Análisis de Algoritmos 2022-1
Tarea 8

Alumnos:
Hernández Sánchez Oscar José
Altamirano Niño Luis Enrique

17 de junio de 2022

Ejercicios

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

Respuesta:

Primero planteamos la ecuación de Bellman que nos da la longitud de la subsecuencia creciente de longitud máxima posible que contiene a s_i:

$$opt(i) = \max_{j<i, s_j<s_i} (opt(j)) + 1$$

Ahora mostraremos por inducción que la ecuación es correcta:

Demostración:

Sea $S = s_1, s_2, \ldots, s_n$ una secuencia de n términos. Por inducción en el número de términos en la secuencia:

<u>*Caso base:*</u> Si tenemos una secuencia de un solo término, entonces:

$$opt(1) = \max_{j<1, s_j<s_1} (opt(j)) + 1$$

pero como solo tenemos un solo término, entonces no hay términos que aparezcan antes que s_1, por lo que:

$$opt(1) = \max_{j<1, s_j<s_1} (opt(j)) + 1 = 0 + 1 = 1$$

que es en efecto la longitud de la subsecuencia creciente de longitud máxima posible para una secuencia de un solo término y que contiene a s_1.

<u>*Hipótesis de inducción:*</u> Supongamos que para una secuencia de $n = k$ términos, $opt(n) = opt(k)$ devuelve la longitud de la subsecuencia creciente de longitud máxima posible que contiene a s_k.

<u>*Paso inductivo:*</u> Ahora debemos mostrar que para una secuencia de $n = k + 1$ términos, la ecuación de Bellman planteada devuelve el resultado correcto, entonces:

$$opt(n) = opt(k + 1) = \max_{j<k+1, s_j<s_{k+1}} (opt(j)) + 1$$

Ahora por la hipótesis de inducción y como para cada llamada recursiva $opt(j)$ se tiene que $j < k + 1$, entonces el valor de cada una de esas llamadas recursivas devolverá el resultado correcto, entonces la ecuación toma el valor máximo de entre cada una de las llamadas recursivas y le sumará 1 pues estaremos incluyendo al término s_{k+1}, es decir, el algoritmo nos regresa la longitud de la subsecuencia creciente de longitud máxima posible que contiene a s_{k+1}.

Por lo tanto la ecuación planteada es correcta. $\blacksquare$.

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

Respuesta:

Algoritmo 1 Versión recursiva con memorización.

```
#Calcula el valor de la longitud de la subsecuencia
#creciente de longitud máxima posible que contiene a s_i
def opt(A,mem,i):
  if mem[i] !=-1:
    return mem[i]
  for j in range(i):
    if A[j] < A[i]:
      mem[i] = max(mem[i],1+opt(A,mem,j))
    else:
      mem[i] = 1
  return mem[i]

#Devuelve el valor de la la subsecuencia
#creciente de longitud máxima posible.
def subseqMax(A):
  res = [1]*(len(A))
  for i in range(len(A)):
    mem = [-1]*(i+1)
    res[i] = opt(A,mem,i)
  return max(res)
```

Corrección:

El algoritmo que implementa la función `opt` es correcto pues implementa la ecuación de Bellman que se demostró en el inciso anterior. En cuanto a la función `subseqMax`, esta simplemente va calculando las longitudes de las subsecuencias de longitud máxima posible que contienen como último término a cada uno de los términos de la secuencia y devuelve el máximo de estos valores, que es en efecto el resultado buscado y por lo tanto el algoritmo es correcto.

Complejidad:

Como `subseqMax` va calculando las longitudes de las subsecuencias de longitud máxima posible que contienen como último término a cada uno de los términos de la secuencia y en particular cada subsecuencia puede tener tamaño a lo más n, entonces tendremos $n \cdot n$ llamadas recursivas y la complejidad será $O(n^2)$.

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

Respuesta:

Algoritmo 2 Versión iterativa del algoritmo

```
def subseqMax(A):
  longitudes = [1]*(len(A))
  for i in range(len(A)):
    for j in range(i):
      if A[j] < A[i]:
        longitudes[i] = max(longitudes[i],1+longitudes[j])
  return max(longitudes)
```

Corrección:

La corrección se transfiere por el inciso anterior, pues se tiene exactamente el mismo algoritmo pero que funciona de manera iterativa.

Complejidad:

Ya que por cada término en la secuencia calculamos el valor de la subsecuencia de longitud máxima posible que contiene como último término al término sobre el que se está iterando, y la longitud máxima de la secuencia es n, entonces en el peor caso por cada término haremos n operaciones, y entonces la complejidad será $O(n^2)$.

d) Propón el algoritmo que, a partir de la tabla generada por el algoritmo anterior, calcula la subsecuencia creciente de longitud máxima(los algoritmos anteriores calculan sólo su valor).

Respuesta:

Algoritmo 3

```
def construyeSolucion(A):
  #Obtenemos el valor de la subsecuencia de longitud maxima posible.
  long_max = subseqMax(A)
  #Obtenemos el indice en la tabla generada por el algoritmo
  #donde esta el valor de la subsecuencia maxima.
  indice_max = longitudes.index(long_max)
  sol = []
  aux = A[indice_max]
  for i in range(indice_max,-1,-1):
    if A[i] < aux:
      sol.append(A[i])
      aux = A[i]
  sol.reverse()
  sol.append(A[indice_max])
  return sol
```