МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования

«Гомельский государственный технический университет

имени П. О. Сухого»

Факультет автоматизированных и информационных систем

Кафедра «Информационные технологии»

Специальность 1-40 05 01-01 Информационные системы и технологии

(в проектировании и производстве)

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

к курсовому проекту

по дисциплине «Оптимизация проектного решения»

на тему:

«Оптимизация проектного решения»

Исполнитель: студент гр. ИТП-41

Федоренко С.Н.

Руководитель: профессор

Михайлов М.И.

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

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

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

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

Подписи членов комиссии

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

Гомель 2023

**СОДЕРЖАНИЕ**

[Введение 4](#_Toc153509888)

[1 Обзор методов оптимизации задачи о рюкзаке и транспортной задачи 5](#_Toc153509889)

[1.1 Постановка и цель транспортной задачи 5](#_Toc153509890)

[1.2 Обзор методов решения транспортной задачи 5](#_Toc153509891)

[1.3 Постановка и цель задачи о рюкзаке 10](#_Toc153509892)

[1.4 Обзор методов решения задачи о рюкзаке 10](#_Toc153509893)

[1.5 Выбор языка программирования 13](#_Toc153509894)

[2 Математическая модель оптимизационной](#_Toc153509895) [задачи 14](#_Toc153509896)

[2.1 Метод северо-западного угла в многокритериальных системах 14](#_Toc153509897)

[2.2 Метод динамического программирования в многокритериальных системах 19](#_Toc153509898)

[3 Программная реализация задачи 23](#_Toc153509899)

[3.1 Архитектура приложения 23](#_Toc153509900)

[3.2 Описание исходных данных 24](#_Toc153509901)

[3.3 Структура программного комплекса 26](#_Toc153509902)

[3.4 Функционал программного продукта 27](#_Toc153509903)

[4 Верификация программных средств 32](#_Toc153509904)

[4.1 Верификация транспортной задачи 32](#_Toc153509905)

[4.2 Верификация задачи о рюкзаке 35](#_Toc153509906)

[Заключение 39](#_Toc153509907)

[Список использованных источников 40](#_Toc153509908)

[Приложение А Листинг программы 41](#_Toc153509909)

[Приложение Б Схемы алгоритмов 50](#_Toc153509910)

# **ВВЕДЕНИЕ**

Эффективность и производительность программных систем в программировании тесно связаны с оптимизацией проектных решений. Она представляет собой сложный и важный процесс, направленный на достижение оптимального использования ресурсов, сокращение времени выполнения программ и минимизацию использования системных ресурсов. В условиях постоянного увеличения сложности программных проектов и повышенных требований к их быстродействию, оптимизация становится особенно актуальной.

В современном программировании, где часто возникают ситуации с многозадачностью, большими объемами данных и высокими требованиями к отзывчивости системы, оптимизация становится неотъемлемой частью работы. Это особенно важно в областях, где производительность является критическим фактором, таких как вычислительная наука, разработка игр, обработка больших данных и прочие.

Оптимизация позволяет более эффективно использовать доступные аппаратные и программные ресурсы, что способствует сокращению энергопотребления. В условиях растущей проблемы энергетической эффективности в современных вычислительных системах, оптимизация способствует разработке программного обеспечения, которое более устойчиво и экологически чисто.

Оптимизация кода может значительно сократить время, затрачиваемое на разработку и поддержку программных продуктов. Улучшенная производительность, оптимальная структура и архитектура кода снижают количество ошибок и ускоряют процесс разработки, что в свою очередь уменьшает трудозатраты на обслуживание.

Применение оптимизированных подходов оказывает воздействие на восприятие пользователей. Повышение реактивности приложений, сокращение времени ожидания и улучшение производительности программного обеспечения непосредственно влияют на уровень удовлетворенности конечных пользователей.

Оптимизация играет важную роль в процессе архитектурного проектирования. При выборе алгоритмов, структур данных и использовании многопоточности уделяется особое внимание оптимизации, с целью создать системы, которые являются более эффективными и масштабируемыми.

Все указанные аспекты подчеркивают важность оптимизации в программировании. Она оказывает влияние не только на технические аспекты разработки программного обеспечения, но и на экономику, устойчивость и удовлетворенность конечных пользователей. Оптимизация продолжает оставаться в фокусе внимания разработчиков, и их стремление к созданию высокопроизводительных и эффективных систем будет продолжаться в будущих технологических сдвигах.

