8주차 결과보고서

전공: 수학/컴퓨터공학 학년: 3학년 학번: 20181288 이름: 윤성호

1. 실습시간에는 테트리스 게임을 할때 가장 최적의 위치에 블럭을 배치할 수 있도록 추천해주는 프로그램을 작성하였다. 현블록과 그 이후에 나올 블록들이 배치될 수 있는 모든 경우의 수를 생각하여 가장 높은 점수를 얻을 수 있는 블록배치를 찾아낸다. 그리고 그러한 블록 배치가 가능할 수 있도록 현재 블록의 배치를 추천해주는 것이다. 상황이 그 이전 블록에 의해 결정되는 게임이므로, 모든 경우의 수를 고려하기에는 트리 자료구조가 가장 적합하다. 현재 블록이 배치될 수 있는 모든 위치정보를 트리의 노드에 저장하고, 노드를 따라 내려가면서 다음 블록의 모든 위치정보를 또 다시 그 노드의 자식노드로 연결하며 경우의 수를 생각한다. 이러한 알고리즘은 앞으로 등장할 블럭의 개수를 몇개 고려하느냐에 따라 걸리는 시간과 사용하는 메모리공간의 차이가 급격히 난다. 그 이유는 루트 노드의 자식노드는 배치될 수 있는 모든 경우의 수만큼 생기므로 회전횟수 4번, 놓일 수 있는 x좌표가 최대 9이므로 블럭은 한개의 블럭은 최대 4\*9 = 36개의 위치에 놓일 수 있다. 그러나, 테트리스 게임의 특성상 블럭의 회전횟수가 4이면서 9군데에 놓일 수 있는 경우의 수는 없고, 최대 34개의 위치를 고려해야하는 상황이 생긴다. 첫번째 블럭 이후, 각각의 경우마다 최대 34개의 경우의 수가 추가로 매번 생기는 것이다. 따라서, 이는 고려해야하는 블럭의 개수를 VISIBLE\_BLOCKS이라고 한다면 시간복잡도는 O(34^VISIBLE\_BLOCKS)이 된다. 공간복잡도 또한 메모리해제를 해준다면 O(34^VISIBLE\_BLOCKS)가 된다.

이렇게 시간복잡도 및 공간복잡도가 큰 알고리즘은 n이 매우 작을때는 정상적으로 작동하지만 n이 조금이라도 커지면 계산과정이 기하급수적으로 많아져 속도가 매우 느려지고 메모리도 아주 많이 쓰게 된다. 이러한 문제점을 해결하기 위해 가장 점수를 얻을 수 있는 노드만을 남기고 나머지 노드들은 삭제하는 방법(pruning)을 선택하여 플레이한다. 이렇게 프로그램을 짠다면 시간복잡도는 기존의 방법과 크게 차이가 없지만 공간복잡도는 노드의 가지치기 덕분에 O(34\*VISIBLE\_BLOCKS) = O(VISIBLE\_BLOCKS)이 된다. 이렇게 되면 메모리 공간을 기존의 방법보다 훨씬 낮게 사용하게 되어 공간복잡도가 급격하게 상승하게 된다.

