

CPS740 - Lista 4

Pedro Maciel Xavier
116023847

12 de setembro de 2020

Questão 1.:

Seguindo as especificações do grafo obtemos:

$$V = \{(2, 3), (2, 5), (2, 7), (3, 2), (3, 5), (3, 7), \\ (5, 2), (5, 3), (5, 7), (7, 2), (7, 3), (7, 5)\}$$

$$E_1 = \{((2, 3), (3, 2)), ((2, 5), (5, 2)), ((2, 7), (7, 2)), \\ ((3, 5), (5, 3)), ((3, 7), (7, 3)), ((5, 7), (7, 5))\}$$

$$E_2 = \{((2, 3), (2, 5)), ((2, 3), (2, 7)), ((2, 5), (2, 7)), \\ ((3, 2), (3, 5)), ((3, 2), (3, 7)), ((3, 5), (3, 7)), \\ ((5, 2), (5, 3)), ((5, 2), (5, 7)), ((5, 3), (5, 7)), \\ ((7, 2), (7, 3)), ((7, 2), (7, 5)), ((7, 3), (7, 5))\}$$

a) Desenhemos $G(V, E_1 \cup E_2)$ com os respectivos pesos indicados em cada uma das arestas. As arestas pertencentes a E_1 foram coloridas de **azul**, enquanto as de E_2 estão vestidas de **vermelho**.

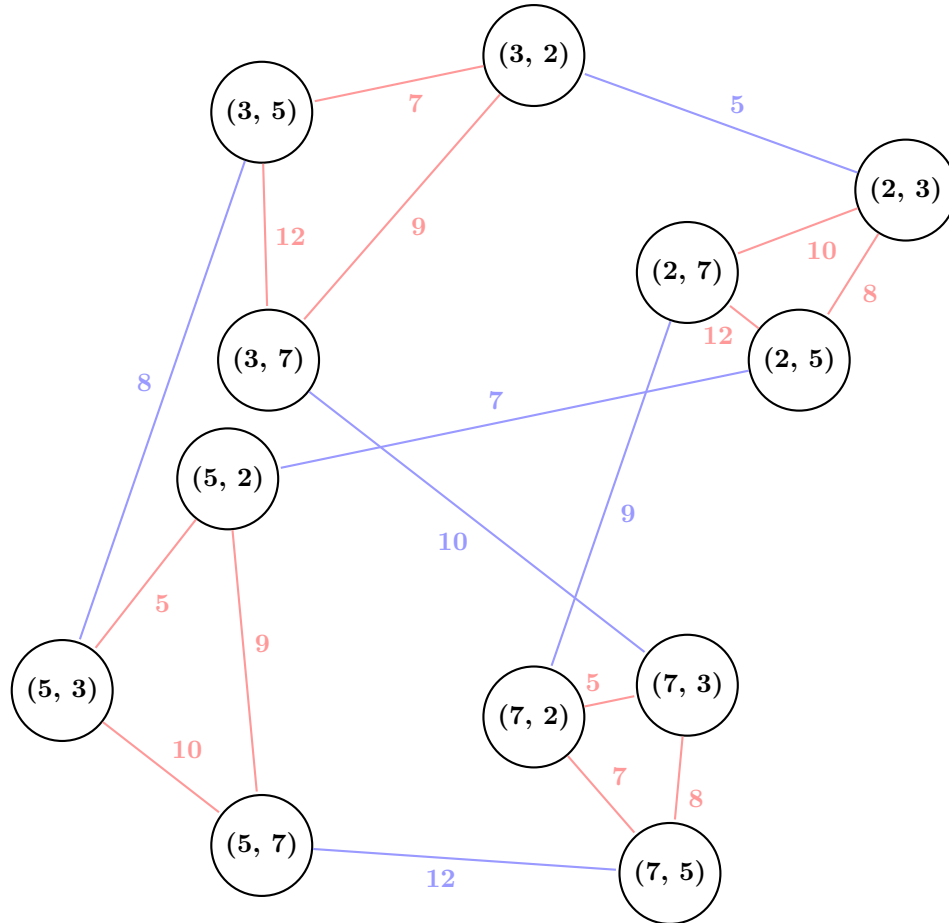
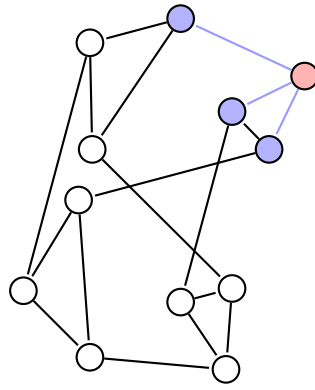
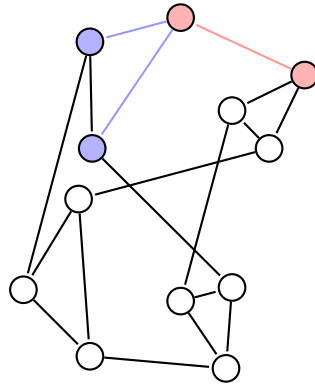


