**Федеральное агентство образования**

**Санкт-Петербургский государственный электротехнический университет «ЛЭТИ»**

Факультет компьютерных технологий и информатики

Кафедра МО ЭВМ

Дисциплина: «Построение и анализ алгоритмов.»

**Отчет по лабораторной работе № 2**

Выполнил: студент группы 1381

Грейнер Андрей

Проверил: \_\_\_\_\_\_\_\_\_\_\_

**Санкт-Петербург**

**2013**

**Ход работы:**

1. **Содержательная постановка задачи**

Y пентамино. Найти наименьшее n, такое, что область размера 12\*5n можно покрыть элементами Y- пентамино вида

Найти все значения n для которых это возможно

1. **Анализ и пример решения задачи.**
   1. ***Анализ задачи***

Необходимо написать программу, которая будет использовать поиск с возвращением для поиска решения заполнения поля (размеры которого определяются пользователем) y-пентамино в разных положениях.

* 1. ***Особенности алгоритма***

Для тестирования программы было решено не фиксировать размеры поля, а позволить пользователю выбирать размеры самому (например m x k(n), где n =1) если программа не сможет найти решение, то n увеличивается на 1, следвательно ширина поля увеличивается на k .

1. **Формальная постановка задачи.**
   1. *Исходные данные:*

* Количество строк;
* Количество столбцов;
* Выбор режима визуализации;
  1. *Ограничения на исходные данные:*

-Чтобы количество клеток было кратно 5.

-Вводимые элементы – натуральные числа.

* 1. *Результирующие данные:*
* Время выполнения;
* Количество действий;

1. **КОД.**

**Figs.h**

//фигуры

int f[8][4][4] = {

{{2,0,0,0},//0

{2,2,0,0},

{2,0,0,0},

{2,0,0,0}},

{{2,2,2,2},//1

{0,0,2,0},

{0,0,0,0},

{0,0,0,0}},

{{0,2,0,0},//2

{2,2,0,0},

{0,2,0,0},

{0,2,0,0}},

{{0,2,0,0},//3

{2,2,2,2},

{0,0,0,0},

{0,0,0,0}},

{{2,0,0,0},//4

{2,0,0,0},

{2,2,0,0},

{2,0,0,0}},

{{2,2,2,2},//5

{0,2,0,0},

{0,0,0,0},

{0,0,0,0}},

{{0,2,0,0},//6

{0,2,0,0},

{2,2,0,0},

{0,2,0,0}},

{{0,0,2,0},//7

{2,2,2,2},

{0,0,0,0},

{0,0,0,0}}};

**Head.h**

#include <conio.h>

#include <iostream>

#include <windows.h>

#include "figs.h"

int chet=0,M=0,N=0;

bool boot,b=false,ololo=true;

int action=1, num=0, i\_st=0, j\_st=0, si=0, bomb=0;

int size[2] ={4,2};

int \*\*stack,\*\*shape,\*pos,\*\*field;

void pole1(int width,int height){

int ch=width\*height;

if(ch%5!=0){

ololo=false;

}

N=width+4;

M=height+4;

field=new int\*[N];

for(int i=0;i<N;i++){

field[i]=new int[M];

}

for(int i=0;i<N;i++){

for(int h=0;h<M;h++){

if(i>1 && i<N-2){

field[i][h]=1;

} else field[i][h]=0;

}

}

for(int i=0;i<N;i++){

for(int h=0;h<M;h++){

if(field[i][h]!=0){

if(h>1 && h<M-2){

field[i][h]=1;

} else field[i][h]=0;

}

}

}

chet=(width\*height)/5;

pos=new int[chet];

for(int i=0;i<chet;i++){

pos[i]=0;

}

shape=new int \*[chet];

for(int i=0;i<chet;i++){

shape[i]=new int[8];

}

for(int i=0;i<chet;i++){

for(int j=0;j<8;j++){

shape[i][j]=0;

}

}

stack=new int \*[chet];

for(int i=0;i<chet;i++){

stack[i]=new int[4];

