Algoritmo Greedy: Problema de los electricistas

Yábir G. Benchakhtir

18 de abril de 2018

Doble Grado en Ingeniería Informática y Matemáticas

Descripción del problema

Tenemos un conjunto de tareas T y un electricista. Cada tarea $T[i]$ tiene asignado un tiempo t_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?

Candidato a algoritmo

El tiempo de espera total es el siguiente:

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

Y queremos minimizar este tiempo.

Algoritmo propuesto

Algorithm 1 Asignación de tareas

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

Por qué este algoritmo es el mejor

Suponemos P un vector no ordenado ascendentemente cuya solución es óptima y P' el mismo vector P sobre el que se ha realizado una permutación consistente en hacer que para $l < m$ $p_l < p_m$. Esto es posible ya que no está totalmente ordenado.

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

$$T(P) = (n-m)p_l + (n-l)p_m + \sum_{k=0, k\notin\{l,m\}}^{n-1} (n-k)p_k$$

Como $T(P') > T(P)$

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

¿Es nuestro algoritmo greedy?

- **Un conjunto de candidatos.**
  En nuestro caso los candidatos son las tareas que tenemos que completar (P).

¿Es nuestro algoritmo greedy?

- **Un conjunto de candidatos.**
  En nuestro caso los candidatos son las tareas que tenemos que completar (P).
- **Una lista de candidatos ya usados.**
  La lista de candidatos usados se resume a la lista de tareas que ya están asignadas.

¿Es nuestro algoritmo greedy?

- **Un conjunto de candidatos.**
  En nuestro caso los candidatos son las tareas que tenemos que completar (P).
- **Una lista de candidatos ya usados.**
  La lista de candidatos usados se resume a la lista de tareas que ya están asignadas.
- **Función para determinar si nuestra solución es candidata a pertenecer al conjunto de soluciones válidas.**
  Nuestras soluciones siempre van a ser válidas aunque hay soluciones que son mejores que otras.

¿Se puede resolver nuestro problema con un algoritmo greedy?

- **Una función de selección que indique el mejor candidato.** En nuestro algoritmo siempre vamos eligiendo la tarea que se encuentra en nuestro vector de candidatos en la primera posición. Nuestra función de selección es sencillamente

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

¿Se puede resolver nuestro problema con un algoritmo greedy?

- **Una función de selección que indique el mejor candidato.**
  En nuestro algoritmo siempre vamos eligiendo la tarea que se encuentra en nuestro vector de candidatos en la primera posición. Nuestra función de selección es sencillamente

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

- **Una función objetivo.**
  Nuestra función objetivo es la que asigna a cada solución el tiempo de espera total de los clientes y que ya se ha mostrado:

$$T(P) = \sum_{k=0}^{n-1} (n-k)p_k$$

Que ocurre si tenemos varios electricistas

Nuestro principal problema es como realizar el reparto de tareas a cada trabajador.

Que ocurre si tenemos varios electricistas

Nuestro principal problema es como realizar el reparto de tareas a cada trabajador.

Una vez repartidas siempre usamos el algoritmo descrito anteriormente.

Algoritmo

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

Algoritmo

En esta solución aplicamos una idea semejante a la anterior. Partimos también de un vector ordenado y además estamos repartiendo las tareas de manera que la media de tiempos de cada trabajador tiende a ser la misma.

Ejemplo

$$P = [9, 8, 3, 6, 1, 7, 5] w = 3$$

Ejemplo

$$P = [9, 8, 3, 6, 1, 7, 5] w = 3$$

$$sorted(P) = [1, 3, 5, 6, 7, 8, 9]$$

Ejemplo

$$P = [9, 8, 3, 6, 1, 7, 5] w = 3$$

$$sorted(P) = [1, 3, 5, 6, 7, 8, 9]$$

$$w_1 = [1, 6, 9] \quad w_2 = [3, 7] \quad w_3 = [5, 8]$$

Ejemplo

$$P = [9, 8, 3, 6, 1, 7, 5] w = 3$$

$$sorted(P) = [1, 3, 5, 6, 7, 8, 9]$$

$$w_1 = [1, 6, 9] \quad w_2 = [3, 7] \quad w_3 = [5, 8]$$

$$T(w_1) = 1+7+16 = 24 \quad T(w_2) = 3+10 = 13 \quad T(w_3) = 5+13 = 18$$