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 permitir aos alunos um 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, `fake_ordering()` 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:

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

Escreva a recorrência que descreve a complexidade C_N da execução do código em função de N (para simplificar, assuma que N é uma potencia de 2). `Justifique` brevemente a fórmula que escreveu.

```
#define swap(a, b) {int t; t = a; a = b; b = t;}

void fake_ordering (int* vec, int iL, int iR) {
   int tmp;
   int iM = (iR - iL) / 2;

   if (vec[iL] > vec[iM])
      swap(vec[iL], vec[iM]);

   if (vec[iM] > vec[iR])
      swap(vec[iM], vec[iR]);

   if (vec[iL] > vec[iM])
      swap(vec[iL], vec[iM]);

   fake_ordering(vec, iL, iM);
   fake_ordering(vec, iM + 1, iR);
   return;
}
```

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

2 Alocação e Libertação de Memória

O excerto de código seguinte, que está incompleto, (disponível em `p2.c`) recebe como argumento um nome de um ficheiro. O ficheiro deve conter inicialmente um número inteiro, N. De seguida é reservado espaço para uma matrix de dimensão $N \times N$. Essa matriz é depois preenchida, linha a linha, com números inteiros lidos do ficheiro (que deve portanto conter um total de $N^2 + 1$ números). De seguida é chamada a função `check_property()` que efectua algum tipo de computação sobre os elementos da matriz (ou sobre os elementos de uma dada linha da matriz) cujo resultado é depois impresso.

2.1. **Complete o código** da função `main()` incluindo as instruções necessárias para abrir o ficheiro e para alocação e libertação de memória (o `valgrind` não deve indicar nenhum erro ou aviso).

2.2. **Complete o código** da função `check_property()` de forma a esta calcular ...
Nota: No dia da avaliação será pedido a cada aluno a verificação ou cálculo de uma dada propriedade. Um exemplo seria determinar o maior elemento de cada linha.

2.3. Indique, **justificando**, a complexidade do programa em função de N.

Nota: No dia da avaliação, o código estará disponível na máquina de cada aluno para ser completado de acordo com o pedido.

```
...
int check_property (int *..., ...)  {
   ...
   return(...);
}


int main (int argc, char *argv[]) {
   FILE *fp;
   int i, j, N, result = 0, **vec;

   /* check arguments; omitted here to save space */
   ...
   /* open file and read N, partially ommitted here to save space */
   fp = fopen(argv[1], "r");
   result = fscanf(fp, "%d", &N);

   /* allocate memory, read in the array and print it */
   vec = ???  malloc(???);
   ???

   for (i = 0; i < N; i++)
      for (j = 0; j < N; j++)
         fscanf(fp, "%d", &vec[i][j]);

   /* Now process array */
   for (i = 0; i < N; i++) {
      result = check_property(vec[i], 0, N-1); /* this is an example */
      printf("result[line %d]:  %d\n", i, result);
   }

   /* free memory */
   ...;
   exit(0);
}
```

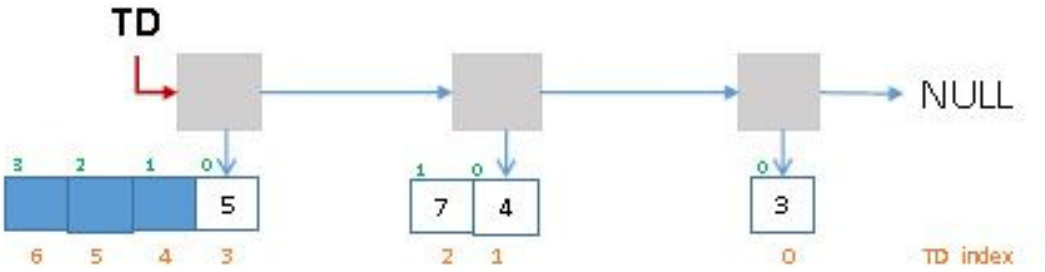
3 Vetores Dinâmicos

Neste problema propõe-se um mecanismo para representar tabelas (vectores) cujo tamanho pode ser variável e função das necessidades de armazenamento (por exemplo, para implementar pilhas). É sabido que uma representação em tabela pressupõe que se conheça qual o volume máximo de dados que é suposta armazenar. Quando tal não se pode assegurar, a alternativa é usar listas. No entanto, na utilização de listas é impossível fazer uso de álgebra de ponteiros para aceder aos dados nela constantes.

O mecanismo que se propõe é, na sua essência, uma lista de tabelas que podem ser vistas como segmentos de uma única tabela. Se, num dado momento, todas as tabelas dos vários segmentos ou elementos da lista estiverem cheias e for necessário adicionar um novo elemento, há que alocar um novo segmento para a lista que por sua vez contém uma nova tabela.

Define-se como **tabela dinâmica** (ou tabela em segmentos) uma lista de tabelas em que cada tabela (cada segmento) adicionada tem o dobro do tamanho da sua antecessora imediata. Por exemplo, uma tabela dinâmica em que a lista que a representa possua quatro elementos (quatro segmentos) serve para armazenar um máximo de 15 objectos. No primeiro elemento da lista há uma tabela de tamanho 8, no segundo uma tabela de tamanho 4, no terceiro uma tabela de tamanho 2 e no quarto, e último, uma tabela de tamanho 1. Se esta tabela dinâmica vier a ficar cheia, tornar-se-á necessário adicionar uma nova tabela de tamanho 16, que será adicionada no início da lista.

