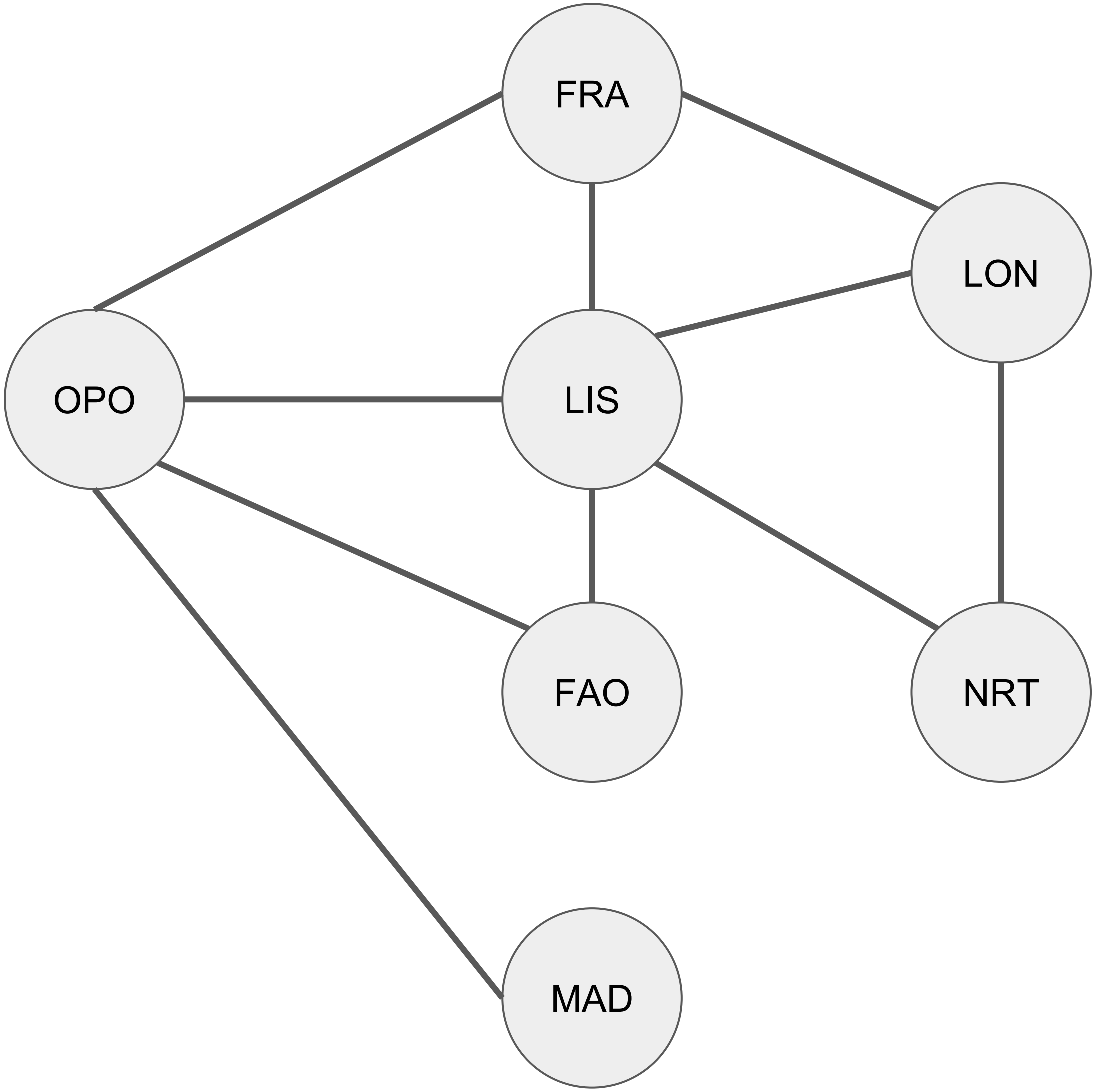
Laboratórios de Algoritmia 2

Teórica 2


FRA
LON
OPO
LIS
FAO
NRT
MAD

Representar grafos não orientados e não pesados

```
OPO LIS
OPO FAO
LIS FAO
MAD OPO
LIS LON
FRA OPO
LIS NRT
LON NRT
LON FRA
LIS FRA
```

```python
import sys

def parse(adj):
    for l in sys.stdin:
        o,d = l.split()
        if o not in adj:
            adj[o] = []
        if d not in adj:
            adj[d] = []
        adj[o].append(d)
        adj[d].append(o)

adj = {}
parse(adj)
```

