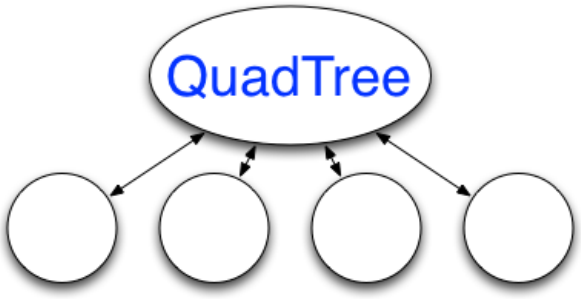
Házi feladat

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

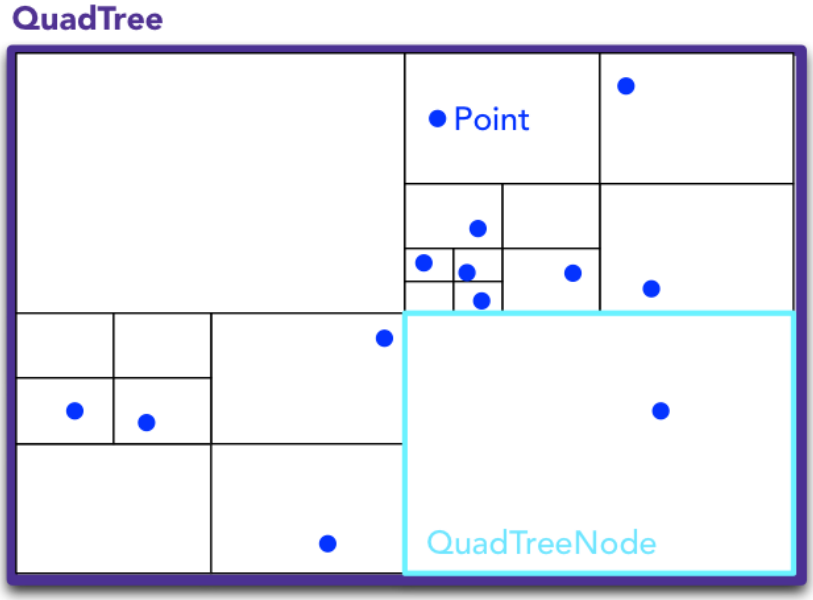
Az adatok pontokban tárolhatók el. A pontnak két koodrinátája van (x és y) és egy változója, amelyben a generikus adat tárolható.

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)

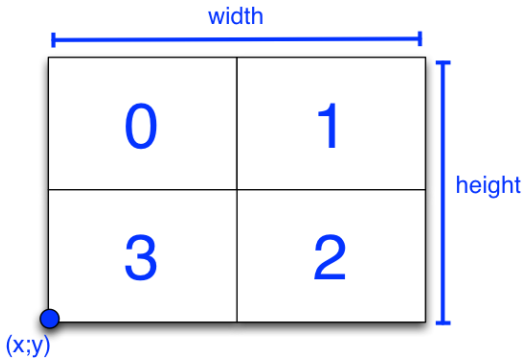
tagfüggvényei:

- 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.

