**Relatório 2º projeto ASA 2023/2024**

**Grupo:** AL058

**Aluno(s):** David Quintino (107095) e Carlota Domingos (107016)

**Descrição do Problema e da Solução**

O objetivo deste projeto é calcular o maior salto possível de um grafo direcionado. Para tal utiliza um algoritmo de procura em profundidade (DFS iterativa) duas vezes: primeiramente aplica-o no grafo original para ordenar os vértices por tempo de fecho e depois no grafo transposto calcular o maior salto possível a partir das componentes fortemente ligadas (SCC’s).

**Análise Teórica**

**Segunda DFS (pseudo-código)**

Function second\_dfs(order, queue):

Initialize parent, value, max\_jump, and node variables

While order is not empty:

Set node to the top of the order stack and remove it

If parents[node] is 0 (node not visited yet):

Push node to the queue

Set parent to node

Set value to 0

While queue is not empty (while in the strongly connected component):

Set node to the top of the queue and remove it

Set parents[node] to parent

For each neighbour of node in the transposed graph:

If parents[neighbour] is not equal to parent:

If parents[neighbour] is not visited:

If max\_jump\_node[parents[neighbour]] is greater or equal to value:

Set value to max\_jump\_node[parents[neighbour]] + 1

Else:

Push neighbour to the queue

Set max\_jump\_node[parent] to value

If max\_jump\_node[parent] is greater than max\_jump:

Set max\_jump to max\_jump\_node[parent]

Return max\_jump

**Complexidade (Sendo V, E o número de vértices e arcos no grafo.):**

**Tratar do input:**

* Loop “while” que em cada iteração faz dois “push\_back”s de complexidade O(1) e realiza um total de E iterações. Complexidade O(E)

**DFS iterativa:**

* 2 partes, loop principal e o dfs visit. O principal usa um for para visitar todos os vértices não visitados e em cada um fazer um dfs visit. Para não ser recursiva, usamos uma pilha que guarda os vértices a visitar e usamos um “while queue not empty” para cada iteração do dfs visit .
* Uma vez que marcamos o estado dos vértices (White, Gray, Black), nenhum destes é percorrido mais que uma vez e, se para cada vértice vemos todas as suas adjacências 1 vez, então todos os arcos do grafo são percorridos 1 vez. Concluindo, a complexidade da DFS iterativa é O(E + V).

A **segunda DFS** tem a mesma complexidade que a primeira (O(E +V)), uma vez que percorre o grafo segundo as mesmas regras, ou seja, cada vértice e cada arco são visitados uma única vez. A única diferença em relação ao outro algoritmo é que em vez de cada vértice ter um estado, tem o maior número de saltos possível a partir dele e um identificador da SCC em que se encontra.

Sendo assim, a **complexidade total** da solução pode ser dada por O(E) + 2O(V + E) = O(V + E);

**Avaliação experimental dos resultados**

O seguinte gráfico representa e relação temporal da execução do programa em função do número de vértices e de arcos. No eixo dos xx está a soma destes (V + E) e o no eixo dos yy o tempo (ms) que o programa demorou a calcular o valor máximo (Fig.1).

|  |  |  |  |
| --- | --- | --- | --- |
| V | E | V + E | Tempo(ms) |
| 120 | 80 | 200 | 6 |
| 600 | 400 | 1000 | 6 |
| 1200 | 800 | 2000 | 7 |
| 6000 | 4000 | 10000 | 9 |
| 12000 | 8000 | 20000 | 16 |
| 60000 | 40000 | 10000 | 65 |
| 120000 | 80000 | 20000 | 117 |
| 600000 | 400000 | 10000 | 641 |
| 1200000 | 800000 | 20000 | 1406 |
| 2400000 | 16000000 | 40000 | 2881 |
| 3600000 | 24000000 | 60000 | 4283 |
| 4800000 | 32000000 | 80000 | 5691 |
| 6000000 | 40000000 | 100000 | 7613 |

A graph with a line

Description automatically generated

Fig1: Gráfico que relaciona a area da placa com o tempo de execução

Fig2: Tabela com os pontos do gráfico

Como se pode ver a função aparenta ser linear, tal como esperávamos. Sendo assim, a nossa análise sobre a complexidade deste algoritmo estava certa.