INSTITUTO SUPERIOR TÉCNICO

Introdução aos Algoritmos e Estruturas de Dados

2017/2018 - 2º semestre

Enunciado do 1º Projecto (v1.0)

Data de entrega: 27 de Abril de 2017 (23h59m)

1. Introdução

Uma matriz onde grande parte dos seus valores não é representada, ou é zero, é designada por *matriz esparsa*. Tal é o caso de folhas de cálculo, por exemplo. Também a representação de circuitos eléctricos e digitais, bem como inúmeras outras aplicações em engenharia, física e matemática resultam em matrizes esparsas. Ferramentas como o Matlab ou o Wolfram suportam matrizes esparsas em simultâneo com matrizes onde todos os elementos são representados.
As matrizes esparsas permitem poupanças significativas de memória quando poucos valores necessitam de ser efectivamente representados. Por exemplo, um circuito com 1000 componentes necessitaria de uma matriz de 1000x1000 para representar as ligações entre os componentes, No entanto o número de ligações é muito inferior a um milhão (1000x1000) pois muitos componentes estão apenas ligados a um outro e a maioria das ligações têm menos de 10 componentes. Numa matriz esparsa apenas as ligações realizadas são efectivamente representadas. No entanto, existe um custo adicional em cada ligação, pelo que uma representação esparsa apenas compensa se mais de metade dos elementos da matriz puderem ser omitidos. Esses elementos que podem ser omitidos são chamados de *zeros*. A *densidade* de uma matriz mede o número de elementos efectivamente representados face ao número total de elementos possíveis na matriz. Quanto menos densa for a matriz esparsa, mais eficiente é a sua representação e manipulação.

O objectivo deste projeto é o desenvolvimento, em linguagem C, de um programa para gestão de matrizes esparsas. Mais especificamente, a interacção com o programa deverá ocorrer através de um conjunto de linhas compostas por uma letra (comando) e um número de argumentos dependente do comando a executar. Os possíveis comandos são listados na Tabela seguinte e indicam as operações a executar.

Comando	Descrição
`a`	Adiciona um elemento à matriz.
`p`	Lista todos os elementos representados.
`i`	Lista os limites da matriz e a sua densidade.
`l`	Lista uma linha completa da matriz.
`c`	Lista uma coluna completa da matriz.
`o`	Ordena as linhas, ou colunas, da matriz.
`z`	Define o elemento *zero* da matriz.
`s`	Comprime a matriz (double-offset indexing).
`w`	Guarda os elementos da matriz num ficheiro.
`q`	Termina o programa

2. Especificação do programa

Um elemento da matriz é representado por uma *linha* e uma *coluna* (dois inteiros não negativos com pelo menos 32 bits) e um *valor* real (um número em vírgula flutuante em precisão dupla). Assuma que a matriz nunca terá mais de 10000 elementos representados e que a compactação nunca necessitará de mais do dobro dos elementos existentes. A aplicação deverá ainda guardar o último nome utilizado para designar o ficheiro onde é guardada a matriz, nome esse que terá no máximo 80 caracteres.

3. Dados de Entrada

O programa deverá ler os dados de entrada a partir da linha de comandos e do terminal. A linha de comandos permite a indicação *opcional* do nome do ficheiro de entrada com os elementos da matriz a ler.

```
./proj1
./proj1 <filename>
```

Durante a execução do programa as instruções devem ser lidas do terminal (standard input) na forma de um conjunto de linhas iniciadas por um carácter, que se passa a designar por *comando*, seguido de um número de informações dependente do comando a executar; o comando e cada uma das informações são separados por um espaço.

Os comandos disponíveis são descritos de seguida e correspondem às letras `a`, `p`, `i`, `l`, `c`, `o`, `z`, `s`, `w`, `q`. 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[1]:

`a`: adiciona um novo elemento à matriz (n.i.l. = 3)[2]

`a linha coluna valor`

- `linha`: número inteiro não negativo;
- `coluna`: número inteiro não negativo;
- `valor`: número real em precisão dupla.

Output: Não tem qualquer output.

Adiciona o elemento, `valor` na referida posição da matriz. Se já existir um elemento nessa posição, este deverá ser substituído pelo novo `valor`.

