Projeto Final de Teoria de Grafos TEG

Lucas M. Pereira[1], **Rafael S. Pereira**[1]

[1]Bacharel de Ciência da Computação – Universidade Estadual de Santa Catarina (UDESC)
Joinville – SC – Brasil

Abstract. *The article serves as a basis for the subject of Graph Theory, where an algorithm developed by the authors will be presented to solve a hypothetical situation. Using techniques and algorithms seen in the discipline, to allow walking through the graph as well as evaluate the minimum cut to achieve the goal. Graph theory concepts such as Ford Fulkerson graphs, pairing, minimum cut and network flow, as well as the C ++ programming language were used.*

Resumo. *O artigo serve como base para a matéria de Teoria de Grafos, onde estará sendo apresentado um algoritmo desenvolvido pelos autores para resolução de uma situação hipotética. Utilizando técnicas e algoritmos vistas na disciplina, para permitir caminhar pelo grafo bem como avaliar o corte mínimo para atingir o objetivo. Foram utiliza-dos conceitos de teoria dos grafos como grafos Ford Fulkerson, emparelhamento, corte mínimo e fluxo de rede, bem como a linguagem de programção C++.*

1. Introdução

O objetivo deste artigo é apresentar os algoritmos feitos pelos alunos, utilizando técnicas presentes na matéria de Teoria de Grafos, para a resolução de uma situação hipotética, apresentada pelo professor como trabalho final para a matéria.

A situação apresentada pelo professor consiste percorrer um grafo G qualquer, onde em duas pontas existiram nós fixos e imutáveis, e o resto do grafo possuirá uma valoração, tanto nós quanto as arestas. O objetivo será remover o menor numero de vértices e/ou arestas para deixar o grafo desconexo, em relação aos dois nós fixos. Desconectando dentro do contexto o computador de uma empresa do seu principal servidor.

2. Metodologia

Para realizar a solução do problema foi produzido um algoritmo utilizando técnicas e conceitos de teoria dos grafos como: Grafo Ford Fulkerson, emparelhamento, corte mínimo e fluxo de rede, o código foi produzido utilizando a linguagem C++.

2.1. $calcFordFulkerson()$

"O algoritmo de Ford-Fulkerson, também conhecido como algoritmo dos pseudocaminhos aumentadores, resolve o problema do fluxo máximo. Cada iteração começa com um fluxo F que respeita as capacidades dos arcos."

Dentro do código ela se apresenta como a função das funções utilizada na Main e comandado as outras funções na parte mais interna do código.

2.2. $recursiveFF()$

Dentro do programa ele serviu como a base para toda a estrutura de procura, sendo o pilar do código. O código foi executado como uma versão reversa de Ford Fulkerson. E sendo utilizado para destacar possíveis vértices e arestas para serem eliminados para fazer com que o grafo ficasse desconexo.

2.3. $findPathBetween()$

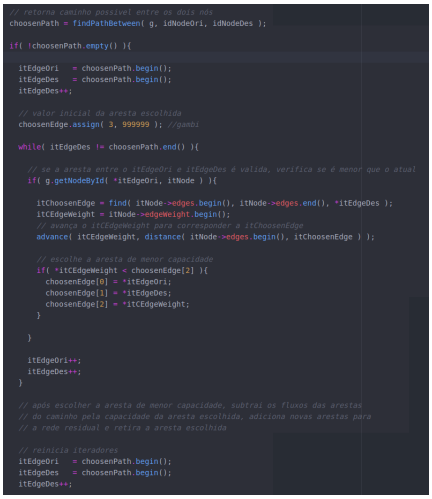
O algoritmo desenvolvido consiste em caminhar por todo o grafo, trazendo os caminhos ligando os dois pontos fixos do grafo.

2.4. $findPath()$

Funciona como a parte de execução do $findPathBetween()$, sendo a parte de recursão dentro do código. Testando possiveis remoções de integrantes do grafo.

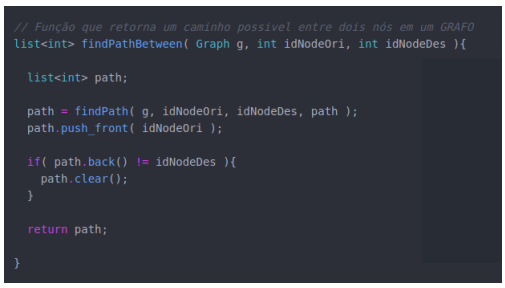
3. Imagens Do Algoritmo

Aqui estão alguns trechos de códigos, do programa que são citados na parte de Metodologias.