Travessia em profundidade

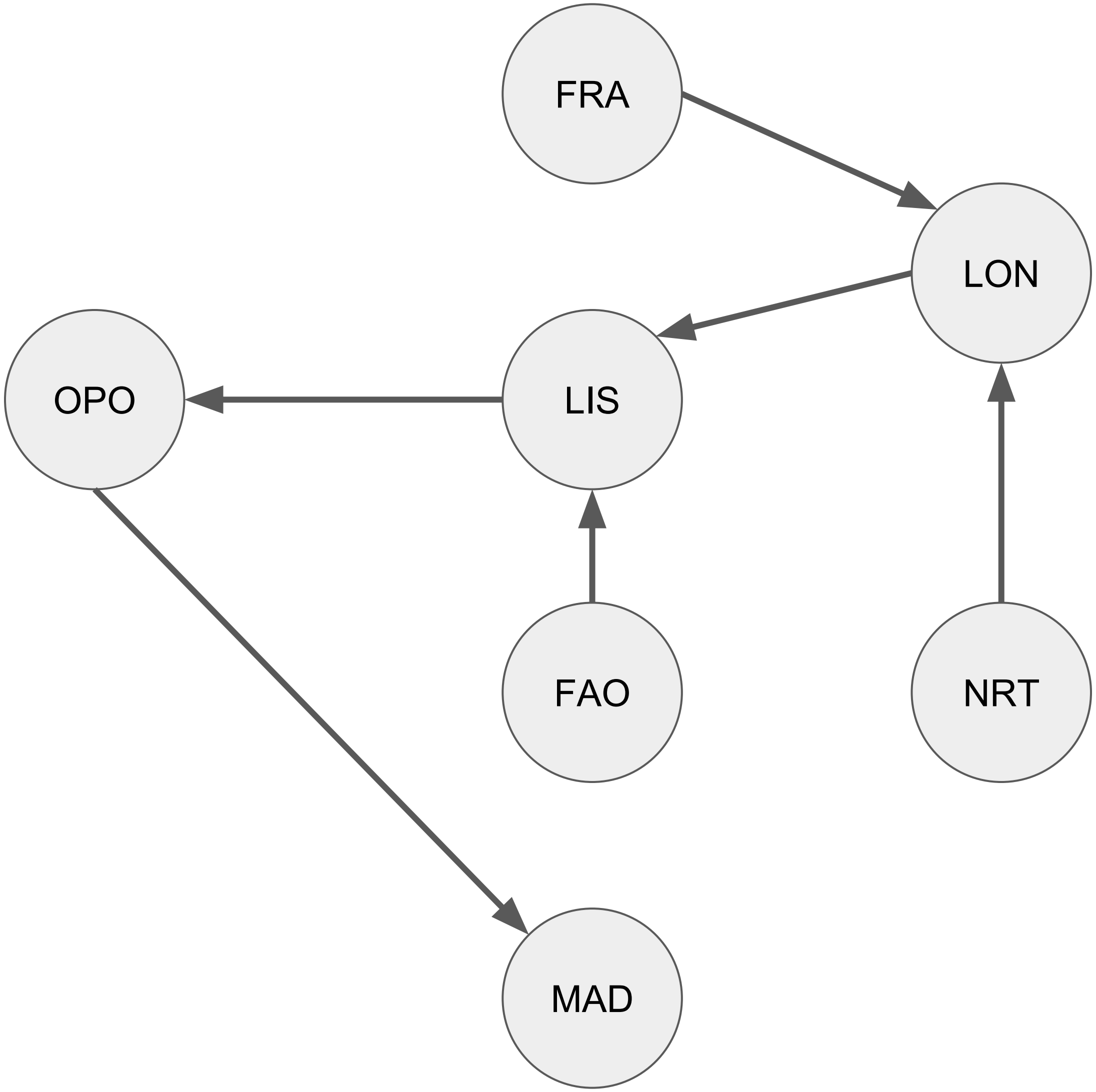
```
MAD
OPO LIS
OPO FAO
LIS FAO
MAD OPO
LIS LON
FRA OPO
LIS NRT
LON NRT
LON FRA
LIS FRA
```

```python
import sys

def dfs_aux(adj,o,discovered,parent):
    discovered.append(o)
    for d in adj[o]:
        if d not in discovered:
            parent[d] = o
            dfs_aux(adj,d,discovered,parent)
    return parent

def dfs(adj,o):
    return dfs_aux(adj,o,[],{})

origem = sys.stdin.readline().split()[0]
adj = {}
parse(adj)
print(dfs(adj,origem))
```