No exemplo abaixo, se os mesmos dados fossem representados numa tabela estática, de nome `a`, o valor 3 seria o `a[0]`, o valor 4 seria o `a[1]`, o 7 seria o `a[2]` e o 5 seria o `a[3]`. Ou seja, a tabela dinâmica é preenchida, em termos de segmentos, do fim para o princípio (a 1^a posição da tabela corresponde ao valor armazenado no último segmento, i.e. o último elemento da lista).



Assuma que o elemento básico da lista é definido de acordo com o que se apresenta abaixo, em que `size` indica o tamanho do segmento da tabela, `free` indica quantos elementos da tabela desse segmento estão vazios, `table` é um ponteiro para uma tabela de inteiros de dimensão `size` e `next` é um ponteiro para o próximo elemento da lista (ou segmento da tabela dinâmica).

```
typedef struct _vec_dyn {
   int size;
   int free;
   int * table;
   struct _vec_dyn *next;
} VecDyn;
```

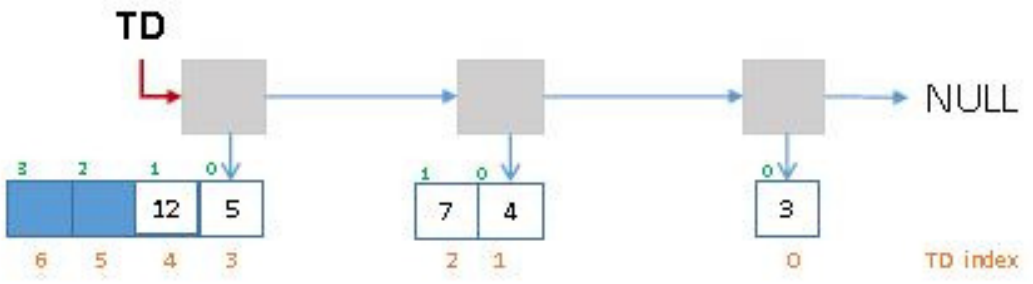
No ficheiro `p3.c`, disponível no laboratório na máquina de cada aluno, foi implementado um interface simples para manipulação com vetores dinâmicos e em `VecDynList.c`/`VecDynList.h` um conjunto de funcionalidades sobre vetores dinâmicos. O programa `p3` é invocado com um argumento, um ficheiro em que em cada linha deve existir um comando e os argumentos correspondentes. Exemplos de comandos disponibilizados incluem:

Comando	Descrição
`i a`	insere o valor `a` na primeira posição livre do vetor
`g k`	retorna o valor armazenada na posição `x` do vetor
`p`	imprime o vetor
`s`	retorna o tamanho (alocado) do vetor
`o`	retorna o número de posições ocupadas no vetor
`x b`	executa a operação ... com argument `b`
`y`	executa a operação `y`

para uma listagem mais completa sugere-se a análise cuidada de `p3.c`. Note-se que embora alguns dos comandos estejam previstos e indicados em `p3.c`, a sua implementação em `VecListDyn` pode estar de momento vazia e ser alvo de avaliação (ver abaixo).

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. Escreva o código da função `VecDyn * insertVecDyn(VecDyn * vecDyn, int value)`, que insere o inteiro `value` na primeira posição livre da tabela. Caso a tabela dinâmica esteja já cheia (ou seja inexistente, i.e. `table == NULL`), deverá ser feita a alocação de memória necessária, de acordo com as regras da tabela dinâmica, acima descritas (i.e. criado um novo segmento da tabela dinâmica). Por exemplo, se a função fosse invocada para a tabela do exemplo acima e para inserir o inteiro 12, esse valor seria colocado na segunda posição da tabela do segmento de dimensão 4, dado que a primeira posição é a única ocupada (ver abaixo).



A função deverá actualizar o valor do campo `free` e deverá devolver o ponteiro para a tabela dinâmica (`TD`). Teste a sua função com alguns exemplos fornecidos ou crie você mesmo/a um exemplo de teste.

3.2. Escreva o código da função `int modifyVecDynValue(VecDyn *vecDyn, int index, int value)`, que coloca o valor `value` na posição de índice dado por `index`, (o índice aqui refere-se à tabela dinâmica, considerando todos os seus segmentos). No exemplo acima, se `index` fosse 2, e o valor a colocar fosse 8, ficaria a[2] = 8. Se a função for invocada com um índice superior ao da última posição ocupada (contando de trás para a frente) ou com um índice negativo, deverá sair com `exit(1)`. Indique, justificadamente, qual a complexidade da função que codificar (se lhe for mais conveniente, faça a análise de pior caso) ou se preferir escreva e resolva uma recorrência que a descreva.

3.3. Adicione código que implemente o comando ... (**a ver no laboratório mas terá a ver com a manipulação dos valores contidos nesta estrutura de dados**)