Nota: apesar de se poder adicionar o elemento *zero* à matriz, este não deverá ficar representado. Assume-se que no início o elemento *zero* é o número 0.
Nota2: Apesar de não existir remoção explícita de elementos de uma matriz, esta pode ser efectivamente realizada através da introdução do elemento *zero* na posição a remover.

[1] Se encontrar dificuldades, veja p.f. o exemplo de linha de comandos dado nas aulas teóricas.
[2] Poderá criar um vector de estruturas "valor" onde guarda a `linha`, a `coluna`, e o `valor`.

`p`: lista todos os elementos representáveis da matriz (n.i.l. = 0)

```
p
```

Output: Lista todos os elementos representáveis da matriz, ie, não *zero,* pela ordem pela qual foram introduzidos (ou pela ordem actual caso o comando `o` abaixo já tenha sido executado), com a forma

```
[<linha>;<coluna]=<valor>
```

e onde `<valor>` deverá ser impresso com 3 casas decimais (substituir `<linha>`, `<coluna>` e `<valor>` pelos respectivos elementos).

Caso não exista nenhum elemento na matriz deverá imprimir

```
empty matriz
```

`i`: imprime as caracterísiticas da matriz (n.i.l. = 0)

```
i
```

Output: Imprime uma só linha com

```
[<mini> <minj>] [<maxi> <maxj>] <nelem> / <size> = <dens>%
```

onde `<mini>`, `<minj>`, `<maxi>`, `<maxj>` são os menores e maiores índices de todos os elementos representados, ou seja os limites da matriz; `<nelem>` é o número de elementos efectivamente representados e `<size>` é a dimensão da *matriz completa*[3]. A densidade `<dens>` é o rácio entre `<nelem>` e `<size>` e deverá ser representada em percentagem com 3 casas decimais.

`l`: listar uma linha (n.i.l. = 1)

```
l linha
```

- `linha`: número inteiro não negativo;

Output: Lista a linha completa `linha` da matriz, incluindo os elementos *zero*, numa só linha do terminal. Cada valor (de `<minj>` a `<maxj>`) deverá ser precedido de um espaço em branco e impresso com 3 casas decimais. Se só houver elementos *zero* na linha, ou a linha indicada estiver fora dos limites, deverá ser impressa a mensagem

```
empty line
```

[3] Uma matriz diz-se *completa* se todas as posições de todas as colunas (entre `<minj>` e `<maxj>`) de todas as linhas (entre `<mini>` e `<maxi>`) estiverem representadas.

`c`: listar uma coluna (n.i.l. = 1)

```
c coluna
```

- `coluna`: número inteiro não negativo;

Output: Lista a coluna completa da matriz, incluindo os elementos *zero*. Cada valor (de `<mini>` a `<maxi>`) deverá ser impresso numa linha do terminal com o formato do comando `p`. Se só houver elementos *zero* na coluna, ou a coluna indicada estiver fora dos limites, deverá ser impressa a mensagem

```
empty column
```

`o`: ordena (n.i.l. = 0 ou 1)

```
o
o column
```

Output: não tem output.
Por omissão, sem indicar parâmetros, ordena por ordem crescente do número de linha e, dentro de cada linha, por ordem crescente do número de coluna. Se for indicado o parâmetro `column`, ordena por ordem crescente de número de coluna e, dentro de cada coluna, por ordem crescente do número de linha.

Observações: a ordem dos elementos armazenados é alterada pelo que posteriores listagens (comando `p`) refletem a nova ordenação.

`z`: redefine o valor considerado como *zero* na matriz (n.i.l. = 1)

```
z valor
```

- `valor`: número real em precisão dupla.

Output: Não imprime output.

A definição de um novo valor *zero*, implica que todos elementos representados com valores iguais ao novo valor *zero* deverão ser eliminados.

`w`: salvaguarda a lista de elementos representados num ficheiro (n.i.l. = 0 ou 1)

```
w
w filename
```

- `filename`: nome do ficheiro a utilizar.

O nome do ficheiro pode ser omitido caso já tenha sido definido por um comando `w` anterior ou caso tenha sido indicado na linha de comandos. O ficheiro será criado, caso não exista, ou re-criado, caso já exista.

