DEEC
DEPARTAMENTO DE ENGENHARIA ELETROTÉCNICA E DE COMPUTADORES
TÉCNICO LISBOA

AED - 2019-2020 - 1^o Semestre
Algoritmos e Estruturas de Dados

Laboratório de Avaliação Individual

Semana de 21 a 25 de Outubro de 2019 Duração: 1h30mn

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. Noutros casos é apresentado um problema tipo que poderá ser modificado no momento da avaliação. O objectivo da divulgação do enunciado (completo ou parcial) dos problemas é tornar a avaliação mais justa e independente da altura em que cada aluno é avaliado.

---

1. Considere a função abaixo, `process_vec()` que implementa um algoritmo recursivo para processar os elementos de uma tabela de inteiros `vec[ ]`, de tamanho `N`, retornando essa informação numa segunda tabela, `ret[ ]`.

A função é inicialmente chamada da seguinte forma:

`process_vec(vec, ret, 0, N)`

a) Escreva, justificando, uma recorrência que descreva a complexidade computacional do algoritmo em função de `N`. Assuma que o custo computacional de cada chamada à função auxiliar `comp_dist( )` é da ordem de $\mathcal{O}(1)$.

b) Resolva a recorrência e determine essa complexidade em função de `N`.

```
void process_vec(int *vec[], int *ret[], int left, int right) {
    int i, d = 0, index;

    if ( left == right ) return;
    for (i = left; i < right; i++) {
      d = comp_dist(vec, left, i, right); /* local computation */
      if ( d > (vec[left] + vec[right])/2 ) {
         index = i;
      }

    ret[index] = 1;
    process_vec(vec, ret, left, index);
    process_vec(vec, ret, index, right);
    return;
}
```

Nota: o código indicado é apenas ilustrativo. Na avaliação, para cada aluno, será dado código ligeiramente diferente e pedido para fazer o mesmo tipo de análise.

2. Em `p2-pub.c` é disponibilizado um código, incompleto, que recebe como argumento o nome de um ficheiro. Esse ficheiro deve conter $M * N + 2$ números inteiros, distribuídos por uma ou mais linhas, sem qualquer formatação específica. O primeiro inteiro a ser lido indicará o valor de M e o segundo número o valor de N. A esses dois números segue-se uma sequência de outros $M * N$ números inteiros, positivos ou negativos (ou seja, se o primeiro inteiro for, por exemplo, 4 e o segundo número for 3, haverá mais $4 * 3 = 12$ inteiros no ficheiro). O código deve ler os dois primeiros números, alocar espaço para uma matriz $M * N$ (M linhas e N colunas), ler sucessivamente os elementos da matriz, linha a linha, e armazenar no espaço anteriormente alocado. Os dados devem a partir daqui ser acessíveis a partir da variável `data` (ver código em `p2-pub.c`).

De seguida deve ser chamada a função `func(...)` para processar os dados e, após o retorno da função, deve ser colocada no terminal a informação desejada.

Nota: a funcionalidade pretendida para a função `func(...)` será indicada na avaliação e será diferente para cada aluno/a. O texto que se segue é nesse aspecto meramente ilustrativo.

No final da execução, e antes de terminar, deverá ser correctamente libertada a memória entretanto alocada.

2.1. **Complete o código** incluindo as instruções necessárias para ler os argumentos de chamada e para alocação e libertação de memória. Teste o seu código com alguns dos exemplos fornecidos ou outros que construa. Verifique com `valgrind` que não há problemas/erros de memória.

2.2. Nesta alínea implemente em `func()` um algoritmo que calcule o máximo dos valores médios dos elementos de cada linha da matriz. A função deve retornar esse valor que deverá então ser impresso no terminal.

2.3. Indique, **justificando**, a complexidade do programa por si criado, em função de N (assuma nesta análise que $M = N$), seja em termos de memória, seja em termos computacionais. Apresente o seu resultado utilizando a notação $\mathcal{O}(\ldots)$ estudada.