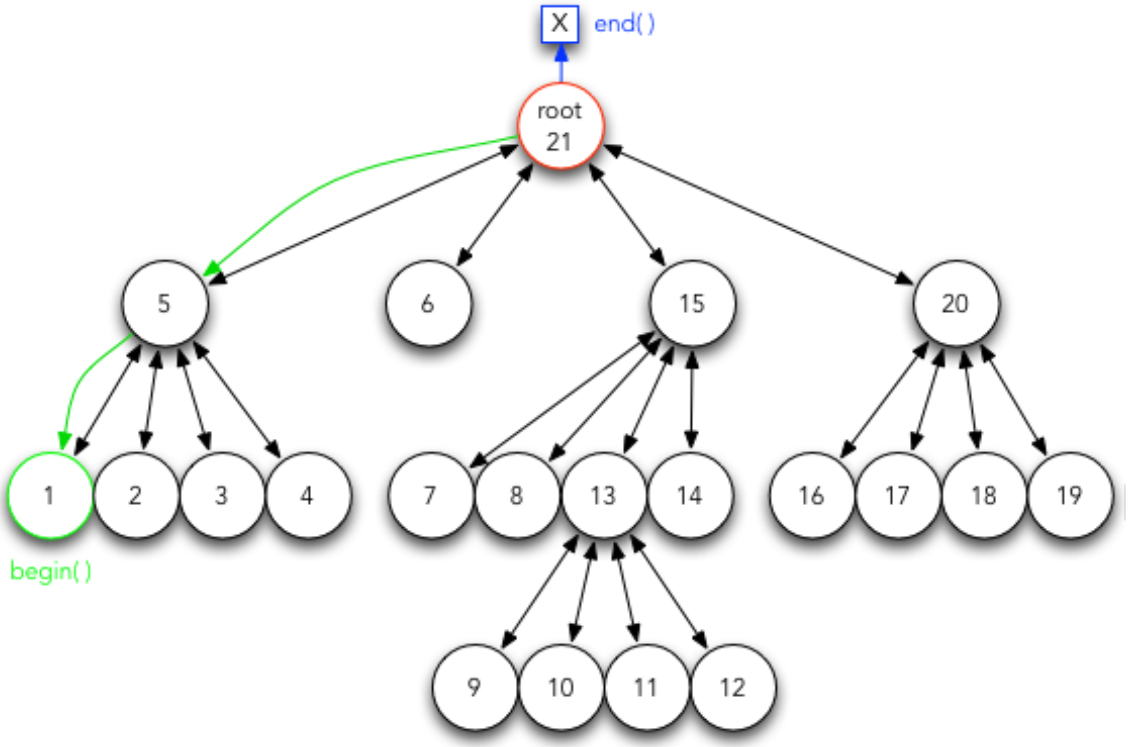
2.a A szomszédos csúcsra a következő elem, ha az levél.

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

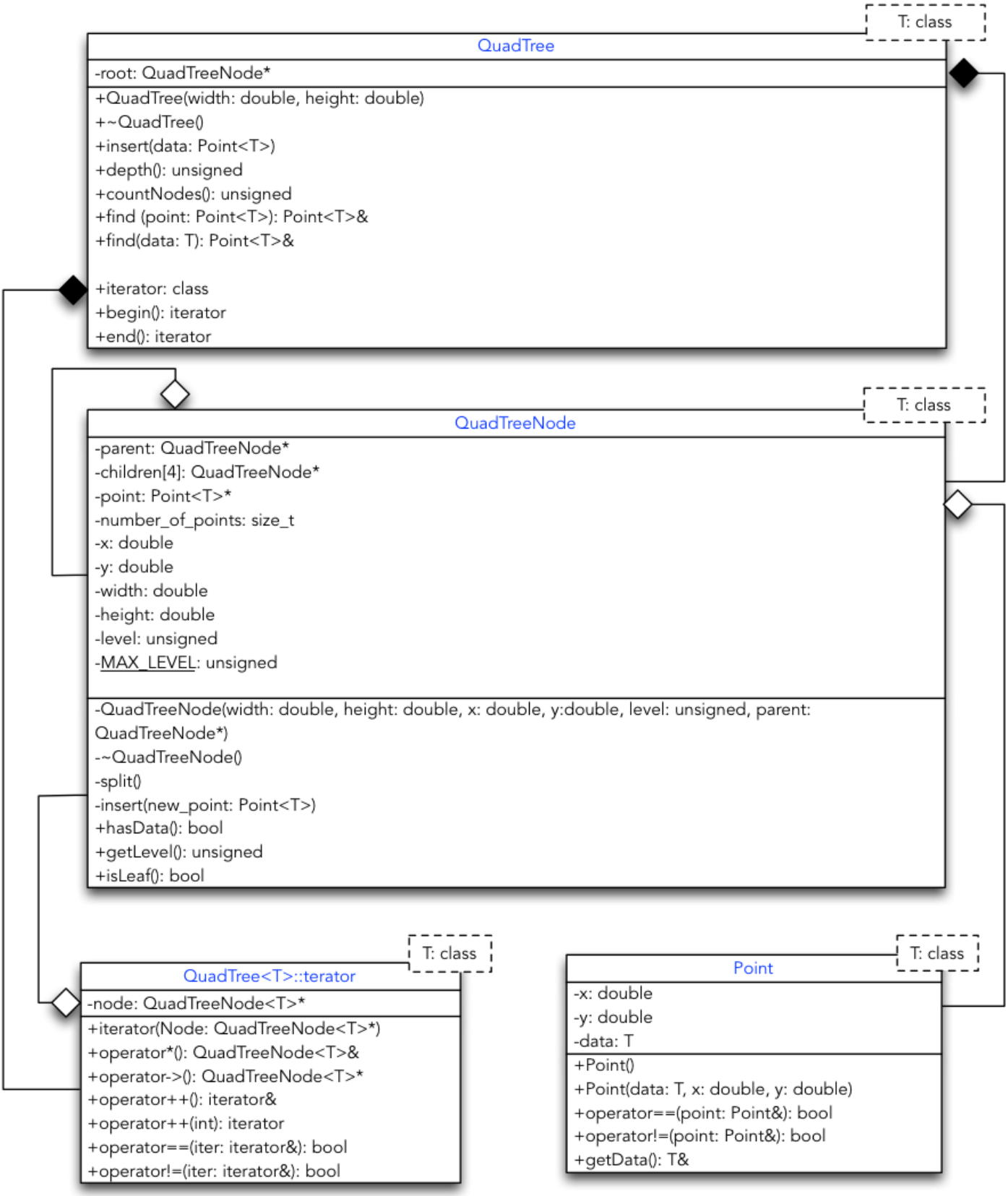
2.c Ha a negyedik csúcsnál vagyunk, akkor a következő elem a szülő.

