10주차 예비보고서

전공: 컴퓨터공학과 학년: 2학년 학번: 20221532 이름: 김민기

1.

-recommend 함수

int recommend(RecNode \*root){

int max=0; // 미리 보이는 블럭의 추천 배치까지 고려했을 때 얻을 수 있는 최대 점수

int rot, x, y, child\_idx = 0, recScore;

int i, j, cnt = 0, delete=0;

root->child = (RecNode\*\*)malloc(sizeof(RecNode\*)\*CHILDREN\_MAX);

for(rot = 0; rot<NUM\_OF\_ROTATE; rot++) {

for(x = -2; x<WIDTH; x++) { //block이 배열의 idx=2번째 부터 있는 경우도 있으므로 -2부터 시작

//해당 x줄에 아예 블록을 놓을 수 없는지 검사

if(!CheckToMove(root->recField, nextBlock[root->lv], rot, 0, x)) continue;

y = 0;

while(CheckToMove(root->recField, nextBlock[root->lv], rot, y+1, x))

y++;

recScore = 0;

root->child[child\_idx] = (RecNode\*)malloc(sizeof(RecNode));

root->child[child\_idx]->lv = root->lv + 1;

for(i = 0; i<HEIGHT; i++)

for(j = 0; j<WIDTH; j++)

root->child[child\_idx]->recField[i][j] = root->recField[i][j];

AddBlockToField(root->child[child\_idx]->recField, nextBlock[root->lv], rot, y, x, &recScore);

recScore += DeleteLine(root->child[child\_idx]->recField);

if(root->child[child\_idx]->lv < BLOCK\_NUM)

recScore += recommend(root->child[child\_idx]);

if(recScore >= max) {

max = recScore;

if(root->lv == 0){

root->recBlockX = x;

root->recBlockY = y;

root->recBlockRotate = rot;

}

}

child\_idx++;

}

}

return max;

모든 회전과, x값을 반복하며 블록을 놓을 수 있으면 현재 블록을 필드에 추가한 뒤 다음 블록을 재귀적으로 호출하여 계산한다. 더 이상 다음 블록이 존재하지 않을 경우 점수를 비교하여 점수가 최대일 경우 첫 번째 블록의 위치를 기억하여 필드에 R로 그린다.

위의 코드를 보면 이중for문으로 모든 회전, x값을 반복하며 CheckToMove함수로 블록을 놓을 수 있는지 판단, 후 필드 맨 아래까지 블록을 내린다. 이후 트리 구조를 만들기 위해 자식 노드를 동적할당 해준 후 값을 대입해 준다. 이후 현재 블록을 추가하고, 완성된 줄을 지우며 점수를 계산한다. 이후 만약 현재 블록이 nextBlock[2]가 아니라면 다음 블록으로 재귀 호출을 하고, 만약 최댓값이고, 현재 블록이 nextBlock[0]이면 루트 노드의 값을 현재 블록으로 설정한다. (루트 노드를 R로 필드에 출력함)

해당 함수에서 사용하는 자료구조는 tree 이다. Root노드부터 시작되어 모든 회전과 x값에 해당하는 노드들을 자식으로 갖는다.

- modified\_recommend 함수

int modified\_recommend(MRecNode \*root) {

int max=0; // 미리 보이는 블럭의 추천 배치까지 고려했을 때 얻을 수 있는 최대 점수

int rot, x, y, recScore, child\_idx = 0;

int i, j, cnt = 0, delete=0, min;

root->child = (MRecNode\*\*)malloc(sizeof(MRecNode\*)\*CHILDREN\_MAX);

for(rot = 0; rot<NUM\_OF\_ROTATE; rot++) {

for(x = -2; x<WIDTH; x++) { //block이 배열의 idx=2번째 부터 있는 경우도 있으므로 -2부터 시작

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

for(j = 0; j<HEIGHT-root->recField[i]; j++)

recoveredField[j][i] = 0;

for(j = HEIGHT-root->recField[i]; j<HEIGHT; j++)

recoveredField[j][i] = 1;

}

//해당 x줄에 아예 블록을 놓을 수 없는지 검사

y = 0;

if(!CheckToMove(recoveredField, nextBlock[root->lv], rot, y, x)) continue;

while(CheckToMove(recoveredField, nextBlock[root->lv], rot, y+1, x))

y++;

recScore = 0;

root->child[child\_idx] = (MRecNode\*)malloc(sizeof(MRecNode));

root->child[child\_idx]->lv = root->lv + 1;

for(i = 0; i<WIDTH; i++)

root->child[child\_idx]->recField[i] = root->recField[i];

AddBlockToField(recoveredField, nextBlock[root->lv], rot, y, x, &recScore);

recScore += DeleteLine(recoveredField);

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

for(j = 0; j<HEIGHT; j++){

if(recoveredField[j][i] == 1){

root->child[child\_idx]->recField[i] = HEIGHT-j;

break;

}

}

}

if(root->child[child\_idx]->lv < BLOCK\_NUM)

recScore += modified\_recommend(root->child[child\_idx]);

if(recScore >= max) {

max = recScore;

if(root->lv == 0){

root->recBlockX = x;

root->recBlockY = y;

root->recBlockRotate = rot;

}

}

else

free(root->child[child\_idx]);

child\_idx++;

}

}

return max;

위의 코드는 과제에서 진행한 modified\_recommend함수이다. 변경된 점은 총 두가지로 우선 free를 통해 최댓값이 아닌 노드의 가지를 잘라낸다. 이를 통해 변경 전에는 모든 노드에 대한 메모리를 할당했지만 max를 판단하며 메모리를 최소화 할 수 있게 되었다.

두번째 방법은 노드 구조체가 갖고 있는 field를 일차원 배열로 변경한 것이다. 이를 복구하며 필드 데이터의 누수가 존재하게 되지만 더 많은 양의 메모리를 아낄 수 있다.

함수를 한 번 실행할 때 시간 복잡도는 O(WIDTH^4\*HEIGHT^3\*BLOCK\_NUM)이다. 공간 복잡도는 O(BLOCK\_NUM\*CHILDREN\_MAX) 만큼이 사용된다.

2.

첫 번째 코드를 두 번째 코드로 변경하며 바로 위의 문단에서 설명한 공간 복잡도가 작아졌다. 하지만 필드 데이터를 복구하는데 있어 데이터 손실이 불가피하여 이는 해결해야할 문제로 남아있다. 또한 시간 복잡도는 오히려 필드 데이터를 복구하고 압축하는 과정이 추가되어 증가하였다.

3.

이번 실험과 실습을 통해 한 프로그램에서 단계적으로 기능을 추가해 나가는 경험을 하게 되었다. 초기 주차에 하던 실습과 비교했을 때, 작성 해야 할 코드의 양이 증가하여 디버깅하며 오류를 잡아내는 것이 쉽지 않았다. 하지만 주차가 거듭되며 코드 수정을 통해 더 빠르게 오류를 잡아낼 수 있었다.

자료구조 과목에서 미리 학습한 Linked\_list와 tree구조를 활용하여 테트리스의 기능을 추가한 경험은 굉장히 흥미로웠다. 미리 학습하지 않았다면 더 긴 시간이 필요했겠지만 미리 공부한 자료구조를 활용하는 방식이었기 때문에 더 쉽게 자료구조를 사용할 수 있었다.