



UNIVERSIDADE ESTADUAL DE CAMPINAS  
FACULDADE DE ENGENHARIA ELÉTRICA E COMPUTAÇÃO

TRABALHO COMPUTACIONAL  
TEORIA DOS GRAFOS

---

## Algoritmos de Prim, Kruskal, Dijkstra e Ford-Moore-Bellman

---

*Autor:*

Jelther Oliveira Gonçalves  
(097254)

*Professor:*

Dr. Ricardo C. L. F. Oliveira

*Otimização Linear (IA881)*

*2º Semestre de 2016*

22 de Novembro de 2016

## Resumo

Neste presente trabalho foram aplicados os algoritmos de Prim, Kruskal, Dijkstra e Ford-Moore-Bellman em 2 grafos : Rede Óptica Italiana e Rede Rodoviária dos EUA.

A linguagem de programação utilizada foi o *Python 2.7* juntamente com o pacote *Networkx* como estrutura para os grafos e para efeito de comparação com os algoritmos já existentes neste pacote.

Para os algoritmos de Prim e Kruskal, foram gerados arquivos texto com os dados da árvore geradora mínima (suas arestas e nós) assim como o número de iterações, custo da árvore e o tempo computacional gasto.

Já para os algoritmos de Dijkstra e Ford-Moore-Bellman, foram apresentados os caminhos mínimos gerados para cada nó-fim solicitado, bem como o tamanho do caminho mínimo, número de iterações e relaxações aplicadas e o nó-anterior ao nó-fim.

# Conteúdo

<b>Conteúdo</b>	<b>2</b>
<b>1 A linguagem Python</b>	<b>4</b>
<b>2 O pacote Networkx</b>	<b>5</b>
<b>3 Set-up do computador utilizado</b>	<b>6</b>
<b>4 Algoritmo de Prim</b>	<b>7</b>
4.1 Resumo . . . . .	7
4.2 Rede Italiana . . . . .	7
4.3 Rede USA . . . . .	8
<b>5 Algoritmo de Kruskal</b>	<b>12</b>
5.1 Resumo . . . . .	12
5.2 Rede Italiana . . . . .	12
5.3 Rede USA . . . . .	13
<b>6 Algoritmo de Dijkstra</b>	<b>17</b>
6.1 Resumo . . . . .	17
6.2 Rede Italiana . . . . .	17
6.2.1 De 1 a 7 . . . . .	17
6.2.2 De 1 a 14 . . . . .	17
6.2.3 De 1 a 21 . . . . .	18
6.3 Rede USA . . . . .	18
6.3.1 De 1 a 10 . . . . .	18
6.3.2 De 1 a 20 . . . . .	19
6.3.3 De 1 a 30 . . . . .	19
6.3.4 De 1 a 40 . . . . .	20
6.3.5 De 1 a 50 . . . . .	20
6.3.6 De 1 a 60 . . . . .	21
6.3.7 De 1 a 70 . . . . .	21
<b>7 Algoritmo de Ford-Moore-Bellman</b>	<b>22</b>
7.1 Resumo . . . . .	22
7.2 Rede Italiana . . . . .	22
7.2.1 De 1 a 7 . . . . .	22
7.2.2 De 1 a 14 . . . . .	22
7.2.3 De 1 a 21 . . . . .	23

7.3	Rede USA . . . . .	23
7.3.1	De 1 a 10 . . . . .	23
7.3.2	De 1 a 20 . . . . .	24
7.3.3	De 1 a 30 . . . . .	24
7.3.4	De 1 a 40 . . . . .	25
7.3.5	De 1 a 50 . . . . .	25
7.3.6	De 1 a 60 . . . . .	26
7.3.7	De 1 a 70 . . . . .	26
	<b>Bibliografia</b>	<b>27</b>
	<b>Apêndices</b>	<b>28</b>
	<b>A Prim</b>	<b>28</b>
	<b>B Kruskal</b>	<b>31</b>
	<b>C Dijkstra</b>	<b>34</b>
	<b>D Ford-Moore-Bellman</b>	<b>38</b>

# Capítulo 1

## A linguagem Python

A linguagem Python, conforme visto em [1] e [2] é uma linguagem de alto nível, interpretada, de script, multiplataforma, orientada a objetos, funcional, de tipagem dinâmica e forte. Foi criada em 1991 e é amplamente utilizada mundialmente.

Apesar de facilitar o desenvolvimento, em relação as linguagens compiladas existe uma penalidade na performance dos scripts criados [3]. Entretanto, utilizando de pacotes já desenvolvidos sobre rotinas compiladas, a perda de performance é menor.

## Capítulo 2

# O pacote Networkx

O pacote Networkx [4] é utilizado para criação,manipulação e estudo de grafos e redes.

O pacote já possui vários métodos implementados [5] mas neste trabalho nos limitamos a utilizar somente a estrutura dos grafos e seus métodos para acessar os nós e arestas.

## Capítulo 3

# Set-up do computador utilizado

O computador em que estes algoritmos foram rodados possui a seguinte configuração:

- Processador AMD FX-4300, Black Edition, Cache 8Mb, 3.8GHz, AM3+ FD4300WMHKBOX
- 8 GB RAM Kingston 1333Mhz DDR3 CL9 - KVR13N9S8/4
- Sistema Operacional Microsoft Windows 10 (build 14393), 64-bit

## Capítulo 4

# Algoritmo de Prim

### 4.1 Resumo

Nesta implementação o conjunto franja é construído a cada iteração. Saliendo que este não é o melhor caminho em termos de performance, como pode ser visto com a comparação do tempo de execução com o algoritmo do pacote Networkx.

### 4.2 Rede Italiana

```
1 *****
2 Prim's Algorithm
3 *****
4 Graph Name:  rede_italiana
5 Start Node:  1
6
7 |Original Algorithm|
8 Nodes of MST is:  [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16,
9   ↪ 17, 18, 19, 20, 21]
10 Edges of MST is:
11 (From,To,Weight) = (1, 3, 110.0)
12 (From,To,Weight) = (2, 3, 110.0)
13 (From,To,Weight) = (2, 7, 90.0)
14 (From,To,Weight) = (3, 8, 95.0)
15 (From,To,Weight) = (3, 5, 90.0)
16 (From,To,Weight) = (4, 5, 85.0)
17 (From,To,Weight) = (6, 7, 90.0)
18 (From,To,Weight) = (8, 9, 55.0)
19 (From,To,Weight) = (9, 10, 60.0)
20 (From,To,Weight) = (9, 12, 110.0)
21 (From,To,Weight) = (11, 14, 130.0)
22 (From,To,Weight) = (12, 13, 120.0)
23 (From,To,Weight) = (12, 14, 170.0)
24 (From,To,Weight) = (13, 15, 180.0)
25 (From,To,Weight) = (15, 18, 90.0)
26 (From,To,Weight) = (16, 18, 100.0)
27 (From,To,Weight) = (17, 20, 420.0)
28 (From,To,Weight) = (18, 19, 200.0)
```



```

28 (From,To,Weight) = (19, 21, 210.0)
29 (From,To,Weight) = (20, 21, 150.0)
30 Total Cost is: 2665.0
31 Time Elapsed (in seconds): 0.00833308740875
32 Number of Iterations: 20
33
34 |Networkx Algorithm|
35 Nodes of MST is: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16,
    ↪ 17, 18, 19, 20, 21]
36 Edges of MST is:
37
38 (From,To,Weight) = (1, 3, 110.0)
39 (From,To,Weight) = (2, 3, 110.0)
40 (From,To,Weight) = (2, 7, 90.0)
41 (From,To,Weight) = (3, 8, 95.0)
42 (From,To,Weight) = (3, 5, 90.0)
43 (From,To,Weight) = (4, 5, 85.0)
44 (From,To,Weight) = (6, 7, 90.0)
45 (From,To,Weight) = (8, 9, 55.0)
46 (From,To,Weight) = (9, 10, 60.0)
47 (From,To,Weight) = (9, 12, 110.0)
48 (From,To,Weight) = (11, 14, 130.0)
49 (From,To,Weight) = (12, 13, 120.0)
50 (From,To,Weight) = (12, 14, 170.0)
51 (From,To,Weight) = (13, 15, 180.0)
52 (From,To,Weight) = (15, 18, 90.0)
53 (From,To,Weight) = (16, 18, 100.0)
54 (From,To,Weight) = (17, 20, 420.0)
55 (From,To,Weight) = (18, 19, 200.0)
56 (From,To,Weight) = (19, 21, 210.0)
57 (From,To,Weight) = (20, 21, 150.0)
58 Total Cost is: 2665.0
59 Time Elapsed (in seconds): 0.00051966099967