FRA
LON
OPO
LIS
FAO
NRT
MAD

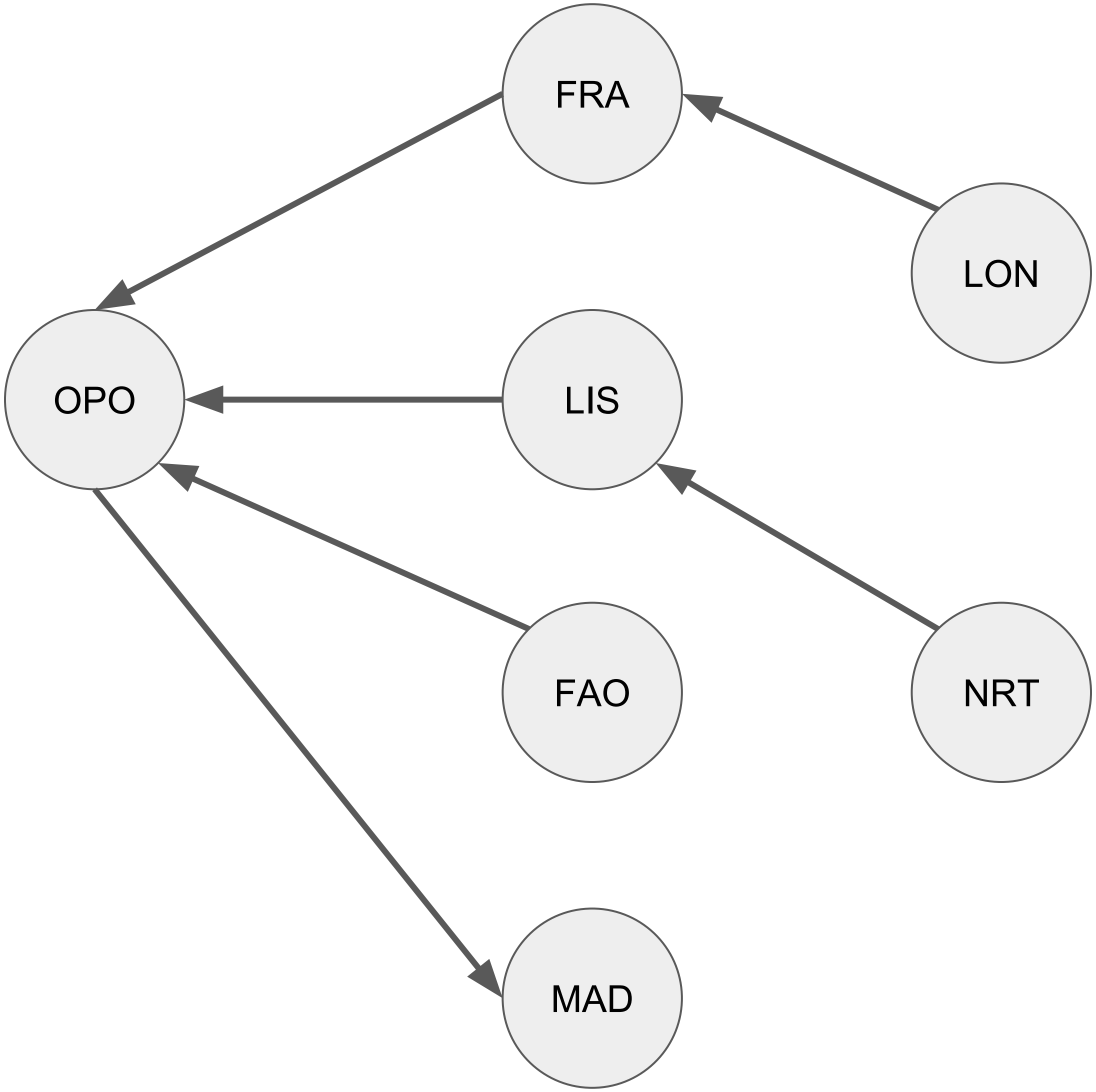
Travessia por níveis

```
MAD
OPO LIS
OPO FAO
LIS FAO
MAD OPO
LIS LON
FRA OPO
LIS NRT
LON NRT
LON FRA
LIS FRA
```

```python
import sys

def bfs(adj,o):
    parent = {}
    discovered = []
    queue = []
    discovered.append(o)
    queue.append(o)
    while queue:
        c = queue.pop(0)
        for n in adj[c]:
            if n not in discovered:
                discovered.append(n)
                parent[n] = c
                queue.append(n)
    return parent

origem = sys.stdin.readline().split()[0]
adj = {}
parse(adj)
print(bfs(adj,origem))
```

