Estructuras de Datos y Algoritmos – IIC2133
I3

1. Quicksort

Considera la siguiente descripción recursiva de *quicksort* para ordenar un arreglo S **sin elementos repetidos**:

1. Si el número de elementos en S es 0 o 1, entonces *return*.
2. Elige cualquier elemento v en S; a este elemento le llamamos el pivote.
3. Particiona $S - \{v\}$ en dos grupos disjuntos: $S_1 = \{ x \in S - \{v\} \mid x < v \}$ y $S_2 = \{ x \in S - \{v\} \mid x > v \}$.
4. *return* {quicksort(S_1) seguido de v seguido de *quicksort*(S_2)}.

a) Si S tiene n elementos, ¿cuál es la complejidad de la partición —el paso 3? Justifica.

Respuesta: [2 ptos.] La complejidad es O(n), ya que se deben comparar los n - 1 elementos con el v y cada comparación es O(1).

b) Considera la estrategia de elegir como pivote siempre el primer elemento de S. Demuestra —es decir, argumenta rigurosamente— que esta estrategia puede producir que *quicksort* tenga un desempeño cuadrático con respecto al número, n, de elementos de S.

Respuesta: [2 ptos.]

Dada un lista ordenada S de largo n y v su primer elemento.

Se particiona $S - \{v\}$ en los grupo: $S_1 = \{ x \in S - \{v\} \mid x < v \}$ y $S_2 = \{ x \in S - \{v\} \mid x > v \}$ y como sabemos que $\forall x \in S\ (v < x)$, entonces $|S_1| = 0\ y\ |S_2| = n - 1$.

Como se sabe de a), cada partición es $O(n)$, entonces en cada iteración se tendrán que hacer la siguiente cantidad de iteraciones: $n + (n - 1) + \ldots + 1 + 0 = \sum_{i=0}^{n} (n - i) = \frac{n(n-1)}{3} = O(n^2)$.

Este corresponde al peor caso de quicksort.

c) Explica cómo modificar *quicksort* para convertirlo en un algoritmo para encontrar el k-ésimo elemento más pequeño de S. Este algoritmo debe tomar tiempo O(n) en el caso promedio.

Respuesta: [2 ptos.]

1. Si el número de elementos en S es 0 o 1, entonces *return*.
2. Elige cualquier elemento v en S; a este elemento le llamamos el pivote.
3. Particiona $S - \{v\}$ en dos grupos disjuntos: $S_1 = \{ x \in S - \{v\} \mid x < v \}$ y $S_2 = \{ x \in S - \{v\} \mid x > v \}$.
4. Si la posición de v es igual a k, entonces *return* v.
5. Si es menor a k, entonces $S = S_2$ y $k = k - |S_1|$ y volver a 2.
6. Si es mayor a k, entonces $S = S_1$ y volver a 2.

También estaban correctos otros cambias, pero que fueran O(n).

2. Ordenación topológica

En clase estudiamos un algoritmo para ordenar topológicamente un grafo direccional acíclico $G = (V, E)$; el algoritmo consiste esencialmente en hacer un recorrido DFS del grafo.

a) ¿Cuál es la complejidad del algoritmo estudiado en clase?

Respuesta: [1.5 ptos] La complejidad del algoritmo es de O(|V|+|E|).

Considera ahora el siguiente algoritmo (que fue sugerido por algunos de ustedes en esa misma clase) para ordenar topológicamente un grafo direccional acíclico $G = (V, E)$.

1. Encontrar un vértice que no tenga aristas que lleguen a él.
2. Imprimir el vértice, y sacarlo del grafo junto a todas las aristas que salen de él.
3. Si aún quedan vértices en el grafo, volver a 1.

Si representamos el grafo mediante listas de vértices adyacentes, tal como vimos en clase, responde:

b) Explica claramente cómo implementar el algoritmo anterior —es decir, ¿cómo encuentras un vértice que no tenga aristas que lleguen a él?, y ¿cómo lo "sacas" del grafo junto a todas las aristas que salen de él?

Respuesta: [1.5 ptos] Es correcta cualquier implementación, en la medida que esté bien explicada.

