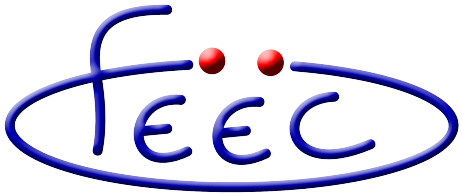

Universidade Estadual de Campinas

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 Março de 2017

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

Capítulo 1

A linguagem Python

A linguagem Python, conforme visto em [1] e [2] é uma linguagem de alto nivel, 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. Saliento 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,
   ↪  17, 18, 19, 20, 21]
9  Edges of MST is:
10 (From,To,Weight) =  (1, 3, 110.0)
11 (From,To,Weight) =  (2, 3, 110.0)
12 (From,To,Weight) =  (2, 7, 90.0)
13 (From,To,Weight) =  (3, 8, 95.0)
14 (From,To,Weight) =  (3, 5, 90.0)
15 (From,To,Weight) =  (4, 5, 85.0)
16 (From,To,Weight) =  (6, 7, 90.0)
17 (From,To,Weight) =  (8, 9, 55.0)
18 (From,To,Weight) =  (9, 10, 60.0)
19 (From,To,Weight) =  (9, 12, 110.0)
20 (From,To,Weight) =  (11, 14, 130.0)
21 (From,To,Weight) =  (12, 13, 120.0)
22 (From,To,Weight) =  (12, 14, 170.0)
23 (From,To,Weight) =  (13, 15, 180.0)
24 (From,To,Weight) =  (15, 18, 90.0)
25 (From,To,Weight) =  (16, 18, 100.0)
26 (From,To,Weight) =  (17, 20, 420.0)
27 (From,To,Weight) =  (18, 19, 200.0)
```

```
(From,To,Weight) =  (19, 21, 210.0)
(From,To,Weight) =  (20, 21, 150.0)
Total Cost is:  2665.0
Time Elapsed (in seconds):  0.00626485266356
Number of Iterations:  20

|Networkx Algorithm|
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]
Edges of MST is:

(From,To,Weight) =  (1, 3, 110.0)
(From,To,Weight) =  (2, 3, 110.0)
(From,To,Weight) =  (2, 7, 90.0)
(From,To,Weight) =  (3, 8, 95.0)
(From,To,Weight) =  (3, 5, 90.0)
(From,To,Weight) =  (4, 5, 85.0)
(From,To,Weight) =  (6, 7, 90.0)
(From,To,Weight) =  (8, 9, 55.0)
(From,To,Weight) =  (9, 10, 60.0)
(From,To,Weight) =  (9, 12, 110.0)
(From,To,Weight) =  (11, 14, 130.0)
(From,To,Weight) =  (12, 13, 120.0)
(From,To,Weight) =  (12, 14, 170.0)
(From,To,Weight) =  (13, 15, 180.0)
(From,To,Weight) =  (15, 18, 90.0)
(From,To,Weight) =  (16, 18, 100.0)
(From,To,Weight) =  (17, 20, 420.0)
(From,To,Weight) =  (18, 19, 200.0)
(From,To,Weight) =  (19, 21, 210.0)
(From,To,Weight) =  (20, 21, 150.0)
Total Cost is:  2665.0
Time Elapsed (in seconds):  0.000379738202458
```

4.3 Rede USA

```
************************************
Prim's Algorithm
************************************
Graph Name:  rede_usa
Start Node:  1

|Original Algorithm|
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]
Edges of MST is:
(From,To,Weight) =  (1, 2, 206.0)
(From,To,Weight) =  (2, 3, 186.0)
(From,To,Weight) =  (2, 4, 220.0)
(From,To,Weight) =  (3, 5, 109.0)
```

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

```
(From,To,Weight) =  (59, 60, 239.0)
(From,To,Weight) =  (60, 64, 377.0)
(From,To,Weight) =  (61, 65, 409.0)
(From,To,Weight) =  (62, 63, 344.0)
(From,To,Weight) =  (63, 70, 346.0)
(From,To,Weight) =  (64, 65, 445.0)
(From,To,Weight) =  (64, 66, 341.0)
(From,To,Weight) =  (65, 68, 503.0)
(From,To,Weight) =  (66, 67, 208.0)
(From,To,Weight) =  (68, 69, 263.0)
(From,To,Weight) =  (69, 70, 452.0)
Total Cost is:  16743.0
Time Elapsed (in seconds):  0.141956866772
Number of Iterations:  69

|Networkx Algorithm|
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]
Edges of MST is:

