Universidad Nacional Autónoma de México

Facultad de Ciencias

Análisis y Diseño de Algoritmos

Tarea 7

Martínez Avila Santiago

Reynoso Sánchez Arturo Yitzack

10 de diciembre de 2021

1 Tarea

El problema de dada una gráfica dirigida, $G(V, A)$ y dos vértices s y t de la misma, determinar la longitud de la trayectoria más larga de s a t es un problema en general muy difícil. Pero si la gráfica es suficientemente simple, podemos resolver el problema de forma eficiente. En particular podemos si la gráfica pertenece a la siguiente familia.

Una gráfica dirigida es *ordenable* si sus vértices se pueden indexar de tal forma, que toda flecha de la gráfica apunta de un vértice de índice menor a uno de índice mayor (es decir, para todos $i < j$, si v_i y v_j se conectan por una flecha, esta va de i a j se conectan por una flecha, esta va de i a j, y no al revés).

Considera el problema de, dada una gráfica dirigida $G = (V, A)$ ordenable de n vértices: $v_!, ..., v_n$, ya *ordenada* (es decir, la etiquetación de los vértices ya cumple la propiedad de que toda flecha va de un vértice a otro con índice mayor), calcular la trayectoria más larga del vértice v_1 a v_n.Dá una solución de complejidad $O(\|V\| + \|A\|)$ para este problema, usando programación dinámica.

a) **3pt. 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.**

$$\text{opt}(i) = \begin{cases} 0 & \text{si } i = 0 \\ \max\{\text{opt}(j) + 1 \mid j \in \textit{invecinos}(i)\} & \text{de otra forma} \end{cases}$$

Algorithm 1 Algoritmo Recursivo que devuelve el valor óptimo

```
procedure OPT(invecinos, i)
    if i == 0 then                                  ▷ O(1)
        return 0                                    ▷ O(1)
    else:                                           ▷ O(1)
        options = [ ]                               ▷ O(1)
        for j in invecinos[i − 1] do                ▷
            options.append(OPT(invecinos, j) + 1)   ▷ O(1)
        return max(options)                         ▷ O(1)
```

Ahora vamos a demostrar la corrección del algoritmo OPT por medio de inducción.

CORRECCIÓN

Dado un arreglo *invecinos* de las adyacencias de una gráfica dirigida ordenable $G = (V, A)$, el algoritmo $\textsc{opt}(invecinos, i)$ regresa la longitud de la trayectoria más larga del vértice v_1 al vértice v_{i+1}.

Caso base
Para $i = 0$ buscamos la trayectoria más larga de v_1 a v_1. Como no existe trayectoria de un vértice a sí mismo y $opt(0) = 0$, el algoritmo devuelve el valor óptimo.

Hipótesis de inducción
Supongamos que para $j \leq n$, el algoritmo con entradas $\textsc{opt}(invecinos, j)$ devuelve la longitud de la trayectoria más larga de v_1 a v_{j+1}.

Paso inductivo
Consideremos una gráfica dirigida ordenable de $n+2 > 0$ vértices, $\textsc{opt}(invecinos, n+1)$ regresa el máximo de los $\text{opt}(j) + 1$ tal que v_j es vecino de v_{n+1}, pero todos los v_j necesariamente cumplen que $j < n + 1$ pues la gráfica es ordenable. Y $j < n + 1$ implica que $j \leq n$ así, por hipótesis de inducción, $\textsc{opt}(invecinos, j)$ es la longitud de la trayectoria más larga de v_1 a v_{i+1} para toda $j \leq n$. Notemos que la trayectoria más larga de v_1 a v_{n+2} es igual al vértice v_{n+2} unido la trayectoria más larga de v_1 a cualquier otro vértice, tal que este vértice es vecino de v_{n+2}; esto implica que la longitud de la trayectoria más larga de v_1 a v_{n+2} es igual a 1 más la longitud de la trayectoria más larga de v_1 a cualquier otro vértice, es decir, 1 más el máximo de los $\textsc{opt}(invecinos, j)$ tal que j es vecino de v_{n+2}, que es igual al máximo de los $\textsc{opt}(invecinos, j)+1$ tal que j es vecino de v_{n+2} y esto es igual a $\textsc{opt}(invecinos, n + 1)$. Por lo tanto el algoritmo es correcto.

