BCC202 – Estruturas de Dados I (2023-01)

Departamento de Computação - Universidade Federal de Ouro Preto - MG
Professor: **Pedro Silva** (`www.decom.ufop.br/`)
Estagiário docente: ——

Trabalho Prático I (TP I) - 10 pontos, peso 1.

- Data de entrega: 04/06/2023 até 23:55. O que vale é o horário do *Moodle*, e não do *seu*, ou do *meu* relógio!!!

- Clareza, identação e comentários no código também vão valer pontos. Por isso, escolha cuidadosamente o nome das variáveis e torne o código o mais legível possível.

- O padrão de entrada e saída deve ser respeitado exatamente como determinado no enunciado. Parte da correção é automática, não respeitar as instruções enunciadas pode acarretar em perda de pontos.

- Durante a correção, os programas serão submetidos a vários casos de testes, com características variadas.

- A avaliação considerará o tempo de execução e o percentual de respostas corretas.

- Eventualmente serão realizadas entrevistas sobre os estudos dirigidos para complementar a avaliação;

- O trabalho é em grupo de até 2 (duas) pessoas.

- Entregar um relatório.

- Os códigos fonte serão submetidos a uma ferramenta de detecção de plágios em software.

- Códigos cuja autoria não seja do aluno, com alto nível de similaridade em relação a outros trabalhos, ou que não puder ser explicado, acarretará na perda da nota.

- Códigos ou funções prontas específicas de algoritmos para solução dos problemas elencados não são aceitos

- Não serão considerados algoritmos parcialmente implementados.

- Procedimento para a entrega:.

1. Submissão: via ***Moodle***.
2. Os nomes dos arquivos e das funções devem ser especificados considerando boas práticas de programação.
3. Funções auxiliares, complementares aquelas definidas, podem ser especificadas e implementadas, se necessário.
4. A solução deve ser devidamente modularizada e separar a especificação da implementação em arquivos *.h* e *.c* sempre que cabível.
5. Os arquivos a serem entregues, incluindo aquele que contém *main()*, devem ser compactados (*.zip*), sendo o arquivo resultante submetido via ***Moodle***.
6. Você deve submeter os arquivos *.h*, *.c* e o *.pdf* (relatório) na raiz do arquivo *.zip*. Use os nomes dos arquivos *.h* e *.c* exatamente como pedido.
7. Caracteres como acento, cedilha e afins não devem ser utilizados para especificar nomes de arquivos ou comentários no código.

- Bom trabalho!

Ajudando um rato a sair do labirinto

Um conjunto de caminhos intercalados que são criados com o intuito de desorientar quem o está percorrendo é chamado de labirinto. Por definição, um labirinto é uma "construção de muitas passagens ou divisões, dispostas tão confusamente que com dificuldade se acha a saída" (`dicio.com.br`).

A Figura 1 ilustra um exemplo de labirinto.

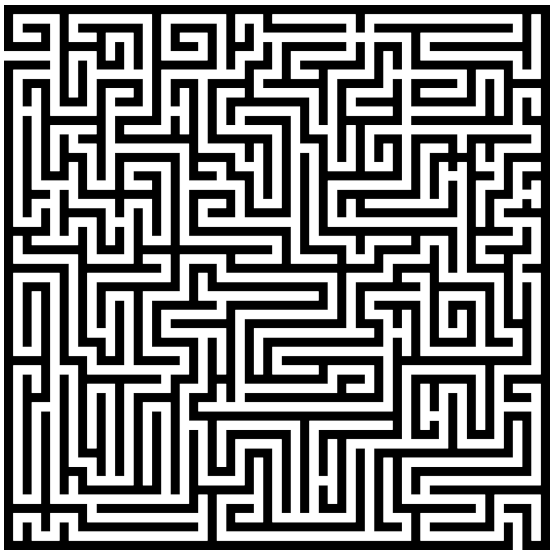

Figura 1: Exemplo de labirinto.

O objetivo deste trabalho é, dado um labirinto qualquer e a posição inicial de um rato, encontrar o menor caminho até saída. Para cada posição, o rato tem até quatro ações (desde que uma parede não o impeça): ir para direita, esquerda, para cima ou para baixo.

Uma forma de resolver esse problema é por meio de busca por força bruta. O algoritmo de busca por força bruta (ou enumeração exaustiva) é um método utilizado para encontrar soluções para problemas, porém pode ser ineficiente em situações nas quais existem muitas possibilidades.

Basicamente, você implementará o método de busca por profundidade utilizando recursão, onde você irá testando os vários caminhos até encontrar o que for desejado.

Imposições e comentários gerais

Neste trabalho, as seguintes regras devem ser seguidas:

- Seu programa não pode ter *memory leaks*, ou seja, toda memória alocada pelo seu código deve ser corretamente liberada antes do final da execução. (Dica: utilize a ferramenta *valgrind* para se certificar de que seu código libera toda a memória alocada).
- Um grande número de *Warnings* ocasionará a redução na nota final.