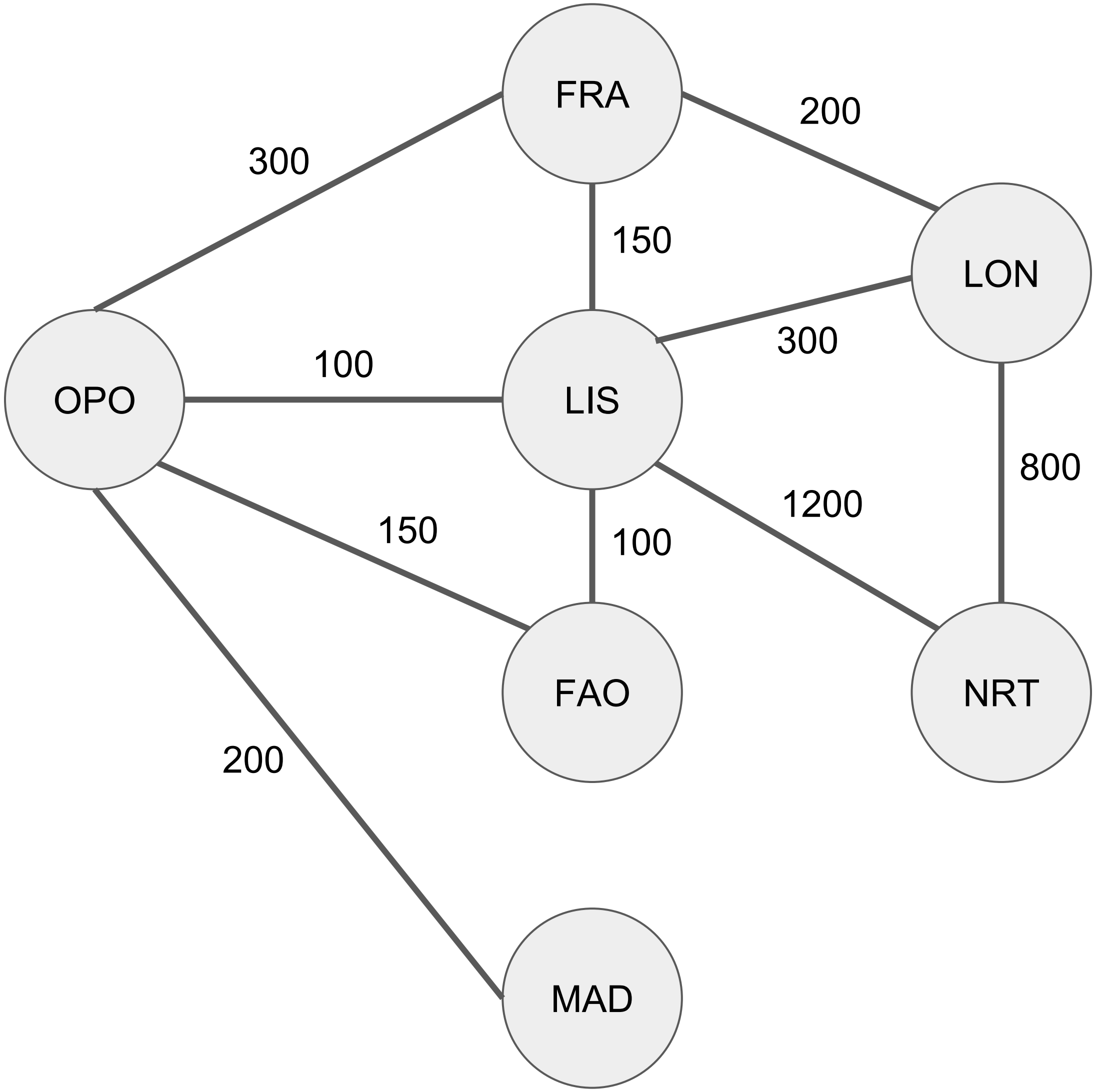

FRA
LON
OPO
LIS
FAO
NRT
MAD

Caminho mais curto

```
MAD NRT
OPO LIS
OPO FAO
LIS FAO
MAD OPO
LIS LON
FRA OPO
LIS NRT
LON NRT
LON FRA
LIS FRA
```

```
import sys

origem,destino = sys.stdin.readline().split()
adj = {}
parse(adj)
parent = bfs(adj,origem)
path = []
path.append(destino)
while destino in parent:
    destino = parent[destino]
    path.insert(0,destino)
print(path)
```


FRA
300
200
150
LON
300
OPO
100
LIS
800
1200
150
100
FAO
NRT
200
MAD

Representar grafos não orientados e pesados

```
OPO LIS 100
OPO FAO 70
LIS FAO 100
MAD OPO 200
LIS LON 300
FRA OPO 300
LIS NRT 1200
LON NRT 800
LON FRA 200
LIS FRA 300
```

```
import sys

def parse(adj):
    for l in sys.stdin:
        o,d,w = l.split()
        if o not in adj:
            adj[o] = []
        if d not in adj:
            adj[d] = []
        adj[o].append((d,int(w)))
        adj[d].append((o,int(w)))

adj = {}
parse(adj)
```

