실험 PRJ-1 3주차 테트리스 프로그램 예비보고서

전공: 컴퓨터공학 학년: 2학년 학번: 20201635 이름: 전찬

**1. 목적**

3주차 실습에서 구현할 테트리스 추천 기능이 어떠한 원리로 작용하는지 이해한다. 또한 추천 기능을 구현할 때 사용하는 tree 구조의 장 · 단점을 파악한다. 마지막으로 tree 구조의 단점을 해결할 수 있는 방법을 2가지 이상 생각해본다.

**2. 추천 기능의 작동 원리**

3주차 실습에서는 recommend 함수를 통해 현재 조작하고 있는 tetris 블록과 이후에 나올 2개의 블록 정보를 바탕으로 현재 조작하고 있는 tetris 블록이 어느 위치에 들어갈 경우에 가장 높은 점수를 기대할 수 있는지 알려주는 역할을 수행한다. 여기에서 recommend 함수 자체에서는 고려하는 블록의 수와 상관없이 작동할 수 있어야 한다. 예를 들어, nextblock[3]이 새롭게 정의된다고 해도, recommend 함수는 제 역할을 수행해야 한다.

recommend의 작동 원리를 알아보자면, 기본적으로 이중 for loop을 사용하는 형태로 child node pointer(\*c)에 할당을 해주는 형태이다. 첫 번째 for loop 에서는 4가지 회전에 대한 경우의 수(rotate=0~3)에 대한 for loop 이며, 두 번째 for loop 에서는 x축으로의 이동(가능한 위치)에 대한 for loop의 구현을 통해서 한 block에서 모든 경우를 구현해줄 수 있다.

두 번째로 이렇게 만들어낸 node를 통해서 최대 점수 accumulatedScore를 얻어내는 과정을 진행해야 한다. 이는 tree 형태를 읽어가는 과정으로 수행해낼 수 있다. 이번 실습에서는 BFS(Breadth First Search)와 DFS(Depth First search)이라는 개념을 통해서 위 과정을 수행할 수 있다. 두 과정 모두 최대 점수가 무엇인지 알아낼 수 있지만, 이번 실습에서는 개인적으로 DFS형태의 구현으로 최대 점수를 찾아낼 예정이다.

이후 DrawRecommended 함수를 통해서 고려하는 블록들에 대해서 recommend 함수를 바탕으로 얻은 현재 블록 위치를 필드 상에 그려준다. 그려줄 때도 필드 값에 추천 블록을 추가해주는 것이 아닌, shadow를 그리는 것과 비슷한 형식으로 수행해서 데이터의 충돌을 막을 예정이다.

**3. tree 구조의 장 · 단점**

위와 같이 recommend 함수를 작성할 때 우리는 Linked list로 구현하는 tree 구조를 사용하게 된다. 이렇게 tree 구조를 사용할 때 장단점이 존재하는데, 아래와 같다.

-장점

기본적으로 tree 구조를 사용하기 때문에, 한 node에서 다양한 node로 link 되는, 간단하게 말해서 자식 노드가 여러 가지인 형태를 쉽게 구현할 수 있다. 또한 BFS나 DFS 형태로 검색을 진행할 때 모든 경우에 대해서 검색을 진행할 수 있기 때문에, 정확하다고 할 수 있다. 또한 현재 tree 에서는 사용하지 않지만, 만약 이미 완성된 tree 내에서 하나의 자료를 검색한다고 할 때, 시간 복잡도가 O(log n)형태로 상당히 효율적이라는 것을 알 수 있다.

-단점

위에서 설명한 것처럼 모든 경우에 대해서 검색을 하기 때문에, 비효율적이다. 이번 실습에서는 nextblock 배열의 길이=n 이라고 할 때 O(36n)형태로 시간 복잡도가 지수함수의 형태로 커질 수 있기 때문에, 배열의 길이가 길 때 사용하기에는 부적합하다. 또한 각 노드를 동적 할당 해주어야 하는 입장이기 때문에, 공간 복잡도도 지수함수의 형태로 증가할 것임을 쉽게 파악할 수 있다. 마지막으로 구현이 어렵다는 점과 함께, 데이터를 삽입 또는 삭제할 경우에 Linked List와 다르게 고려해야 하는 점이 더욱 많은 것도 단점으로 볼 수 있다.

**4. tree 구조의 단점을 해결할 수 있는 방법**

위와 tree에서 발생할 수 있는 문제를 해결하기 위해서 사용할 수 있는 방법에는 우선pruning 방법이 있다. 이 방식은 tree에서 시간 복잡도를 줄이기 위해 사용될 수 있다. Pruning 이라는 단어는 ‘가지치기’를 의미하는데, tree에서 첫 번째 score를 읽을 때, 가장 작은 score 형태의 node들은 그 이후로 검색하지 않고 잘라내는 형식으로 검색하는 데에 걸리는 시간을 줄여낼 수 있다.

두 번째 방법으로는 사용한 tree에 대해서 바로 할당 해제(free 함수)를 사용하는 방법이 있다. 이 방법을 통해서는 공간 복잡도를 줄여낼 수 있다. 위 방법의 예로, 36 x 36 x 36가지의 경우의 수 중에서 1->1->1으로 검색을 진행할 때, 1의 정보를 저장시켜 놓고, 마지막 1을 할당 해제한다. 그 이후에는 1->1->2를 검색하고, 2를 할당 해제하는 형태로 진행할 수 있다. 이 형태를 통해서 공간 복잡도를 줄여낼 수 있다.(사실 시간 복잡도<->공간 복잡도 형태이다)