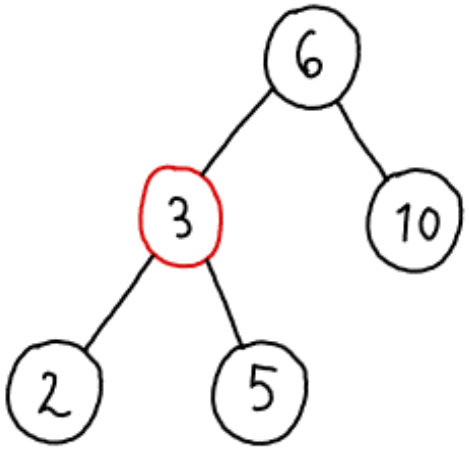
PIROS-FEKETE KERESŐFA

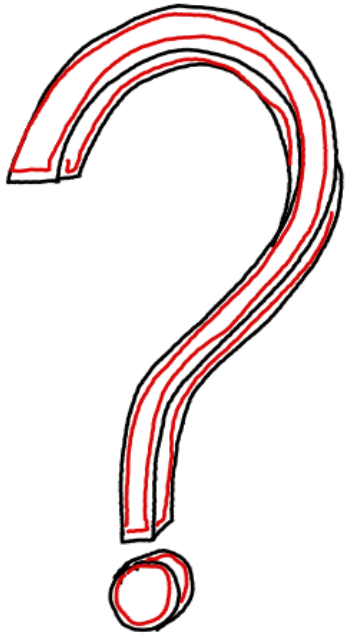


Piros-Fekete Keresőfa

Baki Roland

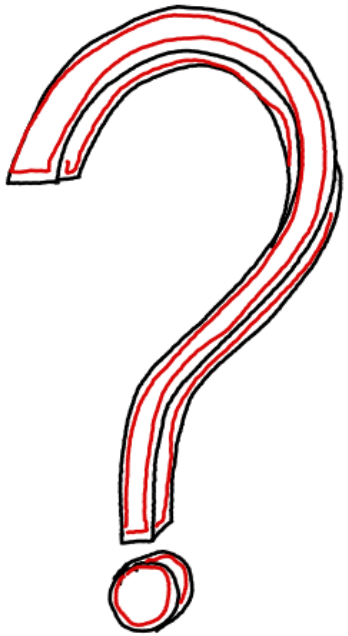
Tartalom:

- Piros-Fekete Fa bemutatása, tulajdonságai
- Piros-Fekete Fa implementálása, szükséges műveletek és azok elemzései
- Feladat felvezetése, megoldása és ennek érvelése
- Esetleges kérdések



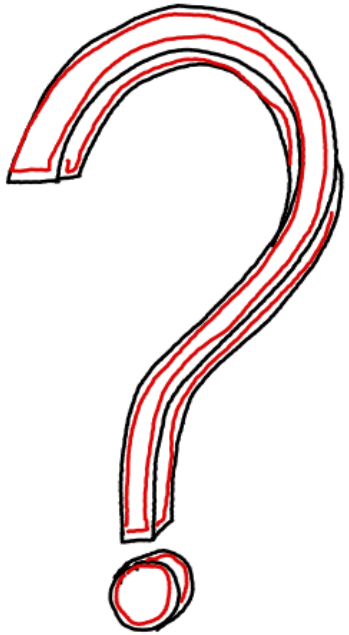
Bináris keresőfa

Bináris keresőfa



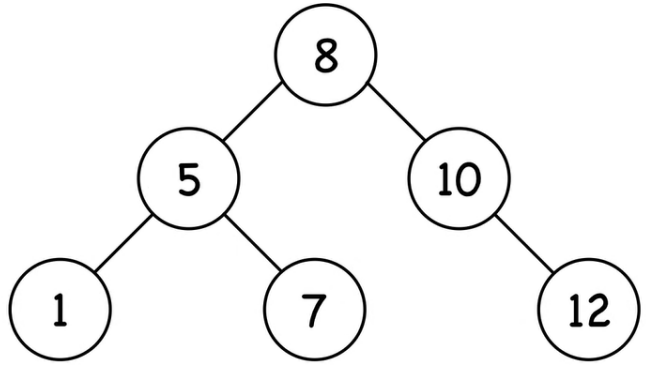
- **Rendezett bináris fa**
- **Kulcsérték**
- **Jobb > Bal**
- **1 gyökér, minden csomópontnak legfeljebb 2 részfája lehet**
- **Az összes levélcsúcs nullpointer**

