МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение
высшего образования
«Тольяттинский государственный университет»

Институт **математики, физики и информационных технологий**

Кафедра **«Прикладная математика и информатика»**

01.03.02 ПРИКЛАДНАЯ МАТЕМАТИКА И ИНФОРМАТИКА

СИСТЕМНОЕ ПРОГРАММИРОВАНИЕ И КОМПЬЮТЕРНЫЕ ТЕХНОЛОГИИ

БАКАЛАВРСКАЯ РАБОТА

на тему **Реализация задачи о рюкзаке методом динамического программирования**

Студент Н.А. Комаров _____

Руководитель Г.А. Тырыгина _____

Консультант по аннотации Н.В. Ященко _____

Допустить к защите
Заведующий кафедрой к.т.н., доцент, А.В. Очеповский_____

«_____» _____ 20___ г.

Тольятти 2017

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение
высшего образования
«Тольяттинский государственный университет»

Институт **математики, физики и информационных технологий**

Кафедра **«Прикладная математика и информатика»**

УТВЕРЖДАЮ
Зав. кафедрой «Прикладная
математика и информатика»
_____ А.В. Очеповский

«____»_____ 2016 г.

ЗАДАНИЕ
на выполнение бакалаврской работы

Студент Комаров Никита Александрович

1. Тема: «Реализация задачи о рюкзаке методом динамического программирования»

2. Срок сдачи студентом законченной выпускной квалификационной работы: июнь, 2017г.

3. Исходные данные к выпускной квалификационной работе: реализация программы должна выполняться на Java-платформе с использованием языка Java.

4. Содержание выпускной квалификационной работы (перечень подлежащих разработке вопросов, разделов):

Введение

Глава 1. Общие подходы использования динамического программирования в прикладных задачах.

Глава 2. Конкретные реализации динамического программирования в прикладных задачах.

Глава 3. Программная реализация.

Заключение

Список литературы

Приложение

5. Ориентировочный перечень графического и иллюстративного материала: математические модели; презентация на тему ВКР, блок-схема алгоритмов, программная реализация; экранные формы демонстрации работоспособности реализованной программы.

6. Дата выдачи задания «26» декабря 2016 г.

Руководитель выпускной
квалификационной работы _____ Г.А. Тырыгина

Задание принял к исполнению _____ Н.А. Комаров

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение
высшего образования
«Тольяттинский государственный университет»

Институт **математики, физики и информационных технологий**

Кафедра **«Прикладная математика и информатика»**

УТВЕРЖДАЮ
Зав. кафедрой «Прикладная
математика и информатика»
_____ А.В. Очеповский

«____»_____ 2016 г.

КАЛЕНДАРНЫЙ ПЛАН
выполнения бакалаврской работы

Студента Комарова Никиты Александровича
по теме «Реализация задачи о рюкзаке методом динамического программирования».

Наименование раздела работы	Плановый срок выполнения раздела	Фактический срок выполнения раздела	Отметка о выполнении	Подпись руководителя
Анализ литературных источников	20.11.16			
Написание введения	20.12.16			
Написание 1 главы. Исследование алгоритмов. Выявление достоинств и недостатков алгоритмов	20.02.17			
Написание 2 главы. Реализация алгоритмов. Тестирование алгоритмов	20.03.17			
Подведение итогов. Редактирование бакалаврской работы. Создание презентационного материала	31.03.16			

Оформление пояснительной записки	05.04.17			
Проверка на наличие заимствований в системе antiplagiat.ru	06.06.17			
Предварительная защита	13.06.17			
Сдача на кафедру комплексы документов для защиты	22.06.17			
Защита ВКР	28.06.17			

Руководитель выпускной квалификационной работы _____ Г.А. Тырыгина

Задание принял к исполнению _____ Н.А. Комаров

АННОТАЦИЯ

Тема выпускной квалификационной работы: «Реализация задачи о рюкзаке методом динамического программирования»

Работа была выполнена студентом Тольяттинского Государственного Университета, института математики, физики и информационных технологий, группы ПМИб-1301, Комаровым Никитой Александровичем. Выпускная квалификационная работа посвящена созданию алгоритма для реализации задачи о рюкзаке методом динамического программирования.

Объект исследования – математическая модель задачи о рюкзаке.

Предмет исследования – оптимизация задачи о рюкзаке методом динамического программирования.

Целью выпускной квалификационной работы является использование метода динамического программирования для решения задачи о рюкзаке.

Задачи работы**:**

1) Изучение общих подходов динамического программирования;
2) Создание алгоритма для решения задачи о рюкзаке;
3) Программная реализация задачи о рюкзаке;

Выпускная квалификационная работа состоит из введения, трёх глав, заключения, списка используемых источников.

Во введении рассказывается об актуальности исследования по выбранному направлению, ставится проблема, цель и задачи исследования, определяются объект, предмет научных поисков, формулируется гипотеза, ставятся цель и задачи, указывается методологическая база исследования, его теоретическая, практическая значимости.

В главе 1 рассматриваются общие подходы использования динамического программирования в прикладных задачах. В главе 2 приводятся конкретные реализации динамического программирования в прикладных задачах. В главе 3 разрабатываются алгоритм и интерфейс программы. В заключении представлены результаты и выводы о выполненной работе.

В работе использовано 3 таблицы, 6 рисунков и 1 приложение, список литературы содержит 21 источник, в том числе 5 источников на иностранном языке. Общий объем выпускной квалификационной работы составляет 57 страниц.

ABSTRACT

The title of the graduation work is «Implementation of the Knapsack Problem by Dynamic Programming».

The aim of this work is the use of the method of dynamic programming for solving knapsack problem.

The object of study is a mathematical model of the knapsack problem.

The subject of the study is optimizing the knapsack problem by dynamic programming.

The graduation work is devoted to solving the knapsack problem by dynamic programming.

In the first part the theoretical basis of the method of dynamic programming is discussed. The rationale of dynamic programming methods is considered in applied problems. The adequacy of dynamic programming methods in applied problems is examined. The work has helped in the writing of the algorithm for solving the problem. The second part provides an overview of the methods of dynamic programming in continuous and discrete case, and their mathematical models. In the third part algorithm and interface are developed.

The result of graduation is the developed algorithm and the program that solves the task about a backpack.

The graduation work consists of an explanatory note on 57 pages, introduction, three parts including 6 figures, 3 tables, the list of 21 references including 5 foreign sources.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Динамическое программирование в математике и теории вычислительных систем — метод решения задач с оптимальной подструктурой и перекрывающимися подзадачами, который намного эффективнее, чем решение «в лоб». Словосочетание «динамическое программирование» впервые было использовано в 1940-х годах Р. Беллманом для описания процесса нахождения решения задачи, где ответ на одну задачу может быть получен только после решения задачи, «предшествующей» ей. В 1953 г. он уточнил это определение до современного.

Впервые сформулированная американским математиком Д.Б. Данцигом задача о рюкзаке (ранце) (англ. Knapsack problem) – одна из NP-полных задач комбинаторной оптимизации, популярность которой вызвана большим количеством её приложений, поскольку многие из реально возникающих задач описываются в рамках данной модели. Основные сферы применения находятся в областях планирования и управления производственными и транспортными системами. Название своё получила от максимизационной задачи укладки как можно большего числа ценных вещей в рюкзак при условии, что общий объём (или вес) всех предметов, способных поместиться в рюкзак, ограничен. Непосредственно термин «рюкзак» может быть интерпретирован достаточно широко. Данная задача и её варианты широко используются для моделирования большого числа практических задач.

Актуальность работы заключается в том, что мы можем решить задачу, в которой присутствует оптимальная подструктура, проделывая следующие три шага.

1) Разбиение задачи на подзадачи меньшего размера.

2) Нахождение оптимального решения подзадач рекурсивно, проделывая такой же трехшаговый алгоритм.

3) Использование полученного решения подзадач для конструирования решения исходной задачи.

Объект исследования – математическая модель задачи о рюкзаке.

Предмет исследования – оптимизация задачи о рюкзаке методом динамического программирования.

Целью выпускной квалификационной работы является использование метода динамического программирования для решения задачи о рюкзаке.

Задачи работы**:**

1) Изучение общих подходов динамического программирования;
2) Создание алгоритма для решения задачи о рюкзаке;
3) Программная реализация задачи о рюкзаке.

Выпускная квалификационная работа состоит из введения, трёх глав, заключения, списка используемых источников.

В главе 1 рассматриваются общие подходы использования динамического программирования в прикладных задачах. В главе 2 приводятся конкретные реализации динамического программирования в прикладных задачах. В главе 3 разрабатываются алгоритм и интерфейс программы. В заключении представлены результаты и выводы о выполненной работе.

ГЛАВА 1 ОБЩИЕ ПОДХОДЫ ИСПОЛЬЗОВАНИЯ ДИНАМИЧЕСКОГО ПРОГРАММИРОВАНИЯ В ПРИКЛАДНЫХ ЗАДАЧАХ

1.1 Обоснование необходимости методов динамического программирования в прикладных задачах

Динамическое программирование – это математический аппарат, который разработан для эффективного решения некоторых задач по математическому программированию. Характеризуется данный класс возможностью естественного (или искусственного) разбиения полностью всей операции на несколько взаимосвязанных между собой этапов.

Связано динамическое программирование с возможностью представить процесс управления в виде цепочки последовательных шагов или действий, которые развернуты во времени и ведущие к цели.

Применяются модели динамического программирования, например, в разработке правил по управлению запасами, которые устанавливают, когда именно были пополнены запасы, а также и сам размер пополняющего заказа; в процессе разработки в производстве принципов календарного планирования; в процессе распределения дефицитных капиталовложений среди возможных новых направлений их использования; в процессе составления календарных планов капитального и текущего ремонта сложного оборудования, а также его замены.

Как научное направление динамическое программирование сформировалось в 1951 - 1953 годах. Его методы связаны с именем американского математика Р. Беллмана и его сотрудников.

Динамическое программирование – это раздел математического программирования, в котором изучаются методы нахождения оптимальных решений в задачах с многошаговой (многоэтапной) структурой.

Однако методы динамическое программирование успешно используются и для решения задач, где не учитывается фактор времени. В связи с этим,

термин многоэтапное планирование является более точным. Он отражает пошаговый характер самого процесса решения задач.

Динамическим программированием называют поэтапное планирование многошагового процесса, в процессе которого происходит оптимизация на каждом этапе только одного шага. Управления, на всех этапах, необходимо выбирать, учитывая все его в будущем последствия.

Особенностями математической модели динамического программирования являются:

1) формулируется задача оптимизации в качестве итогового/последнего многошагового процесса управления;

2) определяется критерий оптимальности операции или показатель эффективности целевой функцией, являющейся аддитивной от любого шага оптимизации:

3) зависит выбор управления X_k на каждом шаге исключительно от состояния самой системы на этом шаге, не имеет влияния на предшествующие шаги (отсутствует обратная связь);

4) после каждого шага управления состояние системы S_k зависит исключительно от предшествующего состояния самой системы и управляющего воздействия X_k (нет последействия) и может быть сохранено в виде уравнения состояния системы;

5) зависит на каждом шаге управление X_k от последнего числа управляющих переменных, от конечного числа параметров зависит состояние системы S_k;

6) оптимальное управление - вектор X*, который определяется последовательностью оптимальных пошаговых управлений, чье число и определяет какое количество у задачи шагов.

