1.1 2-3트리와 동등한 레드 블랙 트리 (미완성)

1.1.1 삽입 및 삭제 알고리즘

삽입 알고리즘

1. key를 삽입할 위치를 찾는다. 알맞은 위치는 이진 탐색 트리에서의 위치와 같다.
2. 트리가 비어있으면 루트 노드에 key를 대입하고 삽입을 끝낸다.
3. 트리에 값이 존재하면 알맞은 위치에 key를 대입한다. 이후 2가지 경우에 맞춰 트리를 조정한다. 대입한 key가 있는 노드는 t라고 한다.
    1. t의 부모의 색깔이 BLACK인 경우
        1. t의 부모의 왼쪽 노드가 없고 오른쪽 노드가 존재하는 경우
           부모의 오른쪽 노드의 색을 BLACK으로 설정하고 부모의 색을 RED로 설정한 후 부모에 대해 Left Rotate를 실행한다
        2. t의 부모의 왼쪽 노드가 존재하고 그 노드가 빨강인 경우
           t의 부모의 오른쪽 노드가 존재하면 오른쪽 노드의 색을 BLACK으로 설정하고 오른쪽 노드가 없으면 왼쪽 노드의 색을 RED로 설정한다.
           t의 조부모가 존재하는 경우
            1. t의 부모가 t의 조부모의 오른쪽 노드이고 조부모의 왼쪽 노드의 색이 BLACK이고 t는 부모의 오른쪽 노드일 경우
               t의 조부모의 색을 RED로 설정하고 t의 조부모에 대해 Left Rotate를 실행한다.
            2. 그 외
               t의 부모가 t의 조부모의 오른쪽 노드일 경우 t의 조부모의 왼쪽 노드의 색을 BLACK으로 설정한다.
        3. t의 부모의 오른쪽 노드가 없고 왼쪽 노드가 존재하는 경우
           t의 부모의 왼쪽 노드의 색을 BLACK으로 설정하고 부모의 색을 RED로 설정한 후 부모에 대해 Right Rotate를 실행한다.
        4. 그 외
           t의 모든 자식의 색을 BLACK으로 설정한다.
    2. 그 외
        1. t가 t의 부모의 오른쪽 노드이고 t의 부모가 t의 조부모의 왼쪽 노드일 경우
           t의 부모의 색을 BLACK으로 설정하고 t의 부모에 대해 Left Right Rotate를 실행한다.
        2. t가 t의 부모의 왼쪽 노드이고 t의 부모가 t의 조부모의 오른쪽 노드일 경우
           t의 부모의 색을 BLACK으로 설정하고 t의 부모에 대해 Right Left Rotate를 실행한다.
        3. t의 조부모가 존재할 경우
            1. t가 부모의 왼쪽 노드일 경우
               t의 색을 BLACK으로 설정

1. t의 조부모의 부모가 존재하고 t의 조부모가 조부모의 부모의 오른쪽 노드일 경우
   t의 조부모의 부모의 색을 RED로 설정하고 t의 조부모에 대해 Right Left Rotate실행
2. 그 외
   t의 조부모에 대해 Right Rotate 실행
   만약 t의 부모의 색이 RED이고 t의 조부모의 색이 RED이면
   t의 부모의 색을 BLACK으로 설정하고 t의 조부모의 부모에 대해 Right Rotate 실행

2. 그 외
   t의 조부모에 대해 Left Rotate 실행

1.1.2 삽입 및 삭제 알고리즘 구현

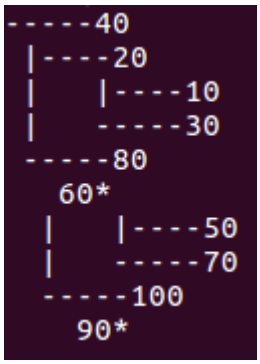
기타 사항

treeRecord에 struct treeRecord * p; 를 추가하였다.