# **1 ОБЗОР МЕТОДОВ ОПТИМИЗАЦИИ ЗАДАЧИ О РЮКЗАКЕ И**

# **ТРАНСПОРТНОЙ ЗАДАЧИ**

* 1. **Постановка и цель транспортной задачи**

Транспортная задача является одной из классических задач линейного программирования. Она возникает в ситуациях, когда требуется определить оптимальный план перевозки ресурсов из источников (например, заводов или складов) к потребителям (например, магазинам или клиентам) с минимальными затратами [5].

Постановка транспортной задачи включает в себя следующие элементы:

* матрица стоимостей перевозок: Это двумерный массив, в котором каждый элемент представляет стоимость перевозки единицы ресурса из определенного источника к определенному потребителю. Эти стоимости могут быть, например, расстояниями, временем или денежными затратами;
* векторы объемов: Вектор объемов источников определяет доступные объемы ресурсов в каждом источнике, а вектор объемов потребителей определяет потребности в ресурсах у каждого потребителя.

Цель транспортной задачи состоит в нахождении плана перевозок, который минимизирует общие затраты при учете ограничений объемов и потребностей. Оптимальный план перевозок должен удовлетворять следующим условиям:

* общий объем перевозимых ресурсов из каждого источника должен быть равен его доступному объему;
* общий объем перевозимых ресурсов к каждому потребителю должен быть равен его потребности;
* общие затраты на перевозки должны быть минимальными.

Решение транспортной задачи может быть достигнуто с использованием различных методов оптимизации, включая симплекс-метод, метод потенциалов или методы северо-западного угла и минимальной стоимости. Эти методы позволяют найти оптимальное распределение ресурсов между источниками и потребителями, учитывая ограничения и целевую функцию.

* 1. **Обзор методов решения транспортной задачи**

Решение транспортной задачи представляет собой критически важный этап в управлении логистикой и цепями поставок, требующий применения сложных методов и алгоритмов оптимизации на основе линейного программирования. Эти методы играют ключевую роль в достижении оптимальных планов перевозок, обеспечивая эффективное использование ресурсов и оптимизацию затрат.

Одним из основных подходов к решению транспортной задачи является применение метода линейного программирования (ЛП). В этом подходе задача формулируется в виде линейной целевой функции, которую необходимо минимизировать при соблюдении линейных ограничений. В случае транспортной задачи, переменные соответствуют количеству груза, перевозимого между различными пунктами, а ограничения связаны с объемом предложения и спроса на ресурсы. Применение линейного программирования позволяет найти оптимальное решение, учитывая ограничения и целевую функцию, и оптимизировать перевозки с учетом экономических и логистических факторов.

Существует несколько методов для решения транспортной задачи:

– метод минимального элемента;

– метод северо-западного угла;

– метод потенциалов.

Эти методы предлагают различные подходы к решению транспортной задачи, учитывая ограничения и целевую функцию. Каждый метод имеет свои особенности и может быть предпочтительным в определенных ситуациях, в зависимости от размера задачи и ее особенностей.

* + 1. *Метод минимального элемента*

Метод минимального элемента является одним из приближенных методов решения задач оптимизации проектных решений. Этот метод используется для нахождения начального приближения оптимального решения и может быть последующим шагом в итерационном процессе оптимизации.

Основные шаги метода:

1. шаг 1: формулировка задачи оптимизации проектных решений. Сначала необходимо ясно сформулировать задачу оптимизации проектных решений. Это включает в себя определение целевой функции, ограничений и переменных, которые будут оптимизироваться;
2. шаг 2: создание начального приближения. На этом шаге создается начальное приближение оптимального решения. Одним из подходов является использование метода минимального элемента. Для этого строится таблица, в которой указываются стоимости (или другие соответствующие значения) переходов между различными переменными или решениями;
3. шаг 3: выбор минимального элемента. Находится минимальный элемент в таблице, который будет оптимальным выбором для текущего шага. Этот элемент указывает на переход, который будет иметь наименьшую стоимость или наибольшую пользу, в зависимости от постановки задачи;
4. шаг 4: обновление таблицы. После выбора минимального элемента происходит обновление таблицы путем вычитания этого элемента из остальных элементов в его строке и столбце. Это может включать перераспределение ресурсов или переходы между переменными;
5. шаг 5: проверка условия остановки. После обновления таблицы происходит проверка условия остановки. Это может быть достижение определенного критерия оптимальности или выполнение других предопределенных условий;
6. шаг 6: итерационный процесс. Если условие остановки не выполняется, происходит повторение шагов 3-5 для получения нового приближения оптимального решения. Этот процесс может продолжаться до достижения оптимального результата или до выполнения других заданных критериев остановки;
7. шаг 7: вывод оптимального решения. По завершении итерационного процесса получается оптимальное решение задачи оптимизации проектных решений. Это решение соответствует наилучшей комбинации переменных, удовлетворяющей заданным ограничениям и минимизирующей (или максимизирующей) целевую функцию.

