RELATÓRIO DO PROJETO 1

ANÁLISE E SÍNTESE DE ALGORITMOS • 2º SEMESTRE, 2016/2017
GRUPO 29: MIGUEL REGOUGA, 83530 • PEDRO CALDEIRA, 83539

INTRODUÇÃO

No âmbito do primeiro projeto da cadeira de Análise e Síntese de Algoritmos, foi-nos apresentado um problema que consiste num sistema de datação relativa de fotografias. O objetivo do programa seria devolver a ordenação por ordem cronológica de uma série de fotografias, sendo que a cada fotografia é atribuído um número e o utilizador especifica, de entre 2 fotografias, qual a que crê ser mais recente.

A nossa solução para este problema baseou-se na utilização da teoria de grafos: cada vértice representa uma fotografia e as arestas representam a relação temporal entre as demais.

Implementámo-la na linguagem de programação C, sendo que como estruturas de apoio usámos uma lista de adjacências. Recorremos também ao algoritmo de ordenação topológica, o qual tivemos de estudar *a priori* através dos slides disponibilizados pelo corpo docente.

DESCRIÇÃO DA SOLUÇÃO

Como referido no enunciado do projeto, a primeira linha de input contém o número de fotografias e o número de ligações entre elas. Na nossa conceptualização, isto traduz-se no número de vértices e no número de arestas do grafo, respetivamente.

Deste modo, a função *main* servirá, numa primeira fase, para ler a primeira linha do input e criar o grafo (lista de adjacências) com o número de vértices que foi lido, usando a função auxiliar ***createNewGraph***. Posteriormente, é criada uma ligação entre os vértices referidos em cada linha de input, usando para esta tarefa a função ***addConnection***.

A função ***addConnection*** tem como principal objetivo criar ligações entre dois vértices do grafo. Nesta função é atualizado também um vetor global a todo o programa, ***entradas***, que regista o número de entradas que um certo vértice tem. Quando é estabelecida uma ligação, a posição do vetor correspondente ao vértice de entrada é incrementada. Este vetor tem uma extrema importância a nível de eficiência quando for utilizado no algoritmo de ordenação topológica, pois evita iterações desnecessárias. De forma a tornar a manipulação dele mais compreensível, o índice zero do vetor nunca terá nenhum valor.

Após o grafo ter sido criado e as ligações entre os vértices terem sido estabelecidas, o programa efetua a ordenação topológica dos vértices, sendo esta a parte fulcral deste projeto. Este algoritmo de ordenação foi utilizado porque dá a ordem dos eventos (no nosso caso, fotografias) de um grafo dirigido acíclico (DAG). No nosso programa, esta ordenação é feita com recurso a uma função auxiliar, a ***topologicalSort***.

Para implementarmos a ordenação topológica, começamos por percorrer o vetor ***entradas*** referido anteriormente. Caso um dos índices desse vetor esteja a zero, isto é, caso haja um vértice que não tenha arcos de entada, o índice desse vértice é colocado no vetor auxiliar ***listQ***.

De seguida, passamos para um novo ciclo que só irá terminar quando a nossa ***listQ*** não tiver nenhum valor no seu interior. No início deste ciclo *while* existe uma verificação que averigua se a nossa ***listQ*** tem mais que um valor ou não. Caso a ***listQ*** tenha mais que um elemento, significa que existem vários vértices que não têm arcos de entrada, logo é impossível estabelecer uma ordenação válida entre os vértices. Assim, é levantada uma *flag* (***flagInsuf***) que será revista mais tarde.

Após esta verificação, examinamos o primeiro elemento (*u*) da ***listQ*** e começamos a iterar a lista de adjacências correspondente ao índice desse mesmo elemento. Esta iteração serve maioritariamente para verificar se os vértices que têm arcos vindos de *u* têm algum arco de entrada (sem ser o de *u*) ou não. Caso não tenham, são colocados na ***listQ*** e serão processados numa nova iteração deste ciclo *while*.

Para concluir o ciclo, é colocado o atual primeiro elemento da ***listQ*** numa outra lista, a ***listL***, que será usada para imprimir as "fotografias" na ordem topológica correta.

Tendo percorrido todos os elementos da ***listQ***, são feitas duas verificações. A primeira apura se o número de elementos na ***listL*** é menor que o número de vértices total no grafo. Se isto se verificar, significa que estamos perante um grafo cíclico, sendo que as ligações criadas entre os vértices não fazem sentido no âmbito do que é pedido no projeto. Logo, é impressa a mensagem "Incoerente" e o programa acaba.

Caso não se tenha verificado a última condição, o programa averigua se o valor da ***flagInsuf*** foi alterado ou não. Em caso afirmativo, é impressa a mensagem "Insuficiente" e computação termina.

A ordem destas duas situações é propositada, uma vez que se o input for insuficiente e incoerente, dá-se primazia à sua incoerência, não necessitando de verificar se é insuficiente ou não.

Como passo final, e caso nenhuma das verificações anteriores tenha sido registada, é impresso no standard output o vetor ordenado ***listL***.

ANÁLISE TEÓRICA

A parte inicial do programa tem complexidade O(V) na inicialização dos vértices e O(E) na criação das arestas.

