Relatório 2º projeto ASA 2023/2024

Grupo: AL002
Alunas: Cecília Correia (106827) e Luísa Fernandes (102460)

1. Descrição do Problema e da Solução

O **problema** está relacionado com a análise da propagação de uma doença na rede TugaNet, considerando que a rede é um grafo dirigido. O objetivo é determinar o número máximo de saltos que a doença pode fazer, considerando que a transmissão ocorre instantaneamente entre pessoas que se conhecem, direta ou indiretamente.
A **solução** proposta consiste numa abordagem a grafos dirigidos e Componentes Fortemente Conectadas (SCCs) para encontrar a ordem topológica e calcular o número máximo de saltos que uma doença pode fazer na rede TugaNet. As três principais etapas da solução são: *1) Ordenação Topológica*. Utiliza uma busca em profundidade (DFS) para obter a ordem topológica do grafo, identificando as conexões direcionadas entre indivíduos. Por exemplo: Na aresta (u, v), o nó u aparece antes de v; *2) Encontrar SCCs.* Aplica uma segunda DFS para identificar as SCCs no grafo transposto, agrupando os indivíduos que têm interações mútuas; e *3) Construção do Grafo Acíclico Direcionado (DAG) e Cálculo de Saltos.* Transforma o grafo original num DAG, considerando as SCCs como nós individuais na nova estrutura. Calcula o número máximo de saltos e imprime.

2. Análise Teórica

Pseudocódigo que ilustra a DFS1 iterativa da solução proposta (a DFS2 funciona de forma semelhante):

```
DFS1
  create stack order
  create vector boolean visited(n + 1, false)
  for i = 1 to i <= n
    if i is not visited
      create stack s
      push node i to s
      hasUnvisitedNeighbor = true
      while (s is not empty)
        current = top of stack s
        hasUnvisitedNeighbor = false
        for each neighbor in graph[current]:
          if neighbor is not visited
            push neighbor to stack s
            visited[Neighbor] = true
            hasUnvisitedNeighbor = true
            break
          endif
        endfor
        if not hasUnvisitedNeighbor
          pop stack s
          add current to order
        endif
    endif
  endfor
end
```

Pequeno exemplo da utilização da DFS1:

1 --> 2 --> 3
| |
v v
4 <--------- 5

Neste exemplo, a ordem topológica resultante é 1, 2, 3, 5, 4. Indica-nos a ordem em que os nós seriam visitados num percurso DFS no grafo original.

3. Análise de Complexidade

- Leitura dos dados de entrada: envolve um loop sobre as relações entre indivíduos (arestas) é executado m vezes. Tem complexidade linear em relação a m. Logo: Θ(m)
- Ordenação topológica: O primeiro algoritmo de DFS gera uma ordem topológica dos vértices do grafo dirigido (graph). A ordem topológica é armazenada em uma pilha (stack) chamada order. A ideia principal é iniciar a busca em profundidade a partir de cada nó não visitado, e quando não há mais vizinhos não visitados, o nó é empilhado. Iteração sobre todos os vértices do grafo, cada vértice é visitado no máximo uma vez. Para cada vértice (n), há uma DFS que percorre todas as arestas adjacentes (m). Logo: O(n + m)
- Encontrar SCCs: A segunda DFS usa a ordem topológica (armazenada em order) para realizar outra busca, desta vez no grafo reverso (reverseGraph), a fim de encontrar as SCCs. Cada SCC é armazenada em sccList. Semelhante à dfs1. Logo: O(n + m)
- Transformar o grafo em uma DAG: envolve iteração sobre todas as arestas do grafo original e a criação de um novo dag. Logo: O(m)
- Calcular o número máximo de saltos: o loop externo percorre todos os vértices do dag (n) e o loop interno percorre as arestas que saem do vértice atual i, o número de iterações é igual ao número de arestas do dag (m). Logo: O(n + m)
- Apresentação dos dados: a impressão de uma variável é de complexidade constante. Logo: Θ(1)
- Complexidade global da solução: O(n + m)

4. Avaliação Experimental dos Resultados

Foram geradas 21 instâncias de tamanho incrementado. Foi corrido o código usando time para saber o tempo de execução, registado na tabela em baixo (coluna da direita). O gráfico gerado do tempo de execução no eixo YY, em função de f(n, m) = n + m no eixo dos XX, que corresponde à complexidade do programa prevista teoricamente, representa uma relação linear (Figura 1).

n	m	n+m	tempo
1000	1000	2000	0
5000	5000	10000	0,01
10000	10000	20000	0,02
20000	20000	40000	0,04
50000	50000	100000	0,11
80000	80000	160000	0,17
90000	90000	180000	0,2
100000	100000	200000	0,21
200000	200000	400000	0,48
250000	250000	500000	0,59
300000	300000	600000	0,72
400000	400000	800000	0,97
500000	500000	1000000	1,24
600000	600000	1200000	1,53
700000	700000	1400000	1,85
750000	750000	1500000	1,98
800000	800000	1600000	2,1
850000	850000	1700000	2,28
900000	900000	1800000	2,38
950000	950000	1900000	2,49
1000000	1000000	2000000	2,68

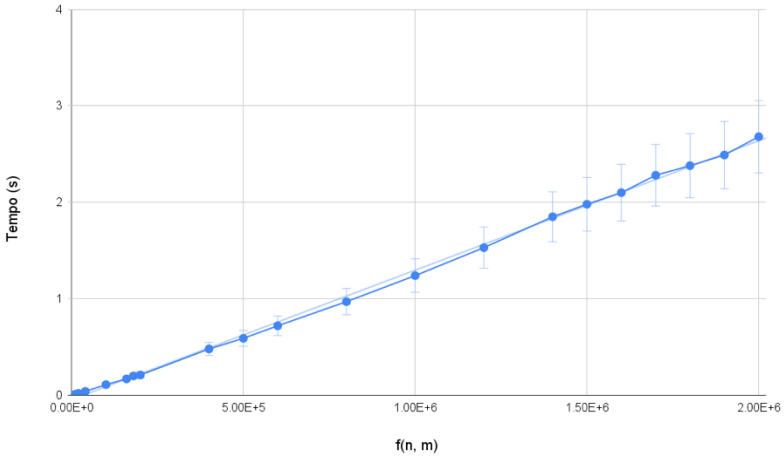

Figura 1. Tempo (s) em função da complexidade estimada f(n, m).

Conclui-se que há uma relação linear entre o tempo e o f(n, m), confirmando que a implementação está de acordo com a análise teórica.