BCC202 – Estruturas de Dados I

Departamento de Computação - Universidade Federal de Ouro Preto - MG
Professora: **Karla A S Joriatti** (`www.decom.ufop.br/`)

Aula Prática III - Listas

- 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. Caracteres como acento, cedilha e afins não devem ser utilizados para especificar nomes de arquivos ou comentários no código.
7. Siga atentamente quanto ao formato da entrada e saída de seu programa, exemplificados no enunciado.
8. Durante a correção, os programas serão submetidos a vários casos de testes, com características variadas.
9. A avaliação considerará o tempo de execução e o percentual de respostas corretas.
10. Eventualmente, serão realizadas entrevistas sobre os estudos dirigidos para complementar a avaliação.
11. Considere que os dados serão fornecidos pela entrada padrão. Não utilize abertura de arquivos pelo seu programa. Se necessário, utilize o redirecionamento de entrada.
12. Os códigos fonte serão submetidos a uma ferramenta de detecção de plágios em software.
13. 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.
14. Códigos ou funções prontas específicos de algoritmos para solução dos problemas elencados não são aceitos.
15. Não serão considerados algoritmos parcialmente implementados.

- Bom trabalho!

Modelagem de Cidades Adjacentes

Crie um programa que leia uma lista de duplas que representa a conexão de cidades por meio de estradas e armazene essas informações em um vetor de listas encadeadas. Cada posição do vetor indica uma cidade (e.g., 0,1,2,...) e a lista encadeada representará as cidades que são adjacentes à ela. O objetivo é que seu algoritmo seja capaz de determinar se é possível chegar de uma cidade a outra.

Especificação da Entrada e da Saída

Seu algoritmo deve ler dois números inteiros N e M que representam a quantidade de cidades e a quantidade de conexões a serem lidas. Em seguida, ele deve ler as próximas M linhas, cada uma contendo uma dupla de inteiros, representando as conexões entre as cidades. Após ler essas linhas, seu programa deve ler uma última dupla de inteiros que representa a pergunta do exercício: é possível chegar a uma cidade B partindo de uma cidade A?

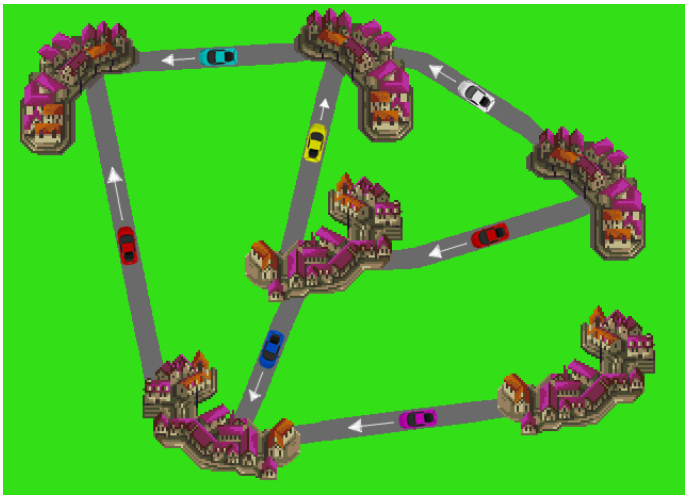

Figura 1: Representação de cidades ligadas por uma estrada

Lembre-se de que as estradas possuem sentido e, portanto, a conexão entre as cidades 0 e 1 significa que 1 estará na lista de adjacências de 0, mas o inverso não é verdade.

A saída do programa deve imprimir na tela SIM ou NÃO, de acordo com a possibilidade de existir um caminho de A para B ou não. Esse caminho não precisa ser direto e pode passar por outras cidades.

Entrada	**Saída**
6 7 1 0 2 1 2 3 3 1 3 4 4 0 5 4 5 0	SIM

Entrada	**Saída**
6 7 1 0 2 1 2 3 3 1 3 4 4 0 5 4 0 5	NAO

Siga o protótipo fornecido, sem alterar os nomes dos arquivos e das funções principais e execute sua implementação com os testes disponibilizados, comparando a saída esperada da sua saída.

A saída e a entrada devem OBRIGATORIAMENTE seguir o padrão fornecido no trabalho e permitir redirecionamento por arquivo.

Diretivas de Compilação

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

```
$ gcc -c aluno.c -Wall
$ gcc -c pratica.c -Wall
$ gcc aluno.o pratica.o -o exe
```

Antes de enviar TESTE o seu código com:

```
$ ./exe < teste1.in
```

Avaliação de *leaks* de memória

Uma forma de avaliar se não há *leaks* de memória é usando a ferramenta *valgrind*. Um exemplo de uso é:

```
1 gcc -g -o exe *.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`.