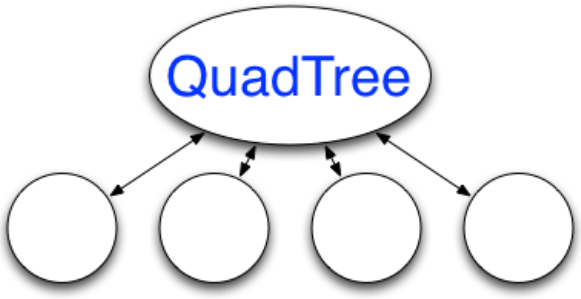
Házi feladat

Szoftver laboratórium 2.

Tóth András
(O8POUA)

Generikus duplán láncolt négy elágazású fa

Tartalom



1. Feladat

Tóth András (O8POUA) részére:

Készítsen GENERIKUS duplán láncolt 4 elágazású fát (quad-tree)! Valósítsa meg az összes értelmes műveletet operátor átdefiniálással (overload), de nem kell ragaszkodni az összes operátor átdefiniálásához! Amennyiben lehetséges használjon iterátort! Demonstrálja a működést külön modulként fordított tesztprogrammal! A programmal mutassa be a generikus szerkezet használatát több egyszerű adathalmazon, amit fájlból olvas be, és egy olyan saját osztályon, amely dinamikus adatot tartalmaz A megoldáshoz NE használjon STL tárolót vagy algoritmust! A tesztprogramot úgy specifikálja, hogy az parancssoros batch alkalmazásként (is) működjön, azaz a szabványos bemenetről olvasson, és a szabványos kimenetre, és/vagy a hibakimenetre írjon! Lehetősége van grafikus, vagy kvázi grafikus interaktív felhasználói felület kialakítására is, de fontos, hogy a Cporta rendszerbe olyan változatot töltsön fel, ami ezt nem használja! Amennyiben a feladat teszteléséhez fájlból, vagy fájlokból kell input adatot olvasnia, úgy a fájl neve *.dat alakú legyen!

2. Pontosított feladatspecifikáció

A feladat egy generikus duplán láncolt négy elágazású fa (továbbiakban négyfa, angolul quadtree) készítése. A négyfa egy olyan fa struktúra, amiben minden csúcsnak pontosan négy gyereke van. A négyfát leggyakrabban két-dimenziós tér felbontására használják oly módon, hogy a tér rekurzívan felbontható kisebb negyedekre. Ezek a területek leggyakrabban négyzetek, vagy téglalapok. A feladat nem specifikálja, hogy milyen módon lehessen használni a négyfát, ezért az előbbiekben leírt két-dimenziós tér felbontására lesz használható. A feladat specifikációja arra sem tér ki, hogy milyen objektumokkal valósítsam meg a fát. A négyfát ezért ezekkel az objektumokkal valósítom meg:

Point

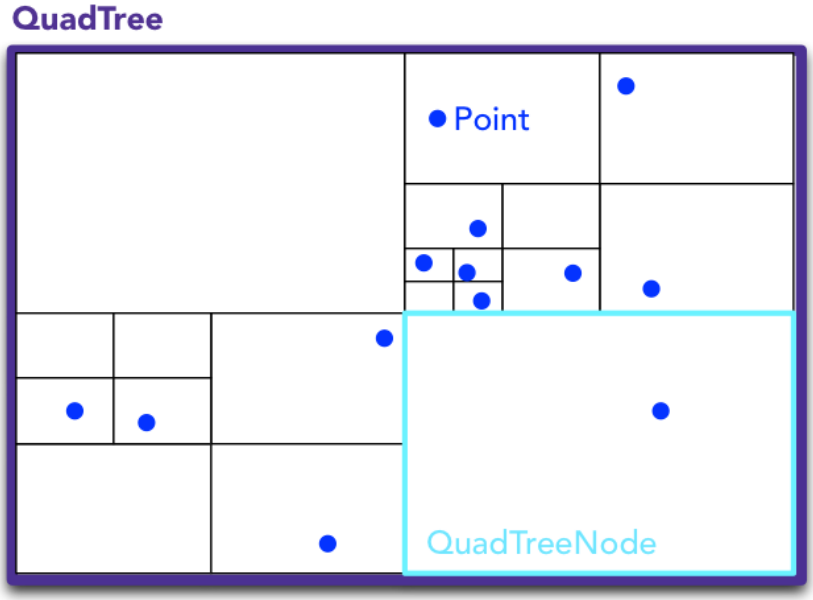
Az adatok pontokban tárolhatók el. A pontnak két koodrinátája van (x és y) és egy változója, amelyben a generikus adat tárolható.

