**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ «ГОМЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ИМЕНИ П. О. СУХОГО»**

Факультет автоматизированных и информационных систем

Кафедра «Информационные технологии»

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту

по дисциплине «Оптимизация проектных решений»

на тему: «**Оптимизация проектного решения**»

Исполнитель: студент гр. ИТП-41

М.А. Латышева

Руководитель: профессор

М.И. Михайлов

Дата проверки: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата допуска к защите: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата защиты: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Оценка работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Подписи членов комиссии

по защите курсового проекта: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

2023

**СОДЕРЖАНИЕ**

[Введение 4](#_Toc152928991)

[1. Анализ предметной области 5](#_Toc152928992)

[1.1.Оптимизация проектного решения 5](#_Toc152928993)

[1.2.Постановка задачи оптимизации проектного решения 7](#_Toc152928994)

[1.3.Многокритериальная оптимизация 9](#_Toc152928995)

[1.4.Язык программирования *Python* 11](#_Toc152928996)

[2. Математическая модель задачи 14](#_Toc152928997)

[2.1.Теоретические основы задачи об оптимальных назначениях 14](#_Toc152928998)

[2.2.Математическая модель задачи об оптимальных назначениях 16](#_Toc152928999)

[2.3.Теоретические основы задачи о рюкзаке 19](#_Toc152929000)

[2.4.Математическая модель задачи о рюкзаке 22](#_Toc152929001)

[2.5.Математическая модель многокритериальной оптимизации 26](#_Toc152929002)

[3. Програмная реализация поставленной задачи 28](#_Toc152929003)

[3.1.Описание разработанного приложения 28](#_Toc152929004)

[3.2.Запуск программного продукта 28](#_Toc152929005)

[3.3.Верификация программного продукта 29](#_Toc152929006)

[3.4.Тестирование программы 31](#_Toc152929007)

[Заключение 34](#_Toc152929008)

[Список использованных источников 35](#_Toc152929009)

[Приложение А. Листинг программы 36](#_Toc152929010)

[Приложение Б. Графические схемы алгоритмов 40](#_Toc152929011)

[Приложение В. Схемы алгоритмов 55](#_Toc152929012)

# **ВВЕДЕНИЕ**

Оптимизация проектного решения является важным аспектом в процессе разработки и реализации проектов. Она направлена на достижение наилучших результатов и оптимального использования ресурсов при решении различных задач.

В современном бизнес и техническом окружении, оптимизация проектного решения играет важную роль в достижении успеха. Она позволяет снизить издержки, улучшить производительность и удовлетворить потребности клиентов и пользователей.

Оптимизация проектного решения включает в себя анализ и оценку различных вариантов, поиск наилучших альтернатив, определение оптимальных параметров и настройку процессов. Применяется в различных областях, включая инженерное дело, информационные технологии, производство, финансы, логистику и другие.

В данной работе будут рассмотрены основные принципы и методы оптимизации для задач об оптимальных назначениях и задач о рюкзаке. Будет рассмотрена многокритериальность данных задач и способы учета различных критериев при поиске оптимального решения. Также будет рассмотрен применяемый метод решения для каждой из задач: динамическое программирование для задачи об оптимальных назначениях и линейное программирование для задачи о рюкзаке.

Оптимизация проектного решения в таких задачах позволяет достичь наилучших результатов, удовлетворяющих ограничениям и критериям, и способствует повышению эффективности и качества проекта.

1. **АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ**
   1. **Оптимизация проектного решения**

Оптимизация проектного решения – это процесс улучшения и оптимизации проекта, направленный на достижение наилучших результатов с учетом имеющихся ресурсов, ограничений и целей проекта. Оптимизация включает в себя анализ, моделирование и принятие решений с целью повышения эффективности, улучшения качества и снижения затрат проекта. Оптимизация может быть применена в различных областях, включая производство, инженерное дело, информационные технологии, логистику и другие. Оптимизация является важным инструментом для успешной реализации проектов и достижения конкурентных преимуществ.

В современном бизнес и техническом окружении, оптимизация проектного решения играет важную роль в достижении успеха. Оптимизация позволяет снизить издержки, улучшить производительность и удовлетворить потребности клиентов и пользователей.

Оптимизация проектного решения включает в себя анализ и оценку различных вариантов, поиск наилучших альтернатив, определение оптимальных параметров и настройку процессов. Применяется в различных областях, включая инженерное дело, информационные технологии, производство, финансы, логистику и другие [1].

При решении определенной оптимизационной задачи исследователь должен выбрать математический метод, приводящий к конечным результатам, прибегая к наименьшим затратам на вычисления или же давая возможность получить как можно больше информации об искомом решении. Выбор того или иного метода в большей степени определяется постановкой задачи оптимизации, а также математической моделью объекта оптимизации, которую используют в выбранном методе. Для решения оптимальных задач используют в основном следующие методы:

* методы исследования функций классического анализа;
* методы, основанные на использовании неопределенных множителей Лагранжа;
* вариационное исчисление;
* динамическое программирование;
* принцип максимума;
* линейное программирование;
* нелинейное программирование.

В настоящее время разработан метод геометрического программирования, он успешно применяется для решения определенного класса задач. Как правило, невозможно выбрать какой-то один метод, который можно применять для решения всех абсолютно задач, возникающих во время практики.

Некоторые методы в решении задач являются более общими, другие более узкими. Большую группу методов, таких как, метод множителей Лагранжа, методы нелинейного программирования, методы исследования функций классического анализа, на каких-либо определенных шагах решения задачи оптимизации можно применять, сочетая с другими методами, например с принципом максимума или динамическим программированием.

Некоторые из методов специально проработаны или наилучшим образом подходят для решения оптимизационных задач с определенного вида математическими моделями. Например, математический аппарат линейного программирования, создан исключительно для решения задач с линейными ограничениями на переменные и линейными критериями оптимальности и позволяет решать класс задач, с формулировкой в такой постановке. Так же геометрическое программирование предназначено для того, чтобы решать оптимальные задачи, где критерий ограничения и оптимальности представляются с помощью специального вида функций полиномов.

Динамическое программирование хорошо приспособлено для решения задач оптимизации многостадийных процессов, особенно тех, в которых состояние каждой отдельной стадии характеризуется относительно маленьким числом переменных состояния. Однако применение метода динамического программирования при избытке значительного числа переменных, что значит при высокой размерности, достаточно затруднительно, так как скорость работы и объем памяти ограничен. Пожалуй, наилучшим способом при выборе метода оптимизации, наиболее применимого для решения соответствующей этому методу задачи, следует признать исследование возможностей и опыта применения разновидных методов оптимизации.

Цель оптимизации проектного решения заключается в улучшении совокупной эффективности проекта, достижении наилучших результатов при ограниченных ресурсах и сокращении времени и затрат на реализацию проекта. Путем оптимизации можно повысить конкурентоспособность продукта или услуги, улучшить удовлетворенность клиентов и повысить рентабельность проекта.

* 1. **Постановка задачи оптимизации проектного решения**

Постановка задачи оптимизации проектного решения включает определение переменных решения, целевой функции, ограничений и функций ограничений. Переменные решения являются изменяемыми параметрами проекта, а целевая функция определяет критерии оптимизации, которые нужно максимизировать или минимизировать. Ограничения устанавливают ограничения на ресурсы, бюджет, время или другие факторы, а функция ограничений определяет допустимые значения переменных решения, учитывая ограничения.

Любой объект проектирования может быть представлен следующим образом, рисунок 1.1.

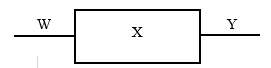


Рисунок 1.1 – Объект проектирования

где *Y* – вектор выходных параметров, описывающих функционирование системы, то есть взаимодействие с внешней средой;

*W* – вектор входных воздействий, то есть непосредственно воздействий внешней среды;

*X* – вектор внутренних характеристик системы, то есть параметров ее элементов.

Взаимосвязь этих векторов определяется соотношением:

*Y* = *F*(*X*,*W*), (1.1)

где вид оператора *F* определяется структурой и предметной областью объекта. В процессе проектирования требуется выбрать такое значение вектора *X*\*, чтобы при известном значении вектора *W*, выходные характеристики *Y* соответствовали требованиям. На значения вектора *X* обычно налагаются условия реализуемости, которые имеют вид:

*a* ≤ *X* ≤ *b*, (1.2)

На основании формул (1.1) и (1.2) есть возможность сформулировать два класса задач проектирования:

* задачи выбора оптимальных проектных решений;
* задачи выбора удовлетворительных проектных решений.

Кроме непосредственного поиска проектных решений математические модели в составе системы автоматизированного проектирования (САПР) используются для теоретической оценки эффективности выбранных проектных решений. В САПР математические методы разделяются на два класса: методы прямых расчетов и методы оптимизации. Эти методы существенно связаны с методами вычислительной математики и численными методами оптимизации, и методами прикладной и вычислительной геометрий.

Задача оптимизации состоит в поиске наилучшего решения, которое удовлетворяет требованиям и целям проекта. Методы оптимизации стремятся найти значения переменных решения, которые минимизируют или максимизируют целевую функцию, при соблюдении всех ограничений.

Решение задачи оптимизации может быть представлено в виде набора значений переменных решения, которые оптимальны с точки зрения заданных критериев оптимизации и удовлетворяют ограничениям. Это может быть достигнуто с использованием различных методов оптимизации, таких как методы линейного и динамического программирования, градиентного спуска и другие. Некоторые из наиболее распространенных методов оптимизации:

1. метод градиентного спуска – это итерационный метод, который использует градиент целевой функции для поиска локального минимума. Методы на основе градиентного спуска могут быть эффективными в задачах с непрерывными переменными и дифференцируемыми целевыми функциями;
2. методы на основе эволюционных алгоритмов – методы основаны на принципах биологической эволюции и генетического программирования. Они работают с популяциями решений, применяя операторы мутации, скрещивания и отбора, чтобы найти оптимальное решение. Эволюционные алгоритмы могут быть полезны в задачах с дискретными или нелинейными целевыми функциями и ограничениями;
3. методы оптимизации на основе симуляции отжига основаны на принципах термодинамики и процесса охлаждения материала. Методы симуляции отжига исследуют пространство переменных с помощью случайных изменений и принимают решения с некоторой вероятностью, основанной на разнице между текущим и новым решениями;
4. методы глобальной оптимизации, стремятся найти глобальный оптимум в пространстве переменных, а не ограничиваться только локальными оптимумами. Они могут использовать различные стратегии, такие как разбиение пространства переменных на подобласти или случайные поисковые методы;
5. математическое программирование – это обширный класс методов, который включает линейное программирование, целочисленное программирование, нелинейное программирование и другие. Математическое программирование позволяет формализовать задачу оптимизации с помощью математических моделей и найти оптимальное решение.

