Algoritmos Greedy y Búsqueda Binaria

Leopoldo Taravilse

Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

Training Camp 2013

Contenidos

Contenidos

Que es un algoritmo greedy?

Que es un algoritmo greedy?

Un algoritmo greedy (conocido en español como algoritmo goloso) es un algoritmo que va seleccionando por pasos parte de la solución, encontrando óptimos locales, hasta armar una solución óptima para un problema.

Activity Selection Problem

Problema

Juan tiene *n* actividades que realizar, y sabe cuándo empieza y cuándo termina cada una. Lamentablemente algunas se superponen y por lo tanto no puede realizarlas todas. El problema pide encontrar la máxima cantidad de actividades que Juan puede realizar sin superposiciones.

Activity Selection Problem

Problema

Juan tiene *n* actividades que realizar, y sabe cuándo empieza y cuándo termina cada una. Lamentablemente algunas se superponen y por lo tanto no puede realizarlas todas. El problema pide encontrar la máxima cantidad de actividades que Juan puede realizar sin superposiciones.

Este problema tiene una solución golosa. Ordenamos las actividades por horario de finalización y vamos recorriendolas de inicio a fin. Cada vez que encontramos una actividad que empieze después del momento en el que finaliza la actividad actual, la agregamos como actividad actual que realiza Juan.

Códigos de Huffman

Uno de los algoritmos de compresión más comunes es el de Huffman. Cada caracter es codificado con un string binario, generando códigos libres de prefijos (ningún código es prefijo de otro código) de modo tal de que un texto dado contenga la menor cantidad de bits posibles.

Códigos de Huffman

Uno de los algoritmos de compresión más comunes es el de Huffman. Cada caracter es codificado con un string binario, generando códigos libres de prefijos (ningún código es prefijo de otro código) de modo tal de que un texto dado contenga la menor cantidad de bits posibles. Para visualizar la solución es conveniente ver los códigos de Huffman en un árbol binario (el árbol de Huffman). Las hojas son los caracteres y los caminos a la raiz son los códigos. En cada paso tomamos los dos nodos con menos frecuencia (la frecuencia de todos los caracteres del subárbol sumada), sumamos su frecuencia y unimos ambos nodos con un nodo padre.

Add All

Se tienen n números y se quiere calcular la suma de todos ellos. Sumar dos números a y b cuesta $a + b$. Si por ejemplo queremos sumar 1+2+3 lo podemos hacer como (1+2)+3 con un costo de 3 + 6 = 9, y si queremos sumar 2+3+5+8 lo podemos hacer como (2+8)+(3+5) con un costo de 10+8+18 = 36 o con un costo de ((2+3)+5)+8 con un costo de 5+10+18 = 33. Cuál es el mínimo costo de sumar los n números.

Add All

Se tienen n números y se quiere calcular la suma de todos ellos. Sumar dos números a y b cuesta $a + b$. Si por ejemplo queremos sumar 1+2+3 lo podemos hacer como (1+2)+3 con un costo de 3 + 6 = 9, y si queremos sumar 2+3+5+8 lo podemos hacer como (2+8)+(3+5) con un costo de 10+8+18 = 36 o con un costo de ((2+3)+5)+8 con un costo de 5+10+18 = 33. Cuál es el mínimo costo de sumar los n números.
La solución es golosa, consiste en cada paso en tomar los dos más chicos y sumarlos, reemplazandolos por su suma.

Contenidos

Ceros de funciones monótonas

Supongamos que tenemos una función monótona creciente o decreciente, y queremos encontrar un cero de esa función. Si sabemos un punto donde la función toma un valor positivo y un punto donde toma un valor negativo, entonces podemos aplicar la técnica de búsqueda binaria.

```
1  double mx = 10000, mn = -10000;
2  // f(mx) > 0,  f(mn) < 0
3  while(mx - mn >1e-9)
4  {
5      double med = (mx + mn)/2;
6      if(f(med)>0)
7          mx = med;
8      else
9          mn = med;
10 }
11 return mx; // o return mn;
```

Búsqueda Binaria en enteros

También podemos hacer búsqueda binaria con números enteros.

Búsqueda Binaria en enteros

También podemos hacer búsqueda binaria con números enteros. Dado un tablero con ceros y unos queremos encontrar el mayor cuadrado de unos (Maximum Square).

Búsqueda Binaria en enteros

También podemos hacer búsqueda binaria con números enteros.
Dado un tablero con ceros y unos queremos encontrar el mayor cuadrado de unos (Maximum Square).
Precomputamos las sumas parciales en dos dimensiones para poder obtener la suma de un cuadrado en o(1), y hacemos búsqueda binaria en la solución.

Maximum Square

Calculamos $F(a, b, c, d)$ como la suma del rectángulo que va de (a, b) (superior izquierdo) a (c, d) (inferior derecho).

Maximum Square

Calculamos $F(a, b, c, d)$ como la suma del rectángulo que va de (a, b) (superior izquierdo) a (c, d) (inferior derecho).

$$F(0,0,0,0) = M[0][0]$$

$$F(0,0,0,i) = M[0][i] + F(0,0,0,i-1)$$

$$F(0,0,i,0) = M[i][0] + F(0,0,i-1,0)$$

$$F(0,0,i,j) = M[i][j] + F(0,0,i-1,j) + F(0,0,i,j-1) - F(0,0,i-1,j-1)$$

Maximum Square

Calculamos $F(a, b, c, d)$ como la suma del rectángulo que va de (a, b) (superior izquierdo) a (c, d) (inferior derecho).

$$F(0,0,0,0) = M[0][0]$$

$$F(0,0,0,i) = M[0][i] + F(0,0,0,i-1)$$

$$F(0,0,i,0) = M[i][0] + F(0,0,i-1,0)$$

$$F(0,0,i,j) = M[i][j] + F(0,0,i-1,j) + F(0,0,i,j-1) - F(0,0,i-1,j-1)$$

Una vez precomputados estos valores tenemos

$$F(a,b,c,d) = F(0,0,c,d) - F(0,0,a-1,d)$$

$$-F(0,0,c,b-1) + F(0,0,a-1,b-1)$$

Teniendo cuidado con los casos $a = 0$ y $b = 0$.

Maximum Square

Teniendo la posibilidad de calcular cada cuadrado en $O(1)$ y con la función *cuad*(*int n*) que nos dice si hay un cuadrado de $n \times n$, el código sería

```
1  int mx = min(n,m)+1, mn = 0;
2  while(mx - mn > 1)
3  {
4      int med = (mx + mn)/2;
5      if(cuad(med)==true)
6          mn = med;
7      else
8          mx = med;
9  }
10 return mn;
```