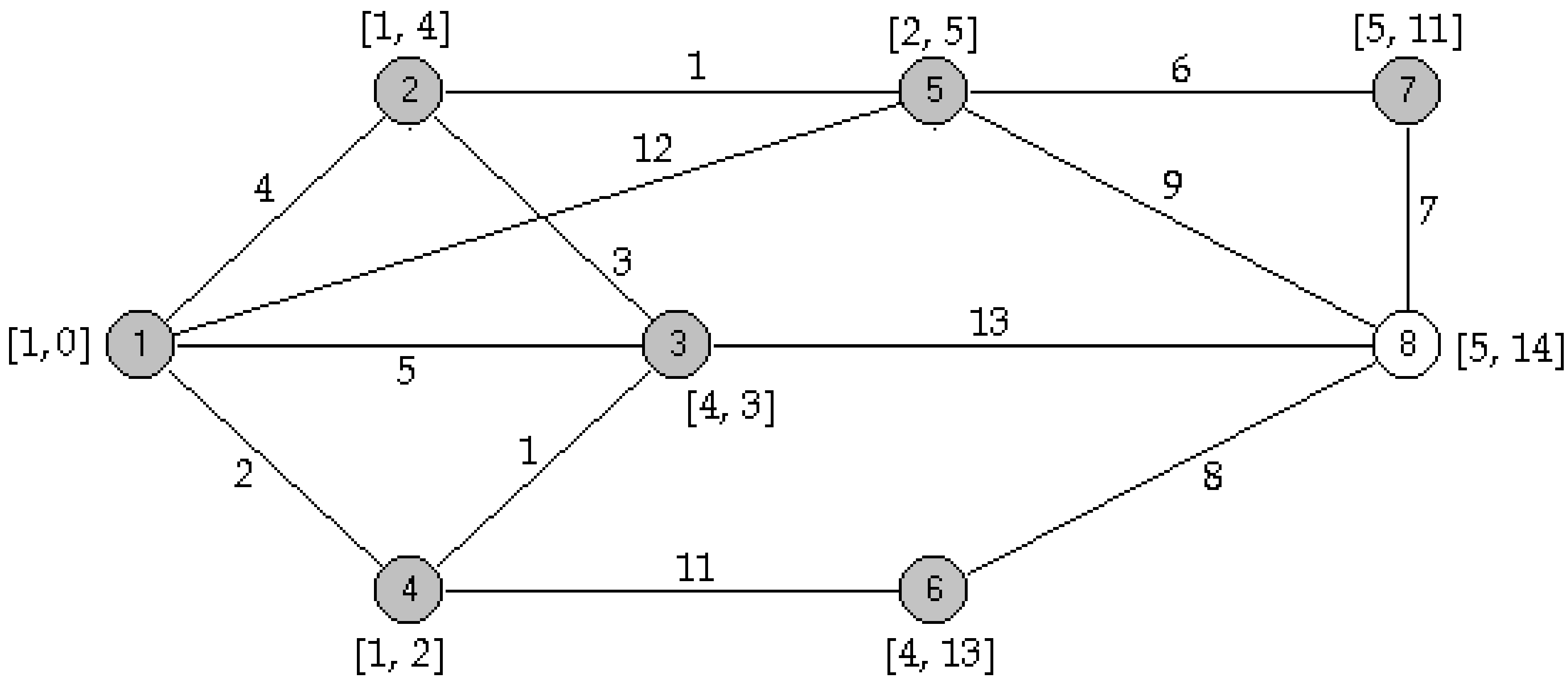
3. O problema do Caminho Mais Curto

3.2. Algoritmo de Dijkstra (exemplo)

Passo 3.

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

$\pi_8 = \min\{\pi_8, \pi_6 + C_{68}\} = \min\{14, 13 + 8\} = 14 \Rightarrow [\xi_8, \pi_8] = [5, 14]$ (sem alteração)



3. O problema do Caminho Mais Curto

3.2. Algoritmo de Dijkstra (exemplo)

Passo 2.

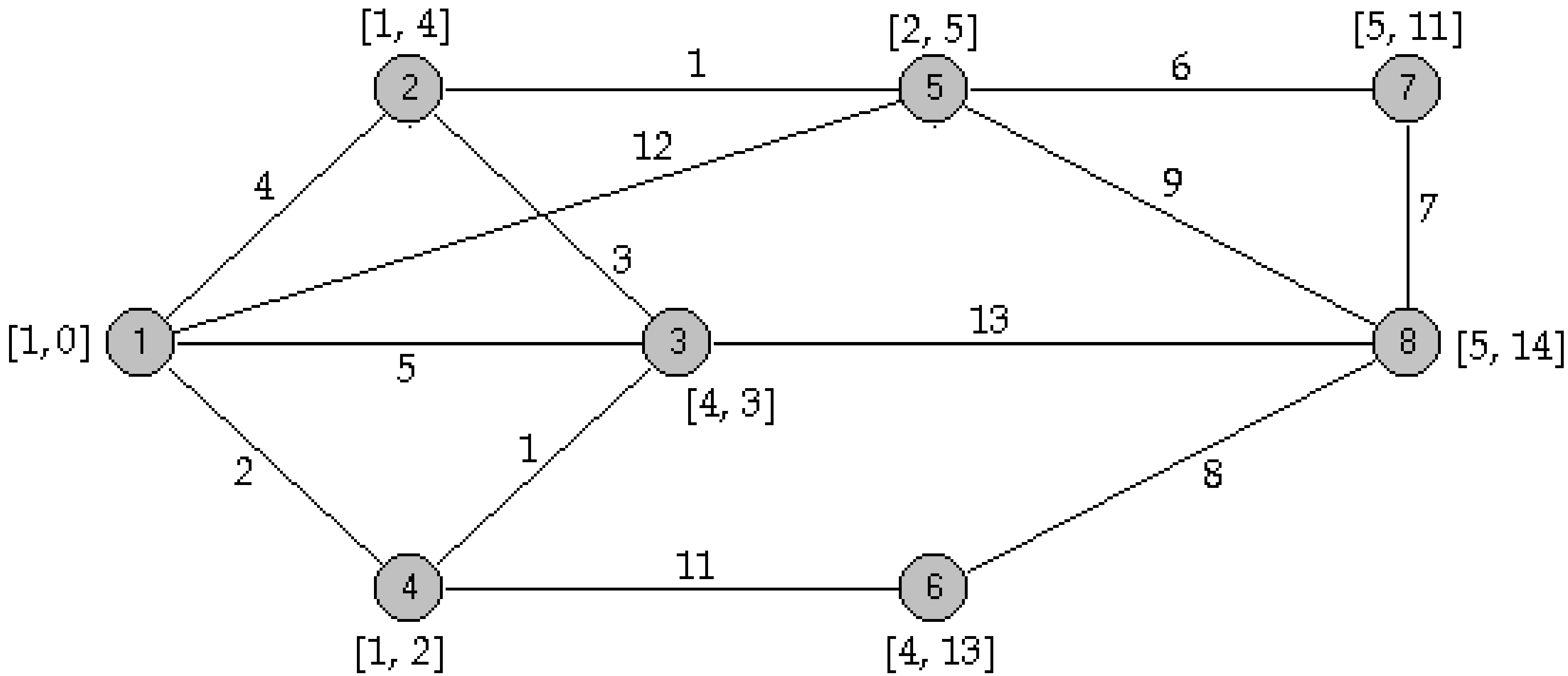
- $k = 8$, pois $\pi_8 = \min \{ \pi_8 \} = \min \{ 14 \}$
- Permanentes = Permanentes $\cup$ { 8 } = { 1, 2, 3, 4, 5, 6, 7, 8 }
- Temporários = Temporários - { 8 } = $\varnothing$

