TRABALHO PRÁTICO 3

Ana Lúcia Canto e Maria Gabriella Basílio

DECISÕES DE PROJETO

- Implementado em Java
- Utilizando listas de adjacência

REPRESENTAÇÃO

- Lista de adjacência
- Cada vértice (vértice, peso, capacidade)

```
(1,0.0,0.0)-> (2,0.0,20.0)-> (3,0.0,10.0)->
(2,0.0,0.0)-> (3,0.0,30.0)-> (4,0.0,10.0)->
(3,0.0,0.0)-> (4,0.0,20.0)->
(4,0.0,0.0)->
```

IMPLEMENTAÇÃO

listaAdjacencia.java

```
public boolean bfs() {
  while (proximoNo != null) {
      if (isDescoberto[proximoNo.getVertice() - 1] &&
proximoNo.getCapacity() > delta) {
  ...
 }
 return isDescoberto[t-1];
}
```

Otimização

```
public void fordFulkerson() {

   while(delta < getCapacidadeSaida(grafo, s)) {
     delta = delta*2;
   }
    delta = delta/2;

    while(delta > 0) {
       ...
    delta = delta/2;
}
```

```
public void getCapacidadeSaida(grafo, int s) {

  No noInicio =  grafo.getListaAdjacencia()[s -1];
  float capacidade = 0;
  while(noInicio.getProximoNo() != null){
    capacidade +=
 noInicio.getProximoNo().getCapacity();
    noInicio = noInicio.getProximoNo();
   }
  return capacidade;
  }
}
```

Determinando menor fluxo

fordFulkerson.java

```
public void fordFulkerson() {
    while(bfs(grafo, pai , isDescoberto,  s,  t,
delta)) {

       float fluxo_caminho = Integer.MAX_VALUE;

       for (int v = t; v!=s; v = pai[v-1]) {
        int u = pai[v-1];
        if (getElementoListaADjacencia(grafo, u, v)
 != null){
         fluxo_caminho = Float.min(fluxo_caminho,
getElementoListaADjacencia(grafo, u,
v).getCapacity());
        }
     }
       ....
}
```

```
 private No getElementoListaADjacencia(Grafo grafo, int
u, int v) {

  No noInicio =  grafo.getListaAdjacencia()[u -1];
  No no = null;
  while(noInicio.getProximoNo() != null){
   if (noInicio.getProximoNo().getVertice() == v ){
    no = noInicio.getProximoNo();
    return no;
   }
   noInicio = noInicio.getProximoNo();
  }
  return no;
 }
```

}

Atualizando arestas da lista de adjacência

fordFulkerson.ja

```
public void fordFulkerson() {
    while(bfs(grafo, pai , isDescoberto,  s,  t,
delta)) {
                ...
    for (int v = t; v!=s; v = pai[v-1]) {
        int u = pai[v-1];
        updateArestas(grafo, u, v, fluxo_caminho);
        updateArestas(grafo, v, v, -fluxo_caminho);
       }

       fluxo_maximo += fluxo_caminho;
}

     return fluxo_maximo;
```

```
 private void updateArestas( grafo, u, v, fluxo_caminho) {
  No noInicio =  grafo.getListaAdjacencia()[u -1];
  No proximoNo = noInicio.getProximoNo();
  boolean foundIt = false;

  while(proximoNo != null){
   if (proximoNo.getVertice() == v ){
    proximoNo.setCapacity(proximoNo.getCapacity() -
fluxo_caminho);
    foundIt = true;

    if (proximoNo.getCapacity() <= 0) {
     noInicio.setProximoNo(proximoNo.getProximoNo());
     proximoNo = noInicio;
    }
   }
   noInicio = proximoNo; proximoNo = proximoNo.getProximoNo();
  }
  if (!foundIt) {
   ListaAdjacencia lista = new
ListaAdjacencia();
   lista.addAresta(u, v, 0, -
fluxo_caminho,grafo);
  }
}
```

TEMPO DE EXECUÇÃO

	Tempo (s)		Tempo (s)
Grafo 1	0,02	**Grafo 5**	1,01
Grafo 2	0,02	**Grafo 6**	5,03
Grafo 3	0,08	**Grafo 7**	8,25
Grafo 4	0,38	**Grafo 8**	48,21

FLUXO MÁXIMO

	Fluxo Máximo
Grafo 1	284
Grafo 2	276820
Grafo 3	291
Grafo 4	253278

	Fluxo Máximo
Grafo 5	618
Grafo 6	548517
Grafo 7	611
Grafo 8	5382665

OBRIGADA PELA ATENÇÃO