제출일 : 2022-12-03

KWANGWOON UNIVERSITY

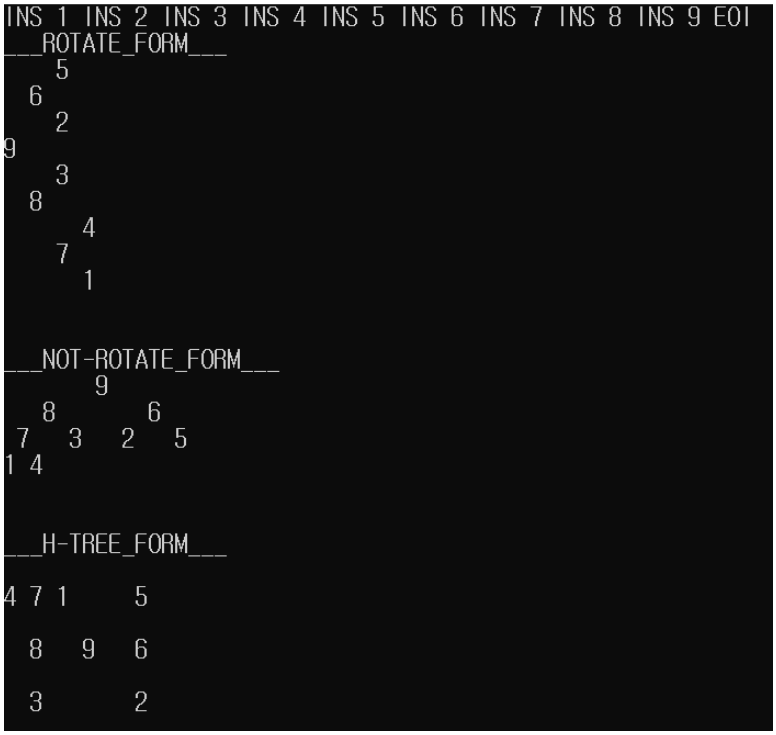
담당교수	김용혁 교수님	학 번	2021203034
학 과	소프트웨어학부	이 름	허찬영

KWANGWOON UNIVERSITY

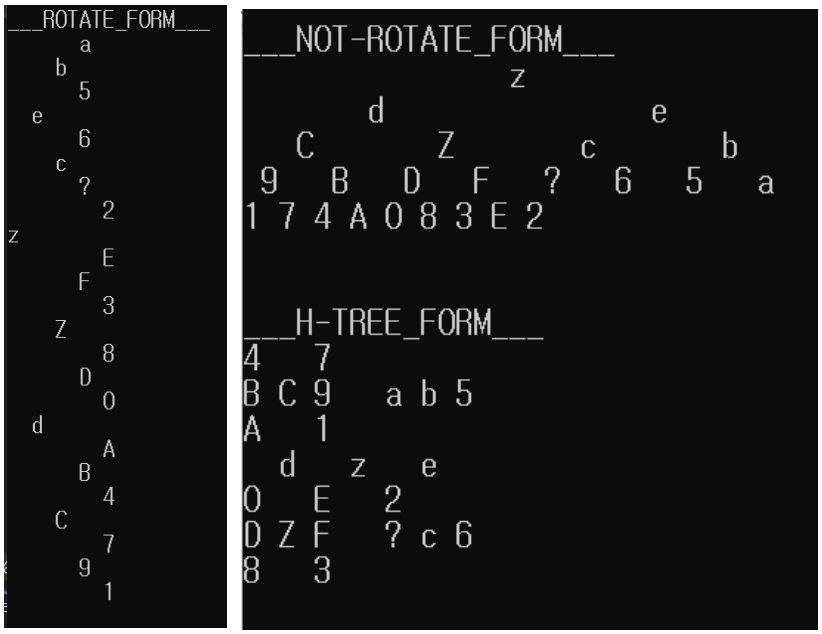
my environment : Visual Studio 2022 (Windows)