```

### 4.3 Rede USA

```

1 *****
2 Prim's Algorithm
3 *****
4 Graph Name: rede_usa
5 Start Node: 1
6
7 |Original Algorithm|
8 Nodes of MST is: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16,
    ↪ 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33,
    ↪ 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50,
    ↪ 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67,
    ↪ 68, 69, 70]
9 Edges of MST is:
10 (From,To,Weight) = (1, 2, 206.0)
11 (From,To,Weight) = (2, 3, 186.0)
12 (From,To,Weight) = (2, 4, 220.0)
13 (From,To,Weight) = (3, 5, 109.0)

```

```
14 (From,To,Weight) = (4, 10, 137.0)
15 (From,To,Weight) = (5, 6, 127.0)
16 (From,To,Weight) = (6, 12, 277.0)
17 (From,To,Weight) = (6, 7, 126.0)
18 (From,To,Weight) = (7, 8, 120.0)
19 (From,To,Weight) = (8, 15, 293.0)
20 (From,To,Weight) = (9, 11, 58.0)
21 (From,To,Weight) = (9, 13, 229.0)
22 (From,To,Weight) = (12, 14, 272.0)
23 (From,To,Weight) = (12, 13, 154.0)
24 (From,To,Weight) = (13, 21, 196.0)
25 (From,To,Weight) = (14, 22, 238.0)
26 (From,To,Weight) = (15, 16, 200.0)
27 (From,To,Weight) = (15, 17, 234.0)
28 (From,To,Weight) = (16, 18, 280.0)
29 (From,To,Weight) = (18, 19, 80.0)
30 (From,To,Weight) = (19, 27, 124.0)
31 (From,To,Weight) = (20, 21, 420.0)
32 (From,To,Weight) = (22, 25, 120.0)
33 (From,To,Weight) = (23, 25, 133.0)
34 (From,To,Weight) = (23, 33, 290.0)
35 (From,To,Weight) = (23, 31, 216.0)
36 (From,To,Weight) = (24, 28, 178.0)
37 (From,To,Weight) = (25, 26, 206.0)
38 (From,To,Weight) = (26, 34, 247.0)
39 (From,To,Weight) = (26, 28, 228.0)
40 (From,To,Weight) = (29, 37, 193.0)
41 (From,To,Weight) = (30, 32, 96.0)
42 (From,To,Weight) = (30, 31, 114.0)
43 (From,To,Weight) = (33, 41, 306.0)
44 (From,To,Weight) = (34, 35, 230.0)
45 (From,To,Weight) = (34, 46, 158.0)
46 (From,To,Weight) = (35, 36, 211.0)
47 (From,To,Weight) = (37, 39, 174.0)
48 (From,To,Weight) = (38, 43, 143.0)
49 (From,To,Weight) = (39, 50, 282.0)
50 (From,To,Weight) = (40, 44, 162.0)
51 (From,To,Weight) = (41, 45, 91.0)
52 (From,To,Weight) = (42, 50, 288.0)
53 (From,To,Weight) = (42, 43, 188.0)
54 (From,To,Weight) = (42, 52, 232.0)
55 (From,To,Weight) = (43, 51, 429.0)
56 (From,To,Weight) = (44, 50, 208.0)
57 (From,To,Weight) = (44, 45, 194.0)
58 (From,To,Weight) = (45, 47, 342.0)
59 (From,To,Weight) = (47, 49, 247.0)
60 (From,To,Weight) = (47, 55, 311.0)
61 (From,To,Weight) = (48, 49, 289.0)
62 (From,To,Weight) = (50, 53, 414.0)
63 (From,To,Weight) = (53, 54, 345.0)
64 (From,To,Weight) = (54, 56, 114.0)
65 (From,To,Weight) = (55, 57, 346.0)
66 (From,To,Weight) = (57, 58, 320.0)
67 (From,To,Weight) = (58, 62, 485.0)
```

```
68 (From,To,Weight) = (59, 60, 239.0)
69 (From,To,Weight) = (60, 64, 377.0)
70 (From,To,Weight) = (61, 65, 409.0)
71 (From,To,Weight) = (62, 63, 344.0)
72 (From,To,Weight) = (63, 70, 346.0)
73 (From,To,Weight) = (64, 65, 445.0)
74 (From,To,Weight) = (64, 66, 341.0)
75 (From,To,Weight) = (65, 68, 503.0)
76 (From,To,Weight) = (66, 67, 208.0)
77 (From,To,Weight) = (68, 69, 263.0)
78 (From,To,Weight) = (69, 70, 452.0)
79 Total Cost is: 16743.0
80 Time Elapsed (in seconds): 0.181805426259
81 Number of Iterations: 69
82
83 |Networkx Algorithm|
84 Nodes of MST is: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16,
    ↪ 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33,
    ↪ 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50,
    ↪ 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67,
    ↪ 68, 69, 70]
85 Edges of MST is:
86
87 (From,To,Weight) = (1, 2, 206.0)
88 (From,To,Weight) = (2, 3, 186.0)
89 (From,To,Weight) = (2, 4, 220.0)
90 (From,To,Weight) = (3, 5, 109.0)
91 (From,To,Weight) = (4, 10, 137.0)
92 (From,To,Weight) = (5, 6, 127.0)
93 (From,To,Weight) = (6, 12, 277.0)
94 (From,To,Weight) = (6, 7, 126.0)
95 (From,To,Weight) = (7, 8, 120.0)
96 (From,To,Weight) = (8, 15, 293.0)
97 (From,To,Weight) = (9, 11, 58.0)
98 (From,To,Weight) = (9, 13, 229.0)
99 (From,To,Weight) = (12, 14, 272.0)
100 (From,To,Weight) = (12, 13, 154.0)
101 (From,To,Weight) = (13, 21, 196.0)
102 (From,To,Weight) = (14, 22, 238.0)
103 (From,To,Weight) = (15, 16, 200.0)
104 (From,To,Weight) = (15, 17, 234.0)
105 (From,To,Weight) = (16, 18, 280.0)
106 (From,To,Weight) = (18, 19, 80.0)
107 (From,To,Weight) = (19, 27, 124.0)
108 (From,To,Weight) = (20, 21, 420.0)
109 (From,To,Weight) = (22, 25, 120.0)
110 (From,To,Weight) = (23, 25, 133.0)
111 (From,To,Weight) = (23, 33, 290.0)
112 (From,To,Weight) = (23, 31, 216.0)
113 (From,To,Weight) = (24, 28, 178.0)
114 (From,To,Weight) = (25, 26, 206.0)
115 (From,To,Weight) = (26, 34, 247.0)
116 (From,To,Weight) = (26, 28, 228.0)
117 (From,To,Weight) = (29, 37, 193.0)
```

```
118 (From,To,Weight) = (30, 32, 96.0)
119 (From,To,Weight) = (30, 31, 114.0)
120 (From,To,Weight) = (33, 41, 306.0)
121 (From,To,Weight) = (34, 35, 230.0)
122 (From,To,Weight) = (34, 46, 158.0)
123 (From,To,Weight) = (35, 36, 211.0)
124 (From,To,Weight) = (37, 39, 174.0)
125 (From,To,Weight) = (38, 43, 143.0)
126 (From,To,Weight) = (39, 50, 282.0)
127 (From,To,Weight) = (40, 44, 162.0)
128 (From,To,Weight) = (41, 45, 91.0)
129 (From,To,Weight) = (42, 50, 288.0)
130 (From,To,Weight) = (42, 43, 188.0)
131 (From,To,Weight) = (42, 52, 232.0)
132 (From,To,Weight) = (43, 51, 429.0)
133 (From,To,Weight) = (44, 50, 208.0)
134 (From,To,Weight) = (44, 45, 194.0)
135 (From,To,Weight) = (45, 47, 342.0)
136 (From,To,Weight) = (47, 49, 247.0)
137 (From,To,Weight) = (47, 55, 311.0)
138 (From,To,Weight) = (48, 49, 289.0)
139 (From,To,Weight) = (50, 53, 414.0)
140 (From,To,Weight) = (53, 54, 345.0)
141 (From,To,Weight) = (54, 56, 114.0)
142 (From,To,Weight) = (55, 57, 346.0)
143 (From,To,Weight) = (57, 58, 320.0)
144 (From,To,Weight) = (58, 62, 485.0)
145 (From,To,Weight) = (59, 60, 239.0)
146 (From,To,Weight) = (60, 64, 377.0)
147 (From,To,Weight) = (61, 65, 409.0)
148 (From,To,Weight) = (62, 63, 344.0)
149 (From,To,Weight) = (63, 70, 346.0)
150 (From,To,Weight) = (64, 65, 445.0)
151 (From,To,Weight) = (64, 66, 341.0)
152 (From,To,Weight) = (65, 68, 503.0)
153 (From,To,Weight) = (66, 67, 208.0)
154 (From,To,Weight) = (68, 69, 263.0)
155 (From,To,Weight) = (69, 70, 452.0)
156 Total Cost is: 16743.0
157 Time Elapsed (in seconds): 0.00226027048127
```

## Capítulo 5

# Algoritmo de Kruskal

### 5.1 Resumo

Nesta implementação o a busca pelo ciclo no algoritmo é feita utilizando o DFS (Depth-first search) [6]. Em termos de performance, utilizar o Disjoint-Set Data Structure [7] tem uma performance melhor mas devido a problemas na implementação ele não foi utilizado.