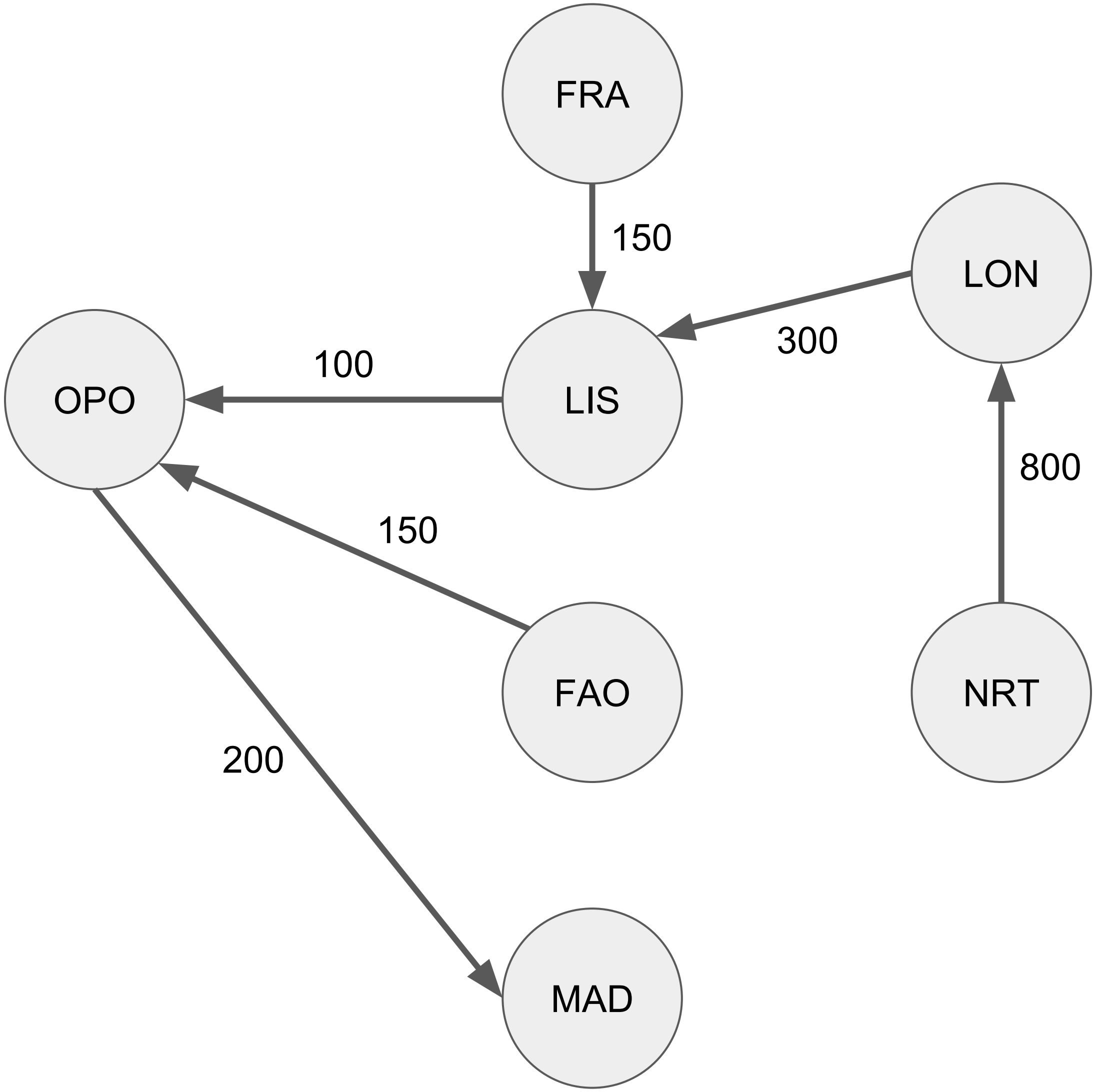
Algoritmo de Dijkstra

```
MAD
OPO LIS
OPO FAO
LIS FAO
MAD OPO
LIS LON
FRA OPO
LIS NRT
LON NRT
LON FRA
LIS FRA
```

```
import sys

def dijkstra(adj,o):
    queue = []
    parent = {}
    dist = {}
    for v in adj:
        dist[v] = float(“inf”)
        queue.append(v)
    dist[o] = 0
    while queue:
        u = min(queue,key=lambda x : dist[x])
        queue.remove(u)
        for (v,w) in adj[u]:
            alt = dist[u] + w
            if alt < dist[v]:
                dist[v] = alt
                parent[v] = u
    return parent,dist

origem = sys.stdin.readline().split()[0]
adj = {}
parse(adj)
print(dijkstra(adj,origem))
```


