

MIGLI, JUAN PABLO  
LU: 424/19

## ej2: Desaprobado

HOJA N° 1  
FECHA 14/05/2021

2) Comenzamos realizando un BFS comienzo en  $s$ . Si  $\text{nivel}(t)$  es par entonces devolvemos el camino generado.

En los contrarios tenemos que buscar el mínimo camino ~~par~~ de longitud par. Para eso haremos ~~un BFS~~ BFS desde  $t$ .

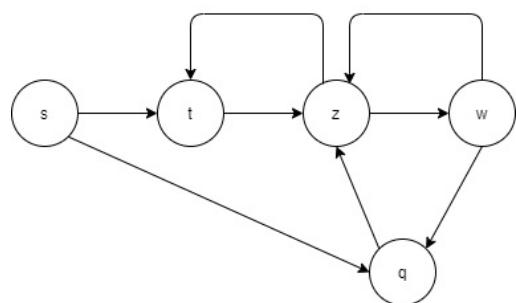
Las primeras que haremos verá dar vuelta la orientación de los aristas en el grafo. Los que quedan en el BFS verá encontrar alguna arista que una 2 nodos cuyos niveles son ambos par o ambos impares.

De esta manera el nuevo camino se podrá formar usando esa arista. En decir, si encontramos  $v \rightarrow w / \text{dist}(t, w) > \text{dist}(t, v)$  en  $G'$  (arista dada vuelta) entonces  $\exists$  minimo en  $G' / C = \{P_G(s, w) + (w \rightarrow v) + P_G(v, t)\}$ . Al agregar

una arista combinan la paridad del camino y el mismo ahora resulta par. Al momento de hacer BFS vamos a ir marcando los vértices para los que tenemos como visitados y si en algún momento encontramos una arista que conecte 2 nodos de mismo mismo paridad calculando el ~~total~~ largo del camino entre:  $\text{nivel}_G(w) + \text{nivel}_{BFS}(v)$  y el camino resó  $P_G(s, w) \rightarrow P_{BFS}(w, t)$ . En decir vemos en el primer BFS

hasta el vértice que tiene un par de mismo paridad. Luego subimos por el segundo BFS hasta  $t$  (normalizando las aristas). El camino mínimo verá el mínimo de todos los encontrados durante el segundo BFS. La complejidad de todo el algoritmo es  $O((m+n)^2) \in O(m^2)$ .

El modelado no resuelve el problema. En el siguiente contraejemplo no existe un vértice  $u$  tal que  $\text{dist}(s, u)$  y  $\text{dist}(u, t)$  sea ambos par o impar.



1) a)

$$G(i, v) = \begin{cases} \max(t_{i,v}, t_{i,s}) & , i = m \\ \min_j [t_{i,j} + G(i+1, v-j+v_i), 0 \leq j \leq v] & , i < m \end{cases}$$

Explicación:

Si llega al último nivel con  $v$  vidas entonces intentará usar todos ya que ~~solo~~ le asegura el mejor resultado. Sin embargo debemos tener en cuenta que a los niveles puede unir 5 vidas. Ese es el caso base.

En el paso recursivo queremos minimizar la suma entre el tiempo de este nivel ( $i$ ) y el tiempo para terminar el juego desde el próximo nivel. Tenemos a los niveles 5 posibilidades, dependiendo de con cuantas vidas llegué. Intentaremos todos y devolveremos la que menor tiempo tardó.

Debemos tener en cuenta para el próximo nivel las vidas perdidas en el nivel  $i$ , más las vidas ganadas ( $v_i$ ).



b) Supongamos un caso donde siempre ganamos  $n$  vidas. Supongamos  $m \geq 5$  esto significa que en todos los niveles cada puesta nos da 5 vidas. Aquí tenemos nuestra cantidad máxima de monedas recibidas. Sabiendo que cada llamada lleva a otros 5 entonces tenemos  $5^n$  llamadas. Viéndolo en símbolos que  $m \in [0, \dots, n^2]$  e  $i \in [1, \dots, m]$ . Esto no deja con a los niveles  $n^3$  problemas distintos. Más allá, en cada intento con  $m \in [5m, \dots, 5n^2]$  se obtendrá el mismo resultado, ya que por restricción en todo el juego no podemos usar más de  $5m$  vidas.

Por lo que se cumple la superposición de subproblemas ya que  $5^n \gg 5 \cdot n^2$ , en orden cuantitativo.



