(1)

Pregunta 1 ♣ "A guardar, a guardar..." dice la profe de jardín, y Eric sabe que tiene que guardar todos los juguetes que usó en los cajones. Sabe que puede usar hasta c cajones y que usó j juguetes, con $c < j$. El ancho y largo de los juguetes es justo el ancho y el largo de los cajones. Cada cajón tiene la misma altura A, y cada juguete i, con $1 \leq i \leq j$, tiene altura a_i. Sabe que hay alguna forma de poner todos los juguetes en los cajones, pero no se acuerda de qué cajón sacó cada juguete. Va a hacer lo único que sabe: buscar la solución con backtracking. ¿Qué complejidades temporales serían las más ajustadas para los algoritmos de backtracking que podría usar?

- ☐ $O(j!)$
- ☐ $O(j! \cdot j)$
- ☐ $O(c^j)$
- ☐ $O(A \cdot \sum_{i=1}^{j} a_i)$

Complejidad temporal de backtracking != # hojas arbol backtracking

Complejidad backtracking = #nodos * complejidad nodo

Tenemos un arbol c-ario de altura j → #nodos = $\sum_{i=1}^{j} c^i = \frac{c^{j+1}-1}{c-1} \in O(c^j)$

En este caso en cada nodo asignamos un numero, esto es O(1)

Luego O(c^j)*O(1) = O(c^j)

En realidad cuando c=1 → es O(j)

Luego es O(c^j + 1)

(2) En programacion dinamica se cumple que:

- Hay más de un posible valor de retorno para cada subproblema.
  Falso. La idea es armar la solucion a partir de las soluciones parciales de cada subproblema. No todo problema tiene soluciones multiples a los subproblemas. Por ejemplo, calcular una sumatoria de una matriz.

- Utilizar memorización garantiza que solo se calculen los subproblemas necesarios, lo que siempre reduce el tiempo de ejecución del algoritmo.
  Falso. No siempre mejora los tiempos por ej si no pesan mucho los subproblemas. Por ejemplo podia utilizarse memorizacion con un enfoque Bottom-up y calcular una solucion para todos los estados posibles, no solo los necesitados.

- Utilizar memorización te obliga a utilizar una solución recursiva para cada problema.
  Falso. El enfoque bottop-up es iterativo.

- Cuándo hay múltiples párametros en la función implementada se necesita una matriz (de 2 o más dimensiones) para memoizar los resultados.

Falso. No necesariamente tenemos que emplear una matriz. Ademas, se pueden usar variables de estado si utilizamos valores anteriores como en fib(n) (es similar si tuviera mas de un parametro).

(3)

Pregunta 3 ♣ Se tiene un arreglo A ordenado decrecientemente de n números naturales consecutivos con k huecos, es decir, en el arreglo todos los elementos son consecutivos excepto en k posiciones. Por ejemplo, el arreglo $A = [19, 16, 15, 13, 12, 11]$ tiene 2 huecos en los valores $(19, 16)$ y $(15, 13)$. El tamaño de un hueco (x, y) se define cómo $|x - y|$. Se puede asumir que el arreglo tiene tamaño potencia de 2. Sea t el tamaño del hueco más grande. ¿Cuál sería la cota más ajustada del mejor algoritmo $D\&C$ en base a A, n, k, t que encuentra el valor de t?

- ☐ $O(log(n) * k)$
- ☐ $O(log(n) * k + t)$
- ☐ $O(n * k)$
- ☐ No se puede calcular su complejidad con el teorema maestro pues resolver su caso base no es $O(1)$.
- ☐ No se puede calcular su complejidad con el teorema maestro pues no siempre se subdivide en la misma cantidad de suproblemas.

No se puede calcular con el teorema maestro porque al dividir puede haber o no huevo en una de las mitades entonces solo se calcula en un lado.

O(k*log(n))

(4)

Pregunta 4 El tiempo de ejecución de un determinado algoritmo responde a la siguente recurrencia, con su caso base para $T(i) \in \Theta(1)$ para todo $i < 10$.

$$T(n) = 3 * T\left(\frac{n}{9}\right) + 5 * n^{1/4}$$

Para determinar la complejidad de este algoritmo, se nos presenta la siguiente demostración utilizando el teorema maestro de que la recurrencia es $\Theta(\sqrt{n})$.
Nos encontramos en el primer caso del teorema maestro. Puesto que para $\epsilon = -6$, se cumple que $f(n) = 5 * n^{1/4} \in O\left(n^{\log_c a - \epsilon}\right) = O\left(n^{\log_9 3+6}\right) = O\left(n^1\right)$, por lo que $T(n) = \Theta\left(n^{\log_c a}\right) = \Theta(\sqrt{n})$.

