Estruturas de Dados — MC202ABC
1º Semestre 2018

Instituto de Computação — UNICAMP

1ª Lista de Exercícios

Entregue os exercícios 1, 2 e 3 até 15/04/2018 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 `l1ex1`; o exercício 2 deve ser entregue na tarefa `l1ex2`; e o exercício 3 deve ser entregue na tarefa `l1ex3`. 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 emails[1] e verifique o spam se necessário.

[1] Considere 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 (`l1ex1`): Considere as estruturas de dados abaixo e as declarações das funções responsáveis pela manipulação destas estruturas. Em conjunto, elas implementam uma fila do tipo FIFO[2]. Implemente as 6 funções declaradas do Código 1 de acordo com suas assinaturas.

```c
typedef struct {
  int campo1;
  double campo2;
  char *campo3;
  /* E outros campos que se fizerem necessários no problema em que se
     for aplicar a estrutura do tipo fila... */
} Registro;

typedef struct no {
  Registro registro;
  struct no *proximo;
} No;

typedef struct {
  int tamanho;
  No *inicio;
  No *fim;
} Fila;

void fila_criar(Fila *fila);
void fila_destruir(Fila *fila);
void fila_enfileirar(Fila *fila, Registro registro);
Registro fila_desenfileirar(Fila *fila);
Registro fila_pegar_proximo(Fila *fila);
int fila_tamanho(Fila *fila);
```

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

[2] Ver https://pt.wikipedia.org/wiki/FIFO.

Exercício 2 (`l1ex2`): 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.

```c
#ifndef VETOR_H
#define VETOR_H

#include <stdlib.h>

int **vetor_alocar(int tamanho);

/* Exercício 2 (a): implementar as duas funções abaixo. */
void vetor_inicializar(int **vec, int tamanho);
void vetor_desalocar(int **vec, int tamanho);

/* Exercício 2 (b): responder questão acerca da assinatura da função
   abaixo. */
void vetor_dobrar_tamanho(int **vec, int tamanho);

#endif  /* VETOR_H */
```

Código 2: vetor.h

```c
#include "vetor.h"

/* Código desta função obtido em: https://stackoverflow.com/a/6127606
   Esta função recebe um vetor e seu tamanho como argumentos e
   embaralha o vetor. */
void shuffle(int *array, size_t n) {
  if (n > 1) {
    size_t i;
    for (i = 0; i < n - 1; i++) {
      size_t j = i + rand() / (RAND_MAX / (n - i) + 1);
      int t = array[j];
      array[j] = array[i];
      array[i] = t;
    }
  }
}

int **vetor_alocar(int tamanho) {
  int i;
  int *array;
  int **vec;

  /* Vetor auxiliar para evitar que `vec` tenha ponteiros contíguos
     apontando para posições contíguas na memória. */
  array = malloc(tamanho * sizeof(int));
  for(i = 0; i < tamanho; ++i)
    array[i] = i;
  shuffle(array, tamanho);  /* Embaralhar `array`. */

  vec = malloc(tamanho * sizeof(int *));
  for(i = 0; i < tamanho; i++)
    vec[array[i]] = malloc(sizeof(int));

  /* Liberar vetor auxiliar utilizado para alocar memória em posições
     aleatórias de `vec`. */
  free(array);

  return vec;
}

void vetor_inicializar(int **vec, int tamanho);

void vetor_desalocar(int **vec, int tamanho);
```

Código 3: vetor.c

Exercício 3 (`l1ex3`): 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.

```
EncontrarPrêmio(labirinto, i, j):
  Se labirinto[i, j] == 'O':
    Retornar 0
  Se labirinto[i, j] == 'P':
    Retornar 1

  labirinto[i, j] = 'O'
  Se (EncontrarPrêmio(labirinto, i-1, j  ) OU
      EncontrarPrêmio(labirinto, i+1, j  ) OU
      EncontrarPrêmio(labirinto, i,   j-1) OU
      EncontrarPrêmio(labirinto, i,   j+1)):
    labirinto[i, j] = '.'
    Retornar 1
  Senão:
    labirinto[i, j] = ' '
    Retornar 0

```

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.)