

Álmos Totálisan Jóarc Ütemezője

1.0

Készítette Doxygen 1.9.1

1. Névtérmutató	1
1.1. Névtérlista	1
2. Osztálymutató	3
2.1. Osztálylista	3
3. Névterek dokumentációja	5
3.1. almtree névtér-referencia	5
3.1.1. Részletes leírás	5
4. Osztályok dokumentációja	7
4.1. Drawer osztályreferencia	7
4.1.1. Részletes leírás	8
4.1.2. Tagfüggvények dokumentációja	8
4.1.2.1. draw()	8
4.1.2.2. drawRunnable()	8
4.1.2.3. treeCToGraphicsC()	8
4.2. Message struktúrareferencia	8
4.2.1. Részletes leírás	9
4.3. Node struktúrareferencia	9
4.3.1. Részletes leírás	9
4.3.2. Tagfüggvények dokumentációja	9
4.3.2.1. isOnLeft()	10
4.3.2.2. sibling()	10
4.4. RbTree osztályreferencia	10
4.4.1. Részletes leírás	11
4.4.2. Tagfüggvények dokumentációja	11
4.4.2.1. fixDoubleBlack()	11
4.4.2.2. fixTreeAfterInsert()	12
4.4.2.3. getLevelOrder()	12
4.4.2.4. getRoot()	12
4.4.2.5. insert()	13
4.4.2.6. popMin()	13
4.4.2.7. rbDelete()	13
4.4.2.8. rbInsert()	13
4.4.2.9. replacementSearch()	14
4.4.2.10. rotateLeft()	14
4.4.2.11. rotateRight()	15
4.4.2.12. search()	15
4.4.2.13. successor()	15
4.5. Server osztályreferencia	16
4.5.1. Részletes leírás	16
4.6. Task osztályreferencia	17

4.6.1. Részletes leírás	17
Tárgymutató	19

1. fejezet

Névtérmutató

1.1. Névtérlista

Az összes dokumentált névtér listája rövid leírásokkal:

[almtree](#)

A piros-fekete fa színeit tartalmazó névtér [5](#)

2. fejezet

Osztálymutató

2.1. Osztálylista

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

Drawer	Az információk megjelenítésért felelős osztály	7
Message	A szerverhez intézett üzenet	8
Node	A csomópont a piros fekete fában	9
RbTree	Nagyjából standard és általános piros-fekete fa implementáció	10
Server	A folyamatokat kezelő, vizualizációt feladatfuttatással összekötő szerver	16
Task	Egy Unix process megtestesítője	17

3. fejezet

Névterek dokumentációja

3.1. almtree névtér-referencia

A piros-fekete fa színeit tartalmazó névtér.

Enumerációk

- enum **COLOR** { **RED** , **BLACK** }

3.1.1. Részletes leírás

A piros-fekete fa színeit tartalmazó névtér.

4. fejezet

Osztályok dokumentációja

4.1. Drawer osztályreferencia

Az információk megjelenítésért felelős osztály.

```
#include <drawer.hpp>
```

Publikus tagfüggvények

- **Drawer** ([RbTree](#) *tree, std::mutex *mutex)
- void [init](#) ()
Inicializálja a rajzolási felületet.
- void [drawRunnable](#) ()
A rajzolást vezérlő runnable függvény.
- void [disableDraw](#) ()
Megállítja a rajzolási folyamatot.
- void [callReDraw](#) ()
Újra rajzolja a piros-fekete fa állapotát.
- void [setRunningTask](#) (std::shared_ptr< [Task](#) > [currentTask](#))
Beállítja az éppen futó folyamatot.

Privát tagfüggvények

- void [draw](#) ()
A rajzolómetódus.
- int [treeCToGraphicsC](#) (almtree::COLOR color)
Átkonvertálja az almtree névtérben levő színt a graphics.h színére.

Privát attribútumok

- [RbTree](#) * **tree**
- std::mutex * **mutex**
- std::shared_ptr< [Task](#) > [currentTask](#)
Az éppen háttérben futó folyamat.
- int **gd**
- int **gm**
- int **window**
- bool **shouldDraw**
- bool **reDraw**

4.1.1. Részletes leírás

Az információk megjelenítésért felelős osztály.