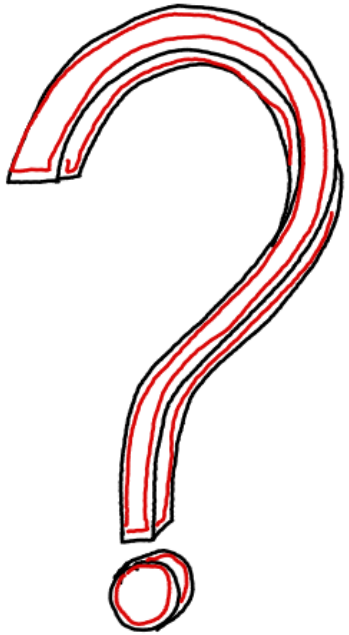
Bináris keresőfa



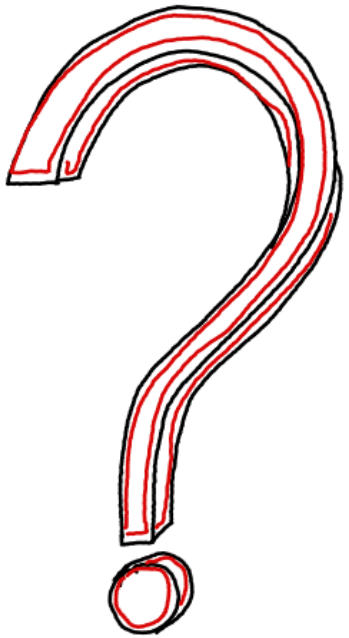
Önkiegyensúlyozó

Bináris keresőfa





Önkiegyensúlyozó



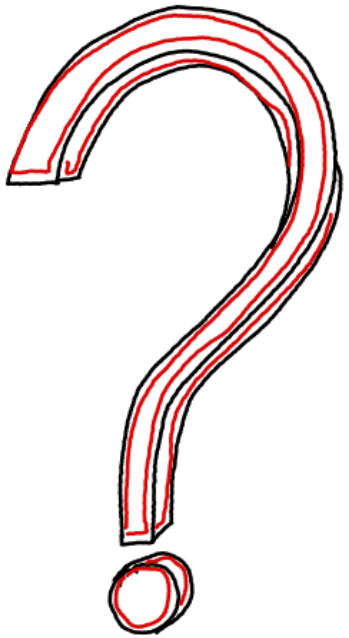
Szabályok:

Csomópont: Piros Fekete

Gyökér és levél fekete

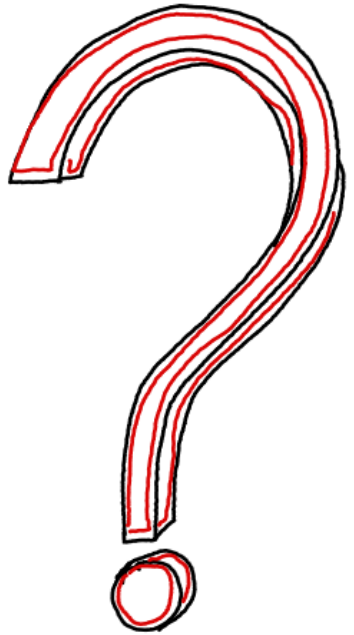
Ha egy csomópont piros, leszármazottjai feketék

