Juan Carlos Llamas Núñez 3º DG Inf-Mat

Ejercicio 17.- Durante la fiesta de Halloween, y con el fin de recaudar dinero para el viaje de fin de curso, Alicia tiene P prendas disponibles para vender a sus n vecinos para que se confecciones sus disfraces. Puede vender a cada uno de ellos un número variable k de prendas con $0 \leq k \leq h$ $\forall i \in \{1 \cdots n\}$ y cierta constante h dada. Además, Alicia sabe que obtendría un beneficio b_{ik} por venderle k prendas al vecino i con $0 \leq k \leq h$ e $i \in \{1 \cdots n\}$.

Implementar un algoritmo que nos diga cuántas prendas habrá de vender Alicia a cada uno de sus vecinos de forma que el beneficio total obtenido sea máximo. Indicar justificadamente su coste en tiempo y en espacio.

En primer lugar definimos

beneficio$(i,j) \equiv$ máximo benificio que se puede obtener vendiendo j prendas a los vecinos $1, 2, \cdots i$. Entonces nos interesa obtener beneficio(n, P).

La definición recursiva es:

$$\text{beneficio}(i,j) = \max_{\substack{0 \leq k \leq h \\ \&\& \\ k \leq j}} \{ \text{beneficio}(i-1, j-k) + b_{ik} \} \qquad \forall i,j \text{ con } 1 \leq i \leq n,\ 0 \leq j \leq P$$

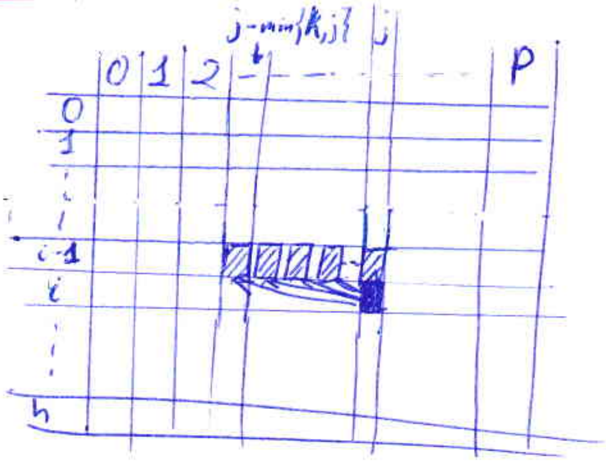
Como al vecino i le podemos vender entre 0 y h prendas y además tenemos que tener suficientes $(k \leq j)$ calculamos el máximo de venderle k prendas a i y $j-k$ a los otros.

Los casos base son:

- $beneficio(0, j) = 0 \quad \forall j$ con $0 \le j \le P$ (Si no quedan vecinos a los que vender, el beneficio es 0)

(Si no quedan prendas por vender, el beneficio puede ser mayor que 0)

Tabla



Código:

```
func maxBeneficio(vector<vector<int>> b[1..n][0..h]) dev
        int maxben, vector<vector<int>> matriz [0..n][0..P] {     vector<vector<int>> decisiones

    matriz[0][0..P] = 0

    decisiones[0..n][0..P]
    for (int i = 1; i <= n; i++) {
        for (int j = 0; j <= P; j++) {
            int k_escogida
            int maximo = 0;
            for (int k = 0; k <= min{j, h}; k++) {
                maximo = max{maximo, matriz[i-1][j-k] + b[i][k]};
                    if (maximo < matriz[i-1][j-k] + b[i][k]) {
                        maximo = matriz[i-1][j-k] + b[i][k];
                        k_escogida = k
                    }
            }
            matriz[i][j] = maximo;     decisiones[i][j] = k_escogida
        }
    }
    maxben = matriz[n][P];
}
```

Obs En color rojo se añaden las modificaciones de escoger la implementación con matriz de decisiones

$O(nhP)$ en tiempo y $O(nP)$ en espacio

Reconstrucción de la solución:

```
func reconstruirSolucion (vector<vector<int>> matriz[0..n][0..P],
vector<vector<int>> b[1..n][0..h]) dev vector<int> solución[1..n]{

int j = P;
for (int i=n, i>0; i--){
   int k=0;
   while (k ≤ min{j, h} && matriz[i][j] != matriz[i-1][j-k]+b[i][k])
   { k++};
   sol[i]=k;
   j-= k;
}
}
```

O(n+P) en tiempo y O(1) en espacio

P puede ser exponencialmente grande.