### 5.2 Rede Italiana

```
1 *****
2 Kruskal's Algorithm
3 *****
4 Graph Name:  rede_italiana
5 Start Node:  1
6
7 |Original Algorithm|
8 Nodes of MST is:  [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16,
   ↪ 17, 18, 19, 20, 21]
9 Edges of MST is:
10
11 (From,To,Weight) = (1, 3, 110.0)
12 (From,To,Weight) = (2, 3, 110.0)
13 (From,To,Weight) = (2, 7, 90.0)
14 (From,To,Weight) = (3, 8, 95.0)
15 (From,To,Weight) = (3, 5, 90.0)
16 (From,To,Weight) = (4, 5, 85.0)
17 (From,To,Weight) = (6, 7, 90.0)
18 (From,To,Weight) = (8, 9, 55.0)
19 (From,To,Weight) = (9, 10, 60.0)
20 (From,To,Weight) = (9, 12, 110.0)
21 (From,To,Weight) = (11, 14, 130.0)
22 (From,To,Weight) = (12, 13, 120.0)
23 (From,To,Weight) = (12, 14, 170.0)
24 (From,To,Weight) = (13, 15, 180.0)
25 (From,To,Weight) = (15, 18, 90.0)
26 (From,To,Weight) = (16, 18, 100.0)
```

```

27 (From,To,Weight) = (17, 20, 420.0)
28 (From,To,Weight) = (18, 19, 200.0)
29 (From,To,Weight) = (19, 21, 210.0)
30 (From,To,Weight) = (20, 21, 150.0)
31 Total Cost is: 2665.0
32 Time Elapsed (in seconds): 0.00445373890321
33 Number of Iterations: 40
34
35 |Networkx Algorithm|
36 Nodes of MST is: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16,
    ↪ 17, 18, 19, 20, 21]
37 Edges of MST is:
38
39 (From,To,Weight) = (1, 3, 110.0)
40 (From,To,Weight) = (2, 3, 110.0)
41 (From,To,Weight) = (2, 7, 90.0)
42 (From,To,Weight) = (3, 8, 95.0)
43 (From,To,Weight) = (3, 5, 90.0)
44 (From,To,Weight) = (4, 5, 85.0)
45 (From,To,Weight) = (6, 7, 90.0)
46 (From,To,Weight) = (8, 9, 55.0)
47 (From,To,Weight) = (9, 10, 60.0)
48 (From,To,Weight) = (9, 12, 110.0)
49 (From,To,Weight) = (11, 14, 130.0)
50 (From,To,Weight) = (12, 13, 120.0)
51 (From,To,Weight) = (12, 14, 170.0)
52 (From,To,Weight) = (13, 15, 180.0)
53 (From,To,Weight) = (15, 18, 90.0)
54 (From,To,Weight) = (16, 18, 100.0)
55 (From,To,Weight) = (17, 20, 420.0)
56 (From,To,Weight) = (18, 19, 200.0)
57 (From,To,Weight) = (19, 21, 210.0)
58 (From,To,Weight) = (20, 21, 150.0)
59 Total Cost is: 2665.0
60 Time Elapsed (in seconds): 0.000456078316695

```

### 5.3 Rede USA

```

1 *****
2 Kruskal's Algorithm
3 *****
4 Graph Name: rede_usa
5 Start Node: 1
6
7 |Original Algorithm|
8 Nodes of MST is: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16,
    ↪ 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33,
    ↪ 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50,
    ↪ 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67,
    ↪ 68, 69, 70]
9 Edges of MST is:
10
11 (From,To,Weight) = (1, 2, 206.0)