При выборе метода оптимизации проектного решения необходимо учесть различные ограничения, которые могут оказать влияние на возможность и эффективность применения конкретного метода.

Ограничения, которые следует учитывать, включают размерность задачи, тип целевой функции, наличие ограничений, доступность вычислительных ресурсов и экономическую эффективность применения метода. Размерность задачи может влиять на эффективность метода, так как некоторые методы могут быть неэффективными при работе с задачами большой размерности. Тип целевой функции также важен, так как разные методы оптимизации могут быть эффективными для разных типов функций. Наличие ограничений, таких как ограничения на ресурсы или технические требования, требует выбора метода, который может учитывать эти ограничения. Доступность вычислительных ресурсов также важна, так как некоторые методы могут требовать больших вычислительных мощностей или времени. Экономическая эффективность метода также необходимо учитывать, так как некоторые методы могут быть более дорогостоящими или требовать специализированного программного обеспечения.

При выборе метода оптимизации следует учитывать все эти факторы и балансировать между эффективностью, доступностью ресурсов и экономической эффективностью.

Оптимизация проектного решения может применяться во многих областях, включая инженерное проектирование, производство, логистику, финансы и другие. Цель состоит в том, чтобы найти наилучшее решение, которое оптимизирует использование ресурсов, улучшает эффективность и достигает поставленных целей проекта.

* 1. **Многокритериальная оптимизация**

Методы решения задач линейного программирования с одним критерием интенсивно разрабатывались в течение нескольких десятков лет. Но вскоре, по мере развития информатики и технологии, до сегодняшнего дня, можно с уверенностью сказать, что любая серьезная проблема характеризуется больше, чем одним критерием. Можно понять, что задачи многокритериальной оптимизации (МО) возникают в тех случаях, когда имеется несколько целей, которые не могут быть отражены одним критерием (например, стоимость, надежность и т. п.). Лица, принимающие решения, в значительно большей степени, чем когда-либо, ощущают необходимость оценивать альтернативные решения с точки зрения нескольких критериев [2].

Результаты исследования задач планирования и управления показывают, что в реальной постановке эти задачи являются многокритериальными. Так, часто встречающееся выражение «достичь максимального эффекта при наименьших затратах» уже означает принятие решения при двух критериях. Оценка деятельности предприятий и планирования как системы принятия решений производится на основе более десятка критериев: выполнение плана производства по объему, по номенклатуре, плана реализации, прибыли по показателям рентабельности, производительности труда и т.д.

Ранее, при исследовании проблемы многокритериальности часто все критерии, кроме одного, выбранного доминирующим, принимались в качестве ограничений, оптимизация проводилась по доминирующему критерию. Такой подход к решению практических задач значительно снижает эффективности принимаемых решений. Здесь требуется найти точку области допустимых решений, которая минимизирует или максимизирует все эти критерии. В связи с этим, исследователи начали развивать имеющиеся теоритические и практические результаты методов решения задач с одним критерием таким образом, чтобы они были применимы к исследованию многокритериальных задач линейного программирования.

Ввиду этого в теории многокритериальной оптимизации понятие оптимальности получает различные толкования, и поэтому сама теория содержит три основных направления:

* разработка концепции оптимальности;
* доказательство существования решения, оптимального в соответствующем смысле;
* разработка методов нахождения оптимального решения.

Многокритериальная задача отличается от обычной задачи оптимизации только наличием нескольких целевых функций вместо одной. Но в отличие от задач оптимизации с одним критерием в многокритериальной оптимизации имеется неопределенность целей. Действительно, существование решения, максимизирующего несколько целевых функций, является редким исключением, поэтому с математической точки зрения задачи многокритериальной оптимизации являются неопределенными и решением может быть только компромиссное решение.

* 1. **Язык программирования** ***Python***

Язык программирования *Python* – это высокоуровневый язык программирования, который был разработан в конце 1980-х годов и впервые выпущен в 1991 году Гвидо ван Россумом. *Python* был создан с упором на простоту чтения и понимания кода, а также на повышение производительности программиста путем предоставления удобных и эффективных инструментов.

Одной из основных черт *Python* является его ясность и чистота кода. Синтаксис *Python* позволяет программистам писать лаконичный и понятный код, что облегчает его чтение и поддержку. Благодаря этому *Python* считается одним из самых читаемых языков программирования [3].

*Python* предлагает обширную стандартную библиотеку, которая включает в себя множество модулей и инструментов для различных задач, таких как работа с файлами, сетевое взаимодействие, обработка строк и многое другое. Богатая стандартная библиотека является одним из ключевых преимуществ *Python*, которые позволяют разработчикам быстро создавать функциональные программы без необходимости писать все с нуля.

*Python* также имеет большое количество сторонних библиотек и фреймворков, которые значительно расширяют его возможности. Например, *NumPy* и *pandas* предоставляют мощные средства для анализа данных и научных вычислений, а *Django* и *Flask* – для разработки веб-приложений.

*Python* также славится своим активным сообществом разработчиков. В Интернете находится множество ресурсов, форумов, сообществ и библиотек *Python*, которые помогут вам изучить язык и решить любые возникшие вопросы.

Язык программирования *Python* широко используется в *Google Colab*, он является одним из наиболее популярных языков программирования для выполнения кода в этой среде. *Google Colab* предоставляет среду выполнения *Python* в облаке, что позволяет пользователям писать и выполнять код *Python* прямо в браузере без необходимости установки *Python* на локальном компьютере.

Преимущества использования *Python* в *Google Colab*:

* интерактивная среда. *Google Colab* предоставляет интерактивную среду, где есть возможность писать код *Python* в ячейках и мгновенно выполнять их. Так же есть возможность использовать ячейки для разделения кода на логические блоки и последовательного выполнения;
* бесплатный доступ к ресурсам. *Google Colab* предоставляет бесплатный доступ к вычислительным ресурсам, включая процессоры и графические процессоры (*GPUs*). Это особенно полезно для выполнения вычислительно интенсивных задач, таких как обучение нейронных сетей или обработка больших объемов данных;
* поддержка библиотек и фреймворков. *Google Colab* предустановлен с множеством популярных библиотек и фреймворков *Python*, таких как *NumPy*, *Pandas*, *Matplotlib*, *TensorFlow* и других. Они могут быть легко импортированы и использованы в проектах;
* облачное хранение и обмен данными. Есть возможность загружать и сохранять файлы в облаке прямо из *Google Colab*. Также доступна интеграция с *Google Drive*, что позволяет легко обмениваться данными между *Colab* и *Google Drive*;
* совместная работа и обмен кодом. Есть возможность пригласить других пользователей для совместной работы над проектами в *Google Colab*. Также есть возможность экспортировать код в различные форматы, такие как *Jupyter Notebook*, *Python*-скрипты или *PDF*-документы, для обмена и публикации.

*Google Colab* также предоставляет дополнительные возможности, такие как подключение к удаленным серверам, использование тензорных процессоров (*TPUs*) и другие расширенные функции.

Недостатки использования *Python* в *Google Colab*:

* ограниченные вычислительные ресурсы. *Google Colab* предоставляет ограниченные ресурсы для выполнения кода, включая ограниченный объем памяти и процессорное время. Это может ограничить возможности выполнения сложных и ресурсоемких задач.
* ограниченный доступ к аппаратному обеспечению. В *Google Colab* нет прямого доступа к аппаратному обеспечению, такому как *GPU* или *TPU*. Хотя *Google Colab* предоставляет возможность использования *GPU*, доступ к ним распределен между пользователями и может быть ограничен в зависимости от загруженности системы.
* ограниченная продолжительность сеанса. Каждая сессия в *Google Colab* имеет ограниченную продолжительность. Если сессия неактивна в течение определенного периода времени, ее можно потерять, и весь код и данные будут удалены. Это может быть неудобно, если требуется длительное время для выполнения задачи или обучения модели.
* зависимость от подключения к Интернету. *Google Colab* требует постоянного подключения к Интернету для работы. Если подключение прерывается или медленное, это может повлиять на работу в *Google Colab*.
* ограниченный доступ к файловой системе. В *Google Colab* есть ограничения на доступ к файловой системе. Например, нельзя обратиться к файлам, находящимся вне среды *Google Colab*, без дополнительных настроек и загрузки файлов.
* возможные ограничения на доступ к сторонним библиотекам. В некоторых случаях может возникнуть ограничение на доступ к некоторым сторонним библиотекам или расширениям в *Google Colab*. Это может ограничить возможности использования определенных инструментов и функций.

Несмотря на эти недостатки, *Google Colab* все же предоставляет удобную и доступную среду для выполнения кода на *Python* в облаке. Многие из ограничений могут быть преодолены с помощью дополнительных настроек и управления ресурсами [4].

В целом, использование *Python* в *Google Colab* обеспечивает удобную и мощную среду для выполнения и отладки кода, а также для проведения экспериментов и исследований в области анализа данных, машинного обучения и других областей программирования.

1. **МАТЕМАТИЧЕСКАЯ МОДЕЛЬ ЗАДАЧИ**
   1. **Теоретические основы задачи об оптимальных назначениях**

Задача об оптимальных назначениях является многокритериальной задачей, которая включает в себя поиск оптимальных решений для различных критериев.

Многокритериальные задачи являются одним из важных направлений в области оптимизации. Задачи возникают в различных областях, таких как экономика, управление, производство и другие. Одной из таких задач является задача об оптимальных назначениях, которая заключается в поиске оптимального распределения ресурсов по некоторым критериям.

Для решения задачи об оптимальных назначениях используются различные подходы и методы. Одним из них является метод взвешенной суммы, при котором каждый критерий оценивается определенным весом, и расчеты проводятся на основе взвешенной суммы значений критериев. Другим подходом является метод компромисса, при котором строится компромиссное решение, удовлетворяющее всем критериям в определенной степени.

Однако решение задачи об оптимальных назначениях часто является нетривиальной задачей, особенно при учете большого числа критериев и сложных ограничений. Для решения таких задач используются специальные методы, включающие в себя математическое программирование, теорию игр, искусственный интеллект и другие.

Таким образом, задача об оптимальных назначениях является многокритериальной задачей, которая предполагает поиск оптимального распределения ресурсов по нескольким критериям. Для ее решения применяются различные подходы и методы, которые позволяют найти компромиссное решение, удовлетворяющее всем требованиям и максимизирующее все указанные критерии.