Дадим формулировку общему принципу, который находится в основе решения любой задачи по динамическому программированию («принцип оптимальности»):

«Каково бы ни было состояние системы S перед очередным шагом, надо выбрать управление на этом шаге так, чтобы выигрыш на данном шаге плюс оптимальный выигрыш на всех последующих шагах был максимальным».

Но в том случае, если в задачах по линейному программированию зависимости целевой функцией и переменными строго линейны, то в задачах ДП у этих зависимостей может быть также и нелинейный характер.

ДП можно применять для решения задач, которые связаны с динамикой системы или процесса, для статических задач, которые связаны, к примеру, с распределением ресурсов. Все это существенно увеличивает область использования ДП для того, чтобы решить задачи управления. Возможность упростить процесс решения можно достичь с помощью ограничения области, а также количества исследуемых в процессе перехода к очередному этапу вариантов, усиливает у данного метода достоинства.

Одновременно с этим, у ДП есть и недостатки. В первую очередь: отсутствие единого универсального метода решения. Почти все задачи, которые решаются данным методом, характеризуются своими особенностями, требует осуществления проведения поиска наиболее приемлемой совокупности методов для ее решения.

Помимо этого, значительные объемы, трудоемкость решения многошаговых задач, которые имеют большое число различных состояний, способны привести к необходимости отбирать небольшие задачи или использовать сжатую информацию. Последнее можно достичь путем анализа вариантов, а также переработки списка состояний.

Фундаментальный принцип, находящийся в основе теории ДП - принцип оптимальности, который определяет в каком порядке будет происходить решение, допускает декомпозицию задачи (более приемлемый путь, в отличии от непосредственного решения задач в исходной постановке) используя рекуррентные вычислительные процедуры.

1.2 Достаточность методов динамического программирования в прикладных задачах

Динамическое программирование — метод решения задачи путём её разбиения на несколько одинаковых подзадач, рекуррентно связанных между собой. Самым простым примером будут числа Фибоначчи — чтобы вычислить некоторое число в этой последовательности, нам нужно сперва вычислить третье число, сложив первые два, затем четвёртое таким же образом на основе второго и третьего, и т.д.

Решение задачи динамическим программированием должно содержать следующее:

- Зависимость элементов динамики друг от друга. Такая зависимость может быть прямо дана в условии (так часто бывает, если это задача на числовые последовательности). В противном случае можно попытаться узнать какой-то известный числовой ряд (вроде тех же чисел Фибоначчи), вычислив первые несколько значений вручную.
- Значение начальных состояний. В результате долгого разбиения на подзадачи необходимо свести функцию либо к уже известным значениям (как в случае с Фибоначчи — заранее определены первые два члена), либо к задаче, решаемой элементарно.

Рассмотрим операцию (управляемый процесс), которая состоит из отдельных шагов/этапов. В том случае, если складывается критерий эффективности всей операции из показателей на отдельных шагах эффективности, то называют его аддитивным критерием.

Примеры многоэтапных операций с аддитивным критерием.

Пример №1. Задача распределения ресурса.

Планируется деятельность 2-х компаний на период равный n лет. На развитие компаний в начале периода выделена определенная сумма, которая в последующем делится между данными компаниями предприятиями. За один год частично амортизируются вложенные средства, а частично сохраняются, затем снова их можно перераспределить. Имеющиеся средства в начале

каждого года перераспределяются между компаниями. Каждая компания приносит за год доход, зависящий от средств, вложенных в него. Сколько средств в начале каждого года требуется вложить в каждую компанию, чтобы за n лет суммарный доход был максимальным?

Процесс, рассматриваемый в задаче, считается многоэтапным, развивается во времени, а этапы соответствуют годам.

Суммарный доход и сумма доходов, которая получена на отдельных этапах, равны, а значит функция, которая определяет общий доход - аддитивная.

Пример №2. Задача о рюкзаке.

Есть определенный набор предметов из конечного числа видов. Указана стоимость предметов и объем рюкзака, которые известен и ограничен. Какое количество предметов каждого вида требуется положить в рюкзак (если суммарный объем меньше объема рюкзака), чтобы была их суммарная стоимость максимальна?

Операцию, которая рассматривается в данной задаче, можно считать многоэтапной. Этап будет связан с укладкой предметов отдельного вида в рюкзак. В этом случае, на этапе k будет происходить решение вопроса о том, какое количество предметов k-го вида требуется положить в рюкзак, если уже какая-то часть общего объема занята другими предметами 1-го, 2-го, . . ., (k-1)-го видов. В данной задаче соответствует переход от одного этапа к другому движению по перечню предметов.

Суммарная стоимость используемых предметов при этом рассчитывается путем сложения стоимости предметов отдельных видов, т.е. и в данной задаче у критерия есть свойство аддитивности.

Пример №3. Задача о прокладке выгодного пути от одного пункта к другому.

Необходимо провести дорогу из пункта А до пункта В таким образом, чтобы на постройку участка были минимальными суммарные затраты.

Такую задачу можно привести к многошаговому процессу. Необходимо данный отрезок от A до B разделить на n частей, провести через точки деления прямые, которые будут перпендикулярны отрезку AB. Если сделать прямые неподалеку друг от друга, то в этом случае отрезки пути, а также их соединяющие, считаются прямолинейными. Получается, что можно связать этап (шаг) с выбором направления для того, чтобы переходить с одной прямой на другую.

Затраты для того, чтобы построить весь путь будут равны затратам на строительство отдельных участков, которые соединяют прямые, расположенные по соседству.

В данной задаче критерий (затраты) являются аддитивными. Сводится переход от одного этапа к следующему сводится к пространственному движению.

В процессе использования метода динамического программирования требуется следовать некоторым принципам.

Принципы динамического программирования.

Зависимость от параметров оптимального значения критерия исходной задачи. Рассматривают оптимальное значение критерия исходной задачи в качестве функции номера этапа, а также одного или же нескольких параметров – параметры состояния.

Принцип вложения. Задачи, которые решаются на всех этапах образуют для всевозможных значений параметров состояния, параметрическое семейство задач. Исходная задача при этом – одна из задач данного семейства.

Наличие рекуррентных уравнений, которые устанавливают связь задач соседних этапов. Выводятся рекуррентные уравнения динамического программирования, которые устанавливают связь задач соседних этапов между оптимальными значениями целевых функций.

Принцип оптимальности.

1-ое определение данному понятию заключается в том, что для того, чтобы получить оптимальное решение многоэтапного процесса решение,

используемое на отдельном этапе, обязательно должно быть оптимальным относительно состояния, в котором оказалась система к началу этого этапа, и не должно находиться в зависимости от решений с предыдущих этапов по чей причине, система находится в таком состоянии.

2-ое определение данному понятию заключается в том, что оптимальным является какой-либо участок для аддитивного функционала оптимальной траектории.

Означает это свойство то, что принимаемое решение на каждом этапе находится вне зависимости от предыстории процесса, другими словами процесс является Марковским.

Решение задачи методом динамического программирования о рюкзаке.

Задачи о рюкзаке и её модификации часто возникают в экономике, прикладной математике, криптографии, генетике и логистике для нахождения оптимальной загрузки транспорта (автомобиля, контейнера, самолёта, поезда, трюма корабля) или склада. Поэтому разработке методов решения задачи, и в первую очередь эффективных, уделено достаточно много внимания.

Требуется рюкзак объема A уложить предметами вида j ($j = \overline{1,n}$), опираясь на максимизацию ценности рюкзака. Есть объем предмета j-го вида: a_j. Ценность предметов вида j, которые взяты в количестве x_j, можно определить значением функции $f_j(x_j)$. Предметы на части делить нельзя.

Математическая модель задачи о рюкзаке

$$f(x) = \sum_{j=1}^{n} f_j(x_j) \to \max$$

$$\sum_{j=1}^{n} a_j x_j \le A \qquad (1.2.1)$$

$$x_j \ge 0, \quad x_j - \text{ целое.}$$

По той причине, что эта задача – задача целочисленного программирования, то есть возможность предположить, что у всех параметров задачи целые значения.

Реализация принципов динамического программирования.

1) Рассмотрим эту задачу в качестве многоэтапного процесса. Установим связь с этапом *k* задачи укладки рюкзака предметами *k*-го вида. До того, как наступил *k*-ый этап предполагаются предшествующие этапы, пройденными *k*-1 этапами. Т.к. нам неизвестна часть рюкзака, в которой будут находиться предметы 1,2,…, *k*-го видов, то в этом случае на *k*-м этапе осуществляем решение задачи оптимальной укладки любой допустимой части объема, которая задана параметром a=0,1,2,…,A. Решаем при этом семейство задач, которые зависят от параметра a=0,1,2,…,A:

$$f(x) = \sum_{j=1}^{k} f_j(x_j) \to \max$$

$$\sum_{j=1}^{k} a_j x_j \le a \qquad (1.2.2)$$

$$x_j \ge 0, \quad x_j - \text{ целое.}$$

Обозначим оптимальное значение критерия задачи (1.2.1, 1.2.2) $F_k(a)$.

$F_k(a)$ определяет максимальную ценность той части рюкзака объема a, где расположены предметы 1,2,…, *k*-го видов.

2) Очевидно, что входит исходная задача в семейство задач (1.2.1, 1.2.2) при $k=n$, $a=A$, и оптимальное значение целевой функции исходной задачи в принятых обозначениях равно $F_n(A)$.

3) В процессе использования метода динамического программирования нужно выразить $F_k(a)$ через значение $F_{k-1}(a)$, вычисленное на предыдущем этапе.

Выведем для этого рекуррентное уравнение:

$$F_k(a) = \max_{\sum_{j=1}^{k} a_j x_j \le a} \sum_{j=1}^{k} f_j(x_j) = \max_{a_k x_k \le a} [\, f_k(x_k) + \max_{\sum_{j=1}^{k-1} a_j x_j \le a - a_k x_k} \sum_{j=1}^{k-1} f_j(x_j)] =$$

$$= \max_{a_k x_k \le a} [\, f_k(x_k) + F_{k-1}(a - a_k x_k)] .$$

Рекуррентные уравнения динамического программирования в задаче о рюкзаке следующего вида:

$$F_k(a) = \max_{a_k x_k \le a} [\, f_k(x_k) + F_{k-1}(a - a_k x_k)] \text{ для k=2…n ,} \qquad (1.2.3)$$

$$F_1(a) = \max_{a_1 x_1 \le a} f_1(x_1) .$$

Во всех формулах (1.2.1, 1.2.3) подразумевается, что $x_j \ge 0$, x_j – целое.

4) На k-м этапе, используя (1.2.1, 1.2.3), можно для всех возможных значений a определить $F_k(a)$, при этом выбирая только управляющее решение $x_k = x_k(a)$ и используя значения $F_{k-1}(a)$, которые найдены на предыдущем этапе. Получается, что решение $x_k(a)$, принимаемое на k-м этапе зависит исключительно от параметра состояния a, не зависит от решений $x_1, x_2, \ldots, x_{k-1}$, используемых на предыдущих этапах. Так реализуется принцип оптимальности.