```

```
12 (From,To,Weight) = (2, 3, 186.0)
13 (From,To,Weight) = (2, 4, 220.0)
14 (From,To,Weight) = (3, 5, 109.0)
15 (From,To,Weight) = (4, 10, 137.0)
16 (From,To,Weight) = (5, 6, 127.0)
17 (From,To,Weight) = (6, 7, 126.0)
18 (From,To,Weight) = (6, 12, 277.0)
19 (From,To,Weight) = (7, 8, 120.0)
20 (From,To,Weight) = (8, 15, 293.0)
21 (From,To,Weight) = (9, 11, 58.0)
22 (From,To,Weight) = (9, 13, 229.0)
23 (From,To,Weight) = (12, 13, 154.0)
24 (From,To,Weight) = (12, 14, 272.0)
25 (From,To,Weight) = (13, 21, 196.0)
26 (From,To,Weight) = (14, 22, 238.0)
27 (From,To,Weight) = (15, 16, 200.0)
28 (From,To,Weight) = (15, 17, 234.0)
29 (From,To,Weight) = (16, 18, 280.0)
30 (From,To,Weight) = (18, 19, 80.0)
31 (From,To,Weight) = (19, 27, 124.0)
32 (From,To,Weight) = (20, 21, 420.0)
33 (From,To,Weight) = (22, 25, 120.0)
34 (From,To,Weight) = (23, 33, 290.0)
35 (From,To,Weight) = (23, 25, 133.0)
36 (From,To,Weight) = (23, 31, 216.0)
37 (From,To,Weight) = (24, 28, 178.0)
38 (From,To,Weight) = (25, 26, 206.0)
39 (From,To,Weight) = (26, 34, 247.0)
40 (From,To,Weight) = (26, 28, 228.0)
41 (From,To,Weight) = (29, 37, 193.0)
42 (From,To,Weight) = (30, 32, 96.0)
43 (From,To,Weight) = (30, 31, 114.0)
44 (From,To,Weight) = (33, 41, 306.0)
45 (From,To,Weight) = (34, 35, 230.0)
46 (From,To,Weight) = (34, 46, 158.0)
47 (From,To,Weight) = (35, 36, 211.0)
48 (From,To,Weight) = (37, 39, 174.0)
49 (From,To,Weight) = (38, 43, 143.0)
50 (From,To,Weight) = (39, 50, 282.0)
51 (From,To,Weight) = (40, 44, 162.0)
52 (From,To,Weight) = (41, 45, 91.0)
53 (From,To,Weight) = (42, 43, 188.0)
54 (From,To,Weight) = (42, 50, 288.0)
55 (From,To,Weight) = (42, 52, 232.0)
56 (From,To,Weight) = (43, 51, 429.0)
57 (From,To,Weight) = (44, 45, 194.0)
58 (From,To,Weight) = (44, 50, 208.0)
59 (From,To,Weight) = (45, 47, 342.0)
60 (From,To,Weight) = (47, 49, 247.0)
61 (From,To,Weight) = (47, 55, 311.0)
62 (From,To,Weight) = (48, 49, 289.0)
63 (From,To,Weight) = (50, 53, 414.0)
64 (From,To,Weight) = (53, 54, 345.0)
65 (From,To,Weight) = (54, 56, 114.0)
```

```
66 (From,To,Weight) = (55, 57, 346.0)
67 (From,To,Weight) = (57, 58, 320.0)
68 (From,To,Weight) = (58, 62, 485.0)
69 (From,To,Weight) = (59, 60, 239.0)
70 (From,To,Weight) = (60, 64, 377.0)
71 (From,To,Weight) = (61, 65, 409.0)
72 (From,To,Weight) = (62, 63, 344.0)
73 (From,To,Weight) = (63, 70, 346.0)
74 (From,To,Weight) = (64, 65, 445.0)
75 (From,To,Weight) = (64, 66, 341.0)
76 (From,To,Weight) = (65, 68, 503.0)
77 (From,To,Weight) = (66, 67, 208.0)
78 (From,To,Weight) = (68, 69, 263.0)
79 (From,To,Weight) = (69, 70, 452.0)
80 Total Cost is: 16743.0
81 Time Elapsed (in seconds): 0.0630622171308
82 Number of Iterations: 210
83
84 |Networkx Algorithm|
85 Nodes of MST is: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16,
    ↪ 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33,
    ↪ 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50,
    ↪ 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67,
    ↪ 68, 69, 70]
86 Edges of MST is:
87
88 (From,To,Weight) = (1, 2, 206.0)
89 (From,To,Weight) = (2, 3, 186.0)
90 (From,To,Weight) = (2, 4, 220.0)
91 (From,To,Weight) = (3, 5, 109.0)
92 (From,To,Weight) = (4, 10, 137.0)
93 (From,To,Weight) = (5, 6, 127.0)
94 (From,To,Weight) = (6, 12, 277.0)
95 (From,To,Weight) = (6, 7, 126.0)
96 (From,To,Weight) = (7, 8, 120.0)
97 (From,To,Weight) = (8, 15, 293.0)
98 (From,To,Weight) = (9, 11, 58.0)
99 (From,To,Weight) = (9, 13, 229.0)
100 (From,To,Weight) = (12, 13, 154.0)
101 (From,To,Weight) = (12, 14, 272.0)
102 (From,To,Weight) = (13, 21, 196.0)
103 (From,To,Weight) = (14, 22, 238.0)
104 (From,To,Weight) = (15, 16, 200.0)
105 (From,To,Weight) = (15, 17, 234.0)
106 (From,To,Weight) = (16, 18, 280.0)
107 (From,To,Weight) = (18, 19, 80.0)
108 (From,To,Weight) = (19, 27, 124.0)
109 (From,To,Weight) = (20, 21, 420.0)
110 (From,To,Weight) = (22, 25, 120.0)
111 (From,To,Weight) = (23, 25, 133.0)
112 (From,To,Weight) = (23, 33, 290.0)
113 (From,To,Weight) = (23, 31, 216.0)
114 (From,To,Weight) = (24, 28, 178.0)
115 (From,To,Weight) = (25, 26, 206.0)
```



```
116 (From,To,Weight) = (26, 34, 247.0)
117 (From,To,Weight) = (26, 28, 228.0)
118 (From,To,Weight) = (29, 37, 193.0)
119 (From,To,Weight) = (30, 32, 96.0)
120 (From,To,Weight) = (30, 31, 114.0)
121 (From,To,Weight) = (33, 41, 306.0)
122 (From,To,Weight) = (34, 35, 230.0)
123 (From,To,Weight) = (34, 46, 158.0)
124 (From,To,Weight) = (35, 36, 211.0)
125 (From,To,Weight) = (37, 39, 174.0)
126 (From,To,Weight) = (38, 43, 143.0)
127 (From,To,Weight) = (39, 50, 282.0)
128 (From,To,Weight) = (40, 44, 162.0)
129 (From,To,Weight) = (41, 45, 91.0)
130 (From,To,Weight) = (42, 50, 288.0)
131 (From,To,Weight) = (42, 43, 188.0)
132 (From,To,Weight) = (42, 52, 232.0)
133 (From,To,Weight) = (43, 51, 429.0)
134 (From,To,Weight) = (44, 50, 208.0)
135 (From,To,Weight) = (44, 45, 194.0)
136 (From,To,Weight) = (45, 47, 342.0)
137 (From,To,Weight) = (47, 49, 247.0)
138 (From,To,Weight) = (47, 55, 311.0)
139 (From,To,Weight) = (48, 49, 289.0)
140 (From,To,Weight) = (50, 53, 414.0)
141 (From,To,Weight) = (53, 54, 345.0)
142 (From,To,Weight) = (54, 56, 114.0)
143 (From,To,Weight) = (55, 57, 346.0)
144 (From,To,Weight) = (57, 58, 320.0)
145 (From,To,Weight) = (58, 62, 485.0)
146 (From,To,Weight) = (59, 60, 239.0)
147 (From,To,Weight) = (60, 64, 377.0)
148 (From,To,Weight) = (61, 65, 409.0)
149 (From,To,Weight) = (62, 63, 344.0)
150 (From,To,Weight) = (63, 70, 346.0)
151 (From,To,Weight) = (64, 65, 445.0)
152 (From,To,Weight) = (64, 66, 341.0)
153 (From,To,Weight) = (65, 68, 503.0)
154 (From,To,Weight) = (66, 67, 208.0)
155 (From,To,Weight) = (68, 69, 263.0)
156 (From,To,Weight) = (69, 70, 452.0)
157 Total Cost is: 16743.0
158 Time Elapsed (in seconds): 0.00176073058381
```

## Capítulo 6

# Algoritmo de Dijkstra

### 6.1 Resumo

A lógica de implementação foi igual ao que foi apresentado em [8].

### 6.2 Rede Italiana

#### 6.2.1 De 1 a 7

```
1 *****
2 Dijkstra's Algorithm
3 *****
4 Graph Name: rede_italiana
5 Start Node: 1
6 End Node: 7
7
8 |Original Algorithm|
9 Shortest Path from 1 to 7 is: [1, 2, 7]
10 Previous Node before 7 is : 2
11 Shortest Path Length from 1 to 7 is: 230.0
12 Time Elapsed (in seconds): 0.000127165365949
13 Number of Iterations: 6
14 Number of Relaxations: 10
15
16 |Networkx Algorithm|
17 Shortest Path from 1 to 7 is : [1, 2, 7]
18 Previous Node before 7 is : 2
19 Shortest Path Length from 1 to 7 is : 230.0
20 Time Elapsed (in seconds): 0.000368618592435
```

#### 6.2.2 De 1 a 14

```
1 *****
2 Dijkstra's Algorithm
3 *****
4 Graph Name: rede_italiana
5 Start Node: 1
```

```

6 End Node: 14
7
8 |Original Algorithm|
9 Shortest Path from 1 to 14 is: [1, 3, 8, 11, 14]
10 Previous Node before 14 is : 11
11 Shortest Path Length from 1 to 14 is: 535.0
12 Time Elapsed (in seconds): 0.000266671674585
13 Number of Iterations: 13
14 Number of Relaxations: 18
15
16 |Networkx Algorithm|
17 Shortest Path from 1 to 14 is : [1, 3, 8, 11, 14]
18 Previous Node before 14 is : 11
19 Shortest Path Length from 1 to 14 is : 535.0
20 Time Elapsed (in seconds): 0.000167944133089

```

### 6.2.3 De 1 a 21

```

1 *****
2 Dijkstra's Algorithm
3 *****
4 Graph Name: rede_italiana
5 Start Node: 1
6 End Node: 21
7
8 |Original Algorithm|
9 Shortest Path from 1 to 21 is: [1, 3, 8, 9, 13, 15, 21]
10 Previous Node before 21 is : 15
11 Shortest Path Length from 1 to 21 is: 970.0
12 Time Elapsed (in seconds): 0.000479150513893
13 Number of Iterations: 20
14 Number of Relaxations: 23
15
16 |Networkx Algorithm|
17 Shortest Path from 1 to 21 is : [1, 3, 8, 9, 13, 15, 21]
18 Previous Node before 21 is : 15
19 Shortest Path Length from 1 to 21 is : 970.0
20 Time Elapsed (in seconds): 0.000195040550728

```

## 6.3 Rede USA

### 6.3.1 De 1 a 10

```

1 *****
2 Dijkstra's Algorithm
3 *****
4 Graph Name: rede_usa
5 Start Node: 1
6 End Node: 10
7
8 |Original Algorithm|
9 Shortest Path from 1 to 10 is: [1, 4, 10]
10 Previous Node before 10 is : 4

```

```

11 Shortest Path Length from 1 to 10 is: 486.0
12 Time Elapsed (in seconds): 0.000184309296218
13 Number of Iterations: 5
14 Number of Relaxations: 11
15
16 |Networkx Algorithm|
17 Shortest Path from 1 to 10 is : [1, 4, 10]
18 Previous Node before 10 is : 4
19 Shortest Path Length from 1 to 10 is : 486.0
20 Time Elapsed (in seconds): 0.0013832587064

