Hard Weeks

Autores:
Ricardo Oliveira - nº42647
Vasco Barnabé - nº42819

14 de Maio de 2021

1 Introdução

O 2º trabalho de Estruturas de Dados e Algoritmos II consiste na implementação de uma solução para o problema Hard Weeks.

Neste problema pretende-se realizar um projeto, composto por tarefas, onde algumas dependem de outras, e por isso, só podem ser realizadas após a resolução das tarefas de que dependem.

Cada tarefa demora uma semana a ser realizada, no entanto, numa semana podem ser realizadas várias tarefas. Uma vez que se pretende realizar o projeto o mais rápido possível, cada tarefa será iniciada imediatamente após todas as tarefas das quais depende estarem concluídas. Consequentemente, existirão semanas em que serão realizadas muitas tarefas. Todas as dependências entre tarefas são fornecidas, assim como o limite para o número de tarefas realizadas numa semana. Este limite pode ser superado, no entanto essa semana será considerada uma *hard week*.

Assim, pretende-se descobrir qual o número máximo de tarefas realizadas numa semana, e em quantas semanas o limite de tarefas por semana foi excedido.

2 Desenvolvimento / Implementação

2.1 Pensamento

Para a resolução deste problema, foram guardadas em diferentes variáveis, três informações: o número de tarefas; o número de precedências diretas entre tarefas; e o limite usado na definição de *hard week*.

Como era nosso objetivo distribuir as tarefas por semana, e só no fim, calcular qual o maior número de tarefas realizadas numa semana e o número de semanas em que o número de tarefas efetuadas superou o limite, o pensamento foi, durante o algoritmo, construir, de forma subentendida, uma ordem topológica das tarefas, isto é, analisar em primeiro lugar a(s) tarefa(s) que não dependesse(m) de nenhuma(s) outra(s), e após a análise a essas tarefas, analisar aquelas que já só dependiam daquela tarefa, levando este pensamento até que todas as tarefas estivessem analisadas (realizadas).

Por fim, calcular os valores pretendidos, o maior número de tarefas realizadas numa semana e o número de semanas em que o número de tarefas efetuadas superou o limite fornecido.

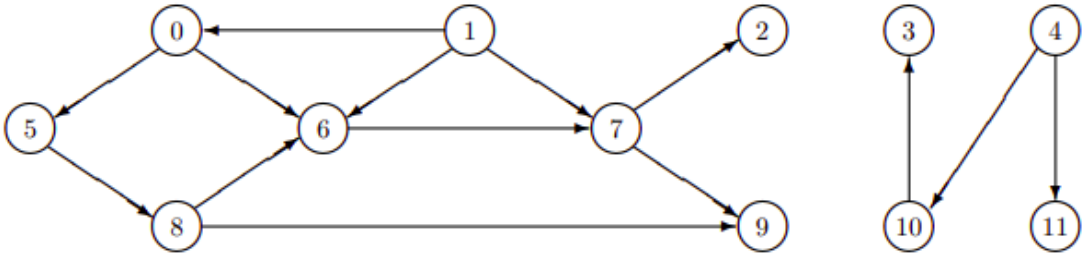
2.2 Descrição do Algoritmo

As tarefas foram representadas num grafo. Para a representação deste grafo, foi criado, sabendo previamente o número de tarefas do problema, um array com tamanho X, sendo X o número de tarefas. Uma vez que as tarefas estariam representadas num grafo, cada nó do grafo corresponde a uma tarefa. Assim, este array criado contém, em cada posição, um objeto do tipo *HardWeeks*. Desta forma, cada posição do array irá corresponder ao número da tarefa, a informação no index 0 vai corresponder à tarefa 0, o index 1 à tarefa 1, até ao fim do array, uma vez que as tarefas são identificadas por inteiros que variam no intervalo [0;T-1], sendo T o número total de tarefas.

