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

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

```
************************************
Prim'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):  0.00833308740875
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):  0.00051966099967
```

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):  0.181805426259
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):  0.00226027048127
```

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

```
***********************************
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):  0.00445373890321
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):  0.000456078316695
```

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):  0.0630622171308
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):  0.00176073058381
```

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

```
************************************
Dijkstra'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):  0.000127165365949
Number of Iterations:  6
Number of Relaxations:  10

|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):  0.000368618592435
```

6.2.2 De 1 a 14

```
************************************
Dijkstra's Algorithm
************************************
Graph Name:  rede_italiana
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):  0.000266671674585
Number of Iterations:  13
Number of Relaxations:  18

|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):  0.000167944133089
```

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):  0.000479150513893
Number of Iterations:  20
Number of Relaxations:  23

|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):  0.000195040550728
```

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):  0.000184309296218
Number of Iterations:  5
Number of Relaxations:  11

|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):  0.0013832587064
```

6.3.2 De 1 a 20

```
************************************
Dijkstra'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):  0.000575731804487
Number of Iterations:  18
Number of Relaxations:  34

|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):  0.000922351325176
```

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):  0.00154530064951
Number of Iterations:  24
Number of Relaxations:  48

|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):  0.00119412034565
```

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):  0.0033572729736
Number of Iterations:  34
Number of Relaxations:  60

|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):  0.000997470106749
```

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):  0.00145220701663
Number of Iterations:  47
Number of Relaxations:  77

|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):  0.000924497576078
```

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):  0.0021736156011
Number of Iterations:  59
Number of Relaxations:  90

|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):  0.00119385206429
```

6.3.7 De 1 a 70

```
************************************
Dijkstra's Algorithm
************************************
Graph Name:  rede_usa
Start Node:  1
End Node:  70

|Original Algorithm|
Shortest Path from  1  to  70  is:  [1, 3, 23, 33, 47, 55, 57, 70]
Previous Node before  70  is :  57
Shortest Path Length from  1  to  70  is:  3429.0
Time Elapsed (in seconds):  0.00464636492168
Number of Iterations:  65
Number of Relaxations:  96

|Networkx Algorithm|
Shortest Path from  1  to  70  is :  [1, 3, 23, 33, 47, 55, 57, 70]
Previous Node before  70  is :  57
Shortest Path Length from  1  to  70  is :  3429.0
Time Elapsed (in seconds):  0.00102563964984
```

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):  0.00121933879375
Number of Iterations:  40
Number of Relaxations:  24

|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):  0.000209259462954
```

7.2.2 De 1 a 14

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

```
***********************************
Graph Name:  rede_italiana
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):  8.26306597307e-05
Number of Iterations:  40
Number of Relaxations:  24

|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):  0.000125555677773
```

7.2.3 De 1 a 21

```
***********************************
Ford-Moore-Bellman'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):  9.68495719571e-05
Number of Iterations:  40
Number of Relaxations:  24

|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):  0.000138164901822
```

7.3 Rede USA

7.3.1 De 1 a 10

```
***********************************
Ford-Moore-Bellman'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):  0.000444005655371
Number of Iterations:  210
Number of Relaxations:  100

|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):  0.000552391325927
```

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):  0.000457151442146
Number of Iterations:  210
Number of Relaxations:  100

|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):  0.000539782101877
```

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):  0.000370228280612
Number of Iterations:  210
Number of Relaxations:  100
```

```

|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):  0.000482906452972
```

7.3.4 De 1 a 40

```
************************************
Ford-Moore-Bellman'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):  0.000309328411265
Number of Iterations:  210
Number of Relaxations:  100

|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):  0.000489881768403
```

7.3.5 De 1 a 50

```
************************************
Ford-Moore-Bellman'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):  0.000394641884623
Number of Iterations:  210
Number of Relaxations:  100

|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):  0.000557220390457
```

7.3.6 De 1 a 60

```
***********************************
Ford-Moore-Bellman'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):  0.000383105786024
Number of Iterations:  210
Number of Relaxations:  100

|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):  0.00051483193514
```

7.3.7 De 1 a 70

