Nome: Rafael Corradini da Cunha Número: 9424322

Projeto Final da Disciplina Modelagem Computacional em Grafos

Estruturas Auxiliares:

Foram usadas neste projeto duas estruturas auxiliares, um mapa e uma lista. A lista é duplamente encadeada, e seus nós tem como elemento um ponteiro vazio, para que possa ser usada para armazenar qualquer tipo de elemento, de forma genérica. A estrutura foi usada na lista de vértices, lista de arestas e na listas de adjacências dos vértices. A lista tem quatro funções:

- initList() – Inicializa e retorna uma lista, a alocando na memória. O(1)
- freeList(L, n) – Sendo “L” a lista e “n” o nó “head”. Desaloca todos os nós da lista e a própria lista, destruindo-a. O(n)
- insertList(L) – Insere um nó no final da lista “L e o retorna para que seu elemento possa ser modificado. O(1)
- removeList(L, n) – Remove o nó “n” da lista “L”. O(1)

O mapa usado tem como base uma árvore de busca binária. Parte desse mapa foi reaproveitado de um trabalho entregue em Estrutura de Dados semestre passado. A estrutura foi usada para armazenar as referências aos vértices e as arestas do grafo(dois mapas, um para vértices, outro para arestas), com seus respectivos identificadores. O mapa tem as funções descritas abaixo:

- define() - Inicializa o mapa, aloca-o na memória, e o retorna. O(1)
- searchMap(M, id) - Retorna o elemento referente ao id do mapa “M”, caso exista. Caso contrário, retorna elemento nulo. O(n) ou O(m)
- insertMap(M, x) – Insere o elemento x no mapa “M” e retorna true(1), caso não exista elemento em “M” com a mesma chave de “x”. Se já existir

elemento em M com a mesma chave de “x”, retorna false(0). O(n) ou O(m)

- replaceMap(M, x) - Substitui por “x” o elemento do mapa “M” com a mesma chave de “x”, se tal elemento existe em “M”, retornando true(1). Caso contrário retorna false(0). O(n) ou O(m)
- removeMap(M, id) - Remove e retorna o elemento do id, caso ele exista no mapa “M”. Caso contrário retorna um elemento nulo. O(n) ou O(m)
- removeEdgesMap(M, id) - Remove as arestas incidentes ao vértice do id. O(n^2)
- distVector(M, D, P) – Inicializa os vetores “D” e “P”, colocando as distâncias entre os vértices adjacentes em “D” e os parentescos entre esses vértices. O(m)
- printMap(M, op) - Imprime a árvore na tela todos os elementos do mapa “M”. O(n) ou O(m)
- freeNodesMap(M) – Desaloca todos os nós do mapa “M”(Não desaloca a árvore, a mesma deve ser desalocada em seguida). O(n) ou O(m)
- getLastMap(T) – Retorna o maior identificador armazenado na árvore. O(1)

Obs: As complexidades são apresentadas como O(n) ou O(m) porque no projeto usou-se o mapa em duas ocasiões: no armazenamento dos identificadores dos vértices, e dos identificadores das arestas, portanto no primeiro caso essas operações seriam O(n) e no segundo caso O(m). Se tem a complexidade O(n) porque a árvore usada não é balanceada, portanto, como os elementos são inseridos sempre com seus identificadores em ordem crescente, a árvore se desencadeia em uma lista.

Estrutura do grafo:

O grafo consiste na manipulação de duas listas, a de vértices e a de arestas, as quais são referenciadas pelo elemento(struct) principal da estrutura chamado “Graph”.
A lista de vértices armazena elementos nomeados de “Vertex”, cada “Vertex”

armazena um inteiro “value” e faz referência a uma lista de adjacências. As listas de adjacências armazenam elementos do tipo “AdjVertex”, que representam os vértices adjacentes ao vértices que armazena a lista, o elemento “AdjVertex” armazena uma referência a esse vértice adjacente e outra referência a aresta que incide nos dois. A lista de arestas armazena elementos nomeados de “Edge”, cada “Edge” armazena um double “value” que representa o peso da aresta, e quatro ponteiros: dois ponteiros que fazem referência aos nós da lista de vértices onde se encontram os vértices nos quais a aresta incide(v1 e v2), e dois ponteiros(adj_v1 e adj_v2) que fazem referência aos mesmos vértices, porém nas listas de adjacências.