Метод минимального элемента обладает высокой алгоритмической сложностью в сравнении с более простым методом северо-западного угла. Однако, он обеспечивает более точные результаты и может быть предпочтителен в случаях, где точность имеет приоритет.

Метод минимального элемента находит широкое применение в областях логистики, транспортировки и поставок. Его способность обрабатывать сложные сценарии с различными стоимостями перевозок и ограничениями делает его ценным инструментом для оптимизации бизнес-процессов.

Несмотря на свою эффективность, метод минимального элемента имеет свои ограничения. В частности, он может столкнуться с проблемами в случае наличия большого числа переменных и сложных структур таблицы стоимостей.

Важно отметить, что метод минимального элемента является приближенным методом и может не гарантировать достижение глобального оптимума. Он может использоваться как начальный шаг в более сложных методах оптимизации проектных решений для получения более точного результата.

* + 1. *Метод северо-западного угла*

Транспортная задача, входящая в область линейного программирования, относится к задачам оптимизации перевозки грузов между различными пунктами. Один из известных методов приближенного решения этой задачи – метод северо-западного угла. Этот метод обладает простотой и понятностью, что способствует его широкому применению на практике.

Основные шаги метода:

1. шаг 1: начало и исходные данные: Метод начинает заполнять таблицу перевозок, начиная с верхнего левого угла (северо-западного угла). Исходные данные для метода включают доступные объемы ресурсов в источниках, потребности потребителей и стоимости перевозок между каждой парой источников и потребителей;
2. шаг 2: заполнение ячеек таблицы: Начиная с северо-западной ячейки, метод заполняет ячейки таблицы по порядку, двигаясь вправо и вниз. В каждой ячейке указывается количество ресурсов, которое можно перевезти, исходя из доступных объемов в источнике и потребностей потребителя;
3. шаг 3: проверка ограничений: После заполнения ячеек таблицы происходит проверка удовлетворения ограничений. Общий объем ресурсов, перевозимых из каждого источника, должен быть равен его доступному объему. Аналогично, общий объем ресурсов, перевозимых к каждому потребителю, должен быть равен его потребности;
4. шаг 4: оптимальность решения: Если заполненная таблица удовлетворяет ограничениям, она является потенциально оптимальным решением. Однако, в большинстве случаев, заполненная таблица не является оптимальным решением;
5. шаг 5: перераспределение ресурсов: Чтобы достичь оптимального решения, происходит перераспределение ресурсов в таблице. Это происходит путем выбора ячейки с наибольшей стоимостью перевозки и переноса ресурсов из этой ячейки в другие ячейки с более низкой стоимостью перевозки, при условии соблюдения ограничений;
6. шаг 6: повторение шагов 4 и 5: Перераспределение ресурсов повторяется до достижения оптимального решения, когда все ограничения соблюдаются и общая стоимость перевозок минимизирована.

Метод северо-западного угла обладает значительным преимуществом в своей эффективности. Он позволяет немедленно приступить к распределению грузов, начиная с верхнего левого угла и двигаясь по диагонали. Этот подход сокращает количество необходимых шагов для получения начального приближенного решения. В реальных ситуациях, где оперативность имеет важное значение, данная скорость может играть решающую роль.

Применение метода северо-западного угла может привести к сокращению затрат ресурсов, поскольку его использование не требует сложного вычислительного оборудования или высокой квалификации специалистов. Это особенно важно в областях, где доступность и оперативность принятия решений имеют первостепенное значение.

Метод северо-западного угла обладает высокой эффективностью, но не всегда гарантирует достижение оптимального решения. В определенных ситуациях требуется использование дополнительных итерационных методов для получения более точного результата.

Метод северо-западного угла представляет собой значимый инструмент для решения транспортных задач благодаря своей простоте и понятности. Его эффективность и удобство делают его широко применимым в реальных условиях логистического планирования и управления цепями поставок.