Podemos guardar en su lugar la matriz de decisiones.

Enlonces

```
func reconstruirSolucion bis (
vector<vector<int>> decisiones[1..n][0..P]) dev vector<int> solucion[1..n]{
int j=P;
 for (int i=n; i>0; i--){
  sol[i]= decisiones[i][j];
  j-= decisiones[i][j];
 }
}
```

O(n) en tiempo

Observaciones sobre el coste de la primera de las reconstrucciones de la solución.

Dado el valor beneficio(i,j) queremos conocer el número k de prendas que se le vende al vecino i. Sabemos, por la definición recursiva, que beneficio(i,j) = $\max_{\substack{0\leq k\leq h \\ \& \\ k\leq j}} \{\text{beneficio}(i-1, j-k) + b_{ik}\}$

donde b_{ik} es también conocido.

Para encontrar k, que es el número de prendas vendidas al vecino i, basta comparar beneficio(i,j) con beneficio(i-1, j-k)+b_{ik} haciendo variar k, en principio entre 0 y min{h,j}, pero basta encontrar el mínimo k que verifica la igualdad. Si denotamos a este k como k_i para cada i se sigue que en realidad hacemos $1+k_i$ comparaciones y este k_i coincide con sol[i], es decir, el número de prendas que se vende al vecino i. Para el vecino i-1 ya solo quedan $j-k_i$ prendas por vender por lo que la disminución de j no es, en general, de uno en uno. ¿Porqué podemos parar de comparar cuando encontramos un k que verifica que beneficio(i,j) = beneficio(i-1, j-k)+b_{ik} y no comparar los 1+min{j,h} k's posibles? Como solo necesitamos una forma de vender las prendas de tal manera que el beneficio sea máximo, esta forma es suficiente. Por tanto, el número de comparaciones es:

$$\sum_{i=1}^{n}(k_i+1) = n + \sum_{i=1}^{n} k_i \leq n + P$$

porque la suma de todas las prendas vendidas a cada vecino es menor que el número total de prendas. Ponemos un ejemplo de ejecución de cada modo de reconstruir la solución:

Sean $n=4$, $h=5$, $P=6$ y B la matriz que en la posición (i,j) tiene el beneficio de vender j prendas al vecino i

$$B = \begin{pmatrix} 0 & 2 & 4 & 6 & 8 & 9 \\ 0 & 1 & 5 & 6 & 7 & 7 \\ 0 & 3 & 6 & 6 & 6 & 6 \\ 0 & 2 & 4 & 5 & 5 & 5 \end{pmatrix}$$

Matriz de max beneficio

(matriz[0..n][0..P])

	0	1	2	3	4	5	6
0	0	0	0	0	0	0	0
1	0	2	4	6	8	9	9
2	0	2	5	7	9	11	13
3	0	3	6	8	11	13	15
4	0	3	6	8	11	13	15

$15-0=15$ $(i=4) \rightarrow sol[4]=0$

$15-0\neq 13$ $(i=3)$

$15-3\neq 11$ $(i=3)$

$15-6=9$ $(i=3) \rightarrow sol[3]=2$

$9-0\neq 8$ $(i=2)$

$9-1\neq 6$ $(i=2)$

$9-5=4$ $(i=2) \rightarrow sol[2]=2$

$4-0\neq 0$ $(i=1)$

$4-2\neq 0$ $(i=1)$

$4-4=0$ $(i=1) \rightarrow sol[1]=2$

Matriz de decisiones

(decisiones[1..n][0..P])

	0	1	2	3	4	5	6
0	—	—	—	—	—	—	—
1	0	1	2	3	4	5	5
2	0	0	2	2	2	2	2
3	0	1	2	1	2	2	2
4	0	0	0	0	0	0	0

$j=6\ (=P)$ $\quad i \quad j$

sol[4] = decisiones[4][6] = 0 ; j = 6-0

sol[3] = decisiones[3][6] = 2 ; j = 6-2 = 4

sol[2] = decisiones[2][4] = 2 ; j = 4-2 = 2

sol[1] = decisiones[1][2] = 2