INSTITUTO SUPERIOR TÉCNICO

# Introdução aos Algoritmos e Estruturas de Dados

**2015/2016 - 2º semestre**

Enunciado do 1º Projecto

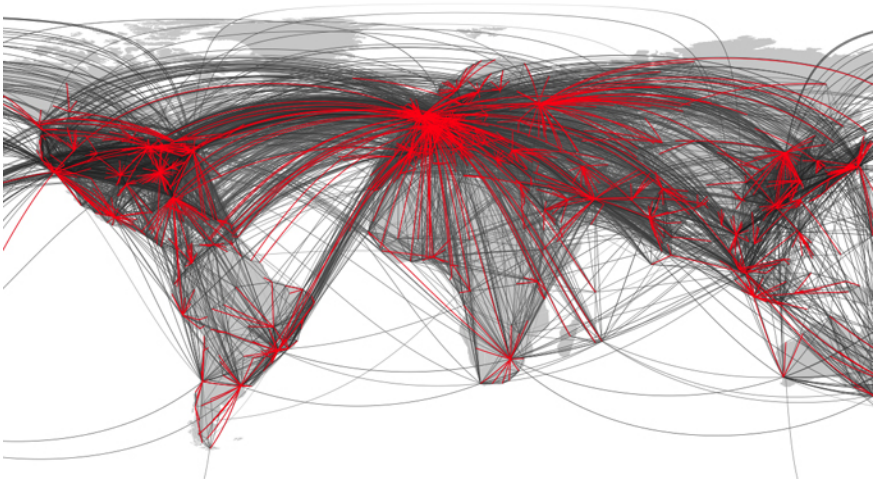
Data de entrega: 2 de Abril de 2016 (11h59am)

## 1. Introdução

Nos nossos dias, os sistemas de gestão e monitorização de tráfego são fundamentais nas áreas de segurança nacional, gestão de calamidades, gestão de impacto ambiental, planeamento de redes de transportes, entre outras.

Uma forma de monitorizar tráfego é através de um grafo (ou rede) em que cada nó (ou vértice) corresponde a um ponto de interesse de estudo, e cada aresta entre dois nós, uma possível ligação entre esses dois pontos. Esta representação permite não só estudar a robustez de uma rede, mas também a distribuição e redundância da mesma.

Neste projecto pretendemos desenvolver um programa em C que registe e manipule a informação referente à gestão de uma rede de aeroportos. Cada aeroporto é representado por um vértice, enquanto que as arestas entre vértices representam ligações aéreas entre dois aeroportos.



## 2. Especificação do programa

Um aeroporto é composto por um *código de identificação* (3 carácteres), uma *capacidade máxima de operação*, e um *estado*. A capacidade máxima de operação deverá ser um número inteiro maior do que zero, e corresponde ao número máximo (total) de voos que podem partir e chegar ao aeroporto; o estado de um aeroporto poderá tomar um de dois valores: 1 (*aberto*) ou 0 (*encerrado*). Ao longo do tempo, a capacidade de operação e o estado de cada um dos aeroportos pode variar. O código de identificação de um aeroporto é fixo. Assume-se que o limite máximo do número total de aeroportos é de $10^3$.

Um *voo* é um par de aeroportos $a_1$, $a_2$ correspondente a um voo do aeroporto $a_1$ para o aeroporto $a_2$. Sempre que um novo voo é criado, é adicionada uma nova ligação entre os dois aeroportos envolvidos.

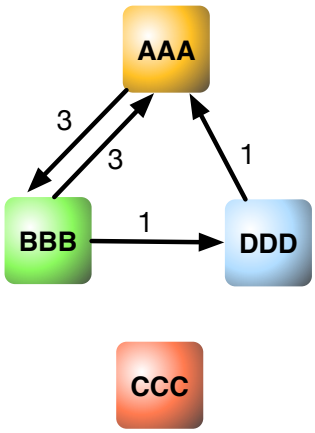
O sistema de gestão de aeroportos permite adicionar aeroportos, alterar a sua capacidade, adicionar e remover rotas, e listar uma série de outras informações.