* + 1. *Метод потенциалов*

В современных исследованиях операций и оптимизации существует ряд методов для решения транспортных задач, среди которых выделяется метод потенциалов. Этот метод базируется на принципах линейного программирования и предоставляет эффективный инструмент для оптимизации распределения ресурсов в условиях транспортных логистических задач. В данном разделе рассматриваются ключевые аспекты метода потенциалов, его применимость и особенности.

Метод потенциалов основан на применении понятия потенциалов к узлам сети транспортной задачи. Рассмотрим транспортную сеть, представленную матрицей стоимостей перевозок, где каждый узел соответствует пересечению строки и столбца. В методе потенциалов вводятся потенциалы для каждого узла, ассоциированные с соответствующими строками и столбцами. Эти потенциалы позволяют эффективно определить стоимость перевозки между узлами.

Основные шаги метода:

1. шаг 1: формулировка задачи транспортной задачи. Определить количество поставщиков и потребителей. Задать стоимости перевозок и объемы поставок. Обеспечить равенство сумм поставок и потребностей;
2. шаг 2: создание начального базисного плана. Создать начальный базисный план, заполнив ненулевые ячейки. Назначить переменные потенциалов для каждого поставщика и потребителя;
3. шаг 3: оценка потенциалов. Рассчитать потенциалы для каждого поставщика и потребителя на основе текущего базисного плана. Использовать метод разностей или другие подходы для определения потенциалов;
4. шаг 4: расчет оценок ячеек. Рассчитать оценки для каждой ненулевой ячейки в базисном плане. Оценка ячейки представляет собой разность между стоимостью перевозки и суммой потенциалов соответствующего поставщика и потребителя;
5. шаг 5: проверка оптимальности. Проверить условие оптимальности: если все оценки ячеек неотрицательны, то текущий базисный план является оптимальным решением. Если есть отрицательные оценки, перейти к следующему шагу;
6. шаг 6: поиск улучшающей ячейки. Выбрать ячейку с отрицательной оценкой, которая будет улучшать общую стоимость перевозок. Эта ячейка станет новой базисной ячейкой, а одна из соседних ячеек, входящих в текущий базис, будет удалена из базиса;
7. шаг 7: обновление базисного плана. Обновить базисный план, добавив новую базисную ячейку и удалив одну из соседних базисных ячеек. Пересчитать потенциалы для обновленного базисного плана;
8. шаг 8: повторение шагов 3-7. Повторять шаги 3-7 до тех пор, пока не будет достигнуто оптимальное решение (все оценки ячеек неотрицательны);
9. шаг 9: вывод оптимального решения. Когда все оценки ячеек станут неотрицательными, текущий базисный план будет оптимальным решением задачи транспортной логистики.

Метод потенциалов является эффективным инструментом для быстрого нахождения оптимальных решений в транспортных задачах с большим объемом данных. Его относительная простота позволяет использовать его широко даже без глубоких знаний линейного программирования. Адаптивность метода к различным формам транспортных задач и сетей делает его универсальным инструментом оптимизации. Метод потенциалов активно применяется в таких областях, как логистика, транспортное планирование и управление цепями поставок. Благодаря его способности решать задачи с большим числом переменных и ограничений, метод потенциалов остается востребованным инструментом в современных бизнес-процессах.

Метод потенциалов является мощным инструментом оптимизации, который позволяет решать сложные транспортные задачи с высокой эффективностью, применимостью и относительной простотой реализации.

* 1. **Постановка и цель задачи о рюкзаке**

Задача о рюкзаке является классической задачей комбинаторной оптимизации, которая возникает в контексте выбора оптимального набора предметов для помещения в рюкзак с ограниченной вместимостью. Формально, дано множество предметов, каждый из которых характеризуется своим весом и стоимостью. Требуется выбрать подмножество предметов таким образом, чтобы их суммарный вес не превышал заданную вместимость рюкзака, а суммарная стоимость была максимальной.

Цель задачи о рюкзаке состоит в нахождении оптимального решения, которое обеспечивает максимальную суммарную стоимость предметов при соблюдении ограничения на вместимость рюкзака.

Задача о рюкзаке имеет важные практические применения. Например, в области логистики и управления запасами она может быть использована для оптимального планирования ресурсов и управления запасами товаров. В финансовой сфере задача о рюкзаке может быть применена для оптимизации портфелей инвестиций, где предметами могут выступать различные финансовые инструменты, а ограничением является доступный капитал. Кроме того, задача о рюкзаке нашла применение в других областях, таких как планирование производства, ресурсное планирование и оптимизация процессов принятия решений.