QuadTree

A felhasználó ilyen QuadTree objektumokat hozhat létre. Az objektum elzárja a külvilág elől a fa felépítését.

A QuadTree tagfüggvényei:

- beszúrás
- mélység megszámolása
- elemek megszámolása
- keresés adott pont szerint
- keresés adat szerint



QuadTree::iterator

A QuadTree osztály iterátora. Segítségével a négyfa csúcsait járhatjuk be.

QuadTreeNode

A QuadTree osztály ilyen objektumokból építi fel a négyfát. Az osztály el van rejtve a külvilág elől.

3. Terv

A generikus négy elágazású fa az alábbi osztályokból épül fel:

Point adattagjai és tagfüggvényei

adattagjai:

- double x
- double y
- T data

(x, y vízszintes és függőleges koordináták, data a generikus adat)

tagfüggvényei:

- konstruktor
- destruktor
- operator==
- operator!=
- getData (visszatér a tárolt generikus adattal)

QoudTree adattagjai és tagfüggvényei

adattagjai:

- root (fa gyökerére mutató pointer)
- iterator

tagfüggvényei:

- konstruktor
- destruktor
- insert (pont beszúrása)
- depth (mélység visszaadása)
- countNodes (csúcsom megszámolása)
- find (pont keresése)
- find (adat keresése)
- begin() (az első elemre mutató iterator)
- end() (az utolsó utáni elemre mutató iterator)

QuadTree::iterator adattagja és tagfüggvényei

adattagjai:

- node (jelenlegi elem)

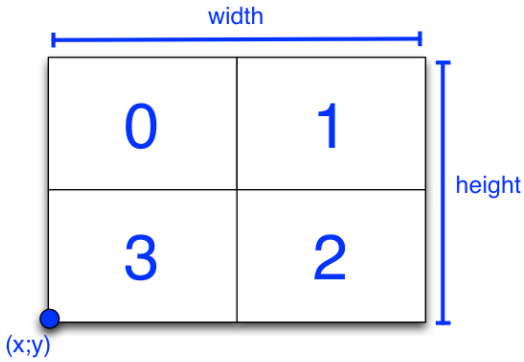
tagfüggvényei:

- konstruktor
- operator*
- operator->
- operator++ (prefix és postfix)
- operator==
- operator!=

QuadTreeNode adattagjai és tagfüggvényei

adattagjai:

- parent (szülőre mutató pointer)
- children[4] (gyerekekre mutató pointer)
- point (tárolt pontok dinamikus tömbje)
- number_of_points (pontok száma)
- x, y, width, height (területre jellmző adatok)
- level (fában lévő szintje)
- MAX_LEVEL (szintek maximális száma)



tagfüggvényei:

- konstruktor
- destruktor
- split (gyerekek létrehozása megfelelő adattagokkal)
- insert (pont beszúrása)
- hasData (igaz, ha van pont/adat a csúcsban)
- getLevel
- isLeaf (igaz, ha levél)

Iterátor működése (négyfa postorder bejárása)