## 3. Sugestão de estrutura de dados

Um sistema de gestão de aeroportos e rotas pode ser visto como um grafo orientado. Visto poderem existir vários voos entre os mesmos aeroportos, estamos na verdade perante um grafo orientado com pesos, i.e., cada aresta tem uma direção e um valor (ou peso) associado.

Há várias estruturas de dados apropriadas para representar um grafo orientado pesado. A estrutura mais simples talvez seja a matriz de adjacências[1], cuja ideia passamos a exemplificar. Dado um grafo de $N$ nós (ou vértices), este pode ser representado por uma matriz $A$, de tamanho $N \times N$, onde cada elemento $A_{ij}$ representa uma ligação entre o aeroporto $i$ e o aeroporto $j$. O valor de $A_{ij}$ poderá representar por exemplo o número de voos semanais de $i$ para $j$.

Considere o seguinte exemplo, constituído por 4 aeroportos (ou nós): AAA, BBB, CCC e DDD. Considere que existe um 3 voos seman<ais de ida e volta entre os aeroportos AAA e BBB, um voo semanal entre BBB e DDD, e outro entre DDD e AAA.




| | AAA | BBB | CCC | DDD |
|---|---|---|---|---|
| AAA | 0 | 3 | 0 | 0 |
| BBB | 3 | 0 | 0 | 1 |
| CCC | 0 | 0 | 0 | 0 |
| DDD | 1 | 0 | 0 | 0 |

Podemos visualizar o grafo resultante do lado esquerdo na figura em cima. Se recorrermos a uma representação do mesmo grafo através de uma matriz de adjacências obtemos a tabela do painel da direita.

## 4. Dados de Entrada

O programa deverá ler os dados de entrada a partir do standard input, na forma de um conjunto de linhas iniciadas por um carácter, que se passa a designar por *comando*, seguido de um

[1] http://en.wikipedia.org/wiki/Adjacency_matrix

número variável de informações; o comando e cada uma das informações são separados por um espaço e assim, dentro de cada informação, não poderão existir outros espaços.

Os comandos disponíveis são descritos de seguida e correspondem às letras `A, I, F, G, R, S, N, P, Q, V, C, O, L, X`. Cada comando indica uma determinada acção que se passa a caracterizar em termos de objectivo, número de informações por linha (n.i.l.), sintaxe e output[2]:

`A`: adiciona um novo aeroporto (n.i.l. = 2)

```
A id capacidade
```

- `id`: string de **3 letras**;
- `capacidade`: inteiro maior que 0.

*Output*: Não tem qualquer output.

Adiciona o aeroporto com o identificador e capacidade respectivas. Poderá assumir que o utilizador nunca insere dois aeroportos com o mesmo identificador.

`I`: altera a capacidade máxima de operação de um aeroporto (n.i.l. = 2)

```
I id capacidade
```

- `id`: string de **3 letras**;
- `capacidade`: inteiro positivo (ou negativo) para aumentar (ou diminuir) a capacidade máxima de operação de um aeroporto..

*Output*: Não tem qualquer output (ver observações).

Altera a *capacidade máxima de operação* do aeroporto `<id>` de acordo com o valor introduzido. A nova capacidade máxima deverá ser sempre superior ou igual ao número de voos actuais do aeroporto.

*Observações:* Deverá retornar a string *`Capacidade de <id> inalterada` caso o aeroporto introduzido não esteja definido ou esteja encerrado, ou caso a nova capacidade máxima seja inferior ao número de voos actual do aeroporto. Neste caso, a capacidade máxima ficará inalterada.

`F`: adiciona voo de ida e volta (n.i.l. = 2)

```
F id_1 id_2
```

- `id_1` e `id_2` :string de **3 letras**.

