PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS
ICEI - INSTITUTO DE CIÊNCIAS EXATAS E INFORMÁTICA
ALGORITMOS EM GRAFOS - PROFESSOR KLEBER JACQUES FERREIRA DE SOUZA

Arthur Andrade Gonçalves

Assuério Batista dos Santos

Lucas Braga Ferreira

Marcos Pablo Souza de Almeida

Rodrigo Gonçalves Ribeiro

Samuel Lucas Oliveira Martins

REDE ÓPTICA

COMO CONECTAR TABAS INDÍGENAS GERANDO O MENOR IMPACTO AMBIENTAL POSSÍVEL

Contagem
2019

SUMÁRIO

1 INTRODUÇÃO……..…...…….**3**

2 PROBLEMA…….……..……..**3**

2.1 OBSERVAÇÕES…….……..……...**3**

2.2 EXEMPLO DE SOLUÇÃO NECESSÁRIA……...**4**

2.3 ENTRADA DE DADOS……..…….... **3**

2.4 SAÍDA DE DADOS…….…..…….. **4**

3 SOLUÇÃO DO PROBLEMA……..…….…….……... **5**

3.1 MODELAGEM DO PROBLEMA…….……...**5**

3.4 CODIFICAÇÃO…….……..…….....**6**

4 TESTES…….……...……..**11**

5 RESULTADOS OBTIDOS…….**11**

6 VALIDAÇÃO FINAL……**12**

1. INTRODUÇÃO

Baseados em conceitos matemáticos, os algoritmos em grafos foram desenvolvidos a fim de solucionar problemas do mundo real. Apresentam soluções a diversas dificuldades enfrentadas no cotidiano, as quais podem ser definidas como distribuir uma rede de computadores de forma otimizada, encontrar o menor caminho possível entre dois pontos, colorir o mapa de um país de tal forma que os estados adjacentes não possuam a mesma cor e etc.

2. PROBLEMA

Utilizando a teoria dos grafos, foi solicitada a solução para o seguinte impasse:

"Os caciques da região de Tutuaçu pretendem integrar suas tribos à chamada 'aldeia global'. A primeira providência foi a distribuição de telefones celulares a todos os pajés. Agora, planejam montar uma rede de fibra ótica interligando todas as tabas. Esta empreitada requer que sejam abertas novas picadas na mata, passando por reservas de flora e fauna. Conscientes da necessidade de preservar o máximo possível o meio ambiente, os caciques encomendaram um estudo do impacto ambiental do projeto. Será que você consegue ajudá-los a projetar a rede de fibra ótica?"

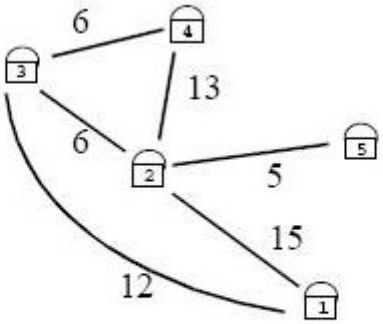
(fonte: https://br.spoj.com/problems/REDOTICA/)

2.1 OBSERVAÇÕES

O impacto ambiental deve ser mitigado ao máximo possível, de tal modo que ainda seja possível estabelecer a conexão entre todas as tabas da aldeia. Dessa forma, deve existir ao menos um caminho entre qualquer par de tabas existentes

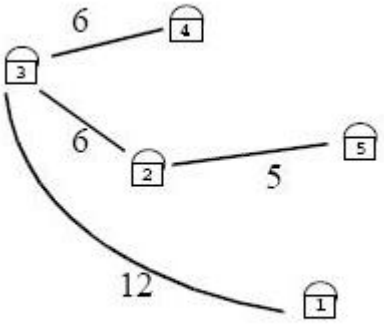
2.2 EXEMPLO DE SOLUÇÃO NECESSÁRIA

Dado o conjunto de tabas apresentado na imagem 1, a solução deve apresentar o resultado contido na imagem 2.



Ramos de rede possíveis com custo ambiental associado

Imagem 1



Interligação das tabas com menor custo ambiental

Imagem 2

2.3 ENTRADA DE DADOS

A entrada é composta de vários conjuntos de teste. A primeira linha de um conjunto de teste contém dois números inteiros positivos N e M que indicam, respectivamente, o número de tabas e o número de ramos de redes possíveis. As tabas são numeradas de 1 a N. As M linhas seguintes contém três inteiros positivos X, Y e Z, que indicam que o ramo de rede que liga a taba X à taba Y tem impacto ambiental Z. Com os conjuntos de teste dados sempre é possível interligar todas as tabas. O final da entrada é indicado quando N = 0.

2.4 SAÍDA DE DADOS

Para cada conjunto de teste da entrada seu programa deve produzir uma lista dos ramos de redes que devem ser construídos. A lista deve ser precedida de uma linha que identifica o conjunto de teste, no formato "Teste n", onde n é numerado a partir de 1. A lista é composta por uma sequência de ramos a serem construídos, um ramo por linha. Um ramo é descrito por um par de tabas X e Y , com X < Y. Os ramos de rede podem ser listados em qualquer ordem, mas não deve haver repetição. Se houver mais de uma

solução possível, imprima apenas uma delas. O final de uma lista de ramos deve ser marcado com uma linha em branco. A grafia mostrada no Exemplo de Saída, abaixo, deve ser seguida rigorosamente.

3. MODELAGEM E SOLUÇÃO DO PROBLEMA

Para solucionar problemas baseados em encontrar o menor custo de conexão entre diferentes pontos, a Teoria dos Grafos apresenta soluções plausíveis de serem implementadas em computadores em forma de algoritmos.

