МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждения образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных технологии

Кафедра Информационных систем и технологий

Специальность 1-40 05 01 Информационные системы и технологии

**РЕФЕРАТ**

на тему:

«Дерево игр»

Выполнил студент Iкурса 3 группы специальности ИСиТ Задорожный Н.П

(Ф.И.О)

Руководитель

(учен. степень, звание, должность, подпись, Ф.И.О.)

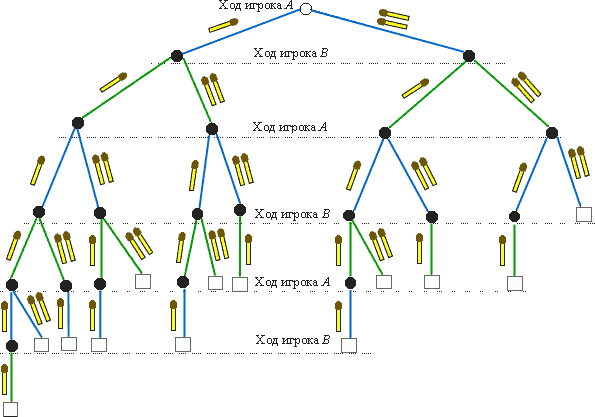
Минск 2019

Деревоигры — один из способов описания игры с помощью графа “[дерево](http://economic_mathematics.academic.ru/1353/%D0%B4%D0%B5%D1%80%D0%B5%D0%B2%D0%BE)”, последовательно по ходам фиксирующего, какой информацией располагают игроки перед каждым ходом, какие варианты они могут выбирать и какими могут быть результаты в конце игры.

Ветка из дерева игры – это фрагмент, часть дерева игры. Важно, что ветка дерева игры имеет одну корневую вершину – какую-то позицию игры и все возможные следующие позиции после этой корневой – до конца игры (до заключительных позиций). Таким образом, ветка дерева игры – это не любая часть дерева игры, а только такая, которая включает все возможные варианты завершения игры, начиная с некоторой позиции, т. е. в ветке нет "оборванных веток и листьев".

Реализация дерева игр.

Пример: Игра со спичками.

На столе лежат 6 спичек. 2 игрока по очереди берут по одной или по две спички. Тот, кто возьмет последним 1 или 2 спички, тот победил. Игру можно представить в виде следующего дерева игры:

Для каждого этапа игры используем структуру Node:

struct Node {

int Count; // Количество спичек

Node \*left = NULL; // Указатель на левый элемент

Node \*right = NULL; // Указатель на правый элемент

};

И создадим указатель на дерево:

Node \*Tree = new Node();

Для построения дерево игр используем функцию BuildTree:

void BuildTree(Node \*node) {

if (node->Count > 1) { // Если количество спичек больше чем 1

Node \*rnode = new Node(); // Добавляем правое звено

rnode->Count = node->Count - 2; // Для данного звена устанавливаем количество оставшихся спичек

node->right = rnode; // Настраиваем указатели

BuildTree(rnode); // Строим новые звенья для созданного звена

Node \*lnode = new Node(); // Та же процедура, но для левого поддерева

lnode->Count = node->Count - 1;

node->left = lnode;

BuildTree(lnode);

}

if (node->Count == 1) { // Если же количество спичек равно 1, то строим только левое поддерево

Node \*lnode = new Node();

lnode->Count = node->Count - 1;

node->left = lnode;

BuildTree(lnode);

}

}

Для вывода дерева игр используем функцию PrintTree:

void PrintTree(Node \*node,int l) {

// l - глубина на которой находиться элемент

if (node->left) // проверка на существование элемента

PrintTree(node->left, l+1);

for (int i = 0; i < l; i++)

cout << " ";

cout << node->Count << endl;

if (node->right) // проверка на существование элемента

PrintTree(node->right, l + 1);

}

Для поиска определенного этапа игры используем функцию Search

Node \*Buf = NULL;

void Search(Node \*node,int count) {

if (node->Count == count) // Если нашли звено с заданным количеством спичек, записываем его адрес в буфер

Buf = node;

if (node->left)

Search(node->left, count);

if (node->right)

Search(node->right, count);

}

Для построения дерева с данного этапа игры используем функцию

Current Pose и буфер, который указывает на ту или иную стадию игры :

void CurrentPose(int count) {

Search(Tree,count);//

Node \*node = Buf;

int l = 0; // Глубина дерева относительно данного этапа

if (node->left) // проверка на существование элемента

PrintTree(node->left, l + 1);

for (int i = 0; i < l; i++)

cout << " ";

cout << node->Count << endl;

if (node->right) // проверка на существование элемента

PrintTree(node->right, l + 1);

}

Функция main:

int main() {

setlocale(LC\_ALL, "Russian");

Tree->Count = 6;

cout << "Дерево относительно начала игры\n\n";

BuildTree(Tree);

PrintTree(Tree, 0);

cout << "Дерево относительно некоторой позиции\n\n";

CurrentPose(3);

system("pause");

return NULL;

}