Minden út a gyökértől, levélig, ugyanannyi fekete csomópontot tartalmaz



Piros-Fekete Keresőfa

Baki Roland



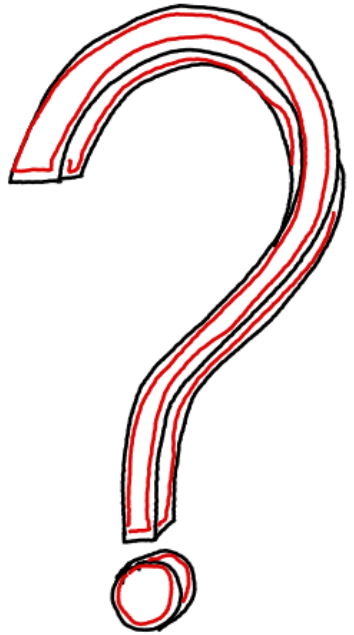
Piros-Fekete Keresőfa

Baki Roland

Magasság

Gyökeret nem bele számolva,hány fekete csomóponton haladunk át a levélig

Minden csomópontnak saját magassága van

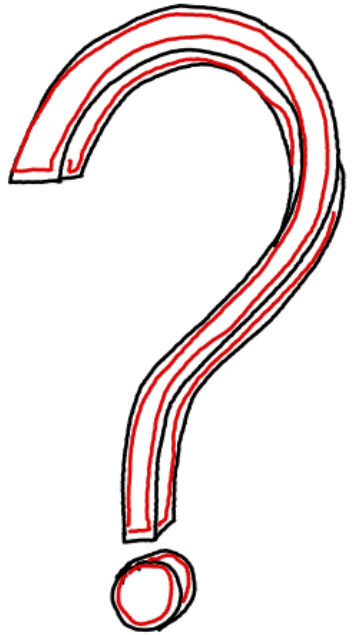


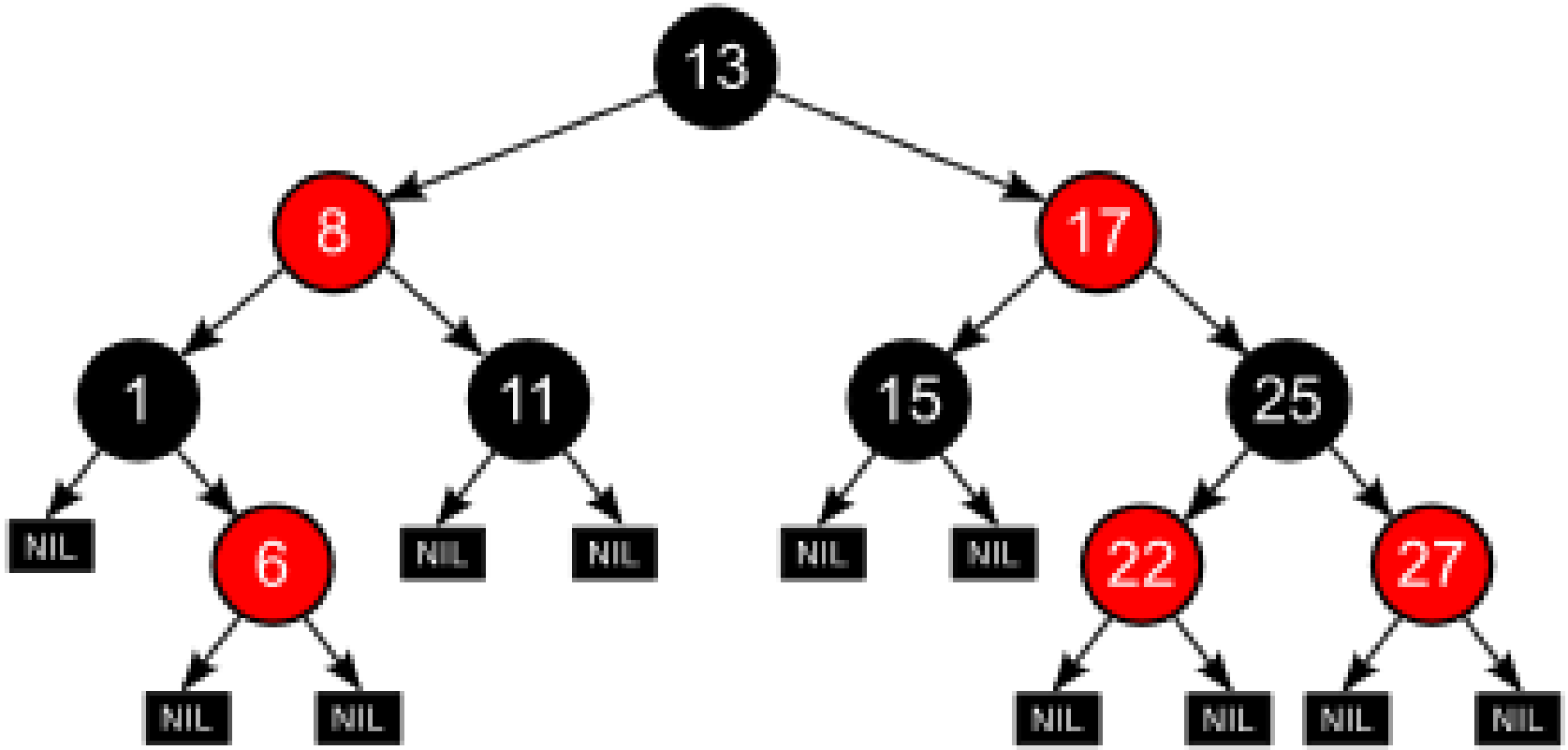
maximális magassága $2 \log_2(n + 1)$.

15 kulcs esetén max 8
100 kulcs esetén max 14

Miért hasznos?

