Práctica 1

Planificación de tareas

Algoritmos Avanzados

Mikayel Mardanyan Petrosyan

22 de Septiembre de 2019

1 Problema

Se dispone de n procesadores (n>0) y de m tareas (m>0) con duración t_i que deben ejecutar dichos procesadores. Las tareas están disponibles desde el inicio y un procesador solo puede ejecutar una tarea al mismo tiempo, ejecutando la tarea en su totalidad.

El objetivo es minimizar el tiempo de espera de todas las tareas, siendo el tiempo de espera de cada una de ellas el tiempo que tardan las tareas anteriores a ella en el mismo procesador en ejecutarse más su tiempo de ejecución.

2 Función de selección

Para minimizar el tiempo de espera total debemos seleccionar las tareas por orden creciente de duración, ya que el tiempo de espera de una tarea es la acumulación del tiempo de su ejecución más los de las tareas precedentes. Por esta razón debemos ejecutar las tareas de menor duración primero.

Asignaremos la tarea más corta al primer procesador, la segunda más corta al segundo procesador y así sucesivamente. Lo normal es que haya más tareas que procesadores, así que utilizaremos la operación módulo para la asignación.

En definitiva, la función de selección consiste en ordenar las tareas por orden creciente de duración y asignarlas en este orden.

3 Implementación

Tendremos dos versiones de este algoritmo, una suponiendo que las tareas están ordenadas y otra sin suponer ninguna ordenación.

La cabecera del método será:

```
public static int tareas (int[] ts, int n)
```

donde el vector *ts* contiene la duración de la ejecución de las tareas, el entero n es el número de procesadores disponibles y el método devuelve el tiempo total de espera de todas las tareas.

3.1 Algoritmo idealizado

Las duraciones de las tareas están ordenadas en tiempo creciente.

```
public static int tareas1 (int[] ts, int n) {

        /* Algoritmo voraz */
        int tiempo = 0;
        int[] tiempos = new int[n];
        for (int i = 0; i < ts.length; i++) {
            tiempos[i%n] += ts[i];
            tiempo += tiempos[i%n];
        }
        return tiempo;
    }
```

3.2 Algoritmo realista

Las tareas pueden estar en cualquier orden. Para la ordenación utilizaremos la ordenación por inserción por índices, de manera que los datos de entrada no se vean modificados.

```
   public static int tareas2 (int[] ts, int n) {

        /* Realizamos la ordenacion */
        int[] ts2 = new int[ts.length];
        ts2[0] = 0;
        for (int i = 1; i < ts.length; i++) {
            int aux = ts[i];
            int j;
            for (j = i-1; j >= 0  && ts[ts2[j]] > aux; j--)
                ts2[j+1] = ts2[j];
            ts2[j+1] = i;
        }

        /* Algoritmo voraz */
        int tiempo = 0;
        int[] tiempos = new int[n];
        for (int i = 0; i < ts2.length; i++) {
            tiempos[i%n] += ts[ts2[i]];
            tiempo += tiempos[i%n];
        }
        return tiempo;
    }
```

4 Análisis de complejidad

El análisis de complejidad en tiempo del algoritmo realista es el siguiente: [1].

$$
\begin{aligned}
T(n) &= 1+2+1+\sum_{i=1}^{n}\Big(3+2+2+\sum_{j=i-1}^{0}\big(6+4\big)+3\Big)+2+1+\sum_{i=0}^{n}\big(3+5\big)+1 = \\
&= 4+\sum_{i=1}^{n}\Big(7+\sum_{j=0}^{i-1}\big(10\big)+3\Big)+3+\sum_{i=0}^{n}\big(8\big)+1 = \\
&= 4+\sum_{i=1}^{n}\Big(7+\sum_{j=1}^{i}\big(10\big)+3\Big)+3+\sum_{i=0}^{n}\big(8\big)+1 = \\
&= 4+\sum_{i=1}^{n}\Big(7+10i+3\Big)+3+\sum_{i=0}^{n}\big(8\big)+1 = \\
&= 4+\sum_{i=1}^{n}\big(7+3\big)+\sum_{i=1}^{n}\big(10i\big)+3+\sum_{i=0}^{n}\big(8\big)+1 = \\
&= 4+10n+10\left(\frac{n\,(n+1)}{2}\right)+3+8n+1 = \\
&= 4+10n+10\left(\frac{n^2+n}{2}\right)+8n+4 = \\
&= 18n+5n^2+5n+18 = \\
&= 5n^2+23n+18 = O(n^2)
\end{aligned}
$$

donde n es el tamaño del array *ts*, es decir, el número de tareas.

5 Conclusión

El tiempo total de espera es menor cuanto menor es el número de actividades y mayor es el número de procesadores, ya que de este modo es menor la acumulación de tiempos de espera.

En cuanto al algoritmo, el tiempo de ejecución varía mucho dependiendo de si las condiciones son reales o ideales. En condiciones ideales el algoritmo tiene coste lineal, pero en condiciones reales se le suma el tiempo de la ordenación, más costosa que el resto del algoritmo. Es por ello que el coste del algoritmo realista es cuadrático y depende del número de tareas.

[1] Las constantes numéricas corresponden a operaciones elementales. Por ejemplo, la instrucción 'ts2[j+1] = ts2[j]' tiene coste 4 porque se realiza un incremento, dos accesos a posición de array y una asignación