1. A legbaloldalibb levéltől indul.

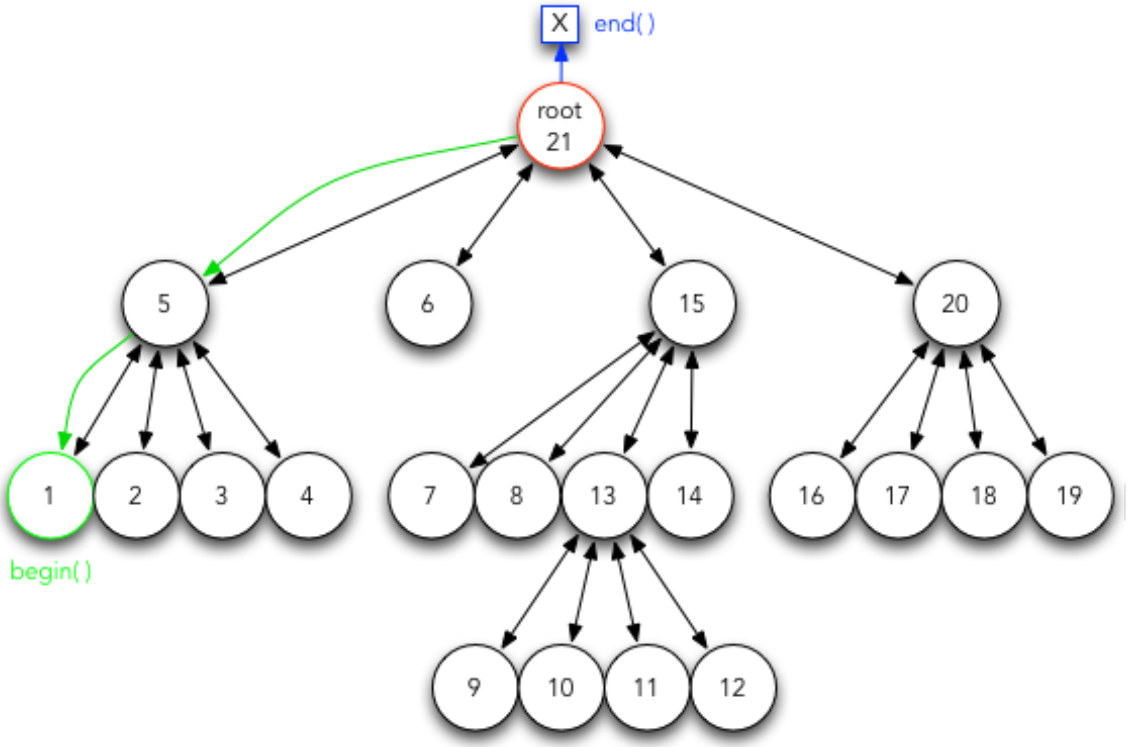
2.a A szomszédos csúcsra a következő elem, ha az levél.

2.b A szomszédos csúcs legbaloldalibb levelél a következő elem, ha létezik.