```
***********************************
Ford-Moore-Bellman's Algorithm
***********************************
Graph Name:  rede_usa
Start Node:  1
End Node:  70

|Original Algorithm|
Shortest Path from  1  to  70  is:  [1, 3, 23, 33, 47, 55, 57, 70]
Previous Node before  70  is :  57
Shortest Path Length from  1  to  70  is:  3429.0
Time Elapsed (in seconds):  0.0003710331247
Number of Iterations:  210
Number of Relaxations:  100

|Networkx Algorithm|
Shortest Path from  1  to  70  is :  [1, 3, 23, 33, 47, 55, 57, 70]
Previous Node before  70  is :  57
Shortest Path Length from  1  to  70  is :  3429.0
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, Visitado em : 22 de Novembro de 2016

[7] 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 - 2^o semestre de 2016, http://www.dt.fee.unicamp.br/ ricfow/IA881/ia881.htm, Visitado em : 22 de Novembro de 2016

Apêndice A

Prim

```
# see documentation at : https://networkx.github.io/
import networkx as nx

# see documentation at: https://docs.python.org/2/library/timeit.html
import timeit

# see documentation at: https://docs.python.org/2/library/itertools.html
import itertools

# see documentation at: https://docs.python.org/2/library/sys.html
import sys

def getWeight(k):
        return k[2]

def return_minimum_edge_fringe(fringe):
        ordered_edges_by_weight =
         ↪  sorted(fringe.edges(data='weight'),key=getWeight)
        return ordered_edges_by_weight[0]

def build_fringe(g,mst):

        fringe = nx.Graph()

        fringe_nodes = set(nx.nodes(g)).difference(set(nx.nodes(mst)))

        fringe.add_nodes_from(set(nx.nodes(g)).difference(set(nx.nodes(mst))))

        for node in itertools.product(nx.nodes(mst),nx.nodes(fringe)):
                if g.has_edge(*node) or g.has_edge(*tuple(reversed(node))):
                        fringe.add_edge(*node,weight=g.get_edge_data(*node)['weight'])

        for i in nx.nodes(fringe):
                if fringe.degree(i) == 0:
                        fringe.remove_node(i)

        return fringe


def prim(G,start_node):

        global qty_of_iterations

        total_cost = 0
```

```python

	mst = nx.Graph()
	mst.add_node(start_node)

	while (nx.number_of_nodes(mst) < nx.number_of_nodes(G)):

		fringe = build_fringe(G,mst)
		min_edge = return_minimum_edge_fringe(fringe)

		mst.add_edge(min_edge[0],min_edge[1],weight=getWeight(min_edge))

		total_cost += getWeight(min_edge)
		qty_of_iterations += 1

	return total_cost,mst

def print_edges(G):
	print ""
	for i in G.edges(data='weight'):
		print "(From,To,Weight) = ",i

# dictionary containing the necessary information
data = [
	{
		'algorithm' : "prim",
		'graph_name' : "rede_italiana",
		'file' : 'redeitaliana.ncol',
		'start_node' : [1]
	},
	{
		'algorithm' : "prim",
		'graph_name' : "rede_usa",
		'file' : 'redeusa.ncol',
		'start_node' : [1]
	}
]


#main loop through data dictionary defined above
for d in data:

	G = nx.read_weighted_edgelist(d['file'],nodetype=int)

	f = file( d['algorithm'] + "_" + d['graph_name'] + '.txt', 'w')
	sys.stdout = f

	for s in d['start_node']:

		#global variables
		time_consumed = 0
		qty_of_iterations = 0

		print "************************************"
		print "Prim's Algorithm"
		print "************************************"
		print "Graph Name: ",d['graph_name']
		print "Start Node: ",s
		print ""

		#run the algorithm
		time_consumed = 0
		start_time = 0
```

```
            start_time = timeit.default_timer()
            total_cost,mst = prim(G,s)
            time_consumed = timeit.default_timer() - start_time

            print "|Original Algorithm|"

            print "Nodes of MST is: ",mst.nodes()
            print "Edges of MST is: ",
            print_edges(mst)

            print "Total Cost is: ",total_cost
            print "Time Elapsed (in seconds): ", time_consumed
            print "Number of Iterations: ",qty_of_iterations
            print ""

            #run the networkx algorithm
            time_consumed = 0
            start_time = 0
            start_time = timeit.default_timer()
            T = nx.prim_mst(G)
            time_consumed = timeit.default_timer() - start_time

            print "|Networkx Algorithm|"

            print "Nodes of MST is: ",T.nodes()
            print "Edges of MST is: "
            print_edges(T)

            total_cost = 0
            for edge in T.edges(data='weight'):
                    total_cost += getWeight(edge)

            print "Total Cost is: ",total_cost
            print "Time Elapsed (in seconds): ", time_consumed
            print ""

    f.close()
```

