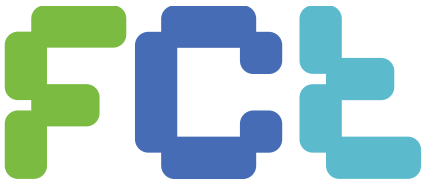

Análise e Desenho de Algoritmos

Relatório do 2º Trabalho

Ana Garcia, MIEI 46783

Abril 2015

Conteúdo

1 Apresentação do Problema

Hard Weeks é um problema de ordenação. Uma empresa tem diversos projectos (tasks) a realizar, sendo que existem projectos com precedências, ou seja, que só podem ser realizados depois de outros.

Numa semana podem ser realizados todos os projectos que não tenham precedências, ou cujas precedências já tenham sido cumpridas.

O nosso objectivo é passar a informação aos funcionários desta fábrica, que dado um número que define o que é considerado uma semana difícil, digamos quantas semanas difíceis (hard weeks) vão existir, e qual o máximo de tarefas existentes numa semana.

2 Resolução do Problema

Após uma leitura cuidada do enunciado é possível verificar que é um simples problema de Ordenação Topológica, mas onde o objectivo não é obter uma ordenação simples, ou seja, uma tarefa de cada vez, mas sim, uma ordenação que realiza todas as tarefas, que fiquem sem precedentes, ao mesmo tempo. Depois disto, é só contar quantas tarefas estão a ser realizadas ao mesmo tempo em cada passo, mantendo sempre um máximo actualizado, e comparando esse número ao que foi dado como sendo uma hard week (semana difícil), se o número for maior, soma-se mais um ao número de semanas difíceis. No fim é só enviar o numero máximo de tarefas e o número de semanas difíceis.

Para isto, realizei algumas alterações no algoritmo de Ordenação Topológica.

Após um primeiro *for* que adiciona todos os vértices sem precedencias numa fila, é retirado o tamanho da fila, e é esse valor que vai corresponder às tarefas dessa semana. Depois os vértices são tratados, um a um, exatamente como o algoritmo faz, retira o vértice a ser tratado da fila, vai aos vértices que tinham esse como precedente, e retira-o de precedente, se esses vértices ficarem com 0 precedentes, adiciona-los à fila. A única diferença aqui, é que existe um *for* que faz com que se pare de tratar vértices quando as tarefas da semana estão feitas, e volta a pedir o comprimento da fila. Este novo comprimento é o número de tarefas da semana seguinte.

Fazendo sempre a análise a cada semana, até o número de tarefas(tasks) / projectos, estarem realizados, é possível enviar então a informação final pedida. O número máximo de tarefas numa semana, e quantas semanas difíceis existem.

3 Implementação do Algoritmo

Na estrutura do programa foi usada uma classe Main, com uma função resolution.

Foi também criada uma classe Graph que implementa a interface Digraph, que representa grafos orientados, e extende a interface AnyGraph.

A interface AnyGraph tem uma função que devolve o número de vértices e outra que dá para adicionar arcos. A interface Digraph tem uma função que a partir de um vértice devolve a lista de vértices de saída.

As estruturas de dados utilizadas na classe Main, é um vector para guardar as precedências por tratar de cada vértice, cujo o tamanho é o número de tarefas (tasks); uma fila FIFO para ir guardando os vértices que já se podem tratar, ou seja, cujas precedências estão tratadas; e tem um objecto do tipo Graph para guardar um grafo.

As estruturas de dados utilizadas na classe Graph é um vector de listas de inteiros, representa os arcos de saída, ou seja, em que cada posição do vector corresponde a um vértice, e a lista aos vértices de saída desse vértice.

4 Análise do Algoritmo

Seja V o número de vértices e A o número de arcos.

A complexidade espacial desta implementação é $\Theta(|V| + |A|)$, porque temos uma lista de inAdjacentes com tamanho $|V|$, e um grafo com tamanho $|V| + |A|$.

A complexidade temporal é de $\Theta(|A|)$ na leitura na main, no método resolution o primeiro ciclo passa por todos os vértices e dentro do while, acaba por também ter de analizar todos os vértices e todos os arcos, o que dá uma complexidade temporal de $\Theta(|V| * 2 + |A|)$. No total temos uma complexidade temporal de $\Theta(|V| + |A|)$.

5 Conclusões

Em suma, o algoritmo de ordenação topológica, com uma pequena alteração, resolveu este problema. A alteração realizada, foi uma de algumas hipóteses pensadas, mas que não foram analizadas, por não alterar a complexidade deste problema.

6 Código Fonte Completo

```java
public interface AnyGraph {
  // Returns the number of vertices.
  int numVertices();

  // Inserts the edge (vertex1, vertex2)
  void addEdge(int vertex1, int vertex2);
}
```

```java
  public interface Digraph extends AnyGraph {
    // Returns the vertices adjacent to the specified vertex along
    // outgoing edges from it.
    Iterable<Integer> outAdjacentVertices(int vertex);
  }
```

```java
  import java.util.LinkedList;
  import java.util.List;

  public class Graph implements Digraph {
    List<Integer>[] outAdjacent;

    @SuppressWarnings("unchecked")
    public Graph(int vertices) {
      outAdjacent = new List[vertices];
      for (int i = 0; i < outAdjacent.length; i++) {
        outAdjacent[i] = new LinkedList<Integer>();
      }
    }
    @Override
    public int numVertices() {
      return outAdjacent.length;
    }
    @Override
    public Iterable<Integer> outAdjacentVertices(int vertex) {
      return outAdjacent[vertex];
    }
    @Override
    public void addEdge(int vertex1, int vertex2) {
      outAdjacent[vertex1].add(vertex2);
    }
  }
```

```java
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.LinkedList;
import java.util.Queue;

public class Main {

  public static void main(String[] args) throws IOException {

    BufferedReader in = new BufferedReader(new InputStreamReader(System.in));

    String[] split = in.readLine().split("␣");
    int tasks = Integer.parseInt(split[0]);
    int precedences = Integer.parseInt(split[1]);
    int hardWeek = Integer.parseInt(split[2]);

    Graph graph = new Graph(tasks);
    int[] inAdjacent = new int[tasks];

    for (int i = 0; i < precedences; i++) {
      String[] edge = in.readLine().split("␣");
      int vertex1 = Integer.parseInt(edge[0]);
      int vertex2 = Integer.parseInt(edge[1]);
      inAdjacent[vertex2] += 1;
      graph.addEdge(vertex1, vertex2);
    }

    resolution(hardWeek, graph, inAdjacent);
  }
```

```
  public static void resolution(int hardWeek, Graph graph, int[] inAdjacent) {
    Queue<Integer> ready = new LinkedList<Integer>();
    int maxTasks = 0;
    int numHardWeek = 0;
    int count = 0;

    for (int i = 0; i < graph.numVertices(); i++) {
      if (inAdjacent[i] == 0) {
        ready.add(i);
        count++;
      }
    }

    while (!ready.isEmpty()) {

      int countWeek = count;
      count = 0;
      if (countWeek > maxTasks) {
        maxTasks = countWeek;
      }
      if (countWeek > hardWeek) {
        numHardWeek++;
      }

      for (; countWeek > 0; countWeek--) {
        int vertex = ready.poll();
        for (int v : graph.outAdjacentVertices(vertex)) {
          inAdjacent[v]--;
          if (inAdjacent[v] == 0) {
            ready.add(v);
            count++;
          }
        }

      }
    }

    System.out.println(maxTasks + "␣" + numHardWeek);
  }

}
```