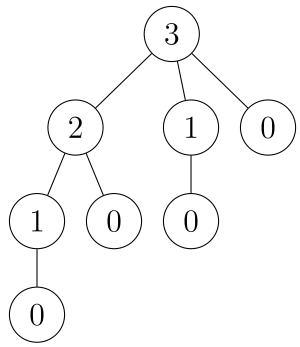
COMPLEJIDAD

Ahora, vamos a mostrar que la complejidad en tiempo del algoritmo OPT es exponencial.
Tomaremos en cuenta el peor de los casos, que es cuando el vértice v_j tiene de vecino a todos los vértices v_i tal que $i < j$.

Para $\text{OPT}(invecinos, 0)$ se hace $2^0 = 1$ llamada, pues el algoritmo calcula para $i = 0$ directamente.
Para $\text{OPT}(invecinos, 1)$ hay un vecino v_1 por lo que se llama a $\text{OPT}(invecinos, 0)$, así se hacen $2^1 = 2$ llamadas.
Para $\text{OPT}(invecinos, 2)$ hay dos vecinos v_1 y v_2 por lo que se llama a $\text{OPT}(invecinos, 1)$

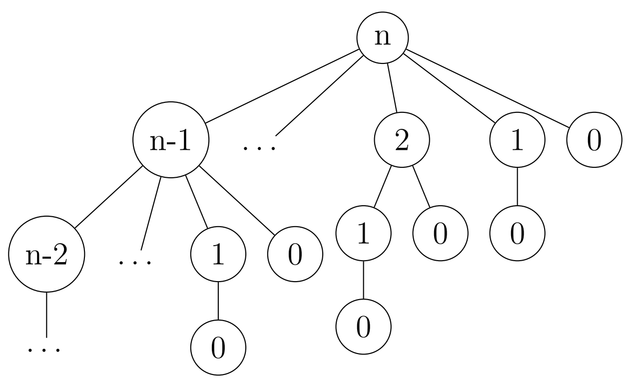
y a OPT($invecinos, 0$), así se hacen $2^2 = 4$ llamadas.

El árbol de recursión para **i=3** es:



$$\text{En el primer nivel tenemos } \binom{3}{0} = 1 \text{ elementos}$$
$$\text{En el segundo nivel tenemos } \binom{3}{1} = 3 \text{ elementos}$$
$$\text{En el tercer nivel tenemos } \binom{3}{2} = 3 \text{ elementos}$$
$$\text{En el cuarto nivel tenemos } \binom{3}{3} = 1 \text{ elementos}$$
$$\text{Por lo tanto, en total, tenemos } \binom{3}{0} + \binom{3}{1} + \binom{3}{2} + \binom{3}{3} = 2^3 = 8 \text{ elementos.}$$

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



$$\text{En el primer nivel tenemos } \binom{n}{0} = 1 \text{ elemento}$$
$$\text{En el segundo nivel tenemos } \binom{n}{1} = n \text{ elementos}$$
$$\vdots$$
$$\text{En el nivel } \textit{n-1} \text{ tenemos } \binom{n}{n-1} \text{ elementos}$$
$$\text{En el nivel } n \text{ tenemos } \binom{n}{n} \text{ elementos}$$
$$\text{Por lo tanto, en total, tenemos } \binom{n}{0} + \binom{n}{1} + \cdots + \binom{n}{n-1} + \binom{n}{n} = 2^n \text{ elementos.}$$

Esto quiere decir que para encontrar la longitud de la trayectoria más larga de v_1 a v_{n+1}, se llama recrsivamente a OPT($invecinos, i$) 2^n veces y en cada llamada se realiza un número constante de instrucciones. Por lo tanto, la complejidad del algoritmo recursivo es $O(2^n)$.

b) **3pt. Propón la versión recursiva con memorización del ejercicio anterior. Analiza su corrección y complejidad.**

Algorithm 2 Algoritmo Recursivo con Memorización que devuelve el valor óptimo

```
n = len(invecinos)
M = [−1] ∗ n
procedure OPT(invecinos, i)
    if M[i]! = −1 then                                   ▷ O(1)
        return M[i]                                      ▷ O(1)
    else
        if i == 0 then                                   ▷ O(1)
            return 0                                     ▷ O(1)
        else:                                            ▷ O(1)
            options = [ ]                                ▷ O(1)
            for j in invecinos[i − 1] do                 ▷
                options.append(OPT(invecinos, j) + 1)    ▷ O(1)
            M[i] = max(options)
            return max(options)                          ▷ O(1)
```

CORRECCIÓN

En este algoritmo creamos un arreglo M tal que, para cada índice i, guarda la longitud de la trayectoria más larga entre v_1 y v_{i+1}. Para cada índice i, el algoritmo del inciso a) y el algoritmo b) tienen la misma salida, i.e., el algoritmo del algoritmo con memorización hereda la corrección del algoritmo recursivo :