(From,To,Weight) =  (1, 2, 206.0)
(From,To,Weight) =  (2, 3, 186.0)
(From,To,Weight) =  (2, 4, 220.0)
(From,To,Weight) =  (3, 5, 109.0)
(From,To,Weight) =  (4, 10, 137.0)
(From,To,Weight) =  (5, 6, 127.0)
(From,To,Weight) =  (6, 12, 277.0)
(From,To,Weight) =  (6, 7, 126.0)
(From,To,Weight) =  (7, 8, 120.0)
(From,To,Weight) =  (8, 15, 293.0)
(From,To,Weight) =  (9, 11, 58.0)
(From,To,Weight) =  (9, 13, 229.0)
(From,To,Weight) =  (12, 14, 272.0)
(From,To,Weight) =  (12, 13, 154.0)
(From,To,Weight) =  (13, 21, 196.0)
(From,To,Weight) =  (14, 22, 238.0)
(From,To,Weight) =  (15, 16, 200.0)
(From,To,Weight) =  (15, 17, 234.0)
(From,To,Weight) =  (16, 18, 280.0)
(From,To,Weight) =  (18, 19, 80.0)
(From,To,Weight) =  (19, 27, 124.0)
(From,To,Weight) =  (20, 21, 420.0)
(From,To,Weight) =  (22, 25, 120.0)
(From,To,Weight) =  (23, 25, 133.0)
(From,To,Weight) =  (23, 33, 290.0)
(From,To,Weight) =  (23, 31, 216.0)
(From,To,Weight) =  (24, 28, 178.0)
(From,To,Weight) =  (25, 26, 206.0)
(From,To,Weight) =  (26, 34, 247.0)
(From,To,Weight) =  (26, 28, 228.0)
(From,To,Weight) =  (29, 37, 193.0)
```

```
(From,To,Weight) =  (30, 32, 96.0)
(From,To,Weight) =  (30, 31, 114.0)
(From,To,Weight) =  (33, 41, 306.0)
(From,To,Weight) =  (34, 35, 230.0)
(From,To,Weight) =  (34, 46, 158.0)
(From,To,Weight) =  (35, 36, 211.0)
(From,To,Weight) =  (37, 39, 174.0)
(From,To,Weight) =  (38, 43, 143.0)
(From,To,Weight) =  (39, 50, 282.0)
(From,To,Weight) =  (40, 44, 162.0)
(From,To,Weight) =  (41, 45, 91.0)
(From,To,Weight) =  (42, 50, 288.0)
(From,To,Weight) =  (42, 43, 188.0)
(From,To,Weight) =  (42, 52, 232.0)
(From,To,Weight) =  (43, 51, 429.0)
(From,To,Weight) =  (44, 50, 208.0)
(From,To,Weight) =  (44, 45, 194.0)
(From,To,Weight) =  (45, 47, 342.0)
(From,To,Weight) =  (47, 49, 247.0)
(From,To,Weight) =  (47, 55, 311.0)
(From,To,Weight) =  (48, 49, 289.0)
(From,To,Weight) =  (50, 53, 414.0)
(From,To,Weight) =  (53, 54, 345.0)
(From,To,Weight) =  (54, 56, 114.0)
(From,To,Weight) =  (55, 57, 346.0)
(From,To,Weight) =  (57, 58, 320.0)
(From,To,Weight) =  (58, 62, 485.0)
(From,To,Weight) =  (59, 60, 239.0)
(From,To,Weight) =  (60, 64, 377.0)
(From,To,Weight) =  (61, 65, 409.0)
(From,To,Weight) =  (62, 63, 344.0)
(From,To,Weight) =  (63, 70, 346.0)
(From,To,Weight) =  (64, 65, 445.0)
(From,To,Weight) =  (64, 66, 341.0)
(From,To,Weight) =  (65, 68, 503.0)
(From,To,Weight) =  (66, 67, 208.0)
(From,To,Weight) =  (68, 69, 263.0)
(From,To,Weight) =  (69, 70, 452.0)
Total Cost is:  16743.0
Time Elapsed (in seconds):  0.00149743320812
```

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

```
**********
Kruskal's Algorithm
**********
Graph Name:  rede_italiana
Start Node:  1

|Original Algorithm|
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]
Edges of MST is:

