**Data structure: Assignment 3**

**결과 보고서**

**1 레드 블랙 트리 (Red-Black Tree)**

**1.1 2-3트리와 동등한 레드 블랙 트리**

**1.1.1 2-3트리와 동등한 레드 블랙 트리: 삽입 및 삭제 알고리즘**

#알고리즘

-삽입

Insert (a, r){

if ( r consists of a single leaf l labeled b ) //that is, if l is the root

create a new root r' //interior node

create a new leaf v labeled a

make l and v children of r' in proper order

update L and M for r'

else Set f to SEARCH(a, r)

create a new leaf l labeled a

if f has 2 children

insert l into proper position

update L and M

else

create a transitory 4-node tree at f (l in proper position)

ADDCHILD (f)

}

SEARCH(a, r){

if ( children of r are leaves )

return r

else if ( a <= k1[r] ) return Search(a, r->left)

else if ( a <= k2[r] ) return Search(a, r->middle)

else return Search(a, r->right)

}

ADDCHILD (v){

create new interior node v'

move 2 rightmost children of v to v'

if ( v has no parent )//that is, if v is the root

make new root r'

make v the left child and v' the right child

update L and M

else

let f be the parent of v

make v' the child of f immediately to the right of v

if f now has 4 children

ADDCHILD(f)

else

update L and M

}

-삭제

Delete (){

Let f be the parent of the node just deleted (or NULL if leaf was root)

while( f is an illegal interior node ){ //not NULL or leaf, with one child

if ( f has no parent )

make the single child of f the new root

delete f

Set f to the root //for while loop condition

else let g be parent of f //known to have 2 or 3 children

if ( one of f's siblings is a 3-node )

move one child from the 3-node into f //That is, f becomes legal

update k1 and k2 everywhere else give f's remaining child to one of f's siblings

delete f

update k1 and k2 everywhere

Set f to g //for while loop condition

}//end-while

}

**1.1.2 2-3트리와 동등한 레드 블랙 트리: 삽입 및 삭제 알고리즘 구현**

**#구현**

1) void insert(int Key)

struct treeRecord안에 constructor을 만들어 this->key=key, this->color=RED,

LChild=MChild=RChild=NULL으로 작성하여 이의 형태를 나타내는 Nptr 변수 하나를 지정한다. 그 후 add함수를 이용하고 이를 fix하는 함수를 호출한다.

2) Nptr add(Nptr root, Nptr b)

root가 NULL이면 b를 return하고 root의 key가 b의 key보다 클 때 root의 LChild가 add(root->LChild,b)로 설정한다. 그리고 root의 LChild의 MChild가 root가 된다.

그 root의 key가 b의 key보다 작을 경우 RChild에 add를 하며 root의 RChild의 MChild가 root가 된다. 그 외의 경우에는 root를 리턴한다.

3) void RBPrettyPrint(const std::string& prefix, Nptr node, bool isLeft)

DS\_assignment\_3 본문에 주어진 함수를 그대로 사용하였다.

**1.1.3 2-3트리와 동등한 레드 블랙 트리: 삽입 및 삭제 알고리즘 테스트**

**#구현**

1) void RBVerify()

root를 verify\_property들을 호출하여 2-3트리와 동일한 레드블랙트리인지를 검사한다.

2) void verify\_property\_()

1.1에 주어진 특성들을 나타내는 함수이다.

**#실행 결과**

2-3 트리를 레드 블랙 트리에 적용하는 것이 어려워 성공하지 못하였다.

verify\_property들을 나타내는 것을 어떻게 해야 할지 몰랐다.

**1.2 2-3-4트리와 동등한 레드 블랙 트리**

**1.2.1 2-3트리와 동등한 레드 블랙 트리: 삽입 및 삭제 알고리즘**

**#알고리즘**

-삽입

insert(l)

{

Node node = root;

DataItem dataitem = new DataItem(l);

do

{

for(; node.isFull(); // if node full

node = getNextChild(node, l))

{

split(node); // split it

node = node.getParent(); //back up

}

if(!node.isLeaf()) // if node is leaf,

{

//node is not full,not a leaf; so go to lower level

node = getNextChild(node, l);

} else

{

node.insertItem(dataitem); // insert new DataItem

return;

}

} while(true);

} // end insert()

-삭제

DELETE (*T, z*)

**if** left[z] = nil[T] or right[z] = nil[T]

**then** y http://staff.ustc.edu.cn/~csli/graduate/algorithms/images/arrlt12.gif z

**else** y http://staff.ustc.edu.cn/~csli/graduate/algorithms/images/arrlt12.gif TREE-SUCCESSOR(z)

**if** left[y] http://staff.ustc.edu.cn/~csli/graduate/algorithms/images/noteq.gif nil[T]

**then** x http://staff.ustc.edu.cn/~csli/graduate/algorithms/images/arrlt12.gif left[y]

**else** x http://staff.ustc.edu.cn/~csli/graduate/algorithms/images/arrlt12.gif right[y]

p[x] http://staff.ustc.edu.cn/~csli/graduate/algorithms/images/arrlt12.gif p[y]

**if** p[y] = nil[T]

**then** root[T] http://staff.ustc.edu.cn/~csli/graduate/algorithms/images/arrlt12.gif x

**else if** *y = left*[*p*[*y*]]

**then** *left*[*p*[*y*]]http://staff.ustc.edu.cn/~csli/graduate/algorithms/images/arrlt12.gif x

**else** *right*[*p*[*y*]] http://staff.ustc.edu.cn/~csli/graduate/algorithms/images/arrlt12.gif *x*

**if** *y* http://staff.ustc.edu.cn/~csli/graduate/algorithms/images/noteq.gif z

**then** *key*[*z*] http://staff.ustc.edu.cn/~csli/graduate/algorithms/images/arrlt12.gif *key*[*y*]

http://staff.ustc.edu.cn/~csli/graduate/algorithms/book6/273_a.gif If *y* has other fields, copy them, too.

