테트리스 3주차 예비보고서

전공: 철학과 학년: 3학년 학번: 20180032 이름: 남기동

1. 3주차의 추천 기능의 작동 원리, Tree 구조의 장점과 단점을 기술(효율성 측면)

3주차에 구현할 추천 기능은 현재 주어진 블록과 그 다음의 블록 등을 고려하여 현재 블록을 필드의 어느 위치에 놓는 것이 가장 높은 점수를 산출할 수 있는 것인지 알려주는 알고리즘이다. 3주차의 실습에서는 추천 기능을 구현하기 위해서 tree 자료구조를 활용하고 있다. Tree 자료구조에서는 parent가 child를 갖는 구조를 갖기 때문에 현재의 블록이 놓일 수 있는 경우의 수를 기반으로 다음으로 고려하는 블록이 놓일 수 있는 경우의 수를 체계적으로 정리하고 정보를 저장할 수 있다. 실질적으로 구현하기 위해서는 우선 고려하는 블록의 회전 수를 고려해야 한다. 회전 수에 따라 필드에 놓일 수 있는 위치가 달라지기 때문이다. 그 다음 정해진 각각의 회전 수에 따라 블록이 필드 위에 놓일 수 있는 위치를 구하고 이러한 정보들을 바탕으로 child를 만들고 정보를 저장한다. 그 후에 현재 tree의 level, 즉 depth가 최대라면 recommend 함수에서 고려하려고 했던 블록을 모두 고려했다는 뜻이다. 따라서 이 경우에는 현재 점수를 계산하고 그렇지 않은 경우에는 아직 고려해야 할 블록이 더 있다는 뜻이기 때문에 재귀적으로 recommend 함수를 호출하게 된다. 이러한 방식을 통해 마지막에는 최대 누적 점수를 갖는 블록의 위치와 회전수를 얻고 drawRecommend 함수를 통해 shadow 함수가 작동하는 것과 유사하게 recommend 되는 위치를 필드에 그리도록 한다.

앞서 간단히 이야기했지만 tree 자료 구조를 사용하는 것의 장점은 고려되는 블록이 늘어나더라도 기존 블록의 경우에서 파생되는 다음 블록의 경우를 연결 짓는 구조이기 때문에 가능한 경우의 수를 체계적으로 구축할 수 있으며, parent와 child를 노드 포인터로 연결 지어 놨기 때문에 이동도 용이하다고 볼 수 있다. 따라서 tree를 이용하여 더 많은 블록을 고려할수록 더 나은 점수를 달성할 수 있다는 장점이 있다. 이를 비교하고자 singly linked list를 고려해보면 head에서 시작해서 첫 번째로 고려하는 블록의 가짓수를 차례대로 연결 짓는다고 하더라도 두 번째로 고려하는

블록이 생겼을 때 여전히 차례대로 연결 지어야 하기 때문에 첫 번째 블록의 경우에서 두 번째 블록이 어떤 위치에 놓이고 어떤 누적 점수를 갖는 지와 같이 추천 기능을 구현하기 위해 필요한 정보를 얻기가 매우 힘들어진다.

그러나 tree 구조를 사용하는 것에 단점도 존재한다. 첫 번째는 고려되는 블록이 늘어날수록, 즉 tree의 level이 늘어날수록 시간 복잡도가 급격하게 증가하는 것이다. 예를 들어 회전수와 블록의 x좌표를 고려해봤을 때 각 블록에 대해 34가지의 블록 위치가 가능하다고 고려해보면, tree의 노드 수는 level이 증가할 때마다 34의 지수가 한 단계씩 증가하는 수준으로 굉장히 급격한 속도로 증가하게 되는 것이다. 이런 경우 level이 4만 되어도 100만 개가 넘는 노드를 저장해야 하기 때문에 이를 처리하기 위한 시간이 오래 걸린다. 동시에 정보를 저장하는 노드가 급격히 늘어나는 만큼 시간적 측면 외에도 공간적 측면에서도 굉장한 비효율을 초래한다. 하나의 노드에 필요한 메모리 공간을 a라고 한다면 tree의 level이 n일 때, a x 34^n 만큼의 메모리 공간을 사용하게 된다. 결과적으로 tree 구조는 level이 올라갈수록 시간적, 공간적 비효율이 높아진다고 정리할 수 있다.