Основная сложность задачи о рюкзаке заключается в том, что для поиска оптимального решения необходимо перебрать все возможные комбинации предметов, что может быть вычислительно затратно при большом количестве предметов. В связи с этим были разработаны различные эвристические и приближенные алгоритмы для решения задачи о рюкзаке, позволяющие найти хорошее приближенное решение за разумное время.

* 1. **Обзор методов решения задачи о рюкзаке**

Задача о рюкзаке, представляет собой проблему оптимизации, требующую эффективного выбора предметов для максимизации ценности при ограниченных ресурсах. Существует несколько методов решения этой задачи, каждый из которых имеет свои уникальные характеристики и применимость в различных контекстах [1].

Существует несколько методов для решения задачи о рюкзаке:

* метод полного перебора: перебор всех возможных комбинаций предметов для нахождения оптимального решения. Этот метод гарантирует точное решение, но может быть вычислительно затратным при большом количестве предметов;
* жадные алгоритмы: алгоритмы, основанные на принципе выбора предметов с максимальным отношением стоимости к весу на каждом шаге. Жадные алгоритмы предлагают быстрые и простые решения, но могут не обеспечивать оптимальное решение;
* динамическое программирование: алгоритмы, основанные на разбиении задачи на подзадачи и сохранении оптимальных решений для каждой подзадачи. Динамическое программирование позволяет эффективно находить оптимальное решение, но требует больше вычислительных ресурсов;
* методы ветвей и границ: алгоритмы, которые систематически разбивают пространство поиска на подзадачи и оценивают верхнюю и нижнюю границы для оптимального решения. Метод ветвей и границ обеспечивает точное решение, но требует больше вычислительного времени;

Каждый из этих методов имеет свои преимущества и ограничения, и выбор метода зависит от конкретной задачи и требований к решению.

Далее будут расспотрены некоторые из них.

***1.4.1*** *Жадный алгоритм*

Задача о рюкзаке представляет интерес в мире оптимизации, и различные методы применяются для ее решения. Одним из таких методов является жадный алгоритм. В контексте задачи о рюкзаке жадные алгоритмы стремятся сделать локально оптимальный выбор на каждом этапе, в надежде достичь глобальной оптимальности [7].

Основная идея жадного метода заключается в том, чтобы на каждом шаге выбирать элемент с наилучшим отношением стоимость/вес, стремясь максимизировать этот коэффициент.

Жадный алгоритм обычно применяется в версии задачи о рюкзаке без возврата, что означает, что каждый элемент может быть использован только один раз.

Часто требуется предварительная сортировка элементов по убыванию их стоимости/веса, чтобы жадный алгоритм мог последовательно выбирать наилучшие элементы.

Основные шаги алгоритма:

1. инициализировать пустой рюкзак и нулевую общую стоимость;
2. отсортировать элементы в порядке убывания отношения стоимость/вес;
3. пока есть доступные элементы и рюкзак не превышает грузоподъемность:
   1. выбрать элемент с наилучшим отношением стоимость/вес, который еще не добавлен в рюкзак и не вызывает превышение грузоподъемности;
   2. добавить выбранный элемент в рюкзак;
   3. увеличить общую стоимость на стоимость выбранного элемента;
   4. обновить вес рюкзака.
4. вернуть рюкзак и общую стоимость.

Преимущества жадного алгоритма для задачи о рюкзаке включают его простоту и относительную эффективность в случае задач с ограниченными данными. Однако, жадные методы могут не гарантировать глобальную оптимальность и могут давать подходы, которые не являются оптимальными в общем случае. Это важно учитывать в контексте конкретной задачи и ее требований [2].

***1.4.2*** *Метод динамического программирования*

Задача о рюкзаке представляет собой классическую задачу оптимизации, цель которой состоит в выборе определенного набора предметов с ограниченными ресурсами (весом). Метод динамического программирования (ДП) является мощным инструментом для решения этой задачи, обеспечивая эффективное и оптимальное решение.

Одним из ключевых принципов метода ДП является разбиение задачи на более мелкие и управляемые подзадачи. В случае задачи о рюкзаке, это означает разделение задачи выбора предметов на задачи выбора поднаборов предметов с учетом оставшейся грузоподъемности.

Оптимальное решение задачи о рюкзаке строится на основе оптимальных решений подзадач. Для каждой подзадачи определяется, какие предметы следует включить в рюкзак для максимизации общей стоимости.

Одним из ключевых элементов метода ДП является использование таблицы, где значения заполняются пошагово. В случае задачи о рюкзаке, двумерная таблица может использоваться для отслеживания максимальной стоимости для различных комбинаций предметов и оставшейся грузоподъемности.