Figura 1: O Grafo $G(V, E_1 \cup E_2)$

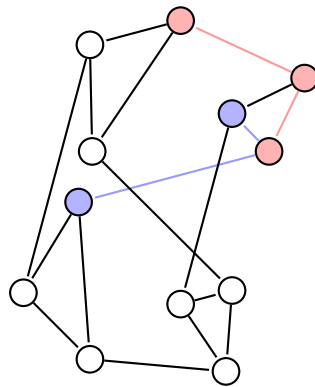
b) Começando no vértice $(2,3)$, vejamos a execução do algoritmo de *Dijkstra*, com a miniatura do grafo ao lado do vetor de distâncias.



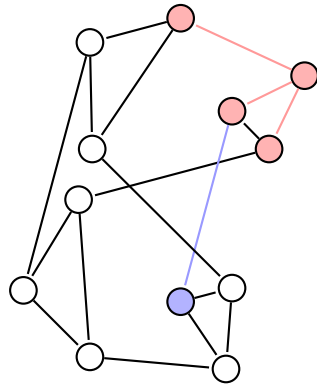
v	$d(u, v)$
$(2, 3)$	0
$(2, 5)$	8
$(2, 7)$	10
$(3, 2)$	5
$(3, 5)$	∞
$(3, 7)$	∞
$(5, 2)$	∞
$(5, 3)$	∞
$(5, 7)$	∞
$(7, 2)$	∞
$(7, 3)$	∞
$(7, 5)$	∞



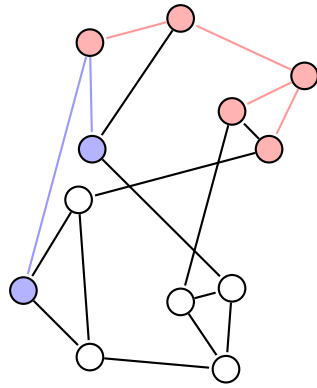
v	$d(u, v)$
$(2, 3)$	0
$(2, 5)$	8
$(2, 7)$	10
$(3, 2)$	5
$(3, 5)$	12
$(3, 7)$	14
$(5, 2)$	∞
$(5, 3)$	∞
$(5, 7)$	∞
$(7, 2)$	∞
$(7, 3)$	∞
$(7, 5)$	∞



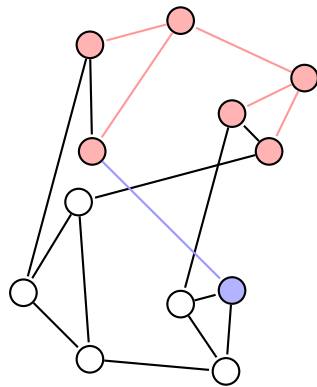
v	$d(u, v)$
$(2, 3)$	0
$(2, 5)$	8
$(2, 7)$	10
$(3, 2)$	5
$(3, 5)$	12
$(3, 7)$	14
$(5, 2)$	15
$(5, 3)$	∞
$(5, 7)$	∞
$(7, 2)$	∞
$(7, 3)$	∞
$(7, 5)$	∞



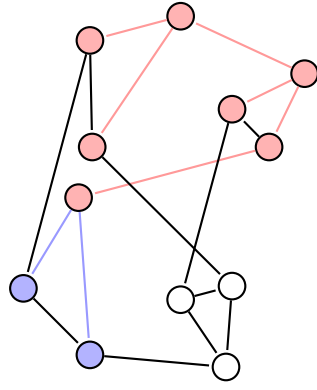
v	$d(u, v)$
(2, 3)	0
(2, 5)	8
(2, 7)	10
(3, 2)	5
(3, 5)	12
(3, 7)	14
(5, 2)	15
(5, 3)	∞
(5, 7)	∞
(7, 2)	19
(7, 3)	∞
(7, 5)	∞



