Análisis de Algoritmos 2022-1
Tarea 7

Alumnos:
Hernández Sánchez Oscar José
Altamirano Niño Luis Enrique

17 de junio de 2022

Ejercicios

a) Propón un algoritmo recursivo (de complejidad exponencial) que resuelva el problema, en su versión de cálculo del valor óptimo, pero no de la estructura óptima. Analiza su corrección y su complejidad.

Respuesta:

Primero planteamos la ecuación de Bellman que nos da el valor de la longitud de la trayectoria mas larga del vértice con índice 1 al vértice con índice j:

$$opt(j) = \begin{cases} 0 & \text{si } j = 1 \\ \max\{1 + opt(i) | (i,j) \in A(G)\} & \text{en otro caso} \end{cases}$$

Ahora mostraremos por inducción que la ecuación es correcta:

Demostración:

Sea $G = (V, A)$ una gráfica dirigida ordenada de n vértices. Por inducción en el número de vértices:

Caso base: Si tenemos una gráfica de $n = 1$ vértices, entonces $opt(n) = opt(1)$ regresa 0, que es en efecto la longitud de la trayectoria más larga en una gráfica de un solo vértice.

Hipótesis de inducción: Supongamos que para una gráfica de $n = k$ vértices, $opt(n) = opt(k)$ devuelve la longitud de la trayectoria más larga del vértice con índice 1 al vértice con índice k.

Paso inductivo: Ahora debemos mostrar que para una gráfica de $n = k + 1$ vértices, la ecuación de Bellman planteada devuelve el resultado correcto, entonces:

$$opt(n) = opt(k+1) = \max\{1 + opt(i) | (i, k+1) \in A(G)\}$$

como $\forall i((i, k+1) \in A(G))$, entonces hay una arista dirigida desde i hasta el vértice $k+1$ para cada vértice i y como G es ordenable entonces se cumple que $\forall i(i < k+1)$ y por la hipótesis de inducción se cumple que cada una de las llamadas recursivas $opt(i)$ regresan el resultado correcto, entonces la ecuación toma el máximo de esas llamadas, es decir, toma el valor de la trayectoria mas larga para algún vecino anterior i y le suma uno para contar la arista que va de i al vértice $k+1$, que es en efecto el resultado correcto.

Por lo tanto la ecuación planteada es correcta. ■.

Ahora planteamos el algoritmo, en donde `adyacencias` es una lista en la que cada elemento es a su vez una lista donde se tienen los índices de los vértices que tienen una arista incidente a ese vértice, por ejemplo, `adyacencias = [[],[1],[1,2]]`, significa que el vértice 1 no tiene una arista que vaya hacia él, para el vértice 2 hay una arista desde el vértice 1 hacia el, y para el vértice 3 hay una arista del vértice 1 hacia el y otro del vértice 2 hacia el.

Algoritmo 1 Algoritmo que calcula el valor óptimo para una subinstancia.

```
        def opt(adyacencias,j):
            if j==1:
                return 0
            else:
                options = []
                //El j-1 es por que los vertices se indexan desde 1
                for vecinoAnterior in adyacencias[j-1]:
                    options ++ [1 + opt(adyacencias,vecinoAnterior)]
                return max(options)
```

Algoritmo 2 Algoritmo que calcula el valor de la trayectoria más larga del vértice 1 al último.

```
    def maxTrayectoria(adyacencias):
        return opt(adyacencias, len(adyacencias)-1)
```

Corrección:

Como el algoritmo implementa la ecuación de Bellman planteada y que se demostró, se transfiere la corrección.

Complejidad:

b) Propón la versión recursiva con memorización del ejercicio anterior. Analiza su corrección y complejidad.

Respuesta:

Algoritmo 3 Algoritmo que calcula el valor óptimo para una subinstancia usando memorización.

```
    def opt(adyacencias,j,mem):
            if mem[j-1]!=-1:
                return mem[j-1]
            else:
                options = []
                //El j-1 es por que los vertices se indexan desde 1
                for vecinoAnterior in adyacencias[j-1]:
                    val = opt(adyacencias,vecinoAnterior,mem)
                    mem[vecinoAnterior-1] = val
                    options ++ [1 + val]
                return max(options)
```

Algoritmo 4 Algoritmo que calcula el valor del conjunto independiente de peso máximo

```
    def maxTrayectoria(adyacencias):
        mem = [-1]*(len(adyacencias))
        mem[0] = 0
        return opt(adyacencias, len(adyacencias)-1,mem)
```

Corrección:

La corrección se transfiere por lo mostrado en el inciso anterior, pues se tiene exactamente el mismo algoritmo con la excepción de que ahora se guardan los valores ya calculados en un arreglo para no calcularlos más de una vez.

Complejidad:

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

Respuesta:

Algoritmo 5 Versión iterativa del algoritmo

```
def maximoIterativo(adyacencias):
    Mopt = [-1]*(len(adyacencias))
    Mopt[0] = 0
    global Mchoice
    Mchoice = [-1]*(len(adyacencias))
    Mchoice[0] = 1
    for i in range (1,len(adyacencias)):
        bestValue = -1
        bestIndex = -1
        for vecinoAnterior in adyacencias[i]:
            if bestValue <= 1 + Mopt[vecinoAnterior]:
                bestValue = 1 + Mopt[vecinoAnterior]
                bestIndex = vecinoAnterior
        Mopt[i] = bestValue
        Mchoice[i] = bestIndex
    return Mopt[len(adyacencias)]
```

Corrección:

La corrección se transfiere por los incisos anteriores, pues se tiene exactamente el mismo algoritmo pero que funciona de manera iterativa.

Complejidad:

d) Propón el algoritmo que, a partir de la tabla generada por el algoritmo anterior, calcula una trayectoria óptima (los algoritmos anteriores calculan sólo su valor).

Respuesta:

La tabla que regresa el algoritmo anterior, ya regresa la estructura de la trayectoria óptima y no hay que hacer más.