테트리스 3주차 결과보고서

전공: 철학과 학년: 3학년 학번: 20180032 이름: 남기동

1. 실습에 작성한 랭킹 시스템의 자료구조와 알고리즘, 완성한 알고리즘의 시간 및 공간 복잡도

실습에서 작성한 랭킹 시스템의 자료구조는 트리 구조를 사용했다.

```
int recommend(RecNode root) {
	int max = 0;
	int i, j; int tmp_blockX; int count = 0;
	int rotation = 0; int term = 0;
	int result = root.accumulatedScore;
	RecNode temp[40];
	for (rotation = 0; rotation < 4; rotation++) { //each rotation of block
		blockX = WIDTH / 2 - 2;
		blockY = -1;
		while (CheckToMove(root.recField, nextBlock[root.level], rotation, blockY, blockX - 1)) {
			blockX--;
		} //멈추는 지점의 blockX가 제일 좌측, 즉 시작지점이다

		for (term = 0; term <= 9; term++) {
			tmp_blockX = blockX + term;
			while (CheckToMove(root.recField, nextBlock[root.level], rotation, blockY + 1, tmp_blockX)) {
				blockY++;
			} //멈추는 지점이 block이 필드와 만나는 지점의 blockY와 blockX이다

			temp[count].recBlockX = tmp_blockX;
			temp[count].recBlockY = blockY;
			temp[count].recBlockRotate = rotation;
			temp[count].level = root.level + 1;

			for (j = 0; j < HEIGHT; j++) {
				for (i = 0; i < WIDTH; i++) {
					temp[count].recField[j][i] = 0;
					if (root.recField[j][i] == 1) {
						temp[count].recField[j][i] = 1;
					}
				}
			}
			result += AddBlockToField(temp[count].recField, nextBlock[root.level], temp[count].recBlockRotate, temp[count].recBlockY, temp[count].recBlockX);
```

```
                        result += DeleteLine(temp[count].recField);

                        if (temp[count].level < MAX_LEVEL) {
                                temp[count].accumulatedScore = result;
                                temp[count].accumulatedScore = recommend(temp[count]);
                        }
                        else {
                                temp[count].accumulatedScore = result;
                        }

                        if (temp[count].accumulatedScore >= max && temp[count].level == 1)
{ //현재까지 최대 누적 점수와 비교해서, 가장 큰 누적 점수를 갖는 블록의 회전수와 위치를 갱신
                                recommendR = temp[count].recBlockRotate;
                                recommendX = temp[count].recBlockX;
                                recommendY = temp[count].recBlockY;
                                max = temp[count].accumulatedScore;
                        }
                        else if (temp[count].accumulatedScore >= max) max =
temp[count].accumulatedScore;

                        blockY = -1;
                        result = root.accumulatedScore;
                        if (CheckToMove(root.recField, nextBlock[root.level], rotation, -1,
tmp_blockX + 1));
                        else break;

                        count++;
                }
        }
        return max;
}
```

각각의 child를 만드는 것은 함수 내부적으로 정적 할당을 사용하였다. 일단 root로부터 정보를 받아와야 한다. 특히 root의 accumulatedScore는 각각의 노드가 높은 점수를 산출할 수 있는 것인지를 판단하는 중요한 징표이기 때문에 이 값을 신경써서 처리해야 한다. Parent로부터 받아온 값에다가 다음 블록을 계산했을 때의 점수를 누적해서 계산해야 하기 때문에 result라는 변수를 설정하여 처음에 parent의 누적 점수를 result에 받아오고 시작된다. 그 다음으로는 이번에 고려한 블록에 관하여 4가지 회전과 필드에 놓일 수 있는 위치를 2개의 for loop을 이용하여 구현하고 각각의 경우에 대하여 RecNode temp에 값을 저장한다. 그리고 addBlocktoField 함수를 호출하여 해당 노드의 경우 점수가 어떻게 누적되는지 result에 더하고 deleteLine함수를 호출하여 같은 작업을 수행한다. 이후 해당 노드의 level이 MAX_LEVEL보다 작다면, 즉 leaf에 해당하지 않는다면 recommend 함수를 재귀적으로 호출한다. 그렇지 않은 경우, 즉 leaf의 경우에는 accumulatedScore를 갱신한다.