Sendo assim, o problema apresentado foi estudado e as entradas de dados foram analisadas. Após isso, a melhor solução encontrada pelo grupo foi criar uma Árvore Geradora Mínima, em que os vértices do grafo são definidos pelas tabas e o peso das arestas que os conectam é definido pelo impacto ambiental.

A fim de obter a Árvore Geradora Mínima de um grafo, dois algoritmos conhecidos apresentam um bom desempenho e possuem o mesmo resultado, sendo esses: Kruskal e Prim. A priori, escolhemos o Algoritmo de Kruskal, o qual efetua a conexão de todos os vértices do grafo com o menor custo possível, sem gerar ciclos. Entretanto, tendo em vista que solucionar o problema dessa maneira seria uma tarefa simples, optamos por não utilizar nenhum destes algoritmos, desenvolvendo outra solução.

Portanto, a solução desenvolvida baseia-se no algoritmo Union-Find, o qual mantém o controle de um conjunto de elementos particionados em subconjuntos disjuntos (não sobre-posicionados). Ele fornece operações com tempo quase constante (delimitadas pela inversa função de Ackermann) para adicionar novos conjuntos, para mesclar conjuntos existentes e para determinar se os elementos estão no mesmo conjunto.

3.1 CODIFICAÇÃO

O algoritmo foi desenvolvido em C# e encontra-se abaixo:

```
using System;
public class Test

{

    class Edge

    {

        public Node node1 { get; set; }

        public Node node2 { get; set; }

        public int height { get; set; }

    }

    class Node

    {

        public int uuid { get; set; }

        public int order { get; set; }

        public Node(int uuid)

        {

            this.uuid = uuid;

            order = uuid;

        }

    }

    static void Sort(Edge[] edges)

    {

        for (int i = 0; i < edges.Length; i++)

        {

            if (edges[i].node2.order < edges[i].node1.order)

            {

                Node aux = edges[i].node1;
```

```
            edges[i].node1 = edges[i].node2;
            edges[i].node2 = aux;
        }
        for (int j = 0; j < edges.Length - i - 1; j++)
        {
            if (edges[j].height > edges[j + 1].height)
            {
                Edge aux = edges[j];
                edges[j] = edges[j + 1];
                edges[j + 1] = aux;
            }
        }
    }
    for (int i = 0; i < edges.Length - 1; i++)
    {
        for (int j = 0; j < edges.Length - i - 1; j++)
        {
            if (edges[j].height == edges[j + 1].height)
            {
                if (edges[j].node1.order > edges[j + 1].node1.order)
                {
                    Edge aux = edges[j];
                    edges[j] = edges[j + 1];
                    edges[j + 1] = aux;
                }
            }
        }
    }
```

```
}
static void Main(string[] args)
{
    int num = 1;
    string response = "";
    string line;
    string[] splitLine;
    Edge[] spanningTree;
    Edge[] edges;
    Node[] nodes;
    line = Console.ReadLine();
    splitLine = line.Split(' ');
    while (splitLine[0] != "0" && splitLine[1] != "0")
    {
        int n = int.Parse(splitLine[0]);
        int m = int.Parse(splitLine[1]);
        nodes = new Node[n];
        for (int i = 0; i < nodes.Length; i++)
        {
            nodes[i] = new Node(i + 1);
        }
        edges = new Edge[m];
        for (int i = 0; i < edges.Length; i++)
        {
            edges[i] = new Edge();
        }
        spanningTree = new Edge[n - 1];
        for (int i = 0; i < m; i++)
```

```
            {
                line = Console.ReadLine();
                splitLine = line.Split(' ');
                edges[i].node1 = nodes[int.Parse(splitLine[0]) - 1];
                edges[i].node2 = nodes[int.Parse(splitLine[1]) - 1];
                edges[i].height = int.Parse(splitLine[2]);
            }

            Sort(edges);

                    for (int i = 0, limit = 0; i < edges.Length && limit <
spanningTree.Length; i++)
            {
                if (i == 0)
                {
                    spanningTree[limit] = edges[i];
                                                spanningTree[limit].node2.uuid   =
spanningTree[limit].node1.uuid;
                    limit++;
                }
                else
                {
                    if (edges[i].node1.uuid != edges[i].node2.uuid)
                    {
                        spanningTree[limit] = edges[i];
                        for (int j = 0; j < nodes.Length; j++)
                        {
                            int uuid = spanningTree[limit].node2.uuid;
                            if (nodes[j].uuid == uuid)
```

```
                                nodes[j].uuid = spanningTree[limit].node1.uuid;
                            }
                            limit++;
                        }
                    }
                }

                response += ("Teste " + num);
                num++;
                for (int i = 0; i < spanningTree.Length; i++)
                {
                    response += ("|" + spanningTree[i].node1.order + " " +
spanningTree[i].node2.order + "|");
                }
                response += ";";

                line = Console.ReadLine();
                splitLine = line.Split(' ');
            }

            string[] splitResponse = response.Split(';');
            for (int i = 0; i < splitResponse.Length; i++)
            {
                string[] aux = splitResponse[i].Split('|');
                for (int j = 0; j < aux.Length; j++)
                {
                    Console.WriteLine(aux[j]);
                }
```

```
            }
        }
    }
```

4. TESTES

Para testar a efetividade do algoritmo, utilizamos o seguinte caso teste:

3 3
1 2 10
2 3 10
3 1 10
5 6
1 2 15
1 3 12
2 4 13
2 5 5
3 2 6
3 4 6
0 0

5. RESULTADOS

O caso teste usado apresentou seguinte saída:

Teste 1
1 2
1 3
Teste 2
2 5
2 3
3 4
1 3

6. VALIDAÇÃO FINAL

Por fim o programa foi submetido ao site para ser validado. Conforme mostrado na imagem abaixo, a solução desenvolvida foi aceita pela plataforma.