```

### 6.3.2 De 1 a 20

```

1 *****
2 Dijkstra's Algorithm
3 *****
4 Graph Name: rede_usa
5 Start Node: 1
6 End Node: 20
7
8 |Original Algorithm|
9 Shortest Path from 1 to 20 is: [1, 4, 10, 20]
10 Previous Node before 20 is : 10
11 Shortest Path Length from 1 to 20 is: 1086.0
12 Time Elapsed (in seconds): 0.000575731804487
13 Number of Iterations: 18
14 Number of Relaxations: 34
15
16 |Networkx Algorithm|
17 Shortest Path from 1 to 20 is : [1, 4, 10, 20]
18 Previous Node before 20 is : 10
19 Shortest Path Length from 1 to 20 is : 1086.0
20 Time Elapsed (in seconds): 0.000922351325176

```

### 6.3.3 De 1 a 30

```

1 *****
2 Dijkstra's Algorithm
3 *****
4 Graph Name: rede_usa
5 Start Node: 1
6 End Node: 30
7
8 |Original Algorithm|
9 Shortest Path from 1 to 30 is: [1, 2, 9, 13, 31, 30]
10 Previous Node before 30 is : 31
11 Shortest Path Length from 1 to 30 is: 1315.0
12 Time Elapsed (in seconds): 0.00154530064951
13 Number of Iterations: 24
14 Number of Relaxations: 48
15
16 |Networkx Algorithm|
17 Shortest Path from 1 to 30 is : [1, 2, 9, 13, 31, 30]

```

```
18 Previous Node before 30 is : 31
19 Shortest Path Length from 1 to 30 is : 1315.0
20 Time Elapsed (in seconds): 0.00119412034565
```

### 6.3.4 De 1 a 40

```
1 *****
2 Dijkstra's Algorithm
3 *****
4 Graph Name: rede_usa
5 Start Node: 1
6 End Node: 40
7
8 |Original Algorithm|
9 Shortest Path from 1 to 40 is: [1, 2, 9, 13, 40]
10 Previous Node before 40 is : 13
11 Shortest Path Length from 1 to 40 is: 1603.0
12 Time Elapsed (in seconds): 0.0033572729736
13 Number of Iterations: 34
14 Number of Relaxations: 60
15
16 |Networkx Algorithm|
17 Shortest Path from 1 to 40 is : [1, 2, 9, 13, 40]
18 Previous Node before 40 is : 13
19 Shortest Path Length from 1 to 40 is : 1603.0
20 Time Elapsed (in seconds): 0.000997470106749
```

### 6.3.5 De 1 a 50

```
1 *****
2 Dijkstra's Algorithm
3 *****
4 Graph Name: rede_usa
5 Start Node: 1
6 End Node: 50
7
8 |Original Algorithm|
9 Shortest Path from 1 to 50 is: [1, 2, 9, 13, 40, 44, 50]
10 Previous Node before 50 is : 44
11 Shortest Path Length from 1 to 50 is: 1973.0
12 Time Elapsed (in seconds): 0.00145220701663
13 Number of Iterations: 47
14 Number of Relaxations: 77
15
16 |Networkx Algorithm|
17 Shortest Path from 1 to 50 is : [1, 2, 9, 13, 40, 44, 50]
18 Previous Node before 50 is : 44
19 Shortest Path Length from 1 to 50 is : 1973.0
20 Time Elapsed (in seconds): 0.000924497576078
```

### 6.3.6 De 1 a 60

```
1 *****
2 Dijkstra's Algorithm
3 *****
4 Graph Name: rede_usa
5 Start Node: 1
6 End Node: 60
7
8 |Original Algorithm|
9 Shortest Path from 1 to 60 is: [1, 4, 10, 20, 37, 42, 52, 60]
10 Previous Node before 60 is : 52
11 Shortest Path Length from 1 to 60 is: 2964.0
12 Time Elapsed (in seconds): 0.0021736156011
13 Number of Iterations: 59
14 Number of Relaxations: 90
15
16 |Networkx Algorithm|
17 Shortest Path from 1 to 60 is : [1, 4, 10, 20, 37, 42, 52, 60]
18 Previous Node before 60 is : 52
19 Shortest Path Length from 1 to 60 is : 2964.0
20 Time Elapsed (in seconds): 0.00119385206429
```

### 6.3.7 De 1 a 70

```
1 *****
2 Dijkstra's Algorithm
3 *****
4 Graph Name: rede_usa
5 Start Node: 1
6 End Node: 70
7
8 |Original Algorithm|
9 Shortest Path from 1 to 70 is: [1, 3, 23, 33, 47, 55, 57, 70]
10 Previous Node before 70 is : 57
11 Shortest Path Length from 1 to 70 is: 3429.0
12 Time Elapsed (in seconds): 0.00464636492168
13 Number of Iterations: 65
14 Number of Relaxations: 96
15
16 |Networkx Algorithm|
17 Shortest Path from 1 to 70 is : [1, 3, 23, 33, 47, 55, 57, 70]
18 Previous Node before 70 is : 57
19 Shortest Path Length from 1 to 70 is : 3429.0
20 Time Elapsed (in seconds): 0.00102563964984
```

## Capítulo 7

# Algoritmo de Ford-Moore-Bellman

### 7.1 Resumo

A lógica de implementação foi igual ao que foi apresentado em [8].

### 7.2 Rede Italiana

#### 7.2.1 De 1 a 7

```
1 *****
2 Ford-Moore-Bellman's Algorithm
3 *****
4 Graph Name:  rede_italiana
5 Start Node:  1
6 End Node:   7
7
8 |Original Algorithm|
9 Shortest Path from 1 to 7 is: [1, 2, 7]
10 Previous Node before 7 is : 2
11 Shortest Path Length from 1 to 7 is: 230.0
12 Time Elapsed (in seconds): 0.00121933879375
13 Number of Iterations: 40
14 Number of Relaxations: 24
15
16 |Networkx Algorithm|
17 Shortest Path from 1 to 7 is : [1, 2, 7]
18 Previous Node before 7 is : 2
19 Shortest Path Length from 1 to 7 is : 230.0
20 Time Elapsed (in seconds): 0.000209259462954
```

#### 7.2.2 De 1 a 14

```
1 *****
2 Ford-Moore-Bellman's Algorithm
```

```

3 *****
4 Graph Name: rede_italiana
5 Start Node: 1
6 End Node: 14
7
8 |Original Algorithm|
9 Shortest Path from 1 to 14 is: [1, 3, 8, 11, 14]
10 Previous Node before 14 is : 11
11 Shortest Path Length from 1 to 14 is: 535.0
12 Time Elapsed (in seconds): 8.26306597307e-05
13 Number of Iterations: 40
14 Number of Relaxations: 24
15
16 |Networkx Algorithm|
17 Shortest Path from 1 to 14 is : [1, 3, 8, 11, 14]
18 Previous Node before 14 is : 11
19 Shortest Path Length from 1 to 14 is : 535.0
20 Time Elapsed (in seconds): 0.000125555677773

```

### 7.2.3 De 1 a 21

```

1 *****
2 Ford-Moore-Bellman's Algorithm
3 *****
4 Graph Name: rede_italiana
5 Start Node: 1
6 End Node: 21
7
8 |Original Algorithm|
9 Shortest Path from 1 to 21 is: [1, 3, 8, 9, 13, 15, 21]
10 Previous Node before 21 is : 15
11 Shortest Path Length from 1 to 21 is: 970.0
12 Time Elapsed (in seconds): 9.68495719571e-05
13 Number of Iterations: 40
14 Number of Relaxations: 24
15
16 |Networkx Algorithm|
17 Shortest Path from 1 to 21 is : [1, 3, 8, 9, 13, 15, 21]
18 Previous Node before 21 is : 15
19 Shortest Path Length from 1 to 21 is : 970.0
20 Time Elapsed (in seconds): 0.000138164901822

```

## 7.3 Rede USA

### 7.3.1 De 1 a 10

```

1 *****
2 Ford-Moore-Bellman's Algorithm
3 *****
4 Graph Name: rede_usa
5 Start Node: 1
6 End Node: 10
7

```



```

8 |Original Algorithm|
9 Shortest Path from 1 to 10 is: [1, 4, 10]
10 Previous Node before 10 is : 4
11 Shortest Path Length from 1 to 10 is: 486.0
12 Time Elapsed (in seconds): 0.000444005655371
13 Number of Iterations: 210
14 Number of Relaxations: 100
15
16 |Networkx Algorithm|
17 Shortest Path from 1 to 10 is : [1, 4, 10]
18 Previous Node before 10 is : 4
19 Shortest Path Length from 1 to 10 is : 486.0
20 Time Elapsed (in seconds): 0.000552391325927

