DEEC
DEPARTAMENTO DE ENGENHARIA ELETROTÉCNICA E DE COMPUTADORES
TÉCNICO LISBOA

Licenciatura em Engenharia Electrotécnica e de Computadores (LEEC)

Algoritmos e Estruturas de Dados
Aula de Laboratório #04 - Laboratório de Avaliação Individual

Objectivos

Este laboratório consiste numa avaliação prática **individual** e será realizado no laboratório da disciplina, obrigatoriamente na área de aluno, utilizador `aed`.

A avaliação envolve a resolução de **três problemas** cuja descrição formal se segue. Nalguns casos o problema é apenas parcialmente apresentado aqui, sendo a descrição completa disponibilizada no momento da avaliação. O objectivo da divulgação do enunciado (completo ou parcial) dos problemas é dar aos alunos tempo de preparação e tornar a avaliação mais justa e independente da altura em que cada aluno é avaliado.

1 Complexidade

Considere a função abaixo, `func_do()` que recebe um vector `vec` de inteiros, e dois índices para o mesmo e que efetua determinado tipo de processamento sobre os dados.
A função é inicialmente chamada da seguinte forma:

```
func_do(vec, 0, N-1, 1)
```

Indique, justificadamente, a complexidade da execução do código em função de N (para simplificar, assuma que N é uma potência de 2).

```
int g(x, d)  {
  if (d == 1) return(x-1);
  return (x/d);
}

void func_do (int *vec, int iL, int iR, int div) {
  int i, j, iM = (iR - iL) / 2;

  for (i = iR; i >= iL; i = g(i, div)) {
    for(j = i; j >= iL; j--) {

      if (vec[j] > vec[iM])
        vec[i] = (vec[i] + vec[j])/2;
      else
        vec[j] = (vec[i] + vec[j])/2;
    }
  }

  return;
}
```

Nota: o código apresentado é meramente ilustrativo. Na avaliação, para cada aluno, serão apresentados códigos diferentes!!

2 Ordenação

Considere a seguinte tabela de números inteiros. Indique o estado da tabela após todos os passos utilizados pelo algoritmo de [redacted].

9	22	5	1	16	11	14	20	3	8	19

3 Matrizes Esparsas

Neste problema vamos desenvolver uma implementação eficiente para armazenar e processar matrizes esparsas. Uma matriz é dita esparsa quando tem estruturalmente uma elevada percentagem de zeros. Muitos problemas na vida real são modelados por representações esparsas e a sua solução envolve a manipulação de matrizes esparsas. Armazenar todos os elementos de uma matriz esparsa é muito ineficiente, tanto em termos de espaço (a maior parte dos elementos têm valor zero) como computacionalmente (a maior parte das operações que envolvem esses zeros são desperdiçadas porque o resultado pode ser conhecido a-priori; por exemplo o resultado da multiplicação de qualquer número por zero ou da soma de zero com qualquer número é conhecido mesmo sem ter feito a respetiva operação).

Neste problema propomos uma implementação muito simples mas eficiente para matrizes esparsas. Considere a matriz de Laplace, a uma dimensão, seguinte (matrizes de Laplace são usadas para representar muitos problemas reais):

$$\begin{bmatrix} 2 & -1 & 0 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & -1 & 2 & -1 \\ 0 & 0 & 0 & -1 & 2 \end{bmatrix} \tag{1}$$

Apesar de ser uma matriz de $5x5$, apenas 13 elementos são diferentes de zero. Obviamente uma forma de representar computacionalmente esta matriz é armazenando todos os elementos, quer sejam 0 ou não. Outra é, por exemplo, usar a seguinte representação que consiste em 3 tabelas como indicado:

val =	2	-1	-1	2	-1	-1	2	-1	-1	2	-1	-1	2
col =	0	1	0	1	2	1	2	3	2	3	4	3	4

rowSt =	0	2	5	8	11	13

Nesta representação os vetores `val` e `col` têm dimensão igual ao número de elementos não zero da matriz, enquanto que o vetor `rowSt` tem dimensão igual ao número de linhas da matriz mais uma unidade (usado por conveniência de indexação).

No exemplo acima pode ver-se que o vetor `val` contém todos os elementos não zero da matriz do exemplo anterior, lidos linha a linha. Ou seja, na 1ª linha há dois elementos não zero, o 2 e o -1 que ocupam as duas primeiras posições de `val`. Como esses dois elementos estão nas duas primeiras colunas, a coluna 0 e a coluna 1 (usamos, como em "C", a notação que os indíces começam em 0), essa informação está nas duas primeiras posições do vetor `col`. Dado que na 1ª linha não há mais elementos diferentes de zero, nada mais é guardado e na posição 2 do vetor `val` começa a 2ª linha que tem 3 elementos, -1, 2 e -1, respetivamente nas colunas 0, 1 e 2. Como a 2ª linha começou na posição 2 do vetor *val*, essa informação é guardada em `rowSt[1]`.

Visto agora de outra forma, se quisermos saber que elementos da matriz na 4ª linha são diferentes de zero, vamos a `rowSt[3]` (a 4ª coluna tem índice 3!) e vemos que a indicação é que `rowSt[3] = 8`, ou seja os valores não zero da 4ª coluna começam no índice 8. Vendo que `rowSt[4] = 11`, a 5ª e última coluna da matriz começa na posição 11. Ou seja todos os elementos entre as posições 8 e 10 (três elementos portanto) estão na 4ª linha. Se os procuramos no vetor `val` vemos que esses valores são -1, 2 e -1 que estão respetivamente nas colunas 2, 3 e 4 informação essa que está nas posições 8 a 10 do vetor `col`.

Esta representação é naturalmente tão mais eficiente quanto maior e mais esparsas forem as matrizes porque o *overhead* de guardar a informação de linhas e colunas é rapidamente compensado pela quantidade de elementos de valor zero que se evita guardar.

No problema que vamos resolver, a implementação a utilizar neste problema é assim suportada na estrutura básica de matriz esparsa definida da seguinte forma:

```
typedef struct _SpMat {
  int nrow, ncol, nnz;
  double *val;
  int *col;
  int *rowSt;
} SpMat;
```

em que `nrow` é o número de linhas da matrix, `ncol` o número de colunas, `nnz` o total de elementos não zero na matriz, `val` o ponteiro para o vetor contendo os elementos não zero da matriz, `col` o ponteiro para o vetor com a informação das colunas em que cada um desses elementos reside e `rowSt` o ponteiro para o vetor que indica o índice em que se inicia a informação de cada uma das linhas da matriz em `val` e em `col` (por conveniência, `rowSt` tem um elemento adicional no fim igual ao número total de não zeros).

No ficheiro `p3.c`, disponível no laboratório na máquina de cada aluno, foi implementado um interface simples para manipulação com matrizes esparsas e em `SpMat.c`/`SpMat.h` um conjunto de funcionalidades sobre matrizes esparsas. O programa `p3` é invocado com um ou dois argumentos. Se for invocado com um argumento, deve ser o nome de um ficheiro que contém a informação sobre uma matriz esparsa. O formato desse ficheiro é muito simples, na 1ª linha tem 3 números, respetivamente o número de linhas, de colunas e de elementos não zero da matriz. As linhas restantes contêm a informação dos elementos não zero, cada um indicando a linha, coluna e valor do elemento. O código dado carrega uma matriz descrita desta forma textual para uma estrutura do tipo `SpMat` indicada. Se invocado com dois argumentos, o 2º argumento será o nome de um ficheiro com os dados de um vetor de dimensão apropriada para a operação pretendida (ver mais abaixo).

O interface permite fazer algumas operações sobre os dados de entrada, nomeadamente calcular valores máximos ou mínimos, valores positivos ou negativos, normas, informações sobre dominância da diagonal, produtos matriz vetor, etc. Algumas dessas funcionalidades estão já implementadas, total ou parcialmente, outras serão adicionadas neste problema.

Nota: as perguntas seguintes são apenas ilustrativas. Na avaliação serão feitas perguntas do mesmo tipo mas potencialmente diferentes, distintas para cada horário de laboratório!!.

Com este enunciado está disponível algum código de apoio (que também estará disponível no laboratório) que os alunos deverão analisar antes de desenvolverem as funcionalidades pedidas.

3.1. Complete o código da função

`SpMat *readSpMatFile(FILE *fp, char *fname)` ,

que dado um ficheiro com informação textual sobre a matriz como indicado acima, cria e preenche essa informação numa estrutura `SpMat`. O código apresentado já faz a lógica de carregamento dos dados, faltando apenas as instruções de alocação de memória para as tabelas da estrutura da matriz.

3.2. Complete o código da função `void sumLinesSpMat(SpMat *spmat)` que dada uma matriz esparsa imprime para o écran a soma dos elementos de cada linha da matriz.

3.3. Complete o código da função `double SpMatXXX(SpMat *spmat)` que dada uma matriz esparsa, calcula ... (**a ver no laboratório mas terá a ver com a manipulação dos valores contidos nesta estrutura de dados**)