Estructuras de Datos no Lineales

Práctica 5

Problemas de árboles parcialmente ordenados y otros árboles

TRABAJO PREVIO

Antes de asistir a la sesión de prácticas es obligatorio:

1. **Implementar y probar** los TAD que sean necesarios para resolver los diferentes problemas.
2. Imprimir copia de este enunciado.
3. Lectura profunda del mismo.
4. Reflexión sobre el contenido de la práctica y generación de la lista de dudas asociada a dicha práctica y a los problemas que la componen.
5. **Esbozo serio de solución** de los problemas en papel (al menos de los que se hayan entendido).

PASOS A SEGUIR

1. Escribir módulos que contengan las implementaciones de los subprogramas demandados en cada problema.
2. Para cada uno de los problemas escribir un programa de prueba, independiente de la representación del TAD elegida, donde se realicen las llamadas a los subprogramas del paso anterior, comprobando el resultado de salida para una batería suficientemente amplia de casos de prueba.

PROBLEMAS

1. Dado un árbol binario de enteros donde el valor de cada nodo es menor que el de sus hijos, implementa un subprograma para eliminar un valor del mismo preservando la propiedad de orden establecida. Explica razonadamente la elección de la estructura de datos.

Nota: Se supone que en el árbol no hay elementos repetidos, y que el número de nodos del mismo no está acotado

2. Un montículo *min-max* tiene una estructura similar a la de un montículo ordinario (árbol parcialmente ordenado), pero la ordenación parcial consiste en que los elementos que se encuentran en un nivel par (0, 2, 4,...) son menores o iguales que sus elementos descendientes, mientras que los elementos que se encuentran en un nivel impar (1, 3,

5,...) son mayores o iguales que sus descendientes. Esto quiere decir que para cualquier elemento *e* de un nivel par se cumple $abuelo \leq e \leq padre$ y para cualquier elemento *e* de un nivel impar $padre \leq e \leq abuelo$.

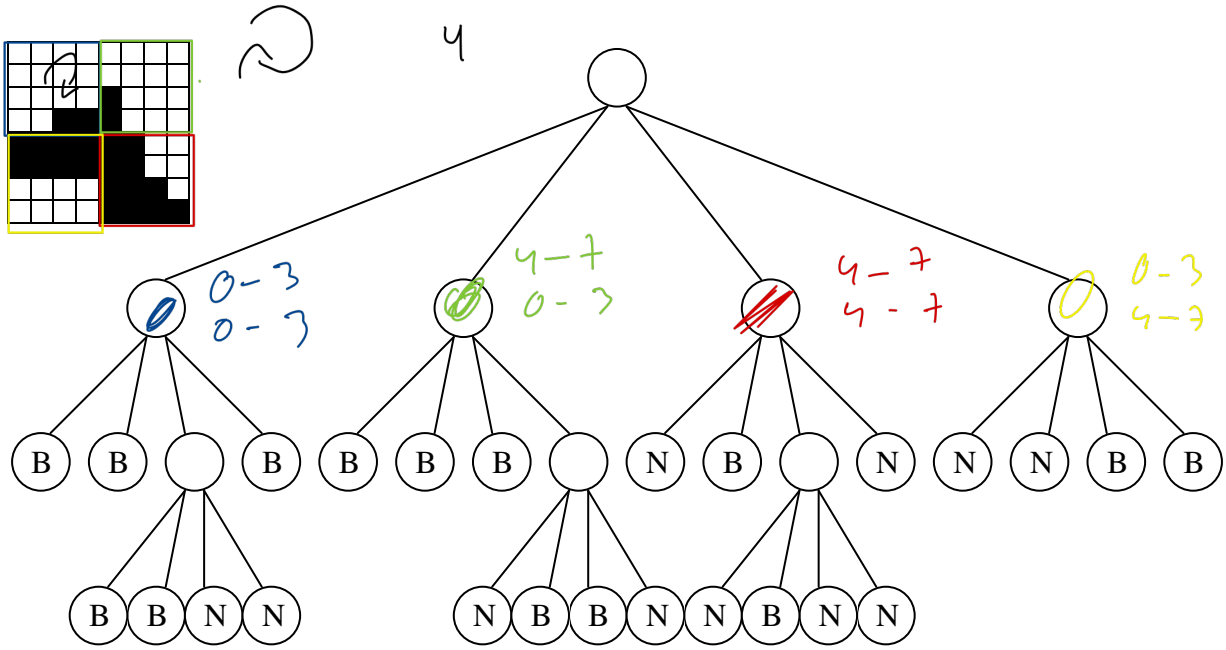
Implementa una operación de orden logarítmico para añadir un elemento a un montículo *min-max* almacenado en un vector de posiciones relativas.

3. Implementa una operación de orden logarítmico para eliminar el elemento máximo de un montículo *min-max* definido como en el problema anterior.

4. Un árbol es estrictamente ternario si todos sus nodos son hojas o tienen tres hijos. Escribe una función que, dado un árbol de grado arbitrario, nos indique si es o no estrictamente ternario.

5. Una forma de representar una figura plana en blanco y negro consiste en utilizar un árbol cuaternario en el que cada nodo o tiene exactamente cuatro hijos, o bien es una hoja. Un nodo hoja puede ser blanco o negro y un nodo interno no tiene color.