Apêndice B

Kruskal

```python
# see documentation at : https://networkx.github.io/
import networkx as nx

# see documentation at: https://docs.python.org/2/library/timeit.html
import timeit

# see documentation at: https://docs.python.org/2/library/sys.html
import sys

def isCyclicUtil(G, v, visited, parent):

    # Mark the current node as visited
    visited[v] = True

    # Search for all the vertices adjacent to this vertex
    for edge in G.edges(v):
        i = edge[1]

        # If the node is not visited then recurse on it
        if visited[i] == False:
            if (isCyclicUtil(G, i, visited, v)):
                return True
        # If an adjacent vertex is visited and not parent of current vertex, is a cycle!
        elif parent != i:
            return True

    return False


def isCyclic(G):

    # Mark all the vertices as not visited
    visited = {}
    for i in G.nodes():
        visited[i] = False

    # Call the recursive helper function to detect cycle in different DFS trees
    for i in G.nodes():
        if visited[i] == False:  # Don't recur for u if it is already visited
            if (isCyclicUtil(G, i, visited, -1)) == True:
                return True

```

```
    return False


def build_ordered_edges(g):
    return sorted(G.edges(data=True), key=lambda (a, b, data): data['weight'])

def kruskal(G,s):

    global qty_of_iterations

    mst = nx.Graph()

    ordered_edges = build_ordered_edges(G)

    total_cost = 0

    for candidate in ordered_edges:

        u = candidate[0]
        v = candidate[1]
        weight = candidate[2]['weight']

        mst.add_edge(u, v, weight=weight)

        if (isCyclic(mst)):
            mst.remove_edge(u,v)
            total_cost -= weight

        total_cost += weight

        qty_of_iterations += 1


    return total_cost,mst

def print_edges(G):
    print ""
    for i in G.edges(data='weight'):
        print "(From,To,Weight) = ",i

# dictionary containing the necessary information
data = [
    {
        'algorithm' : "kruskal",
        'graph_name' : "rede_italiana",
        'file' : 'redeitaliana.ncol',
        'start_node' : [1]
    },
    {
        'algorithm' : "kruskal",
        'graph_name' : "rede_usa",
        'file' : 'redeusa.ncol',
        'start_node' : [1]
    }
]

#main loop through data dictionary defined above
for d in data:

    G = nx.read_weighted_edgelist(d['file'],nodetype=int)

    f = file( d['algorithm'] + "_" + d['graph_name'] + '.txt', 'w')
```

```
    sys.stdout = f

    for s in d['start_node']:

        #global variables
        time_consumed = 0
        qty_of_iterations = 0

        print "************************************"
        print "Kruskal's Algorithm"
        print "************************************"
        print "Graph Name: ",d['graph_name']
        print "Start Node: ",s
        print ""

        #run the algorithm
        time_consumed = 0
        start_time = 0
        start_time = timeit.default_timer()
        total_cost,mst = kruskal(G,s)
        time_consumed = timeit.default_timer() - start_time

        print "|Original Algorithm|"

        print "Nodes of MST is: ",mst.nodes()
        print "Edges of MST is: "
        print_edges(mst)
        print "Total Cost is: ",total_cost
        print "Time Elapsed (in seconds): ", time_consumed
        print "Number of Iterations: ",qty_of_iterations
        print ""


        #run the networkx algorithm
        time_consumed = 0
        start_time = 0
        start_time = timeit.default_timer()
        #it's kruskal algorithm
        T = nx.minimum_spanning_tree(G)
        time_consumed = timeit.default_timer() - start_time

        print "|Networkx Algorithm|"

        print "Nodes of MST is: ",T.nodes()
        print "Edges of MST is: "
        print_edges(T)

        total_cost = 0
        for edge in T.edges(data='weight'):
            total_cost += edge[2]

        print "Total Cost is: ",total_cost
        print "Time Elapsed (in seconds): ", time_consumed
        print ""

    f.close()
```