Основные шаги алгоритма:

1. создается двумерная таблица, инициализируются базовые значения, такие как максимальная стоимость для нулевого веса или нулевого количества предметов;
2. саблица заполняется пошагово, рассматривая каждую подзадачу. Для каждой ячейки определяется максимальная стоимость, исходя из решений предыдущих подзадач;
3. по завершении заполнения таблицы восстанавливается оптимальное решение, определяя, какие предметы включены в рюкзак.

Преимущества метода ДП в решении задачи о рюкзаке включают его эффективность и способность дать точное оптимальное решение. Однако, этот метод может иметь высокую вычислительную сложность для больших наборов данных из-за размера таблицы, что следует учитывать при выборе подходящего метода в зависимости от контекста задачи.

***1.4.3*** *Метод ветвей и границ*

Задача о рюкзаке является классической задачей комбинаторной оптимизации, где требуется выбрать определенные элементы из множества так, чтобы их суммарный вес не превышал заданную грузоподъемность, а суммарная стоимость была максимальной. Метод ветвей и границ является эффективным алгоритмом для решения этой задачи.

На каждом этапе алгоритма рассматривается текущее решение задачи. Если оно не является целочисленным (т.е., допускает выбор части элементов), задача разбивается на две ветви.

Для каждой ветви вычисляется верхняя граница стоимости решения. Это значение используется для оценки того, стоит ли исследовать данную ветвь или отбросить ее.

Процесс разделения и оценки границы повторяется для каждой ветви до тех пор, пока не будет найдено оптимальное целочисленное решение или пока не будут исследованы все ветви.

Основные шаги алгоритма:

1. создать начальную задачу о рюкзаке с непрерывными переменными;
2. если текущее решение является нецелочисленным, разделить задачу на две ветви, добавив ограничение на один из элементов;
3. для каждой ветви вычислить оценку границы, основываясь на уже выбранных элементах и оставшихся частях элементов;
4. рекурсивно применить шаги 2-3 для каждой ветви до тех пор, пока не будет достигнуто оптимальное целочисленное решение.

Метод ветвей и границ является мощным инструментом для решения задачи о рюкзаке, поскольку он позволяет систематически исследовать пространство решений и находить оптимальные значения. Однако, его эффективность может зависеть от структуры конкретной задачи, и в некоторых случаях другие методы, такие как динамическое программирование или жадные алгоритмы, могут быть более эффективными.

Метод ветвей и границ предоставляет эффективный и общий метод решения задачи о рюкзаке. Этот метод представляет собой мощный инструмент для оптимизации, обеспечивая баланс между точностью и эффективностью в решении задач комбинаторной оптимизации.

* 1. **Выбор языка программирования**

Выбор языка программирования зависит от ряда факторов. Для реализации оптимизационных методом наиболее подходят объектно-ориентированные языки программирования.

К объектно-ориентированному языкам программирования относятся языки *C*++, *C*#, *Python*. Эти языки поддерживают основные концепции ООП.

*С*++ – это высокоуровневый язык программирования, который широко используется для разработки программного обеспечения и операционных систем.

*C*# является одним из самых популярных языков программирования и используется для создания различных типов приложений, таких как *Windows*-приложения, игры, веб-приложения и мобильные приложения [3].

*Python* – это очень популярный язык программирования, который имеет множество преимуществ и недостатков. *Python* – язык программирования, который не является строго типизированным, в отличие от двух рассмотренных выше языков. Это может облегчить работу с матричными объектами, но также усложнить процесс обнаружения ошибок при написании кода.

Для выполнения курсового проекта был выбран язык *C*#.

# **2 МАТЕМАТИЧЕСКАЯ МОДЕЛЬ ОПТИМИЗАЦИОННОЙ**

# **ЗАДАЧИ**

**2.1 Метод северо-западного угла в многокритериальных системах**

***2.1.1*** Математическая модель метода северо-западного угла

Метод северо-западного угла является одним из классических алгоритмов в линейном программировании, используемых для решения задачи о транспортной оптимизации. Однако, когда речь идёт о многокритериальных системах, применение этого метода может потребовать некоторой модификации или расширения.

В многокритериальных системах решается задача оптимизации с учетом нескольких критериев, которые могут конфликтовать между собой. В контексте задачи о транспортной оптимизации, критерии могут быть связаны с минимизацией стоимости перевозок, минимизацией времени доставки или максимизацией использования ресурсов.

