10주차 결과보고서

전공: 경제학과 학년: 4학년 학번: 20180501 이름: 김연수

1.

modified\_recommend :

int modified\_recommend(RecNode \*root){

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

int i, j, k, w, tempX = -1, tempY = -1, tempR = 0;

if(root == NULL) return 0;

// user code

int rotPos[7] = {2, 4, 4, 4, 1, 2, 2}; //블럭 id별 가능한 회전 개수

for(i=0;i<rotPos[nextBlock[root->lv]];i++) {

for(j=-2;j<=WIDTH;j++) {

int score\_sum = 0;

if(CheckToMove(root->field, root->curBlockID, i, -1, j) == 0)

continue;

int recommend\_y = -1;

//블럭을 맨 아래로 이동

while (CheckToMove(root->field, root->curBlockID, i, recommend\_y+1, j)){

recommend\_y++;

}

//x좌표 j의 위치에서 블럭이 쌓일 수 없는 경우.

if(recommend\_y == -1) continue;

//childnode에 정보 저장.

RecNode \*childnode = (RecNode\*)malloc(sizeof(RecNode));

childnode->curBlockID = nextBlock[root->lv + 1];

childnode->lv = root->lv+1;

childnode->score = root->score;

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

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

childnode->field[k][w] = root->field[k][w];

childnode->score += modified\_AddBlockToField(childnode->field, root->curBlockID, i, recommend\_y, j);

childnode->score += modified\_DeleteLine(childnode->field);

//더 낮은 위치에 있는 블록에게 추가 점수 부여해준다. -> 추천기능 효율 향상

childnode->score += recommend\_y \* 20;

//가지치기 기준 변수

// int pruning = (childnode->lv - 1) \* (childnode->lv - 1) \* 20 + (recommend\_y \* recommend\_y);

int pruning = 0;

//만든 childnode가 얼만큼 점수를 내는 지 그리고 max에 해당하는 지 검사 후 recRoot에 해당 child이 좌표저장.

if (childnode->lv < VISIBLE\_BLOCK) {

//낮은 스코어는 가지치기를 통해 잘라냄

if (pruning <= childnode->score) {

score\_sum = modified\_recommend(childnode);

}

if (score\_sum > max){

max = score\_sum;

tempX = j;

tempY = recommend\_y;

tempR = i;

}

} else {

if(childnode->score > max){

max = childnode->score;

tempX = j;

tempY = recommend\_y;

tempR = i;

}

}

free(childnode);

//공간 사용량 계산

SpaceSize += sizeof(childnode);

}

}

if(root->lv == 0) {

recommendY = tempY;

recommendX = tempX;

recommendR = tempR;

}

return max;

}

블럭 id별 가능한 회전 개수를 rotPos 배열에 저장하고, 가능한 회전 개수와 WIDTH의 조합으로 loop를 돌도록 했다.

일단 checkToMove함수를 통해 블록이 필드에서 유효한 지 확인한다. 그 위치에서 그대로 블록을 퍄아래로 내린다. 이 때의 y좌표를 recommend\_y에 저장한다. childnode에 메모리를 할당하고 다음 블록의 id, lv, score, field의 정보를 채워넣어 준다. root로 부터 상속받은 score에 추가적으로 현재 블록에 의한 점수까지 계산해줘야 한다. AddBlockToField함수와 DeleteLine함수로 score 변수에 점수를 추가해준다.

현재 노드의 lv이 VISIBLE\_BLOCK(MAX\_LEVEL)에 도달하기 전이라면, recommend함수를 재귀 호출한다. 인자로 childnode를 전달한다. recommend함수의 반환 값으로 scoresum을 받고 이 점수가 max보다 큰 경우 max를 교체하고, 좌표를 저장한다.

현재 노드의 lv이 VISIBLE\_BLOCK(MAX\_LEVEL)에 도달했다면, childnode\_score를 max의 값과 비교해 max보다 큰 경우 max를 교체, 좌표를 저장한다.

