**컴퓨터학부 20182647 이형규**

**1. Alpha – Beta Pruning Algorithm 설명**

- Alpha – Beta Pruning Algorithm은 탐색 트리에서 MinMax Algorithm을 적용할 때 평가하는 노드의 수를 줄이기 위한 알고리즘이다.

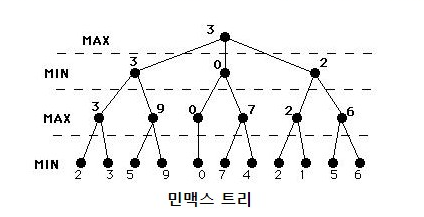
- Alpha – Beta Pruning Algorithm은 2인용 게임 ( ex) 오목, 체스, 틱택토, 과제로 나온 돌게임 등등)에 주로 사용된다.

- 이 알고리즘은 현재 평가중인 노드가 이전에 평가한 노드보다 가능성이 작아지면 평가를 중단하는 알고리즘이다. 현재 노드의 남은 형제와 자식들은 가지치기 되며 평가하지 않는다.

- 이 알고리즘은 MinMax Algorithm을 적용했을 때와 결과는 동일하지만 시간적인 측면에서 이득이 된다.

- 게임은 항상 서로 유리한 쪽으로 진행된다고 가정한다.

MinMax 예시 ( max = 나, min = 상대방 )



- 탐색 시작 전 트리에서 leaf 노드만 유의미한 값을 지닌다.

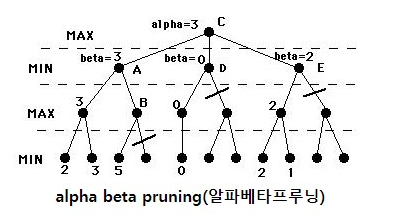
- min max 트리에서는 현재 단계가 max일때는 자식들 중 값이 큰 것, 현재 단계가 min일때는 자식들 중 작은 값을 택하면서 올라간다.

- min max의 경우 노드를 전부 탐색 하기 때문에 look ahead하는 횟수가 많아질수록 시간이 기하급수적으로 증가한다. 따라서 min max를 통해 승리하는 방법을 찾아다고 해도 시간이 올래걸리면 의미가 없다.

- 현재 단계에서 둘 수 있는 수가 최대 3개라고 가정하고 10수 앞을 내다본다고 가정하면 6만개의 가까운 노드를 탐색해야한다.

- 이때 사용하는 기법이 alpha – beta pruning이다.

Alpha – Beta Pruning 적용 예시 ( Max = 나 , Min = 상대방 )

****

**-** 탐색 시작전에는 alpha = -INF, beta = INF로 초기화 한다.

- alpha는 나에게 유리한정도, beta는 나에게 불리한 정도를 나타낸다

- 위 그림의 경우 가장 왼쪽 밑에서 2번째 max에서 3이라는 값을 얻어서 a노드에 돌려준다. 그 다음 B노드를 탐색하는데 B노드에서는 자식들 중 가장 큰 값을 선택할 것이다. B노드의 첫번째 자식을 5라고 가정하고 다음 자식을 확인하기 전에 판단을 한다. 만약 5보다 큰 자식이 나오면 B노드는 그 자식을 선택하지만 A노드 위치에서는 자식들 중 작은 값을 선택할 것이기 때문에 A는 3을 선택할 것이다. 만약 5보다 작은 자식이 나오면 B노드에서 나는 5를 선택하지만 A에서는 3을 선택하기 때문에 B에서 더 이상 자식을 탐색할 이유가 없기 때문에 가지치기 한다.

