Aplicación del algoritmo Greedy - Electricistas

Algorítmica

Doble Grado en Ingeniería Informática y Matemáticas

Yábir García Benchakhtir
yabirgb@correo.ugr.es

18 de abril de 2018

Índice

1. Descripción del problema

Tenemos un conjunto de tareas P y un electricista. Cada tarea $P[i]$ tiene asignado un tiempo p_i que, a efectos de implementación, será un entero que indicará los segundos necesarios para completarla. El dinero que cobra el electricista por cada tarea completada es inversamente proporcional al tiempo de espera del cliente. ¿Cuál es la distribución temporal de las tareas que garantiza la máxima ganancia?

2. Solución del problema

En este problema maximizar las ganancias es equivalente al problema de minimizar el tiempo de espera.

La suma de los tiempos de espera de cada cliente es:

$$T(P) = p_0 + (p_0 + p_1) + (p_0 + p_1 + p_2) \dots \tag{1}$$
$$= np_0 + (n-1)p_1 + \dots p_n \tag{2}$$
$$= \sum_{k=0}^{n-1} (n-k)p_k \tag{3}$$

Es razonable pensar que el tiempo de espera se va a minimizar si ordenamos las tareas en orden no decreciente de tiempo. De esta manera estamos consiguiendo minimazar el tiempo de espera para las primeras tareas y las que requieren más tiempo no penalizan al resto.

3. Algoritmo propuesto

Para solucionar este problema se propone como algoritmo el siguiente:

Algorithm 1 Asignación de tareas

```
tasks = [t_1, t_2, ..., t_n]
schedule = []
procedure DISTRIBUTION
    schedule = sort(tasks)
    return schedule
end procedure
```

3.1. ¿Es nuestro algoritmo greedy?

Si analizamos un algoritmo greedy vemos que estos constan de tres partes diferenciadas:

- Un conjunto de candidatos.
- Una lista de candidatos ya usados.
- Un criterio (función) solución que dice cuándo un conjunto de candidatos forma una solución (no necesariamente óptima).
- Criterio para dictar cuando una solución es válida.
- Una función de selección que indique el mejor candidato.

- Una función objetivo que asocie a cada solución un valor.

En este caso el conjunto de candidatos es el conjunto de todas las tareas.

La lista de candidatos ya usados es la lista que mantenemos con las tareas en el orden que las va a realizar el electricista.

En este caso, trabajando con enteros, podemos realizar en un tiempo $nlog(n)$ ya que comparamos en tiempo constante.

Para nosotros todos los subconjuntos de soluciones son válidas. La función que nos permita elegir el mejor resultado cada vez es sencillamente la función

$$\lambda(P) = P[0]$$

Y la función objetivo simplemente es el cálculo del valor que queremos minimizar.

4. Demostración de la eficiencia de este algoritmo

Supongamos que P es una distribución no ordenada en función de tiempo de las tareas y además óptima.

P' es una permutación de P donde hemos cambiado las posiciones l y m tal que $l < m$ y $p[l] < p[m]$. Esta permutación es distinta de la anterior ya que habíamos supuesto que no estaba ordenado en orden creciente de tiempos. De esta manera:

$$T(P') = (n-l)p_l + (n-m)p_m + \sum_{k=0, k\notin\{l,m\}}^{n-1} (n-k)p_k$$
$$T(P) = (n-m)p_l + (n-l)p_m + \sum_{k=0, k\notin\{l,m\}}^{n-1} (n-k)p_k$$

Por hipótesis $T(P) < T(P')$ por tanto:

$$\begin{aligned}(n-l)p_l + (n-m)p_m &> (n-m)p_l + (n-l)p_m \\ (n-l-n+m)p_l &> p_m(n-l+m+n) \\ (m-l)p_l &> p_m(m-l) \\ (m-l)(p_l - p_m) &> 0\end{aligned}$$

Lo cual es absurdo pues por hipótesis $p_l < p_m$.

5. Caso de tener múltiples electricistas

En el caso de tener múltiples electricistas el problema reside en como asignar el reparto de tareas a cada electricista de modo que una vez hecho el reparto vamos a aplicar el algoritmo anteriormente descrito a las tareas de cada electricista.

Supongamos que tenemos k electricistas. Lo primero que tenemos que hacer es ordenar el vector de tareas, esto es igual a lo que haciamos en el caso de tener un solo electricista.

Una vez tenemos el vector ordenado repartimos a los k elecetricistas las primeras k tareas de manera que tenemos el vector $P[k:]$ (subdivisión de P a partir de la posición k) y volvemos a repartir de igual forma hasta quedarnos sin tareas.

De este manera mantenemos una media de tiempo similar para cada electricista y nos estamos garantizando que las tareas se realizan en el menor tiempo posible una vez que han sido asignadas a un electricista.

Una implementación con pseudo-código sería la siguiente:

Algorithm 2 Asignación de tareas

```
workers = [w_1, w_2, ...w_k]
tasks = sorted([t_1, t_2, ...t_n])
procedure DISTRIBUTION
    assignations = {}
    currentWorker = 0
    for task in tasks do
        assignations[workers[currentWorker]] += task
        currentWorker = (currentWorker + 1) %k
    end for
    return assignations
end procedure
```

En el caso de que la cantidad de tareas fuese menor que el número de trabajadores disponibles el anterior algoritmo sigue siendo válido ya que si consideramos n el número de tareas basta aplicar el algoritmo con las sección de trabajadores $W[:n]$ (lista de los trabajadores hasta la posición n).