Relatório 2º projeto ASA 2023/2024

Grupo: AL016

Alunos: Lara Faria (106059) e Rita Melo (107294)

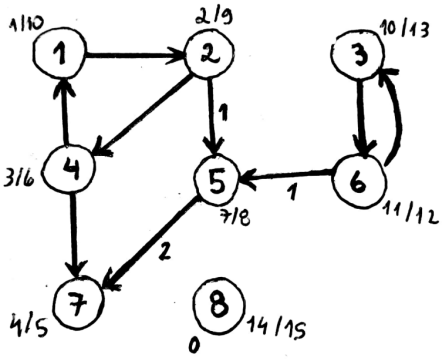
Descrição do Problema e da Solução

Dado uma rede social TugaNet, que corresponde a um grafo G = (V, E), pretende-se calcular o maior número de saltos que uma dada doença pode fazer, sendo que os indivíduos que se conhecem mutuamente ficam infetados instantaneamente. O *input* corresponde ao número de vértices V (número de indivíduos da rede social), ao número de arcos E (número de conexões da rede) e às E linhas em que cada linha representa um arco (u, v), que significa que o indivíduo u conhece o indivíduo v.

A solução do problema foi implementada em C++ e consiste numa junção do algoritmo *Kosaraju* (para encontrar as componentes fortemente ligadas) e do algoritmo *longest path in a DAG* (que utiliza programação dinâmica). Primeiro realizamos uma DFS sobre o grafo e obtemos um vetor que contém os vértices pela ordem topológica inversa. De seguida realizamos uma DFS sobre o grafo transposto respeitando a ordem topológica inversa para descobrir as componentes fortemente ligadas. Por fim, percorremos o grafo para calcular o número máximo de saltos utilizando um vetor dp para armazenar o número de saltos em cada vértice (subproblema).

Pseudocódigo da nossa DFS iterativa:

```
DFS():
  numV = V - 1;
  for each v in G.V:
  if v.color == white:
    stack.push(v);
    while stack not empty:
        let currentv be the top of the stack;
        currentv.pop;
        if currentv.color == white:
            currentv.color = grey;
            stack.push(currentv);
            for each adj int G.adj[currentv]:
                if adj.color == white:
                    adj.color = grey;
                    stack.push(adj);
        else if currentv.color == grey:
            currentv.color = black;
            orderedV[numV] = currentv;
            numV--;
```



Relatório 2º projeto ASA 2023/2024

Grupo: AL016

Alunos: Lara Faria (106059) e Rita Melo (107294)

Análise Teórica

- Leitura de input: leitura do número de vértices e arestas e das E arestas (E arestas). Complexidade: O(E).
- DFS: primeiro *for loop* para iterar pelos vértices O(V), *loop while* O(V), segundo *for loop* para visitar os adjacentes O(E). Complexidade O(V+E).
- DFS no transposto: primeiro *for loop* para iterar pelos vértices O(V), *loop while* O(V), segundo *for loop* para visitar a lista de adjacências do transposto O(E). Complexidade O(V+E).
- Calcular os saltos: primeiro *for loop* para iterar pelos vértices O(V), *loop while* O(V), segundo *for loop* para visitar os adjacentes O(E). Complexidade O(V+E).

Complexidade geral: **O(V+E)**

Análise Experimental dos Resultados

Corremos o programa com diferentes ficheiros de *input* gerados pelo gerador de instâncias e utilizámos o comando *time* do terminal para obtermos o tempo de cada teste. Desta forma, obtivemos os resultados apresentados na tabela, expondo-os na forma do gráfico apresentado abaixo. Este gráfico representa a forma como o tempo vai aumentando linearmente com o aumento do resultado da soma entre o número de vértices e o número de arestas de um grafo, que corresponde também ao valor da complexidade do nosso código, verificando-se, assim, a nossa estimativa.

V+E	SubN	Tempo(s)
2000	200	**0,004**
5000	900	**0,005**
10000	1000	**0,005**
20000	1000	**0,009**
100000	10000	**0,032**
160000	25000	**0,051**
200000	15000	**0,068**
600000	90000	**0,238**
1000000	130000	**0,407**
2000000	300000	**0,917**
10000000	1000000	**5,747**