(From,To,Weight) =  (1, 3, 110.0)
(From,To,Weight) =  (2, 3, 110.0)
(From,To,Weight) =  (2, 7, 90.0)
(From,To,Weight) =  (3, 8, 95.0)
(From,To,Weight) =  (3, 5, 90.0)
(From,To,Weight) =  (4, 5, 85.0)
(From,To,Weight) =  (6, 7, 90.0)
(From,To,Weight) =  (8, 9, 55.0)
(From,To,Weight) =  (9, 10, 60.0)
(From,To,Weight) =  (9, 12, 110.0)
(From,To,Weight) =  (11, 14, 130.0)
(From,To,Weight) =  (12, 13, 120.0)
(From,To,Weight) =  (12, 14, 170.0)
(From,To,Weight) =  (13, 15, 180.0)
(From,To,Weight) =  (15, 18, 90.0)
(From,To,Weight) =  (16, 18, 100.0)
```

```
(From,To,Weight) =  (17, 20, 420.0)
(From,To,Weight) =  (18, 19, 200.0)
(From,To,Weight) =  (19, 21, 210.0)
(From,To,Weight) =  (20, 21, 150.0)
Total Cost is:  2665.0
Time Elapsed (in seconds):  0.00535540118828
Number of Iterations:  40

|Networkx Algorithm|
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]
Edges of MST is:

(From,To,Weight) =  (1, 3, 110.0)
(From,To,Weight) =  (2, 3, 110.0)
(From,To,Weight) =  (2, 7, 90.0)
(From,To,Weight) =  (3, 8, 95.0)
(From,To,Weight) =  (3, 5, 90.0)
(From,To,Weight) =  (4, 5, 85.0)
(From,To,Weight) =  (6, 7, 90.0)
(From,To,Weight) =  (8, 9, 55.0)
(From,To,Weight) =  (9, 10, 60.0)
(From,To,Weight) =  (9, 12, 110.0)
(From,To,Weight) =  (11, 14, 130.0)
(From,To,Weight) =  (12, 13, 120.0)
(From,To,Weight) =  (12, 14, 170.0)
(From,To,Weight) =  (13, 15, 180.0)
(From,To,Weight) =  (15, 18, 90.0)
(From,To,Weight) =  (16, 18, 100.0)
(From,To,Weight) =  (17, 20, 420.0)
(From,To,Weight) =  (18, 19, 200.0)
(From,To,Weight) =  (19, 21, 210.0)
(From,To,Weight) =  (20, 21, 150.0)
Total Cost is:  2665.0
Time Elapsed (in seconds):  0.00040986564485
```

5.3 Rede USA

```
**********************************
Kruskal's Algorithm
**********************************
Graph Name:  rede_usa
Start Node:  1

|Original Algorithm|
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]
Edges of MST is:

(From,To,Weight) =  (1, 2, 206.0)
```

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

```
(From,To,Weight) =  (55, 57, 346.0)
(From,To,Weight) =  (57, 58, 320.0)
(From,To,Weight) =  (58, 62, 485.0)
(From,To,Weight) =  (59, 60, 239.0)
(From,To,Weight) =  (60, 64, 377.0)
(From,To,Weight) =  (61, 65, 409.0)
(From,To,Weight) =  (62, 63, 344.0)
(From,To,Weight) =  (63, 70, 346.0)
(From,To,Weight) =  (64, 65, 445.0)
(From,To,Weight) =  (64, 66, 341.0)
(From,To,Weight) =  (65, 68, 503.0)
(From,To,Weight) =  (66, 67, 208.0)
(From,To,Weight) =  (68, 69, 263.0)
(From,To,Weight) =  (69, 70, 452.0)
Total Cost is:  16743.0
Time Elapsed (in seconds):  0.0486197327464
Number of Iterations:  210

|Networkx Algorithm|
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]
Edges of MST is:

(From,To,Weight) =  (1, 2, 206.0)
(From,To,Weight) =  (2, 3, 186.0)
(From,To,Weight) =  (2, 4, 220.0)
(From,To,Weight) =  (3, 5, 109.0)
(From,To,Weight) =  (4, 10, 137.0)
(From,To,Weight) =  (5, 6, 127.0)
(From,To,Weight) =  (6, 12, 277.0)
(From,To,Weight) =  (6, 7, 126.0)
(From,To,Weight) =  (7, 8, 120.0)
(From,To,Weight) =  (8, 15, 293.0)
(From,To,Weight) =  (9, 11, 58.0)
(From,To,Weight) =  (9, 13, 229.0)
(From,To,Weight) =  (12, 13, 154.0)
(From,To,Weight) =  (12, 14, 272.0)
(From,To,Weight) =  (13, 21, 196.0)
(From,To,Weight) =  (14, 22, 238.0)
(From,To,Weight) =  (15, 16, 200.0)
(From,To,Weight) =  (15, 17, 234.0)
(From,To,Weight) =  (16, 18, 280.0)
(From,To,Weight) =  (18, 19, 80.0)
(From,To,Weight) =  (19, 27, 124.0)
(From,To,Weight) =  (20, 21, 420.0)
(From,To,Weight) =  (22, 25, 120.0)
(From,To,Weight) =  (23, 25, 133.0)
(From,To,Weight) =  (23, 33, 290.0)
(From,To,Weight) =  (23, 31, 216.0)
(From,To,Weight) =  (24, 28, 178.0)
(From,To,Weight) =  (25, 26, 206.0)
```

```
(From,To,Weight) =  (26, 34, 247.0)
(From,To,Weight) =  (26, 28, 228.0)
(From,To,Weight) =  (29, 37, 193.0)
(From,To,Weight) =  (30, 32, 96.0)
(From,To,Weight) =  (30, 31, 114.0)
(From,To,Weight) =  (33, 41, 306.0)
(From,To,Weight) =  (34, 35, 230.0)
(From,To,Weight) =  (34, 46, 158.0)
(From,To,Weight) =  (35, 36, 211.0)
(From,To,Weight) =  (37, 39, 174.0)
(From,To,Weight) =  (38, 43, 143.0)
(From,To,Weight) =  (39, 50, 282.0)
(From,To,Weight) =  (40, 44, 162.0)
(From,To,Weight) =  (41, 45, 91.0)
(From,To,Weight) =  (42, 50, 288.0)
(From,To,Weight) =  (42, 43, 188.0)
(From,To,Weight) =  (42, 52, 232.0)
(From,To,Weight) =  (43, 51, 429.0)
(From,To,Weight) =  (44, 50, 208.0)
(From,To,Weight) =  (44, 45, 194.0)
(From,To,Weight) =  (45, 47, 342.0)
(From,To,Weight) =  (47, 49, 247.0)
(From,To,Weight) =  (47, 55, 311.0)
(From,To,Weight) =  (48, 49, 289.0)
(From,To,Weight) =  (50, 53, 414.0)
(From,To,Weight) =  (53, 54, 345.0)
(From,To,Weight) =  (54, 56, 114.0)
(From,To,Weight) =  (55, 57, 346.0)
(From,To,Weight) =  (57, 58, 320.0)
(From,To,Weight) =  (58, 62, 485.0)
(From,To,Weight) =  (59, 60, 239.0)
(From,To,Weight) =  (60, 64, 377.0)
(From,To,Weight) =  (61, 65, 409.0)
(From,To,Weight) =  (62, 63, 344.0)
(From,To,Weight) =  (63, 70, 346.0)
(From,To,Weight) =  (64, 65, 445.0)
(From,To,Weight) =  (64, 66, 341.0)
(From,To,Weight) =  (65, 68, 503.0)
(From,To,Weight) =  (66, 67, 208.0)
(From,To,Weight) =  (68, 69, 263.0)
(From,To,Weight) =  (69, 70, 452.0)
Total Cost is:  16743.0
Time Elapsed (in seconds):  0.00146565041175
```

Capítulo 6

Algoritmo de Dijkstra

6.1 Resumo

A lógica de implentação 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):  0.000132428318207
13 Number of Iterations:  6
14 Number of Relaxations:  23
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):  0.000179109300375
```

6.2.2 De 1 a 14

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

```
End Node:  14

|Original Algorithm|
Shortest Path from  1  to  14  is:  [1, 3, 8, 11, 14]
Previous Node before  14  is :  11
Shortest Path Length from  1  to  14  is:  535.0
Time Elapsed (in seconds):  0.000198973548106
Number of Iterations:  13
Number of Relaxations:  52

|Networkx Algorithm|
Shortest Path from  1  to  14  is :  [1, 3, 8, 11, 14]
Previous Node before  14  is :  11
Shortest Path Length from  1  to  14  is :  535.0
Time Elapsed (in seconds):  0.000146664362414
```