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;
```

Dokumentáció

Készítette Doxygen 1.8.3.1

Sun May 12 2013 15:33:53

Tartalomjegyzék

1. fejezet

NHF 2013 - QuadTree (Négy elágazású duplán láncolt generikus fa)

Feladat kiírás: 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!

Szerző

Tóth András (O8POUA)

2. fejezet

Osztálymutató

2.1. Osztálylista

Az összes osztály, struktúra, unió és interfész listája rövid leírásokkal:

3. fejezet

Fájlmutató

3.1. Fájllista

Az összes dokumentált fájl listája rövid leírásokkal:

4. fejezet

Osztályok dokumentációja

4.1. QuadTree< T >::iterator osztályreferencia

Iterátor.

`#include <QuadTree.hpp>`

Publikus tagfüggvények

- iterator (QuadTreeNode< T > *node=NULL)
  Konstruktor.
- QuadTreeNode< T > & **operator*** ()
- QuadTreeNode< T > * **operator->** ()
- iterator & operator++ ()
  Prefix operator++.
- iterator operator++ (int)
  Postfix operator++.
- bool operator== (const iterator &iter)
  Egyenlőség operátor.
- bool operator!= (const iterator &iter)

Védett attribútumok

- QuadTreeNode< T > * **node**

4.1.1. Részletes leírás

template<class T>class QuadTree< T >::iterator

Iterátor.

4.1.2. Tagfüggvények dokumentációja

4.1.2.1. template<class T> bool QuadTree< T >::iterator::operator!= (const iterator & *iter*) [inline]

Visszatérési érték

Igaz, ha nem egyezik meg a csomópont.

4.1.2.2. template<class T > QuadTree< T >::iterator & QuadTree< T >::iterator::operator++ ( )

Prefix operator++.

Visszatérési érték

Következő elem.

Ha a gyökér elemnél vagyunk (csak annak nincs szülője), akkor végigértünk az összes elemen.

A szülő hányadig gyerekén állunk?

Ha a következő gyerek levél, akkor ez lesz a következő elem.

Ha a következő gyerek nem levél, akkor a következő elem a legbaloldalibb levél a következő gyerek alatt.

Egyébként a szülő a következő elem.

4.1.2.3. template<class T> bool QuadTree< T >::iterator::operator== (const iterator & *iter*) [inline]

Egyenlőség operátor.

Visszatérési érték

Igaz, ha megegyezik az csomópont.

Ez a dokumentáció az osztályról a következő fájl alapján készült:

- QuadTree.hpp

4.2. Point< T > osztálysablon-referencia

Point (pont) osztály. A négy elágazású generikus fát ilyen pontokkal tölthetjük fel.

`#include <QuadTree.hpp>`

Publikus tagfüggvények

- Point ()

  Default konstruktor.
- Point (T data, double x=0, double y=0)

  Konstruktor adattal.
- bool operator== (Point &point)

  operator==, két pont egyenlősége vizsgálható. Igaz, ha a két pont minden adata megegyezik.
- bool operator!= (Point &point)

  operator!=, két pont egyenlősége vizsgálható. Igaz, ha a két pont valamelyik adata nem egyezik meg.
- T & **getData** ()

Barátok

- class **QuadTree**< **T** >
- class **QuadTreeNode**< **T** >
- std::ostream & **operator** (std::ostream &, const Point &)

4.2.1. Részletes leírás

template<class T>class Point< T >

Point (pont) osztály. A négy elágazású generikus fát ilyen pontokkal tölthetjük fel.

Ez a dokumentáció az osztályról a következő fájl alapján készült:

- QuadTree.hpp

4.3. QuadTree< T > osztálysablon-referencia

QuadTree (generikus négy elágazású fa) osztály. A négy elágazású generikus fa QuadTreeNode-okból épül fel.

`#include <QuadTree.hpp>`

Osztályok

- class iterator

  Iterátor.

Publikus tagfüggvények

- QuadTree (double width=100, double height=100)

  Konstruktor.
- ~QuadTree ()

  Destruktor.
- void insert (Point< T > data)

  Új elem beszúrása.
- unsigned depth () const

  Fa mélységének visszaadása.
- unsigned countNodes () const

  Fában lévő elemek megszámolása.
- Point< T > & find (Point< T > point)

  Adott pont megkeresése. Lehetőség van az adott pontban lévő adat megváltoztatására.
- Point< T > & find (T data)

