INSTITUTO SUPERIOR TÉCNICO

Introdução aos Algoritmos e Estruturas de Dados 2010/2011

Enunciado do 3^{o} Projecto

Data de entrega: 20 de Maio de 2011 às 23h00

1 Introdução

Quando se faz o planeamento de uma viagem de avião existem vários factores a considerar: hora de partida, hora de chegada, preço, número de escalas, etc. Cada pessoa tem prioridades diferentes, pelo que tendo como base o mesmo conjunto de voos disponível, a melhor solução depende das prioridades definidas por cada pessoa que viaja.

Neste projecto pretende-se desenvolver um programa que manipule informação relativa a voos entre aeroportos. Para tal, no *input* é fornecido um conjunto de voos que deverá ser considerado por forma a satisfazer os pedidos de viagem. Nesta fase do projecto é pedido que guarde informação sobre os voos e aeroportos e responda a pedidos de informação, tendo em conta que os voos são frequentemente adiados, ou até cancelados. Em situações extremas, os aeroportos podem mesmo ser temporariamente encerrados e depois reabertos.

2 Especificação do Programa

O programa recebe como *input* um conjunto de voos e um conjunto de pedidos de informação sobre os aeroportos. Recebe também pedidos de cancelamento e adiamento de voos, pedidos de fecho e reabertura de aeroportos, e pedidos de informação relativamente à existência de percursos entre um aeroporto de origem e um aeroporto de destino envolvendo um determinado conjunto de voos.

As linhas que especificam um voo começam com o caracter `v` sendo que para cada voo é dada a seguinte informação: código do voo, aeroporto de origem e de destino, hora de partida no aeroporto de origem, hora de chegada no aeroporto de destino e o preço. O código de um voo é definido como sendo uma sequência de caracteres composta por duas letras seguida por algarismos (no mínimo dois e no máximo quatro). Os aeroportos são identificados por códigos compostos por três letras.

Para além dos voos, o *input* contém linhas que começam com o caracter `i` que correspondem a pedidos de informação relativa a um aeroporto. Quando é lida uma linha de pedido de informação, o programa deverá fazer o *output* da informação relativa a esse aeroporto considerando apenas os voos especificados no *input* **antes** dessa linha.

O *input* contém também linhas que correspondem ao cancelamento de um dado voo. Estas linhas começam com o caracter `c`, seguido de um código de voo. A partir do momento em que um voo é cancelado, este deverá deixar de ser considerado nos pedidos de informação. Esta linha não produz efeito se o voo não existir ou já tiver sido cancelado.

Um voo pode também ser adiado por determinado tempo. Para que este adiamento seja possível, o *input* contém linhas que começam com o caracter `a` seguido do tempo de atraso no formato HH:MM. O tempo de atraso de um voo é sempre inferior a 24 horas. Esta linha não produz efeitos se o voo não existir ou tiver sido cancelado.

Note-se que o adiamento pode fazer com que o voo passe para o dia seguinte. Por exemplo, suponha que no aeroporto LIS há um voo TP001 com partida às 20:00 e outro voo TP002 com partida às 21:00. Considere ainda que o voo TP002 é adiado por 04:00 (4 horas). Neste caso, a nova hora do voo TP002 é 01:00 e o voo com hora de partida mais tardia passou a ser o voo TP001.

Para além de cancelamento e adiamento de voos, podem ainda surgir casos mais complexos que obriguem ao fecho de aeroportos. Para especificar o fecho de um aeroporto, o *input* contém linhas que começam com o caracter `f`, seguido de um código de aeroporto. A partir do momento em que um aeroporto é fechado, e até ao momento em que o aeroporto seja eventualmente reaberto, os voos com origem ou destino neste aeroporto devem deixar de ser considerados nos pedidos de informação. Esta linha não produz efeito se o aeroporto não existir ou já estiver fechado.

Para permitir reabrir um aeroporto, o *input* pode conter também linhas que começam com o caracter `r`, seguido de um código de aeroporto. A partir do momento em que um aeroporto é reaberto, os voos não cancelados com origem ou destino no aeroporto devem voltar a ser considerados nos pedidos de informação. Esta linha não produz efeito se o aeroporto não existir ou se não estiver fechado.