Keresés	$O(\log n)$
Beillesztés	$O(\log n)$
Törlés	$O(\log n)$





Piros-Fekete Keresőfa

Baki Roland

Műveletek

Milyen adatok tárolására alkalmas egy Piros-Fekete Keresőfa?

```
template <typename T>
struct tree {
    node<T>* root;
    int size;
};
```

```
template <typename T>
struct node{
    long long key;
    bool color;
    bool nil;
    node* right;
    node* left;
    node* parent;
    T info;
};
```

Műveletek

Csomópont létrehozás

Előfeltétel ----------

Utófeltétel **Létre jön az adott kulcsú csomópont**

```
template <typename T>
node<T>* CreateNode(long long _key)
{
    node<T>* nd = new node<T>;
    nd->color = RED;
    nd->key =_key;
    nd->left = CreateNil<T>(nd);
    nd->right = CreateNil<T>(nd);
    nd->nil = false;
    return nd;
}
```

Műveletek

Levél létrehozás

Előfeltétel

Legyen egy már létező szülő

```
template <typename T>
node<T>* CreateNil(node<T>* parent)
{
    node<T>* nd = new node<T>;
    nd->color = BLACK;
    nd->key = 0;
    nd->parent = parent;
    nd->left = nullptr;
    nd->right = nullptr;
    nd->nil = true;
    return nd;
}
```

Utófeltétel

Létre jön a levél csatolva a szülőhöz

Műveletek

Előfeltétel

Létezzen a csomópont

```
template <typename T>
void PrintNode(node<T>*nd)
```

```
template <typename T>
long long GetKey(node<T>* nd)
```

```
template <typename T>
void FreeNode(node<T>* nd)
```

```
template <typename T>
bool GetColor(node<T>* nd)
```

Műveletek

Fa létrehozása

Előfeltétel ----------

Utófeltétel **Létre jön a fa**

```
//Fa létrehozása
template <typename T>
tree<T>* CreateRBT()
{
    tree<T>* rb = new tree<T>;
    rb->root = nullptr;
    rb->size = 0;
    return rb;
}
```

Műveletek

Fa törlése

Előfeltétel — **Létezzen a fa**

Utófeltétel — **Törli a fát vagy részfát**

```
template <typename T>
void FreeRBT(tree<T>* rbt)

template <typename T>
void FreeRBTNode(node<T>* nd)
```

Műveletek

Fa kiírás

Előfeltétel — **Létezzen a fa**

Utófeltétel — **Kiírja a fa tartalmát**

```
//Kiválasztható egy csomópont és a részfáját kiirja
template <typename T>
void WoutTree(tree<T>* rbt, node<T>* nd)
{
    if (!nd->nil)
    {
        PrintNode<T>(nd);
        WoutTree<T>(rbt, nd->left);
        WoutTree<T>(rbt, nd->right);
    }
}

//Bejárja a fát és meghívja minden csomópontra a kiiráast
template <typename T>
void PrintTree(tree<T>* rbt)
{
    if (rbt->root != nullptr)
    {
        WoutTree<T>(rbt, rbt->root);
    }
}
```

Műveletek

Maximum/Minimum

Előfeltétel

Létezzen a fa,ne legyen üres

Utófeltétel

Visszatéríti a maximumot/minimumot

```
while (!max->right->nil) {
    max = max->right;
}
return max;
```

```
while (!min->left->nil) {
    min = min->left;
}
```

Műveletek

Keresés

Előfeltétel

Létezzen a fa,legyen benne a kívánt elem

Utófeltétel

Visszatéríti a keresett kulcsú csomópontot

```
template <typename T>
node<T>* Find(tree<T>* rbt, long long key)
{
    if (rbt->root == nullptr)
    {
        return nullptr;
    }

    return Find<T>(rbt, rbt->root, key);
}
```

```
node<T>* Find(tree<T>* rbt, node<T>* nd, long long key)
{
    if (nd->nil)
    {
        return nullptr;
    }

    if (nd->key == key)

    {
        return nd;
    }
    else
    {
        if (nd->key > key)
        {
            return Find<T>(rbt, nd->left, key);
        }
        else
        {
            return Find<T>(rbt, nd->right, key);
        }
    }
}
```

Műveletek

Benne van-e az elem

Előfeltétel

Létezzen a fa,legyen benne a kívánt elem

Utófeltétel

Igazat vagy hamisat térít véssza

```
template <typename T>
bool IsIn(tree<T>*rbt, long long key)
{
    node<T>* nd = rbt->root;
    while (nd != nullptr && nd->key != key)
    {
        if (key > nd->key)
            nd = nd->right;
        else
            nd = nd->left;
    }
    if (nd != nullptr)
        if (nd->key == key)
            return true;
    return false;
}
```

