Relatório 2º projeto ASA 2023/2024

Grupo: AL052
Aluno(s): André Bento (106930) e Pedro Loureiro (107059)

Descrição do Problema e da Solução

Este projeto está relacionado com o estudo de doenças transmissíveis em Portugal. O objetivo é compreender o pior cenário de propagação de uma infeção, determinando o maior número de "saltos" que uma doença pode realizar. Todavia, devido à densidade populacional, é considerado que indivíduos que se conhecem mutuamente de maneira direta ou indireta ficam infetados instantaneamente.

A nossa solução teve como base o algoritmo Kosaraju, que recorre à busca em profundidade (DFS) e à ordenação topológica inversa para identificar os SCCs. A partir disto, é possível diferenciar as situações em que os indivíduos se conhecem mutuamente, isto é, caso pertençam ao mesmo SCC o número de 'saltos' não é alterado. Desta forma, durante a segunda DFS os saltos foram sendo registados, de forma iterativa, guardando apenas o maior valor entre iterações, obtendo assim o resultado pretendido.

Análise Teórica

Pseudo-Código:

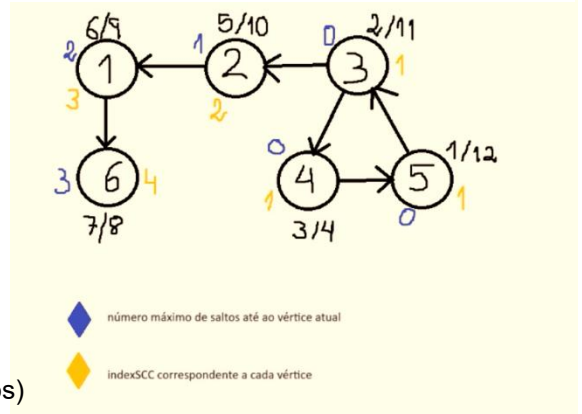
Após a ordenação topológica inversa ter sido calculada, segue-se a 2ªdfs (a que identifica os SCCs):

```
int dfsSCC(int vertice, vector<vector> adjacencias, vector arraySCC, vector visitado, vector resultados){
saltos = 0
let pilha be a stack
pilha.push(vertice)
visitado[vertice] = true

while (!pilha.empty())
temVizinhosNaoVisitados = false
atual = pilha.top()
processNeighbours(adjacencias, atual, visitado, pilha,
temVizinhosNaoVisitados)

if (!temVizinhosNaoVisitados) then continue
    pilha.pop()
    arraySCC[atual] = indexSCC
    arraySCC[vertice] = indexSCC
    for vizinho in adjacencias[vertice] do
          processVertices(arraySCC, vizinho, vertice, resultados, saltos)
    endfor
    if (vértice != atual) then continue
       for vizinho in adjacencias[atual] do
          processVertices(arraySCC, vizinho, atual, resultados, saltos)
        endfor
   endif
endif
}
```

Exemplo do funcionamento da dfsSCC (2ªdfs)



Relatório 2º projeto ASA 2023/2024

Grupo: AL052
Aluno(s): André Bento (106930) e Pedro Loureiro (107059)

```
processVertices(vector arraySCC, int vizinho, int vertice, vector resultados, int saltos){
if (arraySCC[vizinho] == arraySCC[vertice]) then continue
   resultados[vertice] = max(resultados[vizinho], resultados[vertice])
endif
else if (arraySCC[vertice] != 0 and arraySCC[vizinho] != 0) then continue
   resultados[vertice] = max(resultados[vizinho] + 1, resultados[vertice])
endif
saltos = max(saltos, resultados[vertice])
}
processNeighbours(vector<vector>adjacencias, int atual, vector visitado, stack pilha,
temVizinhosNaoVisitados){
for vizinho in adjacencias[atual]) do
        if (!visitado[vizinho]) then continue
           visitado[vizinho] = true
           pilha.push(vizinho)
           temVizinhosNaoVisitados = true
           break
         endif
endfor
}
```

Notas:

1. 'indexSCC' – scc a que cada vértice pertence (variável global);
2. 'calculaMaxSaltos' – função que coordena a chamada das outras funções;

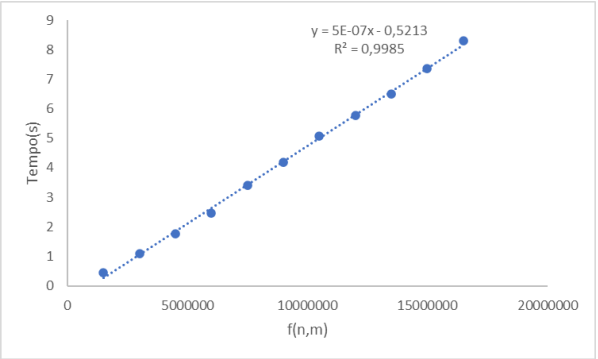
Complexidades:

1. Leitura dos dados de entrada: O(m)
2. Ordenação topológica inversa: O(n + m)
3. Aplicação do algoritmo indicado para cálculo do valor pedido: O(n + m)
4. Apresentação dos dados: O(1)

Complexidade global da solução: O(m) + O(n + m) + O(n + m) + O(1) ≈ O(n + m) = O(V + E)

Avaliação experimental dos resultados

De seguida, encontra-se um gráfico que averigua a relação entre o tempo e a função de complexidade global da solução.



Ao analisarmos o mesmo, chegamos à conclusão de que, colocando o eixo dos XX a variar de acordo com o que declaramos ser a complexidade global da solução na nossa análise teórica (V + E) obtemos os resultados esperados, isto é, uma relação linear entre o tempo e a complexidade global calculada.