Метод решения задачи об оптимальных назначениях – динамическое программирование, который позволяет эффективно решать задачи оптимизации путем разбиения их на более простые подзадачи. В данном случае, динамическое программирование будет применяться для поиска оптимального распределения задач и ресурсов в проекте, учитывая различные критерии.

Динамическое программирование представляет собой математический метод. Метод можно использовать для решения весьма широкого круга задач, включая задачи распределения ресурсов, замены и управления запасами, задачи о нагрузках. Характерными для динамического программирования является подход к решению задачи по этапам, с каждым из которых ассоциирована одна управляемая переменная. Набор рекуррентных вычислительных процедур, связывающих различные этапы, обеспечивает получение допустимого оптимального решения задачи в целом при достижении последнего этапа.

Происхождение названия динамического программирования, связано с использованием динамического программирования в задачах принятия решений через фиксированные промежутки времени. Однако методы динамического программирования успешно применяются также для решения задач, в которых фактор времени не учитывается.

Динамическое программирование – это поэтапное планирование многошагового процесса, при котором на каждом этапе оптимизируется только один шаг. При постановке задач динамического программирования следует руководствоваться следующими принципами:

1. выбрать параметры, характеризующие состояние *S* управляемой системы перед каждым шагом;
2. расчленить операцию на этапы (шаги);
3. выяснить набор шаговых управлений *xi* для каждого шага и налагаемые на них ограничения;
4. определить какой выигрыш приносит на *i*-ом шаге управление *xi*, если перед этим система была в состоянии *S*, то есть. записать «функцию выигрыша»:

(2.1)

1. определить, как изменяется состояние *S* системы *S* под влиянием управление *xi* на *i*-ом шаге оно переходит в новое состояние:

(2.2)

1. записать основное рекуррентное уравнение динамического программирования, выражающее условный оптимальный выигрыш *Wi(S)* (начиная с *i*-го шага и до конца) через уже известную функцию *Wi*+1(*S*):

(2.3)

Выигрышу соответствует условное оптимальное управление на *i*-м шаге *xi* (*S*) (причем в уже известную функцию *Wi*+1(*S*) надо вместо *S* подставить измененное состояние формула (2.2);

1. произвести условную оптимизацию последнего (*m*-го) шага, задаваясь гаммой состояний *S*, из которых можно за один шаг дойти до конечного состояния, вычисляя для каждого из них условный оптимальный выигрыш по формуле:

(2.4)

1. произвести условную оптимизацию (*m*-1)-го, (*m*-2)-го и т.д. шагов по формуле (1.3), полагая в ней *i*=(*m*-1), (*m*-2),…, и для каждого из шагов указать условное оптимальное управление *xi* (*S*), при котором максимум достигается.

Если состояние системы в начальный момент известно, то на первом шаге варьировать состояние системы не нужно – найти оптимальный выигрыш для данного начального состояния *S0*. Это и есть оптимальный выигрыш за всю операцию ;

1. произвести безусловную оптимизацию управления, «читая» соответствующие рекомендации на каждом шаге:
2. взять найденное оптимальное управление на первом шаге ;
3. изменить состояние системы по формуле (2.2);
4. для вновь найденного состояния найти оптимальное управление на втором шаге *х2\** и т.д. до конца.

Динамическое программирование является мощным инструментом для решения задачи об оптимальных назначениях. Оно позволяет разбить сложную задачу на более простые подзадачи и эффективно находить оптимальное решение. Применение динамического программирования в задаче об оптимальных назначениях позволяет найти наилучшее соответствие между исполнителями и задачами, учитывая стоимость или выгоду [5].

* 1. **Математическая модель задачи об оптимальных назначениях**

Математическая модель задачи об оптимальных назначениях, также известной как задача о назначениях, является классической комбинаторной задачей оптимизации. Она решает проблему определения наиболее выгодного соответствия между двумя наборами элементов.

Есть два набора элементов: множество предметов и курьеров. Необходимо назначить каждому курьеру одну задачу таким образом, чтобы общая стоимость или ценность назначений была наибольшей. Каждому назначению можно присвоить определенную стоимость или ценность, которая может быть количественной или качественной мерой.

Математическая модель задачи об оптимальных назначениях динамическим программированием сформулирована следующим образом:

1. пусть есть два множества элементов: множество предметов *I* = {*i1*,…, *in*} и множество курьеров *C* = {*c1*,…, *cm*};
2. пусть каждому предмету *ij* присвоены характеристики, такие как вес предмета *Wj* и стоимость предметов *Sj* и другие;
3. пусть каждый курьер *c1* обладает свойствами, такими как доступность *Кi*, ставка за час *Gi* и другие;
4. вводим такие переменные *x*(*i*, *j*), где *x*(*i*, *j*) = 1, если предмет *ij* назначен курьеру *c j* и *x*(*i*, *j*) = 0 в противном случае;
5. целевая функция будет многокритериальной и будет зависеть от требований конкретной задачи, есть два критерия: вес и стоимость предметов. Тогда целевая функция выглядит следующим образом:

(2.5)

где *α* и *β* – коэффициенты, которые определяют важность каждого критерия;

*W* (*i*, *j*) – стоимость доставки предмета *iⱼ* курьером *cⱼ*;

*S* (*i*, *j*) – время доставки предмета *iⱼ* курьером *cⱼ*.

1. Должны выполняться следующие ограничения:
2. каждый курьер должен доставить предметы, общий вес которых не превышает его вместимость:

+ *Wⱼ* ≤ *Bᵢ*, (2.6)

1. переменные *x*(*i*, *j*) должны быть бинарными *x*(*i*, *j*) ∈ {0, 1};
2. ограничение на общую стоимость ставки за час:

+ *Rᵢ* ≤ *R\_max*, (2.7)

где *R\_max* – максимально допустимая стоимость ставки за час для всех курьеров.

Для решения этой задачи можно использовать различные методы оптимизации, такие как методы линейного программирования, динамического программирования или эволюционные алгоритмы. Окончательный выбор метода зависит от размера задачи, сложности критериев и доступных ресурсов вычислений.

Применение этой модели может быть полезным в различных сферах, где требуется эффективное назначение курьеров для доставки посылок с учетом множества критериев, таких как стоимость, вес, время доставки и т.д.

Абстрактная блок схема реализации представлена на рисунке 2.1.

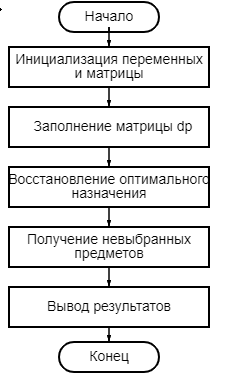


Рисунок 2.1 – Абстрактная блок схема реализации

Подробное описание блоков абстрактной блок схемы:

* программа начинается с блока «Начало программы»;
* затем выполняется инициализация переменных и создание матрицы *dp* размером (*n*, *max\_weight*+1, *max\_cost*+1);
* после этого происходит заполнение матрицы *dp* с помощью вложенных циклов. Внешний цикл *i* перебирает предметы от 0 до *n*-1. Внутренние циклы *w* и *c* перебирают возможные значения веса и стоимости соответственно. В каждом шаге проверяется, может ли предмет быть включен в текущее решение, и выбирается максимум из двух вариантов: включить предмет или не включать его. Значение записывается в матрицу *dp*[*i*, *w*, *c*];
* после заполнения матрицы *dp* выполняется восстановление оптимального назначения с помощью цикла *while*. На каждом шаге проверяется, если значение в матрице *dp* на текущем шаге отличается от значения на предыдущем шаге, то предмет *i* был включен в оптимальное назначение. Его индекс добавляется в список, и значения *w* и *c* уменьшаются на вес и стоимость выбранного предмета соответственно;
* после восстановления оптимального назначения получаем список невыбранных предметов, которые не включены в оптимальное решение;
* в конце программы происходит возврат выбранных и невыбранных предметов;
* программа завершается в блоке «Конец программы».

Подробное описание реализации представлено в приложении В.

* 1. **Теоретические основы задачи о рюкзаке**

Задача о рюкзаке также является многокритериальной задачей, но имеет свои особенности. В данной задаче речь идет о выборе оптимального набора предметов, которые можно поместить в рюкзак с ограниченной вместимостью. Каждый предмет имеет свой вес и стоимость. Цель состоит в том, чтобы максимизировать стоимость предметов, помещенных в рюкзак, при соблюдении ограничений по весу.

Метод решения данной задачи – линейное программирование, которое позволяет найти оптимальное решение задачи с линейной целевой функцией и линейными ограничениями. В данном случае, линейное программирование будет применяться для определения оптимального набора предметов для размещения в рюкзаке с учетом их веса и стоимости, а также ограничений, связанных с вместимостью рюкзака.

Одно из таких ограничений, которые могут быть введены в задачу об оптимальных назначениях, это максимальный вес посылки, который курьер может нести. Это ограничение будет учитываться при определении оптимальных назначений для задачи. Кроме того, может быть установлена стоимость ставки в час для курьера, что также будет учтено при оптимизации проектного решения.

Однако в реальной жизни часто возникают ситуации, когда решение необходимо принимать с учетом нескольких критериев одновременно. Например, помимо максимизации ценности предметов, может быть важно минимизировать их объем или стоимость. В таких случаях задача о рюкзаке становится многокритериальной задачей оптимизации.

Многокритериальная задача о рюкзаке имеет особенности, множество вариантов решений: при наличии нескольких критериев оптимального выбора может быть множество вариантов решений, каждое из которых представляет компромисс между различными критериями:

1. невозможность однозначного решения: из-за наличия нескольких критериев оптимального выбора может быть невозможно найти единственное наилучшее решение. Вместо этого требуется построение множества Парето-оптимальных решений, которые не могут быть улучшены ни по одному из критериев без ухудшения по другим критериям;
2. необходимость анализа компромиссных решений: для выбора оптимального решения требуется анализ всех компромиссных вариантов, чтобы определить наиболее предпочтительные решения, учитывая все критерии одновременно;
3. использование математических моделей: для решения многокритериальной задачи о рюкзаке широко применяются математические модели, такие как многокритериальное программирование или методы искусственного интеллекта, которые позволяют найти оптимальные решения на основе компромиссов между критериями.

Многокритериальная задача о рюкзаке является сложной и интересной областью исследования операционного исследования и искусственного интеллекта. Задача находит применение в различных областях, включая логистику, производство, финансы и другие.