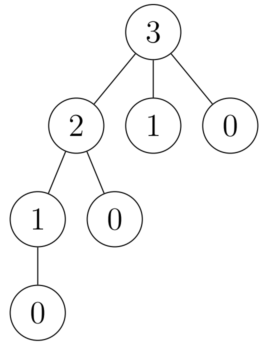
- Si $i = 0$, ambos devuelven 0.
- Si $i > 1$, ambos devuelven el valor máximo entre los $\text{OPT}(invecinos, j)$ tal que v_j es vecino de v_i.

COMPLEJIDAD

Ahora, vamos a mostrar que la complejidad en tiempo del algoritmo OPT es cuadrática. Tomaremos en cuenta el peor de los casos, que es cuando el vértice v_j tiene de vecino a todos los vértices v_i tal que $i < j$.

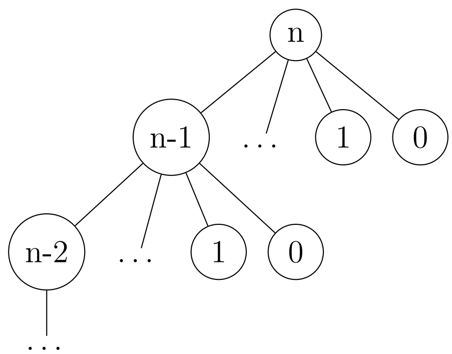
Para $\text{OPT}(invecinos, 0)$ se hace $\frac{0(0+1)}{2} + 1 = 1$ llamada, pues el algoritmo calcula para $i = 0$ directamente.
Para $\text{OPT}(invecinos, 1)$ hay un vecino v_1 por lo que se llama a $\text{OPT}(invecinos, 0)$, así se hacen $\frac{1(1+1)}{2} + 1 = 2$ llamadas.
Para $\text{OPT}(invecinos, 2)$ hay dos vecinos v_1 y v_2 por lo que se llama a $\text{OPT}(invecinos, 1)$ y a $\text{OPT}(invecinos, 0)$, así se hacen $\frac{2(2+1)}{2} + 1 = 4$ llamadas.

El árbol de recursión para **i=3** es:



En el primer nivel tenemos 1 elemento
En el segundo nivel tenemos $i = 3$ elementos
En el tercer nivel tenemos $i - 1 = 2$ elementos
En el cuarto nivel tenemos $i - 2 = 1$ elemento
Por lo tanto, en total, tenemos $1 + 3 + 2 + 1 = \frac{3(3+1)}{2} + 1 = 7$ elementos.
La diferencia con el algoritmo anterior es que para calcular $i = 2$ se calculó $i = 1$ y así al volver a calcular $i = 1$ ya no hace falta calcular $i = 0$ y por eso se disminuyó 1 elemento.

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 de la trayectoria más larga de v_1 a v_{n+1}, se llama recursivamente a $\text{OPT}(invecinos, i)$ $\frac{n(n+1)}{2} + 1$ 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.**

La ecuación de Bellman del problema es

$$opt(i) = \begin{cases} 0, & \text{si i==0} \\ \max\{opt(j)+1 | v_j \in N(i)\}, & \text{si } i > 0 \end{cases}$$

donde `opt(i)` nos da la longitud de la trayectoria más larga de v_1 a v_{i+1}, y $N(i)$ son los vértices invecinos de v_i (aristas que entran a v_i). Necesariamente tenemos que $j < i$. Para este algoritmo usaremos una lista de n listas de adyacencias (flechas que entran a un vértice) *invecinos*: la lista i-ésima contendrá una lista de enteros j que indican que hay una flecha que sale del vértice v_j y entra al vértice v_i. Como la gráfica es ordenable, el primer vértice es v_1 y el índice de la lista empieza en cero, entonces la 0-ésima lista y la 1-ésima lista son vacías.

Crearemos dos arreglos `Mopt` y `Mchoice` de tamaño n donde guardaremos los valores óptimos de `opt` y el índice del vecino j que dió la solución óptima para i en la ecuación recursiva. Supondremos que los índices de los arreglos empiezan en 0.