6.2.3 De 1 a 21

```
**********************************
Dijkstra's Algorithm
**********************************
Graph Name:  rede_italiana
Start Node:  1
End Node:  21

|Original Algorithm|
Shortest Path from  1  to  21  is:  [1, 3, 8, 9, 13, 15, 21]
Previous Node before  21  is :  15
Shortest Path Length from  1  to  21  is:  970.0
Time Elapsed (in seconds):  0.000293328724828
Number of Iterations:  20
Number of Relaxations:  77

|Networkx Algorithm|
Shortest Path from  1  to  21  is :  [1, 3, 8, 9, 13, 15, 21]
Previous Node before  21  is :  15
Shortest Path Length from  1  to  21  is :  970.0
Time Elapsed (in seconds):  0.000173150026055
```

6.3 Rede USA

6.3.1 De 1 a 10

```
**********************************
Dijkstra's Algorithm
**********************************
Graph Name:  rede_usa
Start Node:  1
End Node:  10

|Original Algorithm|
Shortest Path from  1  to  10  is:  [1, 4, 10]
Previous Node before  10  is :  4
```

```
Shortest Path Length from  1  to  10  is:  486.0
Time Elapsed (in seconds):  0.000142029371277
Number of Iterations:  5
Number of Relaxations:  21

|Networkx Algorithm|
Shortest Path from  1  to  10  is :  [1, 4, 10]
Previous Node before  10  is :  4
Shortest Path Length from  1  to  10  is :  486.0
Time Elapsed (in seconds):  0.000724713971386
```

6.3.2 De 1 a 20

6.3.3 De 1 a 30

```
***********************************
Dijkstra's Algorithm
***********************************
Graph Name:  rede_usa
Start Node:  1
End Node:  30

|Original Algorithm|
Shortest Path from  1  to  30  is:  [1, 2, 9, 13, 31, 30]
Previous Node before  30  is :  31
Shortest Path Length from  1  to  30  is:  1315.0
Time Elapsed (in seconds):  0.000567455343516
Number of Iterations:  24
Number of Relaxations:  151

|Networkx Algorithm|
Shortest Path from  1  to  30  is :  [1, 2, 9, 13, 31, 30]
Previous Node before  30  is :  31
Shortest Path Length from  1  to  30  is :  1315.0
Time Elapsed (in seconds):  0.000672404785695
```

6.3.4 De 1 a 40

```
***********************************
Dijkstra's Algorithm
***********************************
Graph Name:  rede_usa
Start Node:  1
End Node:  40

|Original Algorithm|
Shortest Path from  1  to  40  is:  [1, 2, 9, 13, 40]
Previous Node before  40  is :  13
Shortest Path Length from  1  to  40  is:  1603.0
Time Elapsed (in seconds):  0.00077172602435
Number of Iterations:  34
Number of Relaxations:  212

|Networkx Algorithm|
```

```
Shortest Path from  1  to  40  is :  [1, 2, 9, 13, 40]
Previous Node before  40  is :  13
Shortest Path Length from  1  to  40  is :  1603.0
Time Elapsed (in seconds):  0.000723720759
```

6.3.5 De 1 a 50

```
**********************************
Dijkstra's Algorithm
**********************************
Graph Name:  rede_usa
Start Node:  1
End Node:  50

|Original Algorithm|
Shortest Path from  1  to  50  is:  [1, 2, 9, 13, 40, 44, 50]
Previous Node before  50  is :  44
Shortest Path Length from  1  to  50  is:  1973.0
Time Elapsed (in seconds):  0.0010630683244
Number of Iterations:  47
Number of Relaxations:  295

|Networkx Algorithm|
Shortest Path from  1  to  50  is :  [1, 2, 9, 13, 40, 44, 50]
Previous Node before  50  is :  44
Shortest Path Length from  1  to  50  is :  1973.0
Time Elapsed (in seconds):  0.000675053352059
```

6.3.6 De 1 a 60

```
**********************************
Dijkstra's Algorithm
**********************************
Graph Name:  rede_usa
Start Node:  1
End Node:  60

|Original Algorithm|
Shortest Path from  1  to  60  is:  [1, 4, 10, 20, 37, 42, 52, 60]
Previous Node before  60  is :  52
Shortest Path Length from  1  to  60  is:  2964.0
Time Elapsed (in seconds):  0.00135739026162
Number of Iterations:  59
Number of Relaxations:  363

|Networkx Algorithm|
Shortest Path from  1  to  60  is :  [1, 4, 10, 20, 37, 42, 52, 60]
Previous Node before  60  is :  52
Shortest Path Length from  1  to  60  is :  2964.0
Time Elapsed (in seconds):  0.00067273585649
```