FRA
150
LON
300
100
OPO
LIS
800
150
FAO
NRT
200
MAD

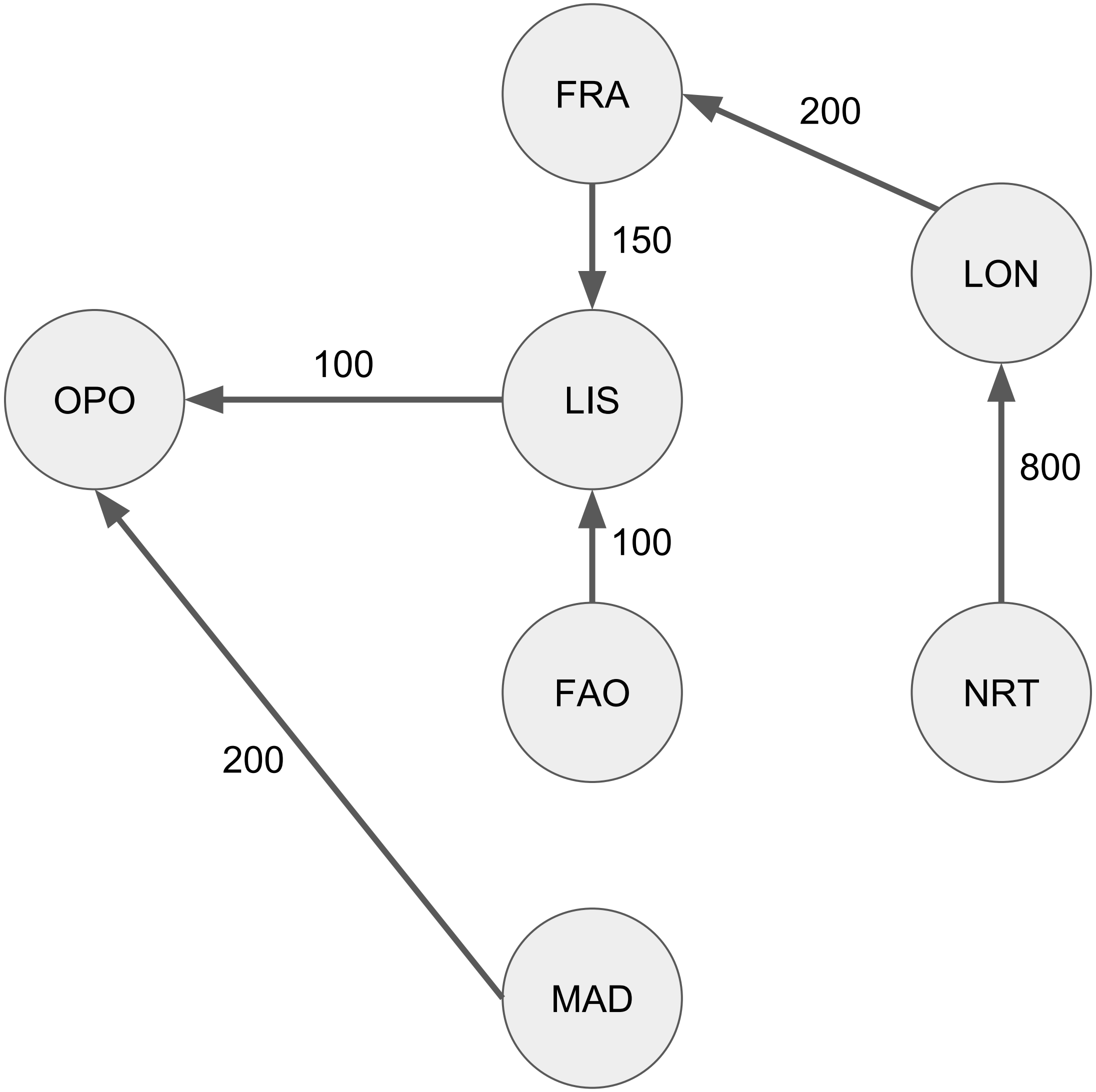
Algoritmo de Prim

```
OPO LIS
OPO FAO
LIS FAO
MAD OPO
LIS LON
FRA OPO
LIS NRT
LON NRT
LON FRA
LIS FRA
```

```
import sys

def prim(adj):
    queue = []
    parent = {}
    cost = {}
    for v in adj:
        cost[v] = float(“inf”)
        queue.append(v)
    cost[list(adj)[0]] = 0
    while queue:
        u = min(queue,key=lambda x : cost[x])
        queue.remove(u)
        for (v,w) in adj[u]:
            if w < cost[v]:
                cost[v] = w
                parent[v] = u
    return parent,cost

adj = {}
parse(adj)
print(prim(adj))
```


FRA
200
150
LON
OPO
100
LIS
800
100
FAO
NRT
200
MAD

Algoritmo de Floyd-Warshall

```
OPO LIS
OPO FAO
LIS FAO
MAD OPO
LIS LON
FRA OPO
LIS NRT
LON NRT
LON FRA
LIS FRA
```

```python
import sys

def fw(adj):
    dist = {}
    for o in adj:
        dist[o] = {}
        for d in adj:
            if o == d:
                dist[o][d] = 0
            else:
                dist[o][d] = float("inf")
        for (d,w) in adj[o]:
                dist[o][d] = w
    for k in adj:
        for o in adj:
            for d in adj:
                if dist[o][d] > dist[o][k] + dist[k][d]:
                    dist[o][d] = dist[o][k] + dist[k][d]
    return dist

adj = {}
parse(adj)
print(fw(adj))
```

	OPO	LIS	FAO	MAD	LON	FRA	NRT
OPO	0	100	150	200	400	250	1200
LIS	100	0	100	300	300	150	1100
FAO	150	100	0	350	400	250	1200
MAD	200	300	350	0	600	450	1400
LON	400	300	400	600	0	200	800
FRA	250	150	250	450	200	0	1000
NRT	1200	1100	1200	1400	800	1000	0