4.1.2. Tagfüggvények dokumentációja

4.1.2.1. draw()

```
void Drawer::draw ( ) [private]
```

A rajzolómetódus.

Lerajzolja a piros-fekete aktuális állapotát jelezve a csomópontokban az eltelt időt amennyit futott a jelzett folyamat, és a folyamat PID-jét. A képernyő alján ugyanakkor kiírja, hogy éppen melyik folyamat fut.

4.1.2.2. drawRunnable()

```
void Drawer::drawRunnable ( )
```

A rajzolást vezérlő runnable függvény.

Addig fut, amíg a shouldDraw változó értéke true. Akkor rajzoltat, ha a reDraw értéke true.

4.1.2.3. treeCToGraphicsC()

```
int Drawer::treeCToGraphicsC (
    almtree::COLOR color ) [private]
```

Átkonvertálja az almtree névtérben levő színt a graphics.h színére.

Paraméterek

<i>color</i>	Egy szín az almtree névtérből (vagy piros vagy fekete).
--------------	---

Visszatérési érték

A graphics.h COLOR enumjának egy tagja (vagy piros vagy fekete).

Ez a dokumentáció az osztályról a következő fájl alapján készült:

- headers/drawer.hpp

4.2. Message struktúrareferencia

A szerverhez intézett üzenet.

```
#include <message.h>
```

Publikus attribútumok

- MSG_TYPE **type**
- int **pTime**
- char **programName** [50]

4.2.1. Részletes leírás

A szerverhez intézett üzenet.

Ez a dokumentáció a struktúráról a következő fájl alapján készült:

- headers/message.h

4.3. Node struktúrareferencia

A csomópont a piros fekete fában.

```
#include <node.hpp>
```

Publikus tagfüggvények

- **Node** (int key)
- **Node** (const **Node** &cpy)
- **Node * sibling** ()
Visszatéríti a csomópont testvérét.
- bool **isOnLeft** ()
Ellenőrzi, hogy a csomópont a szülője bal oldalán van-e.

Publikus attribútumok

- int **key**
- std::shared_ptr< void > **data**
- almtree::COLOR **color**
- **Node * left**
- **Node * right**
- **Node * parent**

4.3.1. Részletes leírás

A csomópont a piros fekete fában.

4.3.2. Tagfüggvények dokumentációja

4.3.2.1. isOnLeft()

```
bool Node::isOnLeft ( )
```

Ellenőrzi, hogy a csomópont a szülője bal oldalán van-e.

Visszatérési érték

true ha a bal oldalon van, másképp false

4.3.2.2. sibling()

```
Node* Node::sibling ( )
```

Visszatéríti a csomópont testvérét.

Visszatérési érték

A csomópont testvére.

Ez a dokumentáció a struktúráról a következő fájl alapján készült:

- headers/node.hpp

4.4. RbTree osztályreferencia

Nagyjából standard és általános piros-fekete fa implementáció.

```
#include <rbtree.hpp>
```

Publikus tagfüggvények

- **Node * rbInsert** (const int &key)
Behelyez egy új csomópontot a piros-fekete fába, és frissíti a minimális csomópontot.
- void **rbDelete** (Node *node)
Kivesz egy csomópontot a piros-fekete fából.
- std::vector< Node * > **getLevelOrder** ()
Visszatéríti a fa csomópontjait szint szerinti sorrendben.
- **Node * getRoot** ()
Visszatéríti a piros-fekete fa gyökerét.
- **Node * search** (const int &key)
Visszatéríti a fa "key" kulcsú csomópontját, ha létezik.
- std::shared_ptr< void > **popMin** ()
Kitörli a minimális csomópontot, és visszatéríti a benne levő adatot.

