**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра Математического обеспечения и применения ЭВМ**

отчет

**по лабораторной работе № 1**

**по дисциплине «Построение и Анализ Алгоритмов»**

Тема: Поиск с возвращением (backtracking)»

|  |  |  |
| --- | --- | --- |
| Студент гр. 6304 |  | Ковынев М.В. |
| Преподаватель |  | Балтрашевич В.Э. |

Санкт-Петербург

2018

# **Цель работы**

Используя поиск с возвращением (backtracking) решить индивидуальную задачу.

# **Постановка задачи**

У Томми есть много бумажных квадратиков. Длина из стороны (размер) изменяется от 1 до N – 1, и у него есть неограниченное число квадратов любого размера. Но ему хочется получить большой квадрат – размера N (см. рис. 1).

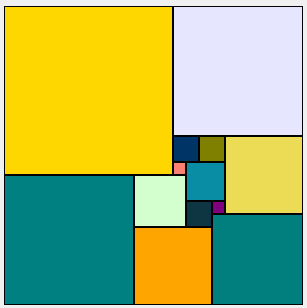
Он может получить такой квадрат, построив его из уже имеющихся квадратов. Например, квадрат размера 7 может быть построен из 9 меньших квадратов. Внутри квадрата не должно быть пустого места, меньшие квадраты не должны выходить за пределы большего и не должны перекрываться. Кроме того, Томми хочет истратить минимально возможное число квадратов

Рисунок 1. Пример решения задачи

# **Ход работы**

1. Реализуем функцию на основе бектрекинга, которая перебирает все возможные варианты расположение малых квадратов в большом квадрате и постепенно находит минимальное количество и расположение квадратов:

inline void **backtracking**(int x, int y, int sum, int S, vector<vector<RGBPIXEL>> &b)

{

color = 0;

if (sum + f[S] >= M)

return;

if (x<N)

{

int tot = 0;

for (int i = 1; i <= N; ++i)

if (a[x + 1][i])

i += r[x + 1][i];

else

{

while (i<N && !a[x + 1][i])

++i;

++tot;

}

if (sum + tot >= M)

return;

}

if (x>N)

{

M = sum;

memcpy(ans, tmp, sizeof(AS)\*M);

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

building(ans[i].x, ans[i].y, ans[i].len, b);

saveAsPPM(b, index);

q->setValue(index\*10);

index++;

return;

}

else if (y>N)

{

int i, top = 0, tot = 0, mid, s = 0, tmp;

++x;

if (x <= N)

{

for (i = 1; i <= N; ++i)

if (!a[x][i])

{

while (i < N && !a[x][i + 1])++i;

{

struct SS n;

n.y = i;

n.high = 0;

stack[top++] = n;

}

}

else

{

if (top&&stack[top - 1].high <= d[x][i])

{

mid = stack[--top].high;

if (top)

while (top&&stack[top - 1].high <= d[x][i])

{

--top;

if (i - stack[top].y - 1 <= stack[top].high-mid)

tmp=(stack[top].high - mid)/(i - tack[top].y-1);

else tmp = 0;

tot += tmp;

s += (stack[top].high - mid)\*(i - stack[top].y -1) -

tmp \* (i - stack[top].y - 1)\*(i - stack[top].y - 1);

mid = stack[top].high;

}

else

if (d[x][i] != mid)

{

if (i - 1 <= d[x][i] - mid)tmp = (d[x][i] - mid) / (i - 1);

else tmp = 0;

tot += tmp;

s += (d[x][i] - mid)\*(i - 1) - tmp \* (i - 1)\*(i - 1);

}

}

struct SS n;

n.y = i + r[x][i] - 1;

n.high = d[x][i];

stack[top++] = n;

i += r[x][i] - 1;

}

mid = stack[top - 1].high;

if (top)

{

while (--top)

s += (stack[top].y -stack[top - 1].y)\*(N -(x + stack[top].high-1));

s += stack[top].y\*(N - (x + stack[top].high - 1));

}

else

if (N != x + mid - 1)

tot += N / (N - (x + mid - 1));

if (sum + tot + f[s] >= M)

return;

}

backtracking(x, 1, sum, S,b);

}

else

if (a[x][y])

backtracking(x, y + r[x][y], sum, S, b);

else

{

int i = y;

while (i - y + 2<N && i<N && i<N - x + y && !a[x][i + 1])

++i;

int j, len = i - y + 1;

for (i = 0; i<len; ++i)

for (j = 0; j<len; ++j)

a[x + i][y + j] = 1;

for (; len; --len)