Output: Não gera output.

`s`: comprime a matriz (n.i.l. = 0)
`s`

Output: Imprime uma linha de valores, uma linha de índices, e uma linha de offsets, que representa a compactação da matriz esparsa de acordo com o algoritmo que se segue.

```
value=<valor1> <valor2> … <valorn>
index=<index1> <index2> … <indexn>
offset=<offset1> <offset2> … <offsetm>
```

Caso a densidade seja superior a `0.5` imprime a mensagem

```
dense matrix
```

A compressão por *double-offset indexing* representa uma matriz esparsa por três vectores:

1) um vector (`value`) contendo todos os valores representáveis da matriz;
2) um vector (`index`) contendo os índices das linhas correspondentes a cada um desses elementos; e
3) um vector de linhas (`offset`) que indica a posição de início de cada linha nos outros dois vectores, ie, o correspondente *offset*.

Na compressão óptima o vector de valores não tem elementos nulos, logo tem o comprimento correspondente ao número de elementos não nulos. No exemplo abaixo a compressão **NÃO É** óptima.

O algoritmo de compressão a utilizar neste projecto é o seguinte:

a) Começando pela linha de *maior densidade* colocar todos os elementos representáveis dessa linha no vector 1) começando na coluna 0, e mantendo a posição relativa dos elementos. No caso de existir mais do que uma linha com a mesma densidade começar pela de menor índice.
b) Preencher as mesmas posições no vector 2) com o *índice* da linha em causa.
c) Marcar o *offset* na posição correspondente a essa linha no vector 3).

   Repetir para todas as linhas seguindo por ordem decrescente de densidade. Caso não seja possível executar o passo a), tentar começando na coluna/offset 1, etc.

d) No final do algoritmo, todas as linhas terão de estar totalmente representadas, ie, todas as suas posições, pelo que poderá ser necessário adicionar *padding* de *zero*.
e) As posições do vector 1) que não tenham sido utilizadas deverão terminar com o valor *zero*, e o respectivo vector 2) será preenchido com o algarismo 0.

Exemplo: Considere a matriz

	Coluna 4	Coluna 5	Coluna 6	Coluna 7	Coluna8
Linha 5		5.5	5.6		
Linha 6				6.7	
Linha 7	7.4	7.5			7.8

A linha de maior densidade neste caso é a 7 pelo que esses elementos são colocados em primeiro lugar no vector 1) e a respectiva linha no vector 2). O *offset* neste caso é 0.

7.4	7.5			7.8	
7	7			7	

	Linha 5	Linha 6	Linha 7
Offset			0

Seguidamente é a vez da linha 5. Esta já não pode ser colocada directamente no vector 1) pois a 2ª posição já contém o valor 7.5. Mas esta linha pode ser colocada correctamente se começarmos na posição 1 do vector 1) em vez de na posição 0. O *offset* neste caso é 1.

7.4	7.5	5.5	5.6	7.8	
7	7	5	5	7	

	Linha 5	Linha 6	Linha 7
Offset	1		0

Finalmente a linha 6 pode ser colocada no vector 1) se começarmos na posição 2 em vez de começarmos em 0. O *offset* neste caso é 2.

7.4	7.5	5.5	5.6	7.8	6.7
7	7	5	5	7	6

	Linha 5	Linha 6	Linha 7
Offset	1	2	0

Nota: A linha 6 não se encontra totalmente representada: começa no offset 2 e seriam necessário representar os seus 5 elementos, só havendo possibilidade de representar 4. Neste caso, é necessário acrescentar o *padding* de *zeros* para esta linha. O índice a usar no vector de índices será o algarismo 0.

A compressão neste caso corresponderia então aos 3 vectores abaixo

7.4	7.5	5.5	5.6	7.8	6.7	*zero*
7	7	5	5	7	6	0

	Linha 5	Linha 6	Linha 7
Offset	1	2	0

q: abandona a aplicação (n.i.l. = 0)

```
q
```

Output: Não imprime output.

Ao sair da aplicação nenhuma informação é guardada.

4. 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 anteriormente 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.