Podem ainda ser efectuados pedidos de informação relativamente a percursos entre um aeroporto de origem e um aeroporto de destino considerando um conjunto predefinido de voos (percurso). As linhas que solicitam informação sobre percursos começam com o caracter `p`, seguido do número de voos a considerar no percurso e de uma lista com os códigos dos respectivos voos. À semelhança do que acontece com as linhas `i`, quando é lida uma linha `p`, o programa deverá fazer o *output* da informação relativa aos percursos considerando apenas a informação do *input* **antes** dessa linha.

Um percurso com k voos $< v_1, v_2, ..., v_k >$ diz-se válido se e só se as seguintes condições são respeitadas:

- A origem do voo v_1 é o aeroporto de origem do percurso;
- O destino do voo v_k é o aeroporto de destino do percurso;

- O destino do voo v_i é o mesmo que a origem do voo v_{i+1} para qualquer $1 \leq i < k$;
- Nenhum dos voos do percurso foi cancelado e nenhum dos aeroportos de origem, destino ou de qualquer escala do percurso está fechado;

Finalmente, existirá uma opção para determinar o número mínimo de voos entre dois aeroportos, isto é, o percurso entre um aeroporto de origem e um aeroporto de destino com o menor número de escalas. As linhas que solicitam informação sobre o percurso com menor número de voos começam com o caracter `m`, seguido dos códigos do aeroporto de origem e de destino. Considere que nesta opção, os aeroportos de origem e destino são sempre diferentes.

No *output*, o programa deverá apresentar a informação relativa aos aeroportos e aos percursos. Por cada linha `i` do *input*, deverá ser apresentado o número de voos que se realizam com origem e destino num dado aeroporto, qual o destino (não considerando escalas) mais barato e qual o destino do voo com hora de partida mais tardia. Nas situações em que existe mais do que um voo com o mesmo preço ou com a mesma hora de partida, devem considerar o que aparece primeiro no *input*.

Por cada linha `p` do *input*, deverá ser apresentada a palavra "OK", seguida do preço total do percurso, e de uma lista com os pares (aeroporto, hora de partida do voo) que permitem efectuar o percurso. Caso o percurso não seja possível com os voos disponíveis no momento em que o pedido de informação é efectuado, deverá ser apresentada a palavra "KO".

Por cada linha `m` do *input*, deverá ser apresentada a palavra "OK", seguida do número mínimo de voos entre o aeroporto de origem e o aeroporto de destino. Caso não seja possível definir um percurso entre esses aeroportos, no momento em que o pedido de informação é efectuado, deverá ser apresentada a palavra "KO".

3 Dados de Entrada

O programa deverá ler os dados de entrada a partir do *standard input*. O formato dos dados de entrada deverá ser um conjunto de linhas `v`, `i`, `c`, `a`, `f`, `r`, `p` ou `m`. O formato de cada possível linha de entrada é descrito de seguida, sendo que o separador de cada campo é sempre um espaço em branco.

- Uma linha `v` que define um voo tem o seguinte formato:
  - um caracter `v`;
  - código de voo;
  - código do aeroporto de origem;
  - código do aeroporto de destino;
  - hora de partida do voo no aeroporto de origem no formato HH:MM;
  - hora de chegada do voo no aeroporto de destino no formato HH:MM;
  - preço do bilhete em euros definido por um número real.

- Uma linha `i` que define um pedido de informação sobre um aeroporto tem o seguinte formato:
  - um caracter `i`;
  - código do aeroporto.
- Uma linha `c` que define um pedido de cancelamento de um voo tem o seguinte formato:
  - um caracter `c`;
  - código do voo.
- Uma linha `a` que define um pedido de adiamento de um voo tem o seguinte formato:
  - um caracter `a`;
  - código do voo;
  - tempo de atraso do voo no formato HH:MM;
- Uma linha `f` que define um pedido de fecho de um aeroporto tem o seguinte formato:
  - um caracter `f`;
  - código do aeroporto.
- Uma linha `r` que define um pedido de reabertura de um aeroporto tem o seguinte formato:
  - um caracter `r`;
  - código do aeroporto.
- Uma linha `p` que define um pedido de informação acerca de um percurso tem o seguinte formato:
  - um caracter `p`;
  - código do aeroporto de origem;
  - código do aeroporto de destino;
  - um inteiro K ($K \geq 1$) que denota o número total de voos a efectuar no percurso;
  - os códigos dos K voos a efectuar no percurso, separados por espaços em branco.
- Uma linha `m` que define um pedido de informação acerca do percurso com menor número de voos tem o seguinte formato:
  - um caracter `m`;
  - código do aeroporto de origem;
  - código do aeroporto de destino.

