Министерство образования и науки Российской Федерации

Санкт-Петербургский государственный электротехнический университет «ЛЭТИ»

*Кафедра МО ЭВМ*

Дисциплина « Алгоритмы и структуры данных»

**Отчет по лабораторной работе №2**

Факультет: КТИ

Группа: 1381

Выполнил: Козицкий Д.Н.

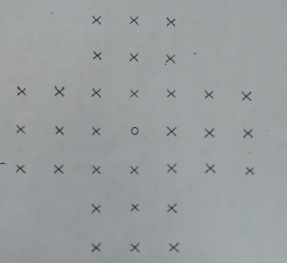
Проверил: Казаков Б.Б.

Санкт-Петербург

2013

**Задание:**

1d. Игра солитер. Тридцать две фишки расставляются, как показано ниже, на все клетки доски, кроме центральной. Фишка переносится на свободную клетку через любую соседнюю фишку, которая при этом снимается с доски (ходить можно только по вертикали и горизонтали).



Найдите последовательность «прыжков», такую, чтобы на доске осталась всего одна фишка в центральной клетке.

**Анализ задачи:**

На поле 33 клетки. Нумеруем их от 0 до 32 в любом порядке. После этого каждую позицию можно представить в виде 33-битного двоичного числа (1 в k-м разряде означает, что на k-м поле есть фишка). Создаём массив возможных ходов. Каждый ход описывается двумя масками: в первой — 2 ненулевых бита, указывающие, на каких полях должны были находиться фишки до хода, во второй — 1 бит, указывающий, где будет фишка после хода. Если первая маска равна U, вторая V, а текущая позиция F, то ход возможен, если выполняется условие (U&~F)==0 && (V&F)==0, а после хода мы окажемся в позиции (F&~U)|V.   
Для инициализации ходов берём карту поля (fld), просматриваем на ней все отрезки длины 3. Пусть отрезок состоит из клеток a,b,c. Если все клетки принадлежат игровому полю (номера неотрицательны), то такой отрезок даёт два хода. Для первого маска U=(1<<a)|(1<<b), V=1<<c, а для второго — U=(1<<b)|(1<<c), V=1<<a.  
Заводим массив, в котором каждой возможной позиции соответствует 1 бит. Позиций у нас 2^33, и такой массив займёт 1 гигабайт. Этот массив нам понадобится, чтобы помечать уже обработанные позиции.   
Выписываем маску start, соответствующую стартовой позиции, и начинаем решение. Решение рекурсивно, функции Solve передаётся единственный параметр — маска s позиции. Проверяем, текущая позиция является завершающейся. Это делается проверкой условия s == 0x000010000LL. Если условие срабатывает, возвращаем true — позиция имеет решение. Если текущая позиция не является последней, то перебираем все ходы, и для каждого смотрим, возможен ли он (проверяя условие (U&~s)==0 && (V&s)==0). Если ход возможен, вызываем Solve для позиции, возникшей после этого хода, и смотрим результат. Если true — мы нашли решение, сохраняем наш ход в стеке Solution, и сами возвращаем true. Если же ни один из ходов не привел к результату, значит, позиция тупиковая — возвращаем false.  
После того, как функция Solve вычислилась (и вернула true), в стеке Solution оказалась в точности та последовательность ходов, которая привела к решению.

**Формальная постановка задачи:**

На вход программе подаётся количество фишек на поле.

На выходе сообщение вида: «Решение найдено» или «Решение не найдено» и количество времени, потраченное на поиск решения.

**Описание функций:**

Функция generate создает случайное поле с n – фишками на нем.

Функции AddStep и InitSteps создают массив всевозможных ходов.

Функция Solve возвращает true, если решение существует и false в противном случае.

**Код программы:**

#include "stdafx.h"

#include<iostream>

#include<memory.h>

#include<time.h>

#include<conio.h>

#include<bitset>

#include<windows.h>

#include<fstream>

using namespace std;

int \*Arr; // 2^30 bytes, 2^28 ints

int L = 1 << 28;

long long start = 0x1fffeffffLL;

long long start2 = 0x1fffeffffLL;

int fld[49] = {-1,-1,0,1,2,-1,-1, -1,-1,3,4,5,-1,-1, 6,7,8,9,10,11,12, 13,14,15,16,17,18,19, 20,21,22,23,24,25,26, -1,-1,27,28,29,-1,-1, -1,-1,30,31,32,-1,-1};

int nsteps = 0;

long long Steps[76][2];

void AddStep(int a,int b,int c)

{

if( a < 0 || b < 0 || c < 0 ) return;

Steps[nsteps][0] = ( 1LL << a ) + ( 1LL << b );

Steps[nsteps][1] = ( 1LL << c );

Steps[nsteps+1][0] = ( 1LL << c ) + ( 1LL << b );

Steps[nsteps+1][1] = ( 1LL << a );

nsteps += 2;

}

void InitSteps()

{

for( int i = 0; i < 49; i++ )

{

if( ( i%7 ) < 5 ) AddStep( fld[i],fld[i+1],fld[i+2] );

if( i < 35 ) AddStep( fld[i],fld[i+7],fld[i+14] );

}

}

//Массив содержит последовательность ходов приводящих к выигрышу

int Solution[33];

int LS = 0;

long h = 0;

bool Solve( long long s )

{

Arr[s >> 5] = 1 ;

if( s == 0x000010000LL ) return true;

for( int k = 0; k < nsteps; k++ )

{

long long u = Steps[k][0],

v = Steps[k][1];

if( ( u & ~s ) == 0 && ( v & s ) == 0 )

{

long long s1 = ( s & ~u ) | v;

if( Arr[s1 >> 5] == 0 )