*Output*: Não tem qualquer output.

Adiciona os voos de ida e volta entre os aeroportos indicados.

[2] Se encontrar dificuldades no tratamento do input, veja p.f. o exemplo de linha de comandos dado nas aulas teóricas.

*Observações:* Deverá retornar a string *`Impossivel adicionar voo RT <id_1> <id_2>` caso algum dos aeroportos introduzidos não esteja definido ou esteja encerrado, ou caso não seja possível adicionar o voo com partida em `<id_1>` e chegada a `<id_2>`. Poderá assumir que os aeroportos são diferentes.

`G`: adiciona rota (n.i.l. = 2)

```
G id_1 id_2
```

- `id_i`:string de **3 letras**.

*Output*: Não tem qualquer output.

Adiciona o voo de ida de `id_1` para `id_2`.

*Observações:* Análogo ao comando `F` retornando no entanto a string *`Impossivel adicionar voo <id_1> <id_2>`. Poderá assumir que os dois aeroportos introduzidos são diferentes.

`R`: remove voo (n.i.l. = 2)

```
R id_1 id_2
```

- `id_1` e `id_2` :string de **3 letras**.

*Output*: Não tem qualquer output.

Remove o voo com partida em `<id_1>` e chegada a `<id_2>`.

*Observações:* Deverá retornar a string *`Impossivel remover voo <id_1> <id_2>` caso não seja possível remover o voo com partida em `<id_1>` e chegada a `<id_2>`.

`S`: remove voo de ida e volta (n.i.l. = 2)

```
S id_1 id_2
```

- `id_1` e `id_2` :string de **3 letras**.

*Output*: Não tem qualquer output.

Remove o voo de ida e volta entre `<id1>` e `<id2>`. *Só deverá ser efectuada caso seja possível remover ambos os voos.*

*Observações:* Deverá retornar a string *`Impossivel remover voo RT <id_1> <id_2>` caso seja impossível remover qualquer um dos voos.

`N`: Número de voos (n.i.l. = 2)

```
N id_1 id_2
```

- `id_1` e `id_2` :string de **3 letras**.

*Output*: `Voos entre cidades id_1:id_2:Noutgoing:Nincoming`

Retorna o número de voos entre os aeroportos <id_1> e <id_2>, saindo de <id_1> para <id_2> (Noutgoing) e de <id_2> para <id_1> (Nincoming).

*Observações:* Deverá retornar a string *Aeroporto <id> inexistente caso algum dos aeroportos introduzidos não esteja definido ou encerrado.

P: Aeroporto com o maior número de voos (n.i.l. = 0)
P

*Output*: `Aeroporto com mais rotas id:Noutgoing:Nincoming`

Retorna o aeroporto <id> com o maior número total de voos (soma das partidas e chegadas).

*Observações:* No caso de existir mais do que um aeroporto na situação acima, deverá retornar o aeroporto introduzido há mais tempo no sistema.

Q: Aeroporto mais conectado (n.i.l. = 0)
Q

*Output*: `Aeroporto com mais ligações id:Naeroportos`

Retorna o aeroporto <id> com o maior número de outros aeroportos conectados.

*Observações:* No caso de existir mais do que um aeroporto na situação acima, deverá retornar o aeroporto introduzido há mais tempo no sistema.

V: Voo mais popular (n.i.l. = 0)
V

*Output*: `Voo mais popular id1:id2:Nvoos`

Retorna o par de aeroportos <id_1> <id_2> com o maior número de voos (Nvoos) a sair de <id_1> e a chegar a <id_2>.

*Observações:* No caso de existir mais do que um par de aeroportos na situação acima, deverá retornar o par cujo aeroporto de partida tenha sido introduzido há mais tempo no sistema; se mesmo assim houver mais do que uma solução, deverá escolher entre essas aquela cujo aeroporto de chegada tenha sido introduzido há mais tempo no sistema.

