Alexander André de Souza Vieira
Matrícula: 13/0039853
Professor: Jan Mendonça

Trabalho1 Métodos de Programação

Compilação: Pra compilar o trabalho, o usuário deve caminhar até "13_0039853_Alexander/Trabalho_1_MP_01_2017/src/" e digitar "make".
Execução: Para executar o trabalho, o usuário deve, depois de compilado o programa, executar o comando "./testa_grafo".

1)Nome da função, parâmetros e significado dos parâmetros. Especificação da função.

Nome: Graf cria_grafo (char *nome);
Parâmetros/Significado: char *nome => char contendo o nome que vai ser dado ao grafo.
Especificação: A função inicializa uma estrutura do tipo Grafo, aloca um espaço na memória para a estrutura criada, e atribui ao nome da função, o nome recebido como parâmetro.

Nome: char *retorna_nome_grafo(Graf);
Parâmetros/Significado: Graf => Grafo que vamos fazer a busca do nome.
Especificação: A função primeiro verifica se o Grafo existe. Se ele não existir, retorna NULL e imprime mensagem informando que não existe. Caso contrário, retorna o nome do grafo recebido como parâmetro.

Nome: void destroi_grafo(Graf);
Parâmetros/Significado: Graf => Grafo que vai ser destruído.
Especificação: A função primeiro verifica se o Grafo existe. Se ele não existir, imprime mensagem informando que não existe. Caso contrário, usa a função free para destruir os componentes do grafo, e por último o grafo.

Nome: void adjacente(Graf, int x, int y);
Parâmetros/Significado: Graf => Grafo que vamos verificar a adjacência.
X =>Identificador do vértice base.
Y =>Identificador do vértice que dever ser adjacente ao base.
Especificação:A função primeiro verifica se o grafo existe. Se ele não existir, imprime mensagem que os vértices não são adjacentes. Caso contrário, verifica se há uma aresta interligando os dois vértices, e imprime uma mensagem informando que os vértices são adjacentes.

Nome: struct Vertice *vizinhos(Graf, int x);
Parâmetros/Significado: Graf => Grafo que vamos fazer a análise.
X=>Identificador do vértice base.
Especificação: A função primeiro verifica se o grafo existe. Se ele não existir, imprime mensagem que não pode ter vizinhos no seu vértice. Caso contrário, retorna a lista dos vértices vizinhos ao vértice base.

Nome: void adiciona_vertice(Graf, int x);
Parâmetros/Significado: Graf => Grafo que vamos adicionar um vértice.
X => Inteiro identificador do vértice que queremos inserir no grafo.
Especificação: A função verifica se o grafo existe. Se ele não existir, imprime mensagem orientando a criar um grafo antes de tentar adicionar um vértice. Caso contrário, adiciona o vértice passado como parâmetro pra função, ao grafo.

Nome: void remove_vertice(Graf, int x);
Parâmetros/Significado: Graf => Grafo que vamos remover um vértice.
X => Inteiro identificador do vértice que queremos retirar.
Especificação: A função primeiro verifica se x é um vértice do grafo. Se não for, ela não faz nada. Caso contrário, retire o vértice.

Nome:void adiciona_aresta(Graf, int x, int y);
Parâmetros/Significado:Graf => Grafo que vamos adicionar a aresta.
X =>Identificador do vértice base da aresta.
Y =>Identificador do vértice que vai ser o destino da aresta.
Especificação: A função primeiro verifica se x e y são vértices do grafo. Se algum dos dois não for, imprime mensagem dizendo que algum dos dois não é vértice do grafo. Caso contrário, adiciona a aresta 1 nos vértices correspondentes.

Nome: void remove_aresta(Graf, int x, int y);
Parâmetros/Significado:Graf => Grafo que vamos remover a aresta.
X =>Identificador do vértice base da aresta.
Y =>Identificador do vértice destino da aresta.
Especificação: A função primeiro verifica se x e y são vértices do grafo. Se algum dos dois não for, imprime mensagem dizendo que algum dos dois não é vértice do grafo. Caso contrário, zera o valor da aresta correspondente aos vértices.

Nome: float retorna_valor_vertice(Graf, int x);
Parâmetros/Significado: Graf => Grafo que vamos obter o valor do vértice.
X =>Identificador do vértice que queremos saber o valor.
Especificação: A função primeiro verifica se x é um vértice do grafo. Se não for, imprime mensagem de erro dizendo que não é vértice do grafo, e retorna -1. Caso contrário, retorna o valor do vértice.

