Examen Algoritmia 16/06/2014

En los ejercicios 1 y 2 debes justificar en cada opción si es verdadero o falso.

1.- (2 puntos) Consideramos el problema de la mochila 0/1 (no se pueden introducir fracciones de objetos). Utilizamos el algoritmo voraz en el que primero se ordenan de forma decreciente los objetos por Beneficio/tamaño. Después se introducen los K primeros objetos donde la suma de sus tamaños es casi tan grande como la capacidad de la mochila, de tal forma que el K+1 objeto ya no cabe. ¿Cuál de las siguientes afirmaciones es verdad?

a) Este algoritmo siempre da una solución con valor total al menos el 50% de la solución óptima.
   Falso. Contraejemplo: Tamaño de la mochila = 3, dos objetos w1 = 1, b1 = 2, w2 = 3, b2 = 5. El algoritmo voraz seleccionaría el objeto 1 ya que tiene ratio 2, y después ya no cabría ningún objeto más). Sin embargo la solución óptima es seleccionar el objeto 2, que cabe en la mochila y tiene beneficio 5)
b) Si todos los objetos tienen el mismo ratio beneficio/tamaño, entonces el algoritmo siempre devuelve la solución óptima (no importa cómo se resuelvan los empates).

   Falso. Contraejemplo: Tamaño de la mochila =4. Tres objetos w1=3, b1 = 3, w2 = 2, b2 = 2, w3 = 2, b3 = 2. Dependiendo de cómo se resuelven los empates no se alcanza la solución óptima.

c) Si todos los objetos tienen el mismo tamaño, entonces el algoritmo devuelve la solución óptima.

   Verdadero. Si todos los objetos tienen el mismo tamaño y están ordenados por el ratio beneficio/ tamaño, todos los objetos que se introducen en la mochila son los de mayor beneficio, por tanto no puede haber otra combinación de objetos que de mayor beneficio. Es decir el algoritmo devuelve la solución óptima.

d) Si todos los objetos tienen el mismo beneficio, entonces el algoritmo devuelve la solución óptima.

   Verdadero. Si todos los objetos tienen el mismo beneficio y están ordenados por el ratio beneficio/ tamaño, se introducen en la mochila el mayor número de objetos para ese tamaño de mochila. Por tanto no puede haber otra combinación de objetos que de mayor beneficio en el mismo tamaño de mochila. Es decir el algoritmo devuelve la solución óptima.

2.- (2 puntos) En clase hemos visto un algoritmo de programación dinámica para el problema del viajante. ¿Este algoritmo implica que P=NP?

a) Sí, eso es.

   Falso. El algoritmo que vimos en clase para resolver el problema del viajante era del orden O(n2n2). Por tanto es una solución no polinomial, es decir de ningún modo implica de N=NP.

b) No. Ya que algunas veces el algoritmo se ejecuta durante un tiempo super-polinomial para resolver un único subproblema complicado, y por tanto no se ejecuta en tiempo polinomial.

Falso. El algoritmo se ejecuta en tiempo polinomial porque tiene un número exponencial de subproblemas.

c) No, un algoritmo que resuelve el problema del viajante de tiempo polinomial no implica que P = NP.

Falso. Si encontrásemos un algoritmo que resuelva el problema del viajante en tiempo polinomial, como este problema pertenece al conjunto de problemas NP-Completos, entonces implicaría que P=NP.

d) No. Ya que existen un número exponencial de suproblemas en ese algoritmo y por tanto no se ejecuta en tiempo polinomial.

Verdadero. En el algoritmo que vimos en clase conforme desarrollamos las llamadas recursivas se van generando subproblemas de forma exponencial.

3.- (2 puntos) Diseña y analiza la eficiencia de un algoritmo "divide y vencerás" para medir la similitud entre dos rankings. Muchos sitios web intentan comparar las preferencias de dos usuarios para realizar sugerencias a partir de las preferencias de usuarios con gustos similares a los nuestros. Dado un ranking de n productos (p.ej. canciones) mediante el cual los usuarios indicamos nuestras preferencias, un algoritmo puede medir la similitud de nuestras preferencias contando el número de inversiones: Dos productos i y j están "invertidos" en las preferencias de A y B si el usuario A prefiere el producto i antes que el j mientras que el usuario B prefiere el producto j antes que el i. Esto es, cuantas menos inversiones existan entre dos rankings, más similares serán las preferencias de los usuarios re presentados por esos rankings.

Imaginemos que un usuario tiene el siguiente ranking de canciones:

1 Like a Rolling Stone, Bob Dylan

2 (I Can't Get No) Satisfaction, The Rolling Stones

3 Imagine, John Lennon

4 What's Going On, Marvin Gaye

5 Respect , Aretha Franklin

6 Good Vibrations, Beach Boys

7 Johnny B. Goode, Chuck Berry

8 Hey Jude, The Beatles

9 Smells Like Teen Spirit, Nirvana

10 What'd I Say, Ray Charles

Sin embargo otro usuario tiene el siguiente ranking:

Respect, Aretha Franklin

What'd I Say, Ray Charles

Imagine, John Lennon

What's Going On, Marvin Gaye

Like a Rolling Stone, Bob Dylan

(I Can't Get No) Satisfaction, The Rolling Stones

Good Vibrations, Beach Boys

Johnny B. Goode, Chuck Berry

Hey Jude, The Beatles

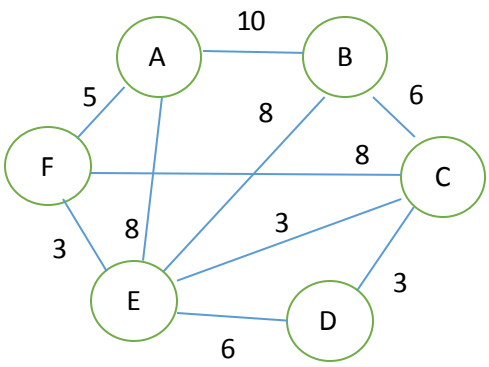
Smells Like Teen Spirit, Nirvana

Utilizamos la primera lista como referencia, por lo tanto like a Rolling Stone sería la canción número 1 y What'd I Say sería la 10. Siguiendo esta numeración la lista del segundo usuario sería: 5, 10, 3, 4, 1, 2, 6, 7, 8, 9

Un algoritmo Divide y Vencerás para contar las inversiones dentro de un vector es una modificación de mergesort que ya implementamos en prácticas. Simplemente en la función merge añadimos un contador en el que cada vez que un elemento del segundo vector (previamente hemos llamado a mergesort con las dos mitades del vector a ordenar) sumamos las inversiones de ese elemento con el primer vector. En las transparencias "inversiones.pdf" está explicado la forma de cálculo de las inversiones.

Por tanto al utilizar mergesort, como ya probamos en clase estaremos utilizando un algoritmo de complejidad O(nlogn).

4.- (2 puntos) Imagina que estás trabajando para la agencia de seguridad nacional y estás espiando una red terrorista. Puedes acceder a las llamadas telefónicas que se realizan entre ellos, pero no puedes escuchar lo que dicen, simplemente tienes la notificación de los dos números de teléfono entre los que se han hecho las llamadas. Para organizar la información creas un grafo en el que cada nodo es un número de teléfono y el arco que hay entre cada nodo contiene el número de llamadas que se han hecho entre esos dos nodos. Identifica las llamadas (arcos) que intervendrías para escuchar el máximo número de comunicaciones entre todos los miembros de la banda. Ten en cuenta que debes seleccionar el menor número de arcos ya que tienes muy pocos espías a tu cargo. ¿Qué algoritmo utilizarías? Utilizando el grafo de la figura muestra los pasos que seguirá tu algoritmo y cuál es la solución final.



