

Estruturas de Dados — MC202ABC
1º Semestre 2018
Instituto de Computação — UNICAMP
1ª Lista de Exercícios

Entregue os exercícios 1, 2 e 3 até 18/04/2018 pelo SuSy.
Leia com cuidado as instruções abaixo.

Instruções:

- Os três exercícios deverão ser entregues separados em três tarefas diferentes no SuSy: o exercício 1 deve ser entregue na tarefa `11ex1`; o exercício 2 deve ser entregue na tarefa `11ex2`; e o exercício 3 deve ser entregue na tarefa `11ex3`. Caso não faça algum dos exercícios, basta não submeter na tarefa referente ao exercício.
- A nota da lista será dada em relação aos três exercícios, apesar da submissão ocorrer em três tarefas diferentes do SuSy.
- Os exercícios podem ser digitados em um computador (preferencialmente) ou serem escritos à mão e digitalizados (você pode inclusive tirar uma foto). Porém, a entrega precisa ser feita necessariamente por meio de um arquivo **PDF** com tamanho de no máximo **2MB**. Outros formatos *não serão aceitos*.
- Garanta que o documento tenha qualidade o suficiente para ser lido. Documentos escritos à mão devem ter letras legíveis e feitos preferencialmente à caneta, já que o lápis tem pouco contraste para a digitalização. Assim, se possível, opte por fazer o exercício no computador ao invés de digitalizar.
- Caso tenha problemas com o tamanho do arquivo gerado, procure por um compressor de PDFs online. Existem vários serviços que recebem um PDF e geram o mesmo PDF com um tamanho menor. Porém, tome cuidado de garantir que o arquivo continue legível.
- Lembre-se de indentar corretamente o seu código para facilitar o entendimento.
- **Não se esqueça de colocar nome e RA em cada exercício.**
- A correção de cada exercício será enviada para o seu email institucional (da DAC). Garanta que há espaço na sua cota para receber os emails¹ e verifique o spam se necessário.

¹Considere fazer o redirecionamento do seu email da DAC para o seu email pessoal: <https://www.dac.unicamp.br/portal/estudantes/webmail-mais-informacoes>.

Exercício 1 (11ex1): Considere as estruturas de dados abaixo e as declarações das funções responsáveis pela manipulação destas estruturas. Implemente as 6 funções declaradas do Código 1, de acordo com suas assinaturas, de modo que, em conjunto, elas implementem uma estrutura de dados que represente uma fila do tipo FIFO².

```
1 typedef struct {
2     int campo1;
3     double campo2;
4     char *campo3;
5     /* E outros campos que se fizerem necessários no problema em que se
6        for aplicar a estrutura do tipo fila... */
7 } Registro;
8
9 typedef struct no {
10     Registro registro;
11     struct no *proximo;
12 } No;
13
14 typedef struct {
15     int tamanho;
16     No *inicio;
17     No *fim;
18 } Fila;
19
20 void fila_criar(Fila *fila);
21 void fila_destruir(Fila *fila);
22 void fila_enfileirar(Fila *fila, Registro registro);
23 Registro fila_desenfileirar(Fila *fila);
24 Registro fila_pegar_proximo(Fila *fila);
25 int fila_tamanho(Fila *fila);
```

Código 1: Estruturas e assinaturas das funções que definem uma fila do tipo FIFO.

A função `fila_criar` deve inicializar uma fila vazia. A função `fila_destruir` deve desalocar toda a memória utilizada pela fila, não importa quantos elementos existem nela. A função `fila_enfileirar` deve adicionar um elemento na fila. A função `fila_desenfileirar` deve retornar o próximo elemento da fila e removê-lo dela. A função `fila_pegar_proximo` deve retornar o próximo elemento da fila, porém deve manter a fila intacta. A função `fila_tamanho` deve retornar o número de elementos contidos na fila.

²Ver <https://pt.wikipedia.org/wiki/FIFO>.

Exercício 2 (11ex2): Considere que por algum motivo de segurança para determinada aplicação devemos *evitar* que cada posição de um vetor de inteiros armazenado na memória esteja na mesma ordem em que tal vetor seja utilizado pelo programa. Isso quer dizer que devemos *evitar* que a posição i deste vetor na memória esteja ao lado da posição $i+1$ na memória.

Neste exercício, será implementado um vetor que evitará que os elementos estejam alocados de forma contígua na memória, mas que poderá ser utilizado como se estivessem alocados contiguamente. A implementação deste vetor permitirá que seu uso aconteça de forma *similar* ao uso de um vetor de inteiros contíguos na memória.

Por decisão de implementação, foi definido o arquivo cabeçalho (*header*) como mostrado no Código 2. O Código 3 já contém a implementação da função `vetor_alocar`.

Observação: Não é necessário aprofundar o entendimento sobre como a função `shuffle` no Código 3 funciona, mas apenas saber o que ela faz. Ao aplicar a função `shuffle` em um vetor `[0, 1, 2, 3, 4, 5, 6, 8, 9]` de tamanho 10, o vetor é embaralhado (aleatoriamente) e no final seu conteúdo passa a ser algo como, por exemplo, `[3, 4, 1, 9, 7, 0, 2, 8, 5, 6]` (observe que o conjunto final de números é igual ao conjunto inicial de números do vetor, apenas suas posições foram embaralhadas).

Faça os itens a seguir:

- a) Implemente as funções `vetor_inicializar` e `vetor_desalocar`. A função `vetor_inicializar` deve inicializar cada inteiro do vetor com o valor 0. A função `vetor_desalocar` deve liberar todo o espaço de memória alocado pela função `vetor_alocar`.
- b) A função `vetor_dobrar_tamanho` é responsável por dobrar o tamanho do vetor. Não é necessário realizar sua implementação, porém, responda às seguintes perguntas: Com base na assinatura desta função, apresentada no Código 2, é possível implementá-la de modo que ela tenha o comportamento esperado? Justifique sua resposta.

```
1 #ifndef VETOR_H
2 #define VETOR_H
3
4 #include <stdlib.h>
5
6 int **vetor_alocar(int tamanho);
7
8 /* Exercício 2 (a): implementar as duas funções abaixo. */
9 void vetor_inicializar(int **vec, int tamanho);
10 void vetor_desalocar(int **vec, int tamanho);
11
12 /* Exercício 2 (b): responder questão acerca da assinatura da função
13    abaixo. */
14 void vetor_dobrar_tamanho(int **vec, int tamanho);
15
16 #endif /* VETOR_H */
```

Código 2: vetor.h

```

1 #include "vetor.h"
2
3 /* Código desta função obtido em: https://stackoverflow.com/a/6127606
4 Esta função recebe um vetor e seu tamanho como argumentos e
5 embaralha o vetor. */
6 void shuffle(int *array, size_t n) {
7     if (n > 1) {
8         size_t i;
9         for (i = 0; i < n - 1; i++) {
10             size_t j = i + rand() / (RAND_MAX / (n - i) + 1);
11             int t = array[j];
12             array[j] = array[i];
13             array[i] = t;
14         }
15     }
16 }
17
18 int **vetor_alocar(int tamanho) {
19     int i;
20     int *array;
21     int **vec;
22
23     /* Vetor auxiliar para evitar que 'vec' tenha ponteiros contíguos
24 apontando para posições contíguas na memória. */
25     array = malloc(tamanho * sizeof(int));
26     for(i = 0; i < tamanho; ++i)
27         array[i] = i;
28     shuffle(array, tamanho); /* Embaralhar 'array'. */
29
30     vec = malloc(tamanho * sizeof(int *));
31     for(i = 0; i < tamanho; i++)
32         vec[array[i]] = malloc(sizeof(int));
33
34     /* Liberar vetor auxiliar utilizado para alocar memória em posições
35 aleatórias de 'vec'. */
36     free(array);
37
38     return vec;
39 }
40
41 void vetor_inicializar(int **vec, int tamanho);
42
43 void vetor_desalocar(int **vec, int tamanho);