O que deve ser entregue

- Código fonte do programa em C (**bem identado e comentado**).
- Documentação do trabalho (relatório[1]). A documentação deve conter:
  1. **Introdução:** descrição sucinta do problema a ser resolvido e visão geral sobre o funcionamento do programa.
  2. **Implementação:** descrição sobre a implementação do programa. **Não faça** "*print screens*" de telas. Ao contrário, procure resumir ao máximo a documentação, fazendo referência ao que julgar mais relevante. É importante, no entanto, que seja descrito o funcionamento das principais funções e procedimentos utilizados, bem como decisões tomadas relativas aos casos e detalhes

[1]Exemplo de relatório: `https://www.overleaf.com/latex/templates/modelo-relatorio/vprmcsdgmcgd`.

de especificação que porventura estejam omissos no enunciado. Muito importante: os códigos utilizados na implementação devem ser inseridos na documentação.

3. **Estudo de Complexidade:** estudo da complexidade do tempo de execução dos procedimentos implementados e do programa como um todo (notação O), considerando entradas de tamanho n.
4. **Testes:** descrição dos testes realizados e listagem da saída (não edite os resultados).
5. **Análise**: deve ser feita uma análise dos resultados obtidos com este trabalho. Por exemplo, avaliar o tempo gasto de acordo com o tamanho do problema.
6. **Conclusão**: comentários gerais sobre o trabalho e as principais dificuldades encontradas em sua implementação.
7. **Bibliografia:** bibliografia utilizada para o desenvolvimento do trabalho, incluindo sites da Internet se for o caso.
8. **Formato:** PDF ou HTML.

Como deve ser feita a entrega

Verifique se seu programa compila e executa na linha de comando antes de efetuar a entrega. Quando o resultado for correto, entregue via *Moodle* até a 04/06/2023 até 23:55 um arquivo **.ZIP** com o nome e sobrenome do aluno. Esse arquivo deve conter: (i) os arquivos *.c* e *.h* utilizados na implementação, (ii) instruções de como compilar e executar o programa no terminal, e (iii) o relatório em **PDF**.

Detalhes da implementação

Para atingir o seu objetivo, você deverá construir um Tipo Abstrato de Dados (TAD) `Labirinto` como representação de um labirinto que você quer analisar. O TAD `Labirinto` deverá implementar, pelo menos, as seguintes operações:

1. `alocarLabirinto`: aloca um (ou mais) TAD `Labirinto`.
2. `desalocarLabirinto`: desaloca um TAD `Labirinto`.
3. `leLabirinto`: inicializa o TAD `Labirinto` a partir de dados do **terminal**.
4. `acharSaida`: função **recursiva** que retorna o percurso que deve ser feito para sair do labirinto presente no TAD `Labirinto`.
5. `imprimePercursoNoLabirinto`: imprime o labirinto com o percurso encontrado.

Além do TAD `Labirinto`, deve ser implementado o TAD `Posicao` que armazena as coordenadas (x, y). O TAD `Posicao` deve ser usado como parte do TAD `Percurso` que armazena a sequência de posições (TAD *Posicao*) até a saída e um inteiro com o comprimento da sequência de posições. As funções do TAD `Posicao` e do TAD `Percurso` ficam a cargo do aluno.

Alocação de um ou mais TADs `Labirinto`, `Posicao` e `Percurso` fica a critério do aluno. Os TADs devem ser implementados utilizando a separação interface no *.h* e implementação *.c* discutida em sala, bem como as convenções de tradução. Contudo, todos os TADs podem ficar em um único arquivo. Caso a operação possa dar errado, devem ser definidos retornos com erro, tratados no corpo principal. **A alocação da TAD necessariamente deve ser feita de forma dinâmica**.

O código-fonte deve ser modularizado corretamente em três arquivos: *main.c*, *labirinto.h* e *labirinto.c*. O arquivo *main.c* deve apenas invocar e tratar as respostas das funções e procedimentos definidos no arquivo *labirinto.h*. A separação das operações em funções e procedimentos está a cargo do aluno, porém, **não deve haver acúmulo** de operações dentro de uma mesma função/procedimento.

O limite de tempo para solução de cada caso de teste é de apenas **um segundo**. Além disso, o seu programa não pode ter *memory leaks*, ou seja, toda memória alocada pelo seu código deve ser corretamente liberada antes do final da execução. (Dica: utilize a ferramenta *valgrind* para se certificar de que seu código libera toda a memória alocada). *Warnings* ocasionará a redução na nota final. Assim sendo, utilize suas habilidades de programação e de análise de algoritmos para desenvolver um algoritmo correto e rápido!

