|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| *voenmeh* | МИНОБРНАУКИ РОССИИ  Федеральное государственное бюджетное образовательное учреждение  высшего образования  **«Балтийский государственный технический университет «ВОЕНМЕХ» им. Д.Ф. Устинова»**  **(БГТУ «ВОЕНМЕХ» им. Д.Ф. Устинова»)** | | | | |
|  | | | | |
| Факультет | |  | И |  | Информационные и управляющие системы |
|  | |  | шифр |  | Наименование |
| Кафедра | |  | И9 |  | Систем управления и компьютерных технологий |
|  | |  | шифр |  | наименование |
| Дисциплина | |  | Представление знаний в информационнных системах | | |

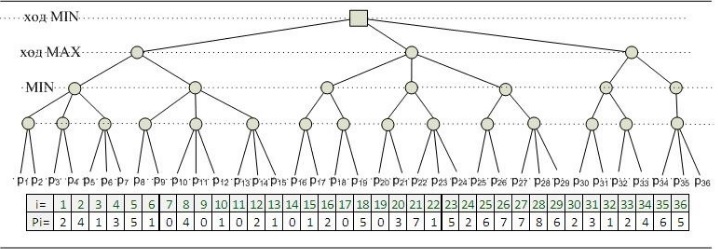
|  |
| --- |
| Лабораторная работа №3 |
| на тему «Поиск на игровых деревьях» |
| Вариант №3 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Выполнил студент группы | | | |  | И967 |
| Васильев Н.А. | | | | | |
| Фамилия И.О. | | | | | |
| **ПРЕПОДАВАТЕЛЬ** | | | | | |
| . | |  |  | | |
| Фамилия И.О. Подпись | | | | | |
| «\_\_\_\_\_» |  | | |  | 2019 г. |

САНКТ-ПЕТЕРБУРГ

2019 г.

**Задание:** Разработать программную реализацию минимаксной процедуры с отсечениями на следующем игровом графе:



Необходимо:

* получить возвращенные оценки неконцевых вершин методом минимакса;
* получить возвращенные оценки неконцевых вершин методом отсечений, показать отсекаемые ветви дерева (альфа и бета).

**Код программы на языке JavaScript:**

'use strict';

class Node {

constructor(value=0, id=0, level=0) {

this.parent = undefined;

this.childrens = [];

this.value = value;

this.level = level;

this.is\_terminal\_node;

this.id = id;

this.color = 0; //1 - alpha, 2 - beta, 0 - default;

}

add\_child(value=0, id=0, level=0, is\_terminal\_node=false) {

let child = new Node(value, id, level);

child.parent = this;

child.is\_terminal\_node = is\_terminal\_node;

this.childrens.push(child);

return child;

}

}

class Tree {

constructor() {

this.vis\_tree = null;

this.vis\_nodes = null;

this.vis\_edges = null;

this.vis\_tree\_options = {

nodes: { font: { size: 60 }, shape: 'circle'},

edges: { width: 15 },

layout: { hierarchical: { direction: "UD" } },

physics: false

};

this.original\_tree\_data = [];

this.root = new Node();

this.nodes = {0: this.root};

this.countID = 0;