Műveletek

Más műveletek

```
//Sorba rendeyve az elemeket egy bizonyos kulcsú elem előtti elemet térít vissza
template <typename T>
node<T>* Before(tree<T>*rbt,long long key) { ... }
```

```
//Sorba rendeyve az elemeket egy bizonyos kulcsú elem utáni elemet térít vissza
template <typename T>
node<T>* After(tree<T>* rbt, long long key) { ... }
```

```
//Vissyatériti egy adott kulcsú csomópont rangját
template <typename T>
long long GetRank(tree<T>* rbt, long long key) { ... }
```

```
//Adott rangú elem keresérse
template <typename T>
node<T>* GetElement(tree<T>* rbt, long long number) { ... }
```

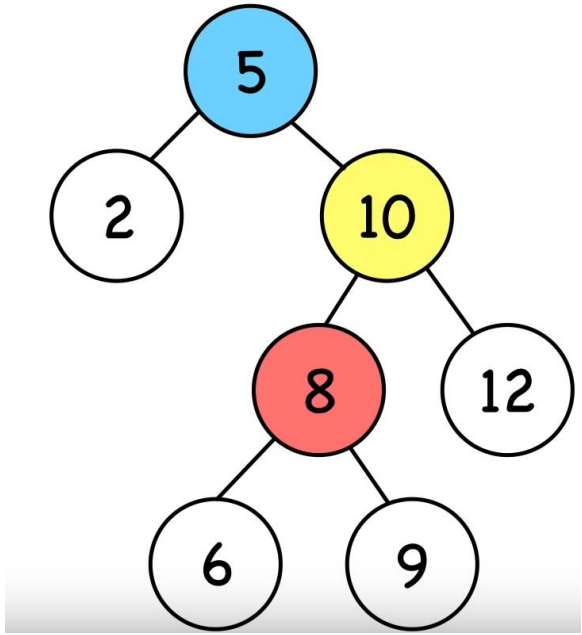
Műveletek

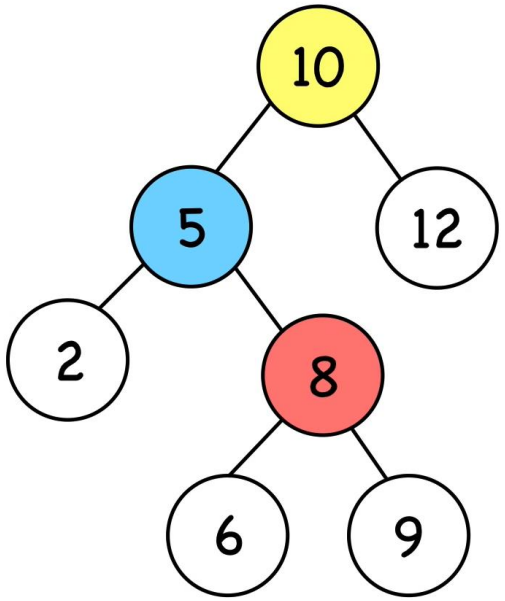
Forgatások

```
template <typename T>
void RotateLeft(tree<T>* rbt, node<T>* nd)
{
    if (nd->right->nil)
    {
        return;
    }
    node<T>* y = nd->right;

    nd->right = y->left;
    if (!y->left->nil)
    {
        y->left->parent = nd;
    }

    y->parent = nd->parent;

    if (nd->parent == nullptr) {
        rbt->root = y;
    }
    else if (nd == nd->parent->left) {
        nd->parent->left = y;
    }
    else {
        nd->parent->right = y;
    }
        y->left = nd;
        nd->parent = y;
}
```