Основные шаги решения задачи о рюкзаке с использованием метода линейного программирования:

1. шаг первый: определение целевой функции. Сначала необходимо определить целевую функцию, которую мы хотим оптимизировать. В случае задачи о рюкзаке обычно целью является максимизация общей стоимости упакованных предметов или минимизация общего веса;
2. шаг второй: определение переменных. Задача о рюкзаке требует определения переменных решения. Обычно каждая переменная соответствует предмету и может принимать значения ноль или единица, где единица означает, что предмет упакован, а ноль – что предмет не упакован;
3. шаг третий: определение ограничений. В задаче о рюкзаке главное ограничение – это вместимость рюкзака, которая определяет суммарный вес упакованных предметов. Также может быть ограничение на общее количество предметов, которые можно упаковать;
4. шаг четвертый: построение математической модели. На основе целевой функции, переменных и ограничений строится математическая модель задачи о рюкзаке в виде линейной программы;
5. шаг пятый: применение алгоритма линейного программирования. Для решения задачи о рюкзаке используются алгоритмы линейного программирования, такие как симплекс-метод или метод ветвей и границ. Эти алгоритмы находят оптимальное решение, удовлетворяющее всем ограничениям;
6. шаг шестой: интерпретация результатов. Когда алгоритм линейного программирования завершает свою работу, можно проанализировать полученное решение. Это позволяет определить, какие предметы следует упаковать в рюкзак для достижения наилучшего результата с точки зрения целевой функции.

Таковы основные шаги метода линейного программирования для решения задачи о рюкзаке. Этот метод широко применяется в оптимизационных задачах и позволяет найти оптимальное решение при заданных ограничениях [6].

Основные принципы и компоненты линейного программирования в оптимизации проектного решения:

1. переменные решения. В линейном программировании оптимизируемые значения называются переменными решения. В контексте проектного решения, переменные решения могут представлять различные параметры, такие как объемы работ, распределение ресурсов, длительности активностей и другие;
2. целевая функция. Целевая функция определяет цель или критерий оптимизации. В проектном решении это может быть минимизация затрат, максимизация прибыли, сокращение времени выполнения проекта и другое. Целевая функция формулируется на основе переменных решения и может включать коэффициенты, отражающие важность различных факторов;
3. ограничения. Ограничения представляют собой условия, которые должны быть удовлетворены при оптимизации проектного решения. Они могут быть связаны с ограничениями на ресурсы, временными ограничениями, логическими ограничениями и другими ограничениями, связанными с конкретным проектом. Ограничения формулируются с использованием переменных решения и могут быть линейными или нелинейными;
4. линейные ограничения. В линейном программировании ограничения являются линейными функциями переменных решения. Это означает, что ограничения могут быть выражены в виде линейных уравнений или неравенств. Например, ограничения могут ограничивать доступные ресурсы, требовать выполнения определенных условий или ограничивать максимальные или минимальные значения переменных решения;
5. математическая модель. Линейное программирование требует построения математической модели, которая объединяет целевую функцию и ограничения. Модель представляет собой систему линейных уравнений или неравенств, которые описывают оптимизационную задачу. Решение этой модели дает оптимальные значения переменных решения, удовлетворяющие ограничениям и максимизирующие или минимизирующие целевую функцию;
6. использование оптимизационных алгоритмов. Для решения задачи линейного программирования в оптимизации проектного решения используются оптимизационные алгоритмы, которые находят оптимальное решение. Существуют различные методы решения, такие как симплексный метод, метод симплекса, внутренняя точечная методика и другие.

Применение линейного программирования в оптимизации проектного решения может быть полезным для принятия решений, связанных с оптимальным использованием ресурсов, планированием и управлением проектами. Это может помочь снизить затраты, улучшить эффективность и достичь поставленных целей проекта.

* 1. **Математическая модель задачи о рюкзаке**

Задача о рюкзаке с ограничением веса груза и ограничением стоимости ставки в час является вариацией классической задачи о рюкзаке, в которой помимо ограничения на вес предметов, добавляется ограничение на стоимость выбранных предметов.

Предположим, есть набор предметов, каждый из которых имеет свой вес и стоимость. Необходимо выбрать некоторые предметы из этого набора для помещения в рюкзак с ограниченной вместимостью и ограниченной стоимостью ставки в час.

Ограничение по весу означает, что суммарный вес выбранных предметов не должен превышать вместимость рюкзака. Ограничение по стоимости означает, что суммарная стоимость выбранных предметов не должна превышать допустимую стоимость ставки в час.

Цель состоит в том, чтобы найти такой набор предметов, который максимизирует суммарную стоимость предметов в рюкзаке, с учетом ограничений по весу и стоимости.

Математическая модель задачи о рюкзаке с ограничением веса груза и ограничением стоимости ставки в час сформулирована следующим образом:

1. пусть есть *n* предметов, каждый из которых имеет два свойства: вес *wi* и стоимость *сi*;
2. рюкзак имеет ограниченную вместимость *W*, выраженную весом;
3. также есть ограничение *С* на стоимость выбранных предметов.

Необходимо максимизировать суммарную стоимость перевезенных предметов:

= *с1* ∙ *x1* + … + *сi* ∙ *xi → max* (2.8)

где *xi* – переменная решения, которая принимает значение единица, если предмет *i* выбран для перевозки, и ноль в противном случае.

При этом должны выполняться следующие ограничения:

1. суммарный вес выбранных предметов не должен превышать вместимость рюкзака:

*w1* ∙ *x1* + … + *wi* ∙ *xi* ≤*W*, (2.9)

1. суммарная стоимость выбранных предметов не должна превышать ограничение стоимости ставки в час:

*сi* ∙ *xi* + … + *сn* ∙ *xn* ≤ *C*, (2.10)

1. ограничение на выбор каждого предмета *xi* = {0, 1}.

где *wi* – вес предмета *i*;

*сi* – стоимость предмета *i*;

*W* – ограничение на вместимость рюкзака;

*С* – ограничение на стоимость выбранных предметов.

Таким образом, необходимо найти такие значения переменных *xi*, которые максимизируют суммарную стоимость предметов *сi*, соблюдая ограничение на вместимость рюкзака по весу и ограничение на стоимость выбранных предметов. Решив данную математическую модель задачи о рюкзаке методом линейного программирования, есть возможность определить оптимальный набор посылок для перевозки, который максимизирует стоимость и при этом удовлетворяет ограничениям курьера.

Решение задачи о рюкзаке методом линейного программирования выполняется следующим образом:

a) необходимо cоставить матрицу размером (*N*+1) × (*W*+1), где *N* – коли-чество предметов, а *W* – вместимость рюкзака. Заполним первую строку и первый столбец нулями, а остальные ячейки единицами;

b) итеративно заполним матрицу следующим образом:

* для каждого предмета *i* от 1 до *N*:
* для каждого веса *j* от 1 до *W*:
* если вес предмета *i* меньше или равен текущему весу *j*:
* в ячейку матрицы (*i, j*) запишем максимум из двух значений:
* значение в ячейке (*i*-1, *j*) – это максимальная стоимость, которую можно получить, не выбирая предмет *i*.
* значение в ячейке (*i*-1, *j*-*wi*) + *pi* – это максимальная стоимость, которую можно получить, выбрав предмет i и добавив его стоимость к максимальной стоимости для веса (j-wi).
* иначе:
* значение в ячейке (*i, j*) будет равно значению в ячейке *(i*-1, *j*) – это максимальная стоимость, которую можно получить, не выбирая предмет *i*.

c) максимальная стоимость, которую можно получить, будет равна значению в последней ячейке матрицы (*N*, *W*).

d) для восстановления выбранных предметов, начнем с ячейки матрицы (*N*, *W*) и проверим, какое значение больше: значение в ячейке (*N*-1, *W*) или значение в ячейке (*N*-1, *W*-*wN*) + *pN*. Если значение в ячейке (*N*-1, *W*) больше, значит предмет *N* не был выбран. В противном случае предмет *N* был выбран. Повторим этот шаг для предыдущих предметов, пока не достигнем ячейки (1, 0).

e) после завершения этого процесса, будут получены выбранные предметы, которые дают максимальную стоимость.

Пример:

* *n* = 4 (количество предметов);
* *x1*, *x2*, *x3*, *x4* - переменные, где *xi*= 1, если предмет *i* выбран, и *xi* = 0, если предмет *i* не выбран;
* *V* = [10, 20, 15, 25] (стоимость предметов);
* *W* = [5, 10, 8, 15] (вес предметов);
* *C* = 20 (максимально допустимый вес посылки).

Тогда математическая модель задачи может быть записана следующим образом:

Целевая функция: *F* = 10 *x1* + 20 *x2* + 15 *x3* + 25 *x4*→ *max*

Ограничения: 5∙1 + 10∙2 + 8∙3 + 15∙4 <= 20; *x1* = 0 или 1, *x2* = 0 или 1, *x3* = 0 или 1, *x4* = 0 или 1.

Решение:

Из полученного решения (10 *x1* + 20 *x2* + 15 *x3* + 25 *x4*), видно, что максимальная суммарная стоимость выбранных предметов равна 35 и достигается, когда предметы 1 и 4 выбраны (*x1* = 1, *x2* = 0, *x3* = 0, *x4* = 1).

Таким образом, решение задачи о рюкзаке заключается в выборе предметов 1 и 4 для максимизации суммарной стоимости в пределах максимально допустимого веса рюкзака (20).

Абстрактная блок схема реализации представлена на рисунке 2.2.

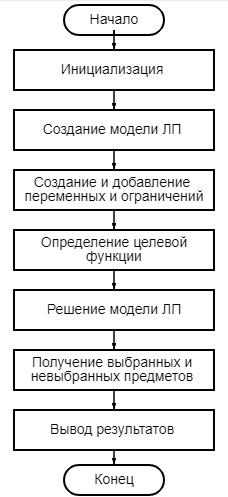


Рисунок 2.2 – Абстрактная блок схема реализации

Подробное описание блоков абстрактной блок схемы:

* программа начинается с блока «Начало программы»;
* затем следует блок «Инициализация», где переменные и константы инициализируются;
* в блоке «Создание модели ЛП» создается экземпляр модели линейного программирования (ЛП);
* затем происходит создание переменных и добавление ограничений в блоке «Создание и добавление переменных и ограничений». Создаются бинарные переменные *x*[*i*], которые представляют выбор каждого предмета;
* в блоке «Определение целевой функции» определяется целевая функция для максимизации количества выбранных предметов;
* затем модель ЛП решается в блоке «Решение модели ЛП» с помощью вызова метода *model.solve*();
* в блоке «Получение выбранных и невыбранных предметов» определяются выбранные и невыбранные предметы на основе значений переменных решения;
* в блоке «Вывод результатов» результаты выводятся на экран;
* программа завершается в блоке «Конец программы».

