Hoja 2: Ejercicio 10

Ramificación y poda

Diego Rodríguez Cubero

UCM

24 de febrero de 2026

Contenidos

Enunciado del problema

Ejercicio 10

Una de las pruebas habituales del concurso Supervivientes es la construcción de una cabaña rudimentaria cuyo techo es una simple lona soportada por cuatro pilares. Los concursantes dispondrán de n de fragmentos de caña de longitudes enteras $l_1, \cdots, l_n$ ensamblando los cuales deberán obtener los cuatro pilares. El objetivo es que sus alturas queden razonablemente equilibradas y sean lo más altas posibles. Precisando, desarrollar el problema buscando maximizar el pilar más bajo de los cuatro.

Idea general

Para resolver este problema, podemos utilizar una técnica de ramificación y poda. La idea es generar todas las posibles combinaciones de los fragmentos de caña para formar los cuatro pilares, y luego podar aquellas combinaciones que no cumplan con el objetivo de maximizar el pilar más bajo.

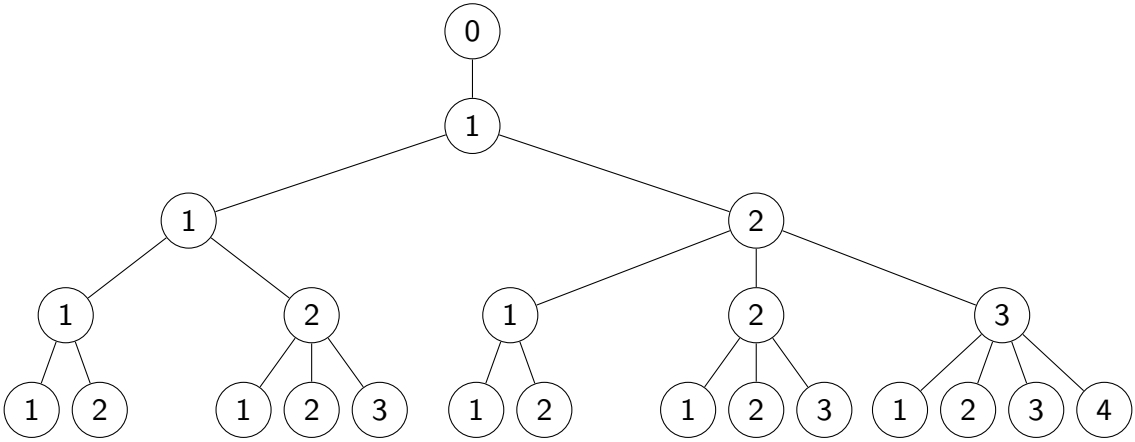
Espacio de soluciones

El espacio de soluciones se puede representar como un vector:

$$(x_1, \cdots, x_n) : \qquad x_i \in \{1, 2, 3, 4\} \quad \forall i \in \{1, \cdots, n\}$$

donde x_i representa la asignación del fragmento de caña l_i a uno de los cuatro pilares. En forma de arbol se representaría de la siguiente manera, donde cada arista de la altura i a un nodo de valor k significa asignar $x_i = k$.

Árbol



Cota optimista

Para calcular la cota optimista, podemos considerar la suma total de las longitudes de los fragmentos de caña y dividirla entre 4, ya que queremos maximizar el pilar más bajo. Esto nos dará una cota superior para la altura del pilar más bajo, si tomamos k como el proximo trozo a asignar, y $h[j]$ es la altura actual del pilar j, entonces la cota optimista se puede calcular como:

$$\text{Cota optimista} = \left\lfloor \frac{\sum_{i=k}^{n} l_i}{4} \right\rfloor + \min_{j=1}^{4} h[j]$$

Que claramente tendrá coste $O(n)$ al recorrer los fragmentos restantes en costes constantes.

Cota optimista - Código

El código para calcular la cota optimista podría ser el siguiente:

```
1        int cota_optimista(vector<int>& longitudes, vector<int>& alturas,
     int k) {
2            int suma_total = 0;
3            for (int i = k; i < longitudes.size(); i++) {
4                suma_total += longitudes[i];
5            }
6            int min_altura = min(alturas[0], min(alturas[1], min(alturas[2],
      alturas[3])));
7            return suma_total / 4 + min_altura; // División entera
8        }
9
```

Cota pesimista

En este problema es claro ver que cualquier solucion es factible, por mala que sea, por lo que para calcular una cota pesimista útil (siempre se podria usar 0, pero no nos serviria para nada), podemos ir asignando los fragmentos de caña al pilar más bajo en cada paso, lo que nos dará una cota inferior para la altura del pilar más bajo.
Este algoritmo también tendrá un coste de $O(n)$, ya que recorremos los fragmentos restantes y en cada paso encontramos el pilar más bajo en tiempo constante.

Cota pesimista - Código

El código para calcular la cota pesimista podría ser el siguiente:

```
1        int cota_pesimista(vector<int>& longitudes, vector<int>& alturas,
     int k) {
2            vector<int> alturas_copia = alturas; // Copia de las alturas
     actuales
3            for (int i = k; i < longitudes.size(); i++) {
4                // Encontrar el pilar más bajo
5                int min_index = 0;
6                for (int j = 1; j < 4; j++) {
7                    if (alturas_copia[j] < alturas_copia[min_index]) {
8                        min_index = j;
9                    }
10               }
11               // Asignar el fragmento al pilar más bajo
12               alturas_copia[min_index] += longitudes[i];
13           }
14           return *min_element(alturas_copia.begin(), alturas_copia.end());
15       }
16
```

Algoritmo del ejercicio

```
1       void ramificacion_poda(vector<int>& longitudes, vector<int>& alturas, int k,
    int& mejor_solucion, vector<int>& mejor_asignacion) {
2           if (k == longitudes.size()) {
3               int min_pilar = min(alturas); // Encontrar el pilar más bajo
4               if (min_pilar > mejor_solucion) { // Actualizar mejor solución
5                   mejor_solucion = min_pilar;
6                   mejor_asignacion = alturas; // Guardar la asignación actual
7               }
8           }
9           else {
10          for (int j = 0; j < 4; j++) {
11              alturas[j] += longitudes[k];          // Asignar fragmento al pilar j
12              int cota_opt = cota_optimista(longitudes, alturas, k + 1);
13              if (cota_opt > mejor_solucion) {      // Poda: solo si puede mejorar
14                  ramificacion_poda(longitudes, alturas, k + 1, mejor_solucion,
    mejor_asignacion);
15              }
16              alturas[j] -= longitudes[k];          // Deshacer la asignacion
17          }
18      }
19
```

Complejidad del algoritmo

La complejidad del algoritmo de ramificación y poda en el peor caso es $O(n4^n)$, ya que en el peor caso se exploran todas las combinaciones posibles de asignación de los fragmentos a los pilares, lo que da lugar a 4^n combinaciones, y para cada combinación se calcula la cota optimista en $O(n)$, auqnue en la práctica, la poda puede reducir significativamente el número de combinaciones exploradas gracias a las podas que se realizan al comparar la cota optimista con la mejor solución encontrada hasta el momento.
El coste en espacio es $O(n)$ debido a la profundidad de la recursión y el almacenamiento de las alturas de los pilares.