Műveletek

Forgatások

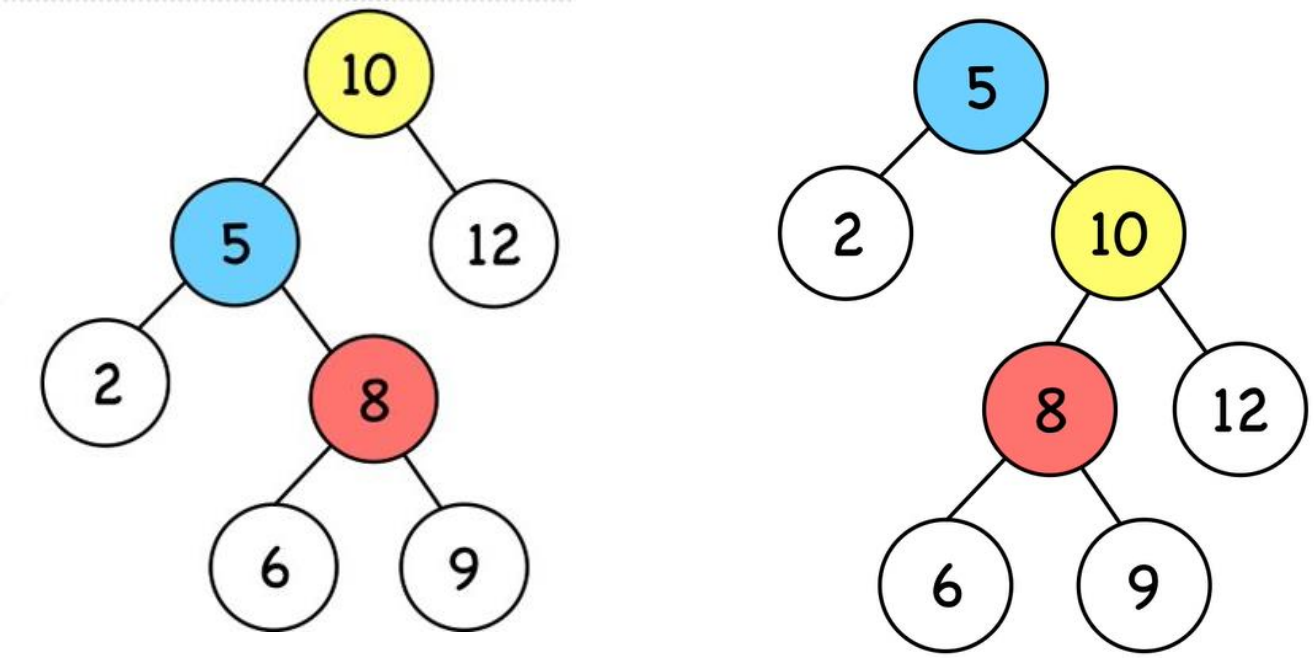





```
template <typename T>
void RotateLeft(tree<T>* rbt, node<T>* nd)
{
    if (nd->right->nil)
    {
        return;
    }
    node<T>* y = nd->right;

    nd->right = y->left;
    if (!y->left->nil)
    {
        y->left->parent = nd;
    }

    y->parent = nd->parent;

    if (nd->parent == nullptr) {
        rbt->root = y;
    }
    else if (nd == nd->parent->left) {
        nd->parent->left = y;
    }
    else {
        nd->parent->right = y;
    }
        y->left = nd;
        nd->parent = y;
}
```

Műveletek

Forgatások




```
void RotateRight(tree<T>* rbt, node<T>* nd)
{
    if (nd->left->nil)
    {
        return;
    }

    node<T>* y = nd->left;

    nd->left = y->right;

    if (!y->right->nil)
    {
        y->right->parent = nd;
    }

    y->parent = nd->parent;
    if (nd->parent == nullptr) {
        rbt->root = y;
    }
    else if (nd == nd->parent->left) {
        nd->parent->left = y;
    }
    else {
        nd->parent->right = y;
    }

    y->right = nd;
    nd->parent = y;
}
```

Műveletek

Elem hozzáadása

Ellenőrizzük, ha levél-e

Beszúrjuk a csomópontot kulcs szerint a helyére, ha gyökér -> feketére színezzük

Ha a nagybátyja piros, színt cserél a magyszülőfel, így feljebb kerül a probléma a fán belül

Ha fekete, forgatások következnek, ezt követően színezés, ha megszegtük a szabályokat

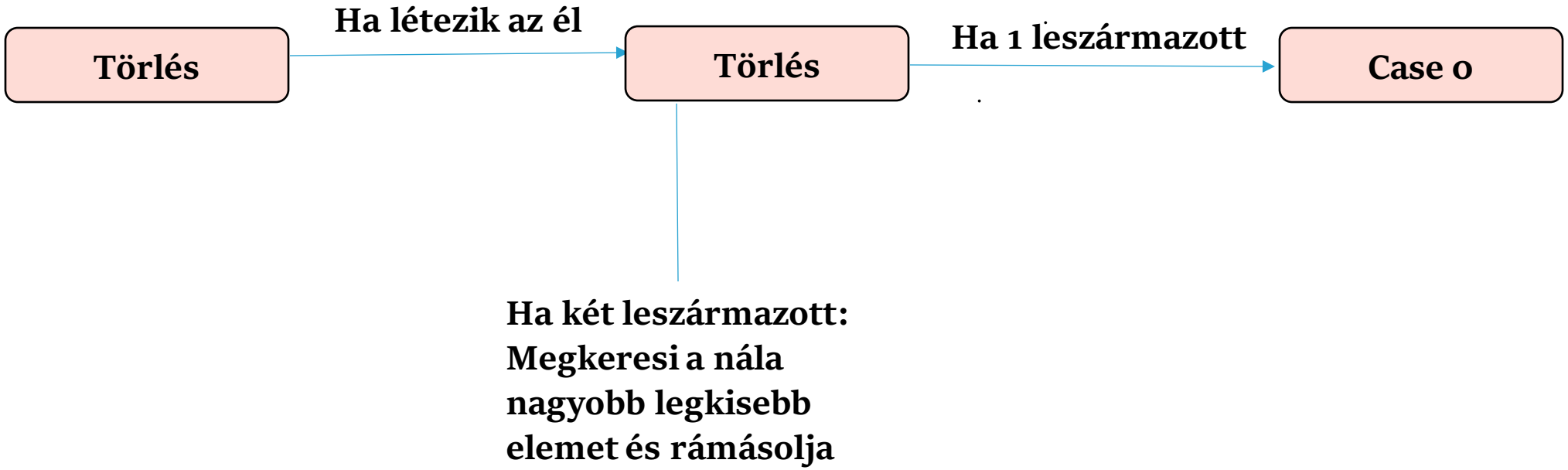
Végső színezés: szülőt feketére, nagyszülőt pirosra, és jobbra forgat a nagyszölő körül.

