Taller Pre-Parcialito 4 [05/12] - Miércoles

1. Aplicar (y explicar) el algoritmo de *Dijkstra* para encontrar el camino mínimo desde A hacia el resto de los vértices en el siguiente grafo.

	A	**B**	**C**	**D**	**E**	**F**	**G**
a1	-1	1	0	0	0	0	0
a2	-4	0	4	0	0	0	0
a3	-9	0	0	9	0	0	0
a4	0	3	0	0	-3	0	0
a5	0	0	-7	0	7	0	0
a6	0	0	-2	0	0	2	0
a7	0	0	0	6	0	-6	0
a8	0	0	0	0	-9	0	9
a9	0	0	0	0	-1	1	0
a10	0	0	0	0	5	-5	0
a11	0	0	0	0	0	-8	8

2. Aplicar el algoritmo de *Kruskal* para obtener el árbol de tendido mínimo del siguiente grafo, mostrando cómo se modifican las estructuras auxiliares en cada iteración. **a)** Aplicar al mismo grafo el algoritmo de *Prim*.

	A	**B**	**C**	**D**	**E**	**F**
A	0	4	2	5	0	7
B	4	0	6	0	6	0
C	2	6	0	1	4	0
D	5	0	1	0	3	8
E	0	6	4	3	0	8
F	7	0	0	8	8	0

3. Se tiene un arreglo con n enteros positivos, y se desea separar todos sus elementos en dos conjuntos disjuntos, tales que la diferencia entre la suma de los elementos de cada conjunto sea mínima. Diseñar un algoritmo *greedy* para formar los dos conjuntos.

Indicar el orden de la solución.

4. El siguiente algoritmo encuentra una farmacia abierta:

(1) Tengo un papel en donde voy a ir anotando la lista de farmacias visitadas, inicialmente vacía.
(2) Ir a la farmacia más cercana que no esté en la lista.
(3) Si está de turno, llegué.
(4) Sino, anotar la farmacia en la que estoy, y repetir desde (2).

¿Qué tipo de técnica de diseño de algoritmos se está implementando? ¿Por qué?

¿Encuentra siempre alguna solución?, y si la encuentra, ¿Podemos estar seguros siempres de que es la óptima?

5. El problema del recorrido del caballo de ajedrez consiste en indicar una serie de 64 movimientos para que un caballo que se encuentra en una dada posición del tablero pise todas las casillas sin repetir ninguna.

Escribir el pseudocódigo de una función `bool recorrido_del_caballo(A, x, y, i)` que resuelva este problema usando *backtracking* sabiendo que:

- El parámetro A es una matriz de enteros de 8x8.
- Los parámetros x e y indican la posición actual del caballo.
- i es el número de paso actual.
- La función devuelve true o false de acuerdo a si se encontró un recorrido o no.
- Se tiene una función `obtener_movimientos(x, y)` que dada una posición devuelve una lista con todos los movimientos posibles, descartando sólo aquellos que se excedan del tablero.

Inicialmente la función se llama con A llena de ceros, algún par de puntos x e y (entre 0 y 8) e i igual a 0. **Si la función devuelve true debe completar la matriz con el orden en el cual se visita cada casilla.**

6. Escribir una función en pseudocódigo que, utilizando *backtracking*, dada una lista de enteros positivos L y un entero n devuelva todos los subconjuntos de L que suman exactamente n.

Implementar la misma función, ahora usando *programación dinámica*.

7. Dado un arreglo de n números reales, se desea encontrar el valor máximo posible para la suma de una ventana contigua. Por ejemplo, para el arreglo `[3 -1 -5 6 7 -9]`, el valor máximo es 13, que se obtiene sumando 6 + 7. Para resolverlo usando *programación dinámica*, se define M[i] como el valor máximo de ventana que se puede obtener utilizando los primeros i elementos del arreglo. Para cada valor de i, el valor de M[i] se puede formar extendiendo la ventana que termina en i - 1, o bien comenzando una ventana nueva en i.

Mostrar paso por paso la resolución para el arreglo `[-15 29 -36 3 -22 11 19 -5 0 4]`.

¿De qué orden es el algoritmo en tiempo y memoria?

8. En este barrio todo viven alrededor de la plaza y los vecinos contiguos se odian entre sí. Hay N vecinos y el primer vecino es contiguo con el último. Cuando le preguntan a cada uno cuanto es lo máximo que va a donar para arreglar la plaza se respuesta se registra en el array *donar* = [d0, d1, .., d(N-1)], pero la condición es: sólo donaré si no dona ninguno de mis vecinos contiguos. Usando programación dinámica resolver cuánto es lo máximo que se puede recolectar por donaciones. Indicar y justificar el orden del algoritmo.

Nota: Lo resolveremos considerando que el primer vecino y el último no son contiguos. Si sobra tiempo se resolverá considerando el caso circular.

9. Se tiene un cable de largo n y una tabla de precios de cable según el largo. Sabiendo que se lo puede dividir para aumentar el valor, implementar un algoritmo que indique cuál es el mayor valor que se le puede sacar al cable usando *programación dinámica*.

Indicar y justificar el orden del algoritmo.

Por ejemplo, si P(1) = \$1, P(2) = \$5, P(3) = \$8 y P(4) = \$9, para n = 4 el algoritmo debe devolver \$10 ya que se obtendría su máximo valor cortándolo a la mitad.

Nota: se puede dividir el cable más de una vez, o inclusive ninguna.