Se descuenta puntaje por detalles en cuanto a la implementación.
Se dio 0 puntos si el algoritmo estaba incorrecto.

c) … y deduce su complejidad en términos del número de vértices, $|V|$, y del número de aristas, $|E|$.

Respuesta: [1.5 ptos] La complejidad debe ser correcta dependiendo de la implementación en (b).

d) Explica cómo mejorar la implementación del algoritmo, en b), de modo que la complejidad sea ahora $O(|V|+|E|)$. En el caso que tu algoritmo en b) ya tenga esa complejidad demuéstrala formalmente.

Respuesta: [1.5 ptos] Una posible forma de hacer que el algoritmo tenga una complejidad de O(|V|+|E|) es agregar, a cada nodo, un contador que indique la cantidad de aristas que llegan hacia el. Al comienzo del algoritmo, setear los valores del contador toma O(|V|+|E|), luego se pueden recorrer todos los nodos, en O(|V|), e ir agregando a una cola (o un *stack*) aquellos nodos que tengan su contador en 0. Luego, se van sacando los nodos de la cola. Cuando se saca un nodo de la cola se debe recorrer su lista de adyacencia y para cada nodo al cual se conecta, disminuir el contador en 1, si el contador llega a 0, agregamos dicho nodo a la cola. El algoritmo termina cuando no hay nodos en la cola. Como solo se está haciendo un recorrido por las listas de adyacencia de cada nodo y sacar y agregar los nodos a la cola toma O(1), el algoritmo posee una complejidad de O(|V|+|E|).

Esta implementación es solo una de las posibles, puede haber más. Lo importante era que tuviese la complejidad solicitada.

3. **MSTs**

Considera el siguiente grafo no direccional con costos, representado matricialmente:

	a	***b***	***c***	***d***	***e***	***f***	***g***
a		2	4	1			
b	2			3	10		
c	4			2		5	
d	1	3	2		7	8	4
e		10		7			6
f			5	8			1
g				4	6	1	

	¿sol?	*dist*	*padre*
a	*F*	0	–
b	*F*	∞	–
c	*F*	∞	–
d	*F*	∞	–
e	*F*	∞	–
f	*F*	∞	–
g	*F*	∞	–

Ejecuta paso a paso el algoritmo de Prim para determinar un árbol de cobertura de costo mínimo, tomando ***a*** como vértice de partida. En cada paso muestra la versión actualizada de la tabla a la derecha, en que *¿sol?* indica si el vértice ya está en la solución: *V* o *F*.

	0		
	¿sol?	dist	padre
a	*F*	0	–
b	*F*	∞	–
c	*F*	∞	–
d	*F*	∞	–
e	*F*	∞	–
f	*F*	∞	–
g	*F*	∞	–

	1		
	¿sol?	dist	padre
a	*V*	0	–
b	*F*	2	a
c	*F*	4	a
d	*F*	1	a
e	*F*	∞	–
f	*F*	∞	–
g	*F*	∞	–

	2		
	¿sol?	dist	padre
a	*V*	0	–
b	*F*	2	a
c	*F*	2	d
d	*V*	1	a
e	*F*	7	d
f	*F*	8	d
g	*F*	4	d

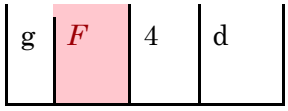
	3		
	¿sol?	dist	padre
a	*V*	0	–
b	*V*	2	a
c	*F*	2	d
d	*V*	1	a
e	*F*	7	d
f	*F*	8	d
g	*F*	4	d

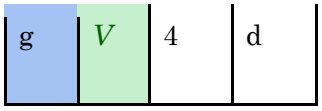
	4		
	¿sol?	dist	padre
a	*V*	0	–
b	*V*	2	a
c	*V*	2	d
d	*V*	1	a
e	*F*	7	d
f	*F*	5	c

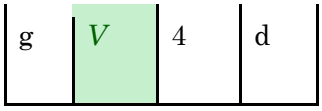
	5		
	¿sol?	dist	padre
a	*V*	0	–
b	*V*	2	a
c	*V*	2	d
d	*V*	1	a
e	*F*	6	g
f	*F*	1	g