Műveletek

Törlés

```
//Töröl egy adott kulcsú elemet a fából
template <typename T>
void DeleteNode(tree<T>*, long long);

//Ha nincs ilyen él leáll, megnézi hány leszármazott van, ha csak1, caseö, ha tobb, megkeresi a legkisebb de az
//elemnél nagyobb elemet, ezt rámásolja a törlendőre
template <typename T>
void DeleteNode(tree<T>*, node<T>*, long long);

//megkeresi az egyetlen leszármazottat,felcseréli a kettőt,ha a csomópont piros, feketére szinezzük,
//ha fekete már, akkor case 1
template <typename T>
void DeleteCase0(tree<T>*, node<T>* );

//ha a gyökér a duplán fekete, akkor vége, ha nem, case2
template <typename T>
void DeleteCase1(tree<T>*, node<T>*);

//forgatást és színezést végez, majd hívja a case3-at
template <typename T>
void DeleteCase2(tree<T>*, node<T>*);

//Színez a testvért ha szükséges
template <typename T>
void DeleteCase3(tree<T>*, node<T>*);

//Hasonló a case3-hoy, csak más feltétel melett színez
template <typename T>
void DeleteCase4(tree<T>*, node<T>*);

//Forgatást végez,színek szerint
template <typename T>
void DeleteCase5(tree<T>*, node<T>*);

//További színezés és megfelelő forgatás
template <typename T>
void DeleteCase6(tree<T>*, node<T>*);
```

Törlés


Törlés
Ha létezik az él
Törlés
Ha 1 leszármazott
Case 0
Ha két leszármazott:
Megkeresi a nála
nagyobb legkisebb
elemet és rámásolja

