Universidad Nacional Autónoma de México

FACULTAD DE CIENCIAS

Análisis y Diseño de Algoritmos

TAREA 8

Martínez Avila Santiago

Reynoso Sánchez Arturo Yitzack

17 de diciembre de 2021

1 Tarea

10 pt. Dada una secuencia de números $s = s_1, s_2, ..., s_n$, una subsecuencia creciente es una subsecuencia de la original, con la propiedad de que sus elementos aumentan monótonamente de forma estricta en valor (es decir, todo elemento es mayor que los previos a él). P.Ej. Si $S = 4, 2, -1, 3, 2, 3$, algunas de sus subsecuencias crecientes son: la secuencia 4; la secuencia 2, 3 y la secuencia -1, 2, 3.

Considera el problema de, dada una secuencia de entrada, $S = s_1, s_2, ..., s_n$, calcular alguna subsecuencia creciente de longitud máxima posible. En el ejemplo anterior, la subsecuencia -1, 2, 3 sería una respuesta válida (para esa secuencia la subsecuencia de longitud máxima es única, pero en general no tiene porqué serlo).

Tip: considera la familia de subproblemas $\mathcal{I}_i(1 \leq i \leq n)$ tal que $\mathcal{I}_i$ es el problema de determinar la longitud de la subsecuencia creciente más larga de S, tal que el s_i es el último elemento de esa secuencia.

(a) 3pt. Propón la ecuación de Bellman del problema, demostrando que es correcta por inducción.

Sea $opt(i)$ la solución al problema $\mathcal{I}_i$. Dado que una subsucesión creciente con longitud más larga tiene un último elemento con índice en {1,2,...,n}, entonces la solución al problema es $\max_{1 \leq i \leq n} opt(i)$. Vamos a mostrar que la ecuación de Bellman para el problema $\mathcal{I}_i$ es correcta:

$$\text{opt}(i) = \begin{cases} 1 & \text{si } i = 1 \text{ o } s_j \geq s_i \text{ para todo } j < i \\ \max\limits_{\{j<i \mid s_j < s_i\}} \{opt(j) + 1\} & \text{de otra forma} \end{cases} \tag{1}$$

Demostración a la ecuación de Bellman para I_i

La demostración la realizaremos por inducción sobre n, el número de elementos en la secuencia de números s dada.

Caso base n = 1

En este caso la secuencia de números $s = s_1$ consta de un solo elemento, por lo tanto, la subsecuencia creciente más larga de esta secuencia es s_1 y su longitud es 1, de modo que $opt(n) = 1$.

Hipótesis de inducción

Supongamos que la ecuación de Bellman es correcta para $k \in \mathbb{N}, k < n$, con $n > 1$.

Paso inductivo

Sea $n > 1$.
Si $s_n \leq s_k \ \forall k < n$, entonces la subsucesión creciente más larga que tiene a s_n como último elemento es s_n, que tiene un elemento. Por lo tanto $opt(n) = 1$.

Ahora supongamos que $\exists k$ tal que $s_k < s_n$. Entonces sabemos que $opt(n) > 1$. Sea

$$\mathcal{K}_n = \{k < n \mid s_k < s_n\}$$

Entonces la subsecuencia más larga que termina en s_n, debe tener alguna $k \in \mathcal{K}_n$. Por **hipótesis de inducción**, para $k \in \mathcal{K}_n$, $opt(k)$ nos da la longitud de la subsucesión creciente más larga que termina en s_k. Para obtener $opt(n)$, podemos obtener el máximo de las $opt(k)$ con $k \in \mathcal{K}_n$ y agregarle 1, que sería el resultado de agregarle s_n a la subsucesión creciente que termina en s_k con k el índice que nos dió el máximo. Por lo tanto, la ecuación de Bellman es correcta.

(b) 3pt. Propón la versión recursiva con memorización del algoritmo que resulta de aplicar la ecuación de Bellman. Analiza su corrección y complejidad.