  Adott adatot tároló pont megkeresése. Lehetőség van az adott pontban lévő adat megváltoztatására.
- iterator begin () const
- iterator end () const

Barátok

- class **QuadTreeNode< T >**
- std::ostream & **operator** (std::ostream &, const QuadTree< T > &)
- std::istream & operator>> (std::istream &is, QuadTree< T > &quadtree)

  Beolvasás. Az beolvasandó adatokat "(x;y): adat" alakban kapjuk.

4.3.1. Részletes leírás

template<class T>class QuadTree< T >

QuadTree (generikus négy elágazású fa) osztály. A négy elágazású generikus fa QuadTreeNode-okból épül fel.

4.3.2. Tagfüggvények dokumentációja

4.3.2.1. template<class T> iterator QuadTree< T >::begin ( ) const [inline]

Visszatérési érték

- Legbaloldalibb levél.

4.3.2.2. template<class T> unsigned QuadTree< T >::countNodes ( ) const [inline]

Fában lévő elemek megszámolása.

Visszatérési érték

Fa elemszáma.

4.3.2.3. template<class T > unsigned QuadTree< T >::depth ( ) const

Fa mélységének visszaadása.

Visszatérési érték

Fa mélysége.

4.3.2.4. template<class T> iterator QuadTree< T >::end ( ) const [inline]

Visszatérési érték

Az utolsó elem után mutat.

4.3.2.5. template<class T > Point< T > & QuadTree< T >::find (Point< T > *point*)

Adott pont megkeresése. Lehetőség van az adott pontban lévő adat megváltoztatására.

Paraméterek

point	- Pont, melyet keresünk.

Visszatérési érték

A keresett pont.

4.3.2.6. template<class T > Point< T > & QuadTree< T >::find (T *data*)

Adott adatot tároló pont megkeresése. Lehetőség van az adott pontban lévő adat megváltoztatására.

Paraméterek

data	- Adat, melyet keresünk.

Visszatérési érték

Az első olyan pont, amely ezt az adatot tartalmazza.

4.3.2.7. template<class T > void QuadTree< T >::insert (Point< T > *point*)

Új elem beszúrása.

Visszatérési érték

- Gyökérre mutató pointer.

Paraméterek

data	- Új pont / adat.
point	- Új pont / adat.

Ha a pont kívül esik a területről, akkor kivételt dob.

Egyébként keressük meg azt a levelet ahova be kéne szúrni az új elemet.

Szúrjuk be az elemet.

Ez a dokumentáció az osztályról a következő fájl alapján készült:

- QuadTree.hpp

4.4. QuadTreeNode< T > osztálysablon-referencia

QuadTreeNode (négy elágazású generikus fa csomópontja) osztály. A négy elágazású generikus fa ilyen csomópontokból épül fel.

`#include <QuadTree.hpp>`

Barátok

- class **QuadTree**< **T** >
- std::ostream & **operator** (std::ostream &, const QuadTreeNode &)

4.4.1. Részletes leírás

template<class T>class QuadTreeNode< T >

QuadTreeNode (négy elágazású generikus fa csomópontja) osztály. A négy elágazású generikus fa ilyen csomópontokból épül fel.

Ez a dokumentáció az osztályról a következő fájl alapján készült:

- QuadTree.hpp

4.5. String osztályreferencia

String osztály. A 'pData'-ban vannak a karakterek (a lezáró nullával együtt), 'len' a hossza. A hosszba nem számít bele a lezáró nulla.

`#include <String.h>`

Publikus tagfüggvények

- String (char ch)

  Konstruktor: egy karakterből.
- String (const char *p="")

  Konstruktor: egy egy nullával lezárt char sorozatból, ez a default konstruktor is.
- const char * c_str () const

  C stringet ad vissza.
- String (const String &s1)

  Másoló konstruktor.
- ~String ()

  Destruktor.
- String & operator= (const String &rhs_s)
- String operator+ (const String &rhs_s) const

  Két Stringet összefűz.
- char & operator[] (unsigned int idx)

  A string egy megadott indexű elemének referenciájával tér vissza.
- const char & operator[] (unsigned int idx) const

  A string egy megadott indexű elemének referenciájával tér vissza.

Barátok

- std::ostream & operator<< (std::ostream &os, const String &s0)

  Kiíró operátor.
- std::istream & operator>> (std::istream &is, String &s0)

  Beolvas az istream-ről egy szót egy String-be.

4.5.1. Részletes leírás