{

r[x + len][y] = 0;

for (i = 0; i<len; ++i)

r[x + i][y] = len;

for (i = 0; i <= len; ++i)

d[x + i][y] = len - i;

struct AS n;

n.len = len;

n.x = x;

n.y = y;

tmp[sum] = n;

backtracking(x, y + len, sum + 1, S - len \* len, b);

for (i = 0; i<len; ++i)

a[x + len - 1][y + i] = 0;

for (i = 0; i<len; ++i)

a[x + i][y + len - 1] = 0;

}

}

} {

QString fileName = QFileDialog::getOpenFileName(this, tr("Open File"), QString(),

tr("Text Files (\*.txt)"));

if (!fileName.isEmpty())

{

QFile file(fileName);

if (!file.*open*(QIODevice::ReadOnly))

{

QMessageBox::critical(this, tr("Error"), tr("Could not open file"));

return;

}

QTextStream in(&file);

ui->textEdit->setText(in.readAll());

file.*close*();

}

}



Рисунок 2. Интерфейс программы

# **Вывод**

В ходе выполнения данное лабораторной работы были изучены основные понятия и приёмы применения алгоритма поиска с возвращением (backtracking) для решения поставленной задачи.

# **ПРИЛОЖЕНИЕ А.**

# **ИХОДНЫЙ КОД MAIN.CPP**

#include "mainwindow.h"

#include <QApplication>

#include <QLabel>

#include <QLayout>

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

{

QApplication a(argc, argv);

MainWindow w;

w.show();

w.setFixedSize(190,390);

w.setWindowIcon(QIcon("D:\\2.png"));

w.setWindowTitle("Лабораторная работа №1");

return a.exec();

}

# **ПРИЛОЖЕНИЕ Б.**

# **ИХОДНЫЙ КОД MAINWINDOW.CPP**

#include "mainwindow.h"

#include "ui\_mainwindow.h"

#include <QMessageBox>

#include "solver.h"

MainWindow::**MainWindow**(QWidget \*parent) :

QMainWindow(parent),

ui(new Ui::MainWindow)

{

ui->setupUi(this);

ui->progressBar->setValue(0);

}

MainWindow::~***MainWindow***()

{

delete ui;

}

int colorist = 1;

int counter = 1;

int maxindex = 1;

int minindex = 0;

void MainWindow::**on\_pushButton\_clicked**()

{

remove("output\_0.ppm");

remove("output\_1.ppm");

remove("output\_2.ppm");

remove("output\_3.ppm");

remove("output\_4.ppm");

remove("output\_5.ppm");

remove("output\_6.ppm");

int value = ui->textEdit->toPlainText().toInt();

ui->warning->setText("Происходит генерация решения.\n

Пожалуйста, подождите.");

if (value <= 42 && value >=1 )

{

ui->progressBar->setValue(0);

maxindex = mainsolver(ui->textEdit->toPlainText().toInt(), colorist,

ui->progressBar, ui->sol);

counter = 1;

QPixmap image("output\_0.ppm");

ui->label->clear();

ui->label->setPixmap(image);

QPixmap image1("output\_1.ppm");

ui->step->setText("Шаг 1");

ui->Current->clear();

ui->Current->setPixmap(image1);

window()->setFixedSize(1150,390);

ui->warning->setText("Генерация завершена.\nУспех!");

}

else

{

ui->label->clear();

ui->Current->clear();

ui->progressBar->setValue(0);

ui->warning->setText("Невозможно решить задачу.\nВведите меньшее число");

ui->step->setText("Шаг 1");

window()->setFixedSize(190,390);

}

}

void MainWindow::**on\_pushButton\_next\_clicked**()

{

if (maxindex > counter +1 )

{

counter++;

char fileNameIndex[3];

sprintf(fileNameIndex, "%d", counter);

char fileName[30] = "output\_";

strcat(fileName, fileNameIndex);

strcat(fileName, ".ppm");

QPixmap image(fileName);

ui->Current->setPixmap(image);

ui->step->setText("Шаг "+QString::number(counter));

}

}

void MainWindow::**on\_pushButton\_prev\_clicked**()

{

if (minindex < counter -1)

{

counter--;

char fileNameIndex[3];

sprintf(fileNameIndex, "%d", counter);

char fileName[30] = "output

strcat(fileName, fileNameIndex;

strcat(fileName, ".ppm");

QPixmap image(fileName);

ui->Current->setPixmap(image);

ui->step->setText("Шаг "+QString::number(counter));

}

}

void MainWindow::**on\_radioButton\_clicked**()

{

if (colorist == 1)

colorist = 0;

else

colorist = 1;

}

void MainWindow::**on\_pushButton\_4\_clicked**()

{

exit(0);

}

void MainWindow::**on\_pushButton\_2\_clicked**()

{

QMessageBox::about(this, tr("Справка"), tr("Задача\n\nУ Томми есть много бумажных квадратиков. Длина из стороны (размер) изменяется от 1 до N – 1, и у него есть неограниченное число квадратов любого размера. Но ему очень хочется получить большой квадрат – размера N.\n\nОн может получить такой квадрат, построив его из уже имеющихся квадратов. Например, квадрат размера 7 может быть построен из 9 меньших квадратов. Внутри квадрата не должно быть пустого места, меньшие квадраты не должны выходить за пределы большего и не должны перекрываться. Кроме того Томми хочет истратить минимально возможное число квадратов."));

return;

}

void MainWindow::**on\_pushButton\_3\_clicked**()

{

QMessageBox::about(this, tr("О программе"), tr("Выполнил: Ковынев М.В.\nГруппа: 6304"));

return;

}

# **ПРИЛОЖЕНИЕ В.**

# **ИХОДНЫЙ КОД MAINWINDOW.H**

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QMainWindow>

namespace Ui {

class MainWindow;

}

class MainWindow : public QMainWindow

{

Q\_OBJECT

public:

explicit MainWindow(QWidget \*parent = 0);

~***MainWindow***();

private slots:

void **on\_pushButton\_clicked**();

void **on\_pushButton\_next\_clicked**();

void **on\_pushButton\_prev\_clicked**();

void **on\_radioButton\_clicked**();

void **on\_pushButton\_4\_clicked**();

void **on\_pushButton\_2\_clicked**();

void **on\_pushButton\_3\_clicked**();

private:

Ui::MainWindow \*ui;

};

#endif // MAINWINDOW\_H

# **ПРИЛОЖЕНИЕ Г.**

# **ИХОДНЫЙ КОД SOLVER.CPP**

#include "solver.h"

#include<cstdio>

#include<cstring>

#include<cstdlib>

#include<cmath>

#include<algorithm>

#include<iostream>

#include<ctime>

#include <vector>

int width = 300;

int M = 20;

int factor = 0;

int index = 1;

int color = 0;

int f[2505];

int N;

bool a[55][55];

int r[55][55], d[55][55];

int changecolor = 1;

QProgressBar \*q;

QLabel \*qq;

RGBPIXEL **setcolor**(int n)

{

RGBPIXEL currentColor;

if (changecolor == 0)

{

currentColor.rgbBlue = 255;

currentColor.rgbGreen = 255;

currentColor.rgbRed = 255;

return currentColor;

}

else

{

switch (n)

{

case 0: {currentColor.rgbRed = 255; currentColor.rgbGreen = 215; currentColor.rgbBlue = 0; return (currentColor); break; }

case 1: {currentColor.rgbRed = 0; currentColor.rgbGreen = 128; currentColor.rgbBlue = 128; return (currentColor); break; }

case 2: {currentColor.rgbRed = 211; currentColor.rgbGreen = 255; currentColor.rgbBlue = 206; return (currentColor); break; }

case 3: {currentColor.rgbRed = 255; currentColor.rgbGreen = 165; currentColor.rgbBlue = 0; return (currentColor); break; }

case 4: {currentColor.rgbRed = 230; currentColor.rgbGreen = 230; currentColor.rgbBlue = 255; return (currentColor); break; }

case 5: {currentColor.rgbRed = 0; currentColor.rgbGreen = 51; currentColor.rgbBlue = 102; return (currentColor); break; }

case 6: {currentColor.rgbRed = 250; currentColor.rgbGreen = 128; currentColor.rgbBlue = 114; return (currentColor); break; }

case 7: {currentColor.rgbRed = 8; currentColor.rgbGreen = 141; currentColor.rgbBlue = 165; return (currentColor); break; }

case 8: {currentColor.rgbRed = 13; currentColor.rgbGreen = 54; currentColor.rgbBlue = 66; return (currentColor); break; }

case 9: {currentColor.rgbRed = 127; currentColor.rgbGreen = 127; currentColor.rgbBlue = 0; return (currentColor); break; }

case 10: {currentColor.rgbRed = 127; currentColor.rgbGreen = 0; currentColor.rgbBlue = 127; return (currentColor); break; }