}

for(int i=0;i<chet;i++){

for(int j=0;j<4;j++){

stack[i][j]=0;

}

}

}

void gotoxy(int xpos, int ypos)

{

COORD scrn;

HANDLE hOuput = GetStdHandle(STD\_OUTPUT\_HANDLE);

scrn.X = xpos; scrn.Y = ypos;

SetConsoleCursorPosition(hOuput,scrn);

}

void sp(){

gotoxy(0,20);

for(int i=0;i<N;i++){

for(int j=0;j<M;j++){

std::cout<<field[i][j];

}

std::cout<<"\n";

}

}

//установка фигуры на поле

int movef(int a[4][4],int n,int m,int i\_st,int j\_st){

int flag=1;

int i,j;

for(i=0;i<n&&flag;i++){

for(j=0;j<m&&flag;j++){

if(a[i][j])if(field[i\_st+i][j\_st+j]!=1)flag=0;

}

}

if(flag){

for(i=0;i<n;i++){

for(j=0;j<m;j++){

if(a[i][j])field[i\_st+i][j\_st+j]=a[i][j];

}

} return 1;

} else return 0;

}

//удаление фигуры с поля

void removef(int a[4][4],int n,int m,int i\_st,int j\_st){

for(int i=0;i<n;i++){

for(int j=0;j<m;j++){

if(a[i][j])field[i\_st+i][j\_st+j]=1;

}

}

}

//поиск опорной точки для установки фигуры

int move\_pointer(int \*i\_st,int \*j\_st){

int j;

for(j=\*j\_st;j<M && field[\*i\_st][j]!=1;j++);

if(j==M){

do {

if(\*i\_st<N-1){

(\*i\_st)++;

}

for(j=0;j<M && field[\*i\_st][j]!=1;j++);

} while(field[\*i\_st][j]!=1 && \*i\_st<N-1);

} \*j\_st=j;

if(\*i\_st==N-1 && \*j\_st==M){

return 0;

} else return 1;

}

//отступ от на начала матрицы 4 на 4

int find\_ind(int f[4][4]){

int j;

for(j=0;j<4 && !f[0][j];j++);

return j;

}

//подбор и установка нужной фигуры

int stand\_fig(int num){

int done;

do {

if(pos[num]==8){

pos[num]=0;

done=0;

break;

}

int n=pos[num]%2?size[1]:size[0];

int m=pos[num]%2?size[0]:size[1];

int d=find\_ind(f[pos[num]]);

done = movef(f[pos[num]],n,m,i\_st,j\_st-d);

if(shape[num][pos[num]]==1){

removef(f[pos[num]],n,m,i\_st,j\_st-d);

done=0;

}

pos[num]++;

} while(!done);

pos[num]--;

if(!done){

pos[num]=0;

} else {

shape[num][pos[num]]=1;

}

return done;

}

//проверка свободных клеток

void count\_sq(int i\_beg,int j\_beg,int \*c){

field[i\_beg][j\_beg]=3;

if((\*c)<4){

if(field[i\_beg][j\_beg+1]==1){

(\*c)++; count\_sq(i\_beg,j\_beg+1,c);

}

if((\*c)<4){

if(field[i\_beg+1][j\_beg]==1){

(\*c)++; count\_sq(i\_beg+1,j\_beg,c);

}

if((\*c)<4){

if(field[i\_beg][j\_beg-1]==1){

(\*c)++; count\_sq(i\_beg,j\_beg-1,c);

}

if(field[i\_beg-1][j\_beg]==1 &&(\*c)<4){

(\*c)++; count\_sq(i\_beg-1,j\_beg,c);

}

}

}

}

field[i\_beg][j\_beg]=1;

}

//проверка свободных клеток около фигуры

int check\_pos(int i\_beg,int j\_beg){

int c=0;

count\_sq(i\_beg,j\_beg,&c);

if(c<4)return 0; else return 1;

}

//вывод поля

void showf(){

for(int i=0;i<N;i++){

for(int j=0;j<M;j++){

if(field[i][j]==1) \_cprintf(".");

if(field[i][j]==0) \_cprintf(" ");