C: Encerra aeroporto (n.i.l. = 1)
C id

- id: string de **3 letras**.

*Output*: Não tem qualquer output.

Encerra o aeroporto actualmente em operação `<id>` removendo todos os voos com partida ou chegada a `<id>`.

*Observações:* Deverá retornar a string *`Aeroporto <id> inexistente` caso o aeroporto introduzido não esteja definido ou já esteja encerrado.

`O`: Reabre aeroporto (n.i.l. = 1)
`O id`

- `id`: string de **3 letras.**

*Output*: Não tem qualquer output.

Reabre o aeroporto actualmente encerrado `<id>`.

*Observações:* Deverá retornar a string *`Aeroporto <id> inexistente` caso o aeroporto introduzido não esteja definido. Ao reiniciar a actividade o aeroporto começa sem qualquer rota activa.

`L`: emitir listagem (n.i.l. =1)
`L tipo`

- `tipo`: número inteiro identificador da listagem pretendida, a definir na Secção 5.

*Observações*: ver Secção 5 para detalhes do output.

`X`: sair do programa (n.i.l. = 0)
`X`

*Output*: `Nvoos:Naeroportos`

Termina o programa retornando o número total de voos e aeroportos actualmente no sistema.

# 5. Dados de Saída

O programa deverá escrever no standard output as respostas a certos comandos apresentados no standard input. As respostas são igualmente linhas de texto formatadas conforme definido neste enunciado. Tenha em atenção o número de espaços entre elementos do seu output, assim como os espaços no final de cada linha. Procure respeitar escrupulosamente as indicações dadas.

As respostas aos pedidos no input são as indicadas na secção anterior:

- A resposta ao comando **`N id_1 id_2`** deverá ser a string

```
Voos entre cidades id_1:id_2:Noutgoing:Nincoming
```

indicando o número de voos entre os aeroportos <id_1> e <id_2>.

- A resposta ao comando **`P`** deverá ser a string

```
Aeroporto com mais rotas id:Noutgoing:Nincoming
```

correspondente ao aeroporto com o maior número total de voos (partidas/outgoing e chegadas/incoming).

- A resposta ao comando **`Q`** deverá ser a string

```
Aeroporto com mais ligações id:Naeroportos
```

correspondente ao aeroporto com o maior número de outros aeroportos conectados.

A resposta ao comando **`V`** deverá ser a string

```
Voo mais popular id1:id2:Nvoos
```

correspondente ao par de aeroportos <id_1> <id_2> com o maior número de voos (Nvoos) a sair de <id_1> e a chegar a <id_2>.

- A resposta ao comando **`X`** deverá ser a string

```
Nvoos:Naeroportos
```

correspondente ao número total de voos e aeroportos actualmente no sistema.

- Como resposta ao comando de listagem **`L 0`**, o programa deverá listar todos os aeroportos ordenados por ordem de entrada no sistema. Cada aeroporto deverá ser impresso numa linha distinta na forma

```
id:capacidade:Noutgoing:Nincoming
```

onde,

- id é o identificador do aeroporto;
- capacidade é a capacidade máxima de operação desse aeroporto;
- Noutgoing/Nincoming: respectivamente o número de voos que partem/chegam ao aeroporto.

- Como resposta ao comando de listagem **`L 1`**, o programa deverá listar todos os aeroportos ordenados por ordem lexicográfica (alfabética)[3]. Cada aeroporto deverá ser impresso numa linha distinta na mesma forma que no caso **`L 0`**.

