0. serach function :

void searchNode(char \*key, struct Node\*\* p, struct Node \*\*q){

if( (\*p) ==NULL) //node is null

return ;

else if( strcmp((\*p)->key, "") == 0 ) //root is empty

return ;

else if ( (\*p)!=NULL && strncmp(key, (\*p)->key, MAXLEN ) == 0) //rootkey == key

return ;

else if( (\*p)!=NULL && strncmp(key, (\*p)->key, MAXLEN ) < 0) {

\*q=\*p;

\*p = (\*p)->leftChild;

return searchNode(key, p, q);

}

else {

\*q=\*p;

\*p = (\*p)->rightChild;

return searchNode(key, p, q);

}

}

함수가 끝나면 p는 key node를, q는 p의 parent node를 가리키고 있다.

단, 예외적으로 p==root일 때 q==root이다.

1. 노드 삽입

Main function call :

void main(){

int i=0;

struct Node root;

struct Node\* rootPtr = &root;

strcpy(root.key, "");

printf("1. node insert : \n");

for(i=0; i<KEYNUM; i++){

insertNode(&rootPtr, keyArr[i]);

printTree(&root);

printf("\n");

}

printf("\n\n");

…

\*keyArr은 헤더파일에 명시되어 있으며 one, two, thr, … 의 순으로 삽입할 key가 저장되어 있다.

Insert function :

void insertNode(struct Node\*\* root, char\* key){

struct Node\* p = \*root;

struct Node\* q = \*root;

struct Node temp;

if(p!=NULL){

searchNode(key, &p, &q);

}

if(strcmp((\*root)->key,"")!=0){

if(strncmp(key, q->key, MAXLEN) <0){

q->leftChild = (struct Node\*)calloc(1, sizeof(struct Node));

allocNode(&(q->leftChild), key);

}

else if(strncmp(key, q->key, MAXLEN) >0){

q->rightChild = (struct Node\*)calloc(1, sizeof(struct Node));

allocNode(&(q->rightChild), key);

}

}

else

allocNode(root, key);

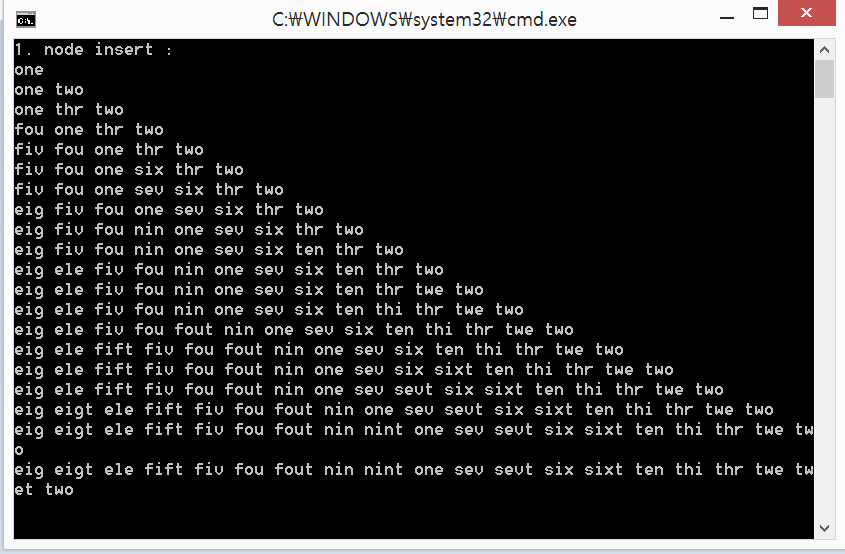
}

\*allocNode(struct Node\*\* node, char\* key) 에서는 node의 children을 null로 초기화 해주고, key를 복사해줌

\*root->key가 “”이 아닐 때 🡺 root가 null이 아닐 때는 key비교를 해서 insert

Root->key가 “”일 때 🡺root가 null일 때(key가 하나도 없을 때)에는 root에 자식과 key를 초기화.

실행 결과 :



2. 노드 삭제

Main function call :

void main(){

…

printf("2. node delete : \n");

printf("1) order = insert\n"); //삽입순서로 삭제

for(i=0; i<KEYNUM; i++){

deleteNode(&root, keyArr[i]);

printTree(&root);

printf("\n");

}

printf("\n\n");

printf("2) order = insert\n"); //삽입순서에 역순으로 삭제

for(i=0; i<KEYNUM; i++){ //re insert

insertNode(&rootPtr, keyArr[i]);

}

for(i=0; i<KEYNUM; i++){

deleteNode(&root, keyArr[KEYNUM-1-i]);

printTree(&root);

printf("\n");

}

Delete function :

0) root의 자식이 없을 때 🡪 root->key를 “”(공백)로 설정. Ret;

1) child가 없을 때 🡪 p와 q의 연결을 끊고 free(p);

2) children이 2개 일 때 🡪 left Subtree의 depth와 right subtree의 depth중 depth가 큰 쪽에서 key 대체.

(단, Left subtree와 right subtree의 depth가 같은 경우는 right subtree에서 key를 대체함)

a. Left subtree depth가 더 클 경우 🡪 left subtree중에 가장 큰 key node를 찾음(getMaxKeyAmongChild function 사용)

b. right subtree depth가 더 거나 양쪽의 depth가 같은 경우 🡪 right subtree중에 가장 작은 key node를 찾음(getMinKeyAmongChild)

tempkey에 get…KeyAmongChild()으로찾은 node에 key를 복사 해 두고

tree에서 tempkey값 삭제.

p->key 에 tempkey 복사.

3) child가 하나일 때 🡪 p가 q의 left/right child인지 판단해서 같은 방향에 q’s child->p’s child와 같이 연결 해 준다.

void deleteNode(struct Node\* root, char\* key){

struct Node\* p = root;

struct Node\* q = root;

searchNode(key, &p, &q);

if(root->leftChild == NULL && root->rightChild==NULL){

strncpy(root->key, "", MAXLEN);

return ;

}

//both left and right not exist

if(p->leftChild==NULL && p->rightChild ==NULL){

if( strncmp(key, q->key, MAXLEN) <0 )

q->leftChild = NULL;

else

q->rightChild = NULL;

free(p);

}

//both left and right exist

else if(p->leftChild!=NULL && p->rightChild!=NULL){

int LeftDepth = getMaxDepth(p->leftChild);

int rightDepth = getMaxDepth(p->rightChild);

if( LeftDepth > rightDepth ){

char temp[MAXLEN];

strncpy(temp, getMaxKeyAmongChild(p->leftChild)->key, MAXLEN);

deleteNode(root, temp);

strcpy(p->key, temp);

}

else{

char temp[MAXLEN];

strncpy(temp, getMinKeyAmongChild(p->rightChild)->key, MAXLEN);

deleteNode(root, temp);

strcpy(p->key, temp);

}

}

//have only one child

else{

struct Node\*\* temp;

if(p->leftChild!=NULL) temp = &(p->leftChild);

else temp = &(p->rightChild);

if( strncmp(key, q->key, MAXLEN) ==0 ){

strncpy(q->key, key, MAXLEN);

p->leftChild= NULL;

p->rightChild = NULL;

}

else if( strncmp(key, q->key, MAXLEN) <0 )

q->leftChild = \*temp;

else

q->rightChild = \*temp;

}

}

실행 결과 )

a. 삽입 순서와 동일하게 삭제



b. 삽입 순서에 역순으로 삭제

