**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

отчет

**по лабораторной работе №1**

**по дисциплине «Построение и анализ алгоритмов»**

Тема: Проблема оптимального выбора

|  |  |  |
| --- | --- | --- |
| Студент гр. 6304 |  | Корытов П.В. |
| Преподаватель |  | Балтрашевич В.Э. |

Санкт-Петербург

2018

**Цель работы.**

Изучить реализацию алгоритма оптимального выбора на языке программирования C++. Реализовать возможность пошагового выполнения алгоритма с отслеживаем бэктрекинга.

**Основные теоретические положения.**

Задача оптимального выбора заключается в поиске оптимального решения данной задачи. Чтобы это сделать, необходимо получить все возможные решения и в процессе получения оставлять оптимальное. Если оптимальность определена с помощью функции то проверка решения на оптимальность выглядит следующим образом:

If (f(solution) > f(optimum)) optimum = solution;

В переменной optimum хранится лучшее решение, а в переменной solution – текущее.

Часто решается задача поиска оптимальной выборки из заданного множества объектов, подчинённой некоторым ограничениям. В этом случае выборки строятся постепенно и каждый объект проверяется на пригодность включения в выборку рекурсивной процедурой, в данной работе – try.

При рассмотрении отдельного объекта его можно либо включить, либо не включить в выборку. Очевидно, что при полном переборе сложность алгоритма будет что есть плохо. Значит, необходимо использовать критерии приемлемости для объектов. На псевдокоде эта схема выглядит так:

void try(int i){

if [включение приемлемо]{

[включить i-й объект]

if (i < n)

try(i+1);

else

[проверить оптимальность]

[удалить i-й объект]

}

if [невключение приемлемо]{

if (i < n)

try(i+1)

else

[проверить оптимальность]

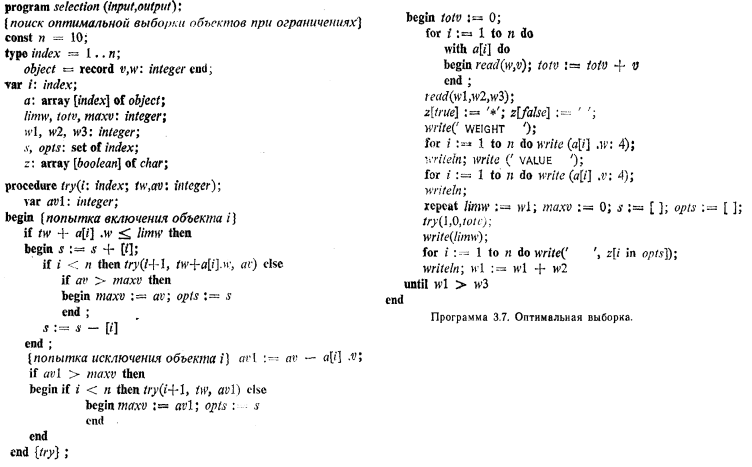
}

}

В данной работе исследуется конкретный пример – у объектов есть вес (weight) и ценность (value). Нужно упаковать их в контейнер с ограниченным максимальным весом так, чтобы суммарная ценность была максимальна.

В таком случае критерий приемлемости включения – удовлетворение объекта необходимому весу, а критерий приемлемости невключения – возможность получить без этого объекта такую общую ценность, которая была бы не меньше полученное до этого оптимальной.

Реализация данного алгоритма у Вирта на языке Паскаль:



**Ход работы**

1. Произведен анализ алгоритма по данным книги Вирта. Сделан вывод, что выборку элементов удобнее хранить в стеке. Алгоритм изменен, чтобы обеспечить возможность пошаговой обработки.
2. Составлен класс, хранящий один элемент выборки.

class Item{

public:

QColor **getcol**();

void **set**(int w, int v);

void **set**(QString w, QString v, bool rnd = 0);

void **set**(Item it);

int **get\_weight**(){ return weight; }

int **get\_value**() { return value; }

bool **ok**() {return ((weight!=0) && (value!=0)); }

QString **print**();

QListWidgetItem\* wItem;

private:

int redcol = 0; int greencol = 0; int bluecol = 0;

void **setItem**(bool randcol=1);

int weight = 0;

int value = 0;

};

Класс содержит набор сеттеров и геттеров и метод проверки инварианта класса. Файлы item.h и item.cpp находятся в приложениях А и Б