Nome:void muda_valor_vertice(Graf, int x, int val);
Parâmetros/Significado:Graf => Grafo que vamos alterar o valor do vértice.
X =>Identificador do vértice que vamos alterar o valor.
Val => Valor que vamos colocar no vértice.
Especificação:A função primeiro verifica se x é um vértice do grafo. Se não for, imprime mensagem de erro dizendo que não é vértice do grafo, e retorna -1. Caso contrário, muda o valor do vértice para o novo valor.

Nome: float retorna_valor_aresta(Graf, int x, int y);
Parâmetros/Significado:Graf => Grafo que vamos obter o valor da aresta.
X =>Identificador do vértice base.
Y => Identificador do vértice destino da aresta.
Especificação:A função primeiro verifica se x e y são vértices do grafo. Se algum dos dois não for, imprime mensagem de erro dizendo que algum dos dois não é vértice do grafo, e retorna -1. Caso contrário, retorna o valor da aresta que está entre os vértices.

Nome: void muda_valor_aresta(Graf, int x, int y, int val);
Parâmetros/Significado:Graf => Grafo que vamos obter o valor da aresta.
X =>Identificador do vértice base.
Y => Identificador do vértice destino da aresta.
Val => Valor que vamos colocar na aresta.
Especificação:A função primeiro verifica se x e y são vértices do grafo. Se algum dos dois não for, imprime mensagem de erro dizendo que algum dos dois não é vértice do grafo, e retorna -1. Caso

contrário, atribui o novo valor ao valor da aresta que está entre os vértices.

Nome:void visualiza_grafo(Graf g);
Parâmetros/Significado: Graf g=> Grafo g que será disponibilizado para o usuário.
Especificação: A função primeiro verifica se o grafo existe. Se ele não existir, imprime mensagem dizendo que o grafo não existe. Caso contrário, printa na tela o grafo, seus vértices e suas arestas.

Nome:int posicao_do_vertice(Graf xis, int aux_vert);
Parâmetros/Significado: Graf xis => grafo que queremos saber a posição do vértice.
aux_vert => vértice cujo o qual queremos descobrir a posição.
Especificação: A função primeiro verifica se o grafo existe. Se ele não existir, imprime mensagem dizendo que o grafo não existe e retorna -1. Caso contrário, retorna a posição do vértice.

Nome: float **estrut_matriz_alocacao(float **estr_mat, int componente);
Parâmetros/Significado: estr_mat =>estrutura float para representar a matriz
componente => componentes da matriz;
Especificação: A função primeiro verifica se a estrutura já existe. Se ela não existir, retorna -1. Caso contrário, aloca as linhas e os vértices da matriz.

Nome:float **estrut_matriz_realocacao(float **estr_mat, int componente);
Parâmetros/Significado: estr_mat =>estrutura float para representar a matriz
componente => componentes da matriz;
Especificação: A função primeiro verifica se a estrutura já existe. Se ela não existir, retorna -1. Caso contrário, realoca as linhas e os vértices da matriz para uma nova matriz.

2) Para cada um dos testes em cada função : Nome de cada teste, O que vai ser testado, Qual deve a ser a entrada , Qual deve ser a saída, Qual é o critério para passar no teste e Se a sua função efetivamente passou no teste ou não.

Teste1:cria_grafo(char *nome)
-Vai ser testado a criação do grafo.
-A entrada deve ser o nome que quero atribuir ao grafo.
-A saída vai ser a estrutura Graf criada.
-O critério para passar no teste é a estrutura ser criada.
- A função passou efetivamente no teste.

Teste2:retorna_nome_grafo(Graf)
-Vai ser testado se vamos obter o nome do grafo.
-A entrada deve ser a estrutura grafo a qual desejamos saber o nome.
-A saída vai ser o nome do grafo.
-O critério para passar no teste é retornar o nome se o mesmo tiver, ou reportar mensagem de erro se o grafo for NULL.
-A função passou efetivamente no teste.

Teste3:adiciona_vertice(Graf, int x)
-Vai ser testado se conseguimos adicionar um vértice ao grafo.
-A entrada deve ser a estrutura grafo e o valor do vértice.
-A saída vai ser o vértice inserido no grafo.
-O critério para passar no teste é que o grafo seja diferente de zero, e que adicione o vértice.
-A função passou efetivamente no teste.

