TEORÍA DE ALGORITMOS
RELACIÓN DE PROBLEMAS
TEMA 4. ALGORIMOS GREEDY

1. Se tiene un barco de mercancías cuya capacidad de carga es de k toneladas y un conjunto de contenedores c1,...., cn, cuyos pesos respectivos (expresados también en toneladas) son p1, ..., pn. Se sabe que la capacidad del barco es <u>inferior</u> a la suma total de los pesos de los contenedores.

   a) Diseñar un algoritmo que maximice el número de contenedores cargados.
   b) Diseñar un algoritmo para maximizar el número de toneladas cargado.

   Indicar las técnicas de diseño utilizadas en ambos problemas y analizar su complejidad.

2. Sean n programas P1, ..., Pn que hay que almacenar en un disco. El programa Pi requiere S_i Kilo bites de espacio y la capacidad del disco es D Kilo bites, donde $D < \sum S_i$. Resolver las siguientes cuestiones:

   - Se desea maximizar el número de programas almacenados en el disco. Construir un algoritmo greedy para resolver el problema (existe un algoritmo que da una solución óptima).
   - Se desea utilizar la mayor capacidad posible del disco. Demostrar que podemos utilizar un algoritmo greedy que seleccione los programas por orden no creciente de sí para obtener la solución exacta o dar un contraejemplo en caso contrario.

3. Tenemos que ejecutar un conjunto de n tareas, cada una de las cuales requiere un tiempo unitario. En un instante T = 1, 2, ... podemos ejecutar únicamente una tarea. La tarea i produce unos beneficios gi (gi > 0) sólo en el caso en el que sea ejecutada en un instante anterior o igual a di. Utilizando la técnica greedy encontrar un algoritmo que nos permita seleccionar el conjunto de tareas a realizar de forma que nos aseguremos que tenemos la mayor ganancia posible.

   Resolver el problema utilizando la siguiente instancia:

i	1	2	3	4
gi	50	10	15	30
di	2	1	2	1

4. Consideremos el grafo no-dirigido G = (V, E). Un conjunto de nodos U se dice un cubrimiento de G si U es un subconjunto de V tal que cada arista en E es incidente al menos en un vértice de U. Un conjunto de nodos es un cubrimiento minimal de G si tiene el mínimo número posible de nodos y verifica esta propiedad. El grado de incidencia de un vértice es el número de aristas que inciden en él.

   a) Dar un algoritmo greedy para este problema.

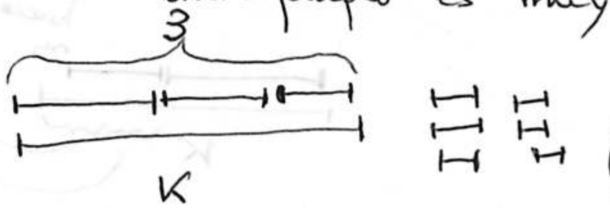
   b) Indicar si este algoritmo genera un conjunto de vértices minimal para cualquier grafo si se utiliza como función de selección el máximo grado de incidencia sobre los nodos no seleccionados.

NOTA: La forma de resolver estos problemas es pensando en posibles funciones de selección y factibilidad, pensar en posibles contraejemplos que hagan que el algoritmo no de una solución óptima y finalmente quedarse con la opción más lógica dando alguna razón (incluso aunque ello no asegure una solución óptima/correcta, salvo que se permita usar otro tipo de técnica que si la dé).

(1) Ej. 1. de la relación.

a) - Lista de candidatos C: el conjunto de los contenedores
- Lista de seleccionados S: solución parcial con los contenedores seleccionados hasta el momento
- Funcion de factibilidad: Si peso$(S \cup \{x\}) <= K$ entonces $S \cup \{x\}$ es factible.
- Función solución: Cuando no queden elementos/candidatos factibles.
- Función objetivo: nº de elementos en S.
- Función selección: Hay dos posibilidades, coger el siguiente elemento de menor peso o coger el siguiente de mayor peso.

La segunda parece poco lógica porque deja el menor espacio posible para colocar más contenedores. De hecho, encontrar un contraejemplo es muy sencillo,



se ve claramente que caben más

Escogeremos por tanto el de menor peso disponible.