Privát tagfüggvények

- `Node * insert (Node *root, Node *newNode)`
Privát rekurzív bináris keresőfa beszúrás.
- `void fixTreeAfterInsert (Node *&root, Node *newNode)`
Kijavítja beszúrás után a piros-fekete tulajdonságot.
- `void rotateLeft (Node *&root, Node *&nodePtr)`
Végrehajt egy bal forgatást a megadott csomóponton.
- `void rotateRight (Node *&root, Node *&nodePtr)`
Végrehajt egy jobb forgatást a megadott csomóponton.
- `Node * replacementSearch (Node *x)`
Megkeresi az x csomópont utódját BST törlés esetén.
- `Node * successor (Node *x)`
Visszatéríti az x csomópont által jelzett al-fa legkisebb elemét.
- `void fixDoubleBlack (Node *x)`
Kijavítja a megadott x csúcs "duplafeketeségét".

Privát attribútumok

- `Node * root`
- `Node * minNode`

4.4.1. Részletes leírás

Nagyjából standard és általános piros-fekete fa implementáció.

4.4.2. Tagfüggvények dokumentációja

4.4.2.1. fixDoubleBlack()

```
void RbTree::fixDoubleBlack (
    Node * x ) [private]
```

Kijavítja a megadott x csúcs "duplafeketeségét".

A dupla-fekete javításnál főleg a testvér színét figyeljük. a) Ha a testvér fekete, és van legalább egy piros gyermeke, akkor maximum két forgatással ki lehet javítani a fekete magasságot. b) Ha a testvér fekete és mindkét gyermeke fekete, akkor átszínezzük a testvért pirosra, a szülőt feketére, s ha a szülő duplafekete lesz, meghívjuk rá is a függvényt. c) Ha a testvér piros, akkor átszínezzük feketére, a szülőt pirosra, forgatunk egyet a szülőn attól függően, hogy hol van a testvér, és visszatérünk az a) vagy b) esetre.

Paraméterek

x	A csúcs amit "megjavítunk".
---	-----------------------------

4.4.2.2. fixTreeAfterInsert()

```
void RbTree::fixTreeAfterInsert (
    Node *& root,
    Node * newNode ) [private]
```

Kijavítja beszúrás után a piros-fekete tulajdonságot.

A kijavítási folyamat tartalmazhat forgatásokat és átszínezéseket, és rotációkat. Mivel minden beszúrt új csomópont piros, ezért leredukálható a probléma az egymást követő piros eset kijavítására. Ha a nagybácsi (a newNode szülőjének testvére) piros, akkor egyszerűen a szülő s a nagybácsi színét feketére állítjuk és a nagyszülőjét pedig pirosra. Fekete esetben 4 eset fordulhat elő:

1. Bal-bal(a szülő a nagyszülő bal gyermeke és a newNode a szülő bal gyermeke): Végrehajtunk egy jobb forgatást a nagyszülőn, kicseréljük a szülő és nagyszülő színét.
2. Bal-jobb: Végrehajtunk egy bal forgatást a szülőn, ezután visszatértünk a bal-bal esethez.
3. Jobb-jobb: Végrehajtunk egy bal forgatást a nagyszülőn, kicseréljük a szülő és a nagyszülő színét.
4. Jobb-bal: Végrehajtunk egy jobb forgatást a szülőn, ezután visszatértünk a jobb-jobb esethez. A fenti eseteket addig folytatjuk, amíg a newNode paraméter fekete nem lesz, vagy pedig a szülő fekete lesz.

Paraméterek

<i>root</i>	A fa gyökere
<i>newNode</i>	Az új beszúrt csomópont.

4.4.2.3. getLevelOrder()

```
std::vector<Node*> RbTree::getLevelOrder ( )
```

Visszatéríti a fa csomópontjait szint szerinti sorrendben.

A getLevelOrder függvény egy várakozási sor segítségével bejárja a piros-fekete fát, szintről-szintre haladva.

Visszatérési érték

A fa csomópontjait tartalmazó vektor.

4.4.2.4. getRoot()

```
Node* RbTree::getRoot ( )
```

Visszatéríti a piros-fekete fa gyökerét.

Visszatérési érték

A fa gyökerét tartalmazó csomópont.

4.4.2.5. insert()

```
Node* RbTree::insert (
    Node * root,
    Node * newNode ) [private]
```

Privát rekurzív bináris keresőfa beszúrás.