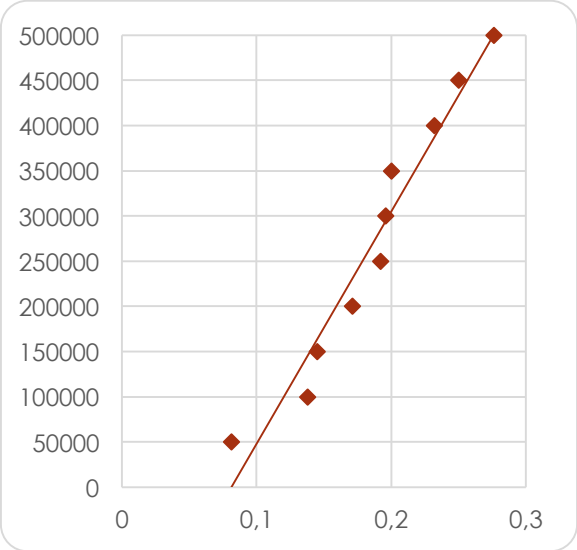
A função ***topologicalSort*** começa por percorrer todos os vértices do grafo através do vetor ***entradas*** referido anteriormente, sendo esta operação de complexidade O(V).

Após esta etapa, entramos no ciclo fundamental do algoritmo. Uma vez que este percorre mais uma vez todos os vértices da ***listQ,*** a complexidade será O(V), já que, no pior caso, todos os vértices podem não ter arcos de entrada.

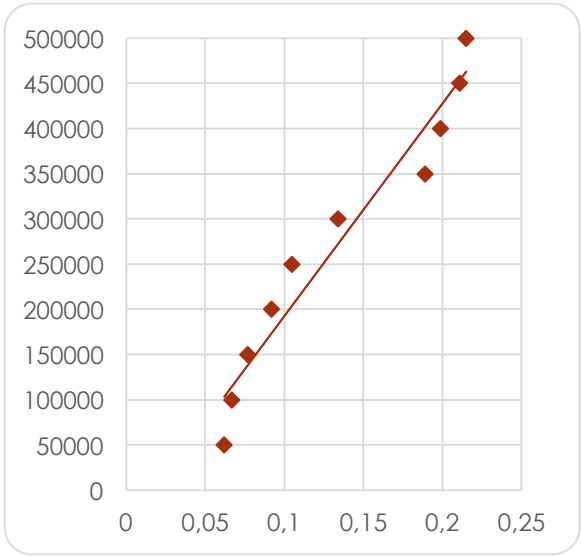
Para esta última verificação, e apoiados pelo pseudo-código, temos de verificar as arestas relativas a cada vértice, sendo esta operação de complexidade O(E).

Assim, a complexidade final do algoritmo é **O(V+E)**.

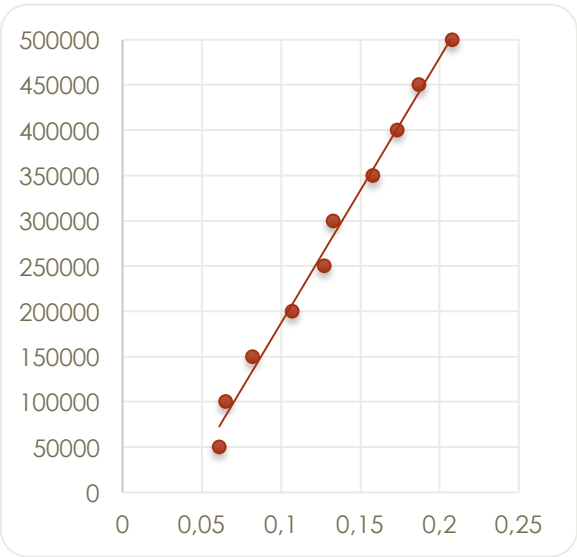
ANÁLISE EXPERIMENTAL



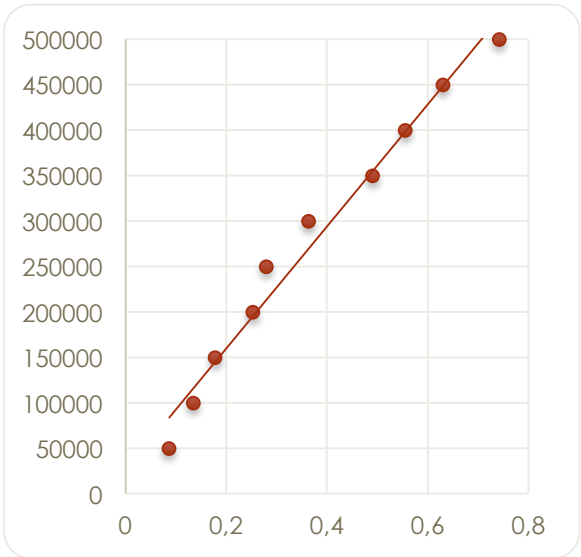
Variação do número de arestas em função do tempo num grafo com um número fixo de vértices (50000) — caso de ordenação topológica bem-sucedida



Variação do número de arestas em função do tempo num grafo com um número fixo de vértices (50000) — caso incoerente



Variação do número de arestas em função do tempo num grafo com um número fixo de vértices (50000) — caso insuficiente



Variação do número de vértices em função do tempo num grafo com um número fixo de arestas (50000) — caso de ordenação topológica bem-sucedida

Apoiados pelos gráficos acima, podemos verificar que a execução do algoritmo é linear (quanto mais arestas, maior é o tempo de execução). O mesmo se verifica quando o número de arestas é constante (quanto mais vértices, maior é o tempo de execução). Assim, podemos então concluir que a complexidade final do nosso programa é **O(V+E)**.