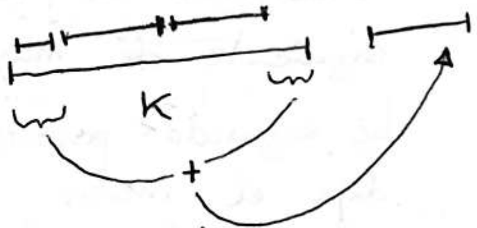
Algoritmo:

```
Maxcont(C, S) {
    S <- Ø
    P <- 0
    mientras (C ≠ Ø) hacer {          // n veces
        x <- buscar_minimo(C)         // n, n-1, n-2 ...
        C <- C - {x}
        Si (p + peso(x) <= K) {       } O(1)        => O(n^2)
            S <- S U {x}
            P <- p + peso(x)
        }
    }
}
```

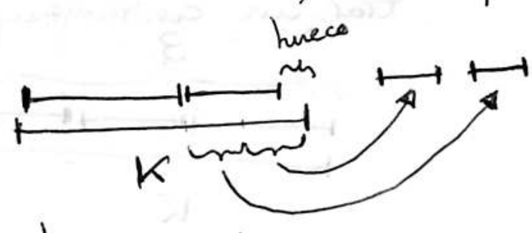
Sin embargo, en este caso se puede realizar una ordenación previa para simplemente ir seleccionando los contenedores mientras quepan. Este algoritmo sería
$n \cdot lg(n) + n \Rightarrow O(n \cdot lg(n))$.
Queda como ejercicio describir/proponer el algoritmo que usando mergesort o quicksort (éstos no habría que describirlos) dé una solución al problema.

b) De las dos posibilidades anteriormente comentadas, para este problema ninguna nos lleva a un algoritmo correcto mediante el enfoque Greedy. Es fácil encontrar contraejemplos:

1. Escoger el menor disponible:



2. Escoger el mayor disponible:



Este apartado está puesto para que se responda que lo mejor es utilizar una técnica de ramificación y poda, y se describa dicho algoritmo.

2) Ej. 2 de la relación.

a) Se resuelve como el apartado a) del ejercicio anterior (es el mismo problema).

b). Es el mismo problema del apartado b) del ejercicio anterior. Bastaría con poner el 2º contraejemplo.

---

3 Ej 3 de la relación.

- Lista de candidatos C: el conjunto de las tareas
- Lista de seleccionados S': las tareas seleccionadas hasta el momento.
- Función solución: cuando no queden más tareas factibles.
- Función objetivo: Beneficios acumulados por las tareas en S'. Beneficio $= \sum_{i \in S} g_i$.
- Función de factibilidad: $(S' \cup \{x\})$ es factible si todas las tareas $i \in (S' \cup \{x\})$ pueden ser ejecutadas en un instante anterior o igual a d_i.
- Función de selección: La mejor opción es coger las tareas con mayor beneficio. Se pueden encontrar contraejemplos fáciles para otras alternativas lógicas. Por ejemplo, escoger primero aquellas que deban ser ejecutadas antes, contraejemplo:

i	1	2	3	4	5
g_i	10	5	50		
d_i	2	2	3		

g_i 30, d_i 3: ésta ya no entraría, sin embargo es mejor que las dos primeras.

Algoritmo detallado:

La parte complicada de éste ejercicio está en poder comprobar de manera eficiente si incluir una nueva tarea genera conflicto. Para ello se puede mantener el vector solución S ordenado (según di), tratando de insertar la nueva tarea en su posición correspondiente. Es fácil comprobar después si alguna de las tareas desplazadas o la propia tarea insertada se encuentra en una posición mayor a su correspondiente di, en cuyo caso añadir la tarea no es factible. (Se numera i desde 0).

```
void factible_insertar (datos * S, int & elems, datos x) {
        // datos es una estructura con los valores di y gi
        int factible=1, i , pos
        for ( i= elems; i>0 && x.d < S[i-1].d && factible; i--)
            if ( S[i-1] > i )
                factible = 0;

      pos= i;
      if (factible && x.d <= pos) { // comprobación factibilidad
          for (i= elems ; i>pos ; i--)
                S[i] = S[i-1];                  // inserción
          S[pos] = x;
          elems ++;
      }
      return;
  }

void MaxBeneficio (datos * C, int n, datos * S, int & elems) {
        int i;
        OrdenarPorBeneficios ( C , n);
        elems = 0;
  ⊛
```

```
(*) for (i=0 ; i<n ; i++)
        factible_insertar (S, elems, G[i]);

    return;

}
```

Instancia :

i	0	1	2	3
gi	50	10	15	30
di	1	0	1	0

Se adaptan los valores para que cuenten desde 0.

Ordenación G :

G				
g	50	30	15	10
d	1	0	1	0

Iteracion 0 :

$S = \begin{array}{c|c} g & 50 \\ d & 1 \\ & 0 \end{array}$

elems = 1

Iteración 1 : $S = \begin{array}{|c|c|} \hline 30 & 50 \\ \hline 0 & 1 \\ \hline \end{array}$ (posiciones 0, 1)

elems = 2.

Iteración 2 y 3: ningún otro elemento puede ser insertado

$\Downarrow$

$S = \begin{array}{|c|c|} \hline 30 & 50 \\ \hline 0 & 1 \\ \hline \end{array}$ (posiciones 0, 1) $\rightarrow$ solución óptima con beneficio 80.

elems = 2

Para ver por qué funciona habría que demostrarlo :

1. Demostrar que la primera elección es la correcta.
2. Demostrar que existen subestructuras optimales.

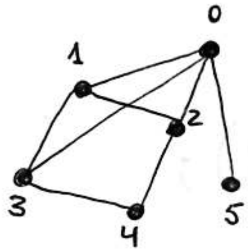
Se deja como ejercicio por resolver.