**2. 구현 자료구조 및 함수 설명**

**- 함수 설명**

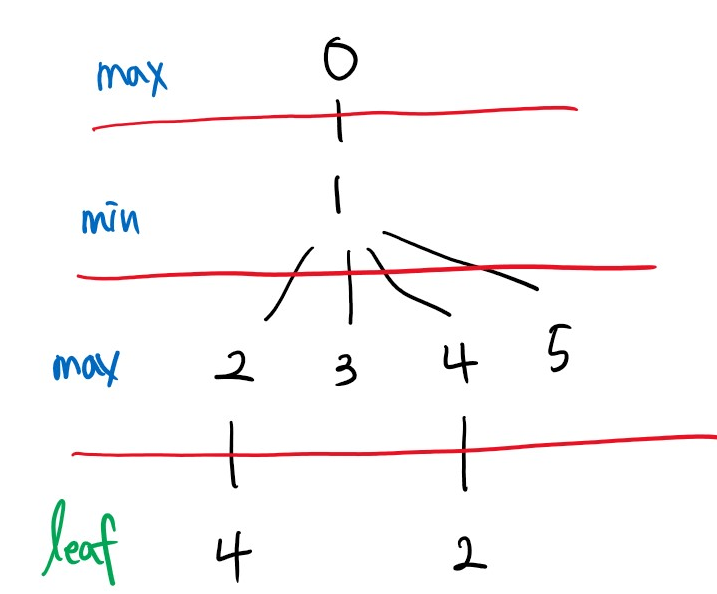
- void check\_prime() //1~n까지 소수인지 아닌지 판별 해주는 함수

- double evaluate(int last\_num, int flag) //flag를 통해 현재 플레이어가 누군지 입력받고 현재노드를 평가하는 함수

- double search(int depth, double alpha, double beta, int last\_num, int flag) //깊이, 알파, 베타, 마지막숫자, 현재플레이어 정보를 인자로 받아 최적의 수를 찾아주는 함수

**- 구현 자료구조**

**입력 : 5 0 일 경우 밑에 그림과 같은 트리 구조로 구현하였음**

****

**-** 현재 가져 갈 수 있는 돌을 체크한다음 자식으로서 노드를 추가한 뒤 이기는 사람이나 평가 함수를 이용하여 가지치기 하거나 리턴 시킴

**- 소스코드 ( 돌 게임 (min-max, alpha-beta x) )**

#include <iostream>

#include <vector>

#include <string>

#define MAX\_TURN 1

#define MIN\_TURN 0

#define INF 999999999

using namespace std;

int num\_of\_stones, num\_of\_taken;

double calculated\_value;

int node\_count = 0;

int max\_depth = 0;

int turn = MAX\_TURN;

vector<bool> prime;

vector<bool> check;

vector<int> max\_player;

vector<int> min\_player;

//소수 판별

void check\_prime()

{

prime[0] = true;

prime[1] = true;

for (int i = 2; i < prime.size(); i++)

{

for (int j = 2; i \* j < prime.size(); j++)

{

prime[i \* j] = true;

}

}

}

double search(int depth, double alpha, double beta, int last\_num, int flag)

{

vector<int> can\_take;

node\_count++;

//루트노드 일경우 가져갈 수 있는 돌 체크

if (last\_num != -1)

{

for (int i = 1; i <= num\_of\_stones; i++)

{

if (check[i] == false && (i % last\_num == 0 || last\_num % i == 0))

can\_take.push\_back(i);

}

}

//트리의 루트노드가 아닐 경우

else

{

for (int i = 1; i <= num\_of\_stones; i++)

{

if (((double)i < (double)num\_of\_stones / (double)2) && (i % 2 == 1)) can\_take.push\_back(i);

}

}

if (max\_depth < depth) max\_depth = depth;

//리프 노드일때 누가 이긴지 판별 후 리턴

if (can\_take.size() == 0)

{

if (flag == 0)

{

if (num\_of\_taken % 2 == 0) return -1.0;

else return 1.0;

}

else if (flag == 1)

{

if (num\_of\_taken % 2 == 0) return -1.0;

else return 1.0;

}

}

//내 차례일때

if (flag == 0)

