Relatório 1º projecto ASA 2020/2021

Grupo: al41

Aluno(s): Gonçalo Guerreiro (95581) e Vasco Correia (94188)

Descrição do Problema e da Solução

Neste primeiro projeto da cadeira de ASA foi-nos apresentado o seguinte problema: Dada uma sequência de peças de dominó, em contacto umas com as outras, determinar o número mínimo de intervenções necessárias para garantir que todos os dominós caem e, o tamanho da maior sequência de dominós a cair.

Fizemos a correspondência da sequência de peças de dominó a um grafo dirigido acíclico, em que os vértices são as peças de dominó e as arestas são as ligações entre estes. Para a primeira parte do problema proposto (número mínimo de intervenções para derrubar todas as peças de dominó), observámos que as peças que não podem ser derrubadas por nenhuma outra são “sources” no grafo, sendo assim apenas necessário contar o número destes no grafo para resolver este problema. Para a segunda parte do problema (número de peças da maior sequência de dominós a cair), concluímos que ao linearizar o grafo (usando o algoritmo DFS para obter uma ordenação topológica) simplificamos a relação entre as peças, ou seja se A derruba B, B aparece depois de A na ordenação topológica, sendo assim apenas necessário percorrer o vetor ordenado para obter o maior caminho.

Referências

Mumit Khan, “Longest Path in a directed acyclic graph (DAG)”

“Lawrence L. Larmore, “DFS and BFS Algorithms using Stacks and Queues”

Análise Teórica

- Leitura dos dados de “input” do “stdin”: simples leitura do input, construindo simultaneamente o grafo acima referido e o seu transposto, com um ciclo a depender linearmente do número de dependências (número de arestas). Logo, O(|E|).

- Processamento do grafo transposto para determinar as “sources” (pois por definição um “source” não tem “in-going edges”, logo no grafo transposto, o mesmo vértice não terá “out-going edges”). Assim percorremos a lista de adjacências do grafo transposto, verificando o número de “out-going edges” e caso este seja igual a 0, adicionamos a um vetor que no final irá ser devolvido. Este processamento tem um ciclo que depende linearmente do número de vértices e para cada um destes, a verificação do respetivo número de “out-going edges” é constante. Logo, O(|V|).

- Aplicação do algoritmo DFS para obter a ordenação topológica dos vértices do grafo. Cada vértice e cada aresta são apenas visitados uma vez, Logo O(|V|+|E|)

- Aplicação de um algoritmo semelhante ao descrito por Mumit Khan em “Longest path in a directed acyclic graph” diferindo no facto da ordenação topológica no nosso caso ser feita previamente. O primeiro ciclo inicializa a distância de todas as “sources” a 0 e depende linearmente do número das mesmas logo, O(|V|). O segundo ciclo percorre

Relatório 1º projecto ASA 2020/2021

Grupo: al41

Aluno(s): Gonçalo Guerreiro (95581) e Vasco Correia (94188)

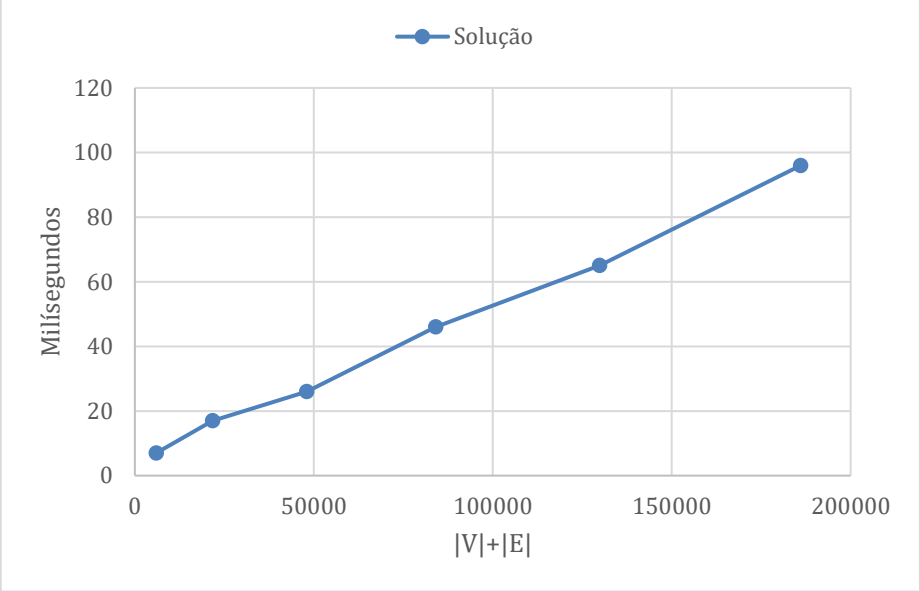
---

todos os vértices da ordenação topológica, percorrendo para cada um destes, todas as suas arestas logo, O(|V| + |E|). O terceiro e último ciclo percorre todos os vértices, comparando as suas distâncias, dependendo assim linearmente do número destes logo, O(|V|). Sendo assim, o algoritmo O(|V| + |E|).

- Apenas é feito “print” de uma string com dois números no “stdout”. Logo, O(N), sendo N o comprimento da string.

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

Avaliação Experimental dos Resultados



Para verificar a complexidade estimada, foi testado o programa com vários grafos gerados, com |V| + |E| a variar entre 5000 e 190000. Verificámos assim que o gráfico gerado do tempo de execução em função do tamanho do input tem crescimento linear, tal como previsto.

O programa foi testado num computador com as seguintes especificações:

- Processador Intel(R) Xeon(R) CPU E5-2620 com 6 cores e 2.00 Ghz
- 16 Gb de Memória RAM
- Sistema Operativo Linux (Debian)