1. 모든 경우를 고려하는 트리와 비교하면 새롭게 구성한 pruning 트리는 공간복잡도 측면에서 향상된 모습을 보인다. 지수적인 공간복잡도를 가졌었던 프로그램을 선형 공간복잡도로 변환할 수 있는 알고리즘이다. 공간을 덜 사용하게 되면 결과적으로 시간이 덜 걸리는 효과또한 얻을 수 있다. 시간복잡도는 변함이 없지만, 컴퓨터는 많은 양의 메모리를 사용하고 있을수록 속도가 저하된다. 만약 메모리를 아주 많이 사용한다면 메모리가 적게 사용도는 코드에 비해 현저히 속도가 저하될 것이다. 그러나 프루닝 트리를 이용한다면 이에 대한 걱정은 덜 수 있다. 프루닝 트리는 공간복잡도 측면에서 우수한 알고리즘이지만, 최대의 점수를 내는것이 목표인 테트리스 게임에는 별로 적합하지 않은 방법일 수도 있다. 모든 경우의 수를 고려한다면 가장 높은 점수를 가지는 현재 블록의 위치를 정확히 알 수 있다. 만약 현재 블록만을 이용해서 점수 계산을 하였을 때 가장 높지 않아도, 그 이후의 블럭을 고려하였을 때 최대 점수를 얻을 수 있을 수도 있다. 그러나 실험자가 선택한 프루닝 알고리즘은, 언제나 최대 점수를 가지는 블럭만을 취하고 나머지는 모두 메모리 해제시켜주는 방법이다. 이는, 위에 언급했던 상황에서는 최선의 블록의 위치를 추천해주지 못할 수도 있다. 그렇기 때문에 새로운 알고리즘을 통해 플레이한 게임의 점수는 기존의 알고리즘의 점수보다 낮을 가능성이 더 크다. 그러나 저런 상황은 아주 빈번히 등장하지는 않고, 대부분의 경우 가장 높은 점수들의 집합이 가장 높은 점수의 결과를 만들어 내기 때문에 충분히 괜찮은 알고리즘이라고 생각한다.
2. 테트리스 프로젝트는 지금까지 어렴풋이 알고있던 코딩지식들을 실제 프로그램에 녹여보는 정말 좋은 경험이었다. 기본적인 이차원 배열을 탐색하기 위한 이중 for loop를 하던 중에는 for loop의 양끝 지점들을 굉장히 신경써서 프로그램을 작성해야한다는 것을 배우게 되었다. 이전까지는 크게 상관하지 않고 0부터 적당한 N까지 for 문을 돌렸는데, 이차원 배열의 값을 참조하거나 할당에서 index out of range 에러에 맞닥뜨리는 상황을 경험 한 지금은 for문을 돌 인덱스를 굉장히 신경쓰게 되었다. 그리고 다양한 자료구조에 대한 심도있는 이해를 할 수 있었다. Linked list 자료구조나 트리 자료구조는 컴퓨터공학 다른 과목시간에서 배워봤지만 실제로 어떻게 활용이 되고, 원하는 상황에서는 어떻게 구현이 되는지는 잘 알기가 힘들었다. 테트리스 프로젝트는 그러한 자료구조를 알 수 있는 아주 좋은 도구로서 이용하였다. 랭킹 시스템에서 linked list를 사용했기 때문에, linked list에서의 원소 추가 및 삭제 과정을 깔끔하게 정리할 수 있었다. 그리고 추천 시스템에서는 트리 자료구조를 사용했기 때문에, 여러개의 자식노드를 연결할때 배열 포인터를 이요해서 연결을 시켜줄 수 있다는 것을 알게 되었다. 이전에 트리는 웬만하면 이진 트리였기 때문에 왼쪽 자식과 오른쪽 자식을 연결시켜주는 두개의 link만 정의하여 문제를 해결하였는데, 테트리스 프로젝트에서는 자식노드가 최대 34개까지 나오는 것이 가능하므로, 이를 코드로 구현하는 것을 배울 수 있게 되었다. 그리고 추천 시스템에서 사용하는 알고리즘은 몇 년 전 알파고와 이세돌과의 바둑 경기를 떠올리게 하였다. 그 때 알파고에서는 모든 경우의 수를 고려하여 이세돌과 경기를 치뤘다고 했는데, 그런 동시에 AI와도 관련이 있다는 것을 들었다. 이러한 생각이 뻗어나가서 그 당시의 알파고가 사용했던 알고리즘이 무엇인지 더 공부해보고 싶은 마음이 생겼고, 더 나아가 인공지능으로 문제를 더 빠르고 효율적으로 해결할 수 있을 것 같아서 더 배워보고 싶은 욕구가 생겼다. 지금까지 해왔던 컴퓨터 실험 주제 중 가장 어려웠지만, 가장 배워가는 것이 많은 프로젝트였고, 이 프로젝트를 진행하면서 겪었던 많은 에러들은 이후의 나를 위한 큰 밑거름이 될 것 같다.