1. После этого составлен класс, решающий задачу. Файлы opt\_task.h и opt\_task.cpp находятся в приложениях В и Г

class Opt\_task{

public:

Opt\_task(int nt); //Конструктор

Opt\_task() = default;

QList<Item> all; //Стек из всех предметов

QStack<Item> s;//Текущий вариант

QStack<Item> opts;//Оптимальный вариант

QStack<try\_parm> series;//Стек запусков функции

int **totv\_all**(); //Суммарная ценность массива

void **try\_item**(); //Инициализация

void **try\_item**(int i, int tw, int av); //Рекурсивное решение

void **try\_item**(int i, int tw, int av, bool steps); //Вариант для пошаговой обработки

void **try\_item**(try\_parm parm);

void **setlimw**(int i);

void **setlimw**(QString i);

void **clrzeros**(); //Очистка от мусора

int **g\_totw**(QStack<Item> s); //Суммарный вес сета

int **g\_totv**(QStack<Item> s); //Суммарная ценность сета

int **g\_nums**() {return nums; } //Вывод количества решений

void **out**(QStack<Item> s); //Вывод сета

void **make\_arr**();

bool steps = 0;

bool go = 1;

private:

void **try\_incl**(int i, int tw, int av);

void **try\_unincl**(int i, int tw, int av);

int n;

int limw = 0; //Максимальный вес

int totw = 0; //Вес текущей выборки

int maxv = 0; //Максимальная ценость оптимальной выборки

int nums = 0; //Количество решений

};

1. Исходя из пункта 3, составлен класс для хранения параметров запуска одной итерации:

class try\_parm{

public:

try\_parm(int it, int twt, int avt){i=it; tw=twt; av=avt; }

try\_parm() = default;

int i;

int tw;

int av;

QStack<Item> s;//Вариант для запуска

};

Работа алгоритма реализована следующим образом:

1. Первый запуск всегда void Opt\_task::**try\_item**(). Он инициализирует необходимые параметры, очищает некоторые объекты в куче из запускает void Opt\_task::**try\_item**(int i, int tw, int av)
2. Запущенная процедура увеличивает счётчики, вытаскивает нужный элемент для обработки и делает проверку на способ выполнения. За способ отвечает переменная steps – при значении true выполнение пошаговое, при значении false – обычное
   1. Если steps == false, то выполнение фактически происходит аналогично описанному алгоритму у Вирта.
   2. Пошаговая работа означает, что данный запуск единственен и вызываемые из Opt\_task::**try\_unincl** и Opt\_task::**try\_incl** методы void Opt\_task::**try\_item**(int i, int tw, int av, bool steps) не начнут рекурсивное выполнение, но запишут задания для проверки в стек из try\_parm.

При рекурсивном выполнении сначала выполняются попытки включения элемента, когда это более невозможно – попытки исключения. Чтобы это обеспечить в данном случае, на вершине стека заданий должны лежать задания включения элементов. Поэтому метод Opt\_task::**try\_unincl**(int i, int tw, int av), делающий попытку выключения, должен запускаться вначале – тогда производные от него будут запущены в конце.

Это аналогично ЛКП-обходу дерева решений – рекурсивно он выглядит как (Посетить Левое, Посетить Корень, Посетить Правое), но для пошагового выполнения через стек необходимо сначала Посетить Правое, чтобы его задания записались последними.

* 1. Уже упомянутый метод Opt\_task::**try\_item**(int i, int tw, int av, bool steps) запускается из методов Opt\_task::**try\_unincl** и Opt\_task::**try\_incl.** Он также проверяет, пошаговое ли выполнение – если да, то просто запускает метод метод Opt\_task::**try\_item**(int i, int tw, int av), а если нет – записывает вместо этого параметры запуска в стек заданий.

1. Алгоритм отлажен и оптимизирован в консольном режиме. Время работы существенно сокращено удалением лишних обходов контейнеров.
2. Реализован графический интерфейс для работы. Интерфейс представлен на рис.1:

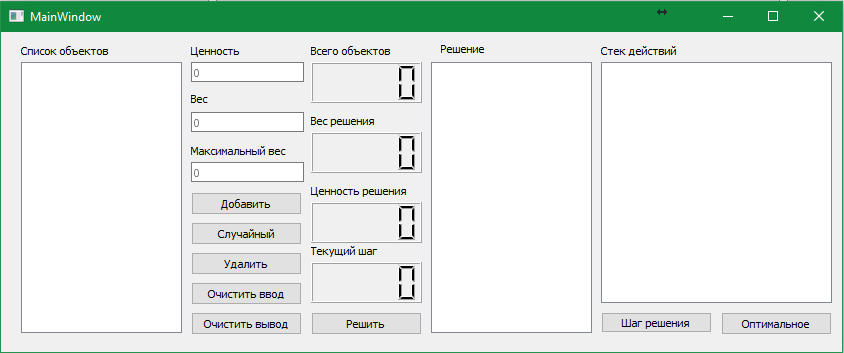


Рисунок 1 – интерфейс программы.

Важно использование классов QListWidget – они отображают содержимое стеков Opt\_task. Для работы с ними в класс Item добавлен элемент QListWidgetItem и переменные для цвета. Наиисаны методы, итерирующиеся через стеки и записывающие содержимое последних в QListWidget’ы. Коды этих методов находятся в mainwindow.cpp в приложении Д.

Из выборки выводятся вес и ценность элемента. Из задания по обработке – какой элемент попытаться включить и исключить и сколько элементов уже лежит в выборке

1. Протестирована работа программы на небольшом примере.

Взята выборка:

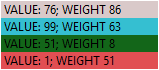


Рисунок 2. Маленькая выборка

Сделана пошаговая проверка:

Таблица 1 – пошаговая проверка

|  |  |  |
| --- | --- | --- |
| № шага | Текущая выборка | Задания в стеке |
| 1 |  |  |
| 2 |  |  |
| 3 |  |  |
| 4 |  |  |
| 5 |  |  |
| 6 |  |  |
| 7 |  |  |
| 8 |  |  |
| 9 |  |  |
| Конец | | |

Получается, что выборка с наибольшей ценностью получилась на последнем, 9-м шаге.

1. Протестирована работа программы на большом примере. Для этого взята выборка из 40 элементов, представленная на рис. 3:



Рисунок 3. Выборка

Получено решение для максимального веса, равного 100:

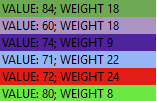


Рисунок 4. Результат

Решение заняло 14 584 шага. Если бы эта задача решалась методом перебора, то пришлось бы сделать 1 099 511 627 776 шагов (), что заняло бы более 20 лет на компьютере автора.

**Выводы.**

Изучена реализация алгоритма оптимального выбора в общем виде и на конкретном примере.

Выявлены возможности пошаговой реализации рекурсивных алгоритмов.

Изучена работа с классами QListWidget и QDebug, а также с некоторыми их производными. Разобран механизм использования итераторов для проходов через контейнеры.

Приложение А

Код ITEM.H

#ifndef ITEM\_H

#define ITEM\_H

#include <QColor>

#include <QString>

#include <QTextStream>

#include <QListWidget>

#include <qDebug>

class Item{

public:

QColor **getcol**();

void **set**(int w, int v);

void **set**(QString w, QString v, bool rnd = 0);

void **set**(Item it);

int **get\_weight**(){ return weight; }

int **get\_value**() { return value; }

bool **ok**() {return ((weight!=0) && (value!=0)); }

QString **print**();

QListWidgetItem\* wItem;

private:

int redcol = 0; int greencol = 0; int bluecol = 0;

void **setItem**(bool randcol=1);

int weight = 0;

int value = 0;

};

#endif // ITEM\_H

Приложение Б

Код ITEM.CPP

#include "item.h"

QColor Item::**getcol**()

{

QColor col;

col.setRgb(redcol, greencol, bluecol);

return col;

}

void Item::**set**(int w, int v) {

weight = w;

value = v;

setItem();

}

void Item::**set**(QString w, QString v, bool rnd)

{

bool okw, okv;

int wt = w.toInt(&okw);

int vt = v.toInt(&okv);

if (okw && okv){

if (!rnd){

weight = wt;

value = vt;

}

else{

weight = rand() % wt + 1;

value = rand() % vt + 1;

}

}

setItem();

}

void Item::**set**(Item it)

{

weight = it.weight;

value = it.value;

redcol = it.redcol;

greencol = it.greencol;

bluecol = it.bluecol;

setItem(0);

}

QString Item::**print**()

{

QString str;