1.1.3 삽입 및 삭제 알고리즘 테스트

RBVerify – 미구현

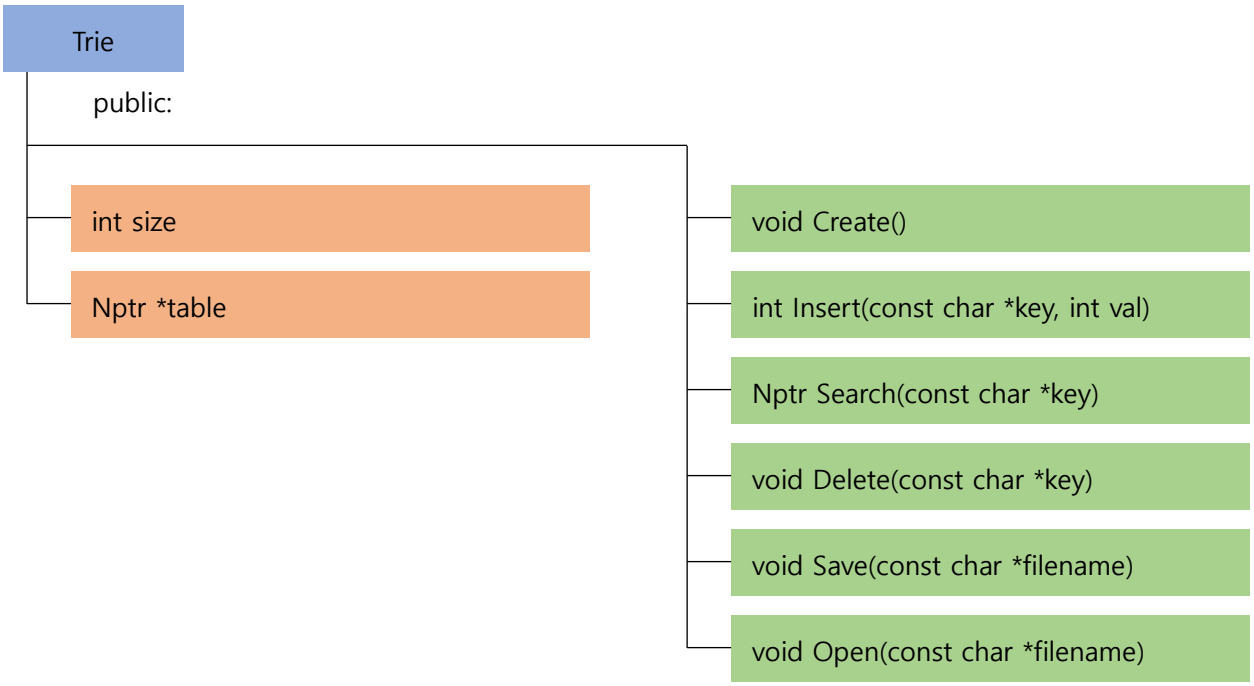
10,20,30,40,50,60,70,80,90,100



2.1 트라이 클래스 구현

코드 구현

클래스 계층도



void Create()

root = CreateTrieNode()를 실행하여 root 를 노드로 만든 후, 값으로 0 을 대입하고 size 를 0 으로 설정한다.

int Insert(const char *key, int val)

Search(key)를 실행하여 같은 key 가 있는지 확인한다.

같은 key 가 있는 경우, 해당 key 의 val 을 인자 val 로 설정한다.

같은 key 가 없는 경우, key 를 한 글자씩 읽어서 해당 글자를 인덱스로 하여 노드를 탐색한다.

빈 노드를 만났을 때는 CreateTrieNode()를 실행하여 노드를 만든 후, val 에 0 을 넣은 뒤 탐색을 계속한다.

탐색이 끝나면 해당 노드의 val 에 인자 val 을 대입하고 size 를 하나 키운다.