3. O problema do Caminho Mais Curto

3.2. Algoritmo de Dijkstra (exemplo)

Passo 3.

- Não existem nós adjacentes a 8, com rótulos temporários.



Passo 2.

- Temporários = ∅ ⇒ Fim do algoritmo.

3. O problema do Caminho Mais Curto

3.2. Algoritmo de Dijkstra (exemplo)

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

    $p = \{ 8 \}$

    $i = 8$
  - como $i \neq 1$ então $\{ i = \xi_i = \xi_8 = 5;\ p = p \cup \{ i \} = \{ 5, 8 \}$
  - como $i \neq 1$ então $\{ i = \xi_i = \xi_5 = 2;\ p = p \cup \{ i \} = \{ 2, 5, 8 \}$
  - como $i \neq 1$ então $\{ i = \xi_i = \xi_2 = 1;\ p = p \cup \{ i \} = \{ 1, 2, 5, 8 \}$
  - como $i = 1$ então termina o processo;
  - logo,
    - o caminho mais curto entre 1 e 8 é $p = \{ 1, 2, 5, 8 \}$ e
    - o comprimento é $C(p) = 14$ ($= \pi_8$)

3. O problema do Caminho Mais Curto

3.3. Algoritmo de Floyd

- Para determinar o caminho mais curto **entre todos os pares** de nós,
    - aplicar o algoritmo de Dijkstra **n** vezes, utilizando cada nó, sucessivamente, como origem
- No entanto, existem outros algoritmos para resolver este problema,
    - é o caso do algoritmo de Floyd, desde que não haja circuitos negativos -- os arcos podem ter comprimentos negativos
- Considere-se uma rede G = (N, A, D),
    - um arco (i, j) designa-se por **arco básico** se constituir 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 ao caminho mais curto)
- O algoritmo de Floyd
    - utiliza a matriz D, de ordem **n**, das distâncias directas mais curtas entre os nós
    - os nós não adjacentes (sem aresta entre eles) têm associado uma distância directa infinita
    - como os nós são (por convenção) adjacentes com eles próprios, têm associado um arco de comprimento 0

3. O problema do Caminho Mais Curto

3.3. 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 efectuada para todos os nós i, j $\neq$ k

- Após efectuar a "tripla ligação" para cada k $\in$ N com i, j $\in$ N - { k },
  - a rede (na 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, mantém-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**

3. O problema do Caminho Mais Curto

3.3. Algoritmo de Floyd

Passo 1.

- D = matriz das distâncias directas
- $P_{ij} = j, \ \forall \ i, j \in N$
- $k \leftarrow 0$

Passo 2.

- **Se** $k \geq n$ **Então** STOP
- $k \leftarrow k + 1$

Passo 3.

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

  Então

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

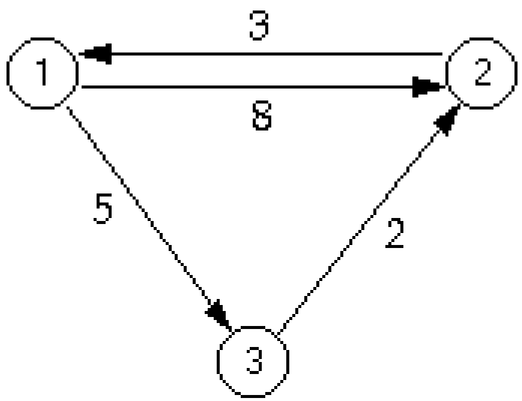
  $P_{ij} \leftarrow P_{ik}$

- No final:
  - os elementos da matriz **D** são as distâncias mais curtas entre qualquer par de nós

3. O problema do Caminho Mais Curto

3.3. Algoritmo de Floyd (exemplo)

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



Passo 1.

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

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

k = 0

Passo 1.
D = matriz das distâncias directas
$P_{ij} = j, \forall\ i, j \in N$
$k \leftarrow 0$

3. O problema do Caminho Mais Curto

3.3. Algoritmo de Floyd (exemplo)

Passo 2.
Se $k \geq n$ **Então** STOP
$k \leftarrow k + 1$
Passo 3.
Se $D_{ij} > D_{ik} + D_{kj}$ $(i, j \neq k)$ **Então**
$D_{ij} \leftarrow D_{ik} + D_{kj}$
$P_{ij} \leftarrow P_{ik}$

Passo 2.

- $0 \geq 3$ $(n = 3)$ Falso
- $k = k + 1 = 1$

Passo 3.

