Trabalho 2

Victor Augusto Machado Clemente Souza – 12/0023474

Métodos de Programação

Turma A – Professor Jan Correa

1 Observações

Segundo o roteiro do trabalho, era necessário utilizar o framework *gtest* para a execução dos testes. Infelizmente isto não foi possível, devido a incompatibilidades com o Windows. Isto foi reportado em https://github.com/google/googletest/issues/606 e devido a todas a limitações de tempo no momento, este quesito foi deixado de lado. Entretando, há um makefile contendo as definições de compilação do código, assim como um código que eu utilizaria para testar o programa. Um teste bastante simples foi possível ser feito com gtest no Windows, mas com o aumento da complexidade do programa, estas limitações ocorreram.

2 Funções

Todas as funções estão extensamente descritas nos códigos fontes, especialmente em *../src/Digraph.c*. Estas são:

```
/*Inicialização do Digrafo.*/
Digraph DIGRAPHinit(char*);

/*Inserção e remoção de arestas.*/
void DIGRAPHinsertE(Digraph, Edge);
void DIGRAPHremoveE(Digraph, Edge);

/*Inserção e remoção de vértices.*/
void DIGRAPHinsertV(Digraph, char*);
void DIGRAPHremoveV(Digraph, char*);

/*Verificação das adjacências de um nó.*/
bool DIGRAPHadj(Digraph, Vertex, Vertex);

/*Desalocação do grafo*/
void DIGRAPHdestroy(Digraph);

/*Impressão do grafo na tela e criação de um arquivo de saída*/
void DIGRAPHshow(Digraph);
void DIGRAPHsave(Digraph);

/*Funções e algoritmos de busca. SPT de Dijkstra é utilizada.*/
float FindPath(Digraph, Vertex, Vertex);
void SPT(Digraph, Vertex, Vertex*, float*);
void initialize(Digraph, Vertex, Vertex*, float*, Vertex*);
bool isConnected(Digraph);
```

```
/*Criação de um vértice da lista de adjacência.*/
link NEWnode(Vertex, float, link);

/*Criação de uma aresta*/
Edge EDGE(int, int, float);

/*Funções auxiliares de busca de índice do vértice e remoção de vírgulas da
string.*/
int VERTEXreturn(Digraph, char*);
void removeComma(char*);
```

3 Testes

Foram feitos sete testes garantindo a funcionalidade das funções, assim como a consistência da estrutura de dados. Dessa forma, estes são:

3.1 Primeiro Teste: Busca por Vértice Inexistente

Este teste verifica se a função *VERTEXreturn* trata o caso em que um vértice inexistente no grafo será buscado e, consequentemente, não encontrado. Para isso, uma mensagem de erro foi criada. Então, ao chamar a função `v = VERTEXreturn(G, "H");` teremos que um erro será impresso e -1 será retornado a "v". Dessa forma, facilmente estende-se isso para `w = VERTEXreturn(G, "I");` e vemos que a função passou no teste.

3.2 Segundo Teste: Inserção de Aresta

Aqui são verificas as funções `DIGRAPHinsertE(G, e);`, `EDGE(v, w, weight);`, `DIGRAPHadj(G, v, w)` e `NEWnode(w, weight, link next)`. As três últimas são verificas em consequência da primeira chamada, portanto, ao tentar inserir *B-1.0->A* o teste irá falhar, já que de acordo com o grafo utilizado, "A" é uma origem. A função passou no teste. Após, foi criada a aresta *B-2.0->F*, a qual funcionou normalmente e a nova aresta foi vista ao ser impresso o grafo. Portanto, as funções passaram no teste.

3.3 Terceiro Teste: Remoção de Aresta

Primeiramente, testa a remoção de uma aresta inexistente *c-10.0->G*, a qual deve ser tratada tanto por VERTEXreturn, quanto por DIGRAPHremoveE. O teste respondeu corretamente, não removendo alguma outra aresta incorreta ou acessando memória incorretamente. Após, para garantir que também o funcionamento está correto, além do tratamento de exceções, a aresta *A-5.0->G* foi removida, sendo nenhum erro impresso e o digrafo atualizado. Função `DIGRAPHremoveE(G, e);` executada com sucesso.