Nptr Search(const char *key)

key 를 한 글자씩 읽어서 해당 글자를 인덱스로 하여 노드를 탐색한다.

빈 노드를 만나면 NULL 을 반환한다.

탐색이 끝나면 해당 노드를 반환한다.

void Save(const char *filename)

재귀적으로 저장하는 함수 RecursiveSave(ofstream& fout, Nptr T)를 정의한다.

void RecursiveSave(ofstream& fout, Nptr T)

T->nodes 를 순서대로 읽어서 T->nodes[i](i 는 0 부터 NUM_ALPHABET)가 비어있지 않을 경우 i 와 T->nodes[i]->val 을 파일에 저장하고 RecursiveSave(fout, T->nodes[i])를 수행한다.

T->nodes 를 다 읽으면 파일에 -1 을 저장한다.

이는 Open 에서 재귀적으로 파일을 읽을 때 해당 노드는 더 이상 연결된 노드가 없음을 알기 위해서이다.

void Open(const char *filename)

재귀적으로 파일을 여는 함수 RecursiveOpen(ifstream& fin, Nptr& T)를 정의한다.

void RecursiveOpen(ifstream& fin, Nptr& T)

파일을 한 줄씩 읽어 index 와 val 을 알아낸다.

index == -1 일 경우 함수를 종료한다.

T->nodes[index]에 val 을 저장하고 RecurSiveOpen(fin, T->nodes[index])와 RecursiveOpen(fin, T)를 실행한다.

2.1.1 트라이를 이용한 Word count 계산 및 사용자 테스트

실행

```
# ./Trie_word_count The-Road-Not-Taken.tokens.txt
# ./Trie_word_count Dickens_Oliver_1839.tokens.txt
# ./Trie_word_count_test The-Road-Not-Taken.trie
input (Exit : "EXIT")> the
9
input (Exit : "EXIT")> a
3
input (Exit : "EXIT")> abcdef
Not found
input (Exit : "EXIT")> not
2
input (Exit : "EXIT")> EXIT
# ./Trie_word_count_test Dickens_Oliver_1839.trie
```

```
input (Exit : "EXIT")> the
9497
input (Exit : "EXIT")> a
3760
input (Exit : "EXIT")> abcdef
Not found
input (Exit : "EXIT")> not
747
input (Exit : "EXIT")> EXIT
```

기타

트라이가 영문 소문자 알파벳에 대해 설계되었으므로 문장 부호를 비롯한 기호들은 제외하고 영어 단어에 대해서만 트라이를 실행하였다.

2.1.2 트라이와 해시 및 이진탐색트리 효율 비교

실행

```
# ./test_Trie_Hash_comp The-Road-Not-Taken.trie The-Road-Not-Taken.hash
N : 1000000
trie elapsed_time: 0.0380721 sec
hash elapsed_time: 0.0693493 sec
# ./test_Trie_Hash_comp Dickens_Oliver_1839.trie Dickens_Oliver_1839.hash
N : 1000000
trie elapsed_time: 0.0491359 sec
hash elapsed_time: 0.183299 sec
```

기타 사항

작성한 모든 프로그램은 Ubuntu 16.04 LTS 에서의 실행을 확인했으며 우분투에서 실행파일을 생성하기 위해 사용한 명령어는 다음과 같다

g++ -o <실행파일명> <소스코드> -std=c++11

우분투는 가상머신 VMware WorkStation 15 Player 에서 구동하였으며, CPU 는 i5-6500 이며 processor 은 2 개를 사용하였고, Ram 은 2GB 를 할당하였다.

Dickens_Oliver_1839.tokens.txt 의 첫 줄

ď » żoliver twist 에 유니코드 포맷 문자가 섞여 있었기에 이를 oliver twist 로 수정하여 실행하였다.

#include <sys/timeb.h> 에서 오류가 발생하여 이를 사용한 모든 코드에서 #include <sys/time.h>로 변경하였다.