Подробное описание реализации представлено в приложении В.

Задача о рюкзаке не имеет известного эффективного алгоритма, который может решить эту задачу за полиномиальное время для большинства входных данных. Вместо этого используются различные эвристические и точные методы для приближенного или оптимального решения.

Математическая модель задачи о рюкзаке с ограничением веса и стоимости является целочисленной линейной программой, где переменные *xi* являются бинарными и определяют, выбран ли предмет *i* для помещения в рюкзак.

Входные данные для задачи о рюкзаке включают веса и стоимости каждого предмета, а также ограничения по весу и стоимости ставки в час. Эти данные определяются в контексте конкретной задачи или проблемы, где требуется оптимизация.

Задача о рюкзаке с ограничениями имеет множество применений в различных областях, таких как логистика, управление запасами, финансовое планирование и другие, может быть использована для оптимального выбора товаров, вложений или проектов с ограничениями ресурсов.

Существуют различные алгоритмы и подходы для решения задачи о рюкзаке с ограничением веса груза и ограничением стоимости ставки в час, включая динамическое и линейное программирование, жадные алгоритмы, алгоритмы ветвей и границ, а также метаэвристические методы [7].

Выбор конкретного алгоритма зависит от размера задачи, требуемой точности, времени выполнения и доступных ресурсов. Для небольших задач можно использовать точные методы, а для более крупных задач может потребоваться приближенное решение с использованием эвристических алгоритмов.

* 1. **Математическая модель многокритериальной оптимизации**

В многокритериальной оптимизации (МКО) решаются задачи принятия решений одновременно по нескольким критериям. Задача МКО ставится следующим образом: требуется найти числа *х1*,…,*хn*, удовлетворяющие системе ограничений *gi*(*х1*,…,*хn*) ≤ *bi*, для которых функции *zk*= *fk*(*х1*,…,*хn*), достигают максимального значения.

Множество точек *X* = (*х1*,…,*хn*), удовлетворяющих системе, образует допустимую область *D* ⸦ *Rn*. Элементы множества *D* называются допустимыми решениями или альтернативами, а числовые функции *fk*– целевыми функциями, или критериями, заданными на множестве *D*. В формулировке задаче присутствует целевых функций. Эти функции отображают множество *D* ⸦ *Rn* в множество *F* ⸦ *Rn*, которое называется множеством достижимости.

В векторной форме математическую модель МКО можно записать следующим образом:

*f*(*X*) = (*f1*(*X*),…, *fk*(*X*)) → *max*, (2.11)

где *f*(*X*) – вектор-функция аргумента *X* ∈ *D*.

В отличие от задач оптимизации с одним критерием в МКО имеется неопределенность целей. Действительно, существование решения, максимизирующего несколько целевых функций, является редким исключением, поэтому с математической точки зрения задачи МКО являются неопределенными и решением может быть только компромиссное решение. Например, при поиске плана предприятия, макимизирующего прибыль и минимизирующего затраты очевидна невозможность достижения обеих целей одновременно, так как чем больше затраты, тем больше должно быть продукции и тем больше прибыль.

В теории МКО понятие оптимальности получает различные толкования, и поэтому сама теория содержит три основных направления: разработка концепции оптимальности, доказательство существования решения, оптимального в соответствующем смысле и разработка методов нахождения оптимального решения.

1. **ПРОГРАМНАЯ РЕАЛИЗАЦИЯ ПОСТАВЛЕННОЙ ЗАДАЧИ**
   1. **Описание разработанного приложения**

В данном подразделе будет описан основной функционал программного приложения.

* 1. решение задачи об оптимальных назначениях методом динамического программирования. В коде представлена реализация алгоритма решения задачи об оптимальных назначениях методом динамического программирования. Данный метод используется для оптимального распределения ресурсов с учетом стоимостей выполнения каждой задачи.
  2. решение многокритериальной задачи о рюкзаке методом линейного программирования: В коде представлена реализация алгоритма решения многокритериальной задачи о рюкзаке с помощью метода линейного программирования. Задача о рюкзаке заключается в выборе оптимального набора предметов с ограничением на вместимость рюкзака и учетом их стоимости.
  3. импорт необходимых библиотек. В коде присутствуют инструкции импорта библиотек, таких как *numpy* и *pulp*. Библиотека *pulp* используется для решения задач линейного программирования.

В разработанной программе нет разделения на роли пользователей, поскольку здесь не требуется многопользовательский доступ с соответствующей ролевой политикой.

* 1. **Запуск программного продукта**

Для запуска программы вам понадобится компьютер с доступом в Интернет и интерпретатор *Python* с установленными необходимыми библиотеками. Одним из удобных способов запуска программы без необходимости установки дополнительного программного обеспечения является использование *Google Collab*. *Google Collab* предоставляет веб-интерфейс для разработки и выполнения кода на языке *Python* в облаке, и все необходимые библиотеки автоматически подключаются.

Чтобы запустить программу в *Google Collab*, нужно создать новый проект и скопировать код в него или загрузить файл с проектом. Затем можно выполнить код, нажав на кнопку «Выполнить все» или используя сочетание клавиш *Shift*+*Enter*. *Google Collab* предоставляет возможность запуска кода по ячейкам, что облегчает отладку и пошаговое выполнение программы. Другие альтернативы для запуска программы на компьютере с установленным интерпретатором *Python* включают использование интегрированных сред разработки (*IDE*) или командной строки. В таком случае может потребоваться установить необходимые библиотеки *Python* с помощью инструмента управления пакетами, такого как *pip*, перед запуском программы.

Также нужно убедиться, что используемый компьютер соответствует системным требованиям для запуска программы и имеет доступ к необходимым ресурсам, таким как интернет и вычислительные мощности для распределенной обработки запросов клиентов.

* 1. **Верификация программного продукта**

В рамках курсового проекта был разработан программный продукт, который решает две задачи. Первая задача, называемая «Задача об оптимальных назначениях методом динамического программирования», решается с использованием метода динамического программирования. Вторая задача, «Многокритериальная задача о рюкзаке методом линейного программирования», решается с помощью метода линейного программирования.

Программа принимает входные данные, которые были предварительно введены, и выполняет их обработку. Результаты обработки представляют собой решения обеих задач. Для задачи об оптимальных назначениях методом динамического программирования и линейного программирования многокритериальной задачи о рюкзаке программа определяет оптимальное распределение предметов в рюкзаке с учетом ограничений на максимальный вес и стоимость ставки в час у курьера.

Полученные результаты работы программы могут быть использованы для принятия решений в соответствующих областях, например, в логистике или планировании ресурсов. При анализе результатов следует учитывать, что они зависят от введенных исходных данных и параметров задачи. Также важно провести дополнительную проверку и анализ результатов, чтобы убедиться в их корректности и соответствии поставленным целям проекта.

В задаче об оптимальных назначениях, решаемой методом динамического программирования, имеются следующие исходные данные, представленные на рисунке 3.1.

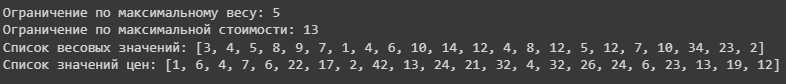


Рисунок 3.1 – Исходные данные

Результатом выполнения метода динамического программирования является список задействованных предметов, полученные результаты указывают на следующие предмет: № 22: веc – 2 стоимость – 12.

Это означает, что при оптимальном распределении предметов в рюкзаке с учетом ограничений на максимальный вес и стоимость ставки в час, выбранные предметы 22 являются наиболее выгодными.

Также, можно рассмотреть само назначение элементов, чтобы определить, какие элементы были выбраны и как они соотносятся с ограничениями, такими как максимальный вес посылки и стоимость ставки в час. Еще стоит учесть, что результаты могут зависеть от конкретных значений в исходных данных и параметров задачи. Фрагмент программы с полученными данными показан на рисунке 3.2.

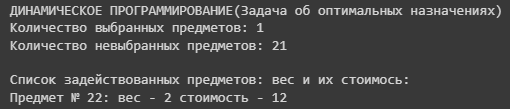


Рисунок 3.2 – Полученные данные

Результатом выполнения метода линейного программирования является список задействованных предметов, полученные результаты указывают на следующие предметы: предмет № 1: веc – 3 стоимость – 1 и предмет № 22: веc – 2 стоимость – 12.

Это означает, что при оптимальном распределении предметов в рюкзаке с учетом ограничений на максимальный вес и стоимость ставки в час, выбранные предметы 1 и 22 являются наиболее выгодными. Они обладают относительно небольшим весом и высокой стоимостью, что делает их оптимальным выбором в данной задаче. Фрагмент программы с полученными данными показан на рисунке 3.4.

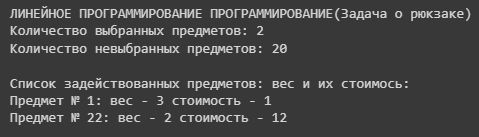


Рисунок 3.4 – Полученные данные

В результате решения двух задач – задачи об оптимальных назначениях методом динамического программирования и задачи о рюкзаке методом линейного программирования – были получены следующие результаты:

1) задача об оптимальных назначениях методом динамического программирования:

– в результате решения задачи об оптимальных назначениях для оптимального распределения курьеров для доставки посылок были выбраны два предмета: предмет №22 с весом два килограмма и стоимостью 12. Эти предметы являются наиболее выгодными с учетом ограничений на максимальный вес и стоимость ставки в час.

2) задача о рюкзаке методом линейного программирования:

– в результате решения задачи о рюкзаке были выбраны два предмета для упаковки: предмет 1 с весом 3 и стоимостью 1, и предмет 22 с весом 2 и стоимостью 12. Эти предметы являются наиболее выгодными с учетом ограничений на максимальный вес и стоимость ставки в час.

Оба решения представляют собой оптимальные результаты в соответствующих задачах.

* 1. **Тестирование программы**

Решение примера для проверки правильной реализации программы. Для решения примера задачи о рюкзаке с ограничениями на максимальный вес и стоимость. Пусть, дано *N* предметов, *W* вместимость рюкзака, *w* = {w1,…,wn} соответствующий ему набор положительных целых весов, p = {p1,…, pn} соответствующий ему набор положительных целых стоимостей.

Исходные данные: *W* = 13, *N* = 5, *w1*= 3, *p1* = 1, *w2*= 4, *p2* = 6, *w3*= 5, *p3* = 4, *w4*= 8, *p4* = 7, *w5*= 9, *p5* = 6.