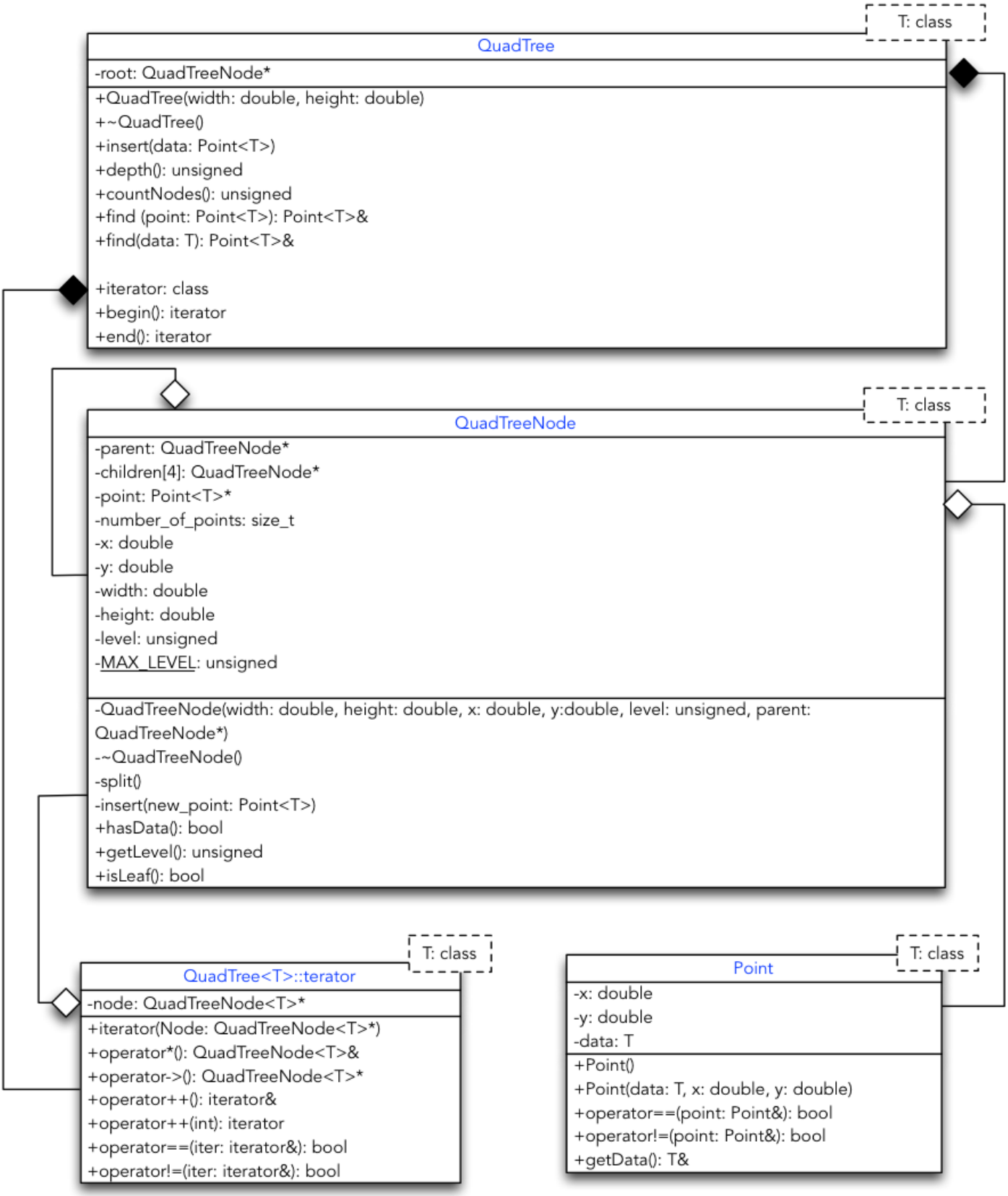
2.c Ha a negyedik csúcsnál vagyunk, akkor a következő elem a szülő.

3. Az utolsó elem a fa gyökere.

Iterátor bejárásának szemléltetése ábrán:
(számok a sorrend)



Osztálydiagram



4. Megvalósítás

Algoritmusok:

- split() – gyerekek létrehozása megfelelő adattagokkal

```
/// @brief Terület felbontása négy egybevágó téglalapra.
template <class T>
void QuadTreeNode<T>::split(){
    /// Bal felső.
    children[0]=new QuadTreeNode<T>(width/2, height/2, x, y+height/2, level+1, this);
    /// Jobb felső.
    children[1]=new QuadTreeNode<T>(width/2, height/2, x+width/2, y+height/2, level+1, this);
    /// Jobb alsó.
    children[2]=new QuadTreeNode<T>(width/2, height/2, x+width/2, y, level+1, this);
    /// Bal alsó.
    children[3]=new QuadTreeNode<T>(width/2, height/2, x, y, level+1, this);
}
```

