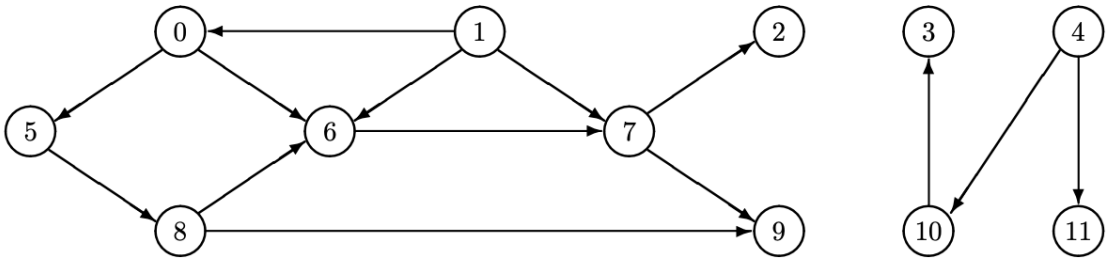
Universidade de Évora

Departamento de Informática

Estruturas e Dados de Algoritmos II

Ano letivo 2020-2021

Hard Weeks



Docente
Professor Vasco Pedro

Discentes
Pedro Claudino nº39870

Ludgero Teixeira nº41348

Grupo do mooshak
g211

Maio, 2021

1. Introdução

Este trabalho consistia na implementação de um programa que dado um plano de um projeto (com suas tarefas e a relação de precedência entre elas) e um determinado número de tarefas, é pretendido que se descubra o número máximo de tarefas a serem realizadas em uma única semana e o número de semanas difíceis (ou seja, o número de semanas com mais de L tarefas para realizar).

*Sendo que L é o número máximo de tarefas que se pode realizar numa semana, sem que essa semana seja considerada uma **hardweek**.

2. Algoritmo

O algoritmo começa por verificar se todos os nós do grafo estão no final (ou sejam se não contêm nenhum nó adjacente), o que corresponde á condição de paragem. Caso ainda existam nós que não tenham chegado ao final, todos os nós são percorridos e as tarefa/nós que poder ser feitas são adicionadas numa fila e o seu peso é mudado para " -1 ", como não pode existir um peso negativo é assim que excluímos o facto de essa tarefa poder ser feita novamente

O algoritmo começa por verificar um array que contem os pesos de cada nó de modo a saber se as tarefas já foram todas concluídas, caso não tenham sido todas concluídas, o algoritmo irá percorrer o array onde é guardado o peso de cada nó, no intuito de encontrar as tarefas que possam ser feitas naquele momento, ou seja, as tarefas que não têm precedentes, e coloca as numa fila, sendo que ao fazer isso muda o seu peso desse nó para "-1" de modo a saber que essa tarefa foi concluída.

A quantidade de tarefas que poderão ser feitas naquele momento, ou seja, as tarefas que se encontram na fila, são contadas e esse número será correspondente ao total de tarefas dessa semana que de seguida será verificado de modo a descobrir se essa semana será uma **hardweek**.

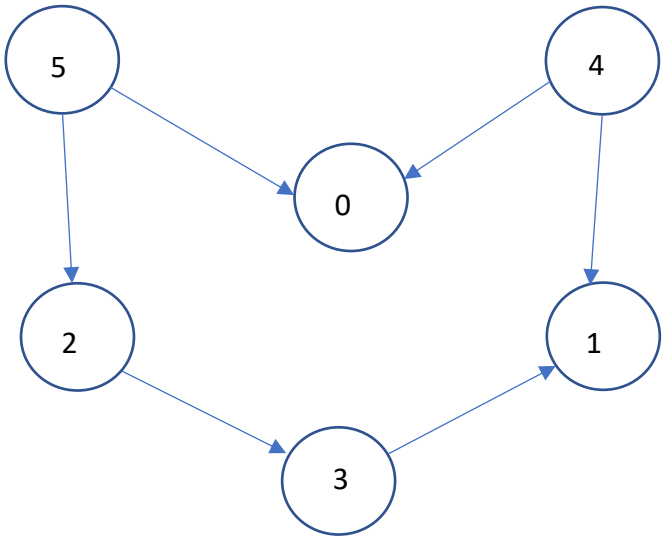
Ainda é feita mais uma verificação de modo a descobrir se o número total de tarefas dessa mesma semana é o número máximo de tarefas, comparando sempre com o número de tarefas da última semana em que se realizaram mais tarefas, caso seja esse valor é guardado numa variável auxiliar.