String osztály. A 'pData'-ban vannak a karakterek (a lezáró nullával együtt), 'len' a hossza. A hosszba nem számít bele a lezáró nulla.

String osztály deklarációja.

Szerző

Tóth András (O8POUA)

4.5.2. Konstruktorok és destruktorok dokumentációja

4.5.2.1. String::String (char *ch*)

Konstruktor: egy karakterből.

Paraméterek

Ch	egy karakter.

4.5.2.2. String::String (const char * *p* = " ")

Konstruktor: egy egy nullával lezárt char sorozatból, ez a default konstruktor is.

Paraméterek

p	Pointer a C stringre.

4.5.2.3. String::String (const String & *s1*)

Másoló konstruktor.

Paraméterek

s1	String, amiből létrehozzuk az új String-et.

4.5.3. Tagfüggvények dokumentációja

4.5.3.1. const char* String::c_str ( ) const [inline]

C stringet ad vissza.

Visszatérési érték

Nullával lezárt karaktersorozatra mutató pointer.

4.5.3.2. String String::operator+ (const String & *rhs_s*) const

Két Stringet összefűz.

Paraméterek

rhs_s	jobboldali String.

Visszatérési érték

Új String, ami tartalmazza a két stringet egymás után.

4.5.3.3. String & String::operator= (const String & *rhs_s*)

Paraméterek

rhs_s	jobboldali String.

Visszatérési érték

Baoldali string.

4.5.3.4. char & String::operator[] (unsigned int *idx*)

A string egy megadott indexű elemének referenciájával tér vissza.

Paraméterek

idx	Karakter indexe.

Visszatérési érték

Karakter referenciája. Indexelési hiba esetén const char* kivételt dob.

4.5.3.5. const char & String::operator[] (unsigned int *idx*) const

A string egy megadott indexű elemének referenciájával tér vissza.

Paraméterek

idx	Karakter indexe.

Visszatérési érték

Karakter referenciája. Indexelési hiba esetén const char* kivételt dob.

4.5.4. Barát és kapcsolódó függvények dokumentációja

4.5.4.1. std::ostream& operator<< (std::ostream & *os,* const String & *s0*) [friend]

Kiíró operátor.

Paraméterek

os	Ostream típusú objektum.
s0	String, amit kiírunk.

Visszatérési érték

os

4.5.4.2. std::istream& operator>> (std::istream & *is,* String & *s0*) [friend]

Beolvas az istream-ről egy szót egy String-be.

Paraméterek

is	Istream típusú objektum.
s0	String, amibe beolvas.

Visszatérési érték

is

Ez a dokumentáció az osztályról a következő fájlok alapján készült:

- String.h
- String.cpp

5. fejezet

Fájlok dokumentációja

5.1. QuadTree.hpp fájlreferencia

Osztályok

- class Point< T >

  Point (pont) osztály. A négy elágazású generikus fát ilyen pontokkal tölthetjük fel.
- class QuadTree< T >

  QuadTree (generikus négy elágazású fa) osztály. A négy elágazású generikus fa QuadTreeNode-okból épül fel.
- class QuadTreeNode< T >

  QuadTreeNode (négy elágazású generikus fa csomópontja) osztály. A négy elágazású generikus fa ilyen csomópontokból épül fel.
- class Point< T >

  Point (pont) osztály. A négy elágazású generikus fát ilyen pontokkal tölthetjük fel.
- class QuadTreeNode< T >

  QuadTreeNode (négy elágazású generikus fa csomópontja) osztály. A négy elágazású generikus fa ilyen csomópontokból épül fel.
- class QuadTree< T >

  QuadTree (generikus négy elágazású fa) osztály. A négy elágazású generikus fa QuadTreeNode-okból épül fel.
- class QuadTree< T >::iterator

  Iterátor.

Függvények

- template<class T >
  std::ostream & operator<< (std::ostream &os, const Point< T > &point)

  Point (pont) kiírása "(x;y): adat" alakban.
- template<class T >
  std::ostream & operator<< (std::ostream &os, const QuadTree< T > &quadtree)

  Fa kiírása iterátor használatával.
- template<class T >
  std::ostream & operator<< (std::ostream &os, const QuadTreeNode< T > &node)

  Csomópont kiírása.

5.1.1. Függvények dokumentációja

5.1.1.1. template<class T > std::ostream & operator<< (std::ostream & *os,* const QuadTreeNode< T > & *node*)

Csomópont kiírása.

Ha van adat, akkor ki kell írni.

Tárgymutató