Для решения многокритериальных задач используются различные методы, такие как метод взвешенных сумм, метод ограничений и другие. Они позволяют учитывать важность каждого критерия и находить компромиссные решения [6].

Вместо использования метода северо-западного угла напрямую, в многокритериальных системах может быть целесообразнее применять более сложные алгоритмы, учитывающие не только количество ресурсов и потребностей каждой точки, но и оптимизирующие значения критериев. Некоторые из таких алгоритмов включают метод Парето, алгоритм эволюционной оптимизации и другие.

Таблица 2.1 – Характеристика транспортной задачи

|  |  |
| --- | --- |
| Характеристика | Описание |
| 1 | 2 |
| Цель | Цель транспортной задачи заключается в оптимальном распределении товаров или ресурсов из источников (поставщиков) в места назначения (потребители) с минимальными затратами или максимальной выгодой. |
| Исходные данные | Входными данными для транспортной задачи являются список поставщиков, список потребителей и матрица стоимостей. |
| Ограничения | Ограничениями в данной задаче являются ограниченные поставки товара. |
| Результат | Результат транспортной задачи представляет собой матрицу, являющуюся оптимальным планом перевозок, который минимизирует затраты. |

Графическая схема метода северо-западного угла представлена на рисунке 2.1.

Инициализация вектора поставщиков , где *m –* количество поставщиков, вектора потребителей , где *n –* количество потребителей, матрицы стоимостей перевозок , где *–* стоимость перевозки груза *i* от поставщика до потребителя *j*

Задание начальной позиции

*i =* 1, *j* *=* 1

*j* < *n*

Начало

;

Конец

да

нет

0

нет

да

Рисунок 2.1 – Графическая схема метода северо-западного угла

Алгоритм метода северо-западного угла для решения транспортной задачи может быть описан следующими шагами:

* шаг 1 – инициализация вектора поставщиков , где *m –* количество поставщиков, вектора потребителей , где *n –* количество потребителей, матрицы стоимостей перевозок , где *–* стоимость перевозки груза *i* от поставщика до потребителя *j*;
* шаг 1 – инициализация вектора поставщиков , где *m –* количество поставщиков, вектора потребителей , где *n –* количество потребителей, матрицы стоимостей перевозок , где *–* стоимость перевозки груза *i* от поставщика до потребителя *j*;
* шаг 2 – задание начальной позиции *i =* 1, *j* *=* 1;
* шаг 3 – присвоение текущему элементу опорного плана значение минимального значения из *j*-го поставщика и *i*-го потребителя (*xi*,*j* = *min*(*ai*, *bj*));
* шаг 4 – вычитание у *j*-го поставщика значение текущего элемента опорного плана (*bj*= *bj* – *xi*,*j*);
* шаг 5 – вычитание у *i*-го потребителя значение текущего элемента опорного плана (*ai*= *ai* – *xi*,*j*);
* шаг 6 – если у *j*-го поставщика значение равно нулю (*bj*= 0), то перейти к следующей строке (*i* = *i* + 1), в противном случае перейти к следующему столбцу (*j* = *j* + 1);
* шаг 7 – повторение шагов 3-6, пока *i* < *n*.

Важно отметить, что применение метода северо-западного угла в многокритериальных системах может быть ограничено и не всегда приведет к оптимальному решению, учитывая все критерии.

***2.1.2***Решение транспортной задачи методом северо-западного угла

Рассмотрим пример построения плана методом северо-западного угла. Исходные данные транспортной задачи представлены в таблице 2.2.

Таблица 2.2 – Исходные данные транспортной задачи

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Запас *ai* | Потребность, *bj* | | | |
| 50 | 15 | 15 | 20 |
| 20 | 1 | 2 | 3 | 4 |
| 30 | 4 | 2 | 3 | 1 |
| 40 | 1 | 2 | 1 | 2 |
| 10 | 3 | 4 | 5 | 3 |

В таблице 2.2в ячейках *ai*указаны объёмы запасов поставщиков. В ячейках *bj* указаны объёмы потребностей потребителей. Во внутренних ячейках таблицы указаны стоимости перевозок продукции.

Требуется составить план перевозок, при котором общая стоимость доставки продукции будет наименьшей.

Для решения задачи необходимо, чтобы cуммарные запасы продукции у поставщиков равнялись суммарной потребности потребителей.

Запасы поставщиков: 20 + 30 + 40 + 10 = 100 единиц продукции.