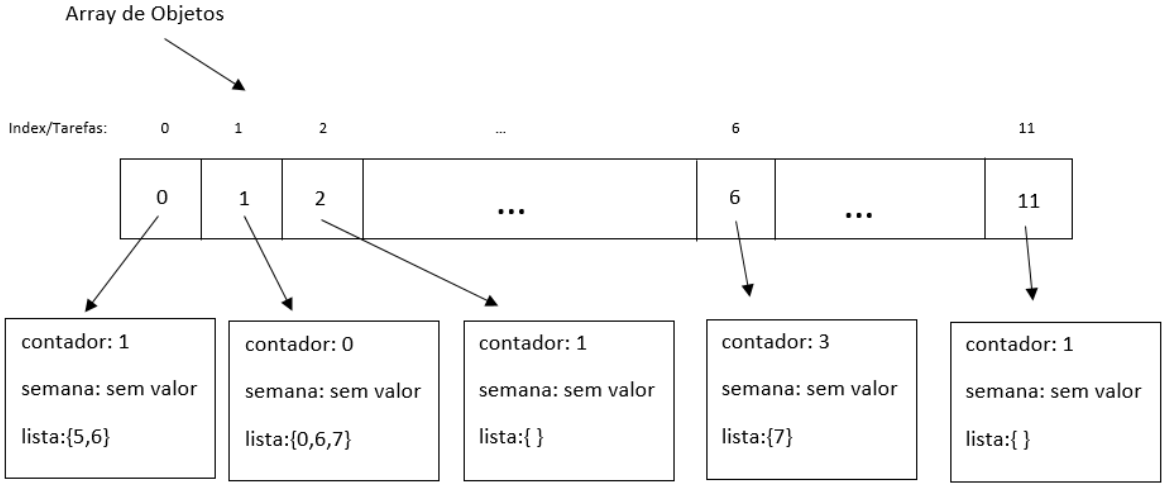
Cada objeto corresponde assim à tarefa da posição em que está inserido no array, e cada objeto terá:

- um contador (inteiro) onde será registado o número de tarefas de que aquela tarefa depende diretamente;
- uma semana (inteiro) onde será registado, durante o desenvolvimento do programa, a semana em que aquela tarefa foi realizada;
- uma lista de tarefas (LinkedList de inteiros), onde estão contidas todas as tarefas que dependem diretamente da tarefa em questão.

Desta forma, visualizando como exemplo o seguinte grafo:



A informação ficará representada, inicialmente, da seguinte forma:



Assim, com a informação disposta desta forma, está representado, através das dependências diretas entre tarefas, o grafo pretendido.

É importante realçar que, nesta fase, nenhuma tarefa tem uma semana atribuída, apenas no fim do algoritmo esse campo estará preenchido em todas as tarefas.

Dá-se então, nesta fase, o início do algoritmo em si.

Em primeiro lugar, é feita uma pesquisa para encontrar todas as tarefas que não dependem de nenhuma outra tarefa, e por isso, não têm nenhum arco a apontar na sua direção. Isto significa que estas tarefas podem, e devem, ser realizadas na primeira semana do projeto, ficando a sua semana definida com o valor 1. Estas tarefas são encontradas através do contador que cada tarefa tem. Uma vez que não dependem de nenhuma outra tarefa, o seu contador estará a 0, e por isso, estas tarefas são selecionadas e colocadas numa lista (LinkedList), que adota um comportamento de uma Queue (fila) do tipo FIFO (first in first out), ou seja, o primeiro elemento a entrar na lista, será o primeiro elemento a sair.

Após todas as tarefas independentes serem colocadas na lista, serão analisadas uma a uma, retirando para análise a primeira tarefa colocada na lista.

Uma vez que já foi selecionada a tarefa para análise, o contador de todas as tarefas que dependem diretamente desta será decrementado em uma unidade (isto para simular que a tarefa "pai"já foi realizada, e que por isso, as tarefas "filhos"já não dependem dela para poderem ser realizadas). Após a decrementação, caso alguma tarefa fique com o seu contador a 0, isso significa que essa tarefa já não depende de nenhuma outra para poder ser realizada, e, por isso, é adicionada à fila de tarefas criada. Além disso, a semana em que essa tarefa foi realizada é atualizada, sendo-lhe atribuída a semana seguinte à semana em que foi realizada a última tarefa de que dependia.

Este procedimento repete-se até que a fila de tarefas se encontre vazia.