```
/// @file QuadTree.hpp
/// @mainpage NHF 2013 - QuadTree (Négy elágazású duplán láncolt generikus fa)
/// Feladat kiírás:
/// 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!
///  @author Tóth András (O8POUA)

template <class T>
class Point;

template <class T>
class QuadTree;

template <class T>
class QuadTreeNode;

template <class T>
std::ostream& operator<<(std::ostream &,const Point<T> &);

template <class T>
std::ostream& operator<<(std::ostream &,const QuadTree<T> &);

template <class T>
std::ostream& operator<<(std::ostream &,const QuadTreeNode<T> &);

/// @brief Point (pont) osztály. A négy elágazású generikus fát ilyen
    pontokkal tölthetjük fel.
template <class T>
class Point{
    /// @brief Az adat helye.
    double x, y;
    /// @brief Az adat.
    T data;

public:
    /// @brief Default konstruktor.
    Point(): x(0), y(0) {}

    /// @brief Konstruktor adattal.
    Point(T data, double x=0, double y=0): data(data), x(x), y(y) {}

    /// @brief operator==, két pont egyenlősége vizsgálható. Igaz, ha a két
        pont minden adata megegyezik.
    bool operator==(Point & point){
        return (x==point.x && y==point.y && data==point.data);
    }

    /// @brief operator!=, két pont egyenlősége vizsgálható. Igaz, ha a két
        pont valamelyik adata nem egyezik meg.
    bool operator!=(Point & point){
        return !((*this) == point);
    }

    T& getData(){return data;}

    friend class QuadTree<T>;
```

```
    friend class QuadTreeNode<T>;
    friend std::ostream& operator<< <T>(std::ostream &, const Point &);
};

/// @brief QuadTreeNode (négy elágazású generikus fa csomópontja) osztály. A
    négy elágazású generikus fa ilyen csomópontokból épül fel.
template <class T>
class QuadTreeNode{
    /// @brief Szülő.
    QuadTreeNode *parent;

    /// @brief Négy gyerek.
    QuadTreeNode *children[4];

    /// @brief Pont, melyben az adatot tárolhatjuk.
    Point<T> *point;