- ☐ No se puede utilizar teorema maestro para esta función de recurrencia
- ☐ La complejidad no es ajustada, la recurrencia es $O\left(n^{1/4}\right)$
- ☐ La complejidad es ajustada, la demostración es incorrecta.
- ☐ La complejidad no es ajustada, la recurrencia es $O\left(n^{1/3}\right)$
- ☐ La demostración es correcta.

Son verdaderas:
La complejidad es ajustada (caso 1 de Teorema Maestro). 😀
. La demostracion es incorrecta (\epsilon > 0) 😀

Pregunta 5 ♣ Un grafo *etiquetado* G es un grafo cuyos vértices tienen cada uno una etiqueta distinta en el conjunto $\{1, \ldots, n\}$, donde n es la cantidad de vértices de G. En G, cada arista se etiqueta utilizando las etiquetas de sus vértices incidentes. Dos grafos etiquetados de n vértices son *iguales* cuando tienen el mismo conjunto de aristas etiquetadas. Una *orientacion acíclica* de un grafo (etiquetado o no) G es un grafo orientado H que no tiene ciclos dirigidos y que resulta de asignarle una dirección a cada arista de G. Ciertamente, si G es etiquetado, entonces H mantiene las mismas etiquetas que G. ¿Cuál/es de las siguientes afirmaciones son verdaderas?:

- ☐ Todo grafo etiquetado con al menos una arista tiene una cantidad impar de orientaciones acíclicas distintas.
- ☐ Toda orientación acíclica de un grafo tiene al menos un *sumidero*[*].
- ☐ Todo grafo etiquetado tiene al menos un *sumidero*[*].
- ☐ Todo grafo etiquetado con al menos una arista tiene una cantidad par de orientaciones acíclicas distintas.

[*]Un sumidero es un vértice con grado de salida cero.

a. Todo grafo etiquetado con al menos una arista tiene una cantidad impar de orientaciones aciclicas distintas.
Falso. La existencia de una orientacion aciclica implica la existencia de otra orientacion aciclica (proveniente de invertir las aristas de la orientacion primera). Luego, es una cantidad par.

b. Toda orientacion acilcica de un grafo tiene al menos un sumidero.
Verdadero. Si todos los vertices tienen grado de salida >0, entonces al menos hay un camino que nunca termina, lo cual implica que hay un ciclo.

c. Todo grafo etiquetado tiene al menos un sumidero.
Falso. No hay restriccion para la forma original del grafo.

d. Todo grafo etiquetado con al menos una arista tiene una cantidad par de orientaciones aciclicas distintas.
Verdadero.

Pregunta 6 ♣ Seguimos en el contexto del grafo etiquetado de la pregunta anterior. ¿Cuál/es de las siguientes afirmaciones son verdaderas? Aclaración: K_n es el grafo completo de n vértices.

- ☐ Si a una orientación acíclica H de un grafo le agregamos un nuevo vértice v junto a una arista $u \to v$ para todo vértice u de H, entonces el grafo resultante es la orientación acíclica de un grafo.
- ☐ Toda orientación acíclica de K_n tiene un único sumidero.
- ☐ Toda orientación acíclica de K_n tiene exactamente dos sumideros (para $n \geq 4$).
- ☐ El grafo K_n etiquetado tiene exactamente n^2 orientaciones acíclicas distintas.

a. Si a una orientacion aciclica H le agregamosun vertice …
Verdadero. El hipotetico ciclo de H' deberia tener a v como miembro, pero un vertice de grado de salida 0 no puede estar en un ciclo → no hay un ciclo en H' → H' es aciclica.

b. Toda orientacion aciclica de Kn tiene un unico sumidero.
Verdadero, si hay dos sumideros que pasaria con la arista entre ellos? imposible

c. Toda orientacion aciclica de Kn tiene exactamente 2 sumideros para n>=4.
Falso. Hay contraejemplos para n=4 y n=5 con un solo sumidero.

d. El grafo Kn etiquetado tiene exactamente n^2 orientaciones aciclicas distintas.
Falso. Hay contraejemplos para n=3 con 6 orientaciones aciclicas distintas (6 != 9). La idea es elegir un sumidero, apuntar el resto de vertices a ese sumidero y elegir las orientaciones del resto de vertices tal que no haya ciclos 😟?

