DEEC
DEPARTAMENTO DE ENGENHARIA ELETROTÉCNICA E DE COMPUTADORES
TÉCNICO LISBOA

AED - 2017-2018 - 1º Semestre
Algoritmos e Estruturas de Dados

Laboratório de Avaliação Individual

Semana de 23 a 27 de Outubro de 2017 Duração: 1h30

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 tornar a avaliação mais justa e independente da altura em que cada aluno é avaliado.

---

1. Considere a função abaixo, `check_data_property()` que recebe uma tabela `vec`, de tamanho N, e determina uma dada propriedade dos elementos contidos na tabela. No exemplo que é mostrado a propriedade testada é determinar quantos elementos repetidos existem na tabela.

A função é inicialmente chamada da seguinte forma:

```
check_data_property(vec, 0, N)
```

Indique detalhadamente, e justifique, qual a complexidade computacional do código apresentado em função da dimensão da tabela, N, assumindo que a chamada à função `munch_data()` tem um custo computacional de $\mathcal{O}(N \log N)$.

```
int check_data_property(int *vec, int iL, int iR) {
  int k, res;

  munch_data(vec, iL, iR);

  for (res = 0, k = iL; k < iR; k++) {
    if (vec[k] == vec[k + 1])
      res++;
  }

  return(res);
}
```

Nota: a propriedade indicada é apenas ilustrativa. Na avaliação, para cada aluno, será dado código que verifica propriedades diferentes utilizando naturalmente código diferente!!

2. O código seguinte, que está incompleto, (disponível em `p2-pub.c`) recebe como argumento dois inteiros M e N, positivos (no código em `p2-pub.c` isso é verificado). De seguida é reservado espaço para M tabelas de dimensão N. Essas tabelas são depois preenchidas uma a uma com números inteiros lidos do teclado (ou de um ficheiro). De seguida é chamada, para cada tabela, uma função `vec_process()`, que permite processar a informação nessa tabela, sendo escrito no écran o resultado dessa chamada. Em `p2-pub.c` está disponível uma possível implementação dessa função que permite testar o seu código. No final toda a memória deve ser libertada.

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.

2.2. Nesta alínea ignore a descrição de `vec_process()` que é disponibilizado em `p2-pub.c`. Indique, **justificando**, a complexidade do programa por si criado, em função de N, considerando os casos em que a complexidade da função `vec_process()` é $\mathcal{O}(\log N)$ ou que é $\mathcal{O}(N)$ ou que é $\mathcal{O}(N \log N)$.

Nota: o código seguinte é apenas ilustrativo. Na avaliação será usado um código ligeiramente diferente, distinto para cada aluno, mas com a funcionalidade básica indicada. Esse código estará disponível na máquina de cada aluno para ser completado de acordo com o pedido.

```
int main(int argc, char *argv[]) {

  int i, j, M, N, result=0, **vec;

  if (argc < 3) {
    fprintf(stderr, "Usage:  %s M N\n", argv[0]);
    exit(1);
  }

  /* read numbers MUST be positive */
  M = atoi(...);
  N = atoi(...);

  /* allocate memory for tables, and read data */
  vec = ???  malloc(???);

  ???

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

  /* Now process array */
  for (i = 0; i < M; i++) {
    result = vec_process(vec[i], 0, N-1);
    printf("result[vector %d]:  %d\n", i, result);
  }

  /* free memory */
  ...;

  exit(0);
}
```

3. A alocação de memória é uma tarefa essencial de qualquer sistema operativo e consiste essencialmente em coordenar e assignar blocos de memória em resposta a pedidos. Tipicamente o sistema de alocação tem disponível um pequeno número de grandes blocos que tem de dividir para satisfazer pedidos para blocos de menores dimensões. Deve naturalmente manter informação sobre os blocos que já atribuiu e que portanto estão a ser utilizados para preservar essa utilização. Deve igualmente garantir que blocos que sejam devolvidos ficam de novo disponíveis para reutilização.

Há muitos algoritmos disponíveis para executar e controlar este processo, com diferentes vantagens e desvantagens. Neste tipo de algoritmos, o sistema de alocação mantém por vezes uma única lista com informação dos blocos disponíveis e alocados, e em cada elemento da lista guarda informação sobre a localização e dimensão de um bloco e o seu estado, isto é, se está disponível ou foi alocado. Para cada pedido de dimensão `N`, o algoritmo procura na lista um bloco com a dimensão desejada, reserva-o, devolve o seu endereço (localização na memória) a quem efectuou o pedido e actualiza a informação da lista. Caso não haja nenhum bloco com a dimensão pretendida é devolvido um código de erro. Se um bloco for previamente alocado for posteriormente libertado e devolvido, o sistema deverá actualizar a informação e, se possível, esse bloco deve ser fundido com o(s) bloco(s) adjacent(es) para formar um bloco disponível de maior dimensão. Como exemplos de algoritmos de alocação incluem-se:

First Fit: a procura por um bloco de dimensão maior ou igual a `X`, para satisfazer o pedido, é feita a partir do início da lista. Assim que for encontrado um tal bloco (de dimensão `N` $\geq$ `X`, ele é partido num bloco do tamanho desejado, que é retornado a quem fez o pedido, e num novo bloco com a parcela de memória restante, `M-X`.