Потребность потребителей: 50 + 15 + 15 + 20 = 100 единиц продукции.

Суммарные запасы продукции у поставщиков равны суммарной потребности потребителей.

Далее, начиная с верхнего левого угла, таблица заполняется значениями пуставок.

Шаг 1. Первая ячейка первого столбца заполняется значением *min*(*a*1, *b*1) = *min*(20, 50) = 20.

Запас, оставшийся в ячейке *a*1 будет равнятся *a*1 – *min*(*a*1, *b*1) = 20 – 20 = 0.

Потребность в ячейке *b*1 будет равнятся *b*1 – *min*(*a*1, *b*1) = 50 – 20 = 30.

В таблице 2.3 показан результат выполнения первого шага алгоритма.

Таблица 2.3 – Результат выполнения первого шага алгоритма.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Запас *ai* | Потребность, *bj* | | | |
| 30 | 15 | 15 | 20 |
| 0 | 201 | 2 | 3 | 4 |
| 30 | 4 | 2 | 3 | 1 |
| 40 | 1 | 2 | 1 | 2 |
| 10 | 3 | 4 | 5 | 3 |

Шаг 2. Так как запасы поставщиков в первой строке исчерпаны, а потребности поставщиков в первом столбце полностью не удовлетворены, следующей заполняется ячейка во второй строке первого столбца.

Вторая ячейка первого столбца заполняется значением *min*(*a*2, *b*1) = *min*(30, 30) = 30.

Запас, оставшийся в ячейке *a*2 будет равнятся *a*2 – *min*(*a*2, *b*1) = 30 – 30 = 0.

Потребность в ячейке *b*1 будет равнятся *b*1 – *min*(*a*2, *b*1) = 30 – 30 = 0.

В таблице 2.4 показан результат выполнения второго шага алгоритма.

Таблица 2.4 – Результат выполнения второго шага алгоритма

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Запас *ai* | Потребность, *bj* | | | |
| 0 | 15 | 15 | 20 |
| 0 | 201 | 2 | 3 | 4 |
| 0 | 304 | 2 | 3 | 1 |
| 40 | 1 | 2 | 1 | 2 |
| 10 | 3 | 4 | 5 | 3 |

Шаг 3. Так как запасы поставщиков во второй строке исчерпаны, а потребности поставщиков в первом столбце полностью удовлетворены, следующей заполняется ячейка в третьей строке второго столбца.

Третья ячейка второго столбца заполняется значением *min*(*a*3, *b*2) = *min*(40, 15) = 15.

Запас, оставшийся в ячейке *a*3 будет равнятся *a*3 – *min*(*a*3, *b*2) = 40 – 15 = 25.

Потребность в ячейке *b*2 будет равнятся *b*2 – *min*(*a*3, *b*2) = 15 – 15 = 0.

В таблице 2.5 показан результат выполнения третьего шага алгоритма.

Таблица 2.5 – Результат выполнения третьего шага алгоритма

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Запас *ai* | Потребность, *bj* | | | |
| 0 | 0 | 15 | 20 |
| 0 | 201 | 2 | 3 | 4 |
| 0 | 304 | 2 | 3 | 1 |
| 25 | 1 | 15­2 | 1 | 2 |
| 10 | 3 | 4 | 5 | 3 |

Шаг 4. Так как запасы поставщиков во третьей строке не исчерпаны, а потребности поставщиков во втором столбце полностью удовлетворены, следующей заполняется ячейка в третьей строке третьего столбца.

Третья ячейка третьего столбца заполняется значением *min*(*a*3, *b*3) = *min*(25, 15) = 15.

Запас, оставшийся в ячейке *a*3 будет равнятся *a*3 – *min*(*a*3, *b*3) = 25 – 15 = 10.

Потребность в ячейке *b*3 будет равнятся *b*3 – *min*(*a*3, *b*3) = 15 – 15 = 0.

В таблице 2.6 показан результат выполнения четвёртого шага алгоритма.

Таблица 2.6 – Результат выполнения четвёртого шага алгоритма

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Запас *ai* | Потребность, *bj* | | | |
| 0 | 0 | 0 | 20 |
| 0 | 201 | 2 | 3 | 4 |
| 0 | 304 | 2 | 3 | 1 |
| 10 | 1 | 15­2 | 151 | 2 |
| 10 | 3 | 4 | 5 | 3 |

Шаг 5. Так как запасы поставщиков во третьей строке не исчерпаны, а потребности поставщиков в третьем столбце полностью удовлетворены, следующей заполняется ячейