Asignatura: Fundamentos de algoritmia
Evaluación continua Vuelta atrás (backtracking). Cuestionario.

Profesor: Isabel Pita.

Nombre del alumno:

1. (1 punto) El esquema general de vuelta atrás visto en clase para resolver un problema en que nos piden encontrar todas las soluciones es:

```
void vueltaAtras (TDatos const& datos, int k, Tupla & sol, Tmarcas & marcas) {
   for(auto i = 0; i < numObjetos;  ++i){
      sol[k] = i;
      marcar(marcas);
      if (esValida(sol, k, marcas)){
         if (k == numEtapas - 1)  tratarSolucion(sol);
         else vueltaAtras(datos, k + 1, sol, marcas);
      }
      desmarcar(marcas);
   }
}
```

   Dibuja el árbol de exploración que recorre la función indicando que representan las ramas y cuál es la altura del árbol. Indica el tamaño máximo del vector solución y lo que guarda cada una de las componentes.

   Respuesta:

   a) El árbol de exploración tiene como ramas los objetos. Cada nodo del árbol tiene `numObjetos` ramas. Esto se observa en el código en el bucle `for` que recorre todos los objetos.

   b) La altura del árbol es el número de etapas. Esto se observa en el código en la condición de ser solución `k == numEtapas - 1`, siendo `k` la etapa en la que estamos. Cada nivel del árbol es una nueva etapa.

   c) El vector solución tiene tamaño el número de etapas, ya que en cada etapa se añade un elemento a la solución.

   d) El vector solución guarda en cada componente el objeto seleccionado en esa etapa. (`sol[k] = i`)

2. (1 punto) La función `esValida` debe comprobar si la solución que se está construyendo cumple las restricciones del problema. Si una restricción nos dice que la solución no puede tener objetos repetidos, indica si utilizarías alguna marca para evitar que la función `esValida` recorra todo el vector solución comprobando dicha restricción. Explica cómo sería esta marca.

   Respuesta:

   Para comprobar que el objeto seleccionado en una etapa no está repetido con los objetos seleccionados en las etapas anteriores en tiempo constante debemos tener como marca un vector de booleanos de tamaño el número de objetos, que en cada componente guarde si el objeto está seleccionado en la solución o no.

   Cuando se selecciona un objeto para formar parte de la solución se marca en el vector, es decir se pone su valor a true. Cuando el objeto deja de estar en la solución se desmarca, poniendo la componente correspondiente al objeto al valor false.

3. (1 punto) La función `esValida` debe comprobar si la solución que se está construyendo cumple las restricciones del problema. Si una restricción nos dice que la solución no puede tener objetos consecutivos iguales, indica si utilizarías alguna marca para evitar que la función `esValida` recorra todo el vector solución comprobando dicha restricción. Explica cómo sería esta marca.

Respuesta:

En este caso no es necesario definir una marca especial para comprobar la restricción, ya que el objeto anterior figura en la posición anterior del vector solución. Si no se lleva explícito el vector solución entonces es necesario tener un parámetro en el que se guarde la solución que se ha asignado en la etapa anterior.

4. (1 puntos) Dado un problema que se resuelve con la técnica de vuelta atrás en el que nos piden calcular una sola solución del problema, indica las modificaciones que hay que realizar en el esquema general para resolverlo.

```
void vueltaAtras (TDatos const& datos, int k, Tupla & sol, Tmarcas & marcas) {
   for(auto i = 0; i < numObjetos;  ++i){
      sol[k] = i;
      marcar(marcas);
      if (esValida(sol, k, marcas)){
         if (k == numEtapas - 1)  tratarSolucion(sol);
         else vueltaAtras(datos, k + 1, sol, marcas);
      }
      desmarcar(marcas);
   }
}
```

Respuesta:

a) Se añade un parámetro por referencia `exito` de tipo bool. En él se indica si se ha encontrado una solución o no. Debe estar inicializado al valor `false`.

b) En la cabecera del bucle `for` se añade una condición: `i < numObjetos && !exito`

c) Al tratar la solución se asigna el valor `true` al parámetro `exito`. De esta forma una vez encontrada la primera solución el bucle `for` para y se devuelve el control. Como el parámetro tendrá el valor `true` van parando todos los bucles de todas las llamadas recursivas que estuviesen activas.

5. (1 puntos) Dado un problema que se resuelve con la técnica de vuelta atrás en el que nos piden calcular la mejor solución del problema, indica las modificaciones que hay que realizar en el esquema general para resolverlo.

```
void vueltaAtras (TDatos const& datos, int k, Tupla & sol, Tmarcas & marcas) {
   for(auto i = 0; i < numObjetos;  ++i){
      sol[k] = i;
      marcar(marcas);
      if (esValida(sol, k, marcas)){
         if (k == numEtapas - 1)  tratarSolucion(sol);
         else vueltaAtras(datos, k + 1, sol, marcas);
      }
      desmarcar(marcas);
   }
}
```

Respuesta:

a) Se añaden parámetros para llevar el coste de la solución actual y de la solución mejor. Se añade también un parámetro para llevar la solución mejor. El coste de la solución actual y de la solución mejor deben estar inicializados.

b) Después de asignar el objeto a la solución se actualiza el coste de la solución actual.

c) Cuando se encuentra una solución y antes de tratarla se debe comprobar que la solución obtenida es *mejor* que la solución que tenemos guardada. En caso de ser *mejor* se modifican los valores de la solución mejor.

d) Antes de realizar la llamada de vuelta atrás se debe calcular la estimación y hacer la llamada solo si se puede encontrar una solución mejor.

e) Después de desmarcar, en el mismo bloque de código en que se actualizó el coste de la solución actual se debe deshacer la actualización ya que se va a probar un nuevo objeto.