```

Código 3: vetor.c

Exercício 3 (11ex3): Suponha que você tenha que simular um robô percorrendo um labirinto bidimensional em busca de um prêmio. A informação do labirinto é recebida como uma matriz de caracteres, como no exemplo abaixo.

```

||||| | | | | | | | | | | | | | | | | | | | |
| |          |||  |||  |  |
|X ||||| | | |||| | | |||| |
| |||      | | || |||| | |||  |||| |
|   | |||| |  ||   | |  ||||   |
|||||| || | ||||| || | ||||| |||
|   || | |  |||||  || |  |  |
| |||  || |  ||||  |||  |||  |  |
| ||||| ||||| |  |  | ||| |  |
| |  |  |  | ||||| ||||| |  |
| | ||||| ||||| | |||  ||||| |
| |  |  |  |  |  ||||   |
| | ||||| | ||||| | |||  ||||| |
| |  |  |  |  |  |  |  |
||||||| ||||| ||| | ||||| |||
|   ||  ||| |  |||  |  |
| |||||  ||||  |||||  ||||P |
|||||

```

O robô só pode se locomover para o norte, sul, oeste ou leste, ou seja, movimentos na diagonal não são permitidos. No exemplo acima, os caracteres ‘|’ denotam as paredes por onde o robô *não* pode passar. Os caracteres ‘ ’ (espaços) indicam posições onde a movimentação pode acontecer livremente. O caractere ‘X’ indica a posição inicial do robô no labirinto e o caractere ‘P’ indica o prêmio que o robô deve encontrar. A solução para o problema é o caminho que o robô deve percorrer para encontrar o prêmio. No exemplo abaixo, a solução para o problema acima é indicada pelos caracteres ‘.’.

```

||||| | | | | | | | | | | | | | | | | | | | | | | |
| |          |||. . . . . |||  |  |
|. ||||| | | |. ||||. | | |||| |
|. |||. . . . . | | ||. ||||. | |||  |||| |
|. . . . . | |||. |  ||. . . . . |  ||||   |
|||||| || |. ||||| |||. ||| ||||| |||
|   || |. |. . . . . |. ||. . . . . |
| ||||| |. |. ||||| |||. ||. | |||||. |
|   ||| ||. |. ||||| |||. ||. |  |. |
| |||  ||. |. . . . . |. ||||. |||  |  |
| ||||| ||||| |. . . . . |  ||| |. |
| |. . . . . |  |. ||||| ||||| |. |
| |. ||||| ||||| | |. |||. . . . . |||||. |
| |. |  |  |  | |. |. . . . . |||||. |
| |. ||||| | ||||| |. |||. . . . . ||||| |
| |. . . . . |. . . . . |. . . . . |
||||||| ||||| ||| | ||||| |||
|   ||  ||| |  |||  |  |
| |||||  ||||  |||||  ||||P. |
|||||

```

Este problema pode ser facilmente resolvido utilizando recursão. Por exemplo, o pseudocódigo mostrado no Código 4 soluciona o problema recursivamente.

```

1 EncontrarPrêmio(labirinto, i, j):
2   Se labirinto[i, j] == '0':
3     Retornar 0
4   Se labirinto[i, j] == 'P':
5     Retornar 1
6
7   labirinto[i, j] = '0'
8   Se (EncontrarPrêmio(labirinto, i-1, j ) OU
9     EncontrarPrêmio(labirinto, i+1, j ) OU
10    EncontrarPrêmio(labirinto, i, j-1) OU
11    EncontrarPrêmio(labirinto, i, j+1)):
12     labirinto[i, j] = '.'
13     Retornar 1
14   Senão:
15     labirinto[i, j] = ' '
16     Retornar 0
17

```

Código 4: Pseudocódigo recursivo para encontrar o prêmio em um labirinto e marcar os passos do robô.

No entanto, para este exercício, considere que o uso de recursão não é permitido. Neste caso,

- a) Dentre as estruturas de dados vistas em sala de aula (vetores, listas, pilhas, filas), qual seria a mais apropriada de se utilizar para a solução deste problema? Justifique sua resposta.
- b) Defina uma estrutura (**struct** em código C) apropriada para se utilizar juntamente com a estrutura de dados indicada na resposta anterior. (Observação: não é necessário definir a estrutura de dados da resposta para o item **a.**)