Problemas: Tema 2

Problema 1

Se tienen n números naturales, siendo n una cantidad par, que tienen que juntarse formando parejas de dos números cada una. A continuación, de cada pareja se obtiene la suma de sus dos componentes, y de todos estos resultados se toma el máximo.

Diseñar un algoritmo voraz que cree las parejas de manera que el valor máximo de las sumas de los números de cada pareja sea lo más pequeño posible, demostrando que la función de selección de candidatos usada proporciona una solución óptima.

Ejemplo: suponiendo que los datos se encuentran en el vector siguiente

5	8	1	4	7	9

vamos a ver un par de formas de resolver el problema (no necesariamente la óptima):

Seleccionamos como pareja los elementos consecutivos

De esta forma conseguimos las parejas (5, 8), (1, 4) y (7, 9); entonces, al sumar las componentes tenemos los valores 15, 5 y 16, por lo que el resultado final es 16.

Seleccionamos como pareja los elementos opuestos en el vector

Ahora tenemos las parejas (5, 9), (8, 7) y (1, 4); sumando conseguimos 14, 15 y 5, por lo que el resultado final es 15 (mejor que antes).

¿Habrá un resultado mejor para este problema? ¿Puede generalizarse un método que nos proporcione un algoritmo voraz correcto para cualquier cantidad de datos, y que además sea independiente del valor de los mismos?

Problema 2

Se tiene que almacenar un conjunto de n ficheros en una cinta magnética (soporte de almacenamiento de recorrido secuencial), teniendo cada fichero una longitud conocida l_1, l_2, …, l_n. Para simplificar el problema, puede suponerse que la velocidad de lectura es constante, así como la densidad de información en la cinta.

Se conoce de antemano la tasa de utilización de cada fichero almacenado, es decir, se sabe la cantidad de peticiones pi correspondiente al fichero i que se van a realizar. Por tanto, el total de peticiones al soporte será la cantidad $\boldsymbol{P} = \sum_{\boldsymbol{i=1}}^{\boldsymbol{n}} \boldsymbol{p_i}$. Tras la petición de un fichero, al ser encontrado la cinta es automáticamente rebobinada hasta el comienzo de la misma.

El objetivo es decidir el orden en que los n ficheros deben ser almacenados para que se minimice el tiempo medio de carga, creando un algoritmo voraz correcto.

Problema 3

Se dispone de un vector V formado por n datos, del que se quiere encontrar el elemento mínimo del vector y el elemento máximo del vector. El tipo de los datos que hay en el vector no es relevante para el problema, pero la comparación entre dos datos para ver cuál es menor es muy costosa, por lo que el algoritmo para la búsqueda del mínimo y del máximo debe hacer la menor cantidad de comparaciones entre elementos posible.

Un método trivial consiste en un recorrido lineal del vector para buscar el máximo y después otro recorrido para buscar el mínimo, lo que requiere un total de aproximadamente 2n comparaciones entre datos. Este método no es lo suficientemente rápido, por lo que se pide implementar un método con metodología Voraz que realice un máximo de $\frac{3}{2}\boldsymbol{n}$ comparaciones.

Problema 4

Se tiene un grafo no dirigido G = < N, A >, siendo N = {1, …, n} el conjunto de nodos y A ⊆ NxN el conjunto de aristas. Cada arista (i, j) ∈ A tiene un coste asociado c_{ij} ($c_{ij} > 0$ ∀i, j ∈ N; si (i, j) ∉ A puede considerarse $c_{ij} = +\infty$). Sea M la matriz de costes del grafo G, es decir, M[i, j] = c_{ij}. (al ser el grafo no dirigido se tiene que (i, j) = (j, i) por lo que la matriz M es simétrica).

Teniendo como datos la cantidad de nodos n y la matriz de costes M, se pide encontrar el árbol soporte mínimo del grafo G utilizando el algoritmo de Prim, utilizando las siguientes ideas:

- A diferencia del algoritmo de Kruskal (que crea el árbol utilizando componentes conexas independientes que se van uniendo entre sí), el algoritmo de Prim se basa en la idea de ir construyendo un árbol cada vez más grande, empezando por un único nodo y acabando por recubrir todo el grafo.
- El algoritmo comienza con un árbol de un nodo, al que se le añade un segundo nodo, luego un tercero, etc, hasta tener los n nodos unidos. La forma de escoger un nodo es buscando el nodo más cercano a todo el árbol, sin que se creen ciclos.
- A medida que crece el tamaño del árbol la búsqueda del nodo más cercano se complica, por lo que para que el algoritmo sea eficiente (el método debe tener $O(n^2)$) hay que crear un estructura de datos que almacene la mejor distancia de cada nodo al conjunto de nodos del árbol.
- Se necesitará almacenar de alguna manera la forma en que se ha creado el árbol, por ejemplo indicando a qué nodo del árbol se está uniendo el nuevo candidato seleccionado.

Problema 5

Se tiene un grafo dirigido $G = < N, A >$, siendo $N = \{1, \ldots, n\}$ el conjunto de nodos y $A \subseteq NxN$ el conjunto de aristas. Cada arista $(i, j) \in A$ tiene un coste asociado c_{ij} ($c_{ij} > 0$ $\forall i, j \in N$; si $(i, j) \notin A$ puede considerarse $c_{ij} = +\infty$). Sea M la matriz de costes del grafo G, es decir, $M [i, j] = c_{ij}$.

Teniendo como datos la cantidad de nodos n y la matriz de costes M, se pide encontrar tanto el camino mínimo entre los nodos 1 y n como la longitud de dicho camino usando el algoritmo de Dijkstra, utilizando las siguientes ideas:

- Crear un estructura de datos que almacene las distancias temporales conocidas (inicializadas al coste de la arista del vértice 1 a cada vértice j, o $+\infty$ si no existe dicha arista) para los vértices no recorridos (inicialmente, todos salvo el 1).
- Seleccionar como candidato el que tenga menor distancia temporal conocida, eliminarle del conjunto de vértices no recorridos, y actualizar el resto de distancias temporales si pueden ser mejoradas utilizando el vértice actual.
- Se necesitará almacenar de alguna manera la forma de recorrer el grafo desde el vértice 1 al vértice n (no necesariamente igual al conjunto de decisiones tomadas).

Problema 6

Shrek, Asno y Dragona llegan a los pies del altísimo castillo de Lord Farquaad para liberar a Fiona de su encierro. Como sospechaban que el puente levadizo estaría vigilado por numerosos soldados se han traído muchas escaleras, de distintas alturas, con la esperanza de que alguna de ellas les permita superar la muralla; pero ninguna escalera les sirve porque la muralla es muy alta. Shrek se da cuenta de que, si pudiese combinar todas las escaleras en una sola, conseguiría llegar exactamente a la parte de arriba y poder entrar al castillo.

Afortunadamente las escaleras son de hierro, así que con la ayuda de Dragona van a "soldarlas". Dragona puede soldar dos escaleras cualesquiera con su aliento de fuego, pero tarda en calentar los extremos tantos minutos como metros suman las escaleras a soldar. Por ejemplo, en soldar dos escaleras de 6 y 8 metros tardaría 6 + 8 = 14 minutos. Si a esta escalera se le soldase después una de 7 metros, el nuevo tiempo sería 14 + 7 = 21 minutos, por lo que habrían tardado en hacer la escalera completa un total de 14 + 21 = 35 minutos.

Diseñar un algoritmo eficiente que encuentre el mejor coste y manera de soldar las escaleras para que Shrek tarde lo menos posible es escalar la muralla, indicando las estructuras de datos elegidas y su forma de uso. Se puede suponer que se dispone exactamente de las escaleras necesarias para subir a la muralla (ni sobran ni faltan), es decir, que el dato del problema es

la colección de medidas de las “miniescaleras” (en la estructura de datos que se elija), y que solo se busca la forma óptima de fundir las escaleras.

Entregables: un ejercicio a elegir entre los problemas 2 y 3, un ejercicio a elegir entre los problemas 4 y 5 y el problema 6.