- $(i = 2; k = 1; j = 2)$: $D_{22} < D_{21} + D_{12}$ $(0 < 3 + 8 = 11)$
- $(i = 2; k = 1; j = 3)$: $D_{23} > D_{21} + D_{13}$ $(\infty > 3 + 5 = 8)$ $\Rightarrow$ $D_{23} = D_{21} + D_{13} = 8$; $P_{23} = P_{21} = 1$
- $(i = 3; k = 1; j = 2)$: $D_{32} < D_{31} + D_{12}$ $(2 < \infty + 8)$
- $(i = 3; k = 1; j = 3)$: $D_{33} < D_{31} + D_{13}$ $(0 < \infty + 5)$

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

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

3. O problema do Caminho Mais Curto

3.3. Algoritmo de Floyd (exemplo)

Passo 2.
Se $k \geq n$ **Então** STOP
$k \leftarrow k + 1$
Passo 3.
Se $D_{ij} > D_{ik} + D_{kj}$ $(i, j \neq k)$ **Então**
$D_{ij} \leftarrow D_{ik} + D_{kj}$
$P_{ij} \leftarrow P_{ik}$

Passo 2.

- $1 \geq 3$ Falso
- $k = k + 1 = 2$

Passo 3.

- $(i = 1; k = 2; j = 1)$: $D_{11} < D_{12} + D_{21}$ $(0 < 8 + 3 = 11)$
- $(i = 1; k = 2; j = 3)$: $D_{13} < D_{12} + D_{23}$ $(5 < 8 + 8 = 16)$
- $(i = 3; k = 2; j = 1)$: $D_{31} > D_{32} + D_{21}$ $(\infty > 2 + 3 = 5)$ $\Rightarrow$ $D_{31} = D_{32} + D_{21} = 5$; $P_{31} = P_{32} = 2$
- $(i = 3; k = 2; j = 3)$: $D_{33} < D_{32} + D_{23}$ $(0 < 2 + 8)$

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

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

3. O problema do Caminho Mais Curto

3.3. Algoritmo de Floyd (exemplo)

Passo 2.
Se $k \geq n$ **Então** STOP
$k \leftarrow k + 1$
Passo 3.
Se $D_{ij} > D_{ik} + D_{kj}$ $(i, j \neq k)$ **Então**
$D_{ij} \leftarrow D_{ik} + D_{kj}$
$P_{ij} \leftarrow P_{ik}$

Passo 2.

- $2 \geq 3$ Falso
- $k = k + 1 = 3$

Passo 3.

- $(i = 1; k = 3; j = 1)$: $D_{11} < D_{13} + D_{31}$ $(0 < 5 + 5 = 10)$
- $(i = 1; k = 3; j = 2)$: $D_{12} > D_{13} + D_{32}$ $(8 > 5 + 2 = 7)$ $\Rightarrow$ $D_{12} = D_{13} + D_{32} = 7$; $P_{12} = P_{13} = 3$
- $(i = 2; k = 3; j = 1)$: $D_{21} < D_{23} + D_{31}$ $(3 < 8 + 5 = 13)$
- $(i = 2; k = 3; j = 2)$: $D_{22} < D_{23} + D_{32}$ $(0 < 8 + 2)$

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

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

3. O problema do Caminho Mais Curto

3.3. Algoritmo de Floyd (exemplo)

Passo 2.
  Se $k \geq n$ **Então** STOP
  $k \leftarrow k + 1$
Passo 3.
  Se $D_{ij} > D_{ik} + D_{kj}$ $(i, j \neq k)$ **Então**
    $D_{ij} \leftarrow D_{ik} + D_{kj}$
    $P_{ij} \leftarrow P_{ik}$

Passo 2.

- $3 \geq 3$ Verdadeiro
- STOP (termina o algoritmo)

- 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) }

4. O problema do Fluxo Máximo

4.1. Conceitos gerais

- Considere uma rede G, em que os valores associados aos arcos desta rede, $\mathbf{b_{ij}}$, representam as respetivas capacidades, isto é, a quantidade máxima de fluxo que pode ser enviada pelos arcos; estes valores terão que ser positivos ($b_{ij} \geq 0$)
- Portanto, pode-se definir a rede da seguinte forma: G = (V, 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 podem ser enviados do nó origem **S** para o nó terminal **T**
- O fluxo no arco (i, j) é designado por $\mathbf{x_{ij}}$

4. O problema do Fluxo Máximo

4.1. Conceitos gerais

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

  $0 \leq x_{ij} \leq b_{ij}$, para todo $(i, j) \in A$ **[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; ou seja,

  $\sum_{i} x_{ij} = \sum_{k} x_{jk}$, para todo $j \neq S, T$ **[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; isto é,

  $\sum_{i} x_{Si} = f = \sum_{j} x_{jT}$ **[3]**

  onde **f** é o valor de fluxo.

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

4. O problema do Fluxo Máximo

4.1. Conceitos gerais

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

Maximizar $f = \sum_{j} x_{Sj}$

sujeito a

$$\sum_{i} x_{ij} = \sum_{k} x_{jk} \text{ , para todo } j \neq S, T \qquad [2]$$

$$\sum_{i} x_{Si} = \sum_{j} x_{jT} \qquad [3]$$

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

- 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 [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 por ele

4. O problema do Fluxo Máximo

4.2. Algoritmo de Ford-Fulkerson

- É um modo sistemático de pesquisar todos os possíveis 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:
  - rotulado e varrido $\Rightarrow$ tem um rótulo e todos os seus vizinhos estão rotulados
  - rotulado e não varrido $\Rightarrow$ tem um rótulo, mas nem todos os seus vizinhos estão rotulados
  - não rotulado $\Rightarrow$ não tem rótulo
- O rótulo do nó **j** tem 2 partes: $[i^+, \varepsilon(j)]$ (ou $[i^-, \varepsilon(j)]$), em que,
  - i^+ (ou i^-): índice de um nó **i**, indicando que pode-se enviar fluxo de **i** para **j** (ou de j para i)
  - $\varepsilon(j)$: fluxo máximo adicional que se pode enviar de S para **j**

4. O problema do Fluxo Máximo

4.2. Algoritmo de Ford-Fulkerson

Passo 1.