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):  0.00144876580118
13 Number of Iterations:  65
14 Number of Relaxations:  396
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):  0.00069491759979
```

Capítulo 7

Algoritmo de Ford-Moore-Bellman

7.1 Resumo

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

7.2 Rede Italiana

7.2.1 De 1 a 7

```
**********
Ford-Moore-Bellman's Algorithm
**********
Graph Name:  rede_italiana
Start Node:  1
End Node:  7

|Original Algorithm|
Shortest Path from  1  to  7  is:  [1, 2, 7]
Previous Node before  7  is :  2
Shortest Path Length from  1  to  7  is:  230.0
Time Elapsed (in seconds):  6.82005838765e-05
Number of Iterations:  40
Number of Relaxations:  40

|Networkx Algorithm|
Shortest Path from  1  to  7  is :  [1, 2, 7]
Previous Node before  7  is :  2
Shortest Path Length from  1  to  7  is :  230.0
Time Elapsed (in seconds):  0.000112564070476
```

7.2.2 De 1 a 14

```
**********
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):  7.48219997868e-05
13 Number of Iterations:  40
14 Number of Relaxations:  40
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):  0.000114550495249
```

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):  8.44230528568e-05
13 Number of Iterations:  40
14 Number of Relaxations:  40
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):  0.000118854415591
```

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
```

```
|Original Algorithm|
Shortest Path from  1  to  10  is:  [1, 4, 10]
Previous Node before  10  is :  4
Shortest Path Length from  1  to  10  is:   486.0
Time Elapsed (in seconds):  0.000351597184839
Number of Iterations:  210
Number of Relaxations:  210

|Networkx Algorithm|
Shortest Path from  1  to  10  is :  [1, 4, 10]
Previous Node before  10  is :  4
Shortest Path Length from  1  to  10  is :  486.0
Time Elapsed (in seconds):  0.00042310847667
```

7.3.2 De 1 a 20

```
**********************************
Ford-Moore-Bellman's Algorithm
**********************************
Graph Name:  rede_usa
Start Node:  1
End Node:  20

|Original Algorithm|
Shortest Path from  1  to  20  is:  [1, 4, 10, 20]
Previous Node before  20  is :  10
Shortest Path Length from  1  to  20  is:   1086.0
Time Elapsed (in seconds):  0.000314848326536
Number of Iterations:  210
Number of Relaxations:  210

|Networkx Algorithm|
Shortest Path from  1  to  20  is :  [1, 4, 10, 20]
Previous Node before  20  is :  10
Shortest Path Length from  1  to  20  is :  1086.0
Time Elapsed (in seconds):  0.000536665759533
```

7.3.3 De 1 a 30

```
**********************************
Ford-Moore-Bellman's Algorithm
**********************************
Graph Name:  rede_usa
Start Node:  1
End Node:  30

|Original Algorithm|
Shortest Path from  1  to  30  is:  [1, 4, 10, 20, 30]
Previous Node before  30  is :  20
Shortest Path Length from  1  to  30  is:   1600.0
Time Elapsed (in seconds):  0.000309551193808
Number of Iterations:  210
Number of Relaxations:  210
```

```
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):  0.000411520998827
```

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):  0.00030425406108
13 Number of Iterations:  210
14 Number of Relaxations:  210
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):  0.000410196715645
```

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):  0.000298956928352
13 Number of Iterations:  210
14 Number of Relaxations:  210
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):  0.000412845282009
```

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):  0.000287038379713
13 Number of Iterations:  210
14 Number of Relaxations:  210
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):  0.000444628078379
```

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):  0.000376427494503
13 Number of Iterations:  210
14 Number of Relaxations:  210
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):  0.000421784193488
```

Bibliografia

[1] Python Project, https://www.python.org/, Visitado em : 22 de Março de 2017

[2] Python - Wikipedia Article, https://pt.wikipedia.org/wiki/Python, Visitado em : 22 de Março de 2017

[3] Why is Python used for high-performance/scientific computing?, https://goo.gl/8aRXjN, Visitado em : 22 de Março de 2017

[4] Networkx, https://networkx.github.io/, Visitado em : 22 de Março de 2017

[5] Networkx Lastest Documentation, https://networkx.readthedocs.io/en/stable/, Visitado em : 22 de Março de 2017

[6] Depth-first Algorithm, https://en.wikipedia.org/wiki/Depth-first_search, Visitado em : 22 de Março de 2017

[7] Disjoint-set data structure, https://en.wikipedia.org/wiki/Disjoint-set_data_structure, Visitado em : 22 de Março de 2017

[8] Notas de Aula IA 881 - 2^o semestre de 2016, http://www.dt.fee.unicamp.br/ ricfow/IA881/ia881.htm, Visitado em : 22 de Março de 2017

Apêndice A

Prim

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