- Como resposta ao comando de listagem **`L 2`**, o programa deverá escrever a distribuição/histograma *d(k)* do número de aeroportos actualmente com *exactamente k* voos (total de partidas e chegadas), sob a forma:

```
0:d(0)
1:d(1)
2:d(2)
...
```

Por exemplo, o output poderá tomar a forma

```
0:2
1:7
2:3
3:1
6:1
```

correspondendo a primeira coluna a *k* e a segunda ao número de aeroportos com *k* voos, i.e., *d(k)*. No exemplo acima, a segunda linha indica que existem 7 aeroportos com apenas 1 voo. Quando *d(k)=0*, a respectiva linha deverá ser omitida. No exemplo acima, por não exisitirem aeroportos com 4 voos, não existe nenhuma linha que comece com 4.

# 6. Exemplos (Input/Output)

## Exemplo 1

**Dados de Entrada:**

```
A LAX 10
A LIS 20
A OPO 30
F LAX LIS
G LAX OPO
G OPO LIS
G LIS LAX
I LIS 15
I LAX -8
X
```

**Dados de Saída:**

```
*Capacidade de LAX inalterada
5:3
```

[3] Poderá utilizar a função `strcmp` disponível em `string.h`

## Exemplo 2

**Dados de Entrada:**

```
A LIS 20
A LAX 10
A OPO 30
F LAX LIS
F LIS OPO
G LAX OPO
G OPO NCE
G NCE LIS
G LIS LAX
F LIS LAX
F OPO LAX
F OPO LAX
I LAX 20
I LIS -15
N LIS LAX
V
G LIS NCE
P
Q
X
```

**Dados de Saída:**

```
*Impossivel adicionar voo OPO NCE
*Impossivel adicionar voo NCE LIS
*Capacidade de LIS inalterada
Voos entre cidades LIS:LAX:3:2
Voo mais popular LIS:LAX:3
*Impossivel adicionar voo LIS NCE
Aeroporto com mais rotas LAX:5:5
Aeroporto com mais ligações LIS:2
12:3
```

## Exemplo 3

**Dados de Entrada:**

```
A LIS 20
A LAX 10
A OPO 30
F LAX LIS
F LIS OPO
G LAX OPO
G OPO LAX
N LAX OPO
R LAX OPO
S LAX OPO
C LAX
N LAX OPO
O LAX
N LAX OPO
L 0
X
```

**Dados de Saída:**

```
Voos entre cidades LAX:OPO:1:1
*Impossivel remover voo RT LAX OPO
Voos entre cidades LAX:OPO:0
Voos entre cidades LAX:OPO:0
LAX:10:0:0
LIS:20:1:1
OPO:30:1:1
2:3
```

## Exemplo 4

**Dados de Entrada:**

```
A LAX 10
A LIS 20
A OPO 30
A NCE 10
F LAX LIS
F LIS OPO
G LAX OPO
G OPO NCE
G NCE LIS
G LIS LAX
F LIS LAX
F OPO LAX
F OPO LAX
N LIS LAX
G LIS NCE
L 2
X
```

**Dados de Saída:**

```
Voos entre cidades LIS:LAX:3:2
3:1
8:1
9:1
10:1
15:4
```

# 7. Compilação do Programa

O compilador a utilizar é o `gcc` com as seguintes opções de compilação: `-Wall`. Para compilar o programa deve executar o seguinte comando:

```
$ gcc -Wall -o proj1 *.c
```

o qual deve ter como resultado a geração do ficheiro executável `proj1`, caso não haja erros de compilação. A execução deste comando não deverá escrever qualquer resultado no terminal. Caso a execução deste comando escreva algum resultado no terminal, considera-se que o programa não compilou com sucesso. Por exemplo, durante a compilação do programa, o compilador não deve escrever mensagens de aviso (warnings).

# 8. Execução do Programa

O programa deve ser executado da forma seguinte:

```
$ ./proj1 < test01.in > test01.myout
```

Posteriormente poderá comparar o seu output com o output previsto usando o comando **`diff`**

```
$ diff test01.out test01.myout
```

# 9. Entrega do Projecto

A entrega do projecto deverá respeitar o procedimento seguinte:

- Na página da disciplina aceda ao sistema para entrega de projectos. O sistema será activado uma semana antes da data limite de entrega. Instruções acerca da forma de acesso ao sistema serão oportunamente fornecidas.