- QuadTree beszúró függvénye

```
/// @brief Új elem beszúrása.
/// @param point - Új pont / adat.
template <class T>
void QuadTree<T>::insert(Point<T> point){
    /// Ha a pont kívül esik a területről, akkor kivételt dob.
    if (point.x > root->x+root->width || point.y > root->y+root->height)
        throw "This point can't be inserted.";
    /// Egyébként keressük meg azt a levelet ahova be kéne szúrni az új elemet.
    QuadTreeNode<T> *temp=root;
    while (!temp->isLeaf()){
        size_t i=3;
        if (point.x <= temp->x+temp->width/2 && point.y > temp->y+temp->height/2) i=0;
        else if (point.x > temp->x+temp->width/2 && point.y >= temp->y+temp->height/2) i=1;
        else if (point.x > temp->x+temp->width/2 && point.y <= temp->y+temp->height/2) i=2;
        temp=temp->children[i];
    }
    /// Szúrjuk be az elemet.
    temp->insert(point);
}
```

- QuadTreeNode beszúró függvénye

```
/// @brief Új pont / adat beszúrása.
/// @param new_point - Új pont.
template <class T>
void QuadTreeNode<T>::insert(Point<T> new_point){
    /// Ha nincs meglévő adat:
    if (!hasData()){
        this->point = new Point<T>[1];
        point[0] = new_point;
        number_of_points=1;
    }
    /// Ha van meglévő adat és még nem érte el a maximális mélységet a fa:
    else if (hasData() && level != MAX_LEVEL){
        /// Felbontás.
        split();
        /// Beszúrjuk a meglévő adatot.
        size_t i=3;
        if (point->x <= x+width/2 && point->y > y+height/2) i=0;
        else if (point->x > x+width/2 && point->y >= y+height/2) i=1;
        else if (point->x > x+width/2 && point->y <= y+height/2) i=2;
        children[i]->insert(*point);
        delete[] point;
        point=NULL;
        number_of_points=0;
        /// Beszúrjuk az új adatot.
        i=3;
        if (new_point.x <= x+width/2 && new_point.y > y+height/2) i=0;
        else if (new_point.x > x+width/2 && new_point.y >= y+height/2) i=1;
        else if (new_point.x > x+width/2 && new_point.y <= y+height/2) i=2;
        children[i]->insert(new_point);
    }
    /// Ha van meglévő adat, de elérte a maximális mélységet a fa:
    else{
        /// Létre kell hozni egy pontokat tároló tömböt amiben egyel több elemet tárolhatunk el.
        Point<T> *temp = new Point<T>[number_of_points+1];
        size_t i;
        /// Átmásolja a meglévő elemeket az új tömbbe.
        for (i=0; i<number_of_points; ++i)
            temp[i]=point[i];
        /// Az új tömbbe berakja az új pontot.
        temp[number_of_points]=new_point;
        /// A csomópontban megnöveli a pontok számát tároló változót.
        ++number_of_points;
        /// A régi tömböt törli.
        delete[] point;
        point = temp;
    }
}
```

- Keresés adott pont szerint.
  (A keresés adott pont szerint a négyfa osztály használatának legnagyobb előnye. Sokkal gyorsabban lehet megkeresni egy adott pontot, mert nem kell megnézni minden egyes csomópontot a fában, csak azokat ahol a pont lehet.)

```
268 /// @brief Adott pont megkeresése. Lehetőség van az adott pontban lévő adat megváltoztatására.
269 /// @param point - Pont, melyet keresünk.
270 /// @return A keresett pont.
271 template <class T>
272 Point<T>& QuadTree<T>::find(Point<T> point){
273     QuadTreeNode<T> *temp=root;
274     while (!temp->isLeaf()){
275         size_t i=3;
276         if (point.x <= temp->x+temp->width/2 && point.y > temp->y+temp->height/2) i=0;
277             else if (point.x > temp->x+temp->width/2 && point.y >= temp->y+temp->height/2) i=1;
278                 else if (point.x > temp->x+temp->width/2 && point.y <= temp->y+temp->height/2) i=2;
279                     temp=temp->children[i];
280                     }
281     for (size_t i=0; i<temp->number_of_points; ++i)
282         if (point==temp->point[i]) return temp->point[i];
283     throw "Point not found";
284 }
```