Teste 4:remove_vertice(Graf, int x)

-Vai ser testado se conseguimos retirar um vértice do grafo.
-A entrada deve ser a estrutura grafo e o valor do vértice que quer que seja retirado.
-A saída vai ser o grafo sem o vértice que foi retirado.
-O critério para passar no teste é que remova o vértice.
-A função passou efetivamente no teste.

Teste 5:adiciona_aresta(Graf, int x, int y)

-Vai ser testado se conseguimos adicionar uma aresta ao grafo.
-A entrada deve ser a estrutura grafo e o valor x e y.
-A saída vai ser a aresta inserida no grafo.
-O critério para passar no teste é que adicione a aresta.
-A função passou efetivamente no teste.

Teste 6:remove_aresta(Graf, int x, int y)

-Vai ser testado se conseguimos retirar uma aresta do grafo.
-A entrada deve ser a estrutura grafo e o valor x e y.
-A saída vai ser o grafo sem a aresta que foi retirada.
-O critério para passar no teste é que remova a aresta.
-A função não passou efetivamente no teste, pois quando fazemos o teste para retirar arestas que não estão inseridas/não existem no grafo, dá segmentation fault no programa.

Teste 7:muda_valor_vertice(Graf, int x, int val)

-Vai ser testado se conseguimos mudar o valor de um vértice.
-A entrada deve ser a estrutura grafo e o valor x e val.
-A saída vai ser o grafo com o valor do vértice atualizado.
-O critério para passar no teste é mudar o valor do vértice.
-A função passou efetivamente no teste.

Teste 8:muda_valor_aresta(Graf, int x, int y, int val)

-Vai ser testado se conseguimos mudar o valor de uma aresta.
-A entrada deve ser a estrutura grafo e o valor x, y e val.
-A saída vai ser o grafo com o valor da aresta atualizado.
-O critério para passar no teste é mudar o valor da aresta.
-A função passou efetivamente no teste.

Teste 9:retorna_valor_vertice(Graf, int x)

-Vai ser testado se conseguimos obter o valor de um vértice.
-A entrada deve ser a estrutura grafo e o valor x.
-A saída vai ser o valor do vértice solicitado.
-O critério para passar no teste é retornar o valor do vértice.
-A função passou efetivamente no teste.

Teste 10:retorna_valor_aresta(Graf, int x, int y)

-Vai ser testado se conseguimos obter o valor de uma aresta.
-A entrada deve ser a estrutura grafo, o valor x e y.
-A saída vai ser o valor da aresta solicitado.
-O critério para passar no teste é retornar o valor do vértice.
-A função passou efetivamente no teste.

Teste 11:adjacente(Graf, int x, int y)

-Vai ser testado se x e y são adjacentes no grafo.
-A entrada deve ser a estrutura grafo, o valor x e y.
-A saída vai ser a resposta se são adjacentes ou não.
-O critério para passar no teste é dizer se x e y são ou não adjacentes.
-A função passou efetivamente no teste.

Teste 12:Vertice *vizinhos(Graf, int x)

-Vai ser testado se vai gerar uma lista dos vértices que podem ser visitados a partir de x.
-A entrada deve ser a estrutura grafo e o valor x.
-A saída vai ser uma lista dos vértices que podem ser visitados a partir de x .
-O critério para passar no teste é gerar a lista.
-A função passou efetivamente no teste.

Teste 13:destroi_grafo(Graf)

-Vai ser testado se vai destruir/excluir o grafo.
-A entrada deve ser a estrutura grafo.
-A saída vai ser todos os campos que eram ocupados pelo grafo como NULL agora.
-O critério para passar no teste é apagar o grafo.
-A função passou efetivamente no teste.

3) Responda em que casos a função não retorna um resultado válido.

3.1) Existem funções que podem corromper a estrutura de dados? Como?

Sim, na função destroi_grafo por exemplo, pois vamos destruir um grafo o qual tínhamos alocado um espaço de memória para ele, e se não liberarmos espaço na memória, pode ocorrer um estouro de memória.

Outro caso que pode corromper a estrutura de dados é na função remove_aresta, quando tentamos remover uma aresta inexistente, o programa apresenta uma falha de segmentação.

3.2) O que pode ser feito para evitar este problema

Na função destroi_grafo devemos dar um free() nos componentes do grafo e na estrutura do grafo.

Na função remove_aresta, não pude identificar uma maneira de resolver esse problema, e então deixei essa parte do teste comentada.