UNIVERSIDADE FEDERAL DE OURO PRETO
Rafael Francisco de Oliveira - 2021.10171
PCC104 - Projeto e Análise de Algoritmos
Github: raffoliveira
Lista 2

Questões práticas

A pasta denominada lista_2 no GitHub contém os respectivos exercícios da lista. A implementação dos algoritmos foi dividida em módulos. Abaixo segue a uma breve descrição dos arquivos disponibilizados:

- A pasta exercicios contém os arquivos de cada exercício.
- O arquivo chamado functions.h é responsável por ter as declarações de todas as funções.
- O arquivo chamado functions.cpp contém as implementações de cada função.
- A pasta testes contém os arquivos contendo as instâncias para testes. Para a escolha de cada arquivo nesta pasta, basta modificar as linhas 59 e 60 do arquivo functions.cpp para o tipo float e as linhas 85 e 86 do arquivo functions.cpp para o tipo string, as quais especificam o caminho do diretório da pasta e o nome do arquivo escolhido, respectivamente.

O ambiente de execução utilizado durante a codificação foi *Linux*. Nenhum tipo de tratamento foi realizado nos dados inseridos pelo usuário. Por isso, siga estritamente as instruções durante a execução. Para a execução dos algoritmos, execute os seguintes comandos abaixo no terminal, mudando apenas o nome do exercício desejado:

```
g++ exercicio_1.cpp -o exe
./exe
```

O primeiro comando irá compilar o código e gerar um arquivo executável nomeado de acordo com nome especificado no comando. No exemplo acima, o executável seria renomeado como *exe*. Para executar o executável, basta executar a segunda linha do comando acima especificando o nome do executável criado.

Instruções para cada exercício

Os exercícios de **1** e **2** irão realizar a leitura dos dados de entrada diretamente dos arquivos da pasta testes. Logo, para as execuções, será apenas necessário escolher o arquivo desejado de acordo com as instruções anteriores. Nos códigos de cada exercício há linhas comentadas. Remova os comentários e comente as outras linhas semelhantes caso queira modificar o tipo de entrada e a execução.

O exercício **3** irá realizar a leitura dos dados de entrada diretamente dos arquivos da pasta testes. Logo, para as execuções, será apenas necessário escolher o arquivo desejado de acordo com as instruções anteriores. Porém, necessitará do dado a ser encontrado no *array*. Esse dado será inserido pelo usuário. No código do exercício há linhas comentadas. Remova os comentários e comente as outras linhas semelhantes caso queira modificar o tipo de entrada e execução.

O exercício **4** necessitará da entrada de duas *strings*. A primeira será a *string* original e a segunda será padrão a ser encontrado. Esses dois dados serão inseridos pelo usuário. Para isso, siga as instruções durante a execução.

Os exercícios **5** e **6** necessitarão de um conjunto de pontos cartesianos. O usuário irá inserir todos os pontos. Primeiro irá especificar o número de pontos a serem inseridos, sendo necessário ser maior ou igual a 2. Depois insere-se a coordenada X e depois a coordenada Y.

O exercício **7** contém quatro matrizes de entradas (3, 4, 5 e 6 cidades). As matrizes correspondem as distâncias entre as cidades. O usuário deve mudar a variável NUMBER_CITY que se encontra na linha 12 do arquivo exercicio_7.cpp que corresponde ao número de cidades como entrada para o algoritmo TravellingSalesmanProblem(). Depois remova os comentários das linhas de instruções que realiza a criação da matriz. A saída é um *array* de pontos, representando o caminho mínimo, onde cada ponto corresponde a posição da matriz. Por exemplo, o caminho $[(0, 5), (5, 4), (4, 2), (2, 3), (3, 1), (1, 0)]$ significa que o caminho começa com a posição $[0, 5]$ da matriz, depois vai para a posição $[5, 4]$ até chegar a última posição $[1, 0]$.

O exercício **8** necessita da entrada do peso da mochila, o número de items e cada respectivo peso e valor de cada item. Todos esses dados serão inseridos pelo usuário. Para isso, siga as instruções durante a execução.

Os exercícios **9** e **10** necessitarão da entrada de um grafo. O usuário irá inserir o número de vértices e o número de arestas do grafo. Após esses dados, as conexões serão inseridas. Por exemplo, o vértice 0 está conectado com o vértice 2 e o vértice 2 está conectado com o vértice 3. Depois das conexões, o usuário irá inserir o número a ser encontrado e o vértice inicial de busca.

OBS.: o número de vértices influencia os números correspondentes de cada vértice (rótulo), logo se o número de vértices for 5, os vértices podem ser rotulado de 0 a 4. Se o número de vértices for 8, os vértices podem ser rotulado de 0 a 7. Caso contrário, irá causar erro de *segmentation fault*.

Questões teóricas

1. Apresente um descrição da classe *vector* apresentando o custo computacional de cada uma de suas operações.

   Vector é uma forma de armazenar informações contíguas, onde os elementos podem ser acessados por interadores (*iterators*) ou por *offsets* de ponteiros para os elementos. A diferença entre um *vector* e um *static array* é que o *vector* é alocado dinamicamente (a memória é alocada a medida que cresce), consequentemente ocupando mais memória enquanto o *static array* aloca uma memória estática. A tabela abaixo apresenta as suas operações e seus custos operacionais.

Operação	**Custo**	**Operação**	**Custo**
swap	constante	clear	linear(n)
operator=	linear(n)	assign	linear(n)
operator[ ]	constant	shrink_to_fit	linear(n)
erase	linear(n)	push_back	constante amortizada
emplace_back	constante amortizada	insert	(n)linear
emplace	linear(n)	resize	linear(n)
pop_back	constant	at	constant
get_allocator	constant	reserve	linear(n)
front	constant	back	constant
empty	constant	size	constant
max_size	constant	capacity	constant
data	constant	destructor	linear(n)
construct	constant/linear(n)	begin/cbegin	constant
end/cend	constant	rbegin/crbegin	constant
rend/crend	constant	swap	constant
erase/erase_if	linear(n)	operator==	constant
operator!=	constant	operator>	linear(n)
operator>=	linear(n)	operator<	linear(n)
operator<=	linear(n)	operator<=>	linear(n)