{

if( Solve( s1 ) )

{

Solution[LS++] = k;

return true;

}

}

}

}

return false;

}

long long generate( int n )

{

long long q = 0;

int mas[32] = {0},random;

for( int j = 1; j <= n; j++ )

{

random = rand()%32;

if( mas[random] == 0 ) mas[random] = 1; else j--;

}

for( int i = 0; i < 30; i++ ) q += ( 1 << i ) \* mas[i];

if( mas[31] == 1 ) q += 0x180000000LL;

return q;

}

int A[7][7] = {0};

void create\_A( long long t )

{

A[0][0] = A[0][1] = A[1][0] = A[1][1] = A[0][5] = A[0][6] = A[1][5] = A[1][6] =

A[5][0] = A[5][1] = A[6][0] = A[6][1] = A[5][5] = A[5][6] = A[6][5] = A[6][6] = -1;

if( ( t >> 32 ) == 1 ) A[0][2] = 1;

unsigned int dec = t;

int x = 0,y = 3;

for( int i = 0; i < 32; i++,dec <<= 1 )

{

if( i == 2 || i == 26 || i == 29 ){ y = 2; x++; }

else if( i == 5 || i == 12 || i == 19 ){ y = 0; x++; }

if( ( 48 + ( dec >> 31 ) ) == 49 ) A[x][y] = 1;

y++;

}

}

void print()

{

for( int i = 0; i < 7; i++ )

{

cout << i << "| ";

for( int j = 0; j < 7; j++ )

{

if( A[i][j] == 0 ) cout << "O ";

else if( A[i][j] == 1 ) cout << "X ";

else cout << " ";

}

cout << endl;

}

cout << " --------------------" << endl << " ";

for( int i = 0; i < 7; i++ )

cout << i << " ";

cout << endl;

}

void save( char file[] )

{

ofstream out( file );

for( int m = LS; --m >= 0; )

{

int k = Solution[m];

long long u = Steps[k][0],

v = Steps[k][1];

for( int i = 0; i < 33; i++ ) if( u & 1LL << i ) out << i << " " ;

for( int i = 0; i < 33; i++ ) if( v & 1LL << i ) out << " " << i << endl;

}

}

void animation( char file[] )

{

ifstream infile( file );

int x,y,z,q = 0;

print();

Sleep( 1500 );

system( "cls" );

for( int i = 0; i < 31; i++ )

{

infile >> x >> y;

infile >> z;

for( int j = 0; j < 7; j++ )

{

for( int k = 0; k < 7; k++ )

{

if( A[j][k] != -1 )

{

if( x == q || y == q ) A[j][k] = 0;

if( z == q ) A[j][k] = 1;

q++;

}

}

}

q = 0;

print();

Sleep( 1500 );

if( i != 30 )

{

system( "cls" );

}

}

}

void zeroing()

{

for( int i = 0; i < 7; i++ )

{

for( int j = 0; j < 7; j++ )

{

if( A[i][j] != -1 )

A[i][j] = 0;

}

}

}

void main()

{

setlocale(LC\_ALL, "rus");

srand((int)time(0));

InitSteps();

create\_A( start2 );

int menu;

while( true )

{

system( "cls" );

cout << "Выберети один из пунктов меню:\n ";

cout << "1. Исходная задача \n ";

cout << "2. Посмотреть решение\n ";

cout << "3. Метод Монте Карло\n\n";

cout << "Выш выбор: ";

cin >> menu;

system( "cls" );

if( menu == 1 )

{

create\_A( start2 );

cout << "Выделяем память под массив...\n";

Arr = new int[L];

memset( Arr,0,L\*4 );

cout << "Начинаем поиск решения...\n";

long c = clock();

Solve( start2 );

cout << "Решение найдено за " << (float)(clock()-c)/1000 << " секунд \n\n";

cout << "Сохранить решение или посмореть: ";

cin >> menu;

if( menu == 1 )

{

char file[256];

cout << "Введите имя файла: ";

cin >> file;

save( file );

cout << "Последовательность ходов была сохранена в файл " << file << "\n";

}

else

{

system( "cls" );

animation( "out.txt" );

}

LS = 0;

delete[] Arr;

system( "pause" );

}

else if( menu == 2 )

{

cout << "Введите имя файла с последовательностью: ";

char file[265];

cin >> file;

animation( file );

system( "pause" );

}

else if( menu == 3 )

{

cout << "Введите количество фишек: ";

int n;

cin >> n;

zeroing();

start = generate( n );

create\_A( start );

cout << "Сгенирировано следующие поле: \n";

print();

float a = n\*( ( 1 << ( n - 1 ) ) / 3758835 );

cout << "Время на решение текущего положения не превышает " << a << " секунд." << endl;

cout << "Выделяем память под массив...\n";

Arr = new int[L];

memset( Arr,0,L\*4 );

cout << "Начинаем поиск решения...\n";

long c = clock();

if( Solve( start ) )

{

cout << "Решение найдено за " << (float)(clock()-c)/1000 << " секунд \n\n";

cout << "Сохранить решение или посмореть: ";

cin >> menu;

if( menu == 1 )

{

char file[256];

cout << "Введите имя файла: ";

cin >> file;

save( file );

cout << "Последовательность ходов была сохранена в файл " << file << "\n";

}

else

{

system( "cls" );

animation( "out.txt" );

}

}

else

cout << "Решение не было найдено. Для того чтобы это осознать потребовалось " << (float)(clock()-c)/1000 << " секунд \n\n";

LS = 0;

delete[] Arr;

system( "pause" );

}

}

}

**Вывод:**

В данной лабораторной работе мы познакомились алгоритмом перебора решений «Backtracking».