case 11: {currentColor.rgbRed = 0; currentColor.rgbGreen = 127; currentColor.rgbBlue = 127; return (currentColor); break; }

case 12: {currentColor.rgbRed = 236; currentColor.rgbGreen = 219; currentColor.rgbBlue = 84; return (currentColor); break; }

case 13: {currentColor.rgbRed = 233; currentColor.rgbGreen = 75; currentColor.rgbBlue = 60; return (currentColor); break; }

case 14: {currentColor.rgbRed = 111; currentColor.rgbGreen = 159; currentColor.rgbBlue = 216; return (currentColor); break; }

case 15: {currentColor.rgbRed = 0; currentColor.rgbGreen = 165; currentColor.rgbBlue = 145; return (currentColor); break; }

case 16: {currentColor.rgbRed = 234; currentColor.rgbGreen = 222; currentColor.rgbBlue = 219; return (currentColor); break; }

case 17: {currentColor.rgbRed = 191; currentColor.rgbGreen = 214; currentColor.rgbBlue = 65; return (currentColor); break; }

default: {currentColor.rgbRed = rand() % 255; currentColor.rgbGreen = rand() % 255; currentColor.rgbBlue = rand() % 255; return (currentColor); break; }

}

}

}

void **addColorToMatrix**(squareCoord curCoord, vector<vector<RGBPIXEL>> &a)

{

RGBPIXEL n = setcolor(color);

for (int i = factor\*(curCoord.x - 1); i < factor \* (curCoord.size + curCoord.x - 1); i++)

{

for (int j = factor\*(curCoord.y - 1); j < factor \* (curCoord.size + curCoord.y - 1); j++)

{

a[i][j] = n;

}

}

RGBPIXEL nn;

nn.rgbBlue = 0;

nn.rgbGreen = 0;

nn.rgbRed = 0;

for (int i = factor \* (curCoord.x - 1); i < factor \* (curCoord.size + curCoord.x - 1); i++)

{

a[i][factor \* (curCoord.y - 1)] = nn;

a[i][factor \* (curCoord.size + curCoord.y - 1)-1] = nn;

}

for (int j = factor \* (curCoord.y - 1); j < factor \* (curCoord.size + curCoord.y - 1); j++)

{

a[factor \* (curCoord.x - 1)][j] = nn;

a[factor \* (curCoord.size + curCoord.x - 1)-1][j] = nn;

}

color++;

}

void **building**(int x, int y, int size, vector<vector<RGBPIXEL>> &a)

{

squareCoord temp;

temp.x = x;

temp.y = y;

temp.size = size;

addColorToMatrix(temp, a);

}

void **saveAsPPM**(vector<vector<RGBPIXEL>> &b, int i)

{

char fileNameIndex[3];

sprintf(fileNameIndex, "%d", i);

char fileName[30] = "./output\_";

strcat(fileName, fileNameIndex);

strcat(fileName, ".ppm");

FILE \*f = fopen(fileName, "wb");

fprintf(f, "P6\n%i %i 255\n", N\*factor, N\*factor);

for (int y = 0; y<N\*factor; y++) {

for (int x = 0; x<N\*factor; x++) {

fputc(b[x][y].rgbRed, f); // 0 .. 255

fputc(b[x][y].rgbGreen, f); // 0 .. 255

fputc(b[x][y].rgbBlue, f); // 0 .. 255

}

}

fclose(f);

}

struct AS

{

int len;

int x, y;

} tmp[2505], ans[2505];

struct SS

{

int y, high;

} stack[55];

inline void **backtracking**(int x, int y, int sum, int S, vector<vector<RGBPIXEL>> &b)

{

color = 0;

if (sum + f[S] >= M)

return;

if (x<N)

{

int tot = 0;

for (int i = 1; i <= N; ++i)

if (a[x + 1][i])

i += r[x + 1][i];

else

{

while (i<N && !a[x + 1][i])

++i;

++tot;

}

if (sum + tot >= M)

return;

}

if (x>N)

{

M = sum;

memcpy(ans, tmp, sizeof(AS)\*M);

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

building(ans[i].x, ans[i].y, ans[i].len, b);

saveAsPPM(b, index);

q->setValue(index\*10);

index++;

return;

}

else if (y>N)

{

int i, top = 0, tot = 0, mid, s = 0, tmp;

++x;

if (x <= N)

{

for (i = 1; i <= N; ++i)

if (!a[x][i])

{

while (i < N && !a[x][i + 1])++i;

{

struct SS n;

n.y = i;

n.high = 0;

stack[top++] = n;