Após a fila estar vazia (altura em que todas as tarefas foram realizadas), no array de tarefas ficou registado, em cada tarefa, a semana em que a mesma foi realizada.

Por isso, é necessário percorrer todas as tarefas e verificar em que semana cada uma foi realizada, e contabilizar o maior número de tarefas realizadas numa semana, e em quantas semanas o limite para o número de tarefas realizadas numa semana foi ultrapassado.

2.3 Complexidade Temporal e Espacial

2.3.1 Complexidade Temporal

A complexidade temporal do algoritmo deve ser analisada consoante a sua estrutura, assumindo-se que as instruções de declaração, atribuição, operação e condição têm custo constante, isto é, O(1).

Numa fase inicial, analisando, consoante o número de tarefas (nós) dadas - T, a complexidade do primeiro ciclo será θ(T), pois é executado sempre T vezes; por outro lado, a complexidade do segundo ciclo é θ(P), sendo P o número de dependências diretas (arcos entre nós), uma vez que este ciclo é executado sempre P vezes e todas as instruções executadas em cada iteração têm custo constante.

Ao longo da restante parte do programa, surge o algoritmo em si, no método *solve*. O primeiro ciclo (*for* - linha 24) será executado T vezes, sendo T, como já foi referido, o número de tarefas, resultando deste ciclo uma complexidade de θ(T).

O ciclo que se inicia na linha 33 é executado até que a fila de tarefas se encontre vazia. No início de cada iteração é retirado um nó da lista, e uma vez que cada nó apenas é inserido e retirado da lista uma vez e no fim a lista tem de se encontrar vazia, este ciclo será executado, sempre, T vezes. No interior deste ciclo, um outro ciclo é executado. Este, por sua vez, a cada iteração do ciclo exterior, é executado A vezes, sendo A o número de tarefas que dependem diretamente da tarefa que foi removida da lista naquela iteração(executada no ciclo exterior). Este ciclo não é sempre executado o mesmo número de vezes em cada iteração do ciclo exterior, mas, uma vez terminada a execução do ciclo da linha 33(exterior), o ciclo interior foi sempre

executado P vezes, sendo P o número de arcos do grafo. Assim, uma vez que ambos os ciclos exterior e interior são sempre executados, respetivamente, T e P vezes, pode-se dizer que estes ciclos apresentam uma complexidade de θ(T+P). Aqui, as operações de remover e adicionar uma tarefa à lista não têm influência na complexidade por apresentarem ambas as operações um custo constante.

Por fim, surge um ciclo com início na linha 53, que contém um ciclo interior que se inicia na linha 56. O ciclo interior é executado T vezes, e o exterior é executado, no pior caso, T vezes (caso em que em cada semana apenas é realizada uma tarefa). Uma vez que todas as restantes operações executadas em cada um dos ciclos, acarretam um custo constante, estes ciclos apresentam uma complexidade de $O(T^2)$.

Assim, pode-se concluir que são estes últimos ciclos analisados os responsáveis por atribuir, ao programa, uma complexidade temporal de $O(T^2)$.

2.3.2 Complexidade Espacial

Relativamente à complexidade espacial, pode-se dizer que a complexidade espacial do algoritmo em si é θ(T), sendo T o número de tarefas, uma vez que é sempre criada uma lista que poderá ter T elementos.

No entanto, no que diz respeito à complexidade espacial do programa, é necessário referir que para guardar a informação inicialmente fornecida, é criada uma lista de adjacências, onde são listadas as adjacências de cada tarefa, e por isso, o programa apresenta uma complexidade espacial de θ(T+P), sendo T o número de tarefas e P o número de dependências diretas.

3 Conclusão

Com este trabalho foi-nos possível aprimorar os nossos conhecimentos sobre grafos, e obter no fim um programa executável que, recebendo como input as informações pedidas, representa estas num grafo. O programa analisa o grafo construído com um algoritmo que cria, de uma forma subentendida, uma ordenação topológica das tarefas do problema, levando a que no fim se saiba em que semana foi realizada cada tarefa.

Por último, utilizando os resultados gerados, é-nos possível encontrar o número máximo de tarefas realizadas numa semana, e em quantas semanas o limite de tarefas por semana foi excedido.