Описание решения примера:

1. создаем таблицу *А*, где каждый элемент инициализируем нулем;
2. на каждом этапе рассматриваем предметы с индексами от 1 до *k* (где *k* - номер текущего предмета в списке);
3. для каждого предмета и каждой возможной вместимости рюкзака:

* если вес предмета меньше или равен текущей вместимости, сравниваем стоимость предмета и стоимость предыдущего набора предметов (который помещается в рюкзак с оставшейся вместимостью) с стоимостью предыдущего набора предметов (без текущего предмета). Записываем максимальное значение в соответствующую ячейку таблицы *A*[*k*][*s*];
* если вес предмета больше текущей вместимости, просто копируем значение из предыдущей ячейки: *A*[*k*][*s*] = *A*[*k*-1][*s*];

1. максимальная стоимость рюкзака будет находиться в ячейке *A*[количество предметов][максимальный вес].
2. для восстановления набора предметов, из которых состоит максимальный рюкзак, начинаем с ячейки *A*[количество предметов][ма-ксимальный вес] и идем в обратном порядке:

* если значение в ячейке *A*[*k*][*s*] отличается от значения в ячейке *A*[*k*-1][*s*], значит предмет *k* был включен в рюкзак. Добавляем его в набор предметов;
* уменьшаем текущую вместимость рюкзака на вес предмета;
* переходим к предыдущей ячейке *A*[*k*-1][*s*] и продолжаем процесс до достижения ячейки *A*[0][0].

Рассмотрим для примера k=3, при каждом *s*⩾5(так как *w*3=5) сравниваем *A*[*k* - 1][*s*] и *A*[*k* - 1][*s*−*w*3]+*p3* и записываем в *A*[*k*][*s*] стоимость либо рюкзака без третьего предмета, но с таким же весом, либо с третьим предметом, тогда стоимость равна стоимости третьего предмета плюс стоимость рюкзака с вместимостью на *w*3 меньше.

Заполняем таблицу, результат заполнения представлен в таблице 3.1.

Таблица 3.1 – Заполненная таблица

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
| k=0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| k=1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| k=2 | 0 | 0 | 1 | 6 | 6 | 6 | 7 | 7 | 7 | 7 | 7 | 7 | 7 |
| k=3 | 0 | 0 | 1 | 6 | 6 | 6 | 7 | 7 | 10 | 10 | 10 | 11 | 11 |
| k=4 | 0 | 0 | 1 | 6 | 6 | 6 | 7 | 7 | 10 | 10 | 10 | 13 | 13 |
| k=5 | 0 | 0 | 1 | 6 | 6 | 6 | 7 | 7 | 10 | 10 | 10 | 13 | 13 |

Максимальная стоимость рюкзака находится в *A*(5,13). Начиная с *A*(5,13) восстанавливаем ответ. Необходимо идти в обратном порядке по *k*. Отмечем фоном и обозначим наш путь в таблице 3.2.

Таблица 3.2 – Результат нахождения пути

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
| k=0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| k=1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| k=2 | 0 | 0 | 1 | 6 | 6 | 6 | 7 | 7 | 7 | 7 | 7 | 7 | 7 |
| k=3 | 0 | 0 | 1 | 6 | 6 | 6 | 7 | 7 | 10 | 10 | 10 | 11 | 11 |
| k=4 | 0 | 0 | 1 | 6 | 6 | 6 | 7 | 7 | 10 | 10 | 10 | 13 | 13 |
| k=5 | 0 | 0 | 1 | 6 | 6 | 6 | 7 | 7 | 10 | 10 | 10 | 13 | 13 |

Таким образом, в набор входит 2 и 4 предмет.

Результат выполнения в программе данного примера:

На рисунке 3.5 представлены исходные данные.

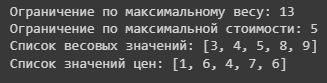


Рисунок 3.5 – Исходные данные примера

На рисунке 3.6. представлен результат работы программы.

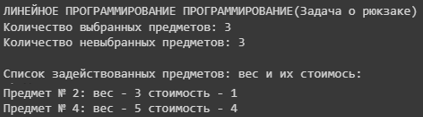


Рисунок 3.6 – Результат работы программы

Таким образом, в набор входит 2 и 4 предмет.

Стоимость рюкзака: 1+ 4 =5.

Вес рюкзака: 3+5=8.

Таким образом, найдена максимальная стоимость и набор предметов для рюкзака, из которых состоит.

Согласно полученным данным, результат расчёта в разработанной программе и расчета вручную являются одинаковым. Можно сделать вывод, что программа реализована верно. Сравнение полученных результатов произведено успешно программа полностью выполняет поставленную задачу курсового проекта.

Также, можно рассмотреть само назначение элементов, чтобы определить, какие элементы были выбраны и как они соотносятся с ограничениями, такими как максимальный вес посылки и стоимость ставки в час. Еще стоит учесть, что результаты могут зависеть от конкретных значений в исходных данных и параметров задачи.

# **ЗАКЛЮЧЕНИЕ**

В результате выполнения заданного курсового проекта были реализованы функции, которые позволяют решать задачу об оптимальных назначениях с использованием метода динамического программирования. Этот метод используется для оптимального распределения курьеров для доставки посылок. Это позволило решить задачу об оптимальных назначениях, учитывая различные ограничения и целевые критерии, и получить эффективное решение, минимизирующее время доставки или затраты.

Также задача о рюкзаке, которая является классической оптимизационной

задачей, где требуется выбрать оптимальный набор предметов для упаковки в рюкзак с учетом ограничений на вес и стоимость. Отличие данного проекта от существующих решений заключается в том, что он предоставляет реализацию задачи об оптимальных назначениях методом динамического программирования и включает в себя использование библиотеки *PuLP* для решения задачи о рюкзаке методом линейного программирования. Это позволяет пользователям использовать эти методы для оптимизации своих задач, связанных с распределением ресурсов и решением многокритериальных задач о рюкзаке.

Метод динамического программирования позволяет решать задачи оптимизации, разбивая их на более мелкие подзадачи и решая их последовательно с сохранением и переиспользованием промежуточных результатов. В случае задачи о рюкзаке, метод линейного программирования позволяет эффективно итерироваться по всем возможным комбинациям предметов и выбирать оптимальные комбинации, удовлетворяющие ограничениям на вес и стоимость.

Одно из направлений развития программы, которое следует рассмотреть, это создание пользовательского интерфейса, который сделает программу более удобной и доступной для пользователей. Курсовой проект выполнен самостоятельно и проверен в системе Антиплагиат с процентом уникальности 85,45.

# **Список использованных источников**

1. Мурашко, И. А. Оптимизация проектных решений – И.А. Мурашко, Д.Е. Храбров. – Гомель: ГГТУ им. П.О. Сухого, 2014. – 98 с.
2. Решение многокритериальной задачи линейного программирова- ния. – Электронные данные. – Режим доступа: <https://xreferat.com/114/346-1-reshenie-mnogokriterial-noiy-zadachi-lineiynogo-programmirovaniya.html>. – Дата доступа: 28.10.2023.
3. Преимущества и недостатки языка программирования Python. – Элек-троные данные. – Режим доступа: [https://skysmart.ru/articles/programming/ preimushestva-i-nedostatkipython](https://skysmart.ru/articles/programming/%20preimushestva-i-nedostatkipython). – Дата доступа: 26.10.2023.
4. Чичулин, А.А. G Colab: Инновации в ваших руках. – А.А. Чичурин – Ру-ководство. М. : Издательство Юрайт, 2023. – 105 с.
5. Динамическое программирование – рюкзак, НВП, НОП. – Электронные данные. – Режим доступа: <https://algorithmica.org/tg/knapsack-gis-gcs>. – Дата доступа: 18.11.2023.
6. Палий, И. А. Линейное программирование. – И.А. Палий – Учебное пособие. М. : Издательство Юрайт, 2018. – 175 с.
7. Шорикова, А. Ф. Математическое программирование. Теория и мето-ды – А.Ф. Шорикова, М.А. Плескунова – Учебное пособие. «Флинта»: Издательский центр «Уральского университет», 2022. – 201 с.
8. Банников, А.С. Численные методы – А.С. Банников, И.Г. Ким, Н.В. Латыпова – Учебное пособие. Ижевск: Издательский центр «Удмуртский университет», 2018. – 80 с.
9. Тюрин, А.Г. Кластерный анализ, методы и алгоритмы кластеризации / А.Г. Тюрин, И.О. Зуев – Вестник МГТУ МИРЭА. 2018. – 12 с.

# **ПРИЛОЖЕНИЕ А**

(обязательное)

**Листинг программы**

**kurs\_OPR.ipynb**

#Подключение библиотек

import numpy as np

!pip install pulp

from pulp import \*

def print\_result(selected\_items, not\_selected\_items, max\_weight, max\_cost, weights, costs):

'''Вывод результатов.'''

print(f'Количество выбранных предметов: {len(selected\_items)}')

print(f'Количество невыбранных предметов: {len(not\_selected\_items)}')

print('\nСписок задействованных предметов: вес и их стоимось:')

if(len(selected\_items) == 0):

print('Предметы под выбранные значения макс. веса и стоимости не найдены!')

else:

for item in selected\_items:

print(f'Предмет № {item + 1}: веc - {weights[item]} стоимость - {costs[item]}')

def optimal\_assignments(weights, costs, max\_weight, max\_cost):

n = len(weights) # Количество предметов

m = len(costs) # Количество курьеров

# Создание матрицы для хранения результатов

dp = np.zeros((n, max\_weight+1, max\_cost+1))

# Заполнение матрицы результатами

for i in range(n):

for w in range(max\_weight+1):

for c in range(max\_cost+1):

if weights[i] > w or costs[i] > c:

dp[i, w, c] = dp[i-1, w, c]

else:

dp[i, w, c] = max(dp[i-1, w, c], dp[i-1, w-weights[i], c-costs[i]] + 1)

selected\_items = []

i = n - 1

w, c = max\_weight, max\_cost

while i >= 0 and w > 0 and c > 0:

# Если предмет i был включен в оптимальное назначение

if dp[i, w, c] != dp[i-1, w, c]:

selected\_items.append(i)

w -= weights[i]

c -= costs[i]

i -= 1

# Получение невыбранных предметов

not\_selected\_items = [i for i in range(n) if i not in selected\_items]

return selected\_items, not\_selected\_items

def knapsack\_multiobjective(weights, costs, max\_weight, max\_cost):

'''Решение задачи о рюкзаке с использованием ЛП.