- Keresés adat szerint.

```
286 /// @brief Adott adatot tároló pont megkeresése. Lehetőség van az adott pontban lévő adat megváltoztatására.
287 /// @param data - Adat, melyet keresünk.
288 /// @return Az első olyan pont, amely ezt az adatot tartalmazza.
289 template <class T>
290 Point<T>& QuadTree<T>::find(T data){
291     for (typename QuadTree<T>::iterator iter=QuadTree<T>::begin(); iter!=QuadTree<T>::end(); ++iter)
292         for (size_t i=0; i<iter->number_of_points; ++i)
293             if (iter->point[i].data==data)
294                 return find(Point<T>(data, iter->point[0].x, iter->point[0].y));
295                 throw "Point not found";
296 }
```

5. Tesztprogram

A feladat kiírása szerint elkészítettem egy dinamikus adatokat tartalmazó String osztályt. A tesztelés során ezt az osztály is felhasználom.

A tesztprogrammal a következőket tesztelem:

- Négyfa létrehozása, adatok beszúrása.

```
// Egészeket tartalmazó négyfa létrehozása.
QuadTree<int> nTree(10, 10);
// Új elemek beszúrása.
nTree.insert(Point<int>(23, 2, 3));
nTree.insert(Point<int>(26, 2, 6));
nTree.insert(Point<int>(63, 6, 3));
nTree.insert(Point<int>(66, 6, 6));
nTree.insert(Point<int>(99, 9, 9));
```

- Beszúrás olyan helyre (x, y), ahol már található adat.

```
try {
    // Beszúrás olyan helyre, ahol már található meglévő adat.
    nTree.insert(Point<int>(99.1, 9, 9));
} catch (...) {
    std::cout << "Hiba beszúrásnál."; // Nem szabad, hogy hiba történjen.
}
```

- Elemek kiírása.

```
// Elemek kiírása.
std::cout << "Int-eket tartalmazó fában lévő elemek:\n" << nTree << std::endl;
```

- Fa mélységének megszámolása.

```
// Fa mélységének kiírása.
std::cout << "Fa mélysége: " << nTree.depth() << std::endl << std::endl;
```

- Kivételkezelés vizsgálata külső pont beszúrása esetén.

```
// Kivételkezelés vizsgálata.
std::cout << "Kivételkezelésének vizsgálata külső pont beszúrása esetén:\n";
try {
    nTree.insert(Point<int>(100, 11, 5)); // Külső pont nem szúrható be.
} catch (const char* c) {
    std::cout << c << std::endl;
}
```

- Keresés a fában adott pont szerint.
- Adott pontban lévő adat átírása.
- Keresés a fában adott adat szerint, kivételkezelés vizsgálata.

```
// Keresés a fában.
try {
    nTree.find(Point<int>(66, 6, 6)).getData()=11; // Pont megkeresése, tárolt adat átírása.
    std::cout << nTree.find(11); // Sikerült átírni.
    std::cout << nTree.find(66); // Már nincs ilyen pont.
} catch (const char* c) {
    std::cout << c << std::endl;
}
```

- Beolvasás fájlból négyfába.
  (Karaktereket tartalmazó fába és Stringeket tartalmazó fába.)

```
// Fájlból beolvasás.
QuadTree<char> chTree(150, 150);
QuadTree<String> sTree(5.1, 4.8);
try {
    std::fstream File;
    // Char-t tartalmazó fába beolvasás.
    File.open(input_ch);
    File >> chTree;
    File.close();
    // String-et tartalmazó fába beolvasás.
    File.open(input_s);
    File >> sTree;
    File.close();
} catch (std::exception& e) {
    std::cout << e.what() << std::endl;
}
std::cout << std::endl << "Fájlból beolvasott char-t tartalmazó fában lévő elemek:\n" << chTree;
std::cout << std::endl << "Fájlból beolvasott String-et tartalmazó fában lévő elemek:\n" << sTree;
```