    /// @brief Pontok száma.
    size_t number_of_points;

    /// @brief A területre jellemző adatok. (x ; y) pont a téglalap bal alsó
        pontja.
    double x, y, width, height;

    /// @brief Melyik szinten található a fában? A gyökér van az 1. szinten.
    unsigned level;

    /// @brief Milyen mélységig nőhet a fa? Ezzel a statikus változóval lehet
        beállítani.
    static unsigned MAX_LEVEL;

    /// @brief Nincs paraméter nélküli konstruktora.
    QuadTreeNode();

    /// @brief QuadTree és QuadTreeNode hozhat csak létre új csomópontot.
        (Adat nélkül)
    QuadTreeNode(double width, double height, double x=0, double y=0, unsigned
        level=1, QuadTreeNode *parent=NULL): parent(parent), children{NULL,
        NULL, NULL, NULL}, width(width), height(height), x(x), y(y), level
        (level), point(NULL), number_of_points(0){}

    /// @return A csomópont rendelkezik-e adattal?
    bool hasData() const{
        return number_of_points==0 ? false : true;
    }

    /// @brief Terület felbontása négy egybevágó téglalapra.
    void split();

    /// @brief Új pont / adat beszúrása.
    void insert(Point<T> new_point);

    /// @brief Csomópontról eldönti hogy levél-e.
    /// @return Levél?
    bool isLeaf() const{
        return (children[0]==NULL && children[1]==NULL && children[2]==NULL &&
            children[3]==NULL);
    }
```

```
    /// @brief Destruktor.
    ~QuadTreeNode(){
        for (size_t i=0; i<4; ++i)
            delete children[i];
        delete[] point;
    }
public:

    friend class QuadTree<T>;
    friend std::ostream& operator<< <T>(std::ostream &, const QuadTreeNode &);
};

/// @brief Milyen mélységig nőhet a fa? Ezzel a statikus változóval lehet
    beállítani.
template <class T>
unsigned QuadTreeNode<T>::MAX_LEVEL=10;

/// @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);
}

/// @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;
    }
}

/// @brief QuadTree (generikus négy elágazású fa) osztály. A négy elágazású
    generikus fa QuadTreeNode-okból épül fel.
template <class T>
class QuadTree{
    /// Fa gyökerére mutató pointer.
    QuadTreeNode<T> *root;
public:
    /// @brief Konstruktor.
    QuadTree(double width=100, double height=100){
        root = new QuadTreeNode<T>(width, height);
    }

    /// @brief Destruktor.
    ~QuadTree(){delete root;}

    /// @return - Gyökérre mutató pointer.
    //QuadTreeNode<T>* getRootNode(){return root;}

    /// @brief Új elem beszúrása.
    /// @param data - Új pont / adat.
    void insert(Point<T> data);

    /// @brief Fa mélységének visszaadása.
    unsigned depth() const;

    /// @brief Fában lévő elemek megszámolása.
    /// @return Fa elemszáma.
    unsigned countNodes() const{
        unsigned nodes=0;
        for (iterator iter=begin(); iter!=end(); ++iter) ++nodes;
        return nodes;
    }

    /// @brief Adott pont megkeresése. Lehetőség van az adott pontban lévő
        adat megváltoztatására.
    Point<T>& find(Point<T> point);
```

```
    /// @brief Adott adatot tároló pont megkeresése. Lehetőség van az adott
        pontban lévő adat megváltoztatására.
    Point<T>& find(T data);

    /// @brief Fa iterátor.
    class iterator;
    /// @return - Legbaloldalibb levél.
    iterator begin() const{
        QuadTreeNode<T> *temp=root;
        while (!temp->isLeaf())
            temp=temp->children[0];
        return iterator(temp);
    }
    /// @return Az utolsó elem után mutat.
    iterator end() const{return iterator(NULL);}

    friend class QuadTreeNode<T>;
    friend std::ostream& operator<< <T>(std::ostream &, const QuadTree<T> &);
    /// Beolvasás. Az beolvasandó adatokat "(x;y): adat" alakban kapjuk.
    friend std::istream& operator>>(std::istream& is, QuadTree<T> & quadtree){
        double x, y;
        T data;
        while(is.good()){
            is.ignore(256, '(');
            is >> x;
            is.ignore(256, ';');
            is >> y;
            is.ignore(256, ')');
            is.ignore(256, ':');
            is >> data;
            is.ignore(256, '\n');
            quadtree.insert(Point<T>(data, x, y));
        }
        return is;
        }
};

