PUC Minas
A B A K Ó S
Instituto de Ciências Exatas e Informática

Gabriel Victor Couto Martins de Paula[1]
Lucas Paixão Soares Ribeiro[2]
Luís Antônio de Souza e Sousa[3]

Trabalho Prático 02 - Teoria de Grafos

Caminhos Disjuntos

[1]Bacharel em Engenharia de SoftwareInstituto de Ciências Exatas e de Informática da PUC Minas, Brasil– gabriel.paula.1265840@sga.pucminas.br
[2]Bacharel em Engenharia de Software, E-mail:lpsribeiro@sga.pucminas.br
Instituto de Ciências Exatas e de Informática da PUC Minas, Brasil.
[3]Bacharel em Engenharia de Software, E-mail:lassousa@sga.pucminas.br
Instituto de Ciências Exatas e de Informática da PUC Minas, Brasil.

1 INTRODUÇÃO

O trabalho a seguir vai tratar da solução do problema da determinação do número máximo de caminhos disjuntos em arestas existentes em um grafo. Especificamente, um método de resolução que receba um grafo e um par de vértices de origem e destino, exibindo a quantidade de caminhos disjuntos em arestas entre os dois vértices dados, além de listar cada um dos caminhos encontrados.

2 DESENVOLVIMENTO

As instâncias cujo os nomes terminam em "d", indicam que se trata de um grafo direcionado

Tabela 1 – Relação de instancias e quantidade de arestas

Instância	Numero de Arestas
elist1440.rmf	5397
elist1440d.rmf	22128
elist160.rmf	285
elist160d.rmf	912
elist200.rmf	483
elist200d.rmf	1340
elist2560.rmf	44160
elist500.rmf	1040
elist500d.rmf	3975
elist640.rmf	3037
elist640d.rmf	12608
elist96.rmf	187
elist960.rmf	2061
elist960d.rmf	9488

Fonte: COUTO, 2022

Como instâncias de teste, foram utilizados grafos que originalmente foram gerados para processamento de fluxo máximo provenientes da OR Library. Em cada um dos arquivos existe um conjunto de dados nas primeiras linhas, que são metadados do grafo como número de vértices, número de arestas, vértice de origem e vértice de destino:

Tabela 2 – Metadados das instancias

n	Número de vértices
m	Número de arestas
s	Identificação do vértice de origem
t	Identificação do vértice de destino

Fonte: COUTO, 2022

As linhas seguintes descrevem as arestas e direções e peso entre os vértices como:

Tabela 3 – Descrição dos dados da arestas na instância

v	w	cap
Vértice de origem	Vértice de destino	Peso da aresta

Fonte: COUTO, 2022

Para os objetivos do trabalho, os pesos das atribuídos as arestas são irrelevantes para a solução final, por isso, apesar dos valores serem lidos, não são levados em consideração no processamento.

Para o processamento da solução foi escolhida a linguagem Java como ferramenta principal, com ela foi escrito um único programa que utiliza uma classe principal *Main* desenvolvida para instanciar e armazenar os atributos do grafo de maneira estática para cada instância. As arestas e seus custos foram dispostos em uma matriz simétrica de números inteiros cuja quantidade de linhas e colunas é a mesma do número de vértices.

A solução utilizada foi uma variação da implementação de *Edmonds-Karp* do método de *Ford-Fulkerson* para se calcular o fluxo máximo em uma rede de fluxo. Ao calcular o fluxo máximo da aresta de origem até a de destino é possível encontrar o número máximo de caminhos disjuntos das arestas.

Algorithm 1 Encontrar Caminhos Disjuntos com o algoritmo de Ford-Fulkerson

```
1  static int EncontrarCaminhosDisjuntos(int graph[][], int s, int t) {
2        V = graph.length;
3        int u, v;
4        int[][] rGraph = new int[V][V];
5        for (u = 0; u < V; u++)
6           for (v = 0; v < V; v++)
7              rGraph[u][v] = graph[u][v];
8        int[] anterior = new int[V];
9        int fluxo_maximo = 0;
10       while (bfs(rGraph, s, t, anterior)) {
11          int fluxo_do_caminho = Integer.MAX_VALUE;
12          for (v = t; v != s; v = anterior[v]) {
13             u = anterior[v];
14             fluxo_do_caminho = Math.min(fluxo_do_caminho,
                  rGraph[u][v]);
15          }
16          for (v = t; v != s; v = anterior[v]) {
17             u = anterior[v];
18             rGraph[u][v] -= fluxo_do_caminho;
19             rGraph[v][u] += fluxo_do_caminho;
20          }
21          caminhos.put(numCaminhos++, getPath(anterior, t));
22          fluxo_maximo += fluxo_do_caminho;
23       }
24       return fluxo_maximo;
25    }
```

Algorithm 2 Marcação de vertices já visitados

```
1   static boolean bfs(int rGraph[][], int s, int t, int anterior[]) {
2
3       V = rGraph.length;
4       boolean[] visitado = new boolean[V];
5       Queue<Integer> q = new LinkedList<>();
6       q.add(s);
7       visitado[s] = true;
8       anterior[s] = -1;
9
10      while (!q.isEmpty()) {
11          int u = q.peek();
12          q.remove();
13
14          for (int v = 0; v < V; v++) {
15              if (visitado[v] == false && rGraph[u][v] > 0) {
16                  q.add(v);
17                  anterior[v] = u;
18                  visitado[v] = true;
19              }
20          }
21      }
22      return visitado[t];
23  }
```

3 CONCLUSÃO

Ao se utilizar uma solução baseada no algoritmo de *Edmonds-Karp*, cuja complexidade de tempo do pior caso é $O(|V||E|^3)$, a solução entrega informações como a quantidade de caminhos disjuntos para cada instância e registra os caminhos em um *HashSet* da biblioteca java.