Apêndice C

Dijkstra

```python
# see documentation at : https://networkx.github.io/
import networkx as nx

# see documentation at: https://docs.python.org/2/library/timeit.html
import timeit

# see documentation at: https://docs.python.org/2/library/sys.html
import sys

#gets the minimum node based on the visited nodes
def select_minimum_node(dist,visited_nodes):

    minimum_value = float("inf")

    minimum_node = 0

    for i in visited_nodes:
        if (visited_nodes[i] == 0):
            if (dist[i] < minimum_value):
                minimum_node = i
                minimum_value = dist[i]

    return minimum_node

#relax the edge
def relax_edge(prev,dist,edge):

    global qty_of_relaxations

    u = edge[0]
    v = edge[1]
    weight = edge[2]

    if (dist[v] > dist[u] + weight):
        dist[v] = dist[u] + weight
        prev[v] = u

        qty_of_relaxations = qty_of_relaxations + 1


#check if unvisited nodes still exist
def unvisited_nodes_exist(visited_nodes):

    for i in visited_nodes:
```

```
        if (visited_nodes[i] == 0):
            return True

    return False

#build list to show the path to reach the end node
def get_minimum_path(prev,end_node):

    min_path = []

    u = end_node
    while prev[u] is not None:
        min_path = [u] + min_path
        u = prev[u]

    min_path = [u] + min_path
    return min_path

# dijkstra algorithm
def dijkstra(G,prev,dist,visited_nodes,start_node,end_node):

    global time_consumed
    global qty_of_iterations

    #the minimum node is the start node
    minimum_node = start_node

    start_time = timeit.default_timer()

    while (unvisited_nodes_exist(visited_nodes)):

        minimum_node = select_minimum_node(dist,visited_nodes)

        if minimum_node == end_node:
            break

        for edge in G.edges(minimum_node,data='weight'):
            relax_edge(prev,dist,edge)

        visited_nodes[minimum_node] = 1

        qty_of_iterations = qty_of_iterations + 1

    time_consumed = timeit.default_timer() - start_time

# dictionary containing the necessary information
data = [
    {
        'algorithm' : "dijkstra",
        'graph_name' : "rede_italiana",
        'file' : 'redeitaliana.ncol',
        'start_node' : [1],
        'end_node' : [7,14,21]
    },
    {
        'algorithm' : "dijkstra",
        'graph_name' : "rede_usa",
        'file' : 'redeusa.ncol',
        'start_node' : [1],
        'end_node' : [10,20,30,40,50,60,70]
    }
]
```

```python


#main loop through data dictionary defined above
for d in data:

    G = nx.read_weighted_edgelist(d['file'],nodetype=int)

    #start node
    start_node = d['start_node']

    #ending nodes
    end_node = d['end_node']

    for s in start_node:
        for e in end_node:

            f = file( d['algorithm'] + "_" + d['graph_name'] + "_" + str(s) + "_" + str(e) + ".txt", 'w' )
            sys.stdout = f

            print "***********************************"
            print "Dijkstra's Algorithm"

            #global variables
            qty_of_relaxations = 0
            qty_of_iterations = 0
            time_consumed = 0

            prev = {}
            dist = {}
            visited_nodes = {}
            for i in G.nodes():
                #stores the last node between s and v
                prev[i] = None
                #stores the minimun path length between s and v.
                dist[i] = float('inf')
                #visited nodes : 1 is visited and 0 is unvisited
                visited_nodes[i] = 0

            # initial node distance is zero
            dist[s] = 0

            print "***********************************"
            print "Graph Name: ",d['graph_name']
            print "Start Node: ",s
            print "End Node: ",e
            print ""

            #run the algorithm
            time_consumed = 0
            start_time = 0
            start_time = timeit.default_timer()
            dijkstra(G,prev,dist,visited_nodes,s,e)
            time_consumed = timeit.default_timer() - start_time

            print "|Original Algorithm|"
            print "Shortest Path from ",s," to ",e," is: ",get_minimum_path(prev,e)
            print "Previous Node before ",e," is : ",prev[e]
            print "Shortest Path Length from ",s," to ",e," is: ",dist[e]
            print "Time Elapsed (in seconds): ", time_consumed
            print "Number of Iterations: ",qty_of_iterations
```

```
            print "Number of Relaxations: ",qty_of_relaxations
            print ""

            #run networkx algorithm
            time_consumed = 0
            start_time = 0
            start_time = timeit.default_timer()
            distance,path = nx.single_source_dijkstra(G, s)
            time_consumed = timeit.default_timer() - start_time

            print "|Networkx Algorithm|"
            print "Shortest Path from ",s," to ",e," is : ",path[e]
            print "Previous Node before ",e," is : ",path[e][len(path[e]) - 2]
            print "Shortest Path Length from ",s," to ",e," is : ",distance[e]
            print "Time Elapsed (in seconds): ", time_consumed
            print ""

            f.close()
```

