UNIVERSIDADE DE ÉVORA

2º Trabalho - Hard Weeks

Estrutura de Dados e Algoritmos II

Professora: Vasco Pedro
Realizado por: Filipe Alfaiate (43315), Miguel de Carvalho (43108)

9 de maio de 2021

1 Algoritmo

O nosso algoritmo consiste na utilização de um **grafo orientado acíclico** onde os vértices representam tarefas e o arco que vai da `tarefa1` à `tarefa2` corresponde a uma dependência da `tarefa2` face à `tarefa1`, ou seja, a `tarefa1` tem que ser concluída antes da `tarefa2` (`tarefa1 < tarefa2`).

A utilização deste **algoritmo** baseia-se num planeamento de projeto que dado um **número de tarefas** mostra o **maior número de tarefas** numa única semana e o **número de semanas más**.

2 Análise do Grafo

Utilizando a **ordenação topoplógica**, a construção do **grafo** (exemplo do enunciado) seria:
`0 < 5, 0 < 6, 1 < 0, 1 < 6, 1 < 7, 7 < 2, 5 < 8,`
`8 < 9, 10 < 3, 8 < 6, 4 < 10, 7 < 9, 4 < 11, 6 < 7`

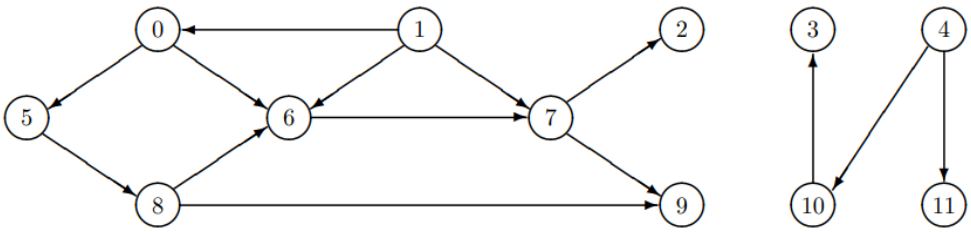

Figura 1: Grafo inicial

Em seguida percorremos todos os **nós** para verificar quais não apresentam antecedentes, colocando os mesmos numa **fila**.

Como estamos a fazer uma pesquisa semanal (pesquisa em largura), as novas tarefas inseridas nunca terão uma semana menor que as semanas que estão à sua frente na fila. Isto significa

que quando **visualizamos a primeira tarefa da fila** vamos verificar que todas as **tarefas restantes na fila** serão da mesma semana, o que indica o número de tarefas que terão de ser realizadas nessa semana. Apenas quando esse grupo de tarefas sair da fila, poderemos iniciar de novo este processo.

Posteriormente retiramos o primeiro elemento da fila e procuramos os seus **precedentes**, atualizando a semana para a semana deste antecedente e decrementamos o número de antecedentes, acrescentando esta tarefa à **fila** se já não tiver **tarefas que a antecedam**.

Este processo é repetido até a **fila ficar vazia**.