Algorithm 1 Algoritmo Recursivo con Memorización que devuelve la longitud de la subsucesión creciente más larga

---

```
procedure OPT_MEMO(S)
    n = S.length
    Mopt = [−1] ∗ n
    for i in range(1, n) do                                   ▷ desde 1 hasta n
        opt(Mopt, S, i)
    return max(Mopt)
```

Algorithm 2 Algoritmo Recursivo con Memorización que devuelve la longitud de la subsucesión creciente más larga

---

```
procedure OPT(Mopt, S, i)
    maximo = 0
    if Mopt[i] != −1 then
        return Mopt[i]
    if i == 1 then
        Mopt[1] = 1
        return 1
    else
        for j in range (1,i-1) do                             ▷ desde 1 hasta i-1
            if S[j] < S[i] then
                maximo = max(maximo, opt(Mopt, S, j) + 1)
        Mopt[i] = max(1, maximo)
        return max(1, maximo)
```

Nota: El conjunto de índices del arreglo $Mopt$ en el algoritmo OPT_MEMO empieza en 1.

Corrección del algoritmo recursivo con memorización

Demostración: La salida del algoritmo coincide con la salida de la ecuación de Bellman.

Se crea el arreglo $Mopt$ de tamaño n, donde se guardan los valores solución para cada i. Recordemos que la salida del algoritmo es $max(Mopt)$.

Vamos a demostrar que $Mopt[i] = opt(i)$ para $i = 1, ..., n$ por inducción.

Caso base n = 1
Tenemos que $Mopt[1] = 1$ que coincide con $opt(1)$.

Hipótesis de inducción
Supongamos que para $k < n$, $Mopt[k] = opt(k)$.

Paso inductivo
Sea $n > 1$. Para cada $j \leq n$ se llama a $\textsc{opt}(Mopt, S, j)$, donde $\textsc{opt}$ regresa $Mopt[j]$ y por H.I. $Mopt[j] = opt(j)$. Ahora veamos que pasa con $\textsc{opt}(Mopt, S, n)$.
Esta llamada nos regresa $max(1, maximo)$ donde *maximo* registra la longitud máxima de las subsucesiones crecientes o, en caso de no haber, no se modifica su valor inicial que es cero.
Como *maximo* es igual al máximo de los $Mopt[k] + 1$ y como $Mopt[k] = opt(k)$ por H.I., notamos entonces que $Mopt[n] = \max\{opt(k) + 1 | s_k < s_n\}$. Por lo tanto $Mopt[n] = opt[n]$.

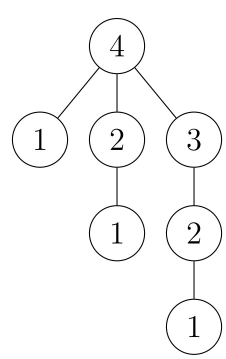
Por lo tanto la salida del algoritmo es correcta y hereda la corrección de la ecuación de Bellman.

Complejidad del algoritmo recursivo con memorización

Ahora, vamos a mostrar que la complejidad en tiempo del algoritmo $\textsc{opt\_memo}$ es cuadrática.
Tomaremos en cuenta el peor de los casos, que es cuando $s_j < s_i$ para toda $j < i$.

Para $\textsc{opt\_memo}(S, 1)$ se hace $\frac{1(1-1)}{2} + 1 = 1$ llamada, pues el algoritmo calcula para $i = 1$ directamente.
Para $\textsc{opt\_memo}(S, 2)$, necesitamos de s_1 por lo que se llama a $\textsc{opt\_memo}(S, 1)$, así se hacen $\frac{2(2-1)}{2} + 1 = 2$ llamadas.
Para $\textsc{opt\_memo}(S, 3)$ necesitamos de s_1 y s_2 por lo que se llama a $\textsc{opt\_memo}(S, 1)$ y a $\textsc{opt\_memo}(S, 2)$, así se hacen $\frac{3(3-1)}{2} + 1 = 4$ llamadas.

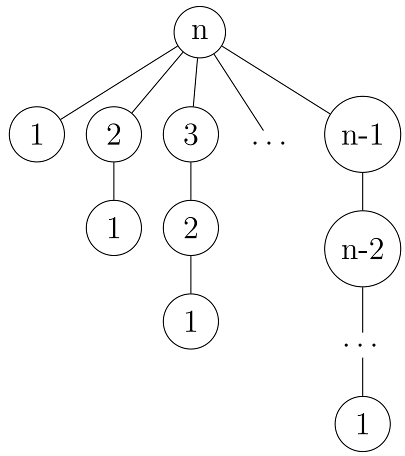
El árbol de recursión para **i=4** es:



En el primer nivel tenemos 1 elemento
En el segundo nivel tenemos $i - 1 = 3$ elementos
En el tercer nivel tenemos $i - 2 = 2$ elementos
En el cuarto nivel tenemos $i - 3 = 1$ elemento

Por lo tanto, en total, tenemos $1 + 3 + 2 + 1 = \frac{4(4-1)}{2} + 1 = 7$ elementos.