Una figura dibujada dentro de un cuadrado de lado 2^k se representa de la forma siguiente: Se divide el cuadrado en cuatro cuadrantes y cada uno se representa como un hijo del nodo raíz. Si un cuadrante está completamente negro corresponde a una hoja negra; si, por el contrario, el cuadrante está completamente blanco, éste corresponde a una hoja blanca; y si un cuadrante está parcialmente ocupado por negro y blanco, entonces corresponde a un nodo interno del árbol y este cuadrante se representa siguiendo el mismo método subdividiéndolo en otros cuatro cuadrantes. Como ejemplo se muestra una figura en blanco y negro y su árbol asociado, tomando los cuadrantes en el sentido de las agujas del reloj a partir del cuadrante superior izquierdo.



Implementa una función que dado un árbol de esta clase, con *k+1* niveles, devuelva la figura asociada, representada como una matriz cuadrada de tamaño 2^k en la que cada celda representa un punto blanco o negro.

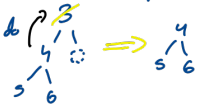
Nota: Por simplificar el problema, se asume que en cada nodo del árbol se incluyen las coordenadas de la esquina superior izquierda y de la esquina inferior derecha del cuadrante que representa.

1- Se supone que si elimino un nodo, sus hijos son más grandes, por tanto cojo el hijo más chico y lo subo, y así de manera recursiva hasta no poder subir más

Partiendo del nodo que me interesa
caso_base n == NODO_NULO
caso_general:
if (A.hijoIzqd(n) == A.hijoDcho(n) == A.NODO_NULO)
→ obtengo el padre y la rama y elimino n ←
else if (elemento(A.hijoIzqd(n)) < A.hijoDcho(n))
A.elemento(n) = A.hijoIzqd(n)
borrar Elmto(A, A.hijoIzqd(n))
else if (elemento(A.hijoIzqd(n)) > A.hijoDcho(n))
A.elemento(n) = A.hijoDcho(n)
borrar Elmto(A, A.hijoDcho(n))

Caso 1: es una hoja → lo elimino
Caso 2: tiene un hijo → lo hundo y elimino
Caso 3: tiene dos hijos → Voy hundiendo el mas chico y elimino

Creo que debería comprobar si solo tiene un hijo ya que solo con subir ese nodo estaría solucionado



2 y 3- no sé

4- Caso_base: n == A.NODO_NULO → return true; o no tiene hijos
Caso_general:

```
bool Ternario = true;
int hijo = 0;
nodo hermano = n;
while (hermano != A.NODO_NULO) {
    hijo++;
    if(hijo > 3) ternario = false;
else if(A.hijoIzqdo(hermano) != A.NODO_NULO)
    ternario = (ternario && esTernario(A, A.hijoIzqdo(hermano)));
    hermano = A.hermDcho(hermano);
}
if (hijo != 0 && hijo != 3)
    ternario = false;
return ternario;
```

Buena lógica

Hijo 1 -> 0 - A la mitad
0 - A la mitad

Hijo 2 -> La mitad+1 - Fin
0 - A la mitad

Hijo 3 -> La mitad+1 - Fin
La mitad+1 - Fin

Hijo 4 -> 0 - A la mitad
La mitad+1 - Fin

El enunciado dice que el nodo puede guardar su rango (x_1, x_2) y $C\begin{pmatrix} x_1, x_2 \\ y_1, y_2 \end{pmatrix}$

El tamaño de la matriz mínimo será 2^h
h: Altura del árbol

Si el hijo es hoja ese subcuadrante estará coloreado

Parámetros de la función:
- Nodo
- Subcuadrante al que pertenece (rango fila (x_1, x_2), rango columna (y_1, y_2))
- & Árbol
- & Matriz

Función que rellena una matriz dado (rango fila (x_1, x_2), rango columna (y_1, y_2)) lo rellena del char q se pase

caso_base: n es hoja -> rellena el subcuadrante con la matriz

caso_general

n tiene hijo por tanto tendrá cuatros

Llamo 4 veces a la func rec

Nodo hijo = A.hijoIzqdo(n)

(llenarMatriz(C hijo, A, $(x_1, x_2/2), (y_1, y_2/2)$, V))

hijo = A.hermanoDcho(hijo);

(llenarMatriz(C hijo, A, $(x_2/2+1, x_2), (y_1, y_2/2)$, V))

hijo = A.hermanoDcho(hijo);

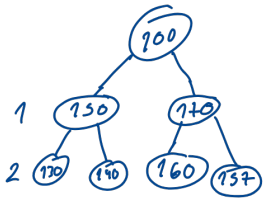
(llenarMatriz(C hijo, A, $(x_2/2+1, x_2), (y_2/2+1, y_2)$, V))

hijo = A.hermanoDcho(hijo);

(llenarMatriz(C hijo, A, $(x_1, x_2/2), (y_2+1, y_2)$, V))

Podría evitar pasar las coordenadas, calculando a partir de altura y profundidad y la pos 1, 2, 3 o 4 a través de que hermano sea.

2 y 3 -> Crear un nuevo TAD



Es un apo no puede haber hojas a la dcha sin haber a la izq

Se va a insertar al final por lo que tengo que insertarlo al final e ir flotando hasta que se cumpla la condición (par: abuelo ≤ padre ≤ e; impar: padre ≤ e ≤ abuelo)

- El máximo si empze está en el nivel 1
- ¿Se puede hundir? -> Pensar si es igual que en un Apo
  -> Lo vemos el 09/04