이진 트리 활용 길 찾기 게임

201811567_주용한

변경한 ADT (BTree.h)

수업 시간에 배운 내용 그대로 이진 트리(BTree.h)를 구성하였고 문제 해결을 위해서 struct Node에 int형 자료 x, y를 추가시켰다. (SetData(node, data) -> SetData(node, data, x, y))

추가한 ADT(ExTree.h)

문제를 해결하기 위해 ADT(ExTree.h)를 추가로 정의했다.

ExTree.h는 struct Tree를 가지고 있으며 트리 초기화 함수, 전위 순회, 후위 순회 함수, 트리에 노드 삽입, 노드 메모리 해제 함수 등을 갖고 있다.

사용자로부터 함수를 받아 y좌표 값에 따라 노드의 level 우선순위를 결정하도록 만들었다.

```
typedef int PriorityComp(int a, int b); // 사용자 정의 함수를 받음 (트리 level의 우선 순위 지정 가능)

typedef struct _Tree{
    BTreeNode *root;              // 트리 전체를 가르키는 root 포함
    PriorityComp *compY;          // 사용자 정의 함수 포함
    int numOfData;                // 트리 안 node의 개수 포함
}Tree;


void TreeInit(Tree *tree,  PriorityComp compY);   // 트리 초기화
int GetNumOfNode(Tree *tree);                     // 노드 수 반환
void InsertByY(Tree *tree, BTreeNode *bt);        // y좌표, 사용자 정의 함수에 따라 노드를 삽입
int PreoderTraverse(BTreeNode *bt, int nodeNum, int arr[]);    // 전위 순회
int PostoderTraverse(BTreeNode *bt, int nodeNum, int arr[]);   // 후위 순회
void freeNode(BTreeNode *bt);                     // 트리 안 노드 메모리 해제
```

ADT 구현 알고리즘

문제 해결을 위해서는 트리에 알맞은 위치에 노드를 삽입하고 전위 순회, 후위 순회 결과를 배열에 담기만 하면 된다.

즉, 알맞은 위치에 노드를 삽입하는 게 문제의 키 포인트이다.

서로 다른 노드는 y를 중복해 가질 수 있지만, x는 중복해 가질 수 없다.

- 모든 노드는 서로 다른 x 값을 가진다.

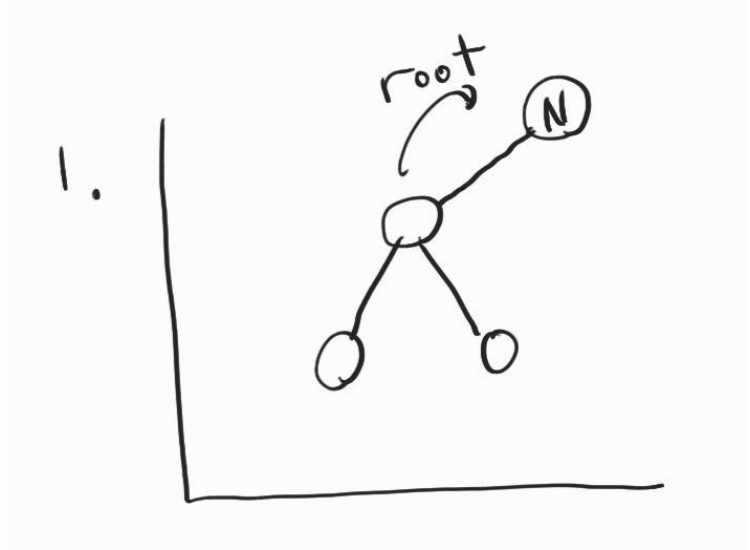
따라서 x를 기준으로 노드를 트리에 삽입하기로 결정했다. (ExTree.c 중 노드 삽입 코드)

```
void InsertByY(Tree *tree, BTreeNode *bt){
    if(tree->numOfData == 0) {                          //첫 삽입인 경우
        tree->root = bt;
    }
    else{
        if(tree->compY(bt->y, tree->root->y) > 0){   // 1. root보다 높은 경우
            bt->left = tree->root;
            tree->root = bt;
        }else{                                          // 2. root 보다 낮은 경우
            BTreeNode *place = tree->root;          //노드가 들어갈 위치를 찾는 포인터 선언
            while(place->right != NULL && tree->compY(bt->y, place->right->y) < 0){                                                 // 사용자 정의함수 활용
                place = place->right;
            }
            bt->left = place->right;                    // 노드 연결
            place->right = bt;
        }
    }
    tree->numOfData++;                                  // 노드 수 추가
}
```

X 오름차순으로 노드를 추가하며 위와 같은 방법으로 노드의 level을 찾아 간다.

위 코드를 사용하면 아래와 같은 방식으로 노드들이 정렬된다.

