PARCIAL-ARBOLES-RESUELTO-2021-ED...

RaizDeX

Estructuras de Datos no Lineales

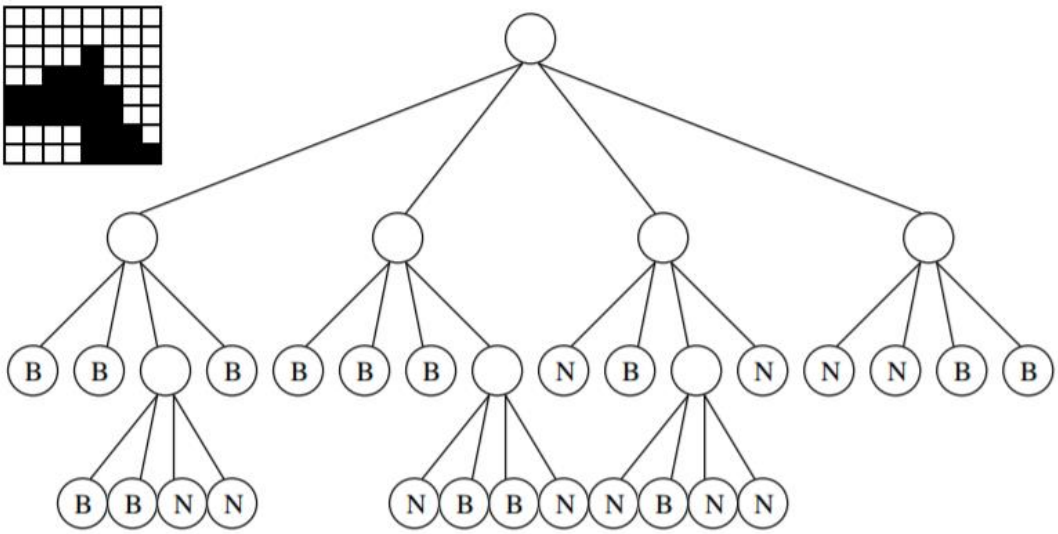
2º Grado en Ingeniería Informática

Escuela Superior de Ingeniería
Universidad de Cádiz

Enunciado Parcial Árboles: 29/04/2021

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.

Cada celda debía contener únicamente el color (a diferencia del ejercicio que aparece en prácticas, que podíamos simplificarlo suponiendo que también guardaba las coordenadas.

Podíamos suponer que existía una constante N predefinida, siendo la matriz de tamaño NxN.

Tipos de datos necesarios:

```
enum Color {b, n, sin_color};

struct cuadrante{
    unsigned x_ini, x_fin, y_ini, y_fin;
};
```

Función llamadora (la que no es recursiva):

```
Color** volcado(const Agen<Color>& A){
    Color **mat = new Color*[N];
    for (int i = 0; i < 8; i++) {
        mat[i] = new Color[N];
    }

    cuadrante cuad;
    cuad.x_ini = cuad.y_ini = 0;
    cuad.x_fin = cuad.y_fin = N-1;
    volcado_rec(A.raiz(), mat, cuad, A);
    return mat;
}
```

Función Recursiva:

```
void volcado_rec(const typename Agen<Color>::nodo& n, Color** mat, cuadrante cuad, const Agen<Color>& A){
    if(n != Agen<Color>::NODO_NULO){
        if(A.elemento(n) == sin_color){
            unsigned n_cuad = 0;
            typename Agen<Color>::nodo hijo = A.hijoIzqdo(n);
            while(hijo != Agen<Color>::NODO_NULO){
                switch(n_cuad){
                    case 0:
                        cuad.x_fin /= 2;
                        cuad.y_fin /= 2;
                        break;
                    case 1:
                        cuad.x_ini = cuad.x_fin+1;
                        cuad.x_fin = 2*cuad.x_fin + 1;
                        break;
                    case 2:
                        cuad.y_ini = cuad.y_fin+1;
                        cuad.y_fin = 2*cuad.y_fin +1;
                        break;
                    case 3:
                        cuad.x_ini = 0;
                        cuad.x_fin /= 2;
                        break;
                }
                volcado_rec(hijo, mat, cuad, A);
                n_cuad++;
                hijo = A.hermDrcho(hijo);
            }
        }else{
            for(unsigned i = cuad.x_ini; i <= cuad.x_fin; i++){
                for(unsigned j = cuad.y_ini; j <= cuad.y_fin; j++){
                    mat[i][j] = A.elemento(n);
                }
            }
        }
    }
}
```

Función main para probar la función (NO SE PEDÍA EN EL EXAMEN):

```
int main(){
    Agen<Color> A;

    A.insertarRaiz( e: sin_color);
    A.insertarHijoIzqdo(A.raiz(), e: b);
    A.insertarHermDrcho(A.hijoIzqdo(A.raiz()), e: n);
    A.insertarHermDrcho(A.hermDrcho(A.hijoIzqdo(A.raiz())), e: b);
    A.insertarHermDrcho(A.hermDrcho(A.hermDrcho(A.hijoIzqdo(A.raiz()))), e: n);
    Color **mat = volcado(A);
    for(int i = 0; i < N; i++){
        for(int j = 0; j < N; j++){
            std::cout << mat[i][j]<< " ";
        }
        std::cout <<std::endl;
    }
}
```

Salida por pantalla:

```
0 0 0 0 1 1 1 1
0 0 0 0 1 1 1 1
0 0 0 0 1 1 1 1
0 0 0 0 1 1 1 1
1 1 1 1 0 0 0 0
1 1 1 1 0 0 0 0
1 1 1 1 0 0 0 0
1 1 1 1 0 0 0 0

Process finished with exit code 0
```

Donde 0 representa color blanco y 1 color negro (por el orden del enum).