}

printf("\n");

}

}

//вывод удаления фигуры

void del\_fig(int num){

int d=find\_ind(f[pos[num]]);

for(int i=0;i<4;i++)

for(int j=0;j<4;j++)

if(f[pos[num]][i][j]==2){

gotoxy(j\_st+j-d,i\_st+i);

\_cprintf(".");

}

}

//вывод установки фигуры

void write\_fig(int num){

int d=find\_ind(f[pos[num]]);

for(int i=0;i<4;i++)

for(int j=0;j<4;j++)

if(f[pos[num]][i][j]==2){

gotoxy(j\_st+j-d,i\_st+i);

\_cprintf("x");

}

}

//запоминание установленных фигур

void push\_stack(int num){

stack[si][0]=num;

stack[si][1]=i\_st;

stack[si][2]=j\_st;

stack[si][3]=pos[num];

si++;

write\_fig(num);

}

//стирание установленных фигур

void pop\_stack(int \*num){

si--;

\*num=stack[si][0];

i\_st=stack[si][1];

j\_st=stack[si][2];

pos[\*num]=stack[si][3];

del\_fig(\*num);

}

//основная функция, бектрекинг

int backtracking(){

int res = 0;

int n,m,d,done,i=0,flag,action=1;

i\_st=0, j\_st=0;

if(ololo && chet!=0)

{

showf();

move\_pointer(&i\_st,&j\_st);

while (action){

do {

res++;

if(i<chet){

num=i;

done=stand\_fig(num);

if(done){

bomb=0;

push\_stack(num);

if(move\_pointer(&i\_st,&j\_st)){

flag=check\_pos(i\_st,j\_st);

} else {

gotoxy(1,20);

b=true;

printf("End.");

action=0;

}

} else flag=0;

}

if(boot){

\_getch();

}

i++;

} while (i!=chet && flag && action);

if((!flag ||(i==chet))&& action){

bomb++;

if(bomb>1){

for(int j=0;j<8;j++){

shape[num][j]=0;

}

}

if(num>0){

pop\_stack(&num);

n=pos[num]%2?size[1]:size[0];

m=pos[num]%2?size[0]:size[1];

d=find\_ind(f[pos[num]]);

removef(f[pos[num]],n,m,i\_st,j\_st-d);

pos[num]=0;

i=num;

} else {

system("cls");

std::cout<<"\nНевозможно собрать.\n";

action=0;

}

}

}

} else {

system("cls");

std::cout<<"\nНевозможно собрать.\n";

}

return res;

}

**Main.cpp**

#include <conio.h>

#include <iostream>

#include <windows.h>

#include <time.h>

#include "head.h"

void main(){

setlocale(LC\_ALL,"RUSSIAN");

double st,en,re;

bool tr=true;

int n=1;

char nmb;

int pole=5, stroka=10,resul;

while(tr)

{

std::cout<<"1-пошаговый режим, 2-автоматический:"<<'\n';

std::cin>>nmb;

std::cout<<"Введите размер поля"<<'\n';

std::cout<<"Количество строк = "<<'\n';

std::cin>> stroka;

std::cout<<"Количество столбцов = "<<'\n';

std::cin>>pole;

if(nmb=='1'){ boot=true;break;}

if(nmb=='2'){ boot=false;break;}

std::cin.clear();

}

system("cls");

while(!b){

st=0;en=0;re=0;

pole1(stroka,pole\*n);

st=clock();

resul = backtracking();

en=clock();

re=en-st;

delete []pos;

delete []stack;

delete []shape;

delete []field;

n++;

std::cout<<"Время : "<<re<<"\n";

std::cout<<"Действий : "<<resul<<"\n";

\_getch();

system("cls");

}

printf("\nPress some key to escape...\n");

\_getch() ;

}

1. **Вывод.**

В ходе лабораторной работе была создана программа, которая перебирает все возможные варианты расстановки у-пентамино в поле разного размера. Был освоен метод поиска с возвратом.