Törlés

Törlés

Ha létezik az él

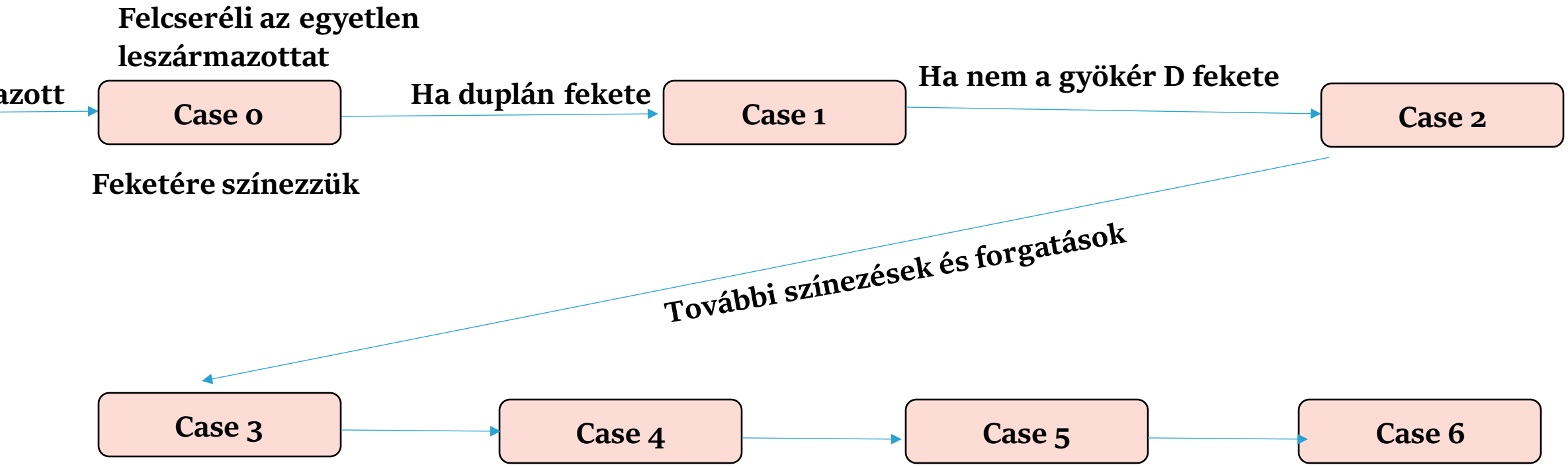
Törlés

Ha 1 leszármazott

Case 0

Ha két leszármazott:
Megkeresi a nála nagyobb legkisebb elemet és rámásolja

Törlés


Felcseréli az egyetlen leszármazottat
azott
Case 0
Ha duplán fekete
Case 1
Ha nem a gyökér D fekete
Case 2
Feketére színezzük
További színezések és forgatások
Case 3
Case 4
Case 5
Case 6

Feladat

Feladat

Egy város parkolóőrei számos panaszt kaptak a büntetési rendszerük miatt, mivel nehézkes volt nyomon követni, hogy melyik gépjárművet és mikor büntették meg, ami gyakran többszöri megbüntetést okozott ugyanannak a járműnek.

Az önkormányzat döntött egy új, hatékonyabb adatbázis létrehozásáról, amely lehetővé teszi a parkolóőröknek a megbízhatóbb és gyorsabb adatrögzítést és ellenőrzést.

Az adatbázisba azonnal feltöltik az összes megbüntetett járművet, és lehetővé teszi az ellenőrzést, hogy az adott jármű már kapott-e büntetést. Az adatbázisban meghatározott időintervallumon belül (5 nap) a gépjárművek mentesek a büntetéstől, és az adatbázisból automatikusan törölve lesznek a határidő után.

Feladat

```
struct car {
    string Rendszam;
    string Szin;
    string Marka;
    string Tipus;
    int nap, honap, ev, hatramaradt;
    long long kulcs;
};
```

Feladat

```
#include <iostream>
#include "RBTrees.h"
#include "RBTrees.cpp"
#include "Feladat.h"
#include <string>
#include <conio.h>
#include <fstream>
#include <vector>
```

ASZ fejlécek

Template

Feladat Fejlécek

Rendszámok, modellek tárolása

Pld _getch()

ASCII Art kiírása

Törlés optimalizálása

Feladat

```
void MoveCursor(int x, int y);
void Here(int option);
void Here2(int option);
```

Feladat

```
//Képet rajzol
void Sport(node<car>* nd) { ... }
//Képet rajzol
void Motor(node<car>* nd) { ... }
//Képet rajzol
void Kamion(node<car>* nd) { ... }
//Képet rajzol
void Antik(node<car>* nd) { ... }
//Képet rajzol
void Sima(node<car>* nd) { ... }
//Képet rajzol
void Terepjaro(node<car>* nd) { ... }
```

Feladat

```
cout << "Rendszam: " << nd->info.Rendszam << endl;
cout << "Szine: " << nd->info.Szin << endl;
cout << "Markaja: " << nd->info.Marka << endl;
cout << "Tipusa: " << nd->info.Tipus << endl;
cout << "Buntetes idopontja: " << nd->info.nap << ":" << nd->info.nap << ":" << nd->info.ev << endl;
cout << "Hatramaradt ido: " << nd->info.hatramaradt;
```

```
//A rendsyámból kulcsot késyít a csomópontoknak
long long ConvertToKey(string Rendszam)
{
    long long key=0;
    int H = Rendszam.length();
    for (int i = 0; i < H; i++)
    {
        if (Rendszam[i] <= 97)
            key = key * 100+Rendszam[i] - 32;
        else
            key = key * 100 + Rendszam[i];
    }
    return key;
}
```

Feladat

```
InsertNode<car>(rbt, kulcs, k);

IsIn<car>(rbt,ConvertToKey(Rendszam));

DeleteNode(rbt, ConvertToKey(rendszam));
```

Feladat

Miért Piros-Fekete Fával?

Feladat

Gyors keresés

Miért Piros-Fekete Fával?

Feladat

Gyors beillesztés és törlés

Gyors keresés

Miért Piros-Fekete Fával?

Feladat

Gyors beillesztés és törlés

Gyors keresés

Miért Piros-Fekete Fával?

Rendezettség

Feladat

Gyors beillesztés és törlés

Gyors keresés

Miért Piros-Fekete Fával?

Rendezettség

Támogatja a büntetéstől való mentesség időzítését

Kérdések?

Köszönöm a figyelmet!