Hoja 2: Ejercicio 11

Ramificación y poda

Diego Rodríguez Cubero

UCM

24 de febrero de 2026

Contenidos

1. Enunciado del problema
2. Idea del ejercicio
3. Espacio de soluciones
4. Cota optimista
5. Cota pesimista
6. Implementación del algoritmo
7. Costes en tiempo y espacio

Enunciado del problema

Ejercicio 11

Alonso Rodríguez tiene que hacer la compra de la semana consistente en n productos. En su barrio hay m supermercados, cada uno de los cuales dispone de todos esos productos. Pero no quiere comprar más de tres productos en cada uno de los supermercados, ya que así pasa más tiempo comprando (se puede suponer que $n \leq 3m$). Dispone de una lista de precios de los productos en cada uno de los supermercados. Escribir un algoritmo de ramificación y poda, aplicando el esquema optimista-pesimista en caso de ser posible, que le permita decidir cómo hacer la compra de forma que el coste total sea mínimo.

Idea del ejercicio

El ejercicio se basa en ir asignando productos a supermercados de forma recursiva, generando un árbol de decisiones. Con la restricción de que no se pueden asignar más de tres productos a cada supermercado, se deben explorar las diferentes combinaciones posibles y utilizar técnicas de poda para evitar explorar ramas que no puedan conducir a una solución óptima, la cual debe ser minima en coste total.

Espacio de soluciones

El espacio de soluciones se puede escribir como un vector:

$$(s_1, s_2, \ldots, s_n)$$

Donde $s_i = j$ representa que el producto i se compra en el supermercado j. La restricción de que no se pueden comprar más de tres productos en cada supermercado se puede expresar como:

$$\sum_{i=1}^{n} \delta(s_i, j) \leq 3 \quad \forall j \in \{1, 2, \ldots, m\}$$

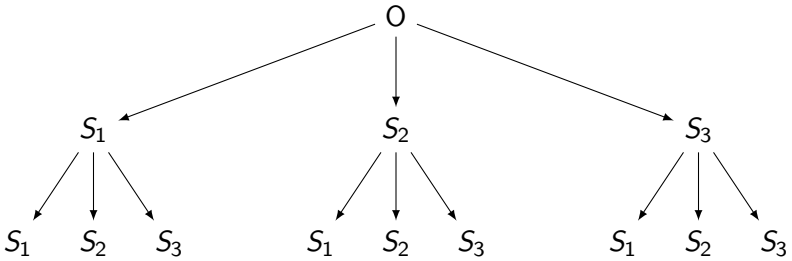
Donde $\delta(s_i, j)$ es una función indicadora que vale 1 si $s_i = j$ y 0 en caso contrario. El objetivo es minimizar el coste total, que se puede expresar como:

$$\text{Coste total} = \sum_{i=1}^{n} P(i, s_i)$$

Donde $P(i, j)$ es el precio del producto i en el supermercado j.

Arbol de decisiones para $n = 2$ y $m = 3$

El siguiente árbol de decisiones muestra cómo se asignan los productos a los supermercados, donde cada arista representa la asignación de un producto a un supermercado, aunque no comprueba la reestriccion de que no haya mas de 3 productos por supermercado. En este caso, con $n = 2$ productos y $m = 3$ supermercados, el árbol tendría la siguiente estructura:



Cota optimista

Para calcular la cota optimista, podemos calcular el coste mínimo de forma dinámica asignando cada producto al supermercado más barato:

```
int cotaOptimista(vector<vector<int>>& precios, vector<int>& asignacion, int
    p_actual) {
    int costeTotal = 0;
    for (int i = p_actual; i < precios.size(); i++) {
        int minPrecio = INT_MAX;
        for (int j = 0; j < precios[i].size(); j++) {
            if (asignacion[j] < 3) { // Verificar que no se han asignado más de
                3 productos al supermercado j
                minPrecio = min(minPrecio, precios[i][j]);
            }
        }
        costeTotal += minPrecio;
    }
    return costeTotal;
}
```

El coste de esta función es $O(n \cdot m)$, ya que recorre los productos restantes y para cada producto recorre los supermercados para encontrar el precio mínimo.

Cota pesimista

Para calcular la cota pesimista, podemos realizar un proceso análogo al de la cota optimista, pero en lugar de asignar cada producto al supermercado más barato, asignamos cada producto al supermercado más caro:

```
int cotaPesimista(vector<vector<int>>& precios, vector<int>& asignacion, int
    p_actual) {
    int costeTotal = 0;
    for (int i = p_actual; i < precios.size(); i++) {
        int maxPrecio = INT_MIN;
        for (int j = 0; j < precios[i].size(); j++) {
            if (asignacion[j] < 3) { // Verificar que no se han asignado más de
                3 productos al supermercado j
                maxPrecio = max(maxPrecio, precios[i][j]);
            }
        }
        costeTotal += maxPrecio;
    }
    return costeTotal;
}
```

El coste de esta función también es $O(n \cdot m)$, ya que recorre los productos restantes y para

Implementación del algoritmo

```
void ramificacionYPoda(vector<vector<int>>& precios, vector<int>& asignacion,
    int p_actual, int costeActual, int& minCos, vector<int>& minAsig) {
    if (p_actual == precios.size() && costeActual < minCos) {
        minCos = costeActual;
        minAsig = asignacion;
    }
    else {
        for (int j = 0; j < precios[p_actual].size(); j++) {
            if (asignacion[j] < 3) { // 3 productos por supermercado
                asignacion[j]++;
                int nuevoCoste = costeActual + precios[p_actual][j];
                int cotaOpt = nuevoCoste + cotaOptimista(precios, asignacion,
                    p_actual + 1);
                if (cotaOpt < minCos) { // Poda
                    ramificacionYPoda(precios, asignacion, p_actual + 1,
                        nuevoCoste, minCos, minAsig);
                }
                asignacion[j]--; // Quitamos el producto
            }
        }
    }
}
```

Costes en tiempo y espacio

El coste en tiempo del algoritmo en el caso peor es $O(m * n * m^n)$, ya que en el peor caso se exploran todas las combinaciones posibles de asignación de productos a supermercados, y para cada asignación se calcula la cota optimista y pesimista. Sin embargo, gracias a la poda, el número de combinaciones exploradas puede ser significativamente menor en la práctica.
El coste en espacio es $O(n)$ para almacenar la asignación actual de productos a supermercados, y $O(n)$ para almacenar la mejor asignación encontrada, lo que da un coste total de $O(n)$ en espacio.