QTextStream strm(&str);

strm << "VALUE: " << get\_value() << "; WEIGHT " << get\_weight() << " ";

return str;

}

void Item::**setItem**(bool randcol)

{

if (value && weight){

wItem = new QListWidgetItem(print());

if (randcol){

redcol = std::rand()%255;

greencol = std::rand()%255;

bluecol = std::rand()%255;

}

wItem->*setBackgroundColor*(getcol());

}

}

Приложение В

Код OPT\_TASK.H

#ifndef OPT\_TASK\_H

#define OPT\_TASK\_H

#include <iostream>

#include <QStack>

#include <QQueue>

#include <QList>

#include <QtDebug>

#include <item.h>

class try\_parm{

public:

try\_parm(int it, int twt, int avt){i=it; tw=twt; av=avt; }

try\_parm() = default;

int i;

int tw;

int av;

QStack<Item> s;//Вариант для запуска

};

class Opt\_task{

public:

Opt\_task(int nt); //Конструктор

Opt\_task() = default;

QList<Item> all; //Стек из всех предметов

QStack<Item> s;//Текущий вариант

QStack<Item> opts;//Оптимальный вариант

QStack<try\_parm> series;//Стек запусков функции

int **totv\_all**(); //Суммарная ценность массива

void **try\_item**(); //Инициализация

void **try\_item**(int i, int tw, int av); //Рекурсивное решение

void **try\_item**(int i, int tw, int av, bool steps); //Вариант для пошаговой обработки

void **try\_item**(try\_parm parm);

void **setlimw**(int i);

void **setlimw**(QString i);

void **clrzeros**(); //Очистка от мусора

int **g\_totw**(QStack<Item> s); //Суммарный вес сета

int **g\_totv**(QStack<Item> s); //Суммарная ценность сета

int **g\_nums**() {return nums; } //Вывод количества решений

void **out**(QStack<Item> s); //Вывод сета

void **make\_arr**();

bool steps = 0;

bool go = 1;

private:

void **try\_incl**(int i, int tw, int av);

void **try\_unincl**(int i, int tw, int av);

int n;

int limw = 0; //Максимальный вес

int totw = 0; //Вес текущей выборки

int maxv = 0; //Максимальная ценость оптимальной выборки

int nums = 0; //Количество решений

};

#endif // OPT\_TASK\_H

Приложение Г

Код OPT\_TASK.CPP

#include "opt\_task.h"

Opt\_task::**Opt\_task**(int nt){

n = nt;

totw = 0;

maxv = 0;

}

int Opt\_task::**totv\_all**(){

int res = 0;

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

res = all[i].get\_value() + res;

}

return res;

}

void Opt\_task::**try\_item**()

{

// make\_arr();

opts.clear();

s.clear();

nums = 0;

clrzeros();

n = all.count();

Item zero;

zero.set(0, 0);

all.push\_back(zero);

totw = 0; maxv = 0;

try\_item(0, 0, totv\_all());

}

void Opt\_task::**out**(QStack<Item> s){

for (auto it : s){

qDebug() << "VALUE: " << it.get\_value() << "; WEIGHT: " << it.get\_weight() << endl;

}

qDebug() << "-----" << endl;

}

int Opt\_task::**g\_totw**(QStack<Item> s){

int res = 0;

for (auto it : s){

res = res + it.get\_weight();

}

return res;

}

int Opt\_task::**g\_totv**(QStack<Item> s){

int res = 0;

for (auto it : s){

res = res + it.get\_value();

}

return res;

}

void Opt\_task::**try\_item**(int i, int tw, int av){

if (!steps)

out(opts);

nums++;

Item it;

it.set(all[i]);

if (!steps){

try\_incl(i, tw, av);

try\_unincl(i, tw, av);

}

else{

try\_unincl(i, tw, av);

try\_incl(i, tw, av);

}

}

void Opt\_task::**try\_incl**(int i, int tw, int av)

{

Item it;

it.set(all[i]);

if ((tw + it.get\_weight() <= limw)){ //Включение возможно?

if (it.get\_value()) s.push(it); //Включение элемента

if (i < n) //Если не конец, то ...

try\_item(i+1, tw+it.get\_weight(), av, steps); //... пробуем дальше,

else if (av > maxv){ //иначе - проверка оптимума

maxv = av;

opts = s;