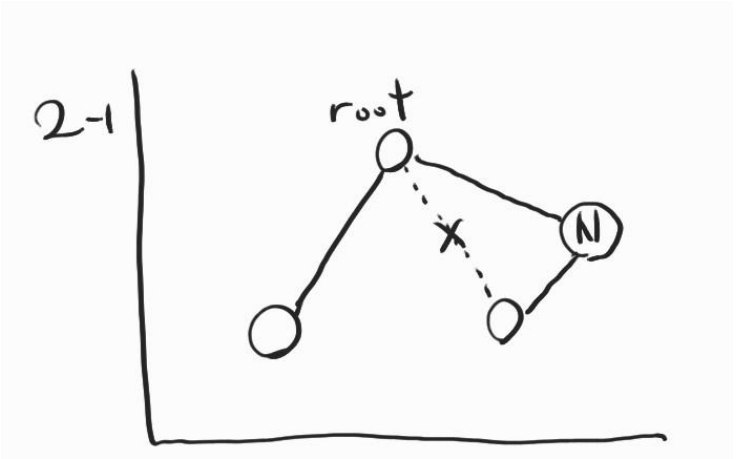
예시 1 – 루트보다 높은 노드를 추가했을 때



기존 루트노드 보다 새로운 노드의 level이 높은 경우

새로운 노드의 left에 기존 루트노드를 연결하고 루트노드를 새로운 노드로 바꾼다.

예시 2.1 – 루트보다 낮지만 루트의 right 노드 보다 높은 노드를 추가했을 때

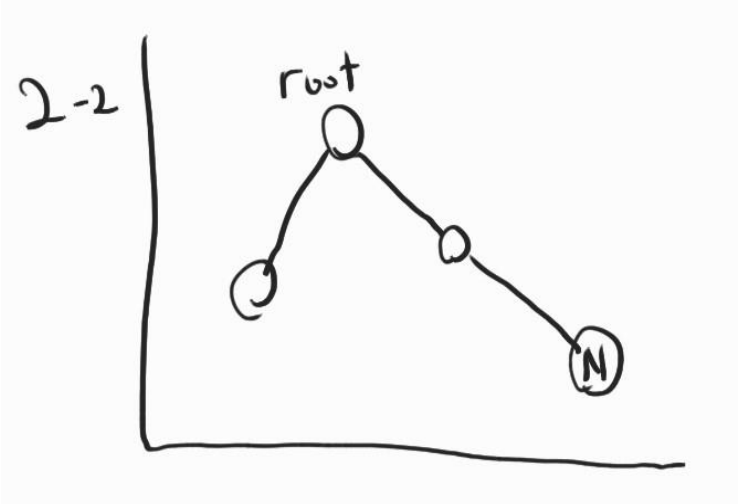


기존 루트노드 보다 새로운 노드의 level이 낮은 경우

루트노드의 right 자식 노드와 level을 비교하는 과정을 반복적으로 거친다.

right보다 낮다면 또 그 노드의 right와 비교한다.

예시 2.2 – 가장 낮은 노드를 추가한 경우



루트보다 낮은 level의 노드가 추가되면

위 2.1에서와 같이 반복적으로 오른쪽으로 내려가며 자신의 위치를 찾아 노드를 연결한다.

ADT를 활용한 함수 구현 알고리즘

일단 Tree에 사용될 사용자 정의 함수를 지정한다.

본 학생은 오른쪽과 같이 정의했으며 사용자 정의를 바꿀 경우 그에 맞는 테스트 케이스가 필요하다.

```
int CompY(int y1, int y2){
    return y1-y2;
}
```

ADT를 정의한대로만 잘 만들면 함수 알고리즘 구현은 아주 간단하다.

입력 받은 문자열 배열에서 작은 X좌표 순서대로 노드를 만들어 삽입하기만 하면 된다.

아래코드는 HW5_201811567.c의 void solution(nodeinfo, output) 함수 구현 중 핵심 부분이다.

```
for(int i = 1, d = 1; i < strlen(nodeinfo); i++){    // for - 가장 작은 x값을 찾기
    if(nodeinfo[i] == '['){                          //      '[' 기호를 찾으면
        x = atoi(&(nodeinfo[++i]));                  //       x 값 저장
        if(x > upper && x < lowX){                   //      x 값 비교, 작은 수 저장
            lowX = x;
            index = i;
            data = d;                                //      노드의 data 저장
        }
        d++;
    }
}                                                    // 작은 x 찾기 for loop 종료
```

문자열을 순회하며 x를 오름차순으로 찾는다.

그 후 data값과 x, y 값을 BTree.h의 SetData(node, data, x, y)를 통해 데이터를 넣고 삽입하면 문제에 맞는 트리가 완성된다.

그후

int PreoderTraverse(rootNode, 0, arr); // 전위 순회
int PostoderTraverse(rootNode, 0, arr); // 후위 순회
void freeNode(rootNode); // 트리 안 노드 메모리 해제
순회하며 데이터를 배열에 담고 freeNode()를 호출해 모든 노드의 메모리를 해제해주면 함수는 종료된다.
freeNode() 함수는 후위 순회를 하며 free(node)해주는 방식으로 작동된다.