Entrada

A entrada é dada por meio do terminal. Para facilitar, a entrada será fornecida por meio de arquivos. [2] A primeira linha especifica as dimensões do labirinto que não necessariamente é quadrado, logo serão informados dois valores: o número de linhas e o número de colunas respectivamente. **A saída do labirinto sempre será o canto inferior direito**. Após essas duas linhas, um caractere é fornecido para definir se o caminho ('c') deve ser impresso ou se o labirinto em si deve ser impresso ('p'). Por fim, é apresentada uma matriz de caracteres que reproduz o labirinto a ser analisado, sendo o caractere '*' representando as paredes do labirinto, ' ' (espaço) um caminho e 'M' a origem de onde se sai. Abaixo um exemplo de entrada.

```
21 21
p
**********************
*        *M  *        *
******* * * * * *** *
*      *   * * *   * *
* * ******* ***** * *
* *      *   *     * *
* *** * * *** ***** *
*   * * * *   *   * *
* *** * * * *** *** *
* *   * *   * *     *
* * *** ***** * *****
* * * *   *     *   *
*** * *** * ***** * *
*   *     *     * * *
* ***** ******* * * *
* *   * *   *   * * *
* * * * * *** *** * *
*   * *   *   *   * *
* *** ***** ******* *
*   *
**********************
```

Saída

A saída depende do caractere informado. Se o caractere informado for o 'c', a saída informada deve ser o tamanho e o **menor** percurso encontrado, onde cada linha é uma coordenada no padrão: `x, y`. Se o caractere informado for o 'p', a saída informada deve ser o tamanho e a impressão do labirinto com o caminho encontrado, para isso, utilize o caractere '.' para informar o caminho percorrido.

Também pode ocorrer que não tenha saída no labirinto dada a posição que você se encontrava. Quando isso ocorrer, apresente a mensagem "EPIC FAIL!".

Exemplo de um caso de teste

Exemplos de saídas esperadas dada diferentes entradas:

Entrada	Saída
7 11 c ********** * * ******** * *M* * * * * *** * * * * * **********	EPIC FAIL!

[2] Para usar o arquivo como entrada no terminal, utilize `./executavel < nome_do_arquivo_de_teste`.

Entrada	Saída
15 15 c *************** * * * * *** * * * *** * *M* * * * * * * *** ***** * * * * * * * *** * * ***** * * * * * * ***** * * * * * * * * * * *** ******* * * * * * *** ******* * * * * ***************	27 4, 1 5, 1 6, 1 7, 1 8, 1 9, 1 10, 1 11, 1 11, 2 11, 3 12, 3 13, 3 13, 4 13, 5 13, 6 13, 7 13, 8 13, 9 13, 10 13, 11 12, 11 11, 11 11, 12 11, 13 12, 13 13, 13 13, 14

Entrada	Saída
15 15 p *************** * * * * *** * * * *** * *M* * * * * * * *** ***** * * * * * * * *** * * ***** * * * * * * ***** * * * * * * * * * * *** ******* * * * * * *** ******* * * * * ***************	27 *************** * * * * *** * * * *** * *M* * * * * *.* *** ***** * *.* * * * *.*** * * ***** *. * * * * *.***** * * * * *.* * * * *.*** ******* * *...* *...* ***.*******.*.* * ....*.. ***************

A SAÍDA DA SUA IMPLEMENTAÇÃO DEVE SEGUIR EXATAMENTE A SAÍDA PROPOSTA.

Diretivas de Compilação

As seguintes diretivas de compilação devem ser usadas (essas são as mesmas usadas no *Moodle*).

```
$ gcc -c labirinto.c -Wall
$ gcc -c main.c -Wall
$ gcc labirinto.o main.o -o exe -lm
```

Avaliação de *leaks* de memória

Uma forma de avaliar se não há *leaks* de memória é usando a ferramenta *valgrind*. O *valgrind* é um *framework* de instrumentação para análise dinâmica de um código e é muito útil para resolver dois problemas em seus programas: **vazamento de memória e acesso a posições inválidas de memória** (o que pode levar a *segmentation fault*). Um exemplo de uso é:

```
1 gcc -g -o exe arquivo1.c arquivo2.c -Wall
2 valgrind --leak-check=full -s ./exe < casoteste.in
```

Espera-se uma saída com o fim semelhante a:

```
1 ==xxxxx== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Para instalar no Linux, basta usar: `sudo apt install valgrind`.

O SEU CÓDIGO SERÁ TESTADO NOS COMPUTADORES DO LABORATÓRIO (AMBIENTE LINUX)

PONTO EXTRA

Será concedido 0,1 extra para quem gerar o relatório em LaTeX(Latex). **Deve ser enviado os *.tex*** e o arquivo *PDF*.

Será concedido 0,1 extra para quem criar um gerador de labirintos aleatórios no formato usado neste trabalho.