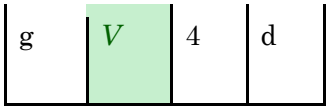
	6		
	¿sol?	dist	padre
a	*V*	0	–
b	*V*	2	a
c	*V*	2	d
d	*V*	1	a
e	*F*	6	g
f	*V*	1	g

	7		
	¿sol?	dist	padre
a	*V*	0	–
b	*V*	2	a
c	*V*	2	d
d	*V*	1	a
e	*V*	6	g
f	*V*	1	g









- 0.85 ptos por cada paso correcto (si no mencionó la columna padre en todas las etapas, entregar puntaje igual) del 1 al 7 (el paso 0 no daba puntaje)
- Si ejecutó el algoritmo incorrectamente pero la tabla final es correcta se entregará el máximo entre 3 y 0.85*N pasos correctos
- Nota, el paso 3 y 4 son intercambiables, no importaba el orden en que se hicieran

4. Rutas más cortas / programación dinámica

I. **(3pts)** El canal DCC TV tiene un nuevo programa llamado EDD donde los participantes tienen la chance de concursar y ganar dinero en premios. El juego consiste en un triángulo de pelotas, donde cada pelota tiene impreso un número íntegro, tal como se muestra en la figura.

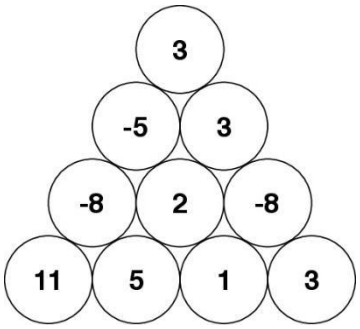


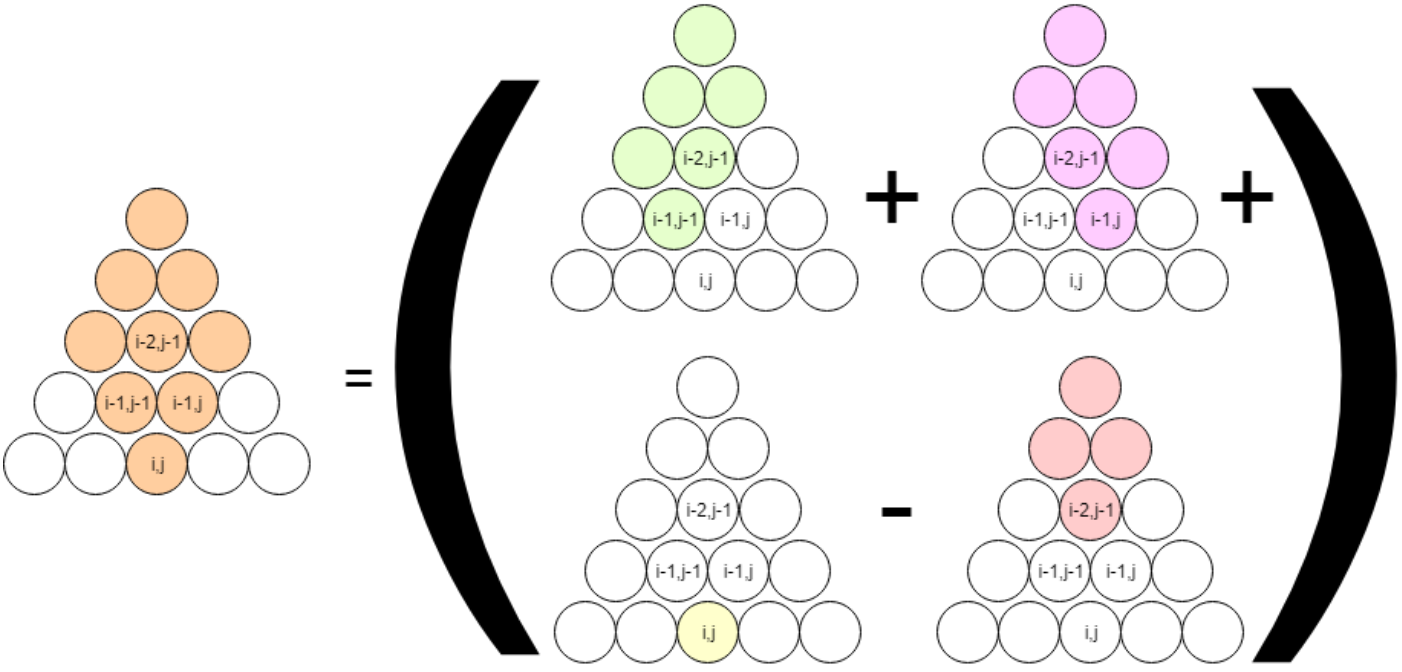
Figura 1

El jugador puede elegir una pelota de la pirámide y sacarla o no sacar ninguna. En caso de que saque una pelota, se quedará con los puntos de esa pelota y **todas las que estén arriba de ella**. En caso de que saque ninguna, se quedará con 0 puntos. En la figura 1, para sacar la pelota 1, se necesitan sacar también las pelotas 2, -8, -5, 3 y 3 por lo que el valor de sacar la pelota 1 sería -4. El presentador está preocupado por el máximo premio que se pueda llevar un concursante, por lo que se te pide ayuda a ti para resolver este problema.

Esta pirámide se representa como arreglo de arreglos M donde M[0] es el nivel superior y M[n] es el nivel de más abajo. Cada nivel M[i] tiene i+1 elementos.

a) Haz la ecuación de recurrencia que permite calcular el premio obtenido de sacar una pelota específica: *P(i, j, M)* donde la pelota elegida es M[i][j].

Respuesta: [1.5 pts] Se acepta como respuesta correcta una función de recurrencia de forma matemática o un pseudocódigo que exprese la ecuación siguiente:



Esto es:

$$p(i,j,M) = \begin{cases} 0 & , \quad i,j < 0 \ o \ j > i \\ M[i][j] + p(i-1,j-1,M) + p(i-1,j,M) - p(i-2,j-1,M), & \quad else \end{cases}$$