Apêndice D

Ford-Moore-Bellman

```
# see documentation at : https://networkx.github.io/
import networkx as nx

# see documentation at: https://docs.python.org/2/library/timeit.html
import timeit

# see documentation at: https://docs.python.org/2/library/sys.html
import sys

#relax the edge
def relax_edge(prev,dist,edge):

	global qty_of_relaxations

	u = edge[0]
	v = edge[1]
	weight = edge[2]

	if (dist[v] > dist[u] + weight):
		dist[v] = dist[u] + weight
		prev[v] = u

		qty_of_relaxations = qty_of_relaxations + 1

#build list to show the path to reach the end node
def get_minimum_path(prev,end_node):

	min_path = []

	u = end_node
	while prev[u] is not None:
		min_path = [u] + min_path
		u = prev[u]

	min_path = [u] + min_path
	return min_path

# ford_moore_bellman algorithm
def ford_moore_bellman(G,prev,dist):

	global time_consumed
	global qty_of_iterations

	for edge in G.edges(data='weight'):
```

```
                relax_edge(prev,dist,edge)
                qty_of_iterations = qty_of_iterations + 1



# dictionary containing the necessary information
data = [
        {
                'algorithm' : "ford_moore_bellman",
                'graph_name' : "rede_italiana",
                'file' : 'redeitaliana.ncol',
                'start_node' : [1],
                'end_node' : [7,14,21]
        },
        {
                'algorithm' : "ford_moore_bellman",
                'graph_name' : "rede_usa",
                'file' : 'redeusa.ncol',
                'start_node' : [1],
                'end_node' : [10,20,30,40,50,60,70]
        }
]


#main loop through data dictionary defined above
for d in data:

        G = nx.read_weighted_edgelist(d['file'],nodetype=int)

        #start node
        start_node = d['start_node']

        #ending nodes
        end_node = d['end_node']

        for s in start_node:
                for e in end_node:

                        f = file( d['algorithm'] + "_" + d['graph_name'] + "_" +
                         ↪  str(s) + "_" + str(e) + ".txt", 'w' )
                        sys.stdout = f

                        print "*************************************"
                        print "Ford-Moore-Bellman's Algorithm"

                        #global variables
                        qty_of_relaxations = 0
                        qty_of_iterations = 0
                        time_consumed = 0

                        prev = {}
                        dist = {}
                        for i in G.nodes():
                                #stores the last node between s and v
                                prev[i] = None
                                #stores the minimun path length between s and v.
                                dist[i] = float('inf')

                        # initial node distance is zero
                        dist[s] = 0

                        print "*************************************"
```

```python
print "Graph Name: ",d['graph_name']
print "Start Node: ",s
print "End Node: ",e
print ""

#run the algorithm
time_consumed = 0
start_time = 0
start_time = timeit.default_timer()
ford_moore_bellman(G,prev,dist)
time_consumed = timeit.default_timer() - start_time

print "|Original Algorithm|"
print "Shortest Path from ",s," to ",e," is: ",get_minimum_path(prev,e)
print "Previous Node before ",e," is : ",prev[e]
print "Shortest Path Length from ",s," to ",e," is: ",dist[e]
print "Time Elapsed (in seconds): ", time_consumed
print "Number of Iterations: ",qty_of_iterations
print "Number of Relaxations: ",qty_of_relaxations
print ""

#run networkx algorithm
time_consumed = 0
start_time = 0
start_time = timeit.default_timer()
predecessor, distance = nx.bellman_ford(G, s)
time_consumed = timeit.default_timer() - start_time

print "|Networkx Algorithm|"
print "Shortest Path from ",s," to ",e," is : ",get_minimum_path(predecessor,e)
print "Previous Node before ",e," is : ",predecessor[e]
print "Shortest Path Length from ",s," to ",e," is : ",distance[e]
print "Time Elapsed (in seconds): ", time_consumed
print ""

f.close()
```