Algorithm 3 Algoritmo de Programación Dinámica que regresa la longitud de la trayectoria más larga del vértice v_1 a v_n

```
Mopt = [0] * n                                   ▷ O(n)
Mchoice = [0] * n                                ▷ O(n)
maxvalue = 0;
maxvecino = 0;
procedure OPT(invecinos)
    for i in range(n-1): do                      ▷ (el rango incluye a n-1)   O(n)
        if i == 0 : then
            pass                                 ▷ do nothing
        else if i > 0 : then
            maxvalue = 0
            maxvecino = 0
            for j in invecinos[i+1]: do          ▷ O(i)
                if max < Mopt[j − 1] : then
                    maxvalue = Mopt[j − 1]
                    maxvecino = j
            Mopt[i] = maxvalue + 1
            Mchoice[i] = maxvecino
    return Mopt[n − 1]
```

CORRECCIÓN

Recordemos que $Mopt[i]$ guarda el valor de la trayectoria más larga de v_1 a v_{i+1} y que $invecinos(i)$ nos da la lista de invecinos de v_i.

Primero vamos a mostrar que la ecuación de Bellman, con entrada n, nos da la solución correcta al problema. Luego vamos a mostrar que cada entrada del arreglo $Mopt[i]$

es la solución a la ecuación de Bellman $opt(i)$. Además mostramos que los valores que guarda el arreglo $Mchoice$ son los $argmax$ de la ecuación de Bellmman. De ahí concluiremos que la salida del algoritmo, $Mopt[n-1]$ es la solución a la ecuación de Bellman $opt(n-1)$ y, por lo tanto, la solución al problema.

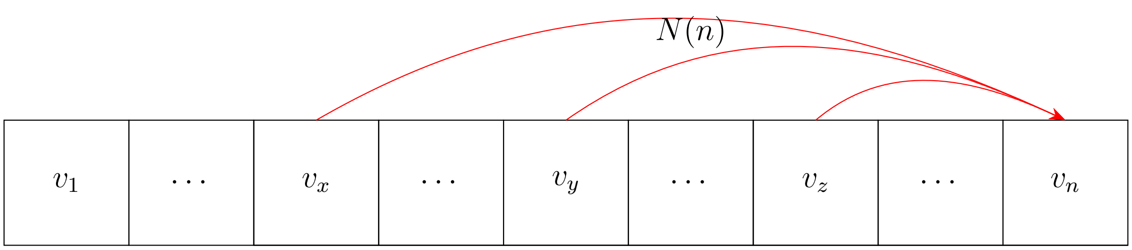
Demostramos la ecuación de Bellman por inducción sobre i, es decir, para $n \in \mathbb{N}$, $opt(n-1)$ nos da la longitud de la trayectoria más larga del vértice v_1 al vértice v_n.

La ecuación de Bellman es correcta

Caso base $i = 0$
Cuando $i = 0$ estamos preguntando cuál es la longitud máxima de la trayectoria que va de v_1 a v_1. La respuesta es cero.

Hipótesis de inducción
Supongamos que la ecuación de Bellman es verdadera para $i < n-1$, de modo que $opt(i)$ nos da la trayectoria más larga que va de v_1 a v_{j+1}.

Paso inductivo
Notamos que cualquier trayectoria que va del vértice v_1 al vértice v_n pasa por un vértice conectado con v_n. La trayectoria con longitud máxima necesariamente tiene a alguno de los vecinos de v_n, (en la figura de abajo son v_x, v_y y v_z) y como suponemos que la gráfica es ordenable, sus vecinos tienen índice menor a n. Ahora, si buscamos la longitud máxima de las trayectorias que tienen como último vértice a los vecinos de v_n, entonces la solución nos la da el máximo de estas soluciones más 1, que es la flecha que va del vecino hacia v_n.



Como la gráfica es ordenable, para todo $j \in N(n)$, $j < n$, y por hipótesis de inducción tenemos que $opt(j)$ nos da la longitud de la trayectoria máxima de v_1 a v_{j+1}. Por lo tanto, $\max\{opt(j) + 1 | v_j \in N(n)\}$ nos da la longitud de la trayectoria más larga que va de v_1 a v_n y la ecuación de Bellman es correcta.

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

Se crean los arreglos $Mopt$ y $Mchoice$ de tamaño n; en $Mopt$ se guardan los valores solución para cada i. Recordemos que la salida del algoritmo es $Mopt[n-1]$ pero necesita de estos valores intermedios. $Mchoice$ guardará las elecciones que se van haciendo para cada i, la idea es que para cada índice i, $Mchoice[i]$ nos guarda el índice j (correspondiente a v_j) que es invecino de v_{i+1} con trayectoria de mayor longitud.

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

Caso base n = 0
Tenemos que $Mopt[0] = 0$ que coincide con $opt(0)$.

