2020년도 2학기 컴퓨터공학설계및실험I

9주차 예비 보고서

20170175 김태안

1. 실습 목적

테트리스 게임을 플레이하는 사용자를 위해 어떤 위치에 블록을 놓으면 높은 점수를 받을 수 있을지를 추천하는 추천 시스템을 구현한다. 추천된 블록은 1주차 숙제의 그림자 기능을 이용해서 화면상에 나타낸다. 이 과정을 통해서 tree와 포인터(pointer)에 대한 이해를 높일 수 있다.

1. 관련 이론

트리 구조란, 정보의 항목들이 가지로 연결될 수 있게 데이터가 조직되는 것을 말한다. 정의에 따르면 트리는 1개 이상의 노드로 이루어진 유한 집합으로, 노드 중에는 root 노드가 하나 있고, 나머지 노드는 루트의 부분집합인 서브 트리로 분리될 수 있다. 노드란 한 정보 아이템에서 다른 노드로 뻗어 있는 가지를 합한 것을 의미한다. 한 노드의 서브 트리의 root를 자식 노드라고 하고, 그 노드를 자식 노드의 부모 노드라고 한다. 부모가 같은 자식 노드들을 형제 노드라고 한다.[1]

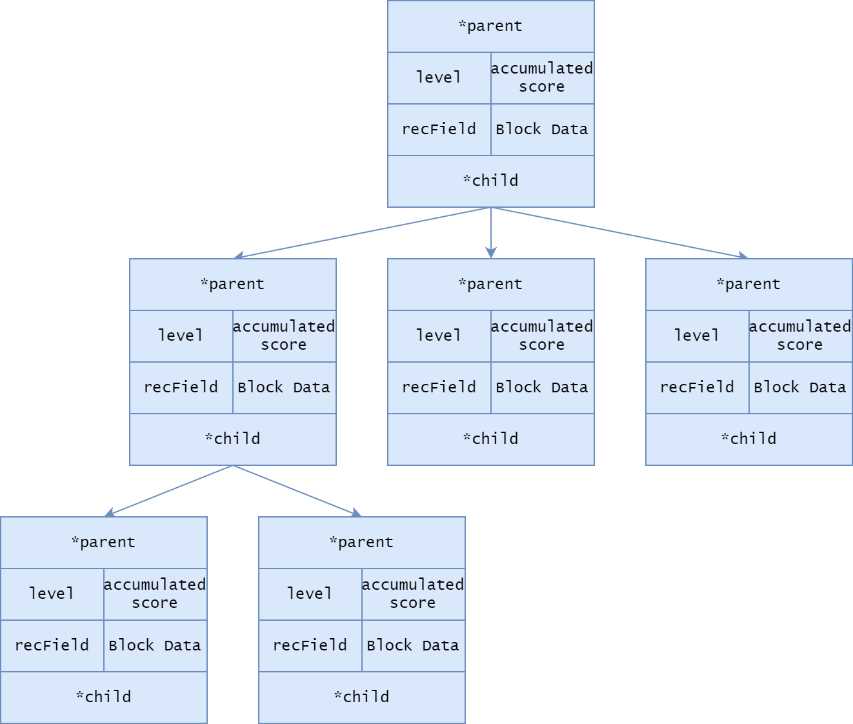
테트리스에 추천 시스템은 현재 블록과 그 다음 블록를 기준으로 가장 많은 점수를 낼 수 있는 방법을 계산한다. 이때, 트리 구조를 사용하여 데이터를 처리한다면 현재 블록에 따른 다음 블록의 상태를 모두 저장할 수 있다. 현재 블록 상태에 다음 블록 상태가 연결된 형식이기 때문에 (현재 블록 상태, 다음 블록 상태)의 쌍으로 데이터를 저장할 때보다 더 적은 메모리를 사용한다. 그러나 여전히 342개의 많은 메모리를 차지하고, 트리의 탐색이 시간이 오래 걸린다는 단점이 존재한다.