Описание вычислительной процедуры.

Номер k определяет номер этапа, параметр a – состояние системы. Используя (1.2.1, 1.2.3), для каждого k=1,…,n найдем значения $F_k(a)$ и $x_k(a)$ для a=0,1,…,A.

При этом $x_k(a)$ - это и есть значение x_k, на котором достигается максимум в рекуррентном соотношении (1.2.1, 1.2.3). Заполняем таблицу 1.1 результатами этих вычислений. Выделенные клетки таблицы заполняем числовыми значениями обозначенных в них выражений. Заполнение таблицы 1.1 - прямой ход динамического программирования.

Таблица 1.1 - Результаты вычислений при прямом ходе метода ДП

A	$F_1(a)$	$x_1(a)$	$F_2(a)$	$x_2(a)$	…	$F_n(a)$	$x_n(a)$
0	$F_1(0)$	$x_1(0)$	$F_2(0)$	$x_2(0)$	…	$F_n(0)$	$x_n(0)$
1	$F_1(1)$	$x_1(1)$	$F_2(1)$	$x_2(1)$	…	$F_n(1)$	$x_n(1)$
…	…	…	…	…	…	…	…
A	$F_1(A)$	$x_1(A)$	$F_2(A)$	$x_2(A)$	…	$F_n(A)$	$x_n(A)$

Благодаря обратному ходу есть возможность по значениям таблицы 1.1 определить решение исходной задачи.

Действительно, оптимальное значение целевой функции исходной задачи $f^* = F_n(A)$.

Найдем оптимальное решение:

$$a = A, \quad x_n^* = x_n(a)$$

$$a = a - a_n x_n, \quad x_{n-1}^* = x_{n-1}(a)$$

$$n) \quad a = a - a_2 x_2, \quad x_1^* = x_1(a)$$

ГЛАВА 2 КОНКРЕТНЫЕ РЕАЛИЗАЦИИ ДИНАМИЧЕСКОГО ПРОГРАММИРОВАНИЯ В ПРИКЛАДНЫХ ЗАДАЧАХ

2.1 Методы динамического программирования в непрерывном случае

Метод динамического программирования вначале разрабатывался Р. Беллманом и его учениками для решения задач управления динамическими объектами, движение которых (переход из состояния в состояние) описывалось дифференциальными уравнениями, осуществлялось под воздействием управлений, а оптимизируемыми функциями являлись интегральные функционалы. Позже он был распространён для решения некоторых классов задач статической оптимизации, в частности, задач комбинаторного характера. Метод предполагает, что процесс управления может быть представлен как последовательный N-шаговый (N-стадийный) процесс принятия решений. При этом выработка решений на очередных этапах не оказывает влияние на величину оптимизируемой функции, достигнутой на предшествующих шагах, т. е. рассматриваются системы без последствия (обратной связи). Если указанные условия выполняются, то оптимальное управление может быть построено, если следовать принципу оптимальности: оптимальная стратегия обладает тем свойством, что каковы бы ни были непосредственно предшествующее состояние и управление, последующее управление должно быть оптимальным по отношению к этому состоянию. Иными словами, в каждом состоянии по отношению к нему необходимо действовать оптимально.

Данный метод предполагает решение поставленной общей задачи оптимального управления несколько иным, чем в принципе максимума, способом. Основывается динамическое программирование на принципе оптимальности Беллмана, который можно сформулировать так: Оптимальное управление определяется только конечной целью управления и состоянием системы в рассматриваемый момент времени, независимо от того, каким образом система пришла в это состояние.

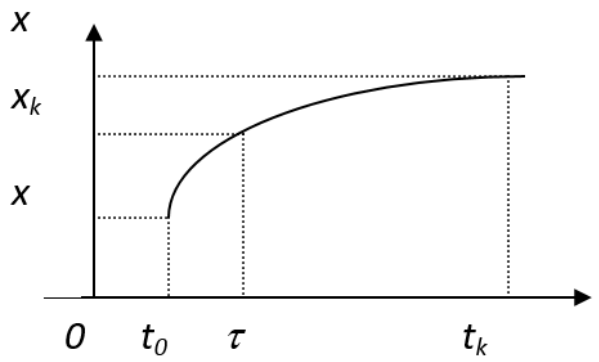


Рисунок 2.1.1 – оптимальная траектория

То есть если траектория $x=f(t), t\in[t_0,t_k]$ является оптимальной, то любая ее часть $x=f(t)$,

$t\in[\tau,t_k]$, приводящая к конечному состоянию X_k является также оптимальной траекторией.

Итак, объект описывается обычной системой дифференциальных уравнений с жестко заданными начальными условиями

$$\dot{x}=f(x,u), \quad x(t_0)=x_0, \quad u(t)\in U$$

Требуется найти уравнения $u = u_{опт}$, обеспечивающие минимум функционалу

$$I=\int_{t_0}^{t_k} f_0(x,u)dt+g_0[x(t_k)]\cdot$$

Введем в рассмотрение некоторую функцию

$$S[x(\tau)]=\min_{u\in U}\left\{\int_{\tau}^{t_k} f_0(x,u)dt+g_0[x(t_k)]\right\}, \qquad (2.1.1)$$

определенную на оптимальной траектории $x_{опт}(t)$, $\tau\le t\le t_k$. Отметим, что $S[x(\tau)]$ не зависит от управления $u(\tau)$, а только от $x(\tau)$ - начала интервала

оптимизации. Разобьем отрезок времени $[\tau,\ t_k]$ на два $[\tau,\tau+\Delta]$, $[\tau+\Delta,t_k]$ где Δ-сколь угодно малая величина. Тогда (2.1.1) представима в виде

$$S(x(\tau)) = \min_{u\in U}\left\{\int_{\tau}^{\tau+\Delta} f_0(x,u)dt + \int_{\tau+\Delta}^{t_k} f_0(x,u)dt + g_0[x(t_k)]\right\}.$$

В соответствии с принципом оптимальности часть

$$\int_{\tau+\Delta}^{t_k} f_0(x,u)dt + g_0[x(t_k)]$$

принимает на оптимальной траектории минимальное значение, следовательно, она может быть обозначена как

$$S(x(\tau+\Delta))$$

получили:

$$S(x(\tau)) = \min_{u\in U}\left\{\int_{\tau}^{\tau+\Delta} f_0(x,u)dt + S[x(\tau+\Delta)]\right\} \qquad (2.1.2)$$

Так как Δ- сколь угодно малая величина, то (метод прямоугольников)

$$\int_{\tau}^{\tau+\Delta} f_0(x,u)dt = f_0(x,u)\cdot\Delta$$

$$x(\tau+\Delta) = x(\tau) + \left(\frac{\partial x}{\partial t}\right)_{t=\tau}\cdot\Delta$$

(ряд Тейлора с первым членом). Учтем, что

$$\left(\frac{\partial x}{\partial t}\right)_{t=\tau}=\left(\frac{dx}{dt}\right)_{t=\tau}=f[x(\tau),u(\tau)]\text{, тогда}$$

$$S[x(\tau+\Delta)]=S\left[x(\tau)+f[x(\tau),u(\tau)]\Delta\right]\approx S[x(\tau)]+\frac{\partial S}{\partial x}\Big|_{x=x(\tau)}\cdot f[x(\tau),u(\tau)]\cdot\Delta$$

Таким образом,

$$S\{x(\tau)\}=\min_{u\in U}\left\{f_0(x,u)\Delta+S[x(\tau)]+\left(\frac{\partial S}{\partial x}\right)_{x=x(\tau)}\cdot f[x(\tau),u(\tau)]\cdot\Delta\right\}$$

Так как $S\{x(\tau)\}$ не зависит от управления $u(\tau)$, то она может быть вынесена за знак минимума, тогда

$$\min_{u\in U}\left\{f_0(x,u)+\left(\frac{\partial S}{\partial x}\right)_{x=x(\tau)}f[x(\tau),u(\tau)]\right\}=0 \qquad (2.1.3)$$

Данное уравнение называется функциональным уравнением Беллмана, позволяющим определять оптимальное уравнение $u_{опт}(t)$. Очевидно, минимум выражения в квадратных скобках достигается либо при условии, что его производная по управлению u равна нулю, либо само выражение равно нулю при любых u, то есть

$$\begin{cases} f_0(x,u)+\dfrac{\partial S}{\partial x}f(x,u)=0, \\ \dfrac{\partial f_0(x,u)}{\partial u}+\dfrac{\partial S}{\partial x}\cdot\dfrac{\partial f(x,u)}{\partial u}=0 \end{cases}$$

исключим из данной системы $\frac{\partial S}{\partial x}$ и получим

$$f_0(x,u)\frac{\partial f}{\partial u} = f(x,u)\frac{\partial f_0}{\partial u} \cdot \qquad (2.1.4)$$

Из решения данного уравнения и определяется оптимальное уравнение

$$u = u_{опт}$$

Пример: дан электродвигатель постоянного тока с независимым возбуждением $T\dot{\omega}+\omega=ku_y$ где ω - угловая скорость вращения;

u_y – управляющее напряжение.

Требуется определить оптимальный закон управления u_y^*, минимизирующий функционал

$$I=\int_0^{t_k}(\rho\omega^2+\alpha u_y^2)dt \cdot$$

Для данной задачи очевидно

$$f(x,u)=-\frac{1}{T}\omega+\frac{ku_y}{T},$$

$$f_0(x,u)=\beta\omega^2+\alpha u_y^2$$

Составим уравнение (2.1.4)

$$f_0(x,u)\frac{\partial f}{\partial u}=f(x,u)\frac{\partial f_0}{\partial u}\Leftrightarrow(\rho\omega^2+\alpha u_y^2)\frac{K}{T}=2\alpha u_y(-\frac{1}{T}\omega+\frac{K}{T}u_y)$$

получаем:

$$\rho\frac{K}{T}\omega^2+\frac{\alpha K}{T}u_y^2=-\frac{2\alpha}{T}u_y\omega+\frac{2\alpha K}{T}u_y^2;$$

$$\frac{\alpha K}{T}u_y^2-\frac{2\alpha}{T}u_y\omega-\frac{\beta K}{T}\omega^2=0;$$

$$u_y^2-\frac{2}{K}\omega u_y-\frac{\beta}{\alpha}\omega^2=0$$

$$u_y=\left(+\frac{1}{K}\pm\sqrt{\frac{1}{K^2}+\frac{\beta}{\alpha}}\right)\omega$$

Как видим, управление получается в виде обратной связи по состоянию объекта. Это одно из основных достоинств метода динамического программирования по сравнению с принципом максимума. Однако, для многомерных систем и ограничений решение уравнения (2.1.4) в явном виде получить сложно. В этом случае приходится применять численное интегрирование (2.1.4) в реальном времени.

Среди направлений теории оптимального управления своей практической результативностью, особенно для многомерных систем выделяется направление, получившее название аналитическое конструирование регуляторов (АКР).