---

3. Um disco é um dispositivo onde pode ser guardada informação a utilizar num sistema computacional. A organização da informação num disco depende do sistema de ficheiros implementado e os ficheiros em si estão tipicamente organizados de forma hierárquica, existindo para cada um deles uma estrutura que toda a informação sobre o ficheiro. Nos sistemas de ficheiros Unix esta informação está contida numa estrutura denominada *inode* (ou *index-node*) que descreve os objectos do sistema de ficheiros, sejam eles ficheiros (*files*) propriamente ditos, ou pastas (*directories*). Um *inode* contém diversos campos para guardar a informação relevante de cada ficheiro, nomeadamente o tipo de ficheiro, o seu nome/identificação, data de criação ou modificação, a quem pertence, que permissões de acesso foram concedidas, qual a sua dimensão, etc. Essa estrutura contém também informação, ponteiros, que permitem aceder aos dados contidos nesse ficheiro. Estes estão tipicamente organizados em blocos de dimensão fixa, possivelmente dispersos pelo disco, sendo o seu número variável dentro de certos limites impostos pelo sistema de ficheiros.

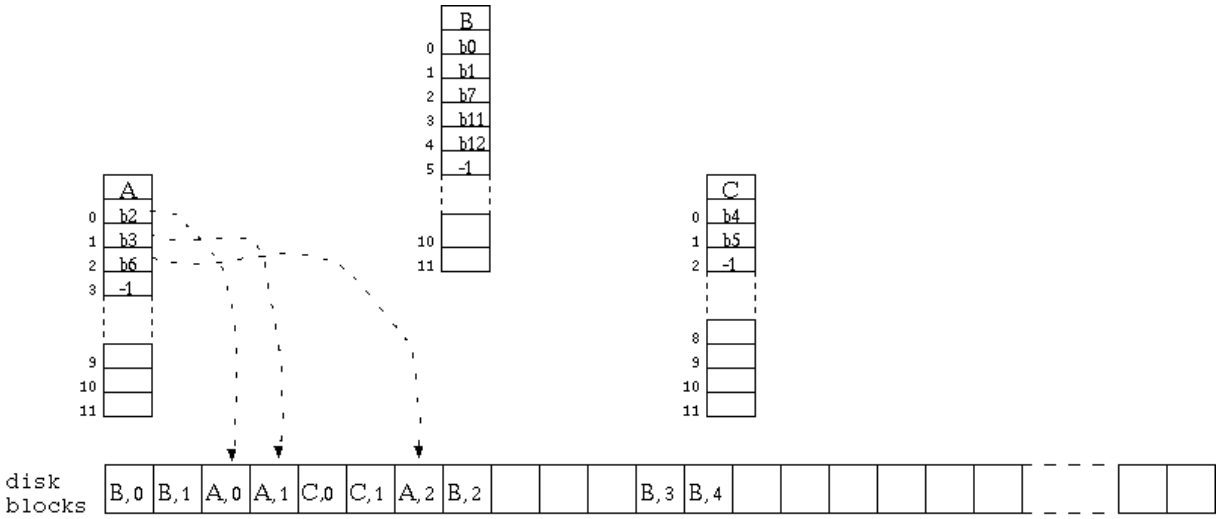

Figura 1: Representação gráfica da configuração dos blocos do disco assumindo que existem 3 ficheiros, A, B e C. Por simplicidade mostra-se graficamente o disco como uma sequência de blocos embora computacionalmente não se represente o disco desta forma.

Nos sistemas originais Unix, a estrutura *inode* contém 12 ponteiros para 12 blocos de dados. Estes ponteiros eram suficiente para endereçar os blocos de dados de ficheiros de dimensão pequena. Para ficheiros de dimensões superiores, ponteiros adicionais para além dos 12 indicados permitem, por endereçamento indirecto, aceder a um número de blocos de dados muito superior. Para mais infromações veja `https://en.wikipedia.org/wiki/Inode` ou `https://en.wikipedia.org/wiki/Inode_pointer_structure`.
Neste problema vamos emular de forma muito simplificada algumas operações relacionadas com a estrutura de um sistema de ficheiros baseados num *inode* muito simplificado. Vamos assumir que a informação contida no *inode* inclui apenas um identificador para o ficheiro (para simplificar assumimos que o nome de cada ficheiro é apenas uma letra) e um máximo de 12 ponteiros para endereçamento direto de blocos de dados (não existem portanto ponteiros para endereçamento indirecto). Vamos além do mais, simplificadamente assumir que todos os blocos de dados de todos os ficheiros se encontram numa zona contigua do disco de dimensão `N` (embora os blocos de cada ficheiro se possam encontrar dispersos). Isto permite endereçar os blocos de dados diretamentepor indexação. Cada ficheiro é assim constituido por vários blocos localizados no disco e o seu *inode* permite saber em que blocos do disco estão os seus blocos de dados.
A Figura 1 exemplifica graficamente uma possível representação do espaço de dados dos ficheiros em disco, bem como a estrutura de um disco nomeadamente os seus blocos de dados.

A representação baseia-se nos seguintes tipos de dados e variáveis:

```
typedef struct _easy_iNode {
  char fileName;
  /* additional file info here */
  int blk[12];
} eiNode;

typedef struct _BlockLoc {
  eiNode *owningFile;
  int fileblkNumber;
  /* save disk location here */
} BlockLoc;

BlockLoc diskInfo[N]; /* table of disk blocks for by all files */
LinkedList *fileList; /* list of files in filesystem */
```

em que `eiNode` pretende representar de forma muito simplificada a informação de cada ficheiro no sistema de ficheiros (uma espécie de *iNode* muito simplificado ignorando quase todas as características de cada ficheiro). Por seu lado `BlockLoc` representa uma estrutura contendo informação de um bloco de dados de um dado ficheiro, nomeadamente a sua localização no disco. A informação da ocupação de blocos no disco é implementada através da tabela `diskInfo`, e a lista de ficheiros no disco é acessível por `fileList`.
Neste problema vamos implementar algumas funções para manipular e operar com ficheiros descritos desta forma. O código de suporte deste tipo de representação é disponibilizado em `filesys.c` e `filesys.h` e está propositadamente incompleto. Disponibiliza-se igualmente um código em `p3-pub.c` onde, se incluem funções de teste que permitem carregar a informação do disco com uma determinada configuração de ficheiros, instanciar comandos para apagar ficheiros, criar novos ficheiros, etc.

O programa `p3-pub` deve ser executado da seguinte forma:

```
$ ./p3-pub N diskSnapshot.txt
```

em que o primeiro argumento é a dimensão N, em número de blocos de dados, do disco (i.e. existirão no disco blocos de dados nas posições 0 a $N-1$). O segundo argumento é o nome de um ficheiro contendo informação sobre um hipotético estado do sistema de ficheiros para sobre ele se poderem fazer computações e análises diversas. Cada uma das linhas desse ficheiro contém informação sobre um dado ficheiro no nosso sistema de ficheiros. A linha inicia-se com uma letra que é o identificador do ficheiro (todos os ficheiros têm identificadores diferentes). Seguem-se uma sequência de indicadores do posicionamento sucessivo dos blocos de dados desse ficheiro no disco. Cada bloco é identificado com um número inteiro entre 0 e $N-1$ e a linha é terminada com o identificador -1. Ao ler cada uma das linhas o estado dos blocos do disco pode ser reconstruido na tabela `diskInfo`.
Após ler o ficheiro o programa deve ler um comando entrado pelo utilizador. Estão disponíveis os comandos 'd' para apagar um ficheiro, 'p' para imprimir o estado do disco, 'c' para apagar todos os ficheiro e 'e' para sair do programa. Mais comandos poderão ser adicionados posteriormente.
Para testar o seu código assuma por exemplo $N = 60$ e utilize um dos ficheiros `snapX.txt` disponibilizados.

3.1. Complete o código da função `filesysLoadFile()` em `p3-pub.c` que lê do ficheiro de dados a informação do posicionamento dos blocos de um ficheiro no sistema de ficheiros e actualiza a informação da lista `diskInfo`.

3.2. Complete o código da função `filesysDeleteFile()` que dado o identificador de um ficheiro no sistema de ficheiros, procura esse ficheiro na lista de ficheiros, liberta o `eiNode` desse ficheiros e liberta no disco os blocos ocupados por esse ficheiro, atualizando a informação disponível através de `diskInfo`.

3.3. Escreva o código de ... / calcule ... / analise ...

Na avaliação, na alínea 3.3. será pedido a cada aluno que acrescente código para adicionar uma nova funcionalidade ou fazer determinado tipo de cálculos. Os alunos deverão analisar o código disponibilizado para este problema antes de desenvolverem as funcionalidades pedidas.