NOTA

Aclaración: es  $6^n$ , no  $5^n$

c) Usaremos una matriz para la memorización. La misma tendrá  $m$  filas (una por nivel) y  $5^n$  columnas (una por cada posible contenido de vides). Esta será iniciada en  $\perp$ . ~~La función~~ La primera que veremos en la función es la entorno en el caso base, en ese caso devolveremos el valor almacenado en  $\mathcal{O}(1)$ . Si no es el caso, preguntaremos si ya tenemos el resultado pedido, si decir si está en memo. Si no está calcularemos el valor con la función recursiva del ejercicio a). Luego, guardamos el valor en memo  $[i, v]$  y devolvemos el valor.



$$d) \text{ memo } [l, \dots, m] [0, \dots, S_m] = \perp$$

def  $G(i, v)$ :

```

if  $i = m$  {
    return max(tiv, tis)
}
if  $\text{memo}[i, v] = \perp$  {
    contor  $[0, \dots, 5] = \max \text{INT}$ 
    posible = min( $v, s$ )
    for  $j [0, \dots, \text{posible}]$  {
        contor  $[j] = T_{ij} + G(i+1, v-j+s)$ 
    }
     $\text{memo}[i, v] = \min(\text{contor})$ 
}
return  $\text{memo}[i, v]$ 
```



# Parcial aprobado

MICELI, JUAN PABLO  
LU : 424/19

ej3: Aprobado

HOJA N° 3

FECHA 14/05/2021

sobre el grafo G

3) Empezamos corriendo DFS y todo nodo tendrá un  $t_{in}$  que es el tiempo en donde se nodo se descubrió. Entonces dada un árbol  $T$  de recorridos el nodo siguiente post-order es decir nos analizamos la rama hasta haber analizado todo lo subarbols. Al analizar cada nodo  $v$  le definiremos un  $t_{out}$ . Tanto como indica hasta donde puede llegar donde se pasea rubis en sus back-edge que salgo de ese nodo, o de uno de sus hijos. Esos niños en la práctica tanto  $t_{in} = t_{min}$ ,  $t_{out} = t_{max}$ ,  $t_{in} < t_{out}$ ,  $v$  hijo y  $w$  hermano.

Hasta ahora tenemos  $O(n)$  en DFS y  $O(m)$  al calcular el  $t_{out}$  de cada nodo.

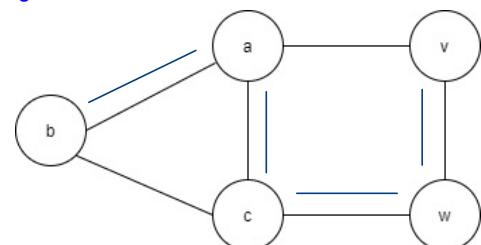
Algo más ~~complejidad~~ analizamos un ordenamiento del grafo para ver si el árbol es más fácil o no. Post-order y nos sirve para ver si el árbol es más fácil.

Ahora iteraremos sobre los aristas  $vw / v, w \in G \setminus \{r\}$  con  $t_{in} < t_{in} w$ .

Para cada una de estos analizaremos el camino en  $T$   $P(v, w)$ .

Si algún nodo en  $P(v, w)$  tiene una arista distinta de  $vw$  que conecte 2 nodos cualesquier de  $P_T(v, w)$ , entonces el árbol no es fácil y se devolverá ERROR. Caso contrario, se eliminan las aristas de monopas entre  $v, w$  y toda orilla de  $P_T(v, w)$ . Haciendo esto para toda arista back-edge se consigue minimizar el árbol. Sabemos que  $m \leq 2n$ . Además, al hacer este segundo paso no se visitará ninguna arista de  $G$  más de una vez, sabemos que el grafo no sea fácil ya que en este caso podríamos pines algunas aristas por segunda vez, pero en este caso el algoritmo termina de inmediato.

No basta con este chequeo para determinar si el grafo es fácil. En el siguiente contraejemplo para la arista  $a \rightarrow v$  que no pertenece al  $T$  (árbol generado por DFS), no existe una arista que conecte dos nodos del camino  $a \rightarrow c \rightarrow w \rightarrow v$  por lo tanto no va a tirar error. Y lo mismo para la arista  $b \rightarrow c$ . Pero el grafo no es fácil.



NOTA

Es por esto que el algoritmo  $\in O(n)$ . Yos que DFS  $\in O(n+m) \in O(n)$  y la segunda parte  $\in O(n+m) \in O(n)$ .

Es importantes aclarar que se debe iterar sobre todos los backedges independientemente si se ha eliminado para ver la altura máxima del ciclo.

La idea en general esta bien.