Термин аналитическое означает, что в данном методе заранее известно аналитическое (формульное) выражение для оптимального управления $u_{опт}(t)$ и требуется определить только параметры этого выражения. Впервые данный термин был введен академиком А.М. Летовым в 1960 г. Рассмотрим методы АКР для линейных многомерных объектов.

Для линейного стационарного детерминированного объекта, описываемого матричным дифференциальным уравнением

$$\dot{x}=ax+bu,\quad где\quad x\in X,\quad u\in U,\quad x(t_0)=0 \qquad (2.1.5)$$

требуется определить управление u, доставляющее минимум функционалу:

$$I = x^T(t_k)\rho x(t_k) + \int_{t_0}^{t_k} (x^T \beta x + u^T k^{-1} u)dt, \qquad (2.1.6)$$

где $x^T(t_k)\rho x(t_k)$ - терминальная составляющая, определяющая требования к конечному состоянию объекта;

ρ, β, k - заданные, положительно определенные, симметричные матрицы коэффициентов функционала.

Рассмотрим положительно определенную квадратичную форму от фазовых координат объекта:

$V = x^T \Gamma x$, где Γ – некоторая неизвестная матрица коэффициентов, и определим её полную производную на уравнениях объекта (2.1.5).

$$\frac{dV}{dt} = \dot{x}^T \Gamma x + x^T \dot{\Gamma} x + x^T \Gamma \dot{x} = (x^T a^T + u^T b^T)\Gamma x + x^T \dot{\Gamma} x + x^T \Gamma (ax + bu).$$

Допустим, что искомое оптимальное управление имеет вид:

$$u = u_{опт} = -kb^T \Gamma x \qquad (2.1.7)$$

Подставим это выражение в формулу для производной.

$$\frac{dV}{dt} = x^T a^T \Gamma x - x^T \Gamma b k b^T \Gamma x + x^T \dot{\Gamma} x + x^T \Gamma a x - u_{опт}^T k^{-1} u_{опт} =$$
$$= x^T (\dot{\Gamma} + \Gamma a + a^T \Gamma - \Gamma b k b^T \Gamma) x - u_{опт}^T k^{-1} u_{опт}.$$

Потребуем, чтобы:

$$\dot{\Gamma} + \Gamma a + a^T \Gamma - \Gamma b k b^T \Gamma = -\beta \qquad (2.1.8)$$

тогда:

$$\frac{dV}{dt} = -x^T \beta x - u_{onm}^T k^{-1} u_{onm} \qquad (2.1.9)$$

В силу положительной определенности матриц β и k получили, что производная есть функция отрицательная. Поэтому, согласно 2-му методу Ляпунова замкнутая AC с управлением u_{onm} является асимптотически устойчивой.

Уравнение (2.1.8) называется матричным дифференциальным уравнением Риккати – его решение (матрица Γ) определяет неизвестные параметры оптимального закона управления u_{onm}. Причем матрица Γ получается положительно определенной независимо от того устойчива или неустойчива исходная система.

Докажем теперь, что управления u_{onm} обеспечивают не только устойчивость замкнутой системы, но и доставляют минимум заданному функционалу качества (то есть являются оптимальными). Для этого проинтегрируем выражение (2.1.9):

$$\int_{t_0}^{t_k} \frac{dV}{dt} dt = -\int_{t_0}^{t_k} x^T \beta x dt - \int_{t_0}^{t_k} u_{onm}^T k^{-1} u_{onm} dt \quad \text{или}$$

$$\int_{t_0}^{t_k} x^T \beta x dt = -x^T(t_k)\Gamma(t_k)x(t_k) + x^T(t_0)\Gamma(t_0)x(t_0) - \int_{t_0}^{t_k} u_{onm}^T k^{-1} u_{onm} dt$$

Подставим данный интеграл в выражение для I

$$I = x^T(t_k)\rho x(t_k) - x^T(t_k)\Gamma(t_k)x(t_k) + \int_{t_0}^{t_k} u^T k^{-1} u - \int_{t_0}^{t_k} u_{onm}^T k^{-1} u_{onm} dt$$

Так как I функция по определению неотрицательная, то ее абсолютный минимум (ноль) достигается при $u=u_{опт}$ и $\Gamma(t_k)=\rho$.И так, полученный результат можно сформулировать в виде следующей теоремы.

Теорема Летова-Калмана: Для объекта

$$\dot{x}=ax+bu, x(t_0)=0, x\in X, u\in U$$

оптимальными в смысле минимума функционала

$$I=x^T(t_k)\rho x(t_k)+\int_{t_0}^{t_k}(x^T\beta x+u^T k^{-1}u)dt$$

служат управления

$$u_{опт}(t)=-kb^T\Gamma(t)x(t),$$

где $\Gamma(t)$- положительно определенная матрица, решение уравнения Риккати

$$\dot{\Gamma}+\Gamma a+a^T\Gamma-\Gamma bkb^T\Gamma=-\beta$$

при граничном условии $\Gamma(t_k)=\rho$

Итак, процедура определения оптимального управления сводится к решению нелинейного дифференциального уравнения Риккати в обратном времени при граничном условии $\Gamma(t_k)=\rho$, в запоминании полученной программы $\Gamma(t)$ и ее реализации в реальном времени.

Отметим, что данная вычислительная процедура существенно упрощается при решении нетерминальных задач оптимального уравнения

(когда не заданы требования к конечному состоянию объекта и времени оптимизации), то есть

$$I=\int_0^{\infty}(x^T\beta x+u^T k^{-1}u)dt,$$

($t_k=\infty$ и $x(t_k)\rho x(t_k)=0$ в силу устойчивости).

В этом случае оптимальный закон получается стационарным ($\Gamma(t)=const$) $u=-kb^T\Gamma x$, а матрица Γ является решением уже алгебраического решения Риккати ($\dot{\Gamma}\to 0$, вынужденное решение)

$$\Gamma a+a^T\Gamma-\Gamma bkb^T\Gamma=-\beta\cdot$$

Метод А.А. Красовского получивший название АКР по критерию обобщенной работы был предложен Александром Аркадьевичем Красовским в 1968 году. Сущность метода заключается в видоизменении функционала качества

$$I=x^T(t_k)\rho x(t_k)+\int_{t_0}^{t_k}(x^T\beta x+u^T k^{-1}u+u_{опт}^T k^{-1}u_{опт})dt$$

путем добавления интегральной составляющей

$$\int_{t_0}^{t_k}u_{опт}^T k^{-1}u_{опт}dt\ ,$$

определяющей ограничения на управления уже в оптимальной системе.

Поскольку оптимальные управления на этапе задания требований к системе (функционала) еще не известны, то функционал получается полуопределенным, хотя и имеет ясный физический смысл.

Но такой искусственный прием позволил существенно снизить вычислительные трудности рассматриваемого метода.

Выведем основные расчетные соотношения для данного метода применительно для того же линейного объекта:

$$\dot{x}=ax+bu, \quad x(t_0)=0, x\in X, u\in U. \tag{2.1.10}$$

Вновь введем в рассмотрение положительно определенную квадратичную форму $V= x^T\Gamma x$ и определим её полную производную на уравнении движения объекта (2.1.10).

$$\dot{V}=\dot{x}^T\Gamma x+x^T\dot{\Gamma}x+x^T\Gamma\dot{x}=(x^Ta^T+u^Tb^T)\Gamma x+x^T\dot{\Gamma}x+x^T\Gamma(ax+bu).$$

Примем, что оптимальное управление имеет вид:

$$u_{опт} = -kb^T\Gamma x,$$

тогда:

$$\dot{V}=x^T(a^T\Gamma+\dot{\Gamma}+\Gamma a)x-u^Tk^{-1}u_{опт}-u_{опт}^Tk^{-1}u.$$

Очевидно, что $\dot{V}$ знакоотрицательна тогда и только тогда, когда выполняются два условия: $u = u_{опт}$ и $\dot{\Gamma}+\Gamma a+a^T\Gamma=-\beta,$ где β - положительно определенная матрица. Последнее уравнение нам уже известно, как уравнение Ляпунова, встречающееся в теории устойчивости.

Так, согласно второму методу Ляпунова его решение (матрица Г) будет положительно определенной (в соответствии с постановкой задачи оптимального управления) лишь в том случае, когда исходный объект (его матрица а) устойчива.

Таким образом, оптимизация по критерию обобщенной работы в отличие от метода Летова-Калмана, предполагает априорную устойчивость исходного объекта, что сужает область применения метода.

Докажем, что управление $u_{опт}$ доставляют минимум функционалу I. Для этого используем тот же прием, что и при доказательстве в методе Летова-Калмана.

$$\int_{t_0}^{t_k} \frac{dx^T \Gamma x}{dt} dt = -\int_{t_0}^{t_k} x^T \beta x dt - \int_{t_0}^{t_k} (u^T k^{-1} u_{опт}^T k^{-1} u) dt$$

$$\int_{t_0}^{t_k} x^T \beta x = -x^T(t_k)\Gamma(t_k)x(t_k) + x^T(t_0)\Gamma(t_0)x(t_0) -$$

$$- \int_{t_0}^{t_k} (u^T k^{-1} u_{опт} + u_{опт}^T k^{-1} u) dt$$

Подставим данное выражение в *I*

$$I = x^T(t_k)\rho x(t_k) - x^T(t_k)\Gamma(t_k)x(t_k) + x^T(t_0)\Gamma(t_0)x(t_0) -$$

$$- \int_{t_0}^{t_k} (u^T k^{-1} u_{опт} + u_{опт}^T k^{-1} u) dt + \int_{t_0}^{t_k} (u^T k^{-1} u + u_{опт}^T k^{-1} u_{опт}) dt =$$

$$= x^t(t_k)\rho x(t_k) - x^T(t_k)\Gamma(t_k)x(t_k) + \int_{t_0}^{t_k} (u - u_{опт})^T k^{-1} (u - u_{опт}) dt$$

Итак, получаем, что абсолютный минимум *I* достигается при $u=u_{опт}$ и $\Gamma(t_k)=\rho$. Вновь сформулируем полученный результат в виде теоремы.

Теорема А.А. Красовского. Для линейного детерминированного устойчивого объекта $\dot{x}=ax+bu, x(t_0)=0, x\in X, u\in U$ оптимальными в смысле минимума функционала

$$I=x^T(t_k)\rho x(t_k)+\int_{t_0}^{t_k}(x^T\beta x+u^T k^{-1}u+u_{опт}^T k^{-1}u_{опт})dt$$

являются уравнения $u=u_{опт}=-kb^T\Gamma(t)x,$ где Γ – положительно определенная симметричная матрица - решение линейного матричного дифференциального уравнения Ляпунова. $\dot{\Gamma}+\Gamma a+a^T\Gamma=-\beta,$ при краевом условии $\Gamma(t_k)=\rho$.

Отметим, что в случае нетерминальной задачи ($t_k=\infty$) управление $u_{опт}$ получается стационарным $u_{опт}=-kb^T\Gamma x$, а матрица Γ определяется из решения уже алгебраического уравнения Ляпунова $\Gamma a+a^T\Gamma=-\beta$, методы решения которого достаточно хорошо разработаны.