Pregunta 7 ¿Cuál de las siguientes propiedades sobre grafos es falsa?

- ☐ Si G tiene exactamente 2 vértices con grado impar, tiene que haber un camino entre ellos.
- ☐ El digrafo resultado de darle una orientación a un grafo completo tiene un camino de longitud $n-1$.
- ☐ Si G es un grafo conexo entonces no es un grafo junta. *
- ☐ La existencia de un ciclo en un grafo asegura que el grafo es conexo.

*Un grafo junta J, $J = G + H$ de G y H es el grafo que se obtiene de $G \cup H$ agregando todas las aristas (v, w) posibles entre un vértice $v \in V(G)$ y otro vértice $w \in V(H)$

a. Si G tiene exactamente 2 vertices de grado impar entonces hay un camino entre ellos.
Verdadero. Lo vemos por el contrarreciproco,

d(w) y d(v) impares y no existe camino entre v y w → G no tiene exactamente 2 vertices de grado impar.

De la premisa vale que v y w estan en diferente componente conexa y tienen grado impar. Pero.. puede haber un unico nodo de grado impar en una componente conexa? No.

Porque sabemos que $\sum_{v \in V} d(v) = 2|E|$ y $d(v_o) \equiv 1(mod\ 2)$, entonces

$\sum_{v \in V} d(v) = d(v_0) + \sum_{v \in V-\{v_0\}} d(v) = 2|E|$, luego si miramos las congruencias en

modulo dos vemos: $2|E| \equiv 0\ (mod\ 2)$ y por consecuencia $d(v_0) + \sum_{v \in V-\{v_0\}} d(v) \equiv 0$

con $d(v_0) \equiv 1(mod\ 2)$, luego

$1 + \sum_{v \in V-\{v_0\}} d(v) \equiv 0\ (mod\ 2) \Leftrightarrow \sum_{v \in V-\{v_0\}} d(v) \equiv 1(mod\ 2)$

Pero por hipotesis, d(x) es par para todo x en V\{v0} (porque v0 es el unico nodo de la componente con grado impar), luego esto es imposible. Entonces, no puede haber un unico nodo de grado impar en una componente conexa.

b. El digrafo resultante de darle una orientacion a un grafo completo tiene un camino de longitud N-1.
Verdadero. Creo que es porque d(x)=|V|-1 para todo x en V, entonces hay algun camino que pasa por todos los nodos.

Sale por inducion. ??? ver resubido

c. Si G es un grafo conexo entonces no es un grafo junta.
Falso. Es facil encontrar un contraejemplo en el que G = A U B y G sea conexo.

d. La existencia de un ciclo implica que el grafo es conexo.

Falso. Es facil encontrar un contraejemplo en el que G tenga un ciclo y no sea conexo.

Pregunta 8 ¿Cual/es de las siguientes afirmaciones son correctas sobre el recorrido BFS?

- ☐ Se necesita que el grafo de entrada sea representado con una matriz de adyacencias para alcanzar la complejidad óptima.
- ☐ Todos los vértices a distancia k de la raíz son visitados antes que los de distancia $k+1$.
- ☐ Para cada grafo G hay un único árbol BFS posible.
- ☐ Para toda arista (u, v) del grafo original, o u es ancestro de v en el árbol BFS, o v es ancestro de u.

a. Se necesita que el grafo de entrada sea representado con ..
Falso. Es con lista de adyacencias y la complejidad es optima.

b. Todos los vertices a distancia k de la raiz son visitados antes que los de distancia k+1.
Verdadero.

c. Para cada grafo G hay un unico arbol BFS posible.
Falso. Hay |V| arboles BFS posibles.

d. Para toda arista (u,v) del grafo original, o u es ancestro de v en el arbol BFS o v es ancestro de u.
Falso. Puede pasar que sean ambos descendientes de un ancestro comun. Se encuentra contraejemplo facil. La afirmacion dada vale para DFS.

Supongamos que tenemos el siguiente algoritmo que toma como entrada un digrafo $D = (V, E)$ con n vértices, donde $V = \{0, \ldots, n-1\}$.

