Relatório 1º projecto ASA 2020/2021

Grupo: al102

Alunos: Francisco Ribeiro (95578) e Diogo Lopes (96732)

Descrição do Problema e da Solução

Foi-nos proposto que calculássemos o número mínimo de intervenções num grafo dirigido acíclico (DAG) de forma a que fosse possível alcançar todos os vértices. Interpretámos este problema da seguinte forma: identificar as *sources*, isto é, vértices que não têm arcos a apontar para si, que funciona uma vez que num DAG existe pelo menos uma *source*.

A segunda parte do problema consistia em identificar a maior distância possível de percorrer dentro do grafo. Para solucionar esta questão, recorremos ao algoritmo de Kahn para ordenação topológica. Conseguimos a partir do algoritmo determinar a distância pedida, dado que ao "cortar" todas as *incoming edges* de um vértice para cada vértice ordenado topologicamente, garantimos que percorremos o maior caminho de entre um dado vértice (*source*) e outro qualquer alcançável a partir dele.

Fontes: https://www.geeksforgeeks.org/topological-sorting-indegree-based-solution/
https://www.mathcs.emory.edu/~cheung/Courses/171/Syllabus/11-Graph/Docs/longest-path-in-dag.pdf

Análise Teórica

```
processInput()

 Let V be the number of inputted vertexes for G
 Let E be the number of inputted edges for G
 Let sources be the list of sources in G
 assert(V >= 2 && E >= 0)

 for each i in {1,..,V}              (1)
  i.dist = 0; i.incoming_edges = 0
  G.V[i] = i
  sources.add(i)

 for each i in {1,..,E}              (2)
  Let a be the origin of the edge
  Let b be the end of the edge
  G.Adj[a].add(b)
  b.incoming_edges++
  sources.remove(b)
 return SUCCESS
```

```
topSortMaxDistance(G)

 Let P be the greatest distance in G
 Let count be the number of seen
    vertexes
 Let Q be a queue

 for each s in sources               (3)
  Q.add(s)

 while Q is not empty                (4)
  Let u be the front vertex of Q
  Q.remove(u)
  for w in G.Adj[u]                  (5)
   w.incoming_edges--
   if w.incoming_edges == 0
    Q.add(w)
    w.dist = u.dist + 1
    if w.dist > P
     P = w.dist
   count++
 assert(count == G.V)
 return P+1
```

Relatório 1º projecto ASA 2020/2021

Grupo: al102

Alunos: Francisco Ribeiro (95578) e Diogo Lopes (96732)

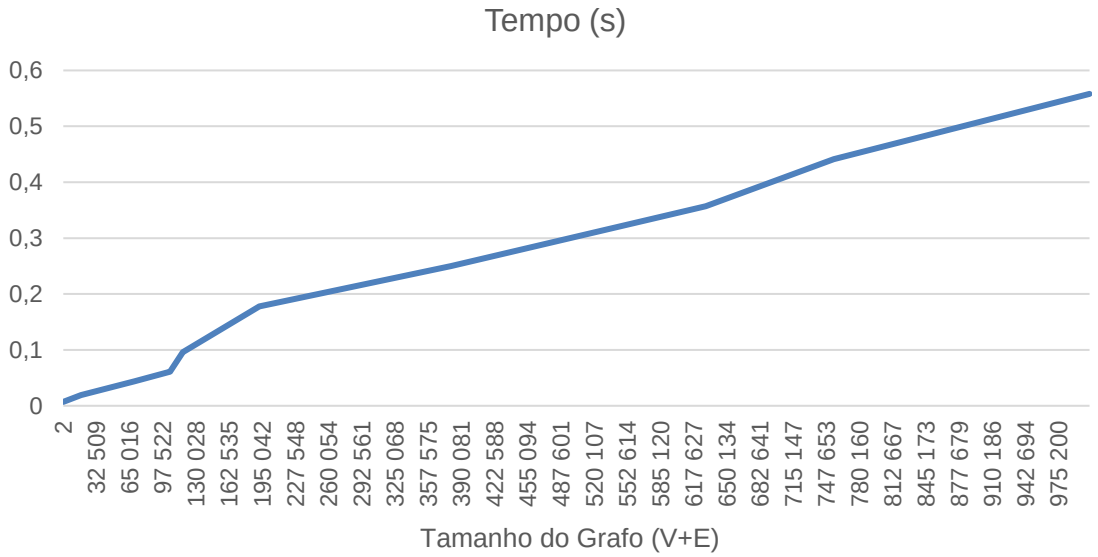
Análise teórica da complexidade total e das várias etapas da solução proposta:

- Leitura dos dados de entrada: simples leitura do input, com ciclos (1) e (2) a depender linearmente de V e E respetivamente. Logo, Θ(V+E).
- Processamento do input na função topSortMaxDistance(G):
  - (3) é um ciclo que, no pior caso tem V iterações. Logo, O(V).
  - (4) ciclo *while* que é executado V vezes (caso não o seja significa que existe um ciclo no grafo). Logo, O(V).
  - (5) ciclo que percorre todos os arcos apenas uma vez. Logo, O(E).
  - Apesar do ciclo (5) se encontrar dentro do ciclo (4), os vértices são percorridos apenas uma vez, tal como os arcos, sendo assim a complexidade total desta função O(V + E).
- Apresentação dos dados. O(1).

Complexidade global da solução: O(V + E).

Avaliação Experimental dos Resultados

Para testar o programa, criámos uma série de grafos usando o algoritmo fornecido pelo professor, *randomDAG*, com 5000 vértices e probabilidade de criação de arcos variável com valores entre 0,001 e 0,08. A *seed* usada foi “1”.



Por observação do gráfico, concluímos que a complexidade teórica coincide com a experimental, apesar dos resultados iniciais não estarem na reta esperada, o que provavelmente se deve às complexidades constantes terem mais peso em relação às lineares.