Проведя сравнительную оценку по трем позициям: класс оптимизируемых объектов, вычислительные трудности применения того или иного метода и возможность практической реализации синтезируемых управлений.

Область применения принципа максимума Понтрягина включает в себя широчайший класс нелинейных нестационарных объектов с достаточно общими ограничениями на фазовые координаты, управления, время и т.д. Однако для него характерны значительные вычислительные трудности, связанные прежде всего с необходимостью решения двухточечной краевой задачи для системы сопряженных уравнений.

Кроме того, оптимальные управления получаются в виде сложных нелинейных законов или программ, что не всегда удобно реализовать на практике.

В этой связи более предпочтительным оказывается динамическое программирование, в котором оптимальный закон управления получается в виде обратной связи по координатам состояния, что легко практически реализуемо.

Однако необходимость решения функционального уравнения Беллмана (в частных производных), неоднозначность получаемого решения приводит к существенным вычислительным трудностям.

Аналитическое конструирование Летова-Калмана-Красовского дает достаточно простой путь реализации оптимального закона управления (в виде аналитической линейной функции от координат объекта) и при этом обеспечиваются сравнительно невысокие вычислительные трудности, связанные с решением уравнений Риккати и Ляпунова.

Но в настоящее время эти методы в основном ограничиваются классом линейных объектов и достаточно узким кругом возможных ограничений.

2.2 Методы динамического программирования в дискретном случае

2.2.1Математическая модель задачи о рюкзаке

Задача о рюкзаке (англ. Knapsack problem) — одна из NP-задач комбинаторной оптимизации. Своё название приобрела от максимизационной задачи укладки как можно большего числа ценных вещей в рюкзак при условии, что общий объём (или вес) всех предметов, способных вместиться в рюкзак, ограничен.

Задачи о загрузке (о рюкзаке) и её модификации часто возникают в экономике, прикладной математике, криптографии, генетике и логистике для нахождения оптимальной загрузки транспорта (самолёта, поезда) или склада.

В общем виде задачу можно сформулировать так: из заданного множества предметов со свойствами «стоимость» и «вес», требуется отобрать некоторое число предметов таким образом, чтобы получить максимальную суммарную стоимость при одновременном соблюдении ограничения на суммарный вес.

Существует большое количество разновидностей задачи о рюкзаке. Отличия заключаются в условиях, наложенных на рюкзак, предметы или их выбор.

Пусть имеется набор предметов, каждый из которых имеет два параметра — вес и ценность. Также имеется рюкзак определенной вместимости.

Задача заключается в том, чтобы собрать рюкзак с максимальной ценностью предметов внутри, соблюдая при этом весовое ограничение рюкзака.

Предполагается, что вместимость рюкзака известна и равна G кг, и для каждого типа $i=\overline{1,m}$ предметов заданы вес q_i и ценность c_i. Не умоляя общности можно считать, что $z_1<z_2<\ldots<z_m$. Рассматриваемый вопрос об оптимальной загрузке рюкзака сводится к решению следующей задачи.

Задача. При заданных величинах G, q_i, c_i, $i=\overline{1,m}$ требуется максимизировать функцию

$$\mu(x)=\sum_{j=1}^{m} c_j x_j$$

на множестве векторов

$$x = (x_1, x_2, \ldots, x_m),$$

с целочисленными неотрицательными компонентами, удовлетворяющими условию

$$\sum_{i=1}^{m} q_i x_i \le G$$

В общем случае, математически задачу можно сформулировать так: имеется n грузов. Для каждого *i*-го груза определён вес $w_i > 0$ и ценность $p_i > 0$, i=1,2,...,n. Дана грузоподъёмность *W*. Необходимо выбрать подмножество грузов так, чтобы их общий вес не превышал *W*, а суммарная их ценность была бы максимальной.

Каждый предмет может быть выбран любое число раз. Задача выбрать количество x_i предметов каждого типа так, чтобы

- максимизировать общую стоимость;
- выполнялось условие совместности.

2.2.2 Методы решения задачи о рюкзаке

Как известно, в методе ветвей и границ имеется два момента алгоритмизации, определяемых спецификой задачи: разбиение исходного множества комбинаций на подмножества с дальнейшим выбором подмножества для очередного разбиения и вычисления нижних (верхних) границ (оценок) значений оптимизируемой функции на подмножествах. Разбиение множества на подмножества называется ветвлением, а выбор подмножества для разбиения – его стратегией. Вычисление ошибок толкуют как решение оценочных задач. Наглядным результатом ветвления и решения оценочных задач является n -ярусное корневое дерево поиска решений с оценками вершин подмножеств каждого яруса. Оценка вершины последнего яруса – рекорд – представляет собой текущее значение оптимизируемой функции, которое далее сравнивается с оценками вершин предшествующих ярусов, в результате чего неперспективные для ветвления подмножества отсеиваются, а перспективные разбиваются, дополняя дерево решений. Алгоритм заканчивает работу тогда, когда будут сравнены с рекордом все текущие и вновь порождаемые оценки вершин. Практика показывает, что более эффективными являются те алгоритмы, которые строят бинарные деревья решений, т. е. реализуют разбиение каждого очередного множества на два подмножества, выбор подмножества для ветвления осуществляют по

максимальной (минимальной) оценке оптимизируемой функции, оценочные задачи формируются так, что более точно вычисляются оценки. Поэтому при разработке алгоритма и наличии выбора следует придерживаться указанных правил.

Что касается рассматриваемой задачи, то процесс разбиения очередного множества осуществляется на два подмножества, первое из которых содержит комбинации компонент вектора с x_i= 0, второе – с x_i=1. Стратегия ветвления состоит в том, чтобы выбирать очередное подмножество для разбиения по максимальной оценке, верхней границы оптимизируемой функции. В результате получаем бинарное дерево поиска решений, изображённое на рисунке 2.2.1

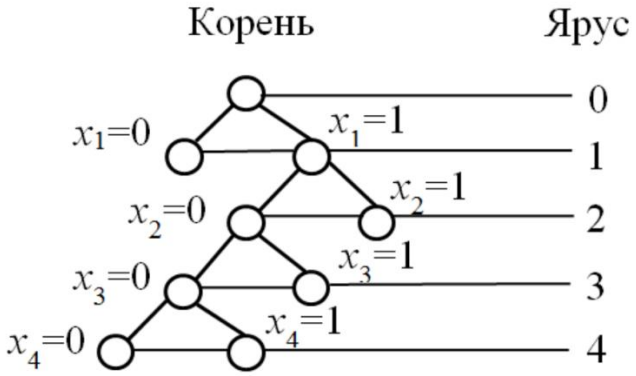


Рисунок 2.2.1 – Бинарное дерево поиска решений

Для вычисления оценок верхних границ оптимизируемой функции на подмножествах формулируется следующая оценочная задача:

Найти

$$Q = max \quad c_i\, x_i, \qquad (2.2.1)$$
$$i \in 1$$

при условиях

$$\sum_{i \in 1} a_i x_i \leq V. \qquad (2.2.2)$$

$$0 \leq x_i 1. i \in I. \qquad (2.2.3)$$

Иными словами, для некоторого x_i допускается, что оно не булево, а лежит в интервале [1 ,0]. Такая релаксация строгости условий задачи приводит к тому, что оценка Q для каждой вершины дерева поиска оказывается больше W, т. е. она действительно является верхней границей оптимизируемой функции на подмножествах комбинаций.

В то же время оценочная задача легко решаема. Согласно [16] следует найти «ценности» единиц весов грузов $\frac{c_i}{a_i}$, $i=1,2,\ldots,n$ и упорядочить их по убыванию $\frac{c_1}{a_1} \geq \frac{c_2}{a_2} \geq \cdots \geq \frac{c_n}{a_n}$. Тогда все $x_i \in I$, выбираемые в порядке последовательности «ценностей» и такие, что

$$\sum_{i=1}^{k} a_i < V,$$

полагаются равными единице. Очередное значение x_{k+1} такое, что

$$\sum_{i=1}^{k+1} a_i x_i > V,$$

вычисляется по выражению

$$x_{k+1} = (V - \sum_{i=1}^{k} a_i) / a_{k+1},$$

т.е. ищется значение той переменной x_i, $x_i \in I$, которая полагается не булевой. Остальные x_i, $i=k+2,\ldots,n$ принимается равными нулю. В результате получаем оценку

$$Q = \sum_{i=1}^{k+1} c_i x_i$$

и ограничение

$$\sum_{i=1}^{k+1} a_i < V.$$

Пусть для ветвления выбрана вершина r-го яруса дерева поиска с оценкой Q_r и значением V_r, которые определены последовательностью x_i, $i=k+2,\ldots,r$. Тогда оценочные задачи для вершины r+1-го яруса должны быть сформулированы так:

Найти

$$Q_{r+1} = \max(Q_r + \sum_{i=r+1}^{n} c_i x_i),$$

при условиях

$$\sum_{i=r+1}^{n} a_i x_i \leq V - V_r,$$

$$0 \leq x_i 1. i \in I.$$

Алгоритм, реализующий описанный метод, приведён ниже.

Шаг 1. Установить n; положить k=0 , рекорд R=0.

Шаг 2. Положить k=k+1.

Шаг 3. Положить x_k, вычислить оценку Q_k при x_k=0 запомнить Q_k.

Шаг 4. Положить x_k=1, вычислить оценку Q_k при x_k=1.

Шаг 5. Выбрать вершину для ветвления v_x с большей оценкой Q_k.

Шаг 6. Если $Q_k \le R$, перейти к шагу 9, иначе перейти к шагу 7.

Шаг 7. Если $k < n$, запомнить большее Q_k, соответствующее x_k, и вернуться к шагу 2; иначе перейти к шагу 8.

Шаг 8. Положить R равным большему Q_k. Запомнить вектор X.

Шаг 9. Положить k=k–1.

Шаг 10. Если k=0 , остановиться; иначе перейти к шагу 11.

Шаг 11. Выбрать запомненное Q_k.

Шаг 12. Если $Q_k \le R$, вернуться к шагу 9, иначе вернуться к шагу 3.

Основные процедуры метода ДП.

Рассмотрим W аналогичных задач

$$f(z) = \max_{x \in X_z} \mu(x), \quad z = 1,\dots, W;$$

X_z={ $x = (x_1, x_2, \dots, x_m) \mid$ $x_j \ge 0,$ x_j – целое, целые, i=1,2,…,m, $\sum_{i=1}^{m} w_i x_i \le z$ } - множество допустимых векторов.

При $z < w^0 = w_1$ множества X_z содержат единственный нулевой вектор, причем $f(z) = 0$ для всех таких z.

При $z \in [w^0, W]$ справедливы рекуррентные соотношения:

$$f(z) = \max_{i \in I_z} \{c_i + f(z - w_i)\}, I_z = \{i \mid w_i \le z\} \qquad (2.2.4)$$

1 этап (прямой ход).