트리의 효율성을 높이기 위한 방법으로 트리가 모든 노드를 가지는 것이 아닌 일부만 가지게 할 수 있다. 예를 들어, 현재 노드와 삽입하려는 노드가 주는 점수가 특정 점수보다 낮다면, 아예 추천에서 배제해 트리의 노드로 추가하지 않는 것이다. 이 경우, 최악의 경우에는 노드의 개수가 그대로이지만, 확률 상 매번 최악의 경우가 나오지는 않으므로 더 적은 메모리를 사용할 것이다.

다른 방법으로는 테트리스의 공략을 이용하는 것이다. 테트리스에서 가장 높은 점수를 받는 방법은 한번에 4줄을 삭제해 1600점을 획득하는 것이다. 이를 위해 많은 플레이어들은 맨 왼쪽이나 오른쪽 칸은 비워두고, □□□□ 블록이 나오면 해당 위치에 블록을 삽입해 4줄을 없애도록 유도한다. 이를 이용해 HEIGHT\*(WIDTH-1)의 필드를 가정하고, □□□□은 무조건 가장자리에 삽입하도록 추천시스템을 구현한다면, 필드 정보가 줄어들어 공간 복잡도에서 효율이 발생하고, 시간 복잡도에서도 블록 하나를 고려하지 않으므로 효율이 발생한다.

1. 실습 방법

실습에서 구현할 트리는 다음과 같은 구조를 가진다.



각 노드는 부모 노드, 자식 노드, 깊이, 블록 정보, 누적 점수, 필드 상태를 원소로 가진다. 자식 노드는 필요한 노드의 개수만큼 동적으로 할당해 상황에 따라 다른 개수를 저장할 수 있도록 한다. 트리가 내려갈 때마다 level은 증가하고, 누적 점수를 더해 최대 점수를 가지는 블록 정보를 택해 반환한다.

recommend 함수의 pseudocode는 다음과 같다.

|  |
| --- |
| recommend() {  for every rotation of block  for every position possible  make field assuming the block is put  store all information  if (current\_level < MAX\_LEVEL)  max value += recommend()  return node with max value  } |

모든 rotate와 position에 대해 field를 만들고, 이때의 점수와 블록을 기록한다. 그리고 이를 재귀적으로 반복해 각 경우에 대해 다음 블록이 놓일 수 있는 위치와 방향에 블록이 놓인 필드를 가정해 만들고, 이때의 점수와 블록을 기록한다. 그리고 가장 큰 값을 가진 노드를 반환한다. 반환 받은 level 1은 level 1에서의 가장 큰 값을 반환해 게임 필드에 그린다.

이때 시간 복잡도는 Rotate\*Position의 연산이 모든 Rotate\*Position에 대해 한번 더 일어나므로 O(Rotate2\*Position2)이고, 트리는 Rotate\*Position개의 field를 저장하므로 공간 복잡도는 O(Rotate2\*Position2\*HEIGHT\*WIDTH)이다.

1. 기타

**Pruning Tree**

프루닝은 그래프를 검색할 때, 모든 가지를 검색할 경우 효율성이 떨어지기 때문에 검색할 필요가 없는 노드는 제외하여 검색하는 방법이다. 현재 평가하는 노드보다 이전에 평가한 노드가 좋지 않을 경우 검색에서 제외하는 방식이다. 테트리스의 예를 들면, 현재 탐색한 결과 a위치에 놓을 경우 200점을 얻는다고 가정해보자. 다음에 탐색한 결과 b가 150점 밖에 얻지 못한다면 b의 결과는 트리에 추가하지 않고 넘어가는 방식이다. 이와 같은 알고리즘은 기계가 플레이하는 2인용 게임에서 주로 사용된다.[2]

1. 참고 문헌

[1] “5. Tree.” *Fundamentals of Data Structures in C*, by Ellis Horowitz et al., University Press, 2008.

[2] “Alpha–Beta Pruning.” *Wikipedia*, en.wikipedia.org/wiki/Alpha%E2%80%93beta\_pruning.