Возврат: список выбранных и невыбранных предметом'''

n = len(weights)

# Создание экземпляра модели ЛП: параметр LpMaximize значает поиск максимума

model = LpProblem("KnapsackMultiobjective", LpMaximize)

# Мы создаем бинарные переменные x[i], где i - индекс предмета. Эти переменные принимают значения True (1) или False (0), обозначающие выбран или не выбран соответствующий предмет.

x = [LpVariable(f"Предмет {i}", cat="Binary") for i in range(n)]

# Ограничение на общий вес

weight\_constraint = lpSum(weights[i] \* x[i] for i in range(n)) <= max\_weight

model.addConstraint(weight\_constraint)

# Ограничение на общую стоимость

cost\_constraint = lpSum(costs[i] \* x[i] for i in range(n)) <= max\_cost

model.addConstraint(cost\_constraint)

# # Ограничение на выбор только одного предмета

# select\_one\_constraint = lpSum(x[i] for i in range(n)) == 1

# model.addConstraint(select\_one\_constraint)

# Целевая функция - максимизация количества выбранных предметов

# Мы хотим максимизировать количество выбранных предметов, поэтому целевая функция - сумма переменных решения `x[i]`

objective = lpSum(x[i] for i in range(n))

model.setObjective(objective)

# Здесь происходит решение модели ЛП, чтобы найти оптимальное решение, удовлетворяющее всем ограничениям.

model.solve()

# Получение выбранных предметов

selected\_items = [i for i in range(n) if x[i].value() == 1]

# Получение невыбранных предметов

not\_selected\_items = [i for i in range(n) if x[i].value() == 0]

return selected\_items, not\_selected\_items

# Измененные значения весов и стоимостей предметов

weights = [2, 4, 3, 1, 5, 7, 1, 4, 6, 10, 14, 12, 4, 8, 6, 5, 12, 7, 10, 34, 23, 2]

costs = [20, 15, 25, 10, 30, 12, 12, 12, 42, 13, 24, 21, 32, 65, 32, 26, 24, 6, 23, 13, 19, 12]

# Максимальный вес посылки и стоимость ставки в час

max\_weight = 7

max\_cost = 27

print(f'Ограничение по максимальному весу: {max\_weight}\nОграничение по максимальной стоимости: {max\_cost}')

print(f'Список весовых значений: {weights}\nСписок значений цен: {costs}\n')

selected\_items\_dp, not\_selected\_items\_dp = optimal\_assignments(weights, costs, max\_weight, max\_cost)

print(f'ДИНАМИЧЕСКОЕ ПРОГРАММИРОВАНИЕ(Задача об оптимальных назначениях)')

print\_result(selected\_items\_dp, not\_selected\_items\_dp, max\_weight, max\_cost, weights, costs)

selected\_items, not\_selected\_items = knapsack\_multiobjective(weights.copy(), costs.copy(), max\_weight, max\_cost)

print(f'ЛИНЕЙНОЕ ПРОГРАММИРОВАНИЕ ПРОГРАММИРОВАНИЕ(Задача о рюкзаке)')

print\_result(selected\_items, not\_selected\_items, max\_weight, max\_cost, weights, costs)

# **ПРИЛОЖЕНИЕ Б**

(обязательное)

**Графические схемы алгоритмов**

Блок схема математической модели задачи об оптимальных назначениях, реализации программы представлены на рисунках Б.1, Б.2, Б.3.

Подробное описание алгоритма решения задачи об оптимальных назначениях с ограничениями на максимальный вес и ставки в час курьера с использованием метода динамического программирования:

1. Ввод данных:

* Получаем входные данные, включающие количество предметов N, максимальный вес посылки W, максимальную ставку в час C и массивы w[], p[], c[], содержащие веса, стоимости и ставки каждого предмета и курьера соответственно.

1. Создание таблицы DP:

* Создаём таблицу DP размером (N+1) x (W+1) x (C+1), где N - количество предметов, W - максимальный вес посылки, C - максимальная ставка в час.
* Инициализируем значения в таблице DP. Устанавливаем все значения в таблице равными нулю.

1. Установка базовых значений:

* Устанавливаем базовые значения в таблице DP:
* Для всех i от 0 до N: DP[i][0][0] = 0, так как при отсутствии предметов стоимость будет равна нулю.
* Для всех w от 0 до W: DP[0][w][0] = 0, так как при отсутствии предметов стоимость будет равна нулю.
* Для всех c от 0 до C: DP[0][0][c] = 0, так как при отсутствии предметов стоимость будет равна нулю.

1. Вычисление оптимальных назначений:

* Вычисляем оптимальные назначения в таблице DP с помощью динамического программирования.
* Для каждого предмета i от 1 до N:
* Для каждого возможного веса w от 1 до W:
* Для каждой возможной ставки c от 1 до C:
* Если вес предмета i (w[i]) меньше или равен текущему весу w и ставка курьера i (c[i]) меньше или равна текущей ставке c:
* DP[i][w][c] = max(DP[i-1][w][c], DP[i-1][w-w[i]][c-c[i]] + p[i]), выбираем максимальное значение между оставлением предмета и включением предмета в рюкзак.
* Иначе:
* DP[i][w][c] = DP[i-1][w][c], оставляем предмет i без изменений.

5)Восстановление оптимальных назначений:

* Устанавливаем текущие индексы в таблице DP: i = N, w = W, c = C.
* Создаём пустой список Назначения.
* Пока i > 0 и w > 0 и c > 0:
* Если DP[i][w][c] не равно DP[i-1][w][c], значит предмет i был включён в оптимальное назначение:
* Добавляем предмет i в список Назначения.
* Вычитаем вес предмета i (w[i]) из текущего веса w.
* Вычитаем ставку курьера i (c[i]) из текущей ставки c.
* Уменьшаем i на 1.

6)Вывод оптимальных назначений:

* Выводим список Назначения, представляющий оптимальные назначения предметов для курьеров.



Рисунок Б.1 – Блок схема задачи об оптимальных назначениях

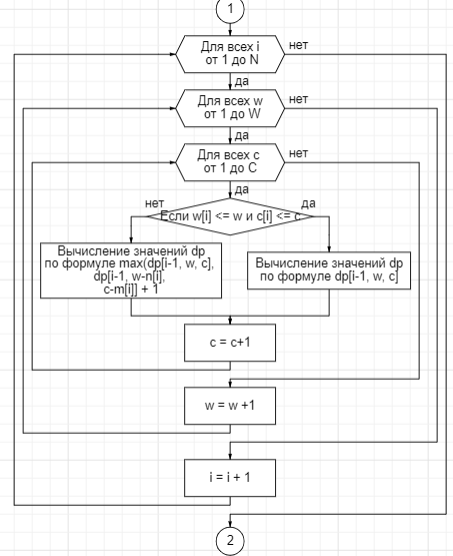


Рисунок Б.2 – Продолжение блок схемы задачи об оптимальных назначениях

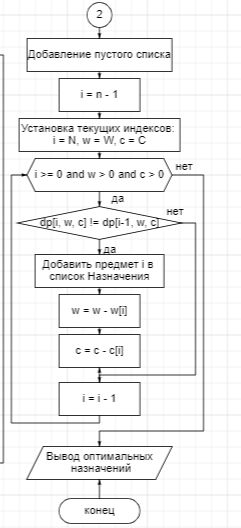


Рисунок Б.3 – Продолжение блок схемы задачи об оптимальных назначениях

Блок схема математической модели задачи о рюкзаке, реализации программы представлены на рисунках Б.4 и Б.5.

Подробное описание алгоритма решения задачи о рюкзаке с ограничениями на максимальный вес посылки и ставку курьера, используя метод линейного программирования:

1. Ввод данных:

Получаем входные данные, включающие количество предметов N, максимальный вес посылки W, максимальную ставку курьера C и массивы w[] и p[], содержащие веса и стоимости каждого предмета соответственно.

1. Создание линейной программы:

* Инициализируем линейную программу с целевой функцией и ограничениями.
* Добавляем переменные решения x[i] для каждого предмета i, обозначающие, будет ли предмет i выбран для помещения в рюкзак (x[i] = 1) или нет (x[i] = 0).
* Добавляем ограничения:
  + Для каждого предмета i от 1 до N:
    - Добавляем ограничение: x[i] >= 0, чтобы переменные решения были неотрицательными.
    - Добавляем ограничение: x[i] <= 1, чтобы переменные решения были не больше 1, что означает, что предмет может быть выбран только один раз.
  + Добавляем ограничение: Сумма весов предметов i \* x[i] <= W, чтобы обеспечить соблюдение ограничения на максимальный вес посылки.
  + Добавляем ограничение: Сумма стоимостей предметов i \* x[i] >= C, чтобы обеспечить соблюдение ограничения на минимальную ставку курьера.
  + Добавляем ограничение: Сумма стоимостей предметов i \* x[i] это целевая функция. Здесь целевая функция может быть максимизирована (если решаем задачу на максимум) или минимизирована (если решаем задачу на минимум).

1. Решение линейной программы:

* Решаем линейную программу с помощью соответствующего алгоритма линейного программирования.
* Получаем значение целевой функции, которое представляет максимальную стоимость рюкзака.

1. Восстановление набора предметов:

* Инициализируем пустой список Набор\_предметов.
* Для каждой переменной решения x[i]:
  + Если x[i] > 0, добавляем предмет i в Набор\_предметов, так как он был выбран для помещения в рюкзак.

1. Вывод результатов:

* Выводим максимальную стоимость рюкзака, полученную из значения целевой функции, и список Набор\_предметов, представляющий оптимальный набор предметов, выбранных для рюкзака.

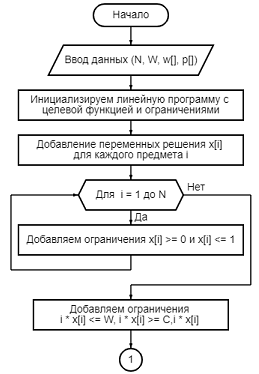


Рисунок Б.4 – Блок схема задачи о рюкзаке

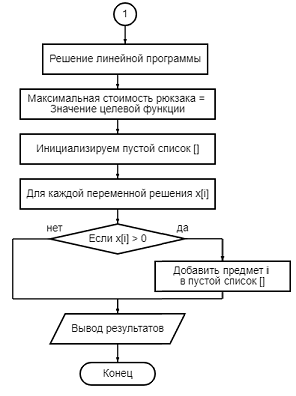


Рисунок Б.5 – Продолжение блок схемы задачи о рюкзаке

Блок схемы реализации программы задачи об назначениях представлены на рисунках Б.6 – Б.9.

