Tarea 8 - Programación y algoritmos 2

Erick Salvador Alvarez Valencia

CIMAT A.C.,
erick.alvarez@cimat.mx

Resumen En la presente tarea se describirá el código implementado para resolver el problema llamado *Squared Ends* usando una técnica de optimización hacia un algoritmo de programación dinámica.

1. Descripción del problema

El problema consiste en que se da un arreglo de N enteros positivos y un valor K, luego se pide imprimir el menor costo al hacer K particiones a dicho arreglo donde el costo de cada partición se define como: $C([A_i, ..., A_j]) = (A_i - A_j)^2$.

Restricciones del problema:

1. $1 \leq N \leq 10^4$
2. $1 \leq K \leq min(100, N)$.
3. $1 \leq A_i \leq 10^6$

2. Algoritmo implementado

Para este problema se ve que podemos aplicar programación dinámica para poder resolverlo, y para ello planteamos la siguiente tabla: $DP[K][N] =$ *Costo mínimo de hacer K particiones con los primeros N elementos del arreglo.* Y para calcular esta tabla se hace una técnica Bottom-Up donde para cada casilla k_i, i se debe iterar un j y ver cuál es la DP que minimiza el corte actual con respecto al anterior, es decir:

$$DP[k_i][i] = min(DP[k_i][i], DP[k_i - 1][j - 1] + (A_i - A_j)^2), \forall k_i \leq j \leq i \quad (1)$$

Por lo cual el algoritmo anterior tendrá costo $O(N^2K)$ y viendo las restricciones del problema este algoritmo se pasará de tiempo. Por lo cual se usará una técnica de optimización a la recurrencia anterior de la DP que consiste en utilizar la estructura *LiChao Tree* para poder encontrar la función mínima de forma más rápida y que en el caso anterior se debe hacer un ciclo.
El *LiChao Tree* se encarga de manejar un conjunto de funciones en un dominio discreto y cerrado por un intervalo, aunque se debe cumplir que estas funciones se intersecten una sóla vez. Además el árbol puede respondernos en complejidad

logarítmica cuál es la función máxima o mínima de todas las existentes en un punto específico del dominio.
Lo primero que hacemos es desarrollar a la relación de DP anterior.

$$(A_i - A_j)^2 = A_i^2 - 2A_i * A_j + T(A_j) \tag{2}$$

Si a la recurrencia de DP mostrada en (1) la interpretamos como la ecuación de una recta que sólo depende de A_j seremos capaces de usar el $LiChao\ Tree$ para calcular la función mínima de todo el conjunto por cada paso de la DP. La idea es calcular la tabla de forma Bottom-Up en donde agregamos al árbol el valor de $DP[k_i - 1][i - 1]$ con su respectivo costo generado por A_i y para obtener $DP[k_i][i]$ pedirle al árbol que de la función mínima con respecto a A_i. Los casos base de esta DP es cuando $K = 1$ ya que aquí sabemos que la partición es prácticamente el arreglo actual y sólo necesitamos calcular el costo establecido por A_n y A_1. El algoritmo quedaría de la siguiente forma.

Algorithm 1 Cálculo de la DP.

```
1: procedure DPLiChao(Arr, K, N)
2:     DP ← Table(K + 1, N + 1)
3:     for i = 1 → N do
4:         DP[1][i] ← (Arr[i] − Arr[1])^2
5:     for ki = 2 → K do
6:         tree ← LiChaoTree
7:         for i = ki → N do
8:             fAct ← Function(DP[ki - 1][i - 1], Arr[i])
9:             tree.insert(fAct)
10:            DP[ki][i] ← tree.get(Arr[i])
11:    return DP[K][N]
```

En el algoritmo anterior hay que tener cuidado al trabajar con el LiChao ya que este debe usar una versión ordenada del arreglo original y en la función insert hay que ver si es la primera vez que se inserta una función esta debe estar en todos los nodos, de caso contrario usar el algoritmo original de inserción. La complejidad final del algoritmo con esta modificación es $O(NKlogN)$ la cual si entraría en tiempo con las restricciones dadas.

3. Resultados

Se envió el código a la plataforma de CS Academy y se obtuvieron los 100 puntos del problema.

(a) Figura 1. Resultado obtenido al enviar el código a CS Academy.