```

### 7.3.2 De 1 a 20

```

1 *****
2 Ford-Moore-Bellman's Algorithm
3 *****
4 Graph Name: rede_usa
5 Start Node: 1
6 End Node: 20
7
8 |Original Algorithm|
9 Shortest Path from 1 to 20 is: [1, 4, 10, 20]
10 Previous Node before 20 is : 10
11 Shortest Path Length from 1 to 20 is: 1086.0
12 Time Elapsed (in seconds): 0.000457151442146
13 Number of Iterations: 210
14 Number of Relaxations: 100
15
16 |Networkx Algorithm|
17 Shortest Path from 1 to 20 is : [1, 4, 10, 20]
18 Previous Node before 20 is : 10
19 Shortest Path Length from 1 to 20 is : 1086.0
20 Time Elapsed (in seconds): 0.000539782101877

```

### 7.3.3 De 1 a 30

```

1 *****
2 Ford-Moore-Bellman's Algorithm
3 *****
4 Graph Name: rede_usa
5 Start Node: 1
6 End Node: 30
7
8 |Original Algorithm|
9 Shortest Path from 1 to 30 is: [1, 4, 10, 20, 30]
10 Previous Node before 30 is : 20
11 Shortest Path Length from 1 to 30 is: 1600.0
12 Time Elapsed (in seconds): 0.000370228280612
13 Number of Iterations: 210
14 Number of Relaxations: 100

```

```

15
16 |Networkx Algorithm|
17 Shortest Path from 1 to 30 is : [1, 2, 9, 13, 31, 30]
18 Previous Node before 30 is : 31
19 Shortest Path Length from 1 to 30 is : 1315.0
20 Time Elapsed (in seconds): 0.000482906452972

```

### 7.3.4 De 1 a 40

```

1 *****
2 Ford-Moore-Bellman's Algorithm
3 *****
4 Graph Name: rede_usa
5 Start Node: 1
6 End Node: 40
7
8 |Original Algorithm|
9 Shortest Path from 1 to 40 is: [1, 2, 9, 13, 40]
10 Previous Node before 40 is : 13
11 Shortest Path Length from 1 to 40 is: 1603.0
12 Time Elapsed (in seconds): 0.000309328411265
13 Number of Iterations: 210
14 Number of Relaxations: 100
15
16 |Networkx Algorithm|
17 Shortest Path from 1 to 40 is : [1, 2, 9, 13, 40]
18 Previous Node before 40 is : 13
19 Shortest Path Length from 1 to 40 is : 1603.0
20 Time Elapsed (in seconds): 0.000489881768403

```

### 7.3.5 De 1 a 50

```

1 *****
2 Ford-Moore-Bellman's Algorithm
3 *****
4 Graph Name: rede_usa
5 Start Node: 1
6 End Node: 50
7
8 |Original Algorithm|
9 Shortest Path from 1 to 50 is: [1, 2, 9, 13, 40, 44, 50]
10 Previous Node before 50 is : 44
11 Shortest Path Length from 1 to 50 is: 1973.0
12 Time Elapsed (in seconds): 0.000394641884623
13 Number of Iterations: 210
14 Number of Relaxations: 100
15
16 |Networkx Algorithm|
17 Shortest Path from 1 to 50 is : [1, 2, 9, 13, 40, 44, 50]
18 Previous Node before 50 is : 44
19 Shortest Path Length from 1 to 50 is : 1973.0
20 Time Elapsed (in seconds): 0.000557220390457

```

### 7.3.6 De 1 a 60

```
1 *****
2 Ford-Moore-Bellman's Algorithm
3 *****
4 Graph Name: rede_usa
5 Start Node: 1
6 End Node: 60
7
8 |Original Algorithm|
9 Shortest Path from 1 to 60 is: [1, 4, 10, 20, 37, 42, 52, 60]
10 Previous Node before 60 is : 52
11 Shortest Path Length from 1 to 60 is: 2964.0
12 Time Elapsed (in seconds): 0.000383105786024
13 Number of Iterations: 210
14 Number of Relaxations: 100
15
16 |Networkx Algorithm|
17 Shortest Path from 1 to 60 is : [1, 4, 10, 20, 37, 42, 52, 60]
18 Previous Node before 60 is : 52
19 Shortest Path Length from 1 to 60 is : 2964.0
20 Time Elapsed (in seconds): 0.00051483193514
```

### 7.3.7 De 1 a 70

```
1 *****
2 Ford-Moore-Bellman's Algorithm
3 *****
4 Graph Name: rede_usa
5 Start Node: 1
6 End Node: 70
7
8 |Original Algorithm|
9 Shortest Path from 1 to 70 is: [1, 3, 23, 33, 47, 55, 57, 70]
10 Previous Node before 70 is : 57
11 Shortest Path Length from 1 to 70 is: 3429.0
12 Time Elapsed (in seconds): 0.0003710331247
13 Number of Iterations: 210
14 Number of Relaxations: 100
15
16 |Networkx Algorithm|
17 Shortest Path from 1 to 70 is : [1, 3, 23, 33, 47, 55, 57, 70]
18 Previous Node before 70 is : 57
19 Shortest Path Length from 1 to 70 is : 3429.0
20 Time Elapsed (in seconds): 0.000539245539152
```

# Bibliografia

- [1] Python Project, <https://www.python.org/>, Visitado em : 22 de Novembro de 2016
- [2] Python - Wikipedia Article, <https://pt.wikipedia.org/wiki/Python>, Visitado em : 22 de Novembro de 2016
- [3] Why is Python used for high-performance/scientific computing?, <https://goo.gl/8aRXjN>, Visitado em : 22 de Novembro de 2016
- [4] Networkx, <https://networkx.github.io/>, Visitado em : 22 de Novembro de 2016
- [5] Networkx Lastest Documentation, <https://networkx.readthedocs.io/en/stable/>, Visitado em : 22 de Novembro de 2016
- [6] Depth-first Algorithm, [https://en.wikipedia.org/wiki/Depth-first\\_search](https://en.wikipedia.org/wiki/Depth-first_search), Visitado em : 22 de Novembro de 2016
- [7] Disjoint-set data structure, [https://en.wikipedia.org/wiki/Disjoint-set\\_data\\_structure](https://en.wikipedia.org/wiki/Disjoint-set_data_structure), Visitado em : 22 de Novembro de 2016
- [8] Notas de Aula IA 881 - 2º semestre de 2016, <http://www.dt.fee.unicamp.br/~ricfow/IA881/ia881.htm>, Visitado em : 22 de Novembro de 2016

# Apêndice A

## Prim

```
1  # see documentation at : https://networkx.github.io/
2  import networkx as nx
3
4  # see documentation at: https://docs.python.org/2/library/timeit.html
5  import timeit
6
7  # see documentation at: https://docs.python.org/2/library/itertools.html
8  import itertools
9
10 # see documentation at: https://docs.python.org/2/library/sys.html
11 import sys
12
13 def getWeight(k):
14     return k[2]
15
16 def return_minimum_edge_fringe(fringe):
17     ordered_edges_by_weight =
18         ↪ sorted(fringe.edges(data='weight'),key=getWeight)
19     return ordered_edges_by_weight[0]
20
21 def build_fringe(g,mst):
22
23     fringe = nx.Graph()
24
25     fringe_nodes = set(nx.nodes(g)).difference(set(nx.nodes(mst)))
26
27     fringe.add_nodes_from(set(nx.nodes(g)).difference(set(nx.nodes(mst))))
28
29     for node in itertools.product(nx.nodes(mst),nx.nodes(fringe)):
30         if g.has_edge(*node) or g.has_edge(*tuple(reversed(node))):
31             fringe.add_edge(*node,weight=g.get_edge_data(*node)['weight'])
32
33     for i in nx.nodes(fringe):
34         if fringe.degree(i) == 0:
35             fringe.remove_node(i)
36
37     return fringe
38
39 def prim(G,start_node):
40
41     global qty_of_iterations
42
43     total_cost = 0
```

```

44
45     mst = nx.Graph()
46     mst.add_node(start_node)
47
48     while (nx.number_of_nodes(mst) < nx.number_of_nodes(G)):
49
50         fringe = build_fringe(G,mst)
51         min_edge = return_minimum_edge_fringe(fringe)
52
53         mst.add_edge(min_edge[0],min_edge[1],weight=getWeight(min_edge))
54
55         total_cost += getWeight(min_edge)
56         qty_of_iterations += 1
57
58     return total_cost,mst
59
60 def print_edges(G):
61     print ""
62     for i in G.edges(data='weight'):
63         print "(From,To,Weight) = ",i
64
65     # dictionary containing the necessary information
66     data = [
67         {
68             'algorithm' : "prim",
69             'graph_name' : "rede_italiana",
70             'file' : 'redeitaliana.ncol',
71             'start_node' : [1]
72         },
73         {
74             'algorithm' : "prim",
75             'graph_name' : "rede_usa",
76             'file' : 'redeusa.ncol',
77             'start_node' : [1]
78         }
79     ]
80
81
82     #main loop through data dictionary defined above
83     for d in data:
84
85         G = nx.read_weighted_edgelist(d['file'],nodetype=int)
86
87         f = file( d['algorithm'] + "_" + d['graph_name'] + '.txt', 'w')
88         sys.stdout = f
89
90         for s in d['start_node']:
91
92             #global variables
93             time_consumed = 0
94             qty_of_iterations = 0
95
96             print "*****"
97             print "Prim's Algorithm"
98             print "*****"
99             print "Graph Name: ",d['graph_name']
100            print "Start Node: ",s
101            print ""
102
103            #run the algorithm
104            time_consumed = 0
105            start_time = 0