Así sucesivamente, el árbol de recursión para **i=n** es:



En el primer nivel tenemos 1 elemento
En el segundo nivel tenemos n elementos
En el tercer nivel tenemos $n - 1$ elementos
$\vdots$
En el nivel *n-1* tenemos 2 elementos
En el nivel n tenemos 1 elemento
Por lo tanto, en total, tenemos $1 + n + (n - 1) \cdots + 2 + 1 = \frac{n(n-1)}{2} + 1$ elementos.

Esto quiere decir que para encontrar la longitud máxima de una subsecuencia, se llama recursivamente a OPT_MEMO(S, i) $\frac{n(n-1)}{2} + 1$ veces y en cada llamada se realiza un número constante de instrucciones. Por lo tanto, la complejidad del algoritmo recursivo con memorización es $O(n^2)$.

(c) 3pt. Propón la versión iterativa de programación dinámica del algoritmo anterior. Analiza su corrección y complejidad.

Algorithm 3 Algoritmo iterativo de Programación Dinámica que devuelve la longitud de la subsucesión creciente más larga

```
n = len(S)
Mopt = [−1] * n
Mchoice = [0] * n
procedure OPT(S)
    for i in range(1, n) do                                   ▷ O(n)
        maximo = 0
        if Mopt[i] != −1 then
            continue
        if i == 1 then
            Mopt[i] = 1
        else
            for j in range (1,i-1) do                         ▷ O(i)
                if S[j] < S[i] and maximo < Mopt[j] + 1 then
                    maximo = Mopt[j] + 1
                    Mchoice[i] = j
            Mopt[i] = max(1, maximo)
    return max(Mopt)
```

Correccion al algoritmo de Programación Dinámica

Vamos a mostrar que cada entrada i del arreglo $Mopt$, que el algoritmo OPT llena, es la solución a la ecuación de Bellman $opt(i)$ y que, por lo tanto, al regresar el elemento máximo del arreglo $Mopt$, resuelve el problema. La demostración la hacemos por inducción sobre n, el tamaño de la sucesión S.

El algoritmo también llenará al arreglo $Mchoice$. Para explicar cada entrada de $Mchoice$, definimos como x_i^* a una subsucesión con la solución al problema $\mathcal{I}_i$. Entonces:

- Si para el índice i, el conjunto $\{j \,|\, (j < i)\ y\ (s_j < s_i)\}$ es vacío, entonces $Mchoice[i] = 0$.
  Es decir, no existen subsucesiones crecientes en S tal que, si le añadimos s_i, la subsucesión incrementa su longitud y sigue siendo creciente.
- Si el conjunto del punto anterior no es vacío, entonces $Mchoice[i]$ es el índice j tal que

$$x_j^* = \arg \max_{\{x_k^* | (k<i)\ y\ (s_k<s_i)\}} \{len(x_k^*)\}$$

> Es decir, de todas las sucesiones crecientes donde el último elemento tiene índice menor a i y además ese último elemento es menor a s_i, nos fijamos en aquella con mayor longitud y obtenemos el índice j de su último elemento.

Mchoice[i] nos asegura devolver el índice del penúltimo elemento de la sucesión cuya longitud es solución al problema $\mathcal{I}_i$. Aplicando sucesivamente Mchoice a los índices que vamos obteniendo, podemos reconstruir la sucesión cuya longitud es solución al problema $\mathcal{I}_i$.

El arreglo $Mopt$ se inicializa con entradas igual a -1 y el arreglo $Mchoice$ se inicializa con entradas igual a 0.

Paso base: n = 1
Si $n = 1$, ponemos $Mopt[1] = 1$ que coincide con $opt(1)$ de la ecuación de Bellman. Además $Mchoice[1] = 0$ pues no hay elementos anteriores a s_1.

Hipótesis de inducción

Supongamos que $Mopt[k] = opt(k)$ (la solución a la ecuación de Bellman) para $k \in \mathbb{N}, k < n$, con $n > 1$.

Paso inductivo

Sea $n > 1$.

En el `for loop` de la línea 5, creamos una variable `maximo` y lo inicializamos a 0; esta variable registrará la longitud máxima de las subsucesiones crecientes cuyo último elemento es menor a s_n y tiene índice menor a n o bien 0 si no hay tal subsucesión. Para el caso de $i > 1$, nos fijamos en todos los índices j menores a i en el *for loop* de la línea 12.