Si queremos escuchar el máximo número de comunicaciones entre todos los miembros de la banda y además tenemos muy pocos espías, tendremos que seleccionar los arcos (comunicaciones) que conecten todos los nodos (terroristas). Es decir tenemos que calcular un árbol recubridor, pero en este caso será de peso máximo. En clase hemos visto varios algoritmos para encontrar el árbol recubridor mínimo. La solución más sencilla es utiliza el algoritmo de Kruskal, pero ordenando los arcos de mayor a menor coste.

Los pasos serían así:

1.- Ordenar los arcos de forma decreciente.

2.- Ir seleccionando los arcos hasta que el árbol conecte todos los nodos, sin que formen ciclos.

Lista ordenada: {A,B} {B,E}{A,E}{F,C}{B,C}{E,D}{A,F}{C,E}{C,D}{F,E}

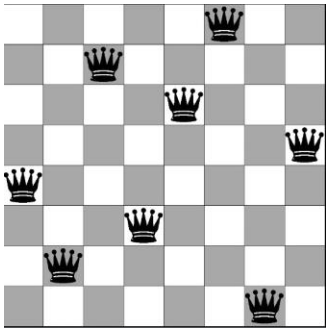
1. T = {A,B}
2. T = {A,B}, {B,E}
3. T = {A,B},{B,E}, {F,C}
4. T = {A,B},{B,E}, {F,C}, {B,C}
5. T = {A,B},{B,E}, {F,C}, {B,C}, {E,D}

Por tanto el coste total del árbol recubridor máximo será de 38.

5.- (2 puntos) Formas parte de una consultora que se dedica a organizar fábricas para maximizar la productividad. En este caso el problema es el siguiente: la fábrica tiene 20 máquinas y 8 trabajadores. Las máquinas tienen restricciones entre sí, por ejemplo: si la máquina 1 está en funcionamiento, entonces las 2 no puede funcionar y la tres es obligatorio que funcione. Existen muchas restricciones de ese tipo. Dependiendo de las máquinas que estén funcionando obtendremos el beneficio. Lo que queremos es asignar los 8 trabajadores a las máquinas para que maximicen el beneficio.

Explica dos algoritmos diferentes con los que podrías resolver este problema, cuáles serían sus ventajas e inconvenientes.

El problema de la asignación de recursos con restricciones es similar al problema de las 8-reinas que vimos en clase. En ese problema debemos colocar las reinas las casillas del tablero de ajedrez de tal forma que no se den jaque entre ellas. En el problema de este ejercicio debemos colocar a los trabajadores en las máquinas de tal forma que se cumplan las restricciones y además obtengamos un beneficio máximo.



Existen varios algoritmos para afrontar el problema:

Fuerza bruta: probar todas las combinaciones de los trabajadores en las máquinas y escoger la que cumpla todas las restricciones y además tenga un beneficio máximo.

Algoritmo voraz: un criterio voraz sería ordenar las máquinas en orden decreciente de beneficio e ir colocando los trabajadores, empezando por la de beneficio máximo y continuando por las siguientes que hagan que se cumplan las restricciones.

Búsqueda: podemos resolver este problema mediante una búsqueda en árbol. De forma similar a como hicimos en el problema de las n-reinas, comenzaríamos colocando a un trabajador en una máquina y expandiríamos el árbol. Los siguientes nodos del nodo raíz representarían al segundo trabajador colocado en cada una de las posibles máquinas en las que se cumplan las restricciones. Podríamos expandir el árbol en anchura o en profundidad hasta encontrar una solución. O podemos expandir el árbol completo para seleccionar la solución óptima. También se puede crear una cota para podar el árbol y no perder tiempo en las ramas que no son prometedoras.

Probabilista: podríamos hacer un algoritmo probabilista en el que asigna trabajadores a las máquinas al azar. Sería un algoritmo tipo las vegas* en la que algunas veces no devuelve una solución, ya que no se cumplen las restricciones. (* En realidad tampoco es las vegas porque no es seguro que devuelva el beneficio máximo). Ejecutaríamos el algoritmo varias veces y nos quedaríamos con la mejor solución.

Las ventajas e inconvenientes (complejidad, si encuentran o no la solución óptima, implementación) de todos estos algoritmos los podéis ver en los apuntes.