5. Exemplos (Input/Output)

Exemplo 1 (./proj1)

Dados de Entrada:	**Dados de Saída:**

`a 9 9 9`
`a 6 6 6`
`p`
`i`
`o`
`p`
`q`
	`[9;9]=9.000`
	`[6;6]=6.000`
	`[6 6] [9 9] 2 / 16 = 12.500%`
	`[6;6]=6.000`
	`[9;9]=9.000`

Exemplo 2 (./proj1)

Dados de Entrada:	**Dados de Saída:**

`p`
`a 0 3 1.2`
`a 3 0 2.1`
`i`
`w mat.in`
`l 0`
`l 3`
`a 4 5 2.2`
`a 2 5 3.2`
`a 3 5 4.2`
`c 5`
`p`
`o`
`p`
`a 0 0 5.2`
`p`
`w`
`q`
	`empty matriz`
	`[0 0] [3 3] 2 / 16 = 12.500%`
	`0.000 0.000 0.000 1.200`
	`2.100 0.000 0.000 0.000`
	`[0;5]=0.000`
	`[1;5]=0.000`
	`[2;5]=3.200`
	`[3;5]=4.200`
	`[4;5]=2.200`
	`[0;3]=1.200`
	`[3;0]=2.100`
	`[4;5]=2.200`
	`[2;5]=3.200`
	`[3;5]=4.200`
	`[0;3]=1.200`
	`[2;5]=3.200`
	`[3;0]=2.100`
	`[3;5]=4.200`
	`[4;5]=2.200`
	`[0;3]=1.200`
	`[2;5]=3.200`
	`[3;0]=2.100`
	`[3;5]=4.200`
	`[4;5]=2.200`
	`[0;0]=5.200`

Exemplo 3 (./proj1 mat.in)

Dados de Entrada:	Dados de Saída:
p z 1.2 p i w q	[0;3]=1.200 [2;5]=3.200 [3;0]=2.100 [3;5]=4.200 [4;5]=2.200 [0;0]=5.200 [2;5]=3.200 [3;0]=2.100 [3;5]=4.200 [4;5]=2.200 [0;0]=5.200 [0 0] [4 5] 5 / 30 = 16.667%

Exemplo 4 (./proj1 mat.in) – em sequência do exemplo 3

Dados de Entrada:	Dados de Saída:
p a 4 2 1.6 i s q	[2;5]=3.200 [3;0]=2.100 [3;5]=4.200 [4;5]=2.200 [0;0]=5.200 [0 0] [4 5] 6 / 30 = 20.000% value = 2.100 5.200 0.000 1.600 0.000 4.200 2.200 3.200 index = 3 0 0 4 0 3 4 2 offset = 1 0 2 0 1

6. 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).

7. Execução do Programa

O programa deve ser executado da forma seguinte:

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

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

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

7.1. Testes Auxiliares

Para testar o seu programa poderá executar os passos indicados acima ou usar os scripts **run.sh** e **runall.sh** distribuídos no ficheiro **Exemplos.zip**.

Se quiserem executar apenas o **teste01.in** deverão executar

```
$ ./run.sh <vosso_ficheiro_c> teste01.in
```

Para executarem todos os testes deverão executar

```
$ ./runall.sh <vosso_ficheiro_c>
```

Estes scripts compilam o ficheiro indicado e comparam o resultado obtido com o resultado esperado. Se apenas indicar o tempo de execução é porque o comando **diff** não encontrou nenhuma diferença. Caso indique mais informação, então é porque o resultado obtido e o resultado esperado diferem. Para obter a informação detalhada das diferenças poderá remover a opção **-q** da linha 10 do ficheiro **run.sh**.

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 `.zip` que inclua todos os ficheiros fonte que constituem o programa. Se o seu código tiver apenas um ficheiro o zip conterá apenas esse ficheiro. Se o seu código estiver estruturado em vários ficheiros (`.c` e `.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. **Tenha especial atenção a este facto na altura da submissão final.** Exemplos de casos de teste serão oportunamente fornecidos.

- Data limite de entrega do projecto: **27 de Abril de 2017 (23h59m)**. 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.

9. 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, indentação, 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 posteriormente.

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.