}

if (it.get\_value()) s.pop(); //Выключение элемента

}

}

void Opt\_task::**try\_unincl**(int i, int tw, int av)

{

Item it;

it.set(all[i]);

int av1 = av - it.get\_value(); //Сколько сможем получить, если не включим

if (av1 > maxv) { //Возможно невключение?

if (i < n)

try\_item(i+1, tw, av1, steps);

else{ //Если мы здесь, то проверка уже не нужна

maxv = av1;

opts = s;

}

}

}

void Opt\_task::**try\_item**(int i, int tw, int av, bool steps)

{

if (!steps){

try\_item(i, tw, av);

}

else{

try\_parm parm(i,tw,av);

parm.s = s;

series.push(parm);

}

}

void Opt\_task::**try\_item**(try\_parm parm)

{

QStack<Item> temp = s;

s = parm.s;

try\_item(parm.i, parm.tw, parm.av);

s = temp;

}

void Opt\_task::**setlimw**(int i)

{

if (limw > 0 )

limw = i;

}

void Opt\_task::**setlimw**(QString i)

{

bool ok;

int t = i.toInt(&ok);

if (ok)

limw = t;

}

void Opt\_task::**clrzeros**()

{

if (!all.isEmpty()){

if (!all.back().ok()){

all.pop\_back();

clrzeros();

}

}

}

Приложение Д

Код Mainwindow.cpp

#include "mainwindow.h"

#include "ui\_mainwindow.h"

#include <QApplication>

#include <QMessageBox>

#include <QString>

#include <QTextStream>

#include <qDebug>

#include <ctime>

#include <item.h>

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

QMainWindow(parent),

ui(new Ui::MainWindow)

{

ui->setupUi(this);

srand(time(nullptr));

}

void MainWindow::**out\_sol**(QStack<Item> s)

{

ui->solWidget->clear();

for (auto it : s){

Item item;

item.set(it);

ui->solWidget->addItem(item.wItem);

}

ui->weightNum->display(op.g\_totw(s));

ui->valueNum->display(op.g\_totv(s));

}

void MainWindow::**out\_parm**(QStack<try\_parm> series)

{

ui->qList->clear();

for (auto it: series){

QString str;

QTextStream strm(&str);

strm << "Element " << it.i << ". There is " << it.s.count() << " elements in list";

ui->qList->addItem(str);

}

}

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

{

delete ui;

}

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

{

Item it;

it.set(ui->weightEdit->text(), ui->valueEdit->text());

if (it.ok()){

ui->listWidget->addItem(it.wItem);

}

op.all.push\_back(it);

ui->weightEdit->clear();

ui->valueEdit->clear();

ui->listNum->display(ui->listWidget->count());

}

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

{

Item it;

it.set(ui->weightEdit->text(), ui->valueEdit->text(), 1);

if (it.ok()){

ui->listWidget->addItem(it.wItem);

}

op.all.push\_back(it);

ui->listNum->display(ui->listWidget->count());

}

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

{

if (ui->listWidget->count()){

ui->listWidget->takeItem(ui->listWidget->count()-1);

Item it;

op.clrzeros();

op.all.pop\_back();

}

ui->listNum->display(ui->listWidget->count());

}

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

{

ui->listWidget->clear();

op.all.clear();

ui->listNum->display(ui->listWidget->count());

}

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

{

op.steps = 0;

op.setlimw(ui->limwEdit->text());

op.try\_item();

out\_sol(op.opts);

ui->stepNum->display(op.g\_nums());

}

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

{

op.go = 1;

op.series.clear();

ui->solWidget->clear();

ui->weightNum->display(0);

ui->valueNum->display(0);

ui->stepNum->display(0);

ui->qList->clear();

}

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

{

op.steps = 1;

op.setlimw(ui->limwEdit->text());

if (op.series.isEmpty()){

if (op.go){

op.go = 0;

op.clrzeros();

op.try\_item();

}

else{

QMessageBox msgBox;

msgBox.setText("Конец");

msgBox.*exec*();

}

}

else{

try\_parm parm = op.series.pop();

out\_sol(parm.s);

op.out(parm.s);

QStack<Item> temp;

op.try\_item(parm);

op.s = temp;

ui->stepNum->display(ui->stepNum->value()+ 1);

}

out\_parm(op.series);

}

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

{

out\_sol(op.opts);

}