---

(4) Ej. 4 de la relación.

b) Se puede encontrar un contraejemplo en el que no funciona al considerar el máximo grado de incidencia sobre los nodos no seleccionados.

Contraejemplos:



incidencias = | 4 | 3 | 3 | 3 | 2 | 1 |
0 1 2 3 4 5
vértices

cogiéndolos en este orden (de mayor a menor grado), la solución sería,

$S =$ | 0 | 1 | 2 | 3 | con 4 vértices.

Sin embargo $S =$ | 0 | 3 | 2 | también es solución con sólo 3 vértices.

El algoritmo así no es correcto.

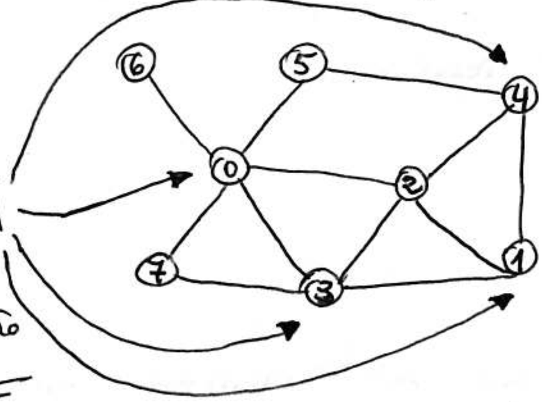
a) Tal cual están planteadas las preguntas a) y b), podemos proponer un algoritmo que seleccione los vértices por su grado de incidencia, sabiendo que es una buena alternativa pero que no es un algoritmo correcto.

Hay más alternativas. Por ejemplo:

1) Seleccionar el vértice al que más aristas le queden sin cubrir. Implica actualizar el número de aristas cada vez que se selecciona un vértice. Sin embargo, aunque funciona para el caso anterior, sigue existiendo un contraejemplo:

S
| 0 | 3 | 1 | 4 |

solución de tamaño 4



incidencias: | 5 | 3 | 4 | 4 | 3 | 2 | 1 | 2 |

libres: | 5 | 3 | 4 | 4 | 3 | 2 | 1 | 2 |
0 1 2 3 4 5 6 7

iteración 0:
S: | 0 |
libres: | 0 | 3 | 3 | 3 | 3 | 1 | 0 | 1 |
0 1 2 3 4 5 6 7

iteración 1:
S: | 0 | 1 |
libres: | 0 | 0 | 2 | 2 | 2 | 1 | 0 | 1 |
0 1 2 3 4 5 6 7

iteración 2:
S: | 0 | 1 | 2 |
libres: | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 1 |
0 1 2 3 4 5 6 7

(*)

⊛ iteración 3:

S: | 0 | 1 | 2 | 3 |

libres: | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 |
0 1 2 3 4 5 6 7

→ iteración 4:

S: | 0 | 1 | 2 | 3 | 4 |

libres: | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
0 1 2 3 4 5 6 7

↓

Solución de tamaño 5 !!!

2) También se pueden combinar ambas medidas

- Escoger el que cubra un mayor número de aristas libres y si hay varios con el mismo número, escoger el de mayor grado de incidencia. Éste funciona en los dos ejemplos propuestos y no resulta fácil encontrar un contraejemplo. Sin embargo también existe un contraejemplo.

Por simplicidad, y puesto que sería correcto responder así en un examen, vamo a resolverlo respecto a únicamente el grado de incidencia.

- Lista de candidatos G: el conjunto de los vértices.
- Lista de seleccionados S: los vértices seleccionados hasta el momento.
- Función objetivo: tamaño de S.
- Función de factibilidad: que el vértice seleccionado tenga aristas por cubrir.
- Función de selección: el de mayor grado de incidencia.

Puesto que el grado de incidencia es fijo se puede realizar una ordenación al principio. Se supone que G es un vector cuyos elementos tienen dos campos, PS 8

grado y vértice (tipo datos).

```
void cubre (datos * G', int n, int * S', int & elems,
            tipo_grafo G ) {
    int i,
    OrdenarPorGrado (G', n);
    elems = 0;
    for (i=0; i<n; i++)
        if (quedan_aristas (G'[i])) {
            S'[elems] = G'[i].vertice;
            ActualizarGrados (G', n, i); // esta operación no
                                         // cambia el orden de los
                                         // elementos en G'.
            elems ++;
        }
    return;
}
```

equivalente a (G'[i].grado != 0)

se recorren todos los elementos desde i en adelante y si su grado es distinto de cero y están conectados con G'[i].vertice se decrementan sus grados en 1.