/// @brief Fa mélységének visszaadása.
/// @return Fa mélysége.
template <class T>
unsigned QuadTree<T>::depth() const{
    if (root==NULL) return 0;
    unsigned max=0;
    for (iterator iter=begin(); iter!=end(); ++iter){
        if (max < (*iter).level)
            max=(*iter).level;
    }
    return max;
}

/// @brief Adott pont megkeresése. Lehetőség van az adott pontban lévő adat
    megváltoztatására.
/// @param point - Pont, melyet keresünk.
/// @return A keresett pont.
template <class T>
Point<T>& QuadTree<T>::find(Point<T> point){
    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];
                    }
    for (size_t i=0; i<temp->number_of_points; ++i)
        if (point==temp->point[i]) return temp->point[i];
    throw "Point not found";
}

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

template <class T>
/// @brief Új elem beszúrása.
/// @param point - Új pont / adat.
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);
}

/// @brief Iterátor.
template <class T>
class QuadTree<T>::iterator{
protected:
    QuadTreeNode<T> *node;
public:
```

```
    /// @brief Konstruktor.
    iterator(QuadTreeNode<T> *node=NULL): node(node){}

    QuadTreeNode<T>& operator*(){return *node;}
    QuadTreeNode<T>* operator->(){return node;}

    /// @brief Prefix operator++
    /// @return Következő elem.
    iterator& operator++();

    /// @brief Postfix operator++
    iterator operator++(int){
        iterator temp(*this);
        ++(*this);
        return temp;
    }

    /// @brief Egyenlőség operátor.
    /// @return Igaz, ha megegyezik az csomópont.
    bool operator==(const iterator& iter){
        return node==iter.node;
    }

    /// @return Igaz, ha nem egyezik meg a csomópont.
    bool operator!=(const iterator& iter){
        return node!=iter.node;
    }
};

/// @brief Prefix inkrement.
/// @return Következő elem.
template <class T>
typename QuadTree<T>::iterator& QuadTree<T>::iterator::operator++(){
    size_t i=0;
    /// Ha a gyökér elemnél vagyunk (csak annak nincs szülője), akkor
       végigértünk az összes elemen.
    if (node->parent==NULL){
        node=node->parent;
        return *this;
    }
    /// A szülő hányadig gyerekén állunk?
    while (node!=node->parent->children[i])
        ++i;
    /// Ha a következő gyerek levél, akkor ez lesz a következő elem.
    if (i<3 && node->parent->children[i+1]->isLeaf())
        node=node->parent->children[i+1];
    /// Ha a következő gyerek nem levél, akkor a következő elem a
       legbaloldalibb levél a következő gyerek alatt.
    else if (i<3){
        QuadTreeNode<T> *temp=node->parent->children[i+1];
        while(!temp->isLeaf())
            temp=temp->children[0];
        node=temp;
    }
    /// Egyébként a szülő a következő elem.
    else
        node=node->parent;
    return *this;
}
```

```
/// @brief Point (pont) kiírása "(x;y): adat" alakban.
template <class T>
std::ostream& operator<<(std::ostream & os, const Point<T> & point){
    return os << '(' << point.x << ';' << point.y << ')' << ':' << ' ' <<
        point.data << std::endl;
}

/// @brief Fa kiírása iterátor használatával.
template <class T>
std::ostream& operator<<(std::ostream & os, const QuadTree<T> & quadtree){
    typename QuadTree<T>::iterator iter=quadtree.begin();
    while (iter!=quadtree.end()){
        os << *iter;
        ++iter;
    }
    return os;
}

/// @brief Csomópont kiírása.
template <class T>
std::ostream& operator<<(std::ostream & os, const QuadTreeNode<T> & node){
    /// Ha van adat, akkor ki kell írni.
    if (node.hasData()){
        for (size_t i=0; i<node.number_of_points; ++i){
            for (unsigned i=0; i<node.level; ++i) os << ' ';
            os << node.point[i];
        }
    }
    return os;
}
```