Programación Dinámica

El problema de los puestos de bondiola

Algoritmos y Estructuras de Datos III
Departamento de Computación
Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires
4 de Abril, 2017

Enunciado

- Tenemos n posibles lugares donde ubicar puestos de venta de bondiola.
- El lugar i está a distancia d_i a la derecha del 0 (Los d_i están en orden creciente).
- Instalar un puesto en el lugar i nos da b_i de beneficio.
- No podemos instalar dos puestos a distancia menor a m.
- Podemos instalar la cantidad de puestos que queramos.

¿Cuál es el mejor beneficio que podemos obtener?

- Queremos dar un algoritmo de complejidad $O(n^2)$ que este basado en programación dinámica.

Definir los subproblemas

Vamos a ver como resolver este problema usando programación dinámica.
Paso 1: elegir cuáles van a ser nuestros subproblemas.
En problemas donde hay un vector de elementos hay dos grandes patrones que suelen funcionar:

- Mirar los prefijos o sufijos del vector (los primeros i o los últimos i).
- Mirar los subintervalos del vector (los elementos entre la posición i y la j).

(Des)Ventajas de cada uno:

- Prefijos: Menos subproblemas, solución más rápida.
- Subintervalos: Subproblemas mas flexibles, hay más problemas donde podemos usar este patrón.

Definiendo los subproblemas

En este problema vamos a usar los prefijos.
Definimos los subproblemas en lenguaje natural:
$S_i :=$ "El mayor beneficio que puedo obtener instalando puestos sólo en los primeros i lugares".
Lo definimos para todo $0 \leq i \leq n$.

Obteniendo la solución

En este problema vamos a usar los prefijos.
Definimos los subproblemas en lenguaje natural:
$S_i :=$ "El mayor beneficio que puedo obtener instalando puestos sólo en los primeros i lugares".
Lo definimos para todo $0 \leq i \leq n$.

Paso 2: ¿Cómo obtengo la respuesta al problema original a partir de los subproblemas?
Es simplemente S_n (no siempre es tan sencillo).

Reducir los subproblemas

Paso 3: Decidir como reducir los subproblemas.
Para esto se suele elegir una parte de la solución y probar todas las posibilidades de resolver esta parte.

- Es importante que lo que nos quede luego de resolver esa parte este relacionado con subproblemas más chicos.
- Conviene que la parte a probar sea lo más chica posible.

Reduciendo los subproblemas

En este problema para reducir S_i $(i > 0)$, decidimos que hacer con el último lugar (el $i-1$-ésimo).
Tenemos dos posibilidades (instalar un puesto o no):

- Si no colocamos un puesto, lo que nos queda resolver es S_{i-1}.
- Si colocamos un puesto, sea j el menor índice menor o igual a $i-1$ tal que $d_{i-1} - d_j < m$, lo que nos queda resolver es S_j.

La respuesta del subproblema es lo mejor de estas dos opciones.

Escribir la recursión

Escribimos la recursión que quedó (no olvidar los casos base):
Si $i > 0$, definimos $j_i = \text{mín}\{k : 0 \leq k < i, d_{i-1} - d_k < m\}$.
En este caso, $S_i = \text{máx}(S_{i-1}, b_{i-1} + S_{j_i})$.
Caso base: $S_0 = 0$.

Código top-down

```
int resolver(int i) {
        if(i == 0) return 0;
        if(dp[i] != -1) return dp[i];

        int res = 0, j = i-1;
        while(j > 0 && d[j-1] - d[i-1] < m) j--;
        res = max(resolver(i-1), b[i-1] + resolver(j));
        dp[i] = res;
        return res;
}
```

Complejidad

La complejidad se puede calcular como la suma de cuanto tardamos en resolver todos los subproblemas (tomando las llamadas recursivas como $O(1)$).

Si tomamos el peor tiempo de todos los subproblemas podemos acotar la complejidad por:
Peor tiempo de un subproblema * Cantidad de subproblemas.

En este caso en cada subproblemas tardamos $O(n)$ y tenemos $O(n)$ subproblemas.
Luego la complejidad es $O(n * n) = O(n^2)$.

Correctitud

Hay que probar que la fórmula recursiva es correcta.

- Probamos que el cálculo de S_i es correcto por inducción en i.
- Si suponemos que hay algo mejor a S_i, viene de alguno de los dos casos, pero en ambos casos por HI como mucho tenemos lo que dice la fórmula.
- Podemos construir una solución que da S_i a partir de las de los i menores.

Además, hay que ver que lo que propusimos como respuesta al problema original efectivamente lo es (en este caso es por definición).

Ejercicios

Para pensar:

- Dar una implementación bottom-up de la solución. ¿En qué orden resolvemos los subproblemas?
- ¿Cómo puedo optimizar el cálculo de cada j? ¿Mejora esto la complejidad de la solución?
- (Difícil) Pensar como calcular todos los j_i en $O(n)$ ($O(1)$ amortizado), implementar una solución $O(n)$ usando esto.

Pasos para usar Programación dinámica

Para resolver un problema con programación dinámica podemos realizar los siguientes pasos:

1. Definir cuales serán nuestros subproblemas.
2. Entender cómo se resuelve el problema original a partir de los subproblemas.
3. Elegir una parte del subproblema y probar todos los casos de resolver esa parte.
4. Relacionar lo que me falta de la solución con los subproblemas más chicos.
5. Escribir la recursión.
6. Implementar y calcular complejidad.

¡Fin!

¿Preguntas?