2. Tree 구조의 비효율성을 해결할 방법 2가지

위와 같은 tree 구조의 비효율성을 해결하기 위해서는 가지치기와 데이터의 단순화가 필요하다. 가지치기는 모든 노드를 다 고려하지 않음으로써 시간적으로도, 공간적으로도 효율적이게 만들 수 있는 것이다. 데이터 단순화의 경우에는 필드의 정보를 모두 저장하는 것이 공간의 비효율성을 초래하기 때문에 공간의 모든 정보를 저장하는 것이 아니라 필드의 높이만을 기억하는 방법이 있을 수 있다. 하지만 이처럼 자료에 나와있는 가지치기와 데이터의 단순화의 경우에는 여전히 문제점이 있다. 가지치기의 경우에는 자료에서도 나와 있듯, 높은 점수를 갖는 가지만을 남기자니 다음 블록까지 고려했을 때 오히려 적은 점수를 산출하는 경우의 수를 출력하게 되기 때문이다. 데이터 단순화의 경우에도 필드에 구멍이 뚫려 있는 구멍에 대한 정보가 소실되기 때문에 추천 알고리즘을 구현하는데 있어서 문제가 발생할 수 있다.

이를 해결하기 위해서는 가지치기와 데이터 단순화에 약간의 변화들을 적용해야 한다. 우선, 가지

치기의 경우에는 가장 높은 누적 점수를 갖는 노드만 남기는 경우에는 여러 개의 블록을 고려하는 경우 오히려 최고 점수를 산출하는 경우의 수를 제거해내는 문제가 발생할 수 있다. 따라서 가장 높은 누적 점수만을 남기기 보다는 상위 10개의 점수를 갖는 노드들에 한해서만 다음 블록에 대해서도 고려해보는 방법을 사용할 수 있다.

다음으로 데이터 단순화의 경우에는 단순히 높이만을 기억하는 경우에는 recommend 함수가 라인을 지우는지에 관해서도 점수가 추가되어야 하는데 구멍이 있는 경우 라인이 지워지지 않는다는 점을 고려하지 못하게 된다. 따라서 높이 이외에도 각각의 라인이 구멍을 갖고 있는지 아닌지에 대한 정보는 저장하고 있어야 한다.

나아가 가지치기와 관련하여, 단순히 가지를 칠 때 해당하는 노드가 기존의 점수 산출 방식에 의존하기 보다는 추가적인 부분들을 고려할 때 보다 정확하게 가지치기를 할 수 있을 거라 생각한다. 지금의 경우에는 점수가 산출되는 방법은 단순히 블록 아래에 필드가 있는 경우 개당 +10점, 라인을 지우는 경우 (지운 줄의 개수^2) x 100을 추가하고 있다. 하지만 블록이 기존의 필드에 깔끔하게 맞아떨어지는 형태로 놓일 때 우리는 테트리스에서 좋은 점수를 얻을 수 있다는 것을 알고 있다. 따라서 단순히 블록 아래의 필드가 닿은 경우 뿐만 아니라 블록 옆면에 필드의 블록이 맞닿은 경우, 나아가 블록이 필드의 벽면에 붙어서 놓여진 경우에도 좋은 선택지라 고려되어야 한다. 따라서 라인을 지우는 경우, 벽면에 놓이는 경우, 필드 블록에 좌우,아래로 맞닿는 경우 가중치를 부여하여 점수를 계산하고 이를 통해 가지치기를 활용한다면 보다 정확하게, 적은 노드들을 남기는 방향으로 가지치기를 하면서 원하는 결과를 얻을 수 있을 것이라 예상된다.