Si falta 1 término de la ecuación (normalmente falta el *-p(i-2, j-1, M)*), se da 1 punto de 1.5

b) Crea una solución iterativa de este problema en la cual se calcula el premio para toda pelota de la pirámide (Versión bottom-up).

Respuesta: [1.5 pts] Para hacer una respuesta iterativa para toda la pirámide debo crear una tabla R que contenga las respuestas e implementarlo de la siguiente forma:

```
premios(M):
    R = Tabla de las mismas dimensiones de M
    for i = 0..n:
        for j = 0..i:
            R[i][j] = M[i][j]
            if i > 0:
                if j > 0:
                    R[i][j] += R[i-1][j-1]
                if j < i:
                    R[i][j] +=R[i-1][j]
                if i - 2 >= 0 and j > 0:
                    R[i][j] -=R[i-2][j-1]
    return R
```

Se acepta como respuesta correcta la implementación iterativa del problema modelado en 4a) independiente de si es correcto para el problema.

II. **(3pts)** La gran ventaja del algoritmo de Floyd-Warshall frente a Dijkstra es que permite tener las rutas precalculadas y almacenadas. El tiempo que toma en construir la matriz de costos mínimos entre todos los pares de nodos es O(|□|^3). Digamos el algoritmo de Floyd-Warshall es muy complicado de entender por lo que queremos utilizar el algoritmo de Dijkstra para calcular esta matriz de costos mínimos. Calcule la complejidad de este método.
Considere que el algoritmo de Dijkstra toma tiempo O(|E| log(|V|)) en su implementación con un min heap como cola de prioridad.

Respuesta: [3 pts] Para rellenar la matriz de distancias mínimas hace falta obtener los valores de min_dist(i, j) para todos los pares de nodos i, j. Dijkstra permite obtener el valor de min_dist desde un nodo i hasta todos los nodos del grafo, por lo que con una llamada a Dijkstra se rellena una fila de la matriz. Por lo tanto solo es necesario usar Dijkstra |V| veces. Esto tiene un costo de O(|V||E|log(|V|)).

Se dio 0 puntos por usar Dijkstra por cada par de nodos.

Se dio 2 puntos por dar mal la complejidad a pesar de haber explicado bien el procedimiento.