3.4 Quarto Teste: Inserção de Vértice

Esta função insere um novo vértice no array de vértices do grafo. Assim, se este vértice já existir, um erro deverá ser impresso, já que isso é uma possível fonte de vazamento de memória. Dessa forma, tenta-se inserir um vértice de nome "A" no digrafo, o que é tratado com sucesso pela função `DIGRAPHinsertV(G, "A");`. Assim, ao inserir um vértice válido, `DIGRAPHinsertV(G, "H");`, a função mostra que é válida e a nova configuração do grafo foi mostrada utilizando-se `DIGRAPHshow(G);`.

3.5 Quinto Teste: Remoção de Vértice

A função de remoção `DIGRAPHremoveV(G, vertexName);` faz a remoção de um vértice com nome "vertexName" do digrafo G. Dessa forma, caso o vértice não exista, consequentemente, a função não surtirá efeito e uma mensagem de erro será impressa. Isso aconteceu ao tentar remover um vértice chamado "I" que não existe no digrafo analisado. Assim, removemos "A", "C" e "H", os quais foram excluídos com sucesso do digrafo, e algumas mensagem de erro foram impressas, mostrando que apenas os nós das listas de adjacência existentes foram removidos ao excluir os vértices, mostrando mais uma vez que o programa está tratando adequadamente exceções. Portanto, a função passou no teste.

3.6 Sexto Teste: O Nosso Digrafo é Conexo?

Aqui é utilizada a função `isConnected(G);` que retorna se o grafo é conexo ou não, ou seja, se a partir dos vértices de origem é possível chegar a todos os outros vértices. Esta função utiliza o algoritmo de árvore de menor caminho do Dijkstra (adaptado do livro Sedgewick e http://www.ime.usp.br/~pf/algoritmos_para_grafos/aulas/dijkstra.html), que foi implementado para o sétimo teste e adaptado aqui. É fácil notar que no digrafo G, já que o vértice "E" não possui lista de adjacência e nem está na lista de adjacência de qualquer outro vértice, então este digrafo realmente não é conexo. A função retorna "false" (zero), indicando o valor que estávamos esperando. Porém, defini um digrafo em "Digraph2.txt" que, por sinal, é conexo, já que a única origem é "A" e a partir dela se chega em todos os outros vetores. Portanto, o retorno da função com H como parâmetro foi "true" e nossa função foi comprovada.

3.7 Sétimo Teste: Existe Caminho?

Por último, é feito o teste de existência de caminho utilizando a função *SPT* que implementa a árvore de menor caminho (Algoritmo de Dijkstra), conforme citado acima. A função *FindPath*( ) retorna -1 caso não haja caminho, caso contrário, a menor distância entre os dois vértices entrados. Para isso, é preciso entrar com o digrafo G e dois vértices para a função. Primeiramente, testa se *B->G* existe com `float path = FindPath(G, "B", "G");`. Assim, vemos que *path* vale -1 como esperado, já que após todas operações durante os testes, *B->G* realmente não existia. Porém, verificando *B->E*, vemos que *path* retorna o valor 2.0 que, apesar de existir mais de um caminho para "E" saindo de "B", é a menor distância entre os vértices. Ou seja, o caminho existe e é a soma de todas as arestas do menor caminho entre os dois vértices. Logo, esta função também passou nos testes, comprovando que o digrafo é consistente e todas as manipulações feitas em cima dele foram válidas e corretamente tratadas.

4 Conclusões

Portanto, a construção do digrafo foi feita com sucesso, assim como a de todas as funções pedidas. No final dos testes foi impresso um arquivo "output.txt" indicando o estado final do digrafo G.