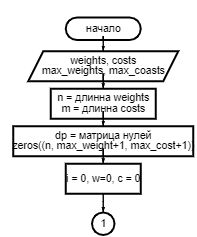
****

Рисунок Б.6 –Блок схема задачи об оптимальных назначениях

1. Начало блок схемы;
2. Вводим переменные;
3. Переменные n и m инициализируются как количество предметов и количество курьеров соответственно, используя функцию len() для получения длины переданных списков weights и costs;
4. Создается трехмерная матрица dp с размерностью (n, max\_weight+1, max\_cost+1) с помощью функции np.zeros(). Эта матрица будет использоваться для хранения результатов;
5. Начальные значения для переменных i, w и c, которые используются в циклах указывают на начальную точку перебора. В данном случае, i = 0 означает, что мы начинаем перебор предметов с индекса 0 (первый предмет), w = 0 означает, что мы начинаем перебор значений веса с 0, и c = 0 означает, что мы начинаем перебор значений стоимости с 0.

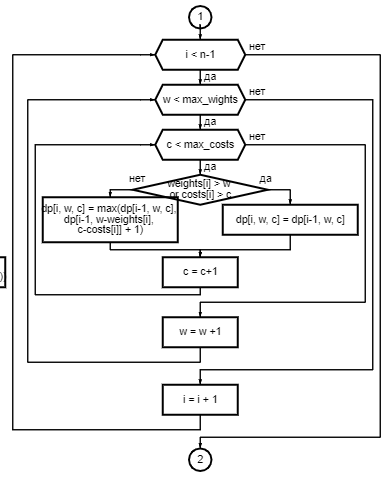
****

Рисунок Б.7 – Продолжение блок схемы задачи об оптимальных назначениях

1. Вложенные циклы используются для заполнения матрицы dp. Внешний цикл i перебирает предметы от 0 до n-1. Внутренние циклы w и c перебирают возможные значения веса и стоимости соответственно.

* На каждом шаге проверяется, если вес предмета i (weights[i]) больше текущего значения веса w или стоимость предмета i (costs[i]) больше текущего значения стоимости c. Если это условие выполняется, значит предмет i не может быть включен в текущее решение, и его значение в матрице dp берется из предыдущего шага (dp[i-1, w, c]).
* В противном случае, когда предмет i может быть включен, выбирается максимум из двух вариантов: включить предмет i (dp[i-1, w-weights[i], c-costs[i]] + 1) или не включать его (dp[i-1, w, c]). Полученное значение записывается в матрицу dp[i, w, c].

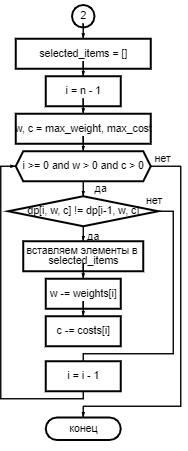
****

Рисунок Б.8 – Продолжение блок схемы задачи об оптимальных назначениях

1. selected\_items = []: Создается пустой список selected\_items, который будет содержать индексы выбранных предметов.
2. i = n - 1: Инициализируется переменная i значением n - 1, где n - количество предметов. Это означает, что мы начинаем с последнего предмета и будем двигаться в обратном порядке.
3. w, c = max\_weight, max\_cost: Инициализируются переменные w и c значениями max\_weight и max\_cost соответственно. Эти переменные будут использоваться для отслеживания оставшегося веса и стоимости при восстановлении оптимального назначения.
4. Начинается цикл while, который выполняется, пока i больше или равно 0 и значения w и c больше 0. Это означает, что мы продолжаем идти назад по предметам, пока не достигнем начала или не исчерпаем ограничения на вес и стоимость.
5. Внутри цикла проверяется, если значение в матрице dp на текущем шаге отличается от значения на предыдущем шаге. Это означает, что предмет i был включен в оптимальное назначение.
6. selected\_items.append(i): Если предмет i был включен в оптимальное назначение, его индекс i добавляется в список selected\_items.
7. w -= weights[i] и c -= costs[i]: Значения w и c уменьшаются на вес и стоимость выбранного предмета соответственно. Это обновляет оставшийся доступный вес и стоимость для следующих итераций цикла.
8. i -= 1: Индекс i уменьшается на 1, чтобы перейти к следующему предмету в обратном порядке.
9. По завершении цикла while список selected\_items будет содержать индексы выбранных предметов на основе оптимального назначения и ограничений на вес и стоимость.

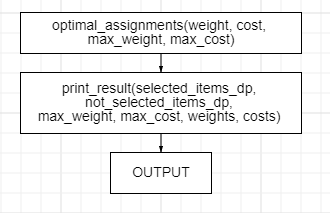


Рисунок Б.9 – Продолжение блок схемы задачи об оптимальных назначениях

Блок схемы реализации программы задачи о рюкзаке представлены на рисунках Б.10 – Б.15.

1. Вызывается функция optimal\_assignments, которая принимает веса weights, стоимости costs, ограничение на общий вес max\_weight и ограничение на общую стоимость max\_cost. Результаты сохраняются в переменные selected\_items\_dp и not\_selected\_items\_dp.
2. Затем вызывается функция print\_result, которая принимает selected\_items\_dp, not\_selected\_items\_dp, max\_weight, max\_cost, weights и costs. Данная функция выводит результаты на экран.
3. Затем выводятся результаты в консоль.

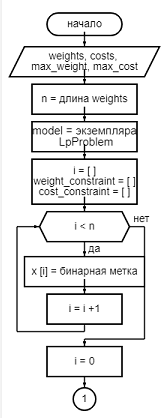
****

Рисунок Б.10 – Блок схема задачи о рюкзаке

1. Начало программы
2. Вводим переменные
3. Вычисляется количество предметов n на основе длины списка weights, который содержит веса предметов.
4. Создается экземпляр модели линейного программирования. LpProblem принимает два аргумента: имя модели ("KnapsackMultiobjective") и тип задачи оптимизации (LpMaximize, что означает поиск максимума). Мы хотим максимизировать общую стоимость выбранных предметов.
5. Создается список бинарных переменных x, где каждая переменная x[i] соответствует выбору предмета с индексом i. LpVariable принимает два аргумента: имя переменной (в данном случае f"Предмет {i}") и тип переменной (cat="Binary", означающий, что переменная принимает значения True (1) или False (0)).

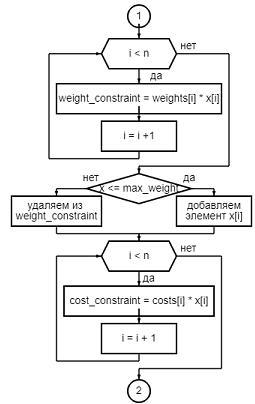
****

Рисунок Б.11 – Продолжение блок схемы задачи о рюкзаке

1. Создается ограничение на общий вес выбранных предметов. lpSum используется для вычисления суммы весов предметов, умноженных на соответствующие переменные x[i]. Ограничение задается с помощью оператора <= (меньше или равно) и ограничения max\_weight.
2. Создается ограничение на общую стоимость выбранных предметов. Аналогично предыдущему шагу, lpSum используется для вычисления суммы стоимостей предметов, умноженных на соответствующие переменные x[i]. Ограничение задается с помощью оператора <= (меньше или равно) и ограничения max\_cost.

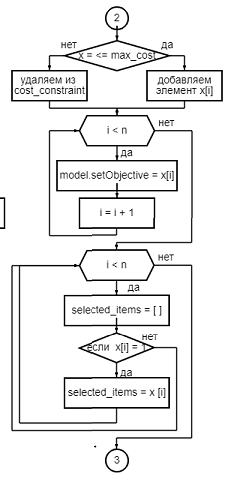
****

Рисунок Б.12 – Продолжение блок схемы задачи о рюкзаке

1. objective = lpSum(x[i] for i in range(n)): Создается целевая функция objective с помощью функции lpSum. В данном случае, целевая функция представляет собой сумму всех переменных x[i], где i принимает значения от 0 до n-1. Это означает, что мы хотим максимизировать сумму выбранных предметов.
2. model.setObjective(objective): Целевая функция objective устанавливается для модели с помощью метода setObjective. Это указывает модели, что мы хотим максимизировать значение целевой функции при решении задачи.
3. model.solve(): Происходит решение модели линейного программирования с помощью метода solve(). Здесь модель находит оптимальное решение, удовлетворяющее всем ограничениям.
4. selected\_items = [i for i in range(n) if x[i].value() == 1]: Создается список selected\_items, который содержит индексы выбранных предметов. Мы проходимся по всем возможным индексам i от 0 до n-1 и проверяем значение переменной x[i]. Если значение равно 1, это означает, что предмет с индексом i был выбран (так как переменная x[i] принимает значение True (1)). В этом случае, индекс i добавляется в список selected\_items.

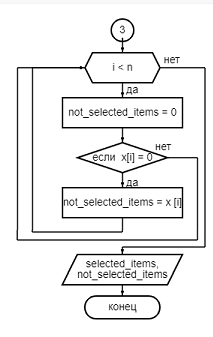
****

Рисунок Б.13 – Продолжение блок схемы задачи о рюкзаке

1. Создается список not\_selected\_items, который содержит индексы предметов, которые не были выбраны. Мы проходимся по всем возможным индексам i от 0 до n-1 и проверяем значение переменной x[i]. Если значение равно 0, это означает, что предмет с индексом i не был выбран (поскольку переменная x[i] принимает значение False (0)). В этом случае, индекс i добавляется в список not\_selected\_items.
2. Вывод selected\_items и not\_selected\_items

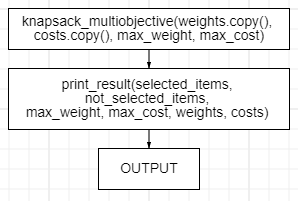


Рисунок Б.14 – Продолжение блок схемы задачи о рюкзаке

1. Вызывается функция knapsack\_multiobjective, которая принимает копии массивов weights и costs, ограничение на общий вес max\_weight и ограничение на общую стоимость max\_cost. Результаты сохраняются в переменные selected\_items и not\_selected\_items.
2. Затем вызывается функция print\_result, которая принимает selected\_items, not\_selected\_items, max\_weight, max\_cost, weights и costs. Данная функция выводит результаты на экран.
3. Затем выводятся результаты в консоль.

# **ПРИЛОЖЕНИЕ В**

(обязательное)

**Схемы алгоритмов**