마지막으로 root->lv이 0인 경우, 최고 점수를 희득할 수 있는 위치의 좌표를 전역변수에 저장한다. 그리고나서 max를 리턴한다.

추천 기능 향상을 위해서 점수 산정 방식 몇 군데를 수정했다.

추천기능 향상을 위해서 저 낮은 y좌표 값을 가진 블록에게 더 큰 점수를 부여해줬다.

childnode->score += recommend\_y \* recommend\_y;

modified\_deleteLine함수는 deleteLine함수에 아래의 코드를 추가한 형태다. storesum에 점수를 합계하고 리턴하게 된다. tempsum을 이용해서 가로로 인접한 블록이 많을 수록 더 많은 점수를 얻도록 했다.

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

if (f[i][j] == 0) {

isLineFull = 0;

storesum += tempsum;

tempsum = 0;

break;

} else {

tempsum += 1;

}

}

storesum += tempsum;

modified\_AddBlockToField함수를 구현하기 위해 AddBlockToField함수에 아래의 코드 한 줄을 추가해줬다. 블록 아래에 빈공간이 있으면 감점을 부여했다.

if (f[i+blockY+1][j+blockX] == 0) count\_block -= 1;

2.

모든 경우를 고려하는 recommend함수와는 달리 modified\_recommend함수는 블록 id당 가능한 회전의 개수를 세어 중복되는 블록의 모양을 제거했다. 이로 인해 공간 및 시간 복잡도의 성능 향상을 기대할 수 있다.

텍스트, 스크린샷, 폰트, 블랙이(가) 표시된 사진

자동 생성된 설명 recommend 함수 적용

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명modified\_recommend함수 적용

modified\_recommend함수를 적용했을 때 시공간 효율성이 더 뛰어나다.

다만, pruning 또는 field 단순화를 적용하지 못해 시공간 복잡도를 더 줄이지 못했다는 것이 아쉽다. pruning을 코드에 적용시켜보려고 시도는 했지만, 효율성을 증가 시킬 수 있는 pruning 변수의 특이점을 찾지 못했다. field 단순화 또한 적용해보려고 노력해봤지만, 효율성이 증가되지 못하고 오히려 성능이 떨어지는 역효과가 생겼다.

3.

테트리스, 바둑, 체스와 같은 게임의 추천 시스템이 트리 구조를 이용해 구현된다는 점을 생각했을 때, 비슷한 원리를 가지는 시스템들에 tree 구조를 적용 시킬 수 있다. 조건에 따라 가능한 결과들이 여러 개가 존재할 때 루트에서 리프 노드까지 경로를 바탕으로 결정을 내리거나, 새 데이터를 분류하기 위해서 트리를 탐색할 수 있다.

tree를 탐색하는 과정에서 리프 노드의 숫자는 level이 증가함에 따라 기하 급수적으로 증가함을 알았다. 자식이 4개인 트리는 level이 3인 경우 모두 4^2 개의 노드를 가진다. level이 5가 되는 경우 4^4 만큼의 노드를 가진다. 따라서 기하급수적으로 증가하는 노드의 개수 때문에 시공간 복잡도의 중요성이 대두된다.

이번 프로젝트에서 시공간 복잡도를 줄이기 위해서 다양한 방면으로 시도를 해봤다. 가지치기, 데이터 단순화, 중복되는 정보 제거 등등. 평소 알고리즘 문제를 풀다 보면 시공간 복잡도는 별로 신경 쓰지 않게 된다. 해결하기만 하면 되는 문제가 대부분이라 문제를 푸는 것에 집중하게 된다. 그러나 이번 프로젝트를 통해 단순히 알고리즘을 설계하는 것은 쉽지만, 시공간 복잡도를 줄이는 것이 사실은 가장 중요하고 어려운 문제임을 체감할 수 있었다.