- $S \leftarrow [S^+, \infty]$ (S é rotulado com S^+ e ∞)
- S fica rotulado e não varrido

Passo 2. (processo de rotulação)

- j (rotulado e não varrido) $\leftarrow [i^+, \varepsilon(j)]$ ou $[i^-, \varepsilon(j)]$
- **Para** todo o $k \in V$ tal que $(j, k) \in A$ e $x_{jk} < b_{jk}$ **Fazer**

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

- **Para** todo o $k \in V$ tal que $(k, j) \in A$ e $x_{kj} > 0$ **Fazer**

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

- **j** fica rotulado e varrido
- Todos os **k** ficam rotulados e não varridos
- **Se** (T está rotulado) ou (não é possível rotular T)

  Então (foi determinado um c.a.f.) ou (não existe c.a.f. $\Rightarrow$ o fluxo atual é máximo)

  Senão Regressar ao Passo 2 (início)

4. O problema do Fluxo Máximo

4.2. Algoritmo de Ford-Fulkerson

Passo 3. (mudança de fluxo)

- $T \leftarrow [k^+, \varepsilon(T)] \Rightarrow x_{kT} = x_{kT} + \varepsilon(T)$
- **Enquanto** $k \neq S$ **Fazer**

  Se $k \leftarrow [j^+, \varepsilon(k)] \Rightarrow x_{jk} = x_{jk} + \varepsilon(T)$

  Se $k \leftarrow [j^-, \varepsilon(k)] \Rightarrow x_{kj} = x_{kj} - \varepsilon(T)$

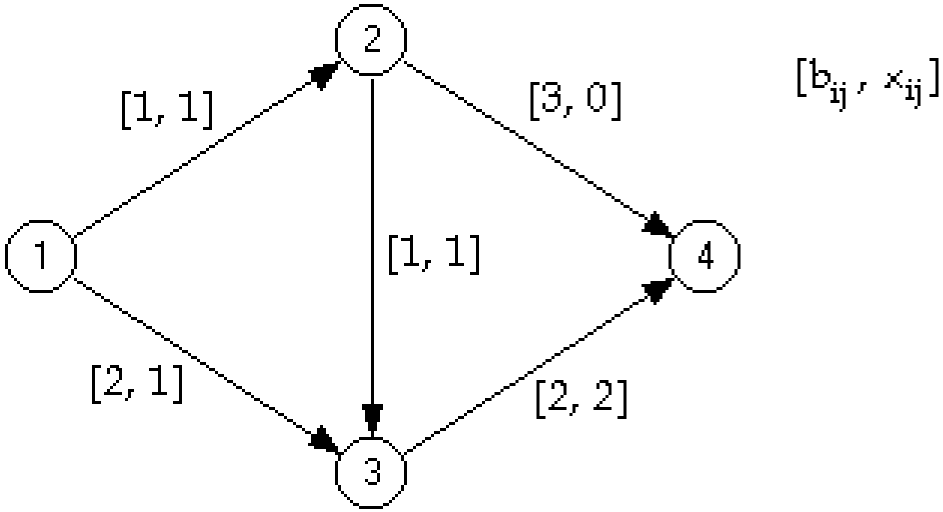
  $k \leftarrow j$
- Apagar os rótulos e regressar ao Passo 1

Passo 1.
$S \leftarrow [S^+, \infty]$ (S é rotulado com S^+ e ∞)
S fica rotulado e não varrido
Passo 2. (processo de rotulação)
j (rotulado e não varrido) $\leftarrow [i^+, \varepsilon(j)]$ ou $[i^-, \varepsilon(j)]$
Para todo o $k \in V$ tal que $(j, k) \in A$ e $x_{jk} < b_{jk}$ **Fazer**
$k \leftarrow [j^+, \varepsilon(k)]$ com $\varepsilon(k) = \min \{ \varepsilon(j), b_{jk} - x_{jk} \}$
Para todo o $k \in V$ tal que $(k, j) \in A$ e $x_{kj} > 0$ **Fazer**
$k \leftarrow [j^-, \varepsilon(k)]$ com $\varepsilon(k) = \min \{ \varepsilon(j), x_{kj} \}$
j fica rotulado e varrido
Todos os **k** ficam rotulados e não varridos
Se (T está rotulado) **ou** (não é possível rotular T)
Então
foi determinado um c.a.f. **ou**
não existe c.a.f. $\Rightarrow$ o fluxo atual é máximo
Senão
Regressar ao Passo 2 (início)

4. O problema do Fluxo Máximo

4.2. Algoritmo de Ford-Fulkerson (exemplo)

- Considere a rede seguinte 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 2 nós é de 2 unidades

4. O problema do Fluxo Máximo

4.2. Algoritmo de Ford-Fulkerson (exemplo)

Passo 1.

- Fluxo atual = 2
- $1 \leftarrow [1^+, \infty]$
- RotuladosVarridos = $\varnothing$
- RotuladosNãoVarridos = { 1 }
- NãoRotulados = { 2, 3, 4 }

Passo 2.

- $j \leftarrow 1$ (rotulado e não varrido)
- Vizinhos (não rotulados) do nó 1: 2 e 3
  - 2 não pode ser rotulado, pois $b_{12} = x_{12}$
  - $3 \leftarrow [1^+, \min \{ \varepsilon(1), b_{12} - x_{12} \}] \equiv [1^+, \min \{ \infty, 1 \}] \equiv [1^+, 1]$
- RotuladosVarridos = { 1 }
- RotuladosNãoVarridos = { 3 }
- NãoRotulados = { 2, 4 }

Passo 1.
$S \leftarrow [S^+, \infty]$ (S é rotulado com S^+ e ∞)
S fica rotulado e não varrido
Passo 2. (processo de rotulação)
j (rotulado e não varrido) $\leftarrow [i^+, \varepsilon(j)]$ ou $[i^-, \varepsilon(j)]$
Para todo o $k \in V$ tal que $(j, k) \in A$ e $x_{jk} < b_{jk}$ **Fazer**
$k \leftarrow [j^+, \varepsilon(k)]$ com $\varepsilon(k) = \min \{ \varepsilon(j), b_{jk} - x_{jk} \}$
Para todo o $k \in V$ tal que $(k, j) \in A$ e $x_{kj} > 0$ **Fazer**
$k \leftarrow [j^-, \varepsilon(k)]$ com $\varepsilon(k) = \min \{ \varepsilon(j), x_{kj} \}$
j fica rotulado e varrido
Todos os **k** ficam rotulados e não varridos
Se (T está rotulado) **ou** (não é possível rotular T)
Então
foi determinado um c.a.f. **ou**
não existe c.a.f. $\Rightarrow$ o fluxo atual é máximo
Senão
Regressar ao Passo 2 (início)

4. O problema do Fluxo Máximo

4.2. Algoritmo de Ford-Fulkerson (exemplo)

Passo 2.