A szülőt azért térítjük vissza, hogy meg tudjuk hívni a piros-fekete tulajdonság kijavítására szolgáló fixTreeAfterInsert függvényt.

Visszatérési érték

Visszatéríti az újonnan beszúrt csomópont szülőjét.

4.4.2.6. popMin()

```
std::shared_ptr<void> RbTree::popMin ( )
```

Kitörli a minimális csomópontot, és visszatéríti a benne levő adatot.

Visszatérési érték

A csomópontban levő adatra mutató megosztott mutató.

4.4.2.7. rbDelete()

```
void RbTree::rbDelete (
    Node * node )
```

Kivesz egy csomópontot a piros-fekete fából.

Először kitörli egy standard BST törléssel, utána pedig, ha fennáll az eset, kijavítja a dupla fekete hibát a fixDoubleBlack függvényel. Hívjuk v-nek a törölt csúcsot és u-nak a csúcsot, amivel lecseréljük ezt a törölt csúcsot. A legegyszerűbb eset akkor áll fenn, ha vagy u vagy v piros, ekkor egyszerűen az u csúcsot átszínezzük pirosra. Ha mindkettő csúcs fekete, akkor törlés után (vagy előtt, ha a leváltó nullptr) meghívjuk a fixDoubleBlack függvényt.

Paraméterek

<i>node</i>	A törlendő csomópontra mutató pointer.
-------------	--

4.4.2.8. rbInsert()

```
Node* RbTree::rbInsert (
```

```
const int & key )
```

Behelyez egy új csomópontot a piros-fekete fába, és frissíti a minimális csomópontot.

Az `rbInsert` függvény először létrehoz egy új csomópontot, majd végrehajt egy standard bináris-keresőfa beszúrását az `insert` módszerrel. utána pedig (ha szükséges) kijavítja az esetleges piros-fekete tulajdonság megsértését a `fixTreeAfterInsert` privát módszerrel.

Paraméterek

<code>key</code>	A kulcs, ami alapján el lesz rendezve a csomópont a keresőfában.
------------------	--

Visszatérési érték

Az újonnan létrehozott csomópont.

4.4.2.9. `replacementSearch()`

```
Node* RbTree::replacementSearch (
    Node * x ) [private]
```

Megkeresi az `x` csomópont utódját BST törlés esetén.

Paraméterek

<code>x</code>	A csomópont aminek a "leváltóját" keressük.
----------------	---

Visszatérési érték

Az `x` csomópont "leváltója".

4.4.2.10. `rotateLeft()`

```
void RbTree::rotateLeft (
    Node *& root,
    Node *& nodePtr ) [private]
```

Végrehajt egy bal forgatást a megadott csomóponton.

Paraméterek

<code>root</code>	A fa gyökere.
<code>nodePtr</code>	A "megforgatott" csomópont.

4.4.2.11. rotateRight()

```
void RbTree::rotateRight (
    Node *& root,
    Node *& nodePtr ) [private]
```

Végrehajt egy jobb forgatást a megadott csomóponton.

Paraméterek

<i>root</i>	A fa gyökere.
<i>nodePtr</i>	A "megforgatott" csomópont.

4.4.2.12. search()

```
Node* RbTree::search (
    const int & key )
```

Visszatéríti a fa "key" kulcsú csomópontját, ha létezik.

Paraméterek

<i>key</i>	A keresett csomópont kulcsa.
------------	------------------------------

Visszatérési érték

A kulcsot tartalmazó csomópont, másképp egy nullptr.

4.4.2.13. successor()

```
Node* RbTree::successor (
    Node * x ) [private]
```

Visszatéríti az x csomópont által jelzett al-fa legkisebb elemét.

Paraméterek

<i>x</i>	Az x csomópont.
----------	-----------------

Visszatérési érték

Az al-fa legbaloldalibb vagy legkisebb nem-nullptr eleme.