Для всех $z \in [w^0, W]$ по формуле (2.2.4) последовательно вычисляют

$f(z)$ и фиксируют индексы i(z), на которых достигаются максимумы в (2.2.4) (если максимум достигается на нескольких индексах, в качестве $i(z)$ берется любой).

Процесс вычисления $f(z)$ удобно представлять в виде табл. 2.1.

Таблица 2.1 - Динамическая шкала

w^0	w^0+1	w^0+2	…	W
$f(w^0)$	$f(w^0+1)$	$f(w^0+2)$	…	$f(W)$
$i(w^0)$	$i(w^0+1)$	$i(w^0+2)$	…	$i(W)$

В конце табл. 2.1 находится искомое значение функции $f(W)$ и номер предмета $i(W)$, на котором оно достигнуто.

2 этап (обратный ход).

Определяется искомый вектор

$x=x_1,x_2,\ldots x_n$, $x_i>0$, целые, $i=1,2,\ldots,n$, при котором $\mu(x)=f(V)$.

В комплектуемый оптимальный набор в первую очередь включается предмет с номером: $i(z_1)$ $z_1=V$

Далее вычисляем: $z_2=z_1-v_{i(z_1)}$

Если $z_2<w^0=v_1$, то процесс заканчивается, иначе в указанный набор включается предмет с номером $i(z_2)$ $z_2-v_{i(z_2)}$.

Комплектование набора заканчивается на некотором шаге j, на котором

$$z_{j+1}<w^0.$$

Пример решения.

Исходные данные:

W =15 – вместимость рюкзака. Данные о предметах в табл.2.2.

Таблица 2.2 – данные о предметах

i	1	2	3
w_i	3	4	5
c_i	12	20	15

Решение

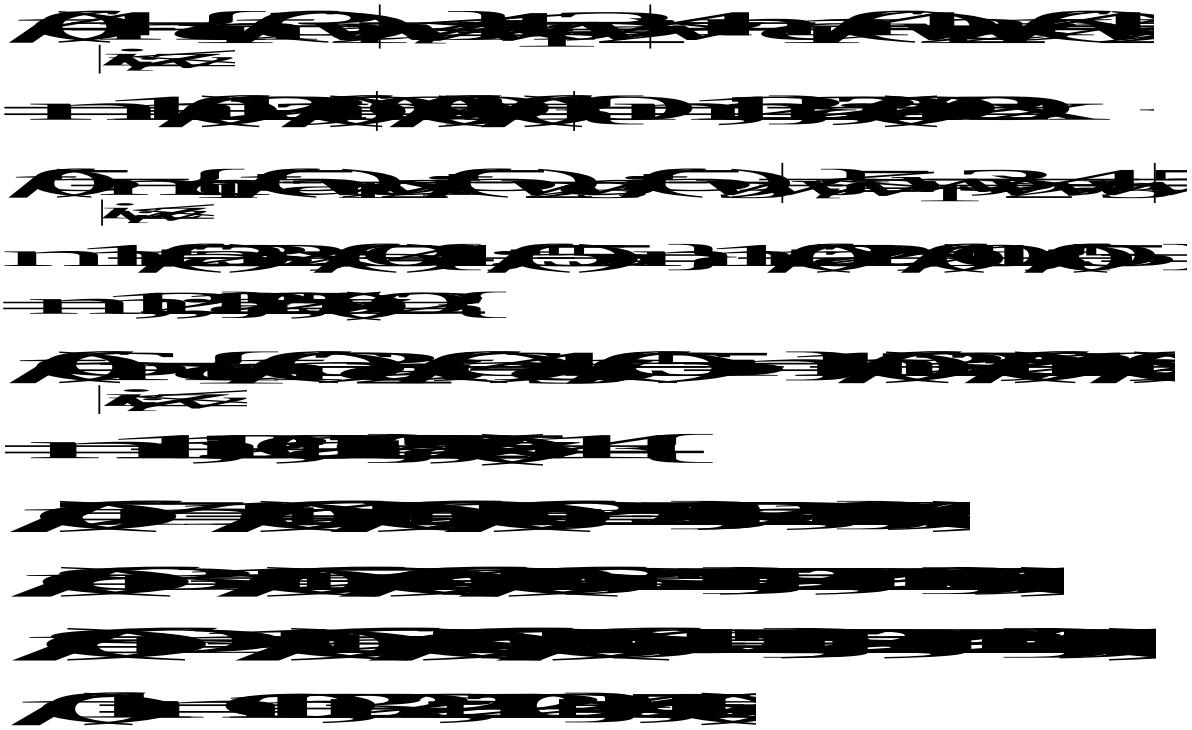
Определим величину $w^0 = \min_i w_i = \min\{3,4,5\} = 3;$

Прямой ход

$f(z) = 0$, если $z < w^0$, что означает – ценность рюкзака, вместимость которого $z < w^0$, равна нулю.

Пусть $z = w^0 = 3$. Найдем  .

Неравенству $w_i \le 3$ удовлетворяет только предмет 1-го вида, т.к. $w_1 = 3$  максимум достигается на 1-м предмете, значит $i(z) = 1$.



$f(11) = \max\{12 + 40; 20 + 32; 15 + 24\} = 52;\ i(z) = 1$

ГЛАВА 3 ПРОГРАММНАЯ РЕАЛИЗАЦИЯ

3.1 Решение задачи о рюкзаке

Решение задачи о рюкзаке методом динамического программирования.

Пусть *d(i,c)* - максимальная стоимость любого возможного числа предметов типов от 1 до i, суммарным весом до c. Заполним *d(0,c)* нулями.

Тогда меняя i от 1 до N, рассчитаем на каждом шаге *d(i,c)*, для c от 1 до W, по рекуррентной формуле: $d(i,c)=max(d(i-1,c-l_{wi})+l_{pi})$ по всем целым l из промежутка $0 \leq l \leq min(b_i, [c/w_i])$.

Можно использовать одномерный массив d(c) вместо двумерного. После выполнения в *d(N,W)* будет лежать максимальная стоимость предметов, помещающихся в рюкзак.

Решенных задач о загрузке рюкзака достаточное множество. Как показывают расчеты выше, за основу решения задачи взят алгоритм Беллмана-Форда, который в последствии был модифицирован добавлением просчета среднеквадратической ошибки, что в итоге позволило именно нашему варианту программы работать существенно быстрее (просчет среднеквадратической ошибки позволяет отбросить и не проходить ненужные циклы расчетов заново).

3.2 Описание интерфейса

В разработанной программе используется Model-View-Controller это схема разделения интерфейса и управляющей логики на три отдельных компонента: модель, представление и контроллер — таким образом, что модификация каждого компонента может осуществляться независимо.

- Модель (Model) предоставляет данные и реагирует на команды контроллера, изменяя свое состояние.
- Представление (View) отвечает за отображение данных модели пользователю, реагируя на изменения модели.
- Контроллер (Controller) интерпретирует действия пользователя, оповещая модель о необходимости изменений.

Схема проектируемой программы представлена на рисунке 3.1

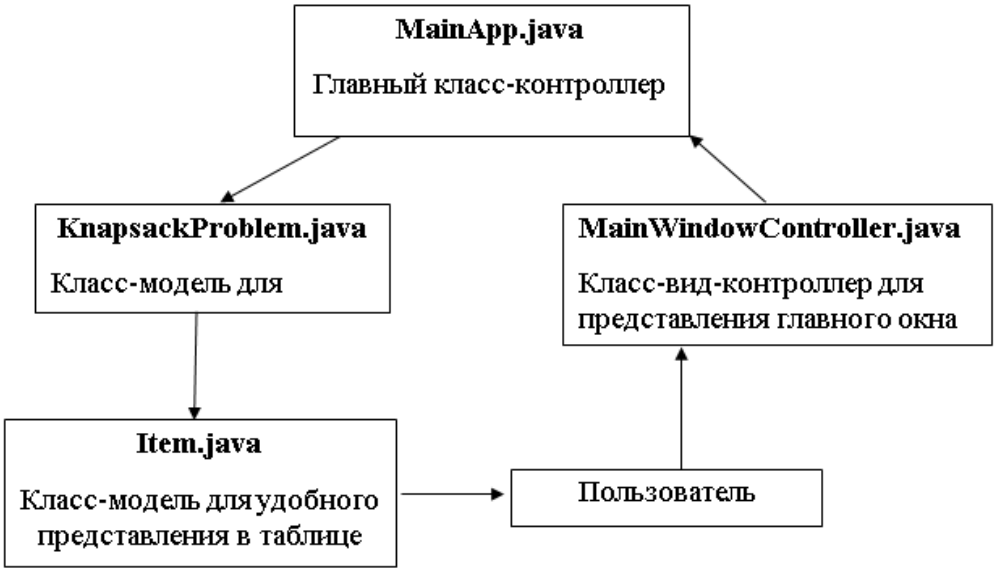


Рисунок 3.1 – Схема проектируемой программы

В силу некоторых особенностей языка Java на нем достаточно сложно программировать, не имея никаких специально предназначенных для этого инструментов. При этом IDE (Integrated Development Environment — интегрированная среда разработки) могут существенно облегчить программирование на Java. Это возможно благодаря наличию четкой спецификации того, что является программой на Java и как программа должна быть преобразована в дерево разбора.

Одной из наиболее популярных IDE является Eclipse, бесплатно распространяющийся организацией Eclipse Foundation. В данной среде и была разработана программа.

Основной алгоритм программы реализован в модуле: **KnapsackProblem**.java:

```
package org.knapsackproblem.model;
import java.util.ArrayList;
import java.util.List;
//Класс-модель для расчетов
public class KnapsackProblem {
    //Алгоритм укомплектования вещей
```

```
	public static List<Integer> getItems(int maxWeight, int[] weights, int[] prices)
{
		int h = 0;
		List<Integer> items = new ArrayList<Integer>();
		while (h < maxWeight) {
			int index = -1;
			int maxValue = 0;
			int size = weights.length;
			for (int i = 0; i< size; i++) {
				if (weights[i] * prices[i] > maxValue && h + weights[i] <=
maxWeight) {
					maxValue = weights[i] * prices[i];
					index = i;
				}
			}
			if (index == -1) {
				break;
			}
			else {
				h += weights[index];
				items.add(index);
			}
		}
		return items;
	}
}
```

Item.java

```
package org.knapsackproblem.model;

import javafx.beans.property.IntegerProperty;
```

```
import javafx.beans.property.SimpleIntegerProperty;
import javafx.beans.property.SimpleStringProperty;
import javafx.beans.property.StringProperty;
/**
 * Класс-модель для удобного представления в таблице
 *
 */
public class Item {
	//Имя
	private StringProperty name;
	//Вес
	private IntegerProperty weight;
	//Ценность
	private IntegerProperty value;
		public Item(String name, int weight, int value) {
		this.name = new SimpleStringProperty(name);
		this.weight = new SimpleIntegerProperty(weight);
		this.value = new SimpleIntegerProperty(value);
	}
		public String getName() {
		return name.get();
	}
```

```
        public void setName(String name) {
        this.name.set(name);
}
        public StringProperty nameProperty() {
        return name;
}

public int getWeight() {
        return weight.get();
}
        public void setWeight(int weight) {
        this.weight.set(weight);
}
        public IntegerProperty weightProperty() {
        return weight;
}
        public int getValue() {
        return value.get();
}
        public void setValue(int value) {
        this.value.set(value);
}
```

```
        public IntegerProperty valueProperty() {
        return value;
    }
}
```

Код остальных модулей программы представлен в Приложении. Программа написана в среде разработке Eclipse. Интерфейс пользователя представлен на рисунке 3.2.