```

```
106         start_time = timeit.default_timer()
107         total_cost,mst = prim(G,s)
108         time_consumed = timeit.default_timer() - start_time
109
110         print "|Original Algorithm|"
111
112         print "Nodes of MST is: ",mst.nodes()
113         print "Edges of MST is: ",
114         print_edges(mst)
115
116         print "Total Cost is: ",total_cost
117         print "Time Elapsed (in seconds): ", time_consumed
118         print "Number of Iterations: ",qty_of_iterations
119         print ""
120
121         #run the networkx algorithm
122         time_consumed = 0
123         start_time = 0
124         start_time = timeit.default_timer()
125         T = nx.prim_mst(G)
126         time_consumed = timeit.default_timer() - start_time
127
128         print "|Networkx Algorithm|"
129
130         print "Nodes of MST is: ",T.nodes()
131         print "Edges of MST is: "
132         print_edges(T)
133
134         total_cost = 0
135         for edge in T.edges(data='weight'):
136             total_cost += getWeight(edge)
137
138         print "Total Cost is: ",total_cost
139         print "Time Elapsed (in seconds): ", time_consumed
140         print ""
141
142     f.close()
```

## Apêndice B

# Kruskal

```
1  # see documentation at : https://networkx.github.io/
2  import networkx as nx
3
4  # see documentation at: https://docs.python.org/2/library/timeit.html
5  import timeit
6
7  # see documentation at: https://docs.python.org/2/library/sys.html
8  import sys
9
10 def isCyclicUtil(G, v, visited, parent):
11
12     # Mark the current node as visited
13     visited[v] = True
14
15     # Search for all the vertices adjacent to this vertex
16     for edge in G.edges(v):
17         i = edge[1]
18
19         # If the node is not visited then recurse on it
20         if visited[i] == False:
21             if isCyclicUtil(G, i, visited, v):
22                 return True
23         # If an adjacent vertex is visited and not parent of current vertex, is a
24         ↪ cycle!
25         elif parent != i:
26             return True
27
28     return False
29
30 def isCyclic(G):
31
32     # Mark all the vertices as not visited
33     visited = {}
34     for i in G.nodes():
35         visited[i] = False
36
37     # Call the recursive helper function to detect cycle in different DFS trees
38     ↪
39     for i in G.nodes():
40         if visited[i] == False: # Don't recur for u if it is already visited
41             if isCyclicUtil(G, i, visited, -1) == True:
42                 return True
```



```

43     return False
44
45
46 def build_ordered_edges(g):
47     return sorted(G.edges(data=True), key=lambda (a, b, data): data['weight'])
48
49 def kruskal(G,s):
50
51     global qty_of_iterations
52
53     mst = nx.Graph()
54
55     ordered_edges = build_ordered_edges(G)
56
57     total_cost = 0
58
59     for candidate in ordered_edges:
60
61         u = candidate[0]
62         v = candidate[1]
63         weight = candidate[2]['weight']
64
65         mst.add_edge(u, v, weight=weight)
66
67         if (isCyclic(mst)):
68             mst.remove_edge(u,v)
69             total_cost -= weight
70
71         total_cost += weight
72
73         qty_of_iterations += 1
74
75     return total_cost,mst
76
77
78 def print_edges(G):
79     print ""
80     for i in G.edges(data='weight'):
81         print "(From,To,Weight) = ",i
82
83     # dictionary containing the necessary information
84     data = [
85         {
86             'algorithm' : "kruskal",
87             'graph_name' : "rede_italiana",
88             'file' : 'redeitaliana.ncol',
89             'start_node' : [1]
90         },
91         {
92             'algorithm' : "kruskal",
93             'graph_name' : "rede_usa",
94             'file' : 'redeusa.ncol',
95             'start_node' : [1]
96         }
97     ]
98
99     #main loop through data dictionary defined above
100    for d in data:
101
102        G = nx.read_weighted_edgelist(d['file'],nodetype=int)
103
104        f = file( d['algorithm'] + "_" + d['graph_name'] + '.txt', 'w')

```

```

105     sys.stdout = f
106
107     for s in d['start_node']:
108
109         #global variables
110         time_consumed = 0
111         qty_of_iterations = 0
112
113         print "*****"
114         print "Kruskal's Algorithm"
115         print "*****"
116         print "Graph Name: ",d['graph_name']
117         print "Start Node: ",s
118         print ""
119
120         #run the algorithm
121         time_consumed = 0
122         start_time = 0
123         start_time = timeit.default_timer()
124         total_cost,mst = kruskal(G,s)
125         time_consumed = timeit.default_timer() - start_time
126
127         print "|Original Algorithm|"
128
129         print "Nodes of MST is: ",mst.nodes()
130         print "Edges of MST is: "
131         print_edges(mst)
132         print "Total Cost is: ",total_cost
133         print "Time Elapsed (in seconds): ", time_consumed
134         print "Number of Iterations: ",qty_of_iterations
135         print ""
136
137
138         #run the networkx algorithm
139         time_consumed = 0
140         start_time = 0
141         start_time = timeit.default_timer()
142         #it's kruskal algorithm
143         T = nx.minimum_spanning_tree(G)
144         time_consumed = timeit.default_timer() - start_time
145
146         print "|Networkx Algorithm|"
147
148         print "Nodes of MST is: ",T.nodes()
149         print "Edges of MST is: "
150         print_edges(T)
151
152         total_cost = 0
153         for edge in T.edges(data='weight'):
154             total_cost += edge[2]
155
156         print "Total Cost is: ",total_cost
157         print "Time Elapsed (in seconds): ", time_consumed
158         print ""
159
160     f.close()

```

## Apêndice C

# Dijkstra

```
1  # see documentation at : https://networkx.github.io/
2  import networkx as nx
3
4  # see documentation at: https://docs.python.org/2/library/timeit.html
5  import timeit
6
7  # see documentation at: https://docs.python.org/2/library/sys.html
8  import sys
9
10 #gets the minimum node based on the visited nodes
11 def select_minimum_node(dist,visited_nodes):
12
13     minimum_value = float("inf")
14
15     minimum_node = 0
16
17     for i in visited_nodes:
18         if (visited_nodes[i] == 0):
19             if (dist[i] < minimum_value):
20                 minimum_node = i
21                 minimum_value = dist[i]
22
23     return minimum_node
24
25 #relax the edge
26 def relax_edge(prev,dist,edge):
27
28     global qty_of_relaxations
29
30     u = edge[0]
31     v = edge[1]
32     weight = edge[2]
33
34     if (dist[v] > dist[u] + weight):
35         dist[v] = dist[u] + weight
36         prev[v] = u
37
38     qty_of_relaxations = qty_of_relaxations + 1
39
40
41 #check if unvisited nodes still exist
42 def unvisited_nodes_exist(visited_nodes):
43
44     for i in visited_nodes:
```

```

45         if (visited_nodes[i] == 0):
46             return True
47
48     return False
49
50     #build list to show the path to reach the end node
51     def get_minimum_path(prev,end_node):
52
53         min_path = []
54
55         u = end_node
56         while prev[u] is not None:
57             min_path = [u] + min_path
58             u = prev[u]
59
60         min_path = [u] + min_path
61         return min_path
62
63     # dijkstra algorithm
64     def dijkstra(G,prev,dist,visited_nodes,start_node,end_node):
65
66         global time_consumed
67         global qty_of_iterations
68
69         #the minimum node is the start node
70         minimum_node = start_node
71
72         start_time = timeit.default_timer()
73
74         while (unvisited_nodes_exist(visited_nodes)):
75
76             minimum_node = select_minimum_node(dist,visited_nodes)
77
78             if minimum_node == end_node:
79                 break
80
81             for edge in G.edges(minimum_node,data='weight'):
82                 relax_edge(prev,dist,edge)
83
84             visited_nodes[minimum_node] = 1
85
86             qty_of_iterations = qty_of_iterations + 1
87
88             time_consumed = timeit.default_timer() - start_time
89
90     # dictionary containing the necessary information
91     data = [
92         {
93             'algorithm' : "dijkstra",
94             'graph_name' : "rede_italiana",
95             'file' : 'redeitaliana.ncol',
96             'start_node' : [1],
97             'end_node' : [7,14,21]
98         },
99         {
100             'algorithm' : "dijkstra",
101             'graph_name' : "rede_usa",
102             'file' : 'redeusa.ncol',
103             'start_node' : [1],
104             'end_node' : [10,20,30,40,50,60,70]
105         }
106     ]

```

```

107
108
109 #main loop through data dictionary defined above
110 for d in data:
111
112     G = nx.read_weighted_edgelist(d['file'], nodetype=int)
113
114     #start node
115     start_node = d['start_node']
116
117     #ending nodes
118     end_node = d['end_node']
119
120     for s in start_node:
121         for e in end_node:
122
123             f = file( d['algorithm'] + "_" + d['graph_name'] + "_" + str(s) + "_"
124                 ↪ + str(e) + ".txt", 'w' )
125             sys.stdout = f
126
127             print "*****"
128             print "Dijkstra's Algorithm"
129
130             #global variables
131             qty_of_relaxations = 0
132             qty_of_iterations = 0
133             time_consumed = 0
134
135             prev = {}
136             dist = {}
137             visited_nodes = {}
138             for i in G.nodes():
139                 #stores the last node between s and v
140                 prev[i] = None
141                 #stores the minimum path length between s and v.
142                 dist[i] = float('inf')
143                 #visited nodes : 1 is visited and 0 is unvisited
144                 visited_nodes[i] = 0
145
146             # initial node distance is zero
147             dist[s] = 0
148
149             print "*****"
150             print "Graph Name: ", d['graph_name']
151             print "Start Node: ", s
152             print "End Node: ", e
153             print ""
154
155             #run the algorithm
156             time_consumed = 0
157             start_time = 0
158             start_time = timeit.default_timer()
159             dijkstra(G, prev, dist, visited_nodes, s, e)
160             time_consumed = timeit.default_timer() - start_time
161
162             print "|Original Algorithm|"
163             print "Shortest Path from ", s, " to ", e, " is:
164             ↪ ", get_minimum_path(prev, e)
165             print "Previous Node before ", e, " is : ", prev[e]
166             print "Shortest Path Length from ", s, " to ", e, " is: ", dist[e]
167             print "Time Elapsed (in seconds): ", time_consumed
168             print "Number of Iterations: ", qty_of_iterations

```

```
167         print "Number of Relaxations: ",qty_of_relaxations
168         print ""
169
170         #run networkx algorithm
171         time_consumed = 0
172         start_time = 0
173         start_time = timeit.default_timer()
174         distance,path = nx.single_source_dijkstra(G, s)
175         time_consumed = timeit.default_timer() - start_time
176
177         print "|Networkx Algorithm|"
178         print "Shortest Path from ",s," to ",e," is : ",path[e]
179         print "Previous Node before ",e," is : ",path[e][len(path[e]) - 2]
180         print "Shortest Path Length from ",s," to ",e," is : ",distance[e]
181         print "Time Elapsed (in seconds): ", time_consumed
182         print ""
183
184     f.close()
```

## Apêndice D

# Ford-Moore-Bellman

```
1  # see documentation at : https://networkx.github.io/
2  import networkx as nx
3
4  # see documentation at: https://docs.python.org/2/library/timeit.html
5  import timeit
6
7  # see documentation at: https://docs.python.org/2/library/sys.html
8  import sys
9
10 #relax the edge
11 def relax_edge(prev,dist,edge):
12
13     global qty_of_relaxations
14
15     u = edge[0]
16     v = edge[1]
17     weight = edge[2]
18
19     if (dist[v] > dist[u] + weight):
20         dist[v] = dist[u] + weight
21         prev[v] = u
22
23         qty_of_relaxations = qty_of_relaxations + 1
24
25 #build list to show the path to reach the end node
26 def get_minimum_path(prev,end_node):
27
28     min_path = []
29
30     u = end_node
31     while prev[u] is not None:
32         min_path = [u] + min_path
33         u = prev[u]
34
35     min_path = [u] + min_path
36     return min_path
37
38 # ford_moore_bellman algorithm
39 def ford_moore_bellman(G,prev,dist):
40
41     global time_consumed
42     global qty_of_iterations
43
44     for edge in G.edges(data='weight'):
```

```

45         relax_edge(prev,dist,edge)
46         qty_of_iterations = qty_of_iterations + 1
47
48
49
50     # dictionary containing the necessary information
51     data = [
52         {
53             'algorithm' : "ford_moore_bellman",
54             'graph_name' : "rede_italiana",
55             'file' : 'redeitaliana.ncol',
56             'start_node' : [1],
57             'end_node' : [7,14,21]
58         },
59         {
60             'algorithm' : "ford_moore_bellman",
61             'graph_name' : "rede_usa",
62             'file' : 'redeusa.ncol',
63             'start_node' : [1],
64             'end_node' : [10,20,30,40,50,60,70]
65         }
66     ]
67
68
69     #main loop through data dictionary defined above
70     for d in data:
71
72         G = nx.read_weighted_edgelist(d['file'],nodetype=int)
73
74         #start node
75         start_node = d['start_node']
76
77         #ending nodes
78         end_node = d['end_node']
79
80         for s in start_node:
81             for e in end_node:
82
83                 f = file( d['algorithm'] + "_" + d['graph_name'] + "_" +
84                     ↪ str(s) + "_" + str(e) + ".txt", 'w' )
85                 sys.stdout = f
86
87                 print "*****"
88                 print "Ford-Moore-Bellman's Algorithm"
89
90                 #global variables
91                 qty_of_relaxations = 0
92                 qty_of_iterations = 0
93                 time_consumed = 0
94
95                 prev = {}
96                 dist = {}
97                 for i in G.nodes():
98                     #stores the last node between s and v
99                     prev[i] = None
100                     #stores the minimum path length between s and v.
101                     dist[i] = float('inf')
102
103                 # initial node distance is zero
104                 dist[s] = 0
105
106                 print "*****"

```



```

106         print "Graph Name: ",d['graph_name']
107         print "Start Node: ",s
108         print "End Node: ",e
109         print ""
110
111         #run the algorithm
112         time_consumed = 0
113         start_time = 0
114         start_time = timeit.default_timer()
115         ford_moore_bellman(G,prev,dist)
116         time_consumed = timeit.default_timer() - start_time
117
118         print "|Original Algorithm|"
119         print "Shortest Path from ",s," to ",e," is:
120         ↪      ",get_minimum_path(prev,e)
121         print "Previous Node before ",e," is : ",prev[e]
122         print "Shortest Path Length from ",s," to ",e," is:
123         ↪      ",dist[e]
124         print "Time Elapsed (in seconds): ", time_consumed
125         print "Number of Iterations: ",qty_of_iterations
126         print "Number of Relaxations: ",qty_of_relaxations
127         print ""
128
129         #run networkx algorithm
130         time_consumed = 0
131         start_time = 0
132         start_time = timeit.default_timer()
133         predecessor, distance = nx.bellman_ford(G, s)
134         time_consumed = timeit.default_timer() - start_time
135
136         print "|Networkx Algorithm|"
137         print "Shortest Path from ",s," to ",e," is :
138         ↪      ",get_minimum_path(predecessor,e)
139         print "Previous Node before ",e," is : ",predecessor[e]
140         print "Shortest Path Length from ",s," to ",e," is :
141         ↪      ",distance[e]
142         print "Time Elapsed (in seconds): ", time_consumed
143         print ""
144
145         f.close()

```