Relatório 1º Projeto ASA 2020/2021

Grupo: al83

Alunos: Catarina Bento (93230) e Luís Freire D'Andrade (94179)

Descrição do Problema e da Solução

Segundo o enunciado, o problema consiste, dada uma sequência de dominós, em determinar qual o número mínimo de dominós que se tem de deitar abaixo, de forma a garantir que todos os dominós caiem, e também qual o número de peças pertencente à maior sequência de dominós a cair, de cada vez que se deita abaixo um dominó. Na nossa solução, implementada em linguagem C++, a sequência de dominós é interpretada como um grafo direcionado e acíclico, pois não pode ter ciclos (DAG).

A resposta à primeira pergunta é facilmente determinada, uma vez que, se partirmos do princípio que é preciso deitar abaixo mais do que um dominó para toda a sequência cair, significa que o DAG tem mais do que uma fonte (nó com grau de entrada igual a zero), dado que são os únicos tipos de nós impossíveis de aceder a partir de qualquer outro. Logo, o número de fontes é o número mínimo de dominós que se tem de deitar abaixo. No código, inicialmente, todos os nós são considerados fontes, mas à medida que os dados de entrada são lidos, as fontes são atualizadas.

A segunda pergunta já não é tão prontamente computada, visto que o problema do maior caminho num grafo não é tão simples quanto o do mais curto. Isto deve-se ao facto de o caminho mais longo não gozar da propriedade da subestrutura ótima (ou seja, os subcaminhos do caminho mais longo não são necessariamente os mais longos entre os nós desse subcaminho). Mas, é fácil perceber que o tamanho do caminho mais longo vai ser a maior distância de um nó a uma das fontes. No código, o foco da implementação é, após identificadas as fontes, a ordenação topológica dos nós do grafo. Em vez de se aplicar uma DFS para cada fonte, são apenas dadas as corretas distâncias aos seus nós adjacentes, que depois propagam estas distâncias para as suas adjacências, e assim consecutivamente. Esta propagação é efetuada pela ordem topológica, de modo a se saber quais nós visitar e "atribuir distância" primeiro (para não se propagar uma distância de um nó que podia ser maior).

(De notar que o código foi inspirado na solução apresentada aqui).

Análise Teórica

Nesta análise teórica, considera-se V como o número de nós do grafo de input, e E como o número de arcos desse mesmo grafo.

- Leitura dos dados de entrada e construção do grafo: simples leitura do input, com um ciclo a depender linearmente de E. Logo, Θ(E)
- Procura das fontes: ciclo que percorre um vetor de inteiros (que representa os in-degrees dos vértices do grafo), onde se a entrada *i* possuir o valor 0, então o nó *i* do grafo é uma fonte. Logo, O(V)[1]
- Ordenação topológica dos nós do grafo: depois de adicionar as fontes ao vetor (futuramente) ordenado, percorrer esse mesmo vetor; por entrada, aceder às suas

[1] A complexidade é O(V) e não Θ(V) uma vez que, antes da procura, se sabe qual o número de fontes presentes no grafo (a partir da leitura dos dados de entrada), logo, o ciclo pode ser interrompido quando todas as fontes forem encontradas.

adjacências; para cada nó adjacente, decrementar o seu valor no vetor dos in-degrees e, se agora o seu in-degree corresponder a 0, adicioná-lo ao vetor ordenado. Logo, O(V+E)

- Obter as distâncias de todos os nós acessíveis (e escolher a maior): para cada fonte, ciclo que percorre o vetor dos nós ordenados topologicamente; para cada nó/entrada, se este estiver ligado à fonte que se está a analisar, acede às suas adjacências e, para cada nó adjacente, se a distância do nó pai (atual) for maior ou igual que a sua, atualizar (todas as distâncias são inicializadas a 0). Cada vez que uma distância é atualizada, é comparada com uma variável que contém a maior distância encontrada até agora, e a variável é atualizada se esta distância for maior. Logo O(V+E)
- Apresentação dos dados. O(1)

Complexidade global da solução: O(V+E)

Avaliação Experimental dos Resultados

As experiências foram realizadas num Windows Subsystem para Linux com o Sistema Operativo Manjaro, com o processador Intel Core i5-10400F e 16GB de Ram.

Foram gerados 10 grafos de tamanho incremental. De seguida, foi cronometrado o tempo de execução do programa para cada um dos grafos gerados. Como resultado, foi originado o gráfico da Figura 1.

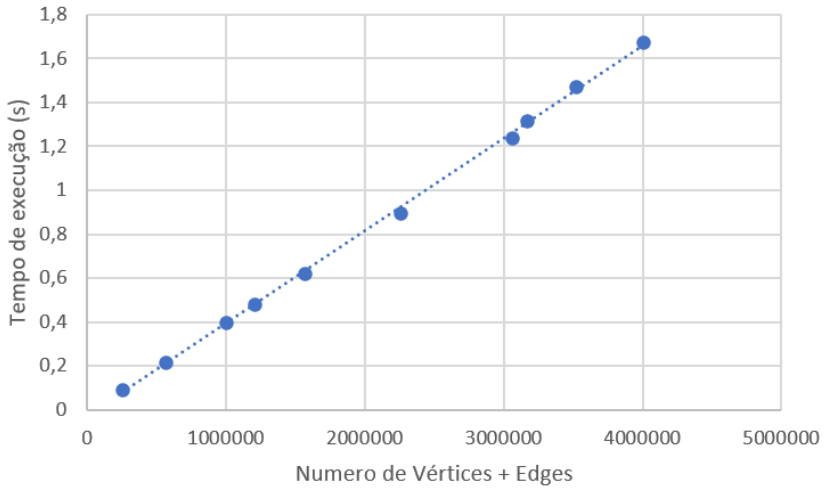

Figura 1 - Gráfico de Tempo de Execução em função de V+E

Sendo que a linha de tendência linear do gráfico (com o tempo de execução em função de V+E) se revela bastante próxima de todos os pontos, pode-se concluir que o gráfico gerado está concordante com análise teórica acima descrita, pois é possível observar que o tempo de execução do programa cresce linearmente com V+E. Logo, a complexidade global da solução do algoritmo verifica-se, O(V+E).