Por fim irá percorrer uma fila selecionando sempre a cabeça da mesma e a tarefa que se encontra na fila nesse instante.

São percorridas as suas tarefas adjacentes e a cada uma dessas tarefas é incrementado o valor "-1" de modo a conseguirmos descobrir qual a próxima tarefa que poderá ser feita e no final de ser feita esta operação sobre as tarefas adjacentes a tarefa que estava no head da fila é removida passando assim á próxima tarefa na fila, este processo repete até que a fila esteja vazia.

Que resulta no print da variável auxiliar que contem o valor máximo de tarefas numa semana e a quantidade de **hardweeks**.

3.Exemplo do funcionamento do algoritmo



Limite de tarefas semanais=1

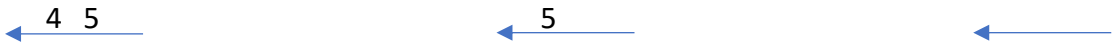
Lista de adjacências :

índice	0	1	2	3	4	5
adjacências	-	-	3	1	0, 1	0, 2

Array de custo :

índice	0	1	2	3	4	5
peso	2	2	1	1	0	0

Fila 1º iteração: Fila 2º iteração: Fila 3º iteração:

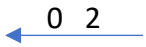


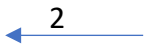
Número de tarefas semanal = 2
Número de hardweeks=1
Número máximo de tarefas=2

Array de custo :

índice	0	1	2	3	4	5
peso	0	1	0	1	-1	-1

Fila 4º iteração: Fila 5º iteração: Fila 6º iteração:



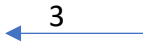


Número de tarefas semanal = 2
Número de hardweeks=2
Número máximo de tarefas=2

Array de custo :

índice	0	1	2	3	4	5
peso	-1	1	-1	0	-1	-1

Fila 7º iteração: Fila 8º iteração:

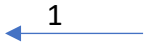


Número de tarefas semanal = 1
Número de hardweeks=2
Número máximo de tarefas=2

Array de custo :

índice	0	1	2	3	4	5
peso	-1	0	-1	-1	-1	-1

Fila 9º iteração: Fila 10º iteração:



Número de tarefas semanal = 1
Número de hardweeks=2
Número máximo de tarefas=2

Array de custo :

índice	0	1	2	3	4	5
peso	-1	-1	-1	-1	-1	-1

4.Análise da complexidade temporal e espacial

4.1 Complexidade temporal

A complexidade temporal do nosso algoritmo é O(n^3), em que **n** são os números de nós/tarefas.

No pior cenário possível, caso os nós/tarefas não tenham sido todos concluídos e a fila **wait** (que contem os nós/tarefas que vão ser realizados numa semana, não se encontre vazia), faz com que exista um ciclo **for** encadeado dentro de 2 ciclos **while**.

4.2 Complexidade espacial

A complexidade espacial do nosso algoritmo é O(n).

private final int[] cost:
Este array contem sempre o número máximo de vértices do grafo.

private final ArrayList<ArrayList<Integer>> adj :
Este array contem sempre o número máximo de vértices do grafo e os vértices adjacentes correspondentes.

Visto que o algoritmo utiliza arrays e a listas, para armazenamento de dados, num pior caso de utilização das mesmas o seu espaço ocupado vai ser **n**, em que **n** é o número de espaços máximo que essas listas ou arrays poderão ter.