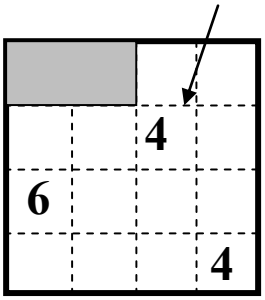
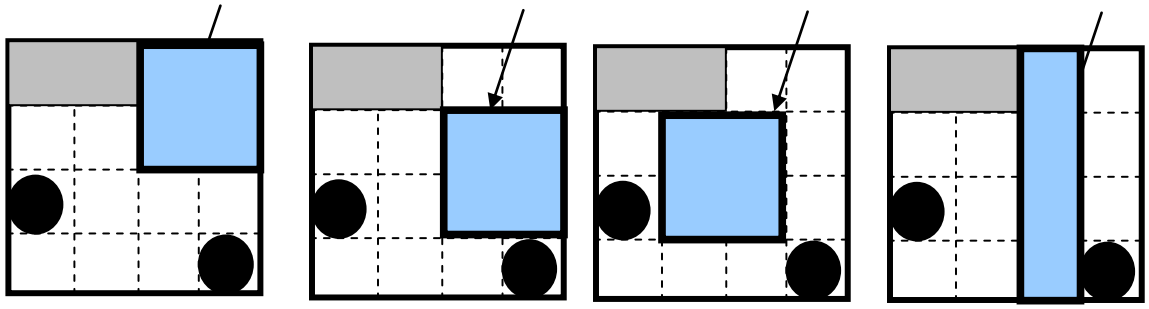
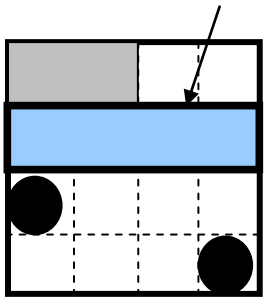
Cuestión 1 (1 punto). En la práctica obligatoria del presente curso 2008/2009 se ha tenido que diseñar y desarrollar un algoritmo para resolver el problema del puzle Shikaku. Dibuje todas las complecciones que tendría que explorar el algoritmo a partir de la situación mostrada en el tablero, para el nodo de la casilla marcada con una flecha, que contiene un 4 (sólo para ese nodo, y suponiendo que es el siguiente a considerar). Las casillas sombreadas indican que ya tienen asignado un rectángulo. Dibuje el tablero correspondiente para cada una de las complecciones.



Solución:





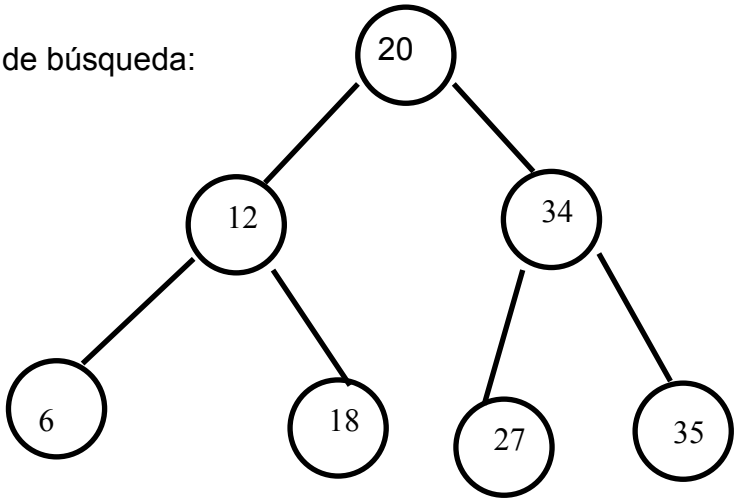
Cuestión 2 (2 puntos). Responda a las siguientes cuestiones.

1. ¿Qué diferencias hay entre un montículo y un árbol binario de búsqueda?
2. Dibuje un montículo (el árbol) y un árbol binario de búsqueda con los siguientes nodos: 6, 12, 18, 20, 27, 34, 35.

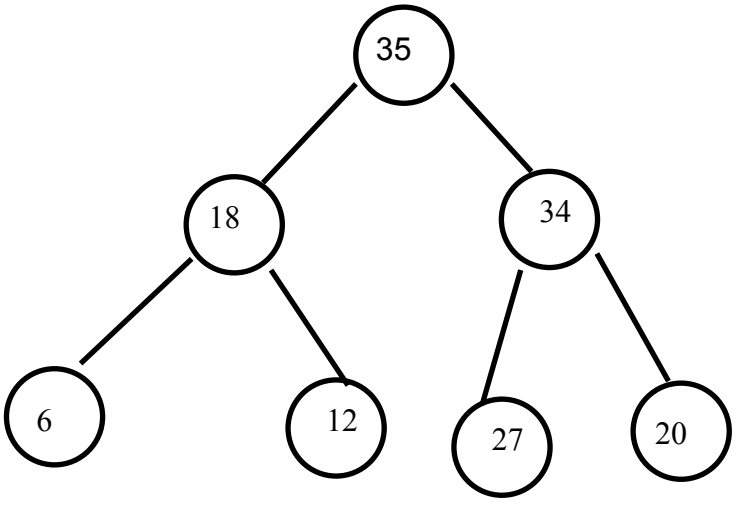
Solución:

1. En un árbol binario de búsqueda el valor contenido en todos los nodos internos es mayor o igual que los valores contenidos en su hijo izquierdo o en cualquiera de los descendientes de ese hijo, y menor o igual que los valores contenidos en su hijo derecho o en cualquiera de los descendientes de ese hijo. Mientras que un montículo es un árbol binario esencialmente completo, en el que el valor de cada uno de los nodos internos es mayor o igual que los valores de sus hijos. Esto se denomina propiedad del montículo. A diferencia de un árbol binario de búsqueda, un montículo se puede implementar como un vector sin ningún puntero explícito.

2. Árbol binario de búsqueda:



Montículo:



Cuestión 3 (3 puntos). Los residentes de una ciudad no quieren pavimentar todas sus calles, sino sólo aquellas que les permitan ir de una intersección a otra cualquiera de la ciudad con comodidad. Quieren gastarse lo menos posible en la pavimentación, teniendo en cuenta que el coste es directamente proporcional a la longitud de las calles que hay que pavimentar. El alcalde querría saber qué calles tiene que pavimentar para gastarse lo menos posible.

1. Indique qué esquema aplicarías para resolver el problema. Justifícalo y escribe el esquema general.
2. Explique los aspectos destacados de cómo resolverías el problema en función del esquema elegido (Por ejemplo: voraz: función de selección, demostración de optimalidad; divide y vencerás: tamaño del problema, cómo se divide del problema, subalgoritmo básico; vuelta atrás: descripción del espacio de búsqueda, compleciones; ramificación y poda: función de cota, compleciones…).
3. Indique el coste asociado y justifíquelo.

NO se pide el algoritmo.

Solución:

1.- Las intersecciones de la ciudad se pueden considerar los vértices de un grafo y las calles de la ciudad las aristas con el coste asociado de pavimentación. Como no se indica nada al respecto en el enunciado, se puede considerar que las calles son de 2 direcciones y por lo tanto estamos ante un grado valorado no dirigido. El objetivo es encontrar un subgrafo que contenga todos los vértices, que siga siendo conexo y que el coste de pavimentación sea mínimo, por lo que se trata de encontrar el **árbol de recubrimiento de coste mínimo**. Este problema se resuelve con un **esquema voraz** cuyo esquema general es el siguiente:

```
fun   voraz(C:conjunto)  dev  (S:conjunto)
      S ← ∅ {C: conjunto de candidatos y S: conjunto solución}
      mientras   ¬ solución(S) ∧ C ≠ ∅   hacer
            x ← seleccionar(C)
            C ← C \ {x}
            si    completable(S ∪ {x})  entonces
                  S ← S ∪ {x}
            fsi
      fmientras
      Si    solucion(S)  entonces  dev    S
            sino  dev   "no hay soluciones"
      fsi
ffun
```

2. y 3.- En el libro base de la asignatura, sección 6.3, se describen dos algoritmos voraces, Kruscal y Prim, que solucionan este problema. La instanciación del esquema general al problema y el cálculo de costes están descritos y explicados en el libro.

Problema (4 puntos).
Tenemos un conjunto de *n* componentes electrónicas ($c_1,\ldots c_n$) para colocar en *n* posiciones sobre una placa. Nos dan dos matrices N y D de dimensiones *n x n*, donde *N[i,j*] indica el número de conexiones necesarias entre la componente c_i y la componente c_j, y *D[p,q]* indica la distancia sobre la placa entre la posición *p* y la posición *q* (ambas matrices son simétricas y con diagonales nulas). Un cableado ($x_1,\ldots x_n$) de la placa consiste en la colocación de cada componente c_i en una posición distinta. La longitud total de este cableado viene dada por la fórmula:

$$\sum_{i<j} N[i,j]D[x_i,x_j]$$

Se pide un algoritmo para encontrar el cableado de longitud mínima.

La resolución de este problema debe incluir, por este orden:

1. Elección del esquema más apropiado, el esquema general y explicación de su aplicación al problema (0,5 puntos).
2. Descripción de las estructuras de datos necesarias (0.5 puntos).
3. Algoritmo completo a partir del refinamiento del esquema general (2,5 puntos).
4. Estudio del coste del algoritmo desarrollado (0.5 puntos).

Solución:

Estructuras de datos y métodos algoritmicos
N. Martí Oliet, et al. pag 485.

Indicaciones:

Esquema: ramificación y poda.

La solución del problema es un cableado en el que se asigna una posición x_i a la componente i. Se tiene que cumplir que las posiciones asignadas sean válidas y que no se repitan, es decir, son permutaciones de {1,...,n}.

funcion ramificacion_poda (*ensayo*) dev *ensayo* {ensayo es un nodo}
  m ← monticulo_vacio();
  cota_superior ← cota_superior_inicial;

```
  solucion ← primera_solucion;
  añadir_nodo(m, ensayo); {asignamos tareas al agente 1}
  mientras ¬ vacio(m) hacer
  nodo ← extraer_raiz(m);
  si valido(nodo) entonces {están completas las asignaciones }
    si coste_asig(nodo) < cota_superior entonces
                solucion ← nodo;
                cota_superior ← coste_asig(nodo)
    fsi
   si no
    si cota_inferior(nodo) ≥ cota_superior entonces dev solucion
    si no
                para cada hijo en compleciones(nodo) hacer
                        si cota_inferior(hijo) < cota_superior
                                añadir_nodo(m, hijo)
                         fsi
             fpara
     fsi
  fsi
fmientras
ffuncion
```

Estructuras de datos:

Mantendremos marcadores con las posiciones ya utilizadas y el coste del cableado que representa la solución parcial en la que nos encontramos.

nodo=tupla
- asignaciones: vector[1..N];
- último_asignado: cardinal;
- componentes_no_asignadas: lista de cardinal;
- coste: real;
- coste_optimo: real

Montículo de mínimos (cota mejor la de menor coste)

Cotas:

Cota inferior:

Sea cost el coste de la solución parcial (x1,...,xk), y sema minD el mínimo global de la matriz D. Entonces una cota inferior a la longitud del cableado de la solución es

$$\text{cota\_inf} = \text{cost} + \text{minD} \sum_{j=k+1}^{n} \sum_{i=1}^{j-1} N[i, j]$$

Cota superior:

Sea maxD el máximo global de la matriz D, entonces una cota superior es:

$$\text{cota\_sup} = \text{cost} + \text{maxD} \sum_{j=k+1}^{n} \sum_{i=1}^{j-1} N[i, j]$$

Coste:
El árbol de exploración tiene n niveles.
En este caso únicamente podemos hallar una cota superior del coste del algoritmo por descripción del espacio de búsqueda. En el caso peor se generan (k-1) hijos por cada nodo del nivel k, habiendo n. Por tanto, el espacio a recorrer siempre será menor que n!.