- Efectue o upload de um ficheiro arquivo com extensão `.zip` que inclua todos os ficheiros fonte que constituem o programa. Se utilizar ficheiros cabeçalho (`.h`) não se esqueça de os juntar também ao pacote.

- Para criar um ficheiro arquivo com a extensão `.zip` deve executar o seguinte comando na directoria onde se encontram os ficheiros com extensão `.c` e `.h` (se for o caso), criados durante o desenvolvimento do projecto:
  ```
  $ zip proj1.zip *.c *.h
  ```

  Se só tiver um único ficheiro (e.g., `proj1.c`), bastará escrever:
  ```
  $ zip proj1.zip proj1.c
  ```

- Como resultado do processo de upload será informado se a resolução entregue apresenta a resposta esperada num conjunto de casos de teste.

- O sistema **não permite submissões com menos de 10 minutos de intervalo** para o mesmo grupo. Exemplos de casos de teste serão oportunamente fornecidos.

- Data limite de entrega do projecto: **2 de Abril de 2016 (11h59m)**. Até à data limite poderá efectuar o número de submissões que desejar, sendo utilizada para efeitos de avaliação a última submissão efectuada. Deverá portanto verificar cuidadosamente que a última submissão corresponde à versão do projecto que pretende que seja avaliada. Não existirão excepções a esta regra.

# 10. Avaliação do Projecto

### a. Componentes da Avaliação

Na avaliação do projecto serão consideradas as seguintes componentes:

1. A primeira componente avalia o desempenho da funcionalidade do programa realizado. Esta componente é avaliada entre 0 e 16 valores.
2. A segunda componente avalia a qualidade do código entregue, nomeadamente os seguintes aspectos: comentários, identação, estruturação, modularidade, abstracção, entre outros. Esta componente poderá variar entre -4 valores e +4 valores relativamente à classificação calculada no item anterior e será atribuída na discussão final do projecto.
3. Durante a discussão final do projecto será averiguada a participação de cada elemento do grupo na realização do projecto, bem como a sua compreensão do

trabalho realizado. A respectiva classificação será ponderada em conformidade, isto é, elementos do mesmo grupo podem ter classificações diferentes. Elementos do grupo que se verifique não terem participado na realização do respectivo projecto terão a classificação de 0 (zero) valores.

### b. Atribuição Automática da Classificação

- A classificação da primeira componente da avaliação do projecto é obtida através da execução *automática* de um conjunto de testes num computador com o sistema operativo GNU/Linux. Torna-se portanto essencial que o código compile correctamente e que respeite o formato de entrada e saída dos dados descrito anteriormente. Projectos que não obedeçam ao formato indicado no enunciado serão penalizados na avaliação automática, podendo, no limite, ter 0 (zero) valores se falharem todos os testes. Os testes considerados para efeitos de avaliação poderão incluir (ou não) os disponibilizados na página da disciplina, além de um conjunto de testes adicionais. A execução de cada programa em cada teste é limitada na quantidade de memória que pode utilizar, até um máximo de 64 Mb, e no tempo total disponível para execução, sendo o tempo limite distinto para cada teste.
- Note-se que o facto de um projecto passar com sucesso o conjunto de testes disponibilizado na página da disciplina não implica que esse projecto esteja totalmente correcto. Apenas indica que passou alguns testes com sucesso, mas este conjunto de testes não é exaustivo. É da responsabilidade dos alunos garantir que o código produzido está correcto.
- Em caso algum será disponibilizado qualquer tipo de informação sobre os casos de teste utilizados pelo sistema de avaliação automática. A totalidade de ficheiros de teste usados na avaliação do projecto serão disponibilizados na página da disciplina após a data de entrega.