```
44
45          total_cost = 0
46
47          mst = nx.Graph()
48          mst.add_node(start_node)
49
50          while (nx.number_of_nodes(mst) < nx.number_of_nodes(G)):
51
52                  fringe = build_fringe(G,mst)
53                  min_edge = return_minimum_edge_fringe(fringe)
54
55                  mst.add_edge(min_edge[0],min_edge[1],weight=getWeight(min_edge))
56
57                  total_cost += getWeight(min_edge)
58                  qty_of_iterations += 1
59
60          return total_cost,mst
61
62  def print_edges(G):
63          print ""
64          for i in G.edges(data='weight'):
65                  print "(From,To,Weight) = ",i
66
67  # dictionary containing the necessary information
68  data = [
69          {
70                  'algorithm' : "prim",
71                  'graph_name' : "rede_italiana",
72                  'file' : 'redeitaliana.ncol',
73                  'start_node' : [1]
74          },
75          {
76                  'algorithm' : "prim",
77                  'graph_name' : "rede_usa",
78                  'file' : 'redeusa.ncol',
79                  'start_node' : [1]
80          }
81  ]
82
83  pasta = "output/"
84  if not os.path.exists(pasta):
85          os.makedirs(pasta)
86
87  #main loop through data dictionary defined above
88  for d in data:
89
90          G = nx.read_weighted_edgelist(d['file'],nodetype=int)
91
92          f = file( pasta + d['algorithm'] + "_" + d['graph_name'] + '.txt', 'w')
93          sys.stdout = f
94
95          for s in d['start_node']:
96
97                  #global variables
98                  time_consumed = 0
99                  qty_of_iterations = 0
100
101                 print "***********************************"
102                 print "Prim's Algorithm"
103                 print "***********************************"
104                 print "Graph Name: ",d['graph_name']
105                 print "Start Node: ",s
```

```
106                 print ""
107
108                 #run the algorithm
109                 time_consumed = 0
110                 start_time = 0
111                 start_time = timeit.default_timer()
112                 total_cost,mst = prim(G,s)
113                 time_consumed = timeit.default_timer() - start_time
114
115                 print "|Original Algorithm|"
116
117                 print "Nodes of MST is: ",mst.nodes()
118                 print "Edges of MST is: ",
119                 print_edges(mst)
120
121                 print "Total Cost is: ",total_cost
122                 print "Time Elapsed (in seconds): ", time_consumed
123                 print "Number of Iterations: ",qty_of_iterations
124                 print ""
125
126                 #run the networkx algorithm
127                 time_consumed = 0
128                 start_time = 0
129                 start_time = timeit.default_timer()
130                 T = nx.prim_mst(G)
131                 time_consumed = timeit.default_timer() - start_time
132
133                 print "|Networkx Algorithm|"
134
135                 print "Nodes of MST is: ",T.nodes()
136                 print "Edges of MST is: "
137                 print_edges(T)
138
139                 total_cost = 0
140                 for edge in T.edges(data='weight'):
141                         total_cost += getWeight(edge)
142
143                 print "Total Cost is: ",total_cost
144                 print "Time Elapsed (in seconds): ", time_consumed
145                 print ""
146
147         f.close()
```

Apêndice B

Kruskal

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

```
43                 return True
44
45     return False
46
47
48 def build_ordered_edges(g):
49     return sorted(G.edges(data=True), key=lambda (a, b, data): data['weight'])
50
51 def kruskal(G,s):
52
53     global qty_of_iterations
54
55     mst = nx.Graph()
56
57     ordered_edges = build_ordered_edges(G)
58
59     total_cost = 0
60
61     for candidate in ordered_edges:
62
63         u = candidate[0]
64         v = candidate[1]
65         weight = candidate[2]['weight']
66
67         mst.add_edge(u, v, weight=weight)
68
69         if (isCyclic(mst)):
70             mst.remove_edge(u,v)
71             total_cost -= weight
72
73         total_cost += weight
74
75         qty_of_iterations += 1
76
77
78     return total_cost,mst
79
80 def print_edges(G):
81     print ""
82     for i in G.edges(data='weight'):
83         print "(From,To,Weight) = ",i
84
85 # dictionary containing the necessary information
86 data = [
87     {
88         'algorithm' : "kruskal",
89         'graph_name' : "rede_italiana",
90         'file' : 'redeitaliana.ncol',
91         'start_node' : [1]
92     },
93     {
94         'algorithm' : "kruskal",
95         'graph_name' : "rede_usa",
96         'file' : 'redeusa.ncol',
97         'start_node' : [1]
98     }
99 ]
100
101
102 pasta = "output/"
103 if not os.path.exists(pasta):
104     os.makedirs(pasta)
```