Assuma que os dados de entrada para o programa não contêm erros sintácticos, isto é, obedecem sempre ao formato descrito nesta secção.

4 Dados de Saída

Os dados de saída correspondem à informação relativa às linhas de tipo `i` e `p`. A informação a escrever no *output* deverá ser Uma sequência de linhas onde cada linha corresponde a informação sobre um aeroporto ou percurso pedida no *input*. A ordem das linhas do tipo `i`, `p` e **t m** do *input* é respeitada no *output* e para cada linha deve ser considerada apenas a informação introduzida até essa altura.

O output para cada linha `i` é o descrito de seguida, onde o separador é um espaço em branco:

- Código do aeroporto a;
- Número de voos com origem no aeroporto a; 0 se o aeroporto estiver fechado;
- Número de voos com destino ao aeroporto a; 0 se o aeroporto estiver fechado;
- Código do aeroporto de destino com viagem mais barata (sem escalas) com origem em a. Se o número de voos com origem em a for 0 ou se o aeroporto estiver fechado, deverá colocar 000;
- Código do aeroporto de destino com hora de partida mais tardia com origem em a; Se o número de voos com origem em a for 0 ou se o aeroporto estiver fechado, deverá colocar 000;

O output para cada linha `p` é o descrito de seguida, onde o separador é um espaço em branco:

- "KO", se o percurso especificado não é válido;
- Caso contrário, se o percurso é válido, o output deverá ser a palavra "OK" seguido do seguinte:
  - Preço total do percurso (real com duas casas decimais);
  - Código do aeroporto e hora de partida de cada voo no percurso; o formato das horas é HH:MM;

O output para cada linha `m` é o descrito de seguida, onde o separador é um espaço em branco:

- "KO", se não existir um percurso entre o aeroporto de origem e o aeroporto de chegada;
- Caso contrário, o output deverá ser a palavra "OK" seguido do número mínimo de voos entre o aeroporto de partida e o aeroporto de chegada.

5 Exemplo

5.1 Dados de Entrada

Vamos admitir que o ficheiro de entrada, que designaremos por `in.txt`, tem o seguinte conteúdo:

```
v TP001 LIS CDG 08:00 11:30 247.56
v IB002 MAD CDG 08:15 10:30 190.67
i LHR
v BA375 LIS LHR 07:00 10:15 185.73
i LIS
m LIS CDG
m LIS MAD
m MAD LIS
v TP002 CDG LIS 10:00 11:30 220.00
v BA836 CDG LHR 10:00 10:40 99.99
i LIS
i CDG
i LHR
f LIS
i LIS
p LIS LIS 2 TP001 TP002
m LIS CDG
r LIS
a TP002 00:50
i LIS
p LIS LIS 2 TP001 TP002
m LIS CDG
p CDG LHR 3 TP002 TP001 BA836
c TP002
p LIS LIS 2 TP001 TP002
f MAD
i MAD
p MAD CDG 1 IB002
r MAD
p MAD CDG 1 IB002
v TP004 LIS CDG 12:00 14:30 200.00
m LIS CDG
v IB003 CDG LIS 10:15 12:30 90.67
m MAD LIS
v TP005 MAD LIS 08:00 9:00 60.00
m MAD LIS
```

5.2 Dados de Saída

Para o ficheiro de entrada `in.txt` descrito na secção anterior, o programa deverá escrever na saída a seguinte informação:

```
LHR 0 0 000 000
LIS 2 0 LHR CDG
OK 1
KO
KO
LIS 2 1 LHR CDG
CDG 2 2 LHR LIS
LHR 0 2 000 000
LIS 0 0 000 000
KO
KO
LIS 2 1 LHR CDG
OK 467.56 LIS 08:00 CDG 10:50
OK 1
OK 567.55 CDG 10:50 LIS 08:00 CDG 10:00
KO
MAD 0 0 000 000
KO
OK 190.67 MAD 08:15
OK 1
OK 2
OK 1
```

6 Compilação do Programa

Deve concretizar o ficheiro `Makefile` onde deverá definir o processo de geração do executável correspondente ao programa realizado. **O compilador a utilizar é o** `gcc` **com as seguintes opções de compilação:** `-ansi -Wall -pedantic -g` **. Este processo deve ser definido na regra** `all` **do ficheiro** `Makefile`. O executável gerado deve ter o nome `proj3`. Para compilar o programa deve executar o seguinte comando:

```
$ make -s all
```

o qual deve ter como resultado a geração do ficheiro executável `proj3`, caso não haja erros de compilação. A execução deste comando não deverá escrever qualquer resultado no terminal (daí a utilização da opção `-s` do comando `make`). 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 *warning*.