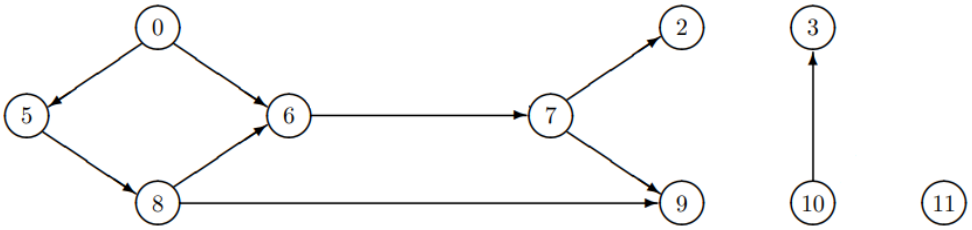

Figura 2: Grafo sem as tarefas da semana 1

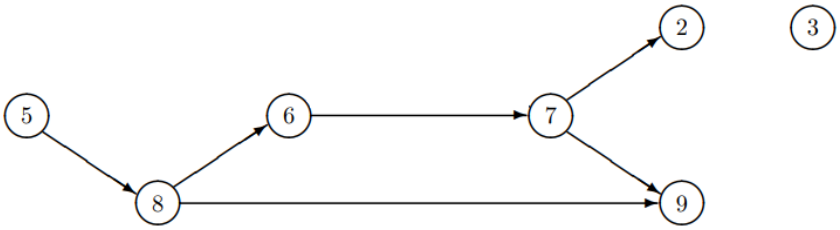

Figura 3: Grafo sem as tarefas da semana 2

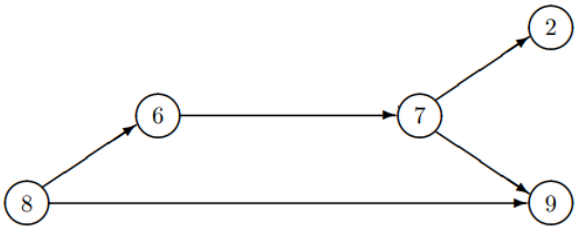

Figura 4: Grafo sem as tarefas da semana 3

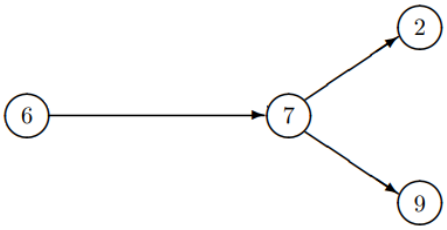

Figura 5: Grafo sem as tarefas da semana 4

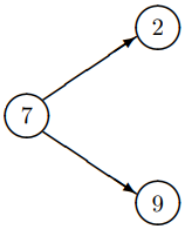

Figura 6: Grafo sem as tarefas da semana 5

Figura 7: Grafo sem as tarefas da semana 6

3 Complexidade

3.1 Temporal

Começámos por proceder à leitura de uma `string` com 3 números, **número de tarefas**, **precedência direta entre tarefas** e o **limite usado para definir uma semana má**, respetivamente, o que origina uma complexidade constane `O(1)`.

Procedeu-se à verificação dos inputs referidos no parágrafo anterior estarem dentro dos limites permitidos do programa, demorando um tempo constante `O(1)` para cada condição.

De seguida declarámos um vetor de listas e uma fila, que apresenta um custo constante `O(1)`.

Posteriormente iniciámos um ciclo para percorrer todos os nós inicializando-os, dando valores e colocando-os numa lista na posição do seu valor, o que tem custo constante, mas como estamos a percorrer todos os nós teremos um custo linear em relação ao número de nós, `O(V)`, sendo `V` o número de nós.

Além disso, teremos um ciclo para criar o grafo, ou seja, estabelecer adjacências entre os **nós**, onde procedemos à leitura de uma string com dois números, representando o arco (t1,t2) que corresponde a `t1 -> t2`. É criada essa ligação e `t2` incrementa o antecedente, todas estas operações apresentam um custo constante `O(1)`, mas como temos que percorrer todas as **precedências diretas entre tarefas** a complexidade é `O(E)` onde `E` é o número de arcos.

Também de forma cíclica percorremos todos os **nós**, após estes terem sido relacionados entre eles, para achar quais os nós sem antecedentes,colocando os mesmos numa **fila**, o que nos dá um complexidade linear `O(V)` onde `V` é o número de nós.

Foram incializidas três variavéis que servem de contadores e apresentam custo constante `O(1)`.

Por fim percorremos a fila até esta se encontrar vazia, como cada **nó** só vai aparecer uma única e exclusiva vez na **fila** iremos ter uma complexidade `O(V)`, sendo `V` o número de **nós**. A cada remoção feita da fila serão realizadas várias afetações e condições, todas elas constantes `O(1)`, excepto a pesquisa de **nós adjacentes** tem custo `O(E)`, onde `E` é o **número de arestas**. Quando a fila estiver vazia, podemos afirmar que o custo é de `O(V+E)`, onde `V` é o **número de nós** e `E` é o **número de arcos**.

$$\mathbf{O(1) + O(1) + O(1) + O(V) + O(E) + O(V) + O(V+E) = O(V+E)}$$

3.2 Espacial

Durante a inicialização do vetor de listas teremos que reservar um espaço em memória onde serão guardados as `V` cabeças da lista, o que irá ocupar em memória `O(V)`, onde `V` é o número de nós do grafo.

Ao inicializar a fila vamos criar um **nó inicial** o que irá ocupar um espaço em memória de `O(1)`

Quando são criados os **nós do grafo** terá de ser guardado individualmente originando assim uma complexidade espacial de `O(V)`, onde `V` é o número de **nós do grafo**. De seguida guardamos os mesmos na cabeça da lista da sua posição.

Quando adicionamos as adjacências pretendidas em cada posição do **vetor de listas** iremos ter uma complexidade de `O(V+E)`, onde `V` é o **número de nós** e `E` é **número de arestas do grafo**.

Para finalizar a **complexidade espacial**, quando manipulamos a fila, adicionamos e removemos **nós** ao longo dessa mesma manipulação, mas tendo em conta o pior caso a fila poderá conter todos os **nós do grafo** o que irá originar uma **complexidade espacial** de `O(V)`, onde `V` é o **número de nós**.

$$\mathbf{O(V) + O(1) + O(V) + O(V+E) + O(V) = O(V+E)}$$

4 Comentários Adicionais

Na realização deste trabalho foi usado uma **disciplina de fila**, pois era a única que mantinha uma ordem sequêncial em relação ao número da semana, mas qualquer outra disciplina serviria para o mesmo fim, apenas não de uma forma tão direta e possivelmente com **maior custo temporal ou espacial**.

Uma outra observação a ter em conta foi a escolha de um **vetor de listas de adjacências**, em vez de uma **matriz de adjacências**. Esta escolha não foi baseada só em termos de **complexidade temporal/espacial** mas também na forma como a informação é guardada em memória, pois numa **matriz de adjacências** a informação teria de ser guardada sequencialmente arranjando um espaço $\mathtt{V}^2$, onde `V` é o **número de nós do grafo**, e um **vetor de listas de adjacências** apenas teria que guardar sequencialmente o vetor onde se encontra a cabeça da lista e essa de seguida tem um apontador para a posição em memória do próximo elemento da lista, isto significa que esse **nó em memória** poderá estar em qualquer parte da memória facilitando a sua **organização em memória**.