Next Fit: a procura por um bloco de dimensão maior ou igual a `X`, para satisfazer o pedido, é feita desde o último bloco que foi visitado na procura anterior. Assim que for encontrado um tal bloco (de dimensão `N` $>$ `X`, ele é partido num bloco do tamanho desejado, que é retornado a quem fez o pedido, e num novo bloco com a parcela de memória restante, `M-X`.

Best Fit: a procura por um bloco de dimensão maior ou igual a `X`, para satisfazer o pedido, é feita a partir do início da lista, mas é procurado o bloco cuja dimensão mais se aproxima de `X`, ou seja é procurado o bloco de dimensão `N` ($\geq$ `X`) que mais se aproxima de `X`.

Worst Fit: a procura por um bloco de dimensão maior ou igual a `X`, para satisfazer o pedido, é feita a partir do início da lista, mas é procurado o bloco de maior dimensão.

Quick Fit: listas separadas são mantidas para dimensões comuns de utilização e a procura é feita na lista que contém blocos de dimensão maior mas mais próxima do pretendido.

Neste problema vamos simular o funcionamento de um sistema de alocação de memória e implementar alguns destes algoritmos. Assume-se que a localização dos blocos (o endereço na memória) é um número inteiro positivo. O nosso sistema vai ler informação da memória sob a forma de uma sequência constituida por uma letra e um número não negativo (separados por um espaço) com o seguinte significado:

- o primeiro par de dados deve ser a letra `M` seguida de um número inteiro positivo, que se assume ser a dimensão total da memória disponível;
- os restantes pares de dados são pares do tipo `R N` ou `F X` com o seguinte significado:

  `R N` corresponde a um pedido (*Request*) de memória de dimensão N; o sistema de alocação deve procurar o bloco a assignar e devolver o número (endereço) do bloco assignado, `X`, alterando a lista de forma a indicar que com início em `X`

existe um bloco de tamanho `N` que está actualmente a ser utilizado; se não puder ser assignado nenhum bloco o simulador deve devolver `-1`.

`F X` corresponde a uma libertação (*Free*) de um bloco previamente pedido e assignado ao endereço `X`; o sistema deve procurar na lista de blocos assignados um bloco com início em `X` e alterar esse bloco como estando disponível, verificando se o bloco com endereços imediatamente antes e depois desse está igualmente disponível; se estiver os dois blocos devem ser juntos num bloco de dimensão igual à de ambos os blocos; se o sistema verificar que não existe nenhum bloco previamente assignado com início no endereço `X`, o simulador deve devolver `-2`;

O simulador termina quando não tiver mais pedidos ou quando encontrar um pedido de alocação de tamanho `0`. Se receber um pedido de alocação de memória o simulador coloca no écran a dimensão N do pedido, seguido do endereço X onde está localizado o bloco que foi assignado (ou -1 caso tenha sido possível efectuar a assignação. Se for um pedido de libertação deve colocar no écran o endereço libertado seguido da sua dimensão que pode ser recuperada da informação na lista. Em caso de erro nos dados de entrada deve ser devolvido `-3`.

O código apresentado, nos ficheiros `p3-pub.c` e `memmgt.c`, que está propositadamente incompleto, deve permitir a leitura da informação no formato pedido, o processamento dos vários pedidos encontrados e a indicação do resultado desse processamento.

3.1. Complete o código dado de forma a que, após ler o primeiro comando, seja alocada na lista uma estrutura que representa a memória disponível que neste caso é toda a memória. Deve ser igualmente adicionado código em `p3-pub.c` para, consoante o pedido lido, efectuar um pedido de alocação ou de libertação de memória. Finalmente deve ainda ser adicionado código para, quando o programa terminar, libertar toda a informação contida na lista de gestão da memória.

3.2. Implemente em `memmgt.c` a função de alocação de memória de acordo com o algoritmo XXXXX. Implemente igualmente a função que permite libertar blocos de memória mantendo a informação correcta na lista que descreve a utilização da memória.

Na avaliação será pedido a cada aluno que implemente um determinado algoritmo de alocação.

3.3. Complete o código da funcão `showMem()` que permite percorrer a lista que descreve os blocos alocados e livres e visualizar o estado da memória em qualquer altura. Esta função deve descrever o estado actual da memória indicando para cada bloco na lista de gestão de memória o seu endereço de início, a sua dimensão e o seu estado: "A" para ocupado (*allocated*) ou "F" para liberto (*free*). Para testar esta funcionalidade adicione em `p3-pub.c` a possibilidade de ler um novo par do tipo `S 0` que chama então esta função.

3.4. Complete o código da função `yyyMem()` que tem como objectivo... Para testar esta funcionalidade adicione em `p3-pub.c` a possibilidade de ler um novo par do tipo `Z 0` que chama então esta função.

Na avaliação será pedido a cada aluno que acrescente código para adicionar uma dada funcionalidade.