■ 프로그램 실행 결과

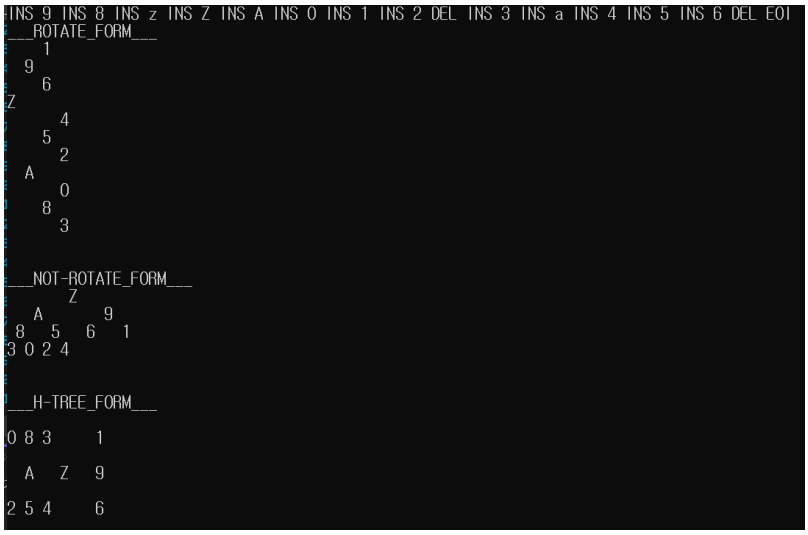
INS 1 INS 2 INS 3 INS 4 INS 5 INS 6 INS 7 INS 8 INS 9 EOI를 입력한 결과:



INS 1 INS 2 INS 3 INS 4 INS 5 INS 6 INS 7 INS 8 INS 9 INS 0 INS ? INS a INS b INS c INS d INS A INS B INS C INS D INS E INS F INS Z INS e INS z EOI를 입력한 결과:



INS 9 INS 8 INS z INS Z INS A INS 0 INS 1 INS 2 DEL INS 3 INS a INS 4 INS 5 INS 6 DEL EOI를 입력한 결과:



DEL 연산 시 현재 트리의 root node에 있는 데이터를 삭제한다. 그렇기 때문에 첫 번째 DEL 입력 시 가장 큰 데이터인 z를 삭제 하고 두 번째 DEL 입력 시 a를 삭제한다.

■ 주요 소스 코드

```
while(true) {
    cin >> command;
    if (!command.compare("EOI"))
        break;
    else if (!command.compare("INS")) {
        cin >> word;

        v.push_back(word);
        push_heap(v.begin(), v.end());
    }
    else if (!command.compare("DEL")) {
        pop_heap(v.begin(), v.end());
        v.pop_back();
    }
}
```

EOI가 입력될 때까지 계속해서 입력을 받는다. command는 Input operation을 저장하는 변수로 INS가 입력되면 추가로 문자 하나를 입력 받고 벡터에 저장한다. 그리고 push_heap() 함수를 호출해 벡터를 heap의 규칙에 맞게 정렬한다. 만약 DEL이 입력되면 pop_heap()을 호출해 부모 노드를 맨 마지막 자식 노드와 교환하고 재정렬한다. 그리고 pop_back()을 호출하여 부모 노드를

삭제한다.

```
typedef struct node {
    struct node* next;
    char data;
}node;

typedef struct Tree {
    node nodes;
    struct Tree* left;
    struct Tree* right;
    char data;
}Tree;
```

not-ratate_form은 연결리스트와 재귀 함수를 이용하여 구현하였다. 따라서 단일 연결리스트를 구현하기 위한 node 구조체와 이진 트리에 대한 구조체를 정의했다.

```
// rotate_form
rotate_opt = v;
for (int i = 0; i < rotate_opt.size(); i++) {

    key = rotate_opt[i];

    root = insert(root, key);

    binaryTree[cnt].data = root->data;
    binaryTree[cnt].left = NULL;
    binaryTree[cnt].right = NULL;
    root = root->next;

    if (cnt % 2 == 0) {
        binaryTree[cnt / 2].left = &binaryTree[cnt];
    }
    else {
        binaryTree[cnt / 2].right = &binaryTree[cnt];
    }
    cnt++;

}
printf("___ROTATE_FORM___\n");
print(&binaryTree[1], 0);
cout << endl << endl;
```

먼저 vector rotate_opt에 vector v를 대입하고 현재 node를 key로 저장한다. 그 후, insert 함수를 호출한다. insert() 함수는 root가 NULL이면 새로운 node 생성하고 다음 node를 NULL로 설정한다. 만약 다음 node가 있으면 재귀 호출을 한다. Complete Binary Tree에 사용할 node를 insert를 통해 생성하고 Tree 구조체 배열 binaryTree을 통해 Complete Binary Tree의 값과 자식 node를 저장한다. (root node는 cnt값이 1) 그리고 cnt의 값이 짝수이면 부모 노드의(cnt/2) 왼쪽 tree에 저장하고 홀수이면 오른쪽 tree에 저장한다. 그 후 print 함수를 호출하여 트리를 출력한단. print 함수는 강의에서 배웠던 것을 이용하여 구현하였다.

```
nrotate_opt = v;
while (sum < nrotate_opt.size()) {
    sum += pow(2, ndepth);
    ndepth++;
}
```

```
int ncnt = 0;
for (i = 0; i < ndepth - 2; i++) {
    cout << setw(pow(2, ndepth - i -1) - 1) << "";

    for (int j = 0; j < pow(2, i); j++) {
        if (ncnt < nrotate_opt.size()) {
            cout << nrotate_opt[ncnt++] << setw(pow(2, ndepth - i) - 1) << "";
        }
    }
    cout << endl;
}
```