Hipótesis de inducción
Supongamos que para $k < n-1$, $Mopt[k] = opt(k)$.

Paso inductivo
$Mopt[n-1] = maxvalue + 1$. Como $maxvalue$ es igual al máximo de los $Mopt[j-1]$ (el valor de la trayectoria de los v_j con mayor longitud invecinos de v_{i+1}) y como $Mopt[j] = opt(j)$ por H.I., notamos entonces que $Mopt[n-1] = \max\{opt(j) | j \in N(n)\}$. Por lo tanto $Mopt[n-1] = opt[n-1]$.

Nota: Resaltamos que en el `for loop` estamos llenando el arreglo $Mopt$ de izquierda a derecha, y la entrada $Mopt[i]$ depende de las entradas de sus vecinos $Mopt[j]$. En la iteración i, si $j < i$, sabemos que $Mopt[j] = opt(j)$, de modo que ya está disponible ese valor para la entrada $Mopt[i]$. Por lo tanto, **el llenado de izquierda a derecha para $Mopt$ es correcto**.

Por lo tanto la salida del algoritmo es correcta y hereda la corrección de la ecuación de Bellman. Además en cada iteración i del for, registramos en $Mchoice[i]$el índice del vértice invecino de $i+1$ que tiene trayectoria máxima, i.e.,

$$Mopt[Mchoice[i]] = \max\{Mopt[j] | j \in N(i)\}$$

COMPLEJIDAD
El costo de inicializar los arreglos $Mopt$ y $Mchoice$ es $O(n)$. Ahora, el costo de actualizar cada entrada de $Mopt$ es proporcional a i, es decir es una constante multiplicada por el número de vecinos que puede tener ese vértice, que para i es a lo más $i-1$. Para procesar $Mopt[i]$ cuesta $cO(i-1)$. Notamos que para la gráfica completa ordenable $|A|$ tiene a lo más $\frac{(n-1)n}{2}$ aristas. Sumando el costo total de procesar todas las entradas nos da una complejidad $O(n^2)$. Por lo tanto, el costo total del algoritmo es $O(n^2+n) = O(n^2)$.

d) 1pt. 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).

Algorithm 4 Algoritmo de Programación Dinámica que recupera la estructura de la solución óptima

```
procedure RECUPERA_ESTRUCTURA(Mchoice, i)
    if i = 1 then
        return [1]
    else if i > 1 then
        return recupera_estructura(Mchoice, Mchoice[i-1]) + [i]
```

Vamos a mostrar que el algoritmo RECUPERA_ESTRUCTURA(MCHOICE, I) nos da los índices de la trayectoria **con longitud máxima que va de v_1 a v_i**, y lo haremos por inducción sobre n. Entonces la solución del problema estaría dado por RECUPERA_ESTRUCTURA(MCHOICE, N)

Caso base: n = 1

Supongamos que n = 1. En este caso el algoritmo nos devuelve la lista [1], que contiene al entero 1, i.e., la trayectoria solución es v_1.

Hipótesis de Inducción

Supongamos que para $k < n$, el algoritmo RECUPERA_ESTRUCTURA(MCHOICE, K) nos devuelve la trayectoria de vértices (representado como una lista de índices) más larga que va del vértice v_1 al vértice v_k.

Paso inductivo

Recordemos que la entrada $Mchoice[i]$ nos da el índice $j < i + 1$ del invecino de v_{i+1} tal que la trayectoria de v_1 a v_j es máxima. Ahora bien, toda trayectoria de v_1 a v_n contiene al vértice v_n, en particular, la trayectoria más larga y solución del problema. Entonces la trayectoria máxima:

- tiene al vértice v_n
- como tiene al vértice v_n, necesariamente tiene a algún invecino de v_n, de otra manera no podemos llegar a v_n.

Por lo tanto, la trayectoria más larga que va a v_1 a v_n es aquella trayectoria más larga que va de v_1 a algún invecino v_j de v_n. Ese invecino con trayectoria más larga nos lo proporciona $Mchoice[n - 1]$. Por hipótesis de inducción, la trayectoria más larga que va de v_1 a $v_{Mchoice[n-1]}$ es RECUPERA_SOLUCION(MCHOICE, MCHOICE[N-1]). Por lo tanto, la solución al problema es RECUPERA_SOLUCION(MCHOICE, MCHOICE[N-1]) + $[n]$.

Este algoritmo tiene árbol de recursión lineal, donde cada instancia tiene un solo hijo, por lo que tiene complejidad $O(n)$.