Algoritmo 3 Procesar digrafo

```
1:  function F(D)
2:      Inicializar visitados como un arreglo de n elementos, todos False.
3:      suma ← 0
4:      for v ∈ V do
5:          if ¬visitados[v] then
6:              DFS(D, v, visitados)
7:              suma ← suma + 1
8:          end if
9:      end for
10:     return suma
11: end function
```

Pregunta 9 ¿Qué complejidad temporal se ajusta mejor a la del algoritmo 5 en el peor caso?

- ☐ $O(|V| \cdot (|V| + |E|))$
- ☐ $O(|V| \cdot |E|)$
- ☐ $O(|V| + |E|)$
- ☐ $O(|E|^2)$
- ☐ $O(|V|)$

(9) La complejidad mas ajustada es O(|V|+|E|) 👍

Pregunta 10 ¿Qué representará el valor de retorno del algoritmo 5?

- ☐ La cantidad de vértices de D con grado de entrada 0.
- ☐ Ninguna de estas respuestas es correcta.
- ☐ La cantidad de componentes fuertemente conexas de D.
- ☐ La cantidad de componentes conexas de D.
- ☐ La cantidad total de vértices de D.

a) La cantidad de vertices de D con grado de entrada 0.
Falso. En todo caso grado de salida (la #veces que se termina una iteracion del ciclo es la #veces que se encontro con un nodo sin salida (dout(.)=0)), pero tampoco0.

c) La cantidad de componentes fuertemente conexas de D.
Falso, puede ser que A -> B -> C y arranquemos a hacer DFS desde B, luego la suma nos va a dar 2 pero solo hay una componente fuertemente conexa.

c) La cantidad de componentes conexas de D.
Falso, puede ser que A -> B -> C y arranquemos a hacer DFS desde B, luego la suma nos va a dar 2 pero solo hay una componente conexa.

c) La cantidad de vertices de D.
Falso, puede ser que A -> B -> C y arranquemos a hacer DFS desde B, luego la suma nos va a dar 2 pero hay 3 vertices.

2.1. Problema A

Alfredo se está tomando unos días de descanso, así que va a recorrer diferentes pueblos de la Argentina y ha decidido que cuando regrese les regalará un alfajor a cada uno de los k demás docentes de la materia. Para esto ha decidido que los comprará en cada uno de los n pueblos que recorrerá. Al llegar al i-ésimo pueblo tiene tres opciones:

- Comprar media docena (6) de alfajores locales por c_i pesos.
- Comerse un alfajor en lugar de ir a la tienda a comprar.
- No hacer ninguna de las dos anteriores.

Y es que a Alfredo le gustan mucho los alfajores, y siente que si pasa g ciudades sin comerse al menos uno, se estresará mucho, lo cual repercutiría en sus clases. Además sabemos que $g \leq n$.

a) Sea c la lista de precios de los alfajores $c_1, c_2, \ldots, c_n$, definir de forma recursiva $f_{c,n,k,g} : \mathbb{N}^3 \to \mathbb{N}$ donde $f_{c,n,k,g}(i, a, h)$ calcula la cantidad mínima de pesos que Alfredo gastará al partir de la ciudad i con a alfajores y h ciudades que le quedan por recorrer sin comer alfajores antes que le agarre mal humor. ¿Qué llamado(s) se necesitan para resolver el problema? ¿Qué operación se debe realizar sobre esos llamados? **Importante:** acompañen a la definición recursiva con una explicación en castellano.

b) Diseñe e implemente un algoritmo empleando dicha función, mencionando las estructuras que utilizaría. La complejidad del algoritmo debe ser $O(n \cdot \text{mín}\{n, k + \frac{n}{g}\} \cdot g)$ o mejor. (Ayuda: argumentar que no necesita comprar más de $k + \frac{n}{g} + 6$ alfajores).

c) Demuestre que el algoritmo cumple con la propiedad de superposición de problemas, comparando con la solución que emplea backtracking.

d) ¿Es posible usar el algoritmo anterior para determinar en qué pueblos tendría que comprar y en cuáles comerse un alfajor? ¿Es posible hacerlo sin empeorar la complejidad temporal? De ser posible, explique cómo lo haría.