다음은 not-rotate form이다. rotate form이 재귀 함수와 연결 리스트를 이용해서 구현을 했다면 not-rotate form은 단순히 배열을 이용해서 구현을 했다. 첫 번째 사진의 연산 결과로 트리의 depth를 구한다. 그리고 depth와 Complete Binary Tree의 특징을 이용하여 배열을 출력하여 구현했다.

```
if (ndepth - i == 2) {
    cout << setw(1) << "";
    for (int j = 0; j < pow(2, i); j++) {
        if (ncnt < nrotate_opt.size()) {
            cout << nrotate_opt[ncnt++] << setw(pow(2, ndepth - i) - 1) << "";
        }
    }
    cout << endl;
    i++;
}

if (ndepth - i == 1) {
    for (int j = 0; j < pow(2, i); j++) {
        if (ncnt < nrotate_opt.size()) {
            cout << nrotate_opt[ncnt++] << setw(pow(2, ndepth - i) - 1) << "";
        }
    }
    cout << endl;
    i++;
}
cout << endl << endl;
```

사실 처음에는 단순히 배열을 이용해서 구현하지 않았다. 그렇기 때문에 위 사진과 같이 depth에

따라 예외적으로 구현을 해야 됐지만 코드를 전반적으로 수정한 뒤에 약간의 수정을 거치니 정상적으로 트리 구조가 출력이 되어 앞서 구현했던 코드와 병합하지 않았다.

```
for (int i = 0; i < Htree_opt.size(); i++) {
    str.push_back(Htree_opt[i]);
    n++;
}

H(1, center(n), center(n), depth(n), 0, 1, 2, 3);

for (int i = 0; i < 10; i++) {
    for (int j = 0; j < 10; j++) {
        if ((H_tree[i][j] >= '0' && H_tree[i][j] <= '9') || (H_tree[i][j] >= 'A' && H_tree[i][j] <= 'z'
            || H_tree[i][j] == '?')
            cout << H_tree[i][j] << " ";
        else
            cout << "  ";
    }
    cout << endl;
}
```

```
void H(int node, int i, int j, int d, int U, int D, int R, int L)
{
    if (node > n) return;

    H_tree[i][j] = str[node - 1];

    if (2 * node <= n) {
        H_tree[i + d * V[L][0]][j + d * V[L][1]] = str[2 * node - 1];
        H(4 * node, i + d * (V[L][0] + V[U][0]),
            j + d * (V[L][1] + V[U][1]), d / 2, D, U, L, R);
        H(4 * node + 1, i + d * (V[L][0] + V[D][0]),
            j + d * (V[L][1] + V[D][1]), d / 2, U, D, R, L);
    }
    if (2 * node + 1 <= n) {
        H_tree[i + d * V[R][0]][j + d * V[R][1]] = str[2 * node];
        H(4 * node + 2, i + d * (V[R][0] + V[D][0]),
            j + d * (V[R][1] + V[D][1]), d / 2, U, D, R, L);
        H(4 * node + 3, i + d * (V[R][0] + V[U][0]),
            j + d * (V[R][1] + V[U][1]), d / 2, D, U, L, R);
    }
}
```

마지막으로 H-tree form이다. 사실 H-tree는 본 적도 없고 배운 적도 없던 형태이기에 자료에 있던 수도 코드에 의존하여 코드를 작성하였다. 새로운 vector인 str을 정의하고 str에 입력된 문자를 저장한다. 그리고 H() 함수를 호출한다. H() 함수 안에서 2차원 배열 H_tree에 올바른 문자를 저장한다. 그리고 이 입력된 문자가 올바르게 입력된 문자면 출력을 하고 아니면 공백을 출력한다.

■ Conclude with some comments on my work

이번 과제를 통해 Tree 구조에 대해 더 잘 이해할 수 있게 되었다. 그리고 Tree 구조를 구현하기

위해 재귀 호출이 굉장히 중요하다는 사실을 다시 한번 깨닫게 되었다. 또한 트리의 많은 형태가 있다는 것을 알 수 있었다. 특히 H-tree는 처음 본 형태의 트리인데 이번 과제를 통해 어떻게 구현되는지 알아 볼 수 있었다.