let json\_data = [

{"0": [0, 1, false]}, {"0": [0, 1, false]}, {"0": [0, 1, false]},

{"1": [0, 2, false]}, {"1": [0, 2, false]},

{"2": [0, 2, false]}, {"2": [0, 2, false]}, {"2": [0, 2, false]},

{"3": [0, 2, false]}, {"3": [0, 2, false]},

{"4": [0, 3, false]}, {"4": [0, 3, false]}, {"4": [0, 3, false]},

{"5": [0, 3, false]}, {"5": [0, 3, false]}, {"5": [0, 3, false]},

{"6": [0, 3, false]}, {"6": [0, 3, false]},

{"7": [0, 3, false]}, {"7": [0, 3, false]},

{"8": [0, 3, false]}, {"8": [0, 3, false]},

{"9": [0, 3, false]}, {"9": [0, 3, false]},

{"10": [0, 3, false]},

{"11": [2, 4, true]}, {"11": [4, 4, true]},

{"12": [1, 4, true]}, {"12": [3, 4, true]},

{"13": [5, 4, true]}, {"13": [1, 4, true]}, {"13": [0, 4, true]},

{"14": [4, 4, true]}, {"14": [0, 4, true]},

{"15": [1, 4, true]}, {"15": [0, 4, true]}, {"15": [2, 4, true]},

{"16": [1, 4, true]}, {"16": [0, 4, true]}, {"16": [1, 4, true]},

{"17": [2, 4, true]}, {"17": [0, 4, true]},

{"18": [5, 4, true]}, {"18": [0, 4, true]},

{"19": [3, 4, true]}, {"19": [7, 4, true]},

{"20": [1, 4, true]}, {"20": [5, 4, true]}, {"20": [2, 4, true]},

{"21": [6, 4, true]}, {"21": [7, 4, true]},

{"22": [7, 4, true]}, {"22": [8, 4, true]}, {"22": [6, 4, true]},

{"23": [2, 4, true]}, {"23": [3, 4, true]},

{"24": [1, 4, true]}, {"24": [2, 4, true]},

{"25": [4, 4, true]}, {"25": [6, 4, true]}, {"25": [5, 4, true]},

];

for (let i = 0; i < json\_data.length; i++) {

let data\_chunk = Object.entries(json\_data[i])[0];

data\_chunk[0] = Number(data\_chunk[0]);

this.original\_tree\_data.push(data\_chunk);

}

}

tree\_reincarnated() {

this.root.childrens = [];

this.root.value = 0;

this.root.level = 0;

this.root.color = 0;

this.countID = 0;

this.nodes = {0: this.root};

this.vis\_tree = null;

this.vis\_nodes = new vis.DataSet();

this.vis\_edges = new vis.DataSet();

this.vis\_nodes.add({id: 0, label: ' 0', level: 0, color: '#e8e8e1', fixed: true});

for (let node\_data of this.original\_tree\_data) {

this.add\_node(node\_data[1][0], node\_data[0], node\_data[1][1], node\_data[1][2]);

}

}

add\_node(value, id, level, is\_terminal\_node=false) {

this.countID++;

this.nodes[this.countID] = this.nodes[id].add\_child(value, this.countID, level, is\_terminal\_node);

this.vis\_nodes.add({id: this.countID, label: ' ' + value.toString(), level: level, color: '#acc', fixed: true});

this.vis\_edges.add({from: id, to: this.countID, color: {color: '#ccc'}, fixed: true});

}

draw() {

let data = {nodes: this.vis\_nodes, edges: this.vis\_edges};

this.vis\_tree = new vis.Network(document.getElementById('tree\_map'), data, this.vis\_tree\_options);

}

tree\_changes\_applying(node) {

for (let child of node.childrens) {

if (child.color == 0) {

this.vis\_nodes.update({id: node.id, label: ' ' + node.value.toString(), level: node.level, color: '#acc', fixed: true});

this.vis\_nodes.update({id: child.id, label: ' ' + child.value.toString(), level: child.level, color: '#acc', fixed: true});

}

else if (child.color == 1) {

this.vis\_nodes.update({id: child.id, label: ' ' + child.value.toString(), level: child.level, color: 'orange', fixed: true});

this.vis\_edges.update({from: node.id, to: child.id, color: {color: 'orange'}, fixed: true});

}

else if (child.color == 2) {

this.vis\_nodes.update({id: child.id, label: ' ' + child.value.toString(), level: child.level, color: '#06b5cc', fixed: true});

this.vis\_edges.update({from: node.id, to: child.id, color: {color: '#06b5cc'}, fixed: true});

}

this.tree\_changes\_applying(child);

}

}

minimax(from\_node=undefined) {

function min\_move(from\_node) {

if (from\_node.is\_terminal\_node) return from\_node.value;

let min = +Infinity;

for (let child of from\_node.childrens) {

if (child.value < min && child.color == 0) {min = child.value;}

}

if (min == +Infinity) {return from\_node.value;}

else {return min;}

}

function max\_move(from\_node) {

if (from\_node.is\_terminal\_node) return from\_node.value;

let max = -Infinity;

for (let child of from\_node.childrens) {

if (child.value > max && child.color == 0) {max = child.value;}

}

if (max == -Infinity) {return from\_node.value;}

else {return max;}

}

if (from\_node == undefined) {from\_node = this.root;}

for (let child of from\_node.childrens) {

this.minimax(child)

}

//max-min

if (from\_node.level % 2) {from\_node.value = max\_move(from\_node);}

else {from\_node.value = min\_move(from\_node);}

}

minimax\_optimization(from\_node, alpha=-Infinity, beta=+Infinity) {

function prune(from\_node, child=undefined, is\_alpha\_pruning) {

let start;

child == undefined ? start = -1 : start = from\_node.childrens.indexOf(child);

for (let node of from\_node.childrens.slice(start+1)) {

prune(node, undefined, is\_alpha\_pruning);

if (is\_alpha\_pruning) node.color = 1;

else node.color = 2;

}

}

if (from\_node == undefined) from\_node = this.root;

//max-min

if (!(from\_node.level % 2)) {

if (from\_node.is\_terminal\_node) return from\_node.value;

let bestValue = +Infinity;

for (let child of from\_node.childrens) {

let value = this.minimax\_optimization(child, alpha, beta);

bestValue = Math.min(bestValue, value);

beta = Math.min(beta, bestValue);

from\_node.value = bestValue;

if (beta < alpha) {

prune(from\_node, child, true);

break;

}

}

return bestValue;

} else {

if (from\_node.is\_terminal\_node) return from\_node.value;

let bestValue = -Infinity;

for (let child of from\_node.childrens) {

let value = this.minimax\_optimization(child, alpha, beta);

bestValue = Math.max(bestValue, value);

alpha = Math.max(alpha, bestValue);

from\_node.value = bestValue;

if (beta < alpha) {

prune(from\_node, child, false);

break;

}

}

return bestValue;

}

}

}

var tree = new Tree();

function button\_minimax() {

tree.tree\_reincarnated();

tree.minimax();

tree.tree\_changes\_applying(tree.root);

tree.draw();

}

function button\_prunings() {

tree.tree\_reincarnated();

tree.minimax\_optimization();

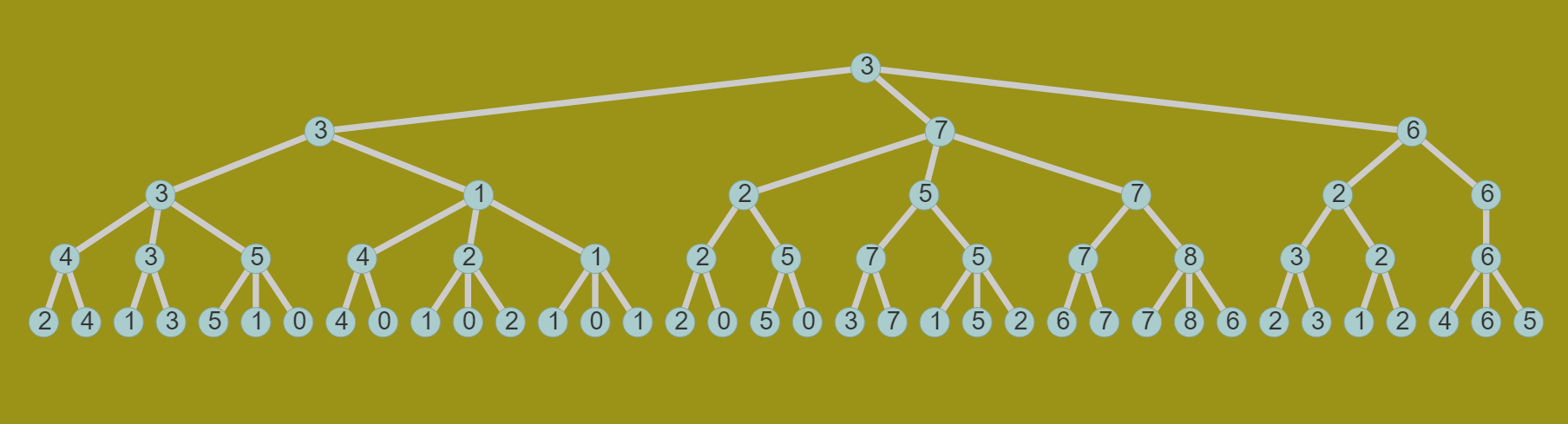
tree.tree\_changes\_applying(tree.root);

tree.draw();

}

button\_minimax();

**Результат работы метода Minimax:**



**Результат работы метода Альфа-Бета отсечений:**