7 Execução do Programa

O programa deve ser executado da forma seguinte:

```
$ ./proj3 < in.txt > out.txt
```

8 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 `.tgz` que inclua o seguinte:
  - Ficheiros fonte (`.c`) e cabeçalho (`.h`) que constituem o programa.
  - Ficheiro `Makefile` que permita criar o executável `proj3`.

  Para criar um ficheiro arquivo com a extensão `.tgz` deve executar o seguinte comando na directoria onde se encontram o ficheiro `Makefile` e os ficheiros com extensão `.c` e `.h`, criados durante o desenvolvimento do projecto:

  ```
  $ tar cfz proj3.tgz Makefile *.c *.h
  ```
- 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 30 minutos de intervalo para o mesmo grupo.** Exemplos de casos de teste serão oportunamente fornecidos.
- Data limite de entrega do projecto: **23h00 do dia 20 de Maio de 2011**. Até à data limite poderá efectuar o número de entregas que desejar, sendo utilizada para efeitos de avaliação a **última** entrega efectuada. Deverá portanto verificar cuidadosamente que a última entrega realizada corresponde à versão do projecto que pretende que seja avaliada. Não serão abertas excepções.

9 Avaliação do Projecto

9.1 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. Será também **utilizado o sistema `valgrind` for forma a detectar fugas de memória ("memory leaks") ou outras incorrecções no código. No caso da existência de incorrecções, haverá uma penalização de 20% na primeira componente**. Aconselha-se por isso que utilizem o valgrind para fazer debug do código e corrigir eventuais incorrecções antes da submissão do projecto.

4. **Grupos que não utilizem as flags correctas de compilação têm penalização de 50% na primeira componente**.

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, sendo a respectiva classificação 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.

9.2 Atribuição Automática da Classificação

A classificação da primeira componente da avaliação do projecto é obtida automaticamente através da execução de um conjunto de testes executados 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 podem 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 32 MBytes, e no tempo total disponível para execução.

Note-se que o facto de um projecto passar com sucesso no 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 disponibilizada 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.

9.3 Detecção de Cópias

A avaliação dos projectos inclui a utilização de um sistema para detecção de situações de cópia entre projectos. A submissão de um projecto pressupõe o compromisso de honra que o trabalho incluso foi realizado única e exclusivamente pelos alunos referenciados nos ficheiros submetidos para avaliação. A quebra deste compromisso, ou seja a tentativa de um grupo se apropriar de trabalho de outrém (sejam colegas ou outra pessoa), tem como consequência a reprovação de todos os alunos envolvidos (incluindo quem possibilitar a ocorrência de cópias) à disciplina.

Toda e qualquer situação de fraude ou facilitação de fraude terá então como consequência a reprovação imediata à disciplina de IAED neste semestre, assim como a comunicação da ocorrência ao respectivo Coordenador de curso e ao Conselho Pedagógico do IST para sanções adicionais de acordo com as regras aprovadas pela UTL e publicadas em Diário da República.

9.4 Considerações adicionais

Todos os programas são avaliados do modo seguinte:

```
$ ./proj3 < in.txt > out.txt; diff out.txt exp.txt
```

em que o ficheiro `exp.txt` representa o resultado esperado da execução do programa para os dados de entrada definidos pelo ficheiro `in.txt`. A impossibilidade de verificar automaticamente o resultado da execução de um dado programa implica uma penalização de **100%** na componente de avaliação automática. Considera-se que um programa passou um teste com sucesso se o resultado produzido por esse programa for exactamente igual ao resultado esperado, o que significa que o comando `diff` não deverá encontrar diferenças entre o resultado produzido pelo programa submetido e o esperado. Para poder ser avaliado, um projecto deverá compilar correctamente num computador com o sistema operativo **GNU/Linux**, sendo o utilizado o compilador **gcc** da **GNU**. A entrega de código não compilável, ou a não inclusão de qualquer dos ficheiros requeridos, ou a utilização de nomes diferentes para o ficheiro executável conduz a uma classificação de 0 (zero) valores. Não serão aceites quaisquer justificações.