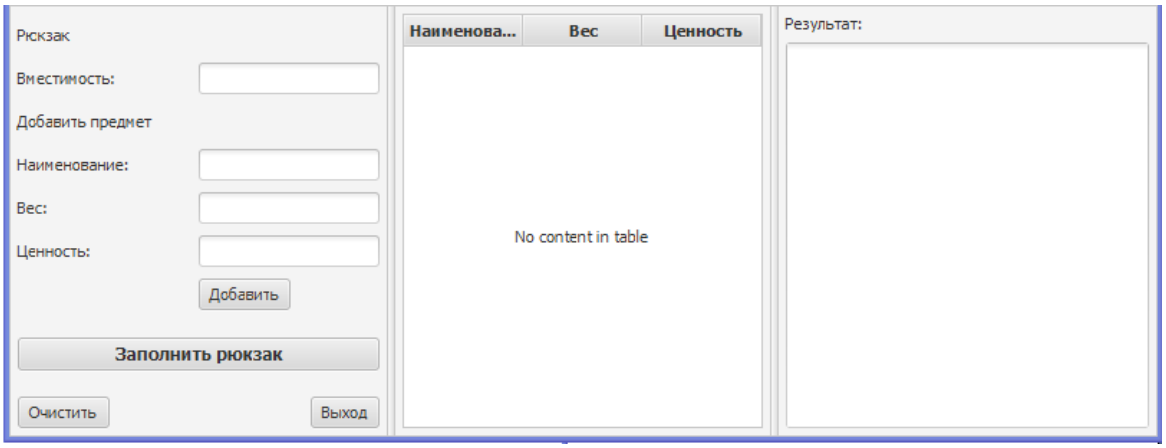


Рисунок 3.2 - Описание интерфейса пользователя

На рисунке 3.2 представлен интерфейс пользователя. Слева можно устанавливать предельный вес рюкзака, а также добавлять предметы. Чтобы добавить предмет в рюкзак, необходимо заполнить информацию о предмете и нажать кнопку «Добавить».

Предмет появится в таблице справа. При этом можно не указывать имя предмета, программа может автоматически нумеровать их. Нажав кнопку «Заполнить рюкзак», в окошке справа появится решение для текущего рюкзака.

Результат выполнения программы представлен на рисунке 3.3

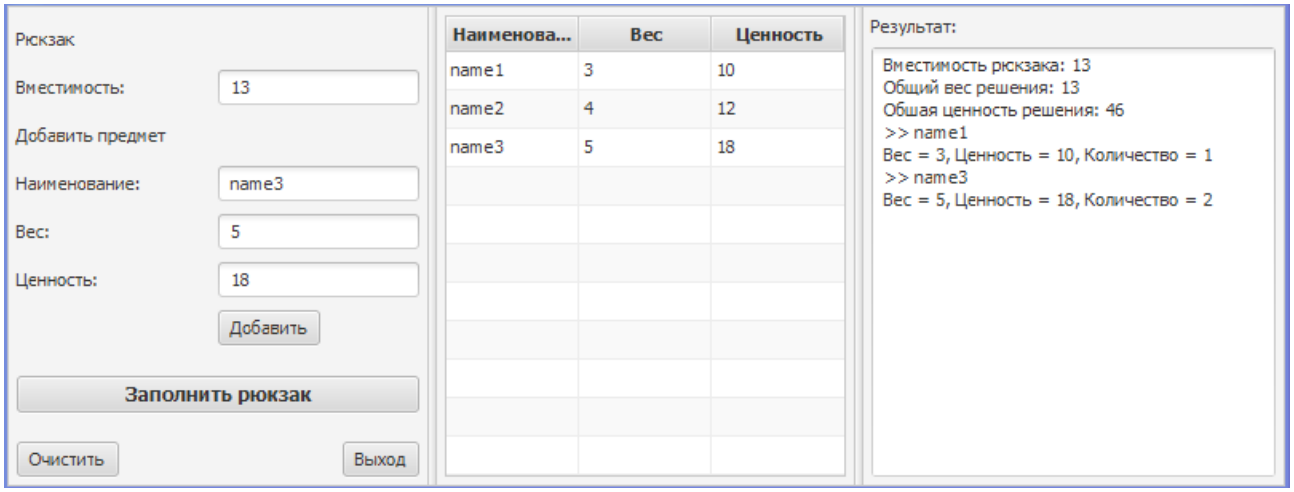


Рисунок 3.3 - Результат выполнения программы

После решения, можно либо дальше добавлять предметы в рюкзак и решать задачу для новых наборов предметов (рис 3.4), либо кнопкой «Очистить» вернуть программу в исходное состояние, как при запуске. По нажатию кнопки «Выход» приложение закроется.

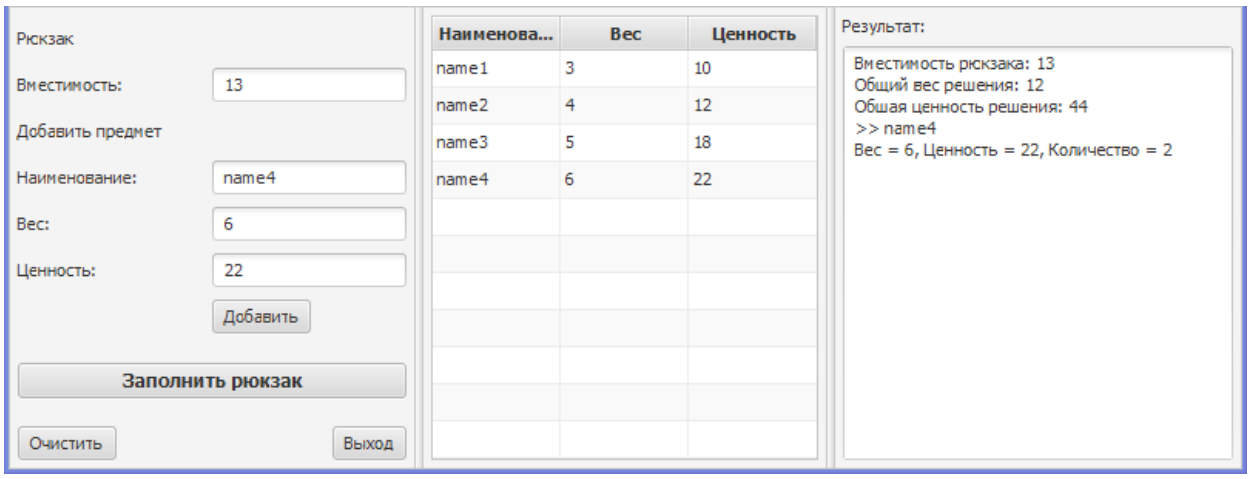


Рисунок 3.4 - Результат выполнения программы

В результате выполнения работы был разобран алгоритм, позволяющий решать задачу о загрузке рюкзака, и разработана программа на основе этого алгоритма.

ЗАКЛЮЧЕНИЕ

В выпускной квалификационной работе представлены общие подходы необходимости использования динамического программирования в некоторых прикладных задачах.

В настоящее время разработаны современные методы и алгоритмы решения задач дискретной оптимизации. Применение пакетов программ возможно без знания алгоритмов решения задач, однако знание алгоритмов и технологий их реализации позволяет более эффективно использовать прикладные программы. В этих программах используются алгоритмы, связанные с перебором большого числа вариантов, комбинаторные алгоритмы.

В работе подробно рассмотрены реализации динамического программирования в дискретном случае, а так же упомянуты реализации динамического программирования в непрерывном случае. Работа посвящена задаче о рюкзаке, методам её решения. Из них рассмотрены: метод ветвей и границ и метод динамического программирования. Программная реализация осуществлена для метода динамического программирования.

Задача о рюкзаке является довольно удачным примером демонстрации комбинаторных методов поиска оптимального решения метода динамического программирования.

Предложенная модификация алгоритма Беллмана-Форда, использованная в программе для решения задачи о рюкзаке, в отличии от существующих программ, позволяет проводить расчеты в несколько раз быстрее, исключая ненужные операции.

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

Научная и методическая литература

1. Акулич, И.Л. Математическое программирование в задачах и упражнениях. - М.: Высшая школа, 1993. - 215с.
2. Банди, Б. Основы линейного программирования. М.: Радио и связь. 1989. - 176с.
3. Беллман, Р. Дрейфус С. Прикладные задачи динамического программирования. – М.: Наука, 1965. - 375с.
4. Вагнер, Г. Основы исследования операций. Т. 1. - М. Мир, 1972, 1973.
5. Воройский, Ф.С. Информатика. Новый систематизированный толковый словарь / Ф.С. Воройский. - М.: [не указано], 2017. - 150 с.
6. Гермейер, Ю.Б. Введение в теорию исследования операций. - М.: Наука, 1976. – 382с.
7. Замкова, Л.И. Булева двухкритериальная задача о рюкзаке / Л.И. Замкова // Известия Южного федерального университета. Технические науки. – 2009. – № 4. – С. 201 – 204.
8. Интрилигатор, М. Математические методы оптимизации и экономическая теория. - М.: Прогресс, 1975. – 592с.
9. Казаков, А.Я. Модифицированные модели Джейнса-Каммингса и квантовая версия задачи о рюкзаке / А.Я. Казаков // Журнал экспериментальной и теоретической физики. – 2003. – Т. 124, Вып. 6 (12). – С. 1264–1270.
10. Конюховский, П.В. Математические методы исследования операций: пособие для подготовки к экзамену. – СПб.:Питер, 2001. – 192с.
11. Корбут, А.А. Дискретное программирование. / А.А. Корбут, Ю.Ю. Финкельштейн. – М.: Наука, 1969. – 368 с.
12. Морозов, В.В., Сухарев А.Г., Федоров В.В. Исследование операций в задачах и упражнениях. - М.: Высшая школа, 1986.
13. Пападимитриу, Х. Комбинаторная оптимизация. Алгоритмы и сложность / Х. Пападимитриу, К. Стайглиц. – М.: Мир, 1985. – 510 с.

14. Саати, Т. Целочисленные методы оптимизации и связанные с ними экстремальные проблемы. / Т. Саати. – М.: Мир, 1973. – 304 с.

15. Таха, Х. Введение в исследование операций. Т.1, 2. - М.: Мир, 1985.

16. Шкурба, В.В. Задачи трёх станков / В.В. Шкурба. – М.: Наука, 1976. – 96 с.

Литература на иностранном языке

17. Fletcher R. Practical Methods of Optimization. Vol. 1, Unconstrained Optimization, and Vol. 2, Constrained Optimization, John Wiley and Sons., 1980.

18. Gembicki F.W. Vector Optimization for Control with Performance and Parameter Sensitivity Indices. Ph.D. Dissertation, Case Western Reserve Univ., Cleveland, Ohio, 1974.