a)

$$
f_{c,n,k,g}(i,a,h) = \begin{cases}
0 & si \quad i = N, h = 0, a - 1 \geq k \\
c_i & si \quad i = N, h = 0, a - 1 + 6 \geq k. \\
+\infty & si \quad i = N, h = 0, a - 1 + 6 < k \\
0 & si \quad i = N, h > 0, a \geq k \\
c_i & si \quad i = N, a < k, a + 6 \geq k \\
+\infty & si \quad i = N, a + 6 < k \\
+\infty & si \quad h < 0 \\
\min(c_i + f(i+1, a+6, h-1), f(i+1, a-1, g), f(i+1, a, h)) & cc.
\end{cases}
$$

b)

```
foo f(i,a,h, memo):
    global c, n, k, g

    if memo[i][a][h]:
        return memo[i][a][h]

    if (i == n):
        if (h == 0):
            if (a − 1 ≥ k):
                res ← 0
            else if (a − 1 + 6 ≥ k):
                res ← c[i]
            else:
                res ← +inf
        else if (h < 0):
            res ← +∞
        else:
            if (a ≥ k):
                res ← 0
            else if (a + 6 ≥ k):
                res ← c[i]
            else:
                res ← +∞
    else:
        compra ← c[i] + f(i + 1, a + 6, h − 1)
        if a>0:
            consumo ← f(i + 1, a − 1, g)
        else:
            consumo ← +∞
        nada ← f(i + 1, a, h − 1)
        res ← min(compra, consumo, nada)

    memo[i][a][h] ← res
    return res
```

La complejidad temporal y espacial es O(#estadosPosibles) porque usamos programacion dinamica. En este caso use un enfoque TopDown.

Esto es $O(N * min\{ N, K + N/G \} * G)$porque…
$i \in [1 \ldots N]$, hay N estados de ciudad actual posibles.
$h \in [1 \ldots G]$, pueden pasar hasta G ciudades sin consumirse alfajores.
$a \in [1 \ldots min\{N, K + N/G\}]$, **

La idea es que esta acotada por min{N, K+N/G} porque como mucho se pueden comprar 6 alfajores por ciudad (6N) y no tiene sentido tener en ninguna instancia (ciudad i, estado h) un a'>K+N/G+6, porque se puede llegar hasta i==n sin comprar ningun alfajor y consumiendo los alfajores restantes con costo 0 (desde esa instancia), mientras que cualquier otra compra aumentaria el costo y siempre "perderia" con la decision de no compra.

c) Demuestre que el algoritmo cumple con la propiedad de superposición de problemas, comparando con la solución que emplea backtracking.

QVQ:

O(#posiblesEstados) << O(#llamadosRecursivos)

Donde:

#posiblesEstados = $O(N * min\{6N, K + N/G\} * G)$

#llamadosRecursivos = $O(3^N)$ como mucho

Pero esta cota no me sirve….
Donde el arbol de backtracking se construye como un arbol 3-ario (no exacto) de N niveles.

Si G=1 → no hay superposicion de problemas (siempre comemos)
Si G=2: #llamadosRecursivos es \Omega(3^{n/4})

d) ¿Es posible usar el algoritmo anterior para determinar en qué pueblos tendría que comprar y en cuáles comerse un alfajor? ¿Es posible hacerlo sin empeorar la complejidad temporal? De ser posible, explique cómo lo haría.

Si! Hay que tener una estructura que vaya guardando las ciudades donde se eligio comprar alfajores (era costo minimo) y otra donde se guarde cuando se elige comer un alfajor. Esto no empeora la complejidad temporal pero si la espacial. Habria que separar el min(..,..,..) en ifs y agregar los valores a las estructuras en los dos casos mencionados.

El resuelto mira la matriz ya procesada para reconstruir las elecciones, tmn se puede hacer tomando elecciones al armar la matriz en O(1).

Demo ej b)

Lema 1: Un DAG tiene al menos un sumidero.

Lema 2: Usando el lema1, vemos por induccion que si no hay ciclos → hay orden de actualizacion seguro

- Como no hay ciclos hay sumidero. Sacamos el sumidero (xq es seguro) y tenemos otro grafo sin ciclos → hay otro sumidero, lo sacamos ,...
- Asi hasta llegar al orden topologico.

c) Algoritmo:

1. Me fijo si hay ciclo **O(n+m)**
2. Tengo una estructura con dOut(.), armo una cola con todos los v | dOut(v) = 0. **O(n+m)**
3. Por cada sumidero en la cola: **v veces**
    a. Saco el sumidero. **O(1)**
    b. Actualizo los grados de todos los padres w del sumidero restando 1. Aca valido si d'(w)=0, si vale esto agrego los w que lo cumplan a la cola de sumideros. **O(d(v))**