}

}

else

{

if (top&&stack[top - 1].high <= d[x][i])

{

mid = stack[--top].high;

if (top)

while (top&&stack[top - 1].high <= d[x][i])

{

--top;

if (i - stack[top].y - 1 <= stack[top].high - mid)

tmp = (stack[top].high - mid) / (i - stack[top].y - 1);

else tmp = 0;

tot += tmp;

s += (stack[top].high - mid)\*(i - stack[top].y - 1) -

tmp \* (i - stack[top].y - 1)\*(i - stack[top].y - 1);

mid = stack[top].high;

}

else

if (d[x][i] != mid)

{

if (i - 1 <= d[x][i] - mid)tmp = (d[x][i] - mid) / (i - 1);

else tmp = 0;

tot += tmp;

s += (d[x][i] - mid)\*(i - 1) - tmp \* (i - 1)\*(i - 1);

}

}

struct SS n;

n.y = i + r[x][i] - 1;

n.high = d[x][i];

stack[top++] = n;

i += r[x][i] - 1;

}

mid = stack[top - 1].high;

if (top)

{

while (--top)

s += (stack[top].y - stack[top - 1].y)\*(N - (x + stack[top].high - 1));

s += stack[top].y\*(N - (x + stack[top].high - 1));

}

else

if (N != x + mid - 1)

tot += N / (N - (x + mid - 1));

if (sum + tot + f[s] >= M)

return;

}

backtracking(x, 1, sum, S,b);

}

else

if (a[x][y])

backtracking(x, y + r[x][y], sum, S, b);

else

{

int i = y;

while (i - y + 2<N && i<N && i<N - x + y && !a[x][i + 1])

++i;

int j, len = i - y + 1;

for (i = 0; i<len; ++i)

for (j = 0; j<len; ++j)

a[x + i][y + j] = 1;

for (; len; --len)

{

r[x + len][y] = 0;

for (i = 0; i<len; ++i)

r[x + i][y] = len;

for (i = 0; i <= len; ++i)

d[x + i][y] = len - i;

struct AS n;

n.len = len;

n.x = x;

n.y = y;

tmp[sum] = n;

backtracking(x, y + len, sum + 1, S - len \* len, b);

for (i = 0; i<len; ++i)

a[x + len - 1][y + i] = 0;

for (i = 0; i<len; ++i)

a[x + i][y + len - 1] = 0;

}

}

}

int **mainsolver**(int t, int changecolore, QProgressBar \*rr, QLabel \*rrr)

{

q=rr;

qq=rrr;

q->setValue(0);

changecolor = changecolore;

memset(f, 127, sizeof(f));

f[0] = 0;

for (int i = 1; i <= 2500; ++i)

{

for (int j = 1; j\*j <= i; ++j)

f[i] = min(f[i], f[i - j \* j]);

++f[i];

}

M = 20;

N = 0;

color = 0;

index = 1;

N = t;

factor = (int)(width / t);

vector<vector<RGBPIXEL>> b(N\*factor, vector<RGBPIXEL>(N\*factor));

memset(a, 0, sizeof(a));

backtracking(1, 1, 0, N\*N, b);

qq->setText("Минимальное число\nквадратов: " + QString::number(M) + "\n\n");

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

{

building(ans[i].x, ans[i].y, ans[i].len, b);

qq->setText(qq->text() + QString::number(i+1) + ". Вершина: (" + QString::number(ans[i].x) + "," + QString::number(ans[i].y) + ") Длина: "+ QString::number(ans[i].len) + "\n\n");

}

saveAsPPM(b, 0);

b.clear();

q->setValue(100);

return index;

}

# **ПРИЛОЖЕНИЕ Г.**

# **ИХОДНЫЙ КОД SOLVER.H**

#include <iostream>

#include <vector>

#include <QProgressBar>

#include <QLabel>

using namespace std;

typedef struct

{

unsigned char rgbBlue;

unsigned char rgbGreen;

unsigned char rgbRed;

} RGBPIXEL;

typedef struct

{

int x;

int y;

int size;

} squareCoord;

RGBPIXEL **setcolor**(int n);

void **addColorToMatrix**(squareCoord, vector<vector<RGBPIXEL>> &a, int);

void **building**(int, int, int, vector<vector<RGBPIXEL>> &a);

void **saveAsPPM**(vector<vector<RGBPIXEL>> &b, int i);

inline void **backtracking**(int, int, int, int, vector<vector<RGBPIXEL>> &b);

int **mainsolver**(int , int, QProgressBar\*, QLabel\*);