Trabalho 3 – Nemo Travels Back Home

Estrutura de Dados e Algoritmos II

Realizado por:

Nome do utilizador no Mooshak: g326
Tiago Pinto 54718
Curso: Engenharia Informática

Introdução

Com este trabalho pretende-se utilizar a linguagem Java para resolver o problema “Nemo travels back home”, que consiste em ajudar o Nemo e os seus amigos a encontrar o seu caminho para casa, tendo em conta as direções e as forças das correntes do oceano. O programa deve determinar o melhor caminho a ser seguido, consumindo o mínimo de energia para chegar a cada local.

Implementação

Como este trabalho trata-se numa solução baseada em grafos, inicialmente foram criadas duas classes para ser possível trabalhar com eles.

Class Graph

Esta classe tem a função de construir um grafo pesado e representá-lo por listas de adjacência.
Esta classe também é capaz de adicionar uma aresta ao grafo, a partir do método addEdge.

Class Edge

Esta classe é capaz de colocar um destino e um peso a cada aresta do grafo.

Class Main

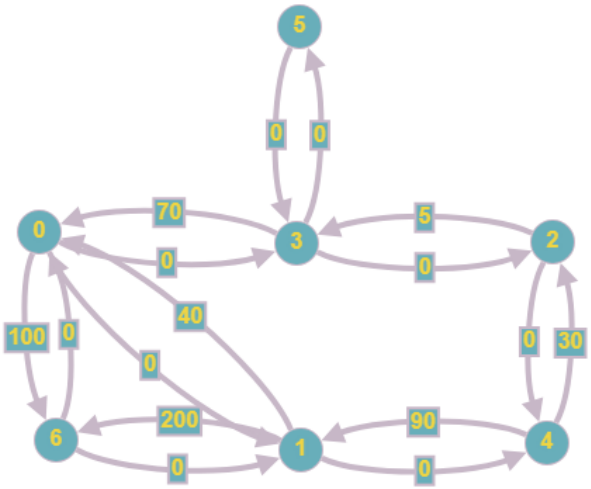
Após a implementação das classes anteriores, foi implementado um método que realiza o algoritmo de “dijkstra” que serve para calcular os caminhos mais curtos de um vértice, para os restantes vértices do grafo. Neste método, primeiramente é colocado todas as distâncias com valor infinito, para indicar que são de valor desconhecido, e depois colocado o valor da distância de um vértice para si mesmo, como valor 0. De seguida, foi criado uma fila de prioridade que é capaz de ordenar os vértices na ordem das suas menores distâncias conhecidas, e colocado o vértice “fonte” nessa fila com distância 0. Seguidamente, ocorre um loop que ocorre até não haver mais vértices na queue. Neste loop primeiramente acontece a extração do vértice com a menor distância e depois verifica-se se a distância “distU” (menor distância conhecida até ao vértice “u”) é maior que a

distância atualmente registada em distancia[u] (“u” é o vértice que está a ser processado atualmente), caso se verifique significa que esse par já foi processado com uma menor distância, então o for é pulado pelo continue. Caso não se verifique, então dá se o relaxamento das arestas, onde estas são atualizadas conforme necessário.

Por fim, para além do método anterior, na main também ocorre a leitura dos valores dados no input (L = locations, C = currents, J = journeys) e depois então ocorre um loop, que constrói o grafo. Depois ainda ocorrem mais dois loops, o primeiro para ler todas as jornadas que foram pedidas no input e outro que serve para chamar o método “dijkstra” e dar o output pedido.

Grafo Utilizado

Como este trabalho trata-se de uma solução baseada em grafos, a figura seguinte, mostra o grafo que foi utilizado para resolver este problema e que também se encontra no enunciado. O grafo pesado e orientado, representa um oceano com várias localizações (vértices), que estão conectados por correntes (arestas), que podem ter valor 0 (caso o caminho seja a favor da corrente), ou valor diferente de 0 (caso seja contra a corrente).



Cálculo das Complexidades

Complexidade Temporal:

Na classe graph existe um loop que percorre todos os nós (L), para inicializar as listas de adjacência para cada um, por isso temos complexidade O(L).
Na main encontra-se o algoritmo de “dijkstra” que possui complexidade O((L + C) log L) e como este algoritmo é realizado para cada uma das jornadas, a total fica J(O(L + C) log L).

Ainda há na main, um loop, para a construção do grafo que percorre todas correntes, com complexidade O(C).
No total a complexidade temporal fica O(L+ C + J(L+C) log L).

Complexidade Espacial:

Quanto á complexidade espacial, na classe Graph existe uma lista de adjacência que tem complexidade O(L + C).
Na main, antes de ocorrer a leitura dos dados, são criados dois arrays (local1, local2) com tamanho J, ou seja, com complexidade O(J).
A priority queue apresenta complexidade O(L), pois esta no pior dos casos pode conter L elementos.
No total a complexidade espacial fica O(L+ C + J).