{

double max\_score = -INF;

double tmp;

int index = can\_take[0];

for (int i = 0; i < can\_take.size(); i++)

{

//자식들중 가장 큰값을 선택

check[can\_take[i]] = true;

tmp = search(depth + 1, alpha, beta, can\_take[i], 1);

if (tmp > max\_score) max\_score = tmp;

check[can\_take[i]] = false;

}

if (depth == 0)

{

calculated\_value = max\_score;

return index;

}

return max\_score;

}

//상대 차례일때

else

{

double tmp;

double min\_score = INF;

for (int i = 0; i < can\_take.size(); i++)

{

//자식들중 가장 작은값 선택

check[can\_take[i]] = true;

tmp = search(depth + 1, alpha, beta, can\_take[i], 0);

if (tmp < min\_score) min\_score = tmp;

check[can\_take[i]] = false;

}

return min\_score;

}

}

int main(int argc, char\* argv[])

{

int last\_num = -1;

if (argc < 2)

{

fprintf(stderr, "input error\n");

exit(1);

}

//총 돌의 갯수

num\_of\_stones = atoi(argv[1]);

//가져간 돌의 갯수

num\_of\_taken = atoi(argv[2]);

check.resize(num\_of\_stones + 1);

prime.resize(num\_of\_stones + 1);

//1~num\_of\_stones사이에 숫자 중 소수 체크

check\_prime();

//가져간 돌들 각자 플레이어한테 저장

if (num\_of\_taken != 0)

{

for (int i = 3; i < argc; i++)

{

if (turn == MAX\_TURN)

{

max\_player.push\_back(atoi(argv[i]));

turn = MIN\_TURN;

}

else if (turn == MIN\_TURN)

{

min\_player.push\_back(atoi(argv[i]));

turn = MAX\_TURN;

}

check[atoi(argv[i])] = true;

}

//마지막돌 체크

if (turn == MAX\_TURN)

last\_num = min\_player[min\_player.size() - 1];

else if (turn == MIN\_TURN)

last\_num = max\_player[max\_player.size() - 1];

}

printf("Best move : %d\n", (int)search(0, -INF, INF, last\_num, 0));

printf("Calculated value : %.1f\n", calculated\_value);

printf("Number of visited Nodes : %d\n", node\_count);

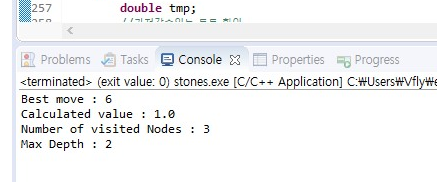
printf("Max Depth : %d\n", max\_depth);

}

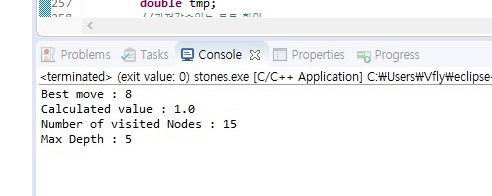
- **alpha-beta 가지치기를 적용한 소스코드는 프로젝트 파일 자체를 압축해 포함시켰습니다.**

**- 실행화면(alpha-beta적용)**

**입력 : 7 3 1 4 2**

****

**입력 ; 20 5 1 10 20 2 4**

****

**3. 참고문헌 및 사이트**

- alpha-beat algorithm 참고 사이트

<https://ko.wikipedia.org/wiki/%EC%95%8C%ED%8C%8C-%EB%B2%A0%ED%83%80_%EA%B0%80%EC%A7%80%EC%B9%98%EA%B8%B0> (코드 참고)

<http://egloos.zum.com/musicdiary/v/4274653> (방법 참고)

- minmax트리, alpha-beta적용시 그림 사이트

<https://m.blog.naver.com/PostView.nhn?blogId=zzabbo&logNo=70076811364&proxyReferer=https%3A%2F%2Fwww.google.com%2F>