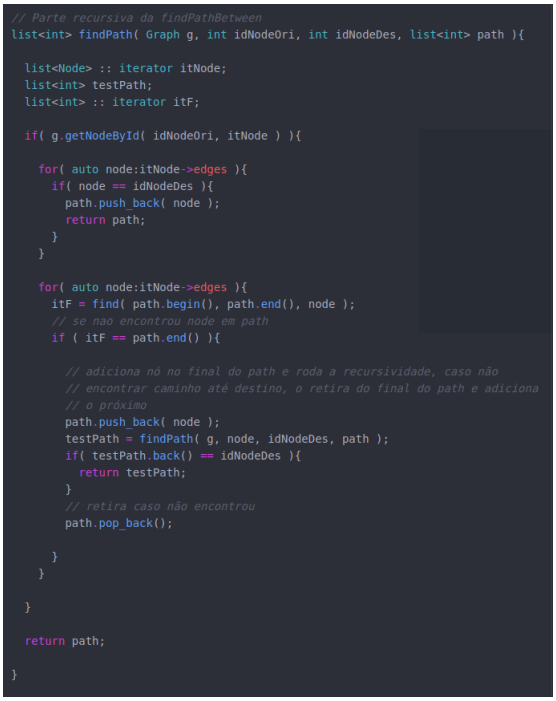
```
// retorna caminho possivel entre os dois nós
choosenPath = findPathBetween( g, idNodeOri, idNodeDes );

if( !choosenPath.empty() ){

  itEdgeOri   = choosenPath.begin();
  itEdgeDes   = choosenPath.begin();
  itEdgeDes++;

  // valor inicial da aresta escolhida
  choosenEdge.assign( 3, 999999 ); //gambi

  while( itEdgeDes != choosenPath.end() ){

    // se a aresta entre o itEdgeOri e itEdgeDes é valida, verifica se é menor que o atual
    if( g.getNodeById( *itEdgeOri, itNode ) ){

      itChoosenEdge = find( itNode->edges.begin(), itNode->edges.end(), *itEdgeDes );
      itCEdgeWeight = itNode->edgeWeight.begin();
      // avança o itCEdgeWeight para corresponder a itChoosenEdge
      advance( itCEdgeWeight, distance( itNode->edges.begin(), itChoosenEdge ) );

      // escolhe a aresta de menor capacidade
      if( *itCEdgeWeight < choosenEdge[2] ){
        choosenEdge[0] = *itEdgeOri;
        choosenEdge[1] = *itEdgeDes;
        choosenEdge[2] = *itCEdgeWeight;
      }

    }

    itEdgeOri++;
    itEdgeDes++;
  }

  // após escolher a aresta de menor capacidade, subtrai os fluxos das arestas
  // do caminho pela capacidade da aresta escolhida, adiciona novas arestas para
  // a rede residual e retira a aresta escolhida

  // reinicia iteradores
  itEdgeOri   = choosenPath.begin();
  itEdgeDes   = choosenPath.begin();
  itEdgeDes++;
```

Figura 1. Ford Furkerson. Fonte Autor



```
// Função que retorna um caminho possivel entre dois nós em um GRAFO
list<int> findPathBetween( Graph g, int idNodeOri, int idNodeDes ){

  list<int> path;

  path = findPath( g, idNodeOri, idNodeDes, path );
  path.push_front( idNodeOri );

  if( path.back() != idNodeDes ){
    path.clear();
  }

  return path;

}
```

Figura 2. Encontrar Caminho Entre. Fonte Autor



```
// Parte recursiva da findPathBetween
list<int> findPath( Graph g, int idNodeOri, int idNodeDes, list<int> path ){

  list<Node> :: iterator itNode;
  list<int> testPath;
  list<int> :: iterator itF;

  if( g.getNodeById( idNodeOri, itNode ) ){

    for( auto node:itNode->edges ){
      if( node == idNodeDes ){
        path.push_back( node );
        return path;
      }
    }

    for( auto node:itNode->edges ){
      itF = find( path.begin(), path.end(), node );
      // se nao encontrou node em path
      if ( itF == path.end() ){

        // adiciona nó no final do path e roda a recursividade, caso não
        // encontrar caminho até destino, o retira do final do path e adiciona
        // o próximo
        path.push_back( node );
        testPath = findPath( g, node, idNodeDes, path );
        if( testPath.back() == idNodeDes ){
          return testPath;
        }
        // retira caso não encontrou
        path.pop_back();

      }
    }

  }

  return path;

}
```

Figura 3. Recursão de caminho. Fonte Autor

4. Conclusão

Com o fim do desenvolvimento do algoritmo, se desenvolveu uma maneira de resolver o problema, para se desconectar um grafo que possui dois nós fixos com menor custo possível. Além de desenvolver uma melhor maneira de realizar o a remoção dos nós. O trabalho se decorreu de maneira linear, possuindo algumas travas e dificuldades, principalmente no desenvolvimento de um algoritmo e uma lógica e para se realizar a remoção dos nós, contudo foi desenvolvido de maneira manual um pseudo-código para se realizar todo o código do trabalho.

5. References

Visitado em 12 de junho de 2019.

www.ime.usp.br/ pf/algoritmos_para_grafos/aulas/flow-FF.html

Apostilas e Slides da aula de TEG do Professor Omir.