- $j \leftarrow 3$ (rotulado e não varrido)
- Vizinhos (não rotulados) do nó 3: 2 e 4.
  - $2 \leftarrow [3^-, \min \{ \varepsilon(3), x_{23} \}] \equiv [3^-, \min \{ 1, 1 \}] \equiv [3^-, 1]$
  - 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) $\leftarrow [i^+, \varepsilon(j)]$ ou $[i^-, \varepsilon(j)]$

Para todo o $k \in V$ tal que $(j, k) \in A$ e $x_{jk} < b_{jk}$ **Fazer**

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

Para todo o $k \in V$ tal que $(k, j) \in A$ e $x_{kj} > 0$ **Fazer**

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

j fica rotulado e varrido

Todos os **k** ficam rotulados e não varridos

Se (T está rotulado) **ou** (não é possível rotular T)

Então

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

não existe c.a.f. $\Rightarrow$ o fluxo atual é máximo

Senão

Regressar ao Passo 2 (início)

4. O problema do Fluxo Máximo

4.2. Algoritmo de Ford-Fulkerson (exemplo)

Passo 2.

- $j \leftarrow 2$ (rotulado e não varrido)
- Vizinhos (não rotulados) do nó 2: 4
    - $4 \leftarrow [2^+, \min \{ \varepsilon(2), b_{24} - x_{24} \}] \equiv [2^+, \min \{ 1, 3 \}] \equiv [2^+, 1]$
- RotuladosVarridos = { 1, 3, 2 }
- RotuladosNãoVarridos = { 4 }
- NãoRotulados = $\varnothing$
- Como T = 4 foi rotulado,

  foi determinado um caminho de aumento de fluxo

Passo 2. (processo de rotulação)
j (rotulado e não varrido) $\leftarrow [i^+, \varepsilon(j)]$ ou $[i^-, \varepsilon(j)]$
Para todo o $k \in V$ tal que $(j, k) \in A$ e $x_{jk} < b_{jk}$ **Fazer**
$k \leftarrow [j^+, \varepsilon(k)]$ com $\varepsilon(k) = \min \{ \varepsilon(j), b_{jk} - x_{jk} \}$
Para todo o $k \in V$ tal que $(k, j) \in A$ e $x_{kj} > 0$ **Fazer**
$k \leftarrow [j^-, \varepsilon(k)]$ com $\varepsilon(k) = \min \{ \varepsilon(j), x_{kj} \}$
j fica rotulado e varrido
Todos os **k** ficam rotulados e não varridos
Se (T está rotulado) **ou** (não é possível rotular T)
Então
foi determinado um c.a.f. **ou**
não existe c.a.f. $\Rightarrow$ o fluxo atual é máximo
Senão
Regressar ao Passo 2 (início)

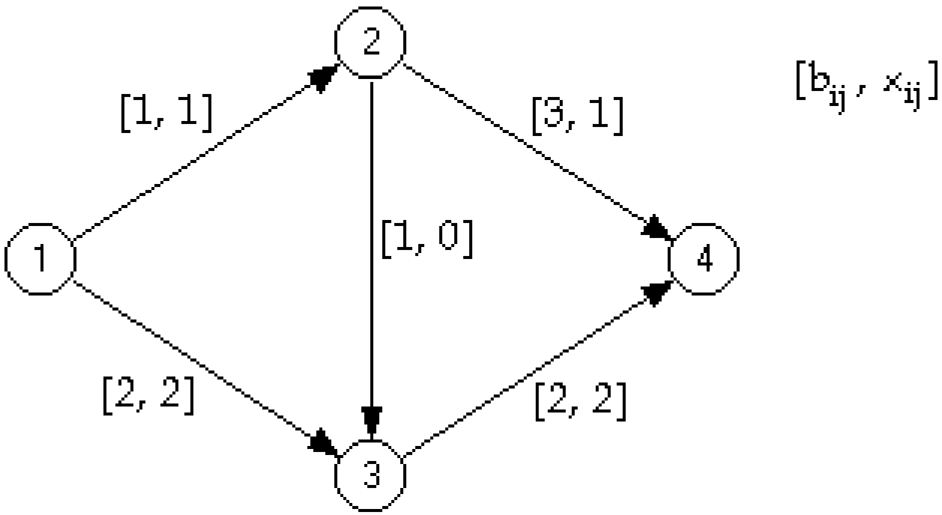
4. O problema do Fluxo Máximo

4.2. Algoritmo de Ford-Fulkerson (exemplo)

Passo 3. (mudança de fluxo: o aumento de fluxo é de $\varepsilon(4)$ = 1 unidade)

- fluxo = fluxo + $\varepsilon(4)$ = 2 + 1 = 3
- $4 \leftarrow [2^+, 1] \Rightarrow x_{24} = x_{24} + \varepsilon(4) = 0 + 1 = 1$
- $2 \leftarrow [3^-, 1] \Rightarrow x_{23} = x_{23} - \varepsilon(4) = 1 - 1 = 0$
- $3 \leftarrow [1^+, 1] \Rightarrow x_{13} = x_{13} + \varepsilon(4) = 1 + 1 = 2$

Passo 3. (mudança de fluxo)
$T \leftarrow [k^+, \varepsilon(T)] \Rightarrow x_{kT} = x_{kT} + \varepsilon(T)$
Enquanto $k \neq S$ **Fazer**
Se $k \leftarrow [j^+, \varepsilon(k)] \Rightarrow x_{jk} = x_{jk} + \varepsilon(T)$
Se $k \leftarrow [j^-, \varepsilon(k)] \Rightarrow x_{kj} = x_{kj} - \varepsilon(T)$
$k \leftarrow j$
Apagar os rótulos e regressar ao **Passo 1**



4. O problema do Fluxo Máximo

4.2. Algoritmo de Ford-Fulkerson (exemplo)

Passo 1.

- Fluxo actual = 3
- $1 \leftarrow [1^+, \infty]$
- RotuladosVarridos = $\varnothing$
- RotuladosNãoVarridos = { 1 }
- NãoRotulados = { 2, 3, 4 }

Passo 2.

