Trabalho 2 - Emparelhamento máximo

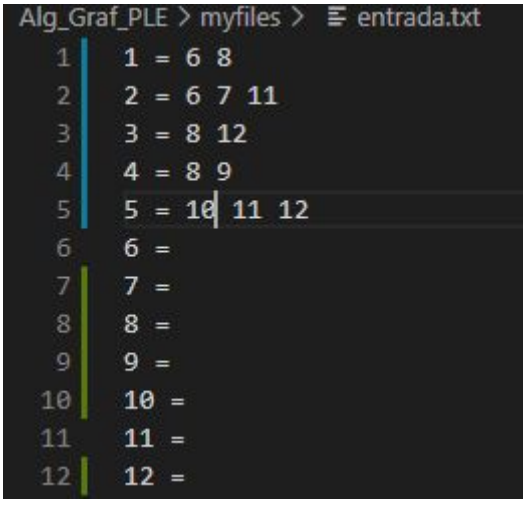
Instruções do programa

```
0 Sair
1 Print
2 Ler de arquivo
3 Escrever em arquivo
4 Adicionar vértice
5 Adicionar aresta
6 Excluir vértice
7 Entrada arquivo texto
8 Algoritmos
Escolha a opção:
```

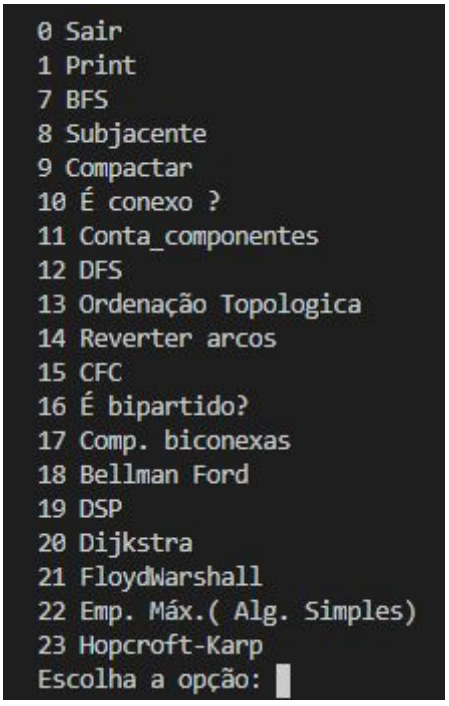
Após a execução, a partir da classe “AlgGrafos.java”, escolher a opção 7 (Entrtada arquivo texto) para ser feito a leitura do grafo a partir do arquivo entrada.txt . O caminho desse arquivo é indicado na classe “FileGraph.java”, na linha 11, como mostra a imagem 2 abaixo.

```
4   public class FileGraph {
5       public Digraph open_text( ) {
6           String thisLine = null;
7           Digraph dg1 = new Digraph( );
8           String pieces[ ];
9
10          try {
11              FileReader file_in = new FileReader("C:/Users/Yago/Desktop/Alg_Graf_PLE/myfiles/entrada.txt");
12              BufferedReader br1 = new BufferedReader( file_in );
13              while ( (thisLine = br1.readLine( )) != null) {
```

Exemplo de arquivo de entrada.txt:



Depois que a leitura do “entrada.txt” for finalizada, escolher a opção 8 (Algoritmos) e logo em seguida, escolher entre a opção 22 (Emp. Max.) ou a 23 (Hopcroft-Karp), para encontrar o emparelhamento máximo em um grafo não-direcionado e bipartido.



A opção 22 de algoritmo, utiliza um algoritmo mais simples para determinar o emparelhamento máximo, a partir do conceito de caminhos de aumento em grafo. Ele utiliza um DFS para encontrar o emparelhamento e aumentá-lo um por vez. E está na ordem de $O(V^2 + VE)$.

Já opção 23, o algoritmo de Hopcroft-Karp, é mais eficiente do que a opção 22 e está na ordem de O($\sqrt{}$ v (v+e)). Esse algoritmo utiliza primeiramente, um BFS adaptado, para encontrar um conjunto maximal de caminhos de aumentos, de comprimento mínimo. E em seguida realiza um DFS adaptado, para encontrar caminhos disjuntos entre o conjunto maximal, e utiliza esses caminhos disjuntos para aumentar o emparelhamento.

A implementação dos métodos utilizados para o trabalho 2, estão contidos entre as linhas 113 a 250 no código da classe “Graph.java”.