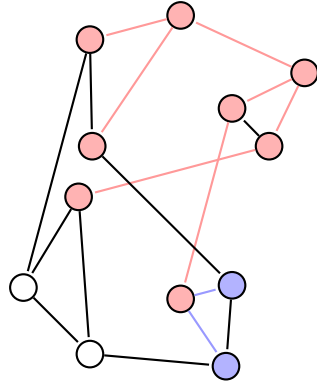
v	$d(u, v)$
(2, 3)	0
(2, 5)	8
(2, 7)	10
(3, 2)	5
(3, 5)	12
(3, 7)	14
(5, 2)	15
(5, 3)	20
(5, 7)	∞
(7, 2)	19
(7, 3)	∞
(7, 5)	∞



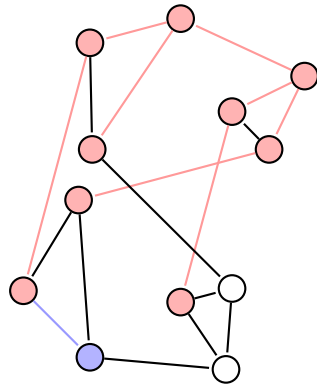
v	$d(u, v)$
(2, 3)	0
(2, 5)	8
(2, 7)	10
(3, 2)	5
(3, 5)	12
(3, 7)	14
(5, 2)	15
(5, 3)	20
(5, 7)	∞
(7, 2)	19
(7, 3)	24
(7, 5)	∞



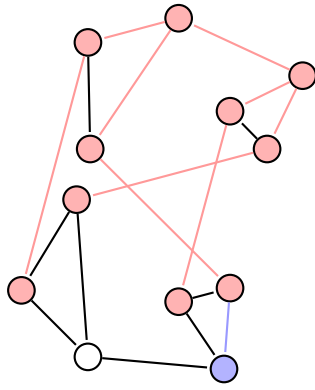
v	$d(u, v)$
(2, 3)	0
(2, 5)	8
(2, 7)	10
(3, 2)	5
(3, 5)	12
(3, 7)	14
(5, 2)	15
(5, 3)	20
(5, 7)	24
(7, 2)	19
(7, 3)	24
(7, 5)	∞



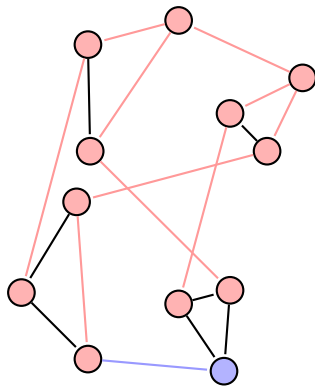
v	$d(u, v)$
(2, 3)	0
(2, 5)	8
(2, 7)	10
(3, 2)	5
(3, 5)	12
(3, 7)	14
(5, 2)	15
(5, 3)	20
(5, 7)	24
(7, 2)	19
(7, 3)	24
(7, 5)	26



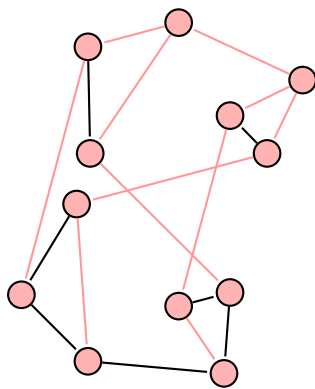
v	$d(u, v)$
(2, 3)	0
(2, 5)	8
(2, 7)	10
(3, 2)	5
(3, 5)	12
(3, 7)	14
(5, 2)	15
(5, 3)	20
(5, 7)	24
(7, 2)	19
(7, 3)	24
(7, 5)	26



v	$d(u, v)$
$(2, 3)$	0
$(2, 5)$	8
$(2, 7)$	10
$(3, 2)$	5
$(3, 5)$	12
$(3, 7)$	14
$(5, 2)$	15
$(5, 3)$	20
$(5, 7)$	24
$(7, 2)$	19
$(7, 3)$	24
$(7, 5)$	26



v	$d(u, v)$
$(2, 3)$	0
$(2, 5)$	8
$(2, 7)$	10
$(3, 2)$	5
$(3, 5)$	12
$(3, 7)$	14
$(5, 2)$	15
$(5, 3)$	20
$(5, 7)$	24
$(7, 2)$	19
$(7, 3)$	24
$(7, 5)$	26



v	$d(u, v)$
$(2, 3)$	0
$(2, 5)$	8
$(2, 7)$	10
$(3, 2)$	5
$(3, 5)$	12
$(3, 7)$	14
$(5, 2)$	15
$(5, 3)$	20
$(5, 7)$	24
$(7, 2)$	19
$(7, 3)$	24
$(7, 5)$	26

Questão 2.:

Algoritmo 1.: Floyd-Warshall++

```
1  def floyd_warshall(G(V, E), P[n][n]):
2      // `G(V, E)` um grafo
3      // `P[n][n]` a matriz de pesos das arestas de G
4      seja n ← |V|
5
6      seja dist[n][n] ← ∞
7      seja kmin[n][n] ← nulo
8
9      para cada [u, v] em E:
10         dist[u][v] ← P[u][v]
11         kmin[u][v] ← v
12
13     para cada v em V:
14         dist[v][v] ← 0
15         kmin[v][v] ← v
16
17     para k de 1 até n:
18         para i de 1 até n:
19             para j de 1 até n:
20                 se dist[i][j] > dist[i][k] + dist[k][j]:
21                     // Preferimos ir de `i` para `k`
22                     // do que de `i` para `j`
23                     dist[i][j] ← dist[i][k] + dist[k][j]
24                     kmin[i][j] ← kmin[i][k]
25                 se dist[j][j] < 0:
26                     // Achamos um ciclo negativo
27                     retorna nulo
28
29     seja caminhos[n][n]
30
31     para cada [u, v] em V × V:
32         se kmin[u][v] = nulo:
33             caminhos[u][v] ← nulo
34         senão:
35             caminhos[u][v] ← lista([u])
36             enquanto u != v:
37                 u ← kmin[u][v]
38                 caminhos[u][v].inserir(u)
39
40     retorna caminhos
```

Questão 3.:

1 .: O algoritmo de *Dijkstra* sempre obtém o caminho mais curto caso o grafo possua arestas com pesos negativos. **Falso**

Seja $G(V, E)$ com $V = \{x, y, z\}$ e $E = \{(x, y), (x, z), (y, z)\}$. Suponha que $P(x, y) = a$, $P(x, z) = b$ e $P(y, z) = -b$ com $0 < a < b$.

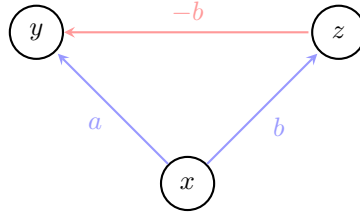


Figura 2: Grafo genérico onde o algoritmo de *Dijkstra* falha.

O Algoritmo de *Dijkstra* consideraria (x, y) o menor caminho entre x e y , negligenciando a existência de (x, z, y) , cujo custo é menor.

2 .: Sejam M e M' dois emparelhamentos perfeitos de G . Seja H o subgrafo induzido pela diferença simétrica de M e M' . Então, toda componente conexa de H é um ciclo ou caminho. **Verdadeiro**

Demonstração. Vamos desconsiderar o caso trivial onde $M = M'$, pois bastaria tomar $G = K_2$ para obter um contra-exemplo ($M \triangle M' = \emptyset$). Dito isto, vamos supor, por absurdo, que uma das componentes conexas de H não é um ciclo ou caminho, isto é, possui ao menos um vértice $w \in H$, $\text{grau}(w) \geq 3$.

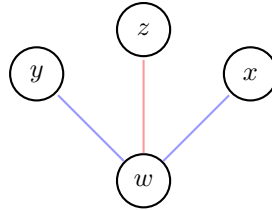


Figura 3: Componente conexa hipotética em H

Vamos supor que w esteja conectado aos vértices $x, y, z \in H$ e que $\{(w, x), (w, y), (w, z)\} \subseteq M \triangle M'$. Como $A \triangle B = (A - B) \cup (B - A)$, então $e \in A \triangle B \implies e \notin A \cap B$. Como uma aresta de H não pode estar em M e M' ao mesmo tempo, temos, pelo Princípio da Gaiola de Pombos (*Pigeonhole*), que um dos dois conjuntos deve possuir ao menos duas arestas. No entanto, as três arestas são incidentes em w . Isto não é possível, uma vez que M e M' são emparelhamentos. ■

Questão 4.:

Teorema. Seja $G(V, E)$ um grafo simples, $\rho(G)$ o tamanho da menor cobertura de arestas C e $\nu(G)$ o tamanho do maior emparelhamento M . Vale que $|V| = \rho(G) + \nu(G) = |C| + |M|$.

Demonstração. Seja M um emparelhamento máximo, isto é, $|M| = \nu(G)$. Cada aresta em M cobre exatamente dois vértices de G , pela definição de emparelhamento.

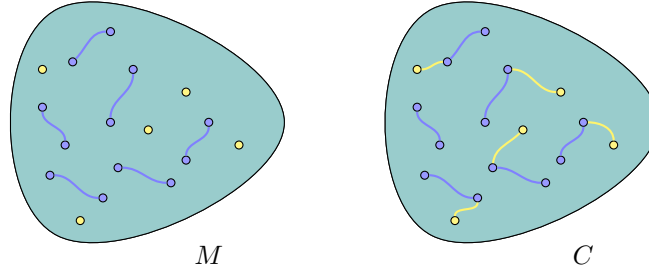


Figura 4: Formando uma cobertura a partir de M .