Ez a dokumentáció az osztályról a következő fájl alapján készült:

- headers/rbtree.hpp

4.5. Server osztályreferencia

A folyamatokat kezelő, vizualizációt feladatfuttatással összekötő szerver.

```
#include <server.hpp>
```

Publikus tagfüggvények

- **Server** ([RbTree](#) *tree, std::mutex *mutex, [Drawer](#) *drawer)
- void **listenRunnable** ()
- void **runTaskRunnable** ()

Privát tagfüggvények

- void **insertTask** (const std::shared_ptr< [Task](#) > &task)
- void **removeTask** (int pid)
- void **runTask** ()
- void **updateOwners** ()

Privát attribútumok

- int **pidCount**
- std::mutex * **mutex**
- [Message](#) **message**
- io_context **context**
- udp::endpoint **receiver**
- udp::socket **serverSocket**
- mutable_buffer **buffer**
- err_code **ec**
- [RbTree](#) * **tree**
- [Drawer](#) * **drawer**
- bool **shouldRun**
- std::unordered_map< int, std::shared_ptr< [Task](#) > > **tasks**

4.5.1. Részletes leírás

A folyamatokat kezelő, vizualizációt feladatfuttatással összekötő szerver.

Ez a dokumentáció az osztályról a következő fájl alapján készült:

- headers/server.hpp

4.6. Task osztályreferencia

Egy Unix process megtestesítője.

```
#include <task.hpp>
```

Publikus tagfüggvények

- **Task** (int **pid**, std::string &name, int pTime, int vruntime)
- std::string **getName** ()
- **Node** * **getNode** ()
- int **getPid** ()
- int **getVRuntime** ()
- void **setNode** (**Node** *treeNode)
- void **setXCoord** (int **x_coord**)
- void **setYCoord** (int **y_coord**)
- void **setVRunTime** (int vruntime)
- void **setLevel** (int level)
- int **getXCoord** ()
- int **getYCoord** ()
- void **initLevel** ()
- int **getLevel** ()
- int **getPTime** ()

Privát attribútumok

- int **pid**
A feladat folyamat azonosítója.
- int **x_coord**
A feladat koordinátái a megjelenített fában.
- int **y_coord**
- std::string **name**
- **Node** * **treeNode**
A feladat csomópontja a piros-fekete fában;.
- int **level**
- int **vruntime**
- int **pTime**

4.6.1. Részletes leírás

Egy Unix process megtestesítője.

Ez a dokumentáció az osztályról a következő fájl alapján készült:

- headers/task.hpp

Tárgymutató

- almtree, [5](#)
- draw
 - Drawer, [8](#)
- Drawer, [7](#)
 - draw, [8](#)
 - drawRunnable, [8](#)
 - treeCToGraphicsC, [8](#)
- drawRunnable
 - Drawer, [8](#)
- fixDoubleBlack
 - RbTree, [11](#)
- fixTreeAfterInsert
 - RbTree, [12](#)
- getLevelOrder
 - RbTree, [12](#)
- getRoot
 - RbTree, [12](#)
- insert
 - RbTree, [12](#)
- isOnLeft
 - Node, [9](#)
- Message, [8](#)
- Node, [9](#)
 - isOnLeft, [9](#)
 - sibling, [10](#)
- popMin
 - RbTree, [13](#)
- rbDelete
 - RbTree, [13](#)
- rbInsert
 - RbTree, [13](#)
- RbTree, [10](#)
 - fixDoubleBlack, [11](#)
 - fixTreeAfterInsert, [12](#)
 - getLevelOrder, [12](#)
 - getRoot, [12](#)
 - insert, [12](#)
 - popMin, [13](#)
 - rbDelete, [13](#)
 - rbInsert, [13](#)
 - replacementSearch, [14](#)
 - rotateLeft, [14](#)
 - rotateRight, [15](#)
 - search, [15](#)
 - successor, [15](#)
- replacementSearch
 - RbTree, [14](#)
- rotateLeft
 - RbTree, [14](#)
- rotateRight
 - RbTree, [15](#)
- search
 - RbTree, [15](#)
- Server, [16](#)
- sibling
 - Node, [10](#)
- successor
 - RbTree, [15](#)
- Task, [17](#)
- treeCToGraphicsC
 - Drawer, [8](#)