6. (1 punto) La técnica de estimación se utiliza para podar el árbol de exploración y no recorrer aquellas partes que no pueden llevar a una solución mejor que una solución ya encontrada. Marca los tipos de problema de vuelta atrás a los que se puede aplicar esta técnica.

   a) Problema en que nos piden encontrar todas las soluciones del problema.

   b) Problema en que nos piden encontrar una solución al problema.

   c) Problema en que nos piden encontrar la mejor solución del problema (problema de optimización)

   Respuesta:

   La solución correcta es la c. La técnica de estimación se utiliza para podar el árbol de exploración en los problemas de optimización.

   Esta técnica no se puede utilizar en los otros casos dado que es necesario comparar la suma del coste actual y el coste estimado con la solución mejor que se haya encontrado en la parte del árbol de exploración ya explorada. En los dos primeros problemas no se tiene esta solución mejor con la que comparar, ya que no se calcula. Existe una excepción si en el problema nos indican que el valor de la solución no puede sobrepasar un cierto umbral, o bien debe ser mayor que un cierto umbral. En este caso se puede hacer una estimación optimista / pesimista y podar aquellas ramas en las que siempre se sobrepase el umbral o nunca se llegue al umbral.

7. (1 punto) Si un problema se puede resolver utilizando un árbol de exploración binario o un árbol de exploración n-ario, indica cuál de las soluciones elegirías. Justifica tu respuesta.

   Respuesta:

   Si el problema se puede resolver recorriendo los dos árboles de exploración, por ejemplo como ocurre en el problema de la mochila, es siempre más eficiente utilizar un árbol binario, ya que el espacio de soluciones en el problema de la mochila tiene tamaño $\mathcal{O}(2^n)$ siendo n el número de objetos mientras que utilizando un árbol de exploración n-ario el espacio de soluciones tiene tamaño $\mathcal{O}(n^n)$.

8. (1.5 punto) Dado el problema del viajante, en el que un vendedor debe recorrer N ciudades pasando una única vez por cada una de ellas y se pide obtener el orden en que debe visitarlas para minimizar la distancia recorrida, dibuja el árbol de exploración que debe recorrer la ejecución del programa, e indica como realizarías una estimación de la distancia que le queda por recorrer al viajante.

   Respuesta:

   El árbol de exploración en el problema del viajante tiene como ramas las ciudades que se deben visitar. Todos los nodos tienen el mismo número de ramas que es el número de ciudades. La altura es el número de días en que se va a viajar que coincide con el número de ciudades.

   El problema de la estimación en este caso es que en un determinado punto del árbol de exploración las ciudades que se han recorrido dependen de las ramas del árbol que se hayan seleccionado. Comprobar en cada nodo las ciudades que quedan por recorrer es costoso ya que requeriría recorrer el vector de marcas donde llevamos las ciudades que ya hemos recorrido.

Un estimación sencilla, pero a su vez poco eficaz es calcular el mínimo de la matriz de distancias, esto es la distancia entre las dos ciudades que estén más cerca. En el momento de realizar la estimación multiplicaremos el número de ciudades que nos quedan por recorrer (N-k) por esa distancia mínima.

Se pueden realizar estimaciones más complicadas pero también de coste constante calculando un vector con la mínima distancia desde cada ciudad a cualquiera de las otras. A partir de aqui podemos:

- ordenar el vector de mínimos de mayor a menor y hacer su vector de acumulados (una vez ordenado), de forma que en la última posición tengamos la distancia mínima, en la penúltima la suma de las dos distancias más pequeñas etc. En el momento de realizar la estimación se puede consultar la suma de las k últimas posiciones del vector de mínimos en el vector de acumulados. Esta es la forma más barata de ir a N-k ciudades, considerando las N-k ciudades diferentes a las que es mas barato ir, aunque no sean las ciudades que nos quedan por visitar.
- sumar todos los valores del vector de mínimos y pasar como parámetro a la función de vuelta atrás el vector de mínimos y la suma. En cada etapa, cuando seleccionamos una ciudad para visitar debemos restar a la suma el valor en el vector de mínimos correspondiente a esa ciudad. Cuando se elimina la selección de la ciudad se sumará de nuevo el valor. De esta forma, la suma es el mínimo de viajar a las ciudades que quedan por visitar.

9. (1.5 punto) Dado el problema de la mochila en el que tenemos N objetos cada uno con un valor y un peso y nos piden obtener los objetos que maximizan el valor de una mochila que soporta un peso máximo P. Dibuja el árbol de exploración que debe recorrer la ejecución del programa, e indica cómo realizarías una estimación del valor que se puede obtener con los objetos que todavía no se han introducido en la mochila.

Respuesta:

El árbol de exploración que debemos emplear para resolver el problema de la mochila es un árbol binario. En cada etapa tratamos un objeto y las ramas que podemos elegir corresponden a añadir el objeto a la mochila o no añadirlo.

Para realizar la estimación del problema de la mochila es necesario ordenar los objetos antes de comenzar a aplicar la vuelta atrás por valor por unidad de peso, de forma que el primer objeto que intentemos añadir a la mochila sea el que tenga más valor por unidad de peso. Una vez en la función de vuelta atrás, en cada nodo del árbol, la estimación aplica un algoritmo voraz. Se completa la mochila con los objetos de las etapas siguientes, en orden y hasta alcanzar el peso permitido, por defecto. Como el vector de objetos está ordenado vamos cogiendo los objetos que más valor nos van a aportar. En muchos casos no se completará completamente la mochila, quedando un peso menor que el del siguiente objeto. Sumaremos la proporción del valor correspondiente al peso que queda libre en la mochila. De esta forma obtenemos una cota superior de la solución real, ya que partimos el último objeto que podemos añadir a la mochila, lo que no está permitido en las soluciones reales.