**if** *color*[*y*] = BLACK

**then** RB-DELETE-FIXUP (*T,x*)

**return** *y*

**1.2.2 2-3-4트리와 동등한 레드 블랙 트리: 삽입 및 삭제 알고리즘 구현**

**#구현방법**

1) void insert(int Key)

struct treeRecord안에 constructor을 만들어 this->key=key, this->color=RED,

LChild=MChild=RChild=NULL으로 작성하여 이의 형태를 나타내는 Nptr 변수 하나를 지정한다. 그 후 add함수를 이용하고 이를 fix하는 함수를 호출한다.

2) Nptr add(Nptr root, Nptr b)

root가 NULL이면 b를 return하고 root의 key가 b의 key보다 클 때 root의 LChild가 add(root->LChild,b)로 설정한다. 그리고 root의 LChild의 MChild가 root가 된다.

그 root의 key가 b의 key보다 작을 경우 RChild에 add를 하며 root의 RChild의 MChild가 root가 된다. 그 외의 경우에는 root를 리턴한다.

3) void RBPrettyPrint(const std::string& prefix, Nptr node, bool isLeft)

DS\_assignment\_3 본문에 주어진 함수를 그대로 사용하였다.

4) void rotateLeft(Nptr &root, Nptr &b)

트리가 왼쪽으로 rotate할 경우를 작성하였다. Nptr변수를 설정하여 이를 b의 Rchild로 설정하고 b의 RChild를 새로 설정한 변수의 LChild로 설정한다. b의 RChild가 NULL 아닐 경우, MChild가 NULL일 경우로 나누어서 작성하였다.

4) void rotateRight(Nptr &root, Nptr &b)

트리가 오른쪽으로 rotate할 경우를 작성하였다. Nptr 변수를 설정하여 이를 b의 LChild로 설정하고 b의 LChild를 새로 설정한 변수의 RChild로 설정한다. 또한 이를 b의 LChild가 NULL이 아닐 경우, MChild가 NULL일 경우로 나누어서 작성하였다.

5) void fixViolation( Nptr &root, Nptr &b)

트리에 insert된 후에 위반하는 경우가 일어날 시에 이를 고쳐주는 함수이다.

1) b의 부모가 b의 grandparent의 LChild일 경우

-b의 삼촌 노드가 RED이며 색깔 바꾸는 경우가 필요할 때

-b의 RChild의 부모가 left rotation이 필요할 경우

-b의 LChild의 부모가 right rotation이 필요할 경우

2) b의 부모가 b의 grandparent의 RChild일 경우

-b의 삼촌 노드가 RED이며 색깔 바꾸는 경우가 필요할 때

-b의 RChild의 부모가 left rotation이 필요할 경우

-b의 LChild의 부모가 right rotation이 필요할 경우

그리고 마지막에 root의 color를 BLACK으로 설정한다.

**1.2.3 2-3-4트리와 동등한 레드 블랙 트리: 삽입 및 삭제 알고리즘 테스트**

**#구현**

1) void RBVerify()

root를 verify\_property들을 호출하여 2-3트리와 동일한 레드블랙트리인지를 검사한다.

2) void verify\_property\_()

1.1에 주어진 특성들을 나타내는 함수이다.

**#실행 결과**

delete함수를 구현하지 못하였고, 또한 verify도 제대로 실행되지 않았다.

![텍스트이(가) 표시된 사진

자동 생성된 설명]()

**2 트라이 (Trie)**

**2.1 트라이 클래스 구현**

**2.1.1 트라이를 이용한 Word count계산 및 사용자 테스트**

**#구현방법**

1) void Create()

Nptr 변수를 설정하여 이를 CreateTrieNode()로 설정한다.

2) int Insert(const char \*key, int val)

const char \*key를 string으로 변환하여 NUM\_ALPHABET만큼 for문을 돌린다.

Index를 s[i]-‘a’로 설정하여 이에 관해 삽입을 한다.

3) Nptr Search(const char \*key)

Insert와 비슷한 방식으로 하여 해당되는 key가 존재하면 노드를 리턴하고 그 외의 경우는 NULL을 리턴한다.

4) void Delete(const char \*key)

주어진 key와 동일한 key가 존재하면 그것을 삭제하고 없으면 없다는 메시지를 내보낸다.

5) void Save(const char \*filename)

해당되는 filename에 내용을 저장하도록 작성하였다.

6) void Open(const char \*filename)

해당되는 filename의 파일이 open되었을 경우 그 file에 있는 내용을 vector로 받아 이를 다시 string으로 바꾼다.

**#실행 결과**

2차원 배열에 어떻게 저장할 지 몰라서 제대로 구현하지 못하였다.

**2.1.2 트라이와 해시 및 이진탐색트리 효율 비교**

소요시간 관련해서 DS\_assignment\_2와 같은 이유로 test를 하지 못하였다.

**3 네비게이터**

**3.1 네비게이터 구현 (java로 작성)**

**#구현 방법**

-GUI

Container, JFrame, JButton을 add하여 GUI를 구현하였다.

**3.2 네비게이터 테스트**

**#실행결과**

시간이 부족하여 제대로 하지 못하였다.