O algoritmo simples utiliza os métodos “*acha_caminho_de_amento()*” e “emp_max ()”, conforme as imagens abaixo.

```
113     ////////////////////  TRABALHO 2 - EMPARELHAMENTO MAXIMO ////////////////////////////////////////////////
114
115     boolean acha_caminho_aumento( Vertex u ) {
116
117         for ( Vertex v : u.nbhood.values() ) {
118
119             if( v.seen) continue;
120
121             v.seen = true;
122
123             if ( v.emparelhado == null || acha_caminho_aumento( v.emparelhado) ) {
124                 u.emparelhado = v;
125                 v.emparelhado = u;
126                 return true;
127             }
128         }
129         return false;
130     }
131
```

```
132     public void emp_max () { // O(V^2 + VE)
133
134         for ( Vertex v : vertex_set.values()) { // Após passar pelo is_bipartite() , insere vertices nas partições distintas A e B.
135             if ( v.ind_set == 1 ) {
136                 this.part_A.add(v);
137             } else {
138                 part_B.add(v);
139             }
140         }
141
142         int max_matching = 0;
143
144         for ( Vertex v : part_A ) {
145             if ( v.emparelhado == null) { // Se true, então v é um vertice livre.
146                 if ( acha_caminho_aumento(v))
147                     max_matching++;
148             }
149         }
150
151         System.out.printf("\n O tamanho do emparelhamento maximo = %d\n", max_matching);
152         for (Vertex v : part_A) {
153             if (v.emparelhado != null) System.out.printf("\n %d está emparelhado com %d\n", v.id, v.emparelhado.id);
154         }
155     }
```

Já o algoritmo de Hopcroft-Karp, utiliza os métodos “*acha_caminhos_de_aumento()”*, *"aumenta_emparelhamento( )”* e *“hopcroft_karp( )”.*

```java
public Integer acha_caminhos_de_aumento() { // BFS adaptado

    Queue<Vertex> lista = new LinkedList<Vertex>();
    Integer k;

    k = 99999999; // Valor de k inicial representa infinito;

    for ( Vertex u : this.part_A ) {
        if ( u.emparelhado == null ) { // Se vertice u for livre, adiciono ele na lista
            u.d = 0;
            lista.add(u);
        } else {
            u.d = 99999999;
        }
    }

    Vertex atual;
    while ( !lista.isEmpty() ) {
        atual = lista.poll();
        if ( atual.d < k) {
            for ( Vertex v : atual.nbhood.values()){
                if ( v.emparelhado == null && k == 99999999) {
                    k = atual.d + 1;
                } else if ( v.emparelhado != null){
                    if ( v.emparelhado.d == 99999999 ) { //Se o vertice adj à v não estiver descoberto, insiro ele na fila.
                        v.emparelhado.d = atual.d + 1;
                        lista.add(v.emparelhado);
                    }
                }
            }
        }
    }
    /*if ( k != 999999999 ){
        return k;
    }*/
    return k;
}
```

```java
public boolean aumenta_emparelhamento ( Vertex v)  { // DFS adaptado
    for ( Vertex u : v.nbhood.values() ) {

        if ( u.emparelhado == null) { // condição de parada
            v.emparelhado = u;
            u.emparelhado = v;
            return true;
        }

        if ( v.emparelhado != null) {
            if (u.emparelhado.d == (v.emparelhado.d + 1)) {
                if ( aumenta_emparelhamento(u.emparelhado) ) {
                    v.emparelhado = u;
                    u.emparelhado = v;
                    return true;
                }
            }
        }

    }

    // Se chegar aqui, significa que a partir de v, não consigo achar um caminho de aumento disjunto.
    v.d = 99999999;
    return false;
}
```

```java
public void hopcroft_karp( ) {   // O(RAIZ(V)(V+E))

    for ( Vertex v : vertex_set.values()) { // Após passar pelo is_bipartite() , insere vertices nas partições distintas A e B.
        if ( v.ind_set == 1 ) {
            this.part_A.add(v);
        } else {
            part_B.add(v);
        }
    }

    int max_matching = 0;
    while (acha_caminhos_de_aumento() != 99999999 ) {
        for (Vertex v : this.part_A)
            if ( v.emparelhado == null && aumenta_emparelhamento(v))
                max_matching++;
    }

    System.out.printf("\n Tamanho do emparelhamento maximo: %d\n", max_matching);

    for (Vertex v : this.part_A) {
        if (v.emparelhado != null){
         System.out.printf("\n %d está emparelhado com %d\n", v.id, v.emparelhado.id);
        }
    }
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

Exemplo de saída do algoritmo de Hopcroft-Karp com o entrada.txt .

```
Escolha a opção: 23

É bipartido

Tamanho do emparelhamento maximo: 5

1 está emparelhado com 6

2 está emparelhado com 7

3 está emparelhado com 8

4 está emparelhado com 9

5 está emparelhado com 10
```

Referência:

Delfino, Giovana G. **EMPARELHAMENTOS EM GRAFOS: ALGORITMOS E IMPLEMENTAÇÕES**. 2017. Monografia - Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, 2017. Disponível em: <https://github.com/gidelfino/MAC0499/raw/master/monografia.pdf> Acessado em: 28 de outubro 2020