Operações do TAD grafo:

- initGraph() - A função aloca o grafo, a lista de arestas e a lista de vértices na heap, utilizando malloc e a função initList() do TAD lista, então retorna um ponteiro que faz referência ao grafo. O(1)
- freeGraph(G) – Desaloca a lista de arestas do grafo “G” e tudo que ela armazena, após faz o mesmo com todas as listas de adjacências de todos os vértices e com a lista de vértices do grafo. Todas as listas são desalocadas da heap com o auxílio da função freeList() do TAD lista. O(m+n(degree(n))+n)
- opposite(G, v, e) – Verifica se um dos ponteiros passados é nulo, se não for retorna o vértice oposto a “v”, através da aresta “e”. O(1)
- areAdjacent(G, v, w) – Compara o tamanho das duas lista de adjacências, de “v” e de “w”, e então percorre a menor delas em busca de um elemento que faça referência ao vértice que seria adjacente ao vértice “dono” da lista, retorna 1 se encontrar o vértice, caso contrário retorna 0. O(min(deg(v), deg(w)))
- replaceVertex(G, v, o) – Verfica se o ponteiro passado é nulo, se não for substitui o valor do campo “value” do vértice “v” por “o”. O(1)
- replaceEdge(G, e, o) – Análogo ao anterior, substitui o “value” da aresta “e”. O(1)

- insertVertex(G, o) – Insere um vértice na lista de vértices do grafo, alocando-o na heap, usando o insertList do TAD lista, aloca também o elemento “Vertex” referenciado pelo nó da lista e atribui o valor “o” ao campo “value” do mesmo, e aloca a lista de adjacências do vértice. O(1)

- insertEdge(G, v, w, o) – Verifica se os ponteiros passados não são nulos, se forem nulos retorna nulo, caso contrário insere uma aresta na lista de arestas do grafo, alocando-a na heap, usando o insertList do TAD lista, aloca também o elemento “Edge” referenciado pelo nó da lista, atribui então “o” ao “value” da aresta, “v” ao campo “v1” do elemento e “w” ao campo “v2”. Então, insere um nó na lista de adjacências de “w”, este nó tem um ponteiro que faz referência ao vértice “v”, e atribui o endereço desse novo nó ao campo “adj_v1”. Faz o processo análogo para o vértice “v”, adicionando-o na lista de adjacências do vértice “w, atribuindo o endereço desse ao campo “adj_v2” da aresta. Retorna um ponteiro que faz referência a aresta inserida. O(1)

- removeVertex(G, v) - Verifica se o ponteiro passado não é nulo, se for retorna -1 ecerrando, se não for prossegue e então remove todas as arestas que incidem no vértice “v” e, usando a função freeList() do TAD lista, desaloca a lista de adjacências do vértice. A seguir, remove o vértice da lista de vértices do grafo “G”, usando a função removeList() do TAD lista, desalocando-o, e desaloca também o elemento “Vertex” da heap usando a função free() . Retorna o campo “value” antes armazenado em “v”. O(deg(v))

- removeEdge(G, e) - Verifica se o ponteiro passado não é nulo, se for retorna -1 ecerrando, se não for prossegue e remove os vértices das listas de adjacências referenciados pelos campos “adj_v1” e “adj_v2”, do elemento armazenado em “e”, deslocando-os utilizando a função removeList() do TAD lista e desalocando os elementos “AdjVertex”, armazenados pelos mesmos, utilizando a função free(). Retorna o campo “value” antes armazenados em “e”. O(1)

- vertexValue(G, v) – retorna o valor do campo “value” referente ao elemento “Vertex” armazenado no endereço “v”. O(1)

- edgeValue(G, e) – retorna o valor do campo “value” referente ao elemento

“Edge” armazenado no endereço “e”. O(1)

- numVertices(G) – Retorna a quantidade de vértices armazenados na lista de vértices do grafo “G”, valor que é armazenado na estrutura do TAD lista, no campo count. O(1)
- numVertices(G) – Retorna a quantidade de arestas armazenados na lista de arestas do grafo “G”, valor que é armazenado na estrutura do TAD lista, no campo count. O(1)