Si $s_j < s_i$ y el valor *maximo* que tenemos es menor a $Mopt[j]+1$, actualizamos *maximo* y registramos el valor de j en Mchoice. Recordamos que, por **hipótesis de inducción**, $Mopt[j] = opt(j)$.

Actualizamos `Mopt[i]` al valor máximo de 1 o `maximo`. $Mopt[i] = 1$ sólo si ninguna condición `if` de la línea 13 se cumplió (si no hay subsucesiones crecientes con último elemento menor a s_n y con índice menor a n). Esto es:

$$\text{Mopt}(n) = \begin{cases} 1 & \text{si } S[j] \geq S[i] \text{ para todo } j < i \\ \max\limits_{\{j<i \mid S[j]<S[i]\}} \{Mopt(j) + 1\} & \text{de otra forma} \end{cases} \tag{2}$$

Como $Mopt[j] = opt(j)$ para $j < n$ y $S[i] = s_i$ para $i \in \{1, 2, ..., n\}$, notamos que para n las ecuaciones (1) y (2) coinciden, de modo que $Mopt[n] = opt(n)$.

Por lo tanto, una vez que hemos llenado el arreglo $Mopt$, regresamos $max(Mopt)$, el valor máximo de Mopt que coincide con el valor máximo de *opt* de la ecuación de Bellman, y por lo tanto, el algoritmo OPT de programación dinámica es correcto.

Complejidad del algoritmo de Programación Dinámica

Por inicializar los arreglos $Mopt$ y $Mchoice$ nos toma tiempo $O(n)$. Tenemos dos ciclos `for`, en la línea 5 y en la línea 12. El ciclo `for` externo corre de 1 a n tomando tiempo O(n) multiplicado por el tiempo que toman las operaciones en el ciclo `for` interno , y el ciclo `for` interno corre de 1 a $i - 1$, tomando tiempo O(i). Las demás operaciones toman tiempo constante. Por lo tanto, el tiempo que toma el algoritmo es $O(n) + O(n^2) = O(n^2)$.

(d) 1pt. Propón el algoritmo que, a partir de la(s) tabla(s) generadas por el algoritmo anterior, calcula una subsecuencia creciente de longitud máxima (los algoritmos anteriores calculan sólo su longitud).

Algorithm 4 Algoritmo que recupera la estructura de la solución óptima

```
procedure RECUPERA_SOLUCION(S, Mopt, Mchoice)
    value = max(Mopt)
    index = Mopt.index(value)
    trayectoria = []
    while index > 0 do
        solucion.insert(0, S[index])
        index = Mchoice[index]
    return solucion
```

El algoritmo que recupera la estructura de la solución toma como parámetros de entrada a la secuencia S, y los arreglos que llenamos en el algoritmo anterior $Mopt$ y $Mchoice$.

Recordemos que $Mchoice[i]$ es igual a 0 si para s_i, no existe ninguna sucesión creciente cuyo último elemento sea menor a s_i y con índice menor a i.

Si $Mchoice[i] \neq 0$, entonces se registra al índice del penúltimo elemento de la subsucesión creciente más larga que termina en s_i.

En RECUPERA_SOLUCION, inicializamos `value` al máximo valor de `Mopt` y es la solución al problema. En `index` registramos el índice de tal valor, de modo que s_{index} es el último elemento de la subsucesión creciente más larga de S. Inicializamos una lista vacía `solucion` que guardará la sucesión de los índices de la subsucesión creciente cuya longitud es la solución al problema.

En este punto sabemos que `index` > 0. Luego, entramos en el `while` loop, cuya condición lógica es que `index` > 0. En la primera iteración, agregamos a la lista `trayectoria` $S[index]$, el índice del último elemento de la sucesión creciente más larga. Una vez que lo agregamos, actualizamos `index` a $Mchoice[index]$, es decir, al índice del penúltimo elemento de la sucesión creciente más larga que terminó en s_{index}, y repetimos el ciclo. De esta forma, podemos agregar los índices a la lista `trayectoria` hasta que `index` sea 0. Esto ocurre cuando ya no hay elementos con índice menor, menores a s_{index}. Por lo tanto, ya no se pueden agregar elementos a la subsucesión creciente más larga y terminamos.