19. Gill P.E., W. Murray, M.A. Saunders, and M.H. Wright. Procedures for Optimization Problems with a Mixture of Bounds and General Linear Constraints. ACM Trans. Math. Software, Vol. 10, pp. 282-298, 1984.

20. Gilmore P.C. The Theory and Computation of Knapsack Functions / P.C. Gilmore, R.E. Gomory // Operations Research. – 1966. – Vol. 14, No. 6. – pp. 1045 – 1074.

21. Zadeh L.A. Optimality and Nonscalar-valued Performance Criteria. IEEE Trans. Automat. Contr., Vol. AC-8, p. 1, 1963.

ПРИЛОЖЕНИЕ А

Листинг кода для реализации задачи о рюкзаке методом динамического программирования

MainApp.java

```
package org.knapsackproblem;

import java.io.IOException;
import java.util.ArrayList;
import java.util.List;

import org.knapsackproblem.model.Item;
import org.knapsackproblem.model.KnapsackProblem;
import org.knapsackproblem.view.MainWindowController;

import javafx.application.Application;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.fxml.FXMLLoader;
import javafx.scene.Scene;
import javafx.scene.control.SplitPane;
import javafx.scene.layout.BorderPane;
import javafx.stage.Stage;

/**
 * Главный класс-контроллер
 *
 */
public class MainApp extends Application {

	//Сцена приложения
	private Stage primaryStage;

	//Данные из таблицы
	private ObservableList<Item> items = FXCollections.observableArrayList();

	//Ссылка на вид-контроллер главного окна
	private MainWindowController controller;

	public ObservableList<Item> getItems() {
		return items;
```

```
    }

    //Заполнить рюкзак елементами
    public void fillInventory(int capacity) {
        int size = items.size();
        int[] weights = new int[size];
        int[] values = new int[size];
        for (int i = 0; i < size; i++) {
            weights[i] = items.get(i).getWeight();
            values[i] = items.get(i).getValue();
        }

        List<Integer> filledItems = KnapsackProblem.getItems(capacity,
weights, values);

        StringBuilder str = new StringBuilder();
        str.append("Вместимость рюкзака: " + capacity + "\n");

        int weightResult = 0;
        int valueResult = 0;
        size = filledItems.size();
        for (int i = 0; i < size; i++) {
            weightResult += items.get(filledItems.get(i)).getWeight();
            valueResult += items.get(filledItems.get(i)).getValue();
        }

        str.append("Общий вес решения: " + weightResult + "\n");
        str.append("Обшая ценность решения: " + valueResult + "\n");

        int[] repetitions = new int[items.size()];
        for (int i = 0; i < items.size(); i++) {
            repetitions[i] = 0;
        }
        for (int i = 0; i < size; i++) {
            repetitions[filledItems.get(i)]++;
        }

        for (int i = 0; i < items.size(); i++) {
            if (repetitions[i] != 0) {
                Item item = items.get(i);
                str.append(">> " + item.getName() + "\n");
                str.append("Вес = " + item.getWeight() + ", Ценность = " +
item.getValue() + ", Количество = "
                        + repetitions[i] + "\n");
```

```
                }
            }

            controller.setResult(str.toString());
        }

        public Stage getPrimaryStage() {
            return primaryStage;
        }

        @Override
        public void start(Stage primaryStage) {
            this.primaryStage = primaryStage;
            this.primaryStage.setTitle("Задача о рюкзаке");

            initRootPane();
        }

        //Инициализировать главное окно
        private void initRootPane() {
            try {
                FXMLLoader loader = new FXMLLoader();

        loader.setLocation(MainApp.class.getResource("view/MainWindow.fxml"));
                SplitPane rootPane = (SplitPane) loader.load();

                controller = loader.getController();
                controller.setMainApp(this);

                primaryStage.setScene(new Scene(rootPane));
                primaryStage.show();
            }
            catch (IOException e) {
                e.printStackTrace();
            }
        }

        public static void main(String[] args) {
            launch(args);
        }
}
```

Item.java

```
package org.knapsackproblem.model;

import javafx.beans.property.IntegerProperty;
import javafx.beans.property.SimpleIntegerProperty;
import javafx.beans.property.SimpleStringProperty;
import javafx.beans.property.StringProperty;

/**
 * Класс-модель для удобного представления в таблице
 *
 */
public class Item {

	//Имя
	private StringProperty name;
	//Вес
	private IntegerProperty weight;
	//Ценность
	private IntegerProperty value;

	public Item(String name, int weight, int value) {
		this.name = new SimpleStringProperty(name);
		this.weight = new SimpleIntegerProperty(weight);
		this.value = new SimpleIntegerProperty(value);
	}

	public String getName() {
		return name.get();
	}

	public void setName(String name) {
		this.name.set(name);
	}

	public StringProperty nameProperty() {
		return name;
	}

	public int getWeight() {
		return weight.get();
	}

	public void setWeight(int weight) {
```

```
		this.weight.set(weight);
	}

	public IntegerProperty weightProperty() {
		return weight;
	}

	public int getValue() {
		return value.get();
	}

	public void setValue(int value) {
		this.value.set(value);
	}

	public IntegerProperty valueProperty() {
		return value;
	}
}
```

KnapsackProblem.java

```
package org.knapsackproblem.model;

import java.util.ArrayList;
import java.util.List;

//Класс-модель для расчетов
public class KnapsackProblem {

	//Алгоритм укомплектования вещей
	public static List<Integer> getItems(int maxWeight, int[] weights, int[] prices)
{
		int h = 0;
		List<Integer> items = new ArrayList<Integer>();
		while (h < maxWeight) {
			int index = -1;
			int maxValue = 0;
			int size = weights.length;
			for (int i = 0; i< size; i++) {
				if (weights[i] * prices[i] > maxValue && h + weights[i] <=
maxWeight) {
					maxValue = weights[i] * prices[i];
```

```
                                index = i;
                        }
                }
                if (index == -1) {
                        break;
                }
                else {
                        h += weights[index];
                        items.add(index);
                }
        }
        return items;
    }
}
```

MainWindowController.java

```
package org.knapsackproblem.view;

import org.knapsackproblem.MainApp;
import org.knapsackproblem.model.Item;

import javafx.fxml.FXML;
import javafx.scene.control.Alert;
import javafx.scene.control.Alert.AlertType;
import javafx.scene.control.TableColumn;
import javafx.scene.control.TableView;
import javafx.scene.control.TextArea;
import javafx.scene.control.TextField;

/**
 * Класс-вид-контроллер для представления главного окна
 * @author THROWABLE
 *
 */
public class MainWindowController {

    //Размер инвентаря
    @FXML
    private TextField capacityField;
    //Имя предмета
    @FXML
    private TextField nameField;
    //Вес предмета
```

```
        @FXML
        private TextField weightField;
        //Ценность предмета
        @FXML
        private TextField valueField;

        //Таблица предметов
        @FXML
        private TableView<Item> itemsView;
        @FXML
        private TableColumn<Item, String> nameColumn;
        @FXML
        private TableColumn<Item, Integer> weightColumn;
        @FXML
        private TableColumn<Item, Integer> valueColumn;

        //Поле для вывода результата
        @FXML
        private TextArea resultArea;

        //Ссылка на главный класс-контроллер
        private MainApp mainApp;

        public void setResult(String result) {
                resultArea.setText(result);
        }

        public void setMainApp(MainApp mainApp) {
                this.mainApp = mainApp;
                itemsView.setItems(mainApp.getItems());
        }

        //Метод инициализации
        @FXML
        private void initialize() {
                nameColumn.setCellValueFactory((cellData) ->
cellData.getValue().nameProperty());
                weightColumn.setCellValueFactory((cellData) ->
cellData.getValue().weightProperty().asObject());
                valueColumn.setCellValueFactory((cellData) ->
cellData.getValue().valueProperty().asObject());
        }

        //Обработчик для кнопки добавить
```

```
    @FXML
    private void handleAdd() {
        if (isInputValidForAdd()) {
            String name = nameField.getText();
            int weight = Integer.parseInt(weightField.getText());
            int value = Integer.parseInt(valueField.getText());

            mainApp.getItems().add(new Item(name, weight, value));
        }
    }

    //Обработчик для кнопки расчета
    @FXML
    private void handleCalculate() {
        if (isInputValidForFillInventory()) {
            int capacity = Integer.parseInt(capacityField.getText());
            mainApp.fillInventory(capacity);
        }
    }

    //Обработчик для кнопки очистить
    @FXML
    private void handleClear() {
        mainApp.getItems().clear();
        resultArea.setText("");
        capacityField.setText("");
        nameField.setText("");
        weightField.setText("");
        valueField.setText("");
    }

    //Обработчик для кнопки выхода
    @FXML
    private void handleExit() {
        System.exit(0);
    }

    //Проверка правильности ввода при нажатии на кнопку расчета
    private boolean isInputValidForFillInventory() {
        StringBuilder errMessage = new StringBuilder();
        if (capacityField.getText() == null || capacityField.getText().isEmpty()) {
            errMessage.append("Вместимость инвентаря не может быть
пустым\n");
        }
```

```
            else {
                try {
                    Integer.parseInt(capacityField.getText());
                }
                catch (NumberFormatException e) {
                    errMessage.append("Вместимость инвентаря должно
быть целым числом\n");
                }
            }

            if (errMessage.toString().isEmpty()) {
                return true;
            }
            else {
                Alert alert = new Alert(AlertType.WARNING);
                alert.initOwner(mainApp.getPrimaryStage());
                alert.setTitle("Ошибка");
                alert.setHeaderText("Неправильно введенные данные");
                alert.setContentText(errMessage.toString());

                alert.showAndWait();
                return false;
            }
        }

        //Проверка правильности ввода при добавлении предмета
        private boolean isInputValidForAdd() {
            StringBuilder errMessage = new StringBuilder();
            if (nameField.getText() == null || nameField.getText().isEmpty()) {
                errMessage.append("Наименование не должно быть
пустым\n");
            }
            if (weightField.getText() == null || weightField.getText().isEmpty()) {
                errMessage.append("Вес предмета не может быть пустым\n");
            }
            else {
                try {
                    Integer.parseInt(weightField.getText());
                }
                catch (NumberFormatException e) {
                    errMessage.append("Вес предмета должен быть целым
числом\n");
                }
            }
```

```
            if (valueField.getText() == null || valueField.getText().isEmpty()) {
                errMessage.append("Ценность предмета не может быть
пустым\n");
            }
            else {
                try {
                    Integer.parseInt(valueField.getText());
                }
                catch (NumberFormatException e) {
                    errMessage.append("Ценность предмета должно быть
целым числом\n");
                }
            }

            if (errMessage.toString().isEmpty()) {
                return true;
            }
            else {
                Alert alert = new Alert(AlertType.WARNING);
                alert.initOwner(mainApp.getPrimaryStage());
                alert.setTitle("Ошибка");
                alert.setHeaderText("Неправильно введенные данные");
                alert.setContentText(errMessage.toString());

                alert.showAndWait();
                return false;
            }
        }
}
```