```
105
106 #main loop through data dictionary defined above
107 for d in data:
108
109     G = nx.read_weighted_edgelist(d['file'],nodetype=int)
110
111     f = file( pasta + d['algorithm'] + "_" + d['graph_name'] + '.txt', 'w')
112     sys.stdout = f
113
114     for s in d['start_node']:
115
116         #global variables
117         time_consumed = 0
118         qty_of_iterations = 0
119
120         print "************************************"
121         print "Kruskal's Algorithm"
122         print "************************************"
123         print "Graph Name: ",d['graph_name']
124         print "Start Node: ",s
125         print ""
126
127         #run the algorithm
128         time_consumed = 0
129         start_time = 0
130         start_time = timeit.default_timer()
131         total_cost,mst = kruskal(G,s)
132         time_consumed = timeit.default_timer() - start_time
133
134         print "|Original Algorithm|"
135
136         print "Nodes of MST is: ",mst.nodes()
137         print "Edges of MST is: "
138         print_edges(mst)
139         print "Total Cost is: ",total_cost
140         print "Time Elapsed (in seconds): ", time_consumed
141         print "Number of Iterations: ",qty_of_iterations
142         print ""
143
144
145         #run the networkx algorithm
146         time_consumed = 0
147         start_time = 0
148         start_time = timeit.default_timer()
149         #it's kruskal algorithm
150         T = nx.minimum_spanning_tree(G)
151         time_consumed = timeit.default_timer() - start_time
152
153         print "|Networkx Algorithm|"
154
155         print "Nodes of MST is: ",T.nodes()
156         print "Edges of MST is: "
157         print_edges(T)
158
159         total_cost = 0
160         for edge in T.edges(data='weight'):
161             total_cost += edge[2]
162
163         print "Total Cost is: ",total_cost
164         print "Time Elapsed (in seconds): ", time_consumed
165         print ""
166
```

```
167         f.close()
```

Apêndice C

Dijkstra

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

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

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

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

Apêndice D

Ford-Moore-Bellman

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

```
45
46  # dictionary containing the necessary information
47  data = [
48          {
49                  'algorithm' : "ford_moore_bellman",
50                  'graph_name' : "rede_italiana",
51                  'file' : 'redeitaliana.ncol',
52                  'start_node' : [1],
53                  'end_node' : [7,14,21]
54          },
55          {
56                  'algorithm' : "ford_moore_bellman",
57                  'graph_name' : "rede_usa",
58                  'file' : 'redeusa.ncol',
59                  'start_node' : [1],
60                  'end_node' : [10,20,30,40,50,60,70]
61          }
62  ]
63
64  pasta = "output/"
65  if not os.path.exists(pasta):
66          os.makedirs(pasta)
67
68
69  #main loop through data dictionary defined above
70  for d in data:
71          G = nx.read_weighted_edgelist(d['file'],nodetype=int)
72
73          #start node
74          start_node = d['start_node']
75
76          #ending nodes
77          end_node = d['end_node']
78
79          for s in start_node:
80                  for e in end_node:
81                          f = file( pasta + d['algorithm'] + "_" + d['graph_name']
                             ↪  + "_" +
82                          str(s) + "_" + str(e) + ".txt", 'w' )
83                          sys.stdout = f
84
85                          print "***********************************"
86                          print "Ford-Moore-Bellman's Algorithm"
87
88                          #global variables
89                          qty_of_relaxations = 0
90                          qty_of_iterations = 0
91                          time_consumed = 0
92
93                          prev = {}
94                          dist = {}
95                          for i in G.nodes():
96                                  #stores the last node between s and v
97                                  prev[i] = None
98                                  #stores the minimun path length between s and v.
99                                  dist[i] = float('inf')
100
101                         # initial node distance is zero
102                         dist[s] = 0
103
104                         print "***********************************"
105                         print "Graph Name: ",d['graph_name']
```

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