- $j \leftarrow 1$ (rotulado e não varrido)
- Vizinhos (não rotulados) do nó 1: 2 e 3
  - 2 não pode ser rotulado, pois $b_{12} = x_{12}$
  - 3 não pode ser rotulado, pois $b_{13} = x_{13}$
- Como não é possível rotular mais nenhum nó, e como o nó T = 4 não foi rotulado, então não existe caminho de aumento de fluxo; logo, o fluxo atual é máximo

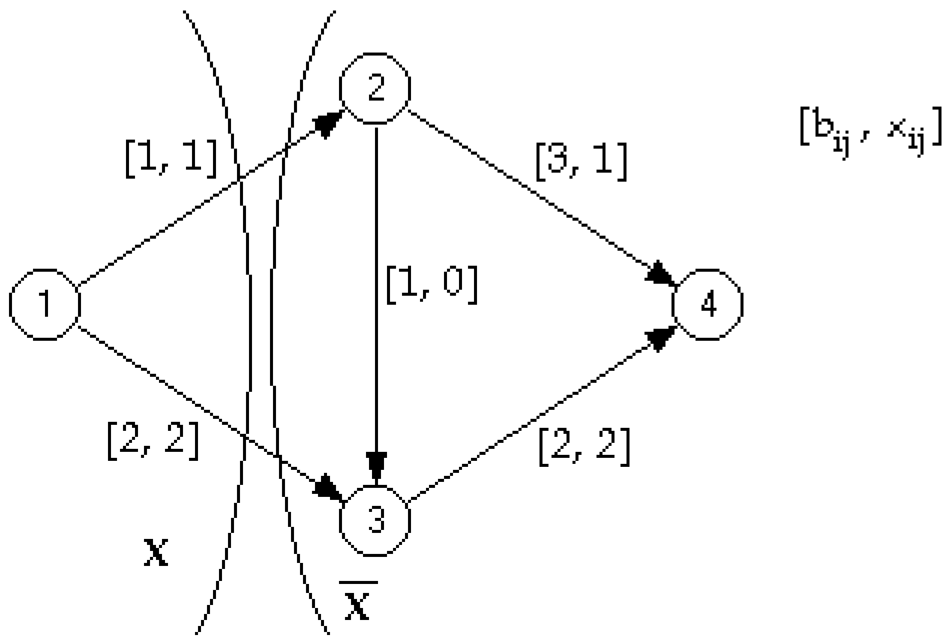
Passo 1.
  $S \leftarrow [S^+, \infty]$ (S é rotulado com S^+ e ∞)
  S fica rotulado e não varrido
Passo 2. (processo de rotulação)
  j (rotulado e não varrido) $\leftarrow [i^+, \varepsilon(j)]$ ou $[i^-, \varepsilon(j)]$
  Para todo o $k \in V$ tal que $(j, k) \in A$ e $x_{jk} < b_{jk}$ **Fazer**
    $k \leftarrow [j^+, \varepsilon(k)]$ com $\varepsilon(k) = \min\{ \varepsilon(j), b_{jk} - x_{jk} \}$
  Para todo o $k \in V$ tal que $(k, j) \in A$ e $x_{kj} > 0$ **Fazer**
    $k \leftarrow [j^-, \varepsilon(k)]$ com $\varepsilon(k) = \min\{ \varepsilon(j), x_{kj} \}$
  j fica rotulado e varrido
  Todos os **k** ficam rotulados e não varridos
  Se (T está rotulado) **ou** (não é possível rotular T)
  Então
    foi determinado um c.a.f. **ou**
    não existe c.a.f. $\Rightarrow$ o fluxo atual é máximo
  Senão
    Regressar ao Passo 2 (início)

4. O problema do Fluxo Máximo

4.2. Algoritmo de Ford-Fulkerson (exemplo)

Resultados a retirar após o final do algoritmo:

- Fluxo máximo = 3
- $X = \{ 1 \}$
- $\overline{X} = \{ 2, 3, 4 \}$
- $C(X, \overline{X}) = b_{12} + b_{13} = 3$ = fluxo máximo



5. O problema dos Casamentos Estáveis (Stable Marriage Problem)

5.1. Definição do problema

- Do ponto de vista de teoria dos Grafos, podemos ver cada um desses problemas como problemas de obter-se emparelhamentos com preferência em grafos bipartidos, o qual é denominada de "emparelhamento estável"

- Os problemas de emparelhamento estável
  - consistem em dividir um ou mais grupos de agentes em pares, onde cada agente possui uma lista de preferências ordenada, e
  - deseja-se encontrar um emparelhamento entre eles que respeite um critério de estabilidade que é baseado nas suas preferências

5. O problema dos Casamentos Estáveis (Stable Marriage Problem)

5.1. Definição do problema

- A versão clássica do problema traduz-se da forma seguinte:

  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

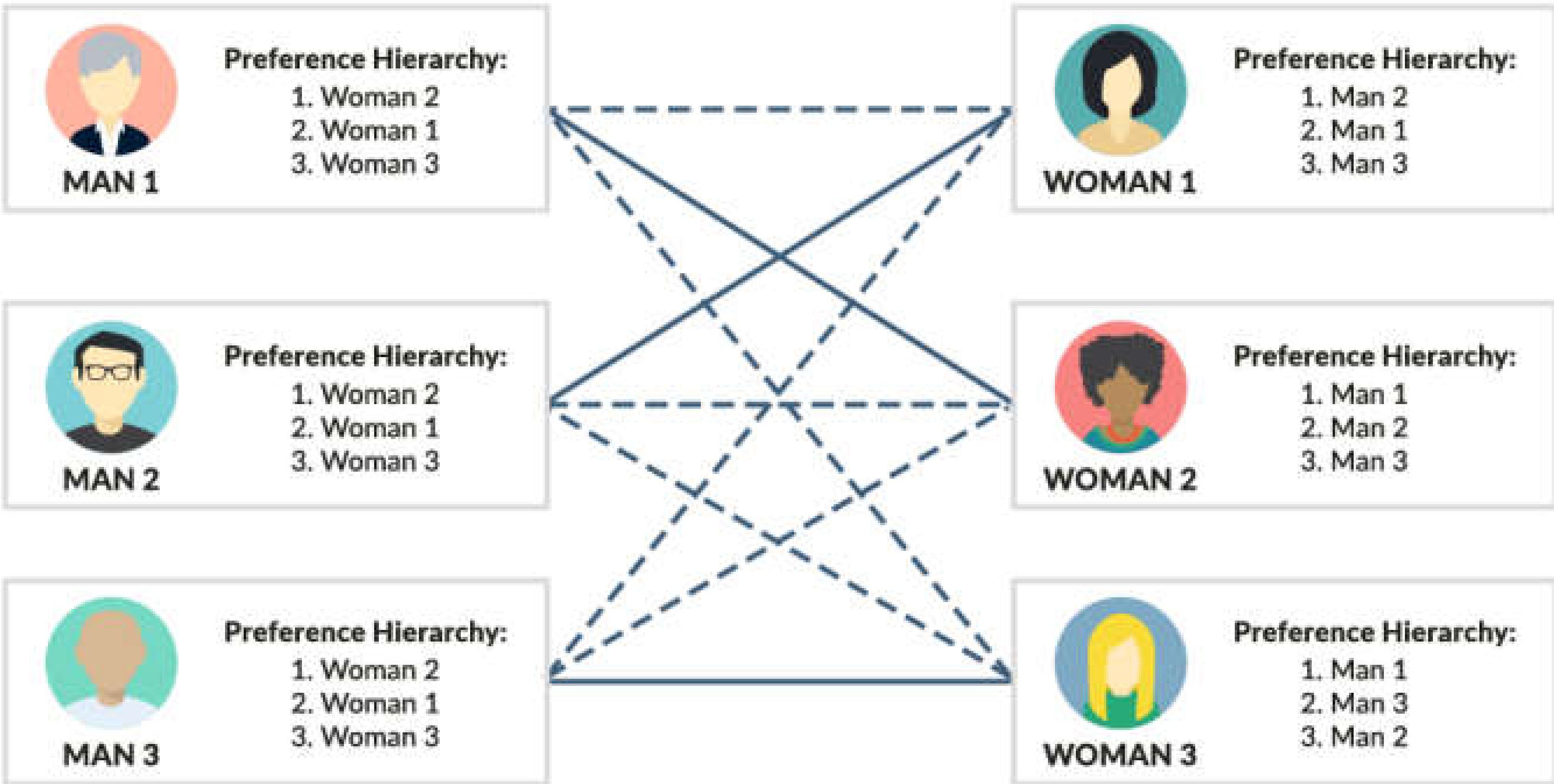
- Sendo $H = \{ h_1, \ldots, h_N \}$ e $M = \{ m_1, \ldots, m_N \}$ os conjuntos de homens e mulheres, um emparelhamento E é uma qualquer função injectiva de H em M. Informalmente, um emparelhamento é, neste caso, um conjunto de N casais (monogâmicos e heterosexuais)

- 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**

5. O problema dos Casamentos Estáveis (Stable Marriage Problem)

5.1. Definição do problema

- Este problema pode ser modelado através do seguinte grafo:



5. O problema dos Casamentos Estáveis (Stable Marriage Problem)

5.1. Definição do problema

- **Exemplo 1:** Para a instância seguinte, em que n = 4 e as listas de preferências se consideram ordenadas por ordem (estritamente) decrescente da esquerda para a direita,

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

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

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

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

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** (que se obtem pelo **Algoritmo de Gale-Shapley**)

- Gale e Shapley 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 daquele algoritmo

5. O problema dos Casamentos Estáveis (Stable Marriage Problem)

5.2. Algoritmo de Gale-Shapley (1962)

Algoritmo:

- **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** estiver livre então

    Emparelhar **h** e **m** (ficam noivos)

  Senão

    Se **m** preferir **h** ao seu actual noivo **h'** então

      Emparelhar **h** e **m** (ficam noivos), voltando **h'** a estar livre

    Senão

      m rejeita **h** e assim **h** continua livre.

- **Fim_Enquanto**

5. O problema dos Casamentos Estáveis (Stable Marriage Problem)

5.2. Algoritmo de Gale-Shapley (1962)

- 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 e
  - cada mulher fica com o pior parceiro
- Obviamente, a situação inverte-se se passarem a ser as mulheres que se propõem
- A versão que se segue do algoritmo, ainda da autoria dos mesmos autores, permitiu reconhecer esta e outras propriedades estruturais das soluções do problema

5. O problema dos Casamentos Estáveis (Stable Marriage Problem)

5.3. Extensão do Algoritmo de Gale-Shapley (1962)

Algoritmo:

- **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

    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_Enquanto**

5. O problema dos Casamentos Estáveis (Stable Marriage Problem)

5.4. Exemplo (versão base)

- 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:

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

5. O problema dos Casamentos Estáveis (Stable Marriage Problem)

5.4. Exemplo (versão base)

V	W	X	Y	Z	A	B	C	D	E
A	A	D	A	C	V		Z	X	
					W				

Iteração 1:

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**, então emparelhar W a A e V fica livre

h = X (V 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**, logo Y continua livre

h = Z (Z está livre)
m = C; C está livre, então emparelhar Z a C

5. O problema dos Casamentos Estáveis (Stable Marriage Problem)

5.4. Exemplo (versão base)

V	W	X	Y	Z	A	B	C	D	E
A	A	D	A	C	V		Z	X	
C			D		W				

Iteração 2:

h = V (V está livre)
m = C; C não está livre, então C prefere V a Z? **Não**, logo 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**, então emparelhar Y a D e X fica livre

h = Z (Z não está livre)

5. O problema dos Casamentos Estáveis (Stable Marriage Problem)

5.4. Exemplo (versão base)

V	W	X	Y	Z	A	B	C	D	E
A	A	D	A	C	V		Z	X	
C		C	D		W			Y	
D								V	

Iteração 3 (1):

h = V (V está livre)

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

5. O problema dos Casamentos Estáveis (Stable Marriage Problem)

5.4. Exemplo (versão base)

V	W	X	Y	Z	A	B	C	D	E
A	A	D	A	C	V	Y	Z	X	
C		C	D		W			Y	
D			B					V	

Iteração 3 (2):

h = V (V está livre)
m = D; D não está livre, então D prefere V a Y? **Sim**, então 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)

5. O problema dos Casamentos Estáveis (Stable Marriage Problem)

5.4. Exemplo (versão base)

V	W	X	Y	Z	A	B	C	D	E
A	A	D	A	C	V	Y	Z	X	
C		C	D		W	X		Y	
D		B	B					V	

Iteração 4 (1):

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**, então emparelhar X a B e Y fica livre

5. O problema dos Casamentos Estáveis (Stable Marriage Problem)

5.4. Exemplo (versão base)

V	W	X	Y	Z	A	B	C	D	E
A	A	D	A	C	V	Y	Z	X	Y
C		C	D		W	X		Y	
D		B	B					V	
			E						

Iteração 4 (2):

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**, então 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)

5. O problema dos Casamentos Estáveis (Stable Marriage Problem)

5.4. Exemplo (versão base)

V	W	X	Y	Z	A	B	C	D	E
A	A	D	A	C	V	Y	Z	X	Y
C		C	D		W	X		Y	
D		B	B					V	
			E						

Iteração 5:

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)

5. O problema dos Casamentos Estáveis (Stable Marriage Problem)

5.4. Exemplo (versão base)

V	W	X	Y	Z	A	B	C	D	E
A	A	D	A	C	V	Y	Z	X	Y
C		C	D		W	X		Y	
D		B	B					V	
			E						

- Como todos os elementos do conjunto H estão emparelhados (não estão livres), então **terminar**

- 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)

5. O problema dos Casamentos Estáveis (Stable Marriage Problem)

5.5. 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 também 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: são os alunos que se propõem às universidades, resultando um emparelhamento ótimo para os alunos
  - nesta variante, designada por StableMarriageWithIncompleteLists:
    - as vagas correspondem às mulheres,
    - os alunos aos homens
  - um **emparelhamento** será um conjunto **E** de pares (**h**, m) com **h** ∈ **H** e **m** ∈ **M**, tal que
    - **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**

5. O problema dos Casamentos Estáveis (Stable Marriage Problem)

5.5. Variantes do Problema dos Casamentos Estáveis

- Alguns anos depois de 1962 descobriu-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 das soluções é reformulado do modo seguinte:

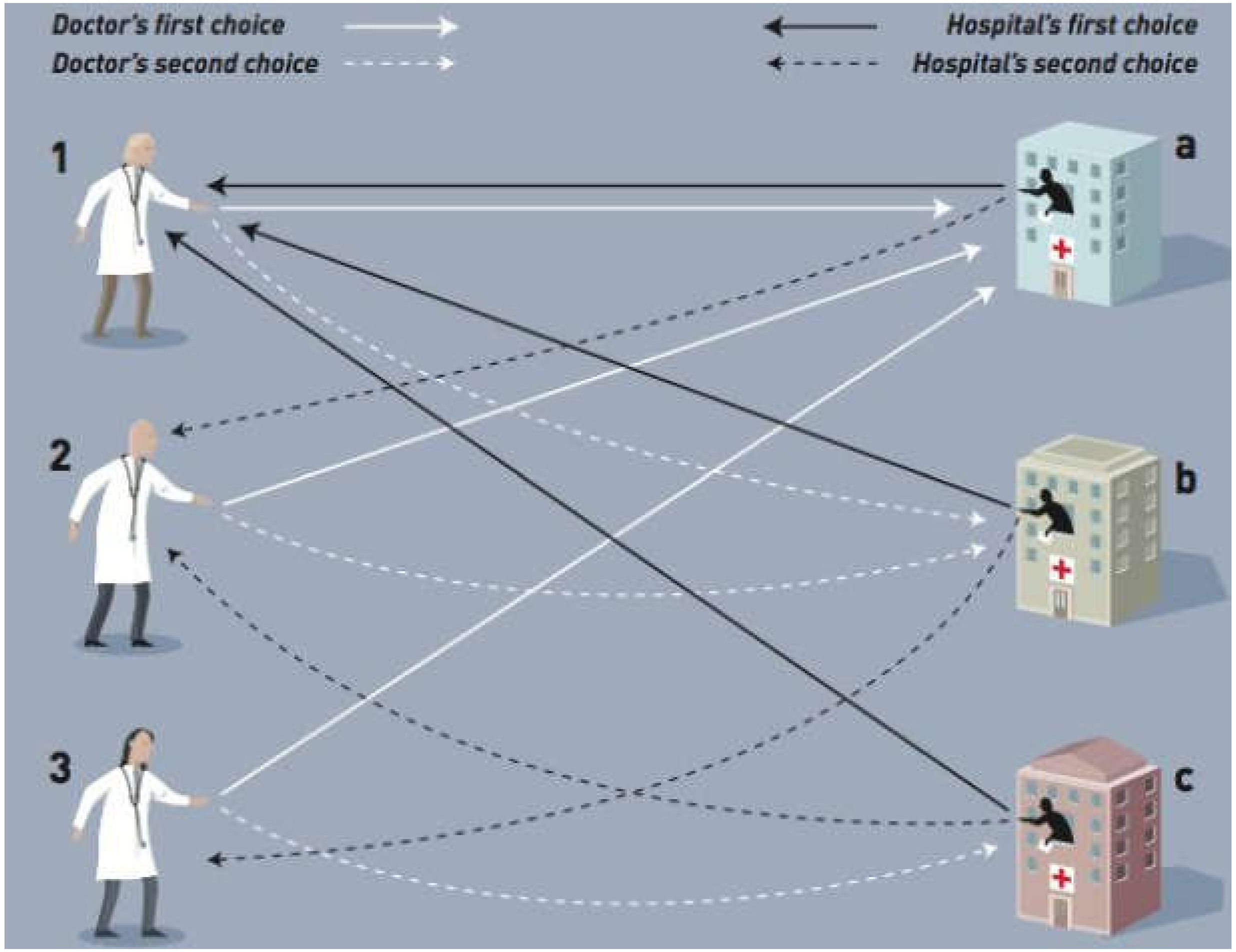
    um **emparelhamento** é **instável** <u>se e só se</u> existir um candidato **r** e um hospital **h** tais que
    - **h** é aceitável para **r** e **r** é aceitável para **h**,
    - o candidato **r** não ficou colocado ou prefere **h** ao seu atual hospital, e
    - **h** ficou com vagas por preencher ou **h** prefere **r** a pelo menos um dos candidatos com que ficou

    caso contrário, diz-se **estável**
  - Nesta situação, o conceito de emparelhamento é o mesmo, se se considerar que a atribuição é de candidatos a vagas

5. O problema dos Casamentos Estáveis (Stable Marriage Problem)

5.5. Variantes do Problema dos Casamentos Estáveis

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



5. O problema dos Casamentos Estáveis (Stable Marriage Problem)

5.5. 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)