그 다음 해당 level의 노드들 사이에서 비교하면서 값을 갱신한 max 값과 해당 노드의 accumulatedScore의 값을 비교하여 만약 해당 노드의 값이 더 크다면 해당 값으로 max 값을 갱신한다. 이 때 중요한 것은 만약 temp의 레벨이, 즉 지금 고려하고 있는 노드들의 level이 1일 때만 해당하는 위치를 recommend하는 x좌표, y좌표, 회전수로 갱신하는 것이다.

이렇게 총 2번의 for loop을 모두 돌고 난 다음에는 max 값을 리턴하면서 함수가 종료되는 구조를 갖는다.

완성된 알고리즘의 시간 복잡도는 worst case를 고려했을 때는 만약 1개의 경우를 고려할 때 4개의 회전수와 9개의 x좌표를 모두 고려해야 하며 depth에 따라 modified recommend 함수를 재귀적으로 호출해야 한다. 만약 depth가 n이라고 가정한다면, O(36^n)의 시간 복잡도를 갖게 된다. 공간 복잡도의 경우 정적 할당을 사용하였기 때문에 O(1)로 볼 수 있다.

2. 모든 경우를 고려하는 tree구조와 비교해서 어떤 점이 더 향상되었는가

완성한 알고리즘, 즉 modified_recommend 함수에는 크게 2가지 변화가 있었다. 첫 번째는 nextBlock의 형태에 따라 고려해야 하는 회전 수가 적어질 수 있기 때문에 이를 반영한 것이다. 절반에 가까운 블록의 형태가 회전수 2개만 고려하면 나머지는 동일한 경우가 있고 네모블록의 경우에는 회전수 1개만 고려하면 나머지는 모두 동일하게 생겼기 때문에 고려하는 노드를 줄일 수 있다. 따라서 for loop을 들어가기 전에 이번에 고려해야하는 블록의 ID 값에 따라 max_rotation을 정하는 방식으로 고려하는 노드를 줄이는 가지치기를 수행했다.

두 번째로 수행한 것은 addBlocktoField함수에서 블록이 필드의 옆면에 닿았을 때, 그 횟수를 조사하여 modified_recommend 함수에서 result 값을 갱신할 때 반영한 것이다. 그리고 addblocktoField 함수와 deleteline함수의 값을 result에 더할 때 라인이 지워지는것에 더 큰 비중을 주고자 가중치를 추가했다. 이러한 변화들을 통해 블록이 라인을 지우는 방향으로 더 추천하게 되었으며 라인 끝 부분이 채워지지 않아 빠르게 종료되던 기존 recommend 함수의 단점을 보완했다. 시간 복잡도를 고려한다면, worst case를 고려했을 때는 만약 1개의 경우를 고려할 때 4개

의 회전수와 9개의 x좌표를 모두 고려해야 하며 depth에 따라 modified recommend 함수를 재귀적으로 호출해야 한다. 만약 depth가 n이라고 가정한다면, O(36^n)의 시간 복잡도를 갖게 된다. 물론 이는 기존 recommend 함수와 동일한 것인데 worst case가 4가지 회전수를 갖고 있는 블록만을 고려하는 것이고 worst case가 아니라 average를 따진다면 modified_recommend함수의 시간복잡도는 낮아질 것으로 예상된다. 공간 복잡도의 경우 정적 할당을 사용하였기 때문에 O(1)로 볼 수 있다.

3. 습득한 내용이나 느낀점

이렇게 재귀적 함수를 잘 사용하면 간단하고 몇 줄 안되는 코드를 가지고 여러 개의 블록을 고려하면서 추천 함수를 짤 수 있다는 것에 놀랐다. 실습에서는 비록 재귀적으로 작동하는 recommend함수를 제대로 이해하지 못하여 어려움을 겪었지만 시간을 갖고 고민한 결과로 더 높은 점수를 산출할 수 있는 recommend 함수를 짤 수 있어서 좋았다. 또한 공간 복잡도의 경우 정적 할당을 사용하는 경우 효율적이라는 생각을 할 수 있었다. 나아가 직접 할애된 시간과 공간을 계산하는 것을 사용함으로써 프로그램을 짤 때 이런 부분들이 중요함을 깨달았다.