Resolução do problema EP01

Geraldo Rodrigues de Melo Neto
Gustavo Duarte Ventino
Maria Luisa Gabriel Domingues
Pedro de Araújo Ribeiro
Lucas Marques Pinho Tiago

O Problema:

A empresa Pacific Island Net (PIN) identificou várias ilhas do pacifico que não possuem uma conexão boa, seu objetivo é criar conexões entre as ilhas de forma que cada ilha tenha alguma conexão com a ilha principal. Os cabos são construídos a uma velocidade de 1km por dia, a PIN também está interessada em saber qual a velocidade média que demora para novos clientes terem acesso a uma internet mais rápida.

Entrada:

O input é dividido da seguinte forma:

- A primeira linha possui um valor **N** que representa o número de ilhas.

As **N** linhas seguintes devem possuir três valores **X**, **Y** e **M** em que (**X**,**Y**) são as coordenadas de cada ilha no mapa e **M** é o número de habitantes da ilha**.**

- A última linha deve terminar com 0 indicando o fim do input, nota-se que é possível que existam vários casos teste.

Saídas:

O programa deve fornecer um output para cada um dos casos teste do input, ou seja, o programa deve ter um ou mais outputs.

O output é o resultado da equação de tempo médio: **(∑ ti ∗ mi)/ ∑ mi**

- **Ti** é a distância de uma ilha a outra:
- **Mi** é a quantidade de habitantes de uma ilha.

Exemplo:

Entrada:

```
7
11 12 2500
14 17 1500
9 9 750
7 15 600
19 16 500
8 18 400
15 21 250
0
```

Saída:

```
Island Group: 1 Average 3.20
```

Nossa Resolução:

- Graph.hpp : Contém, de atributos, um array com os vértices, um array com as arestas e o número de vértices.
  - readGraph ( ) => Leitura e criação do grafo.
  - kruskal ( ) => Implementa algoritmo de kruskal para MST's.
- Vertex.hpp : Contém, de atributos, x, y e id;
- Edges.hpp : Vértice a, Vértice b e peso da aresta.
- UFDS.hpp : Contém, de atributo um array de inteiros que auxilia na lógica na hora da construção da MST por meio do Kruskal.
  - setRoot (int i, int j) => Configura o conjunto de j como conjunto de i.
  - findRoot (int i) => Encontra o conjunto a que i pertence.
  - isConnected(int i, int j) => Verifica se conjunto[i] == conjunto[j].

```hpp
#include "Edges.hpp"
#include <vector>
#include <utility>
#include <iostream>
class Graph
{
  public:
  //ATRIBUTOS:
  std::vector<Edges*> arestas;
  std::vector<Vertex*>vertices;
  int numVertices;

  //METODOS:
  //Construtores:
  Graph();
  Graph(int size);

  //Faz toda a leitura de entrada e cria o grafo:
  static Graph* readGraph(int n);
  Vertex* getByValue(int x, int y);
  //Adiciona novo vertice na lista de vertices:
  void addEdge(Edges* e);
  bool pertence(int x,int y);
  //Imprime todo o grafo:
  void print();
  //Kruskal:
  Graph* kruskal(double *res);
};
```

```hpp
#include "Vertex.hpp"
class Edges
{
  public:
  Vertex *a;
  Vertex *b;
  double peso;
  //Construtores:
  Edges();
  Edges(Vertex* a, Vertex* b, double peso);
  //Printa as informacoes de id e valor do vertice atual
  void print();
};
```

```hpp
#include <vector>
#include <string>
class Vertex
{
  public:
  int x;
int y;
  int id;
  //Construtores:
  Vertex();
  Vertex(int x, int y, int id);
  //Printa as informacoes de id e valor do vertice atual
  void print();
};
```

```hpp
#include <iostream>
#include <vector>

class UFDS{
public:
  //ATRIBUTOS:
  std::vector<int> marked;
  //METODOS:
  UFDS();
  UFDS(int size);
  bool isConnected(int i, int j);
  void setRoot(int i, int j,int w1[], int w2[]);
  int findRoot(int i);
};
```

Método para resolução do problema:

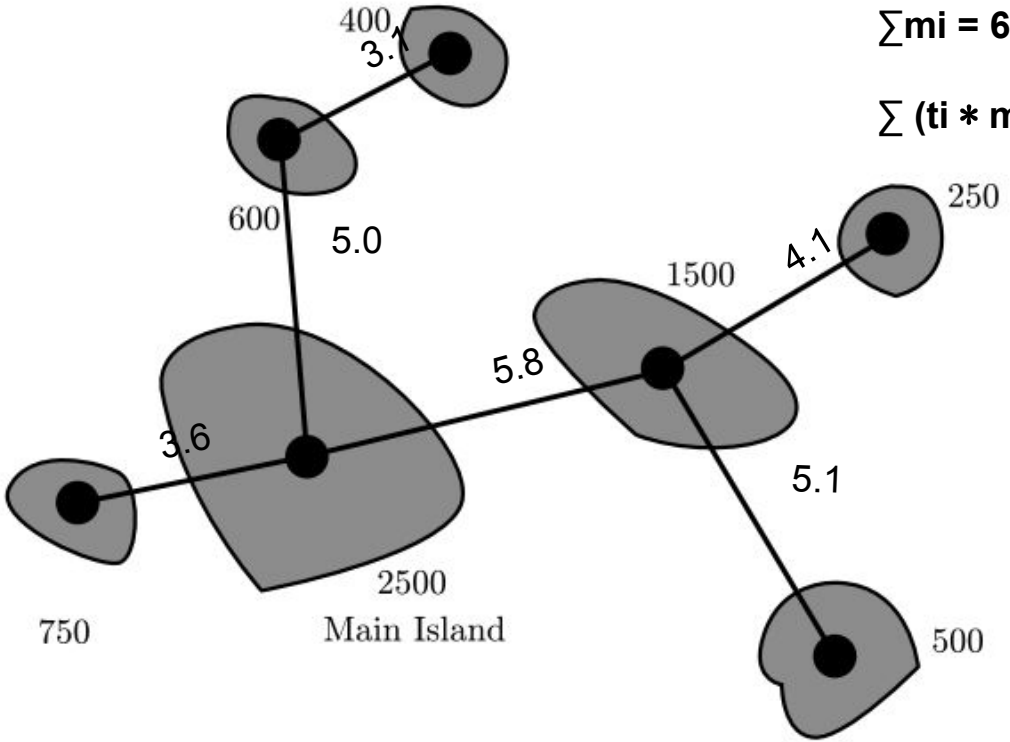
- Leitura dos vértices do grafo e construção deste:
  - Usamos para isso a função Graph* readGraph( ) da nossa classe de Grafo descrita em Graph.hpp e Graph.cpp, essa função é responsável por realizar o cálculo das distâncias e conectar diretamente todos os vértices do grafo, é importante destacar que essa função trata casos de vértices repitidos.
- Encontrando menor caminho:
  - A nossa função Graph* kruskal( ) monta a árvore geradora mínima do nosso grafo e a retorna, isso implica em dizer que, todos os vértices estão conectados, sem ciclos e com menor peso total de arestas possível.
  - Sempre que um novo vértice é inserido no Kruskal a função void setRoot() realiza um rebalanceamento no peso de cada vértice e realiza o cálculo do tempo médio que leva para um habitante ter acesso a internet

Exemplo de um grafo e do Cálculo:

∑ (ti ∗ mi) = 5.8*(2250) + 5.0*(1000) + 3.6*(750) = 20 750

∑mi = 600 + 400 + 750 +1500 +2500 + 250 + 500 = 6 500

∑ (ti ∗ mi)/∑mi = 3.20



```cpp
Graph *Graph::kruskal(double *res) {
  // ordenar as arestas do grafo por peso;
  std::stable_sort(arestas.begin(), arestas.end(),
                   [](Edges *e1, Edges *e2) // funcao lambda
                   { return e1->peso < e2->peso; });

  Graph *g = new Graph();
  // Criar grafo vazio;
  double soma = 0;
  int hab = 0, h1[50], h2[50];
  int i;
  //Criar lista com o tam de cada sub arvore e do numero de habitantes dela
  for (Vertex *it : vertices) {
    //hab é o total de habitantes
    hab += it->inhabitants;
    h2[it->id] = it->inhabitants;
    h1[it->id] = 1;
  }
  UFDS *ufds = new UFDS(numVertices);
  for (Edges *it : arestas) {
    // pega o id de cada vertice da aresta
    int u = it->a->id;
    int v = it->b->id;
    // pega a raiz dos vertices
    int uRep = ufds->findRoot(u);
    int vRep = ufds->findRoot(v);
    // atualiza a raiz e adicionar a aresta no grafo
    if (uRep != vRep) {
      //realizando calculo Peso*Habitantes
      if (uRep == ufds->findRoot(0)) {
        soma += h2[vRep] * it->peso;
      }
      if (vRep == ufds->findRoot(0)) {
        soma += h2[uRep] * it->peso;
      }
      //Junta as sub arvores e altera no H1 e H2
      //o tamanhp da arvore atual e o numero de habitantes
      ufds->setRoot(u, v, h1, h2);
      g->addEdge(it);
    }
  }
  *res = soma / hab;
  return g;
}
```

```cpp
Graph *Graph::readGraph(int n) {
  Graph *g;
  Edges *edge;
  Vertex *v1 = NULL;
  Vertex *v2 = NULL;
  int entry1, entry2, entry3;
  int id = 0;
  int i;
  g = new Graph(n);
  for (i = 0; i < n; i++) {
    //pega o X, Y e o numero de habitantes
    std::cin >> entry1;
    std::cin >> entry2;
    std::cin >> entry3;
    v1 = new Vertex(entry1, entry2, id, entry3);
    id++;
    //verifica se o vertice já existe
    if (g->pertence(entry1, entry2)) {
      v1 = g->getByValue(entry1, entry2);
      v1->inhabitants += entry3;
      id--;
    } else {
      g->vertices.push_back(v1);
    }

    // conecta
    for (int j = id - 1; j >= 0; j--) {
      v2 = g->vertices[j];
      // calculando menor distancia entre dois pontos
      // pontos (v1x,v1y) e (v2x,v2y)
      double a = v2->x - v1->x;
      double b = v2->y - v1->y;
      a = a * a;
      b = b * b;
      edge = new Edges(v1, v2, sqrt(a + b));
      g->addEdge(edge);
    }
  }
  return g;
}
```

FIM

Referências:

Repositório com os códigos fonte: https://replit.com/@Ventinos/S06EP01