Pergunta 1

Simulação do Algoritmo de Edmonds-Karp

Sequência de Caminhos:

- 1 2 3 4 — incremento 3;
-
-

Valor do Fluxo Máximo:

Fluxo Final de Cada Arco de G:

Pergunta 2

Coloração dos Vértices

Percurso em Largura

```
void bfsTraversal( Graph graph )
{
    boolean[] found = new boolean[ graph.numVertices() ];

    for every Vertex v in graph.vertices()
        found[v] = false;

    for every Vertex v in graph.vertices()
        if ( !found[v] )
            bfsExplore(graph, found, v);
}
```

```
Pair<Integer,Integer> bfsTraversal( UndiGraph graph )
{
    boolean[] found = new boolean[ graph.numVertices() ];

    for every Vertex v in graph.vertices()
        found[v] = false;

    int numColours = 0;
    int maxSameColour = 0;

    for every Vertex v in graph.vertices()

        if ( !found[v] )
        {
            numColours++;
            sameColour =  bfsExplore(graph, found, v);
            if ( sameColour > maxSameColour )
                maxSameColour = sameColour;
        }
    return new PairClass<Int,Int>(numColours, maxSameColour);
}
```

Árvore em Largura (iterativo)

```
void bfsExplore( Graph graph,  boolean[] found,  Vertex root )
{   Queue<Vertex> waiting =
        new QueueInArray<Vertex>( graph.numVertices() − 1 );
    waiting.enqueue(root);
    found[root] = true;
    do {
            Vertex vertex = waiting.dequeue();    TREAT(vertex);
            for every Vertex w in graph.outAdjacentVertices(vertex)
                if ( !found[w] )
                {   waiting.enqueue(w);
                    found[w] = true;
                }
        }
    while ( !waiting.isEmpty() );
}
```

```
int bfsExplore( UndiGraph graph,   boolean[] found,   Vertex root )
{   Queue<Vertex> waiting =
        new QueueInArray<Vertex>( graph.numVertices() − 1 );
    waiting.enqueue(root);
    found[root] = true;    int numVertices = 0;
    do {
            Vertex vertex = waiting.dequeue();
            /* TREAT(vertex); */    numVertices++;
            for every Vertex w in graph.adjacentVertices(vertex)
                if ( !found[w] )
                {   waiting.enqueue(w);
                    found[w] = true;
                }
        }
    while ( !waiting.isEmpty() );
    return numVertices;
}
```

Complexidade dos Percursos

Implementação do Grafo (V, A)	**Profundidade** (recursivo) ou **Profundidade** (iterativo) ou **Largura** (iterativo)
Matriz	$\Theta(\|V\|^2 + \|V\| \times \text{custo(TREAT)})$
Vector de Listas	$\Theta(\|V\| + \|A\| + \|V\| \times \text{custo(TREAT)})$

Pergunta 6

Fila Binomial

```
// Returns an array with the entries stored in
// the deepest nodes of the binomial queue,
// if the binomial queue is not empty;
// otherwise, returns null.
protected Entry<K,V>[] deepestEntries( )
{
    if ( head == null )
        return null;

    BinNode<K,V> node = head.getLeftSibling();

    while ( node.getChild() != null )
        node = node.getChild().getLeftSibling();

    Entry<K,V>[] result = (Entry<K,V>[]) new Entry[1];
    result[0] = node.getEntry();
    return result;
}
```

Pergunta 3

NP-Completo

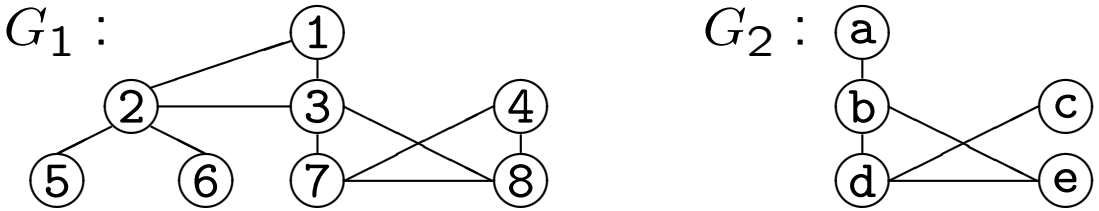
Subgrafo Isomorfo

Sejam $G = (V, A)$ e $G^\star = (V^\star, A^\star)$ dois grafos não orientados. Diz-se que G **tem um subgrafo isomorfo a** $G^\star$ se, e só se, existir uma função injectiva $f : V^\star \longrightarrow V$ tal que:

$$(\forall x, y \in V^\star)\ (x, y) \in A^\star \implies (f(x), f(y)) \in A.$$

SUBGRAFO-ISOMORFO: Dados dois grafos não orientados, G e $G^\star$, G tem um subgrafo isomorfo a $G^\star$?

Instância



O Problema do Subgrafo Isomorfo é **NP**

pelos dois motivos seguintes.

1. Existe um **algoritmo polinomial** que, dados:
   - dois grafos não orientados, $G = (V, A)$ e $G^\star = (V^\star, A^\star)$, e
   - uma função injectiva $f : V^\star \longrightarrow V$,

   verifica se $(\forall x, y \in V^\star)\ (x, y) \in A^\star \implies (f(x), f(y)) \in A$.

   Descrição do algoritmo:

   Cálculo da complexidade do algoritmo (no pior caso):

2. A **dimensão** da função f é **polinomial** na dimensão de $(G, G^\star)$, porque ...

O Problema do Subgrafo Isomorfo é **NP-difícil**

porque o problema da Clique é NP-completo e a seguinte função é uma redução polinomial.

$$\textbf{CLIQUE} \longrightarrow \textbf{SUBGRAFO-ISOMORFO}$$

$$((V, A),\ k) \longmapsto ((V, A),\ (V^\star, A^\star))$$

com:

$$V^\star = \{1, 2, 3, \ldots, k\} \quad \text{e}$$

$$A^\star = \{(v, w) \in V^\star \times V^\star \mid v \neq w\}.$$

O Problema do Subgrafo Isomorfo é

NP-completo porque é NP e NP-difícil.