Adicionamos a M , para cada vértice w não emparelhado por M , uma aresta incidente sobre w . Isto é possível graças à conexidade de G . Assim, obtemos uma cobertura \hat{C} dos vértices de G , onde $|\hat{C}| = |M| + (|V| - 2|M|) = |V| - |M|$. Disso segue que

$$\begin{aligned} \rho(G) = |C| &\leq |\hat{C}| = |V| - |M| \\ \Rightarrow |V| &\geq |C| + |M| \end{aligned} \quad (1)$$

Por outro lado, se C é uma cobertura mínima dos vértices, ou seja, $|C| = \rho(G)$, então podemos, para cada componente conexa no subgrafo $H(V, C)$, escolher uma aresta de modo a compor um emparelhamento \hat{M} .

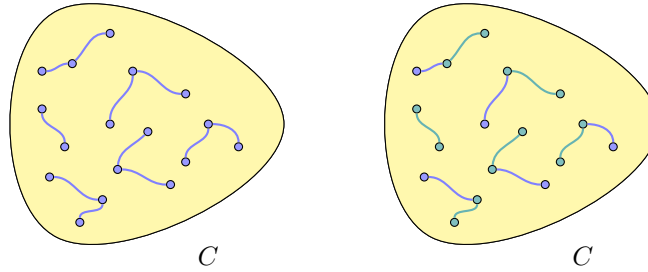


Figura 5: Compondo um emparelhamento a partir de C .

O número de componentes conexas em $H(V, C)$ é $|\hat{M}| = |V| - |C|$. Para melhor entender este fato, basta escrever

$$\begin{aligned} |V| &= \sum_{h \subseteq H} |v(h)| \\ |C| &= \sum_{h \subseteq H} |e(h)| \end{aligned}$$

$$\therefore |V| - |C| = \sum_{h \subseteq H} |v(h)| - |e(h)|$$

$$|V| - |C| = \sum_{h \subseteq H} 1 = |\hat{M}|$$

onde $h \subseteq H$ é uma componente conexa de H . Aqui usamos o fato de que C é cobertura mínima e, portanto, cada $h \subseteq H(V, C)$ é uma árvore. Assim, $|v(h)| - |e(h)| = 1$.

Por fim, sabemos que

$$\begin{aligned} \nu(G) = |M| &\geq |\hat{M}| = |V| - |C| \\ \implies |V| &\leq |C| + |M| \end{aligned} \tag{2}$$

Com (1) e (2) concluímos que $|V| = |C| + |M|$. ■

Conclusão

A própria demonstração nos revela que, tendo um emparelhamento máximo M , construímos uma cobertura \hat{C} de maneira gulosa onde $|\hat{C}| = |V| - |M|$. O teorema nos mostrou que $|C| = |V| - |M|$. Portanto, $|\hat{C}| = |C|$, ou seja, a cobertura construída desta maneira é mínima. O caminho inverso também segue raciocínio análogo, pautado no respectivo trecho da demonstração.

Questão 5.:

Suposição. Não é possível cobrir um tabuleiro de 100×100 utilizando retângulos 2×1 após a retirada de dois cantos opostos.

Demonstração. Imaginemos que o tabuleiro é colorido como se fosse para jogar xadrez ou damas:

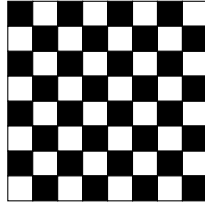


Figura 6: Tabuleiro de Xadrez

Ao posicionar um retângulo 2×1 sobre o tabuleiro, este ocupará sempre uma casa clara e uma casa escura. No entanto, ao removermos cantos opostos do tabuleiro, isto é, casas de uma mesma diagonal principal, estaremos retirando duas casas de mesma cor. Desta forma, tendo um tabuleiro de tamanho $n \times n$, $n \geq 2$ após a remoção dos cantos, podemos posicionar até $n^2/2 - 2$ retângulos. n é necessariamente um número par. Ao tentar incluir o que seria o último retângulo do tabuleiro ferido, nos deparamos com duas casas de mesma cor. Posicionar a peça violaria a necessidade de preencher casas de cores diferentes. ■

Referências

- [1] Jayme Luiz Szwarcfiter, **Teoria Computacional de Grafos**, 1ª edição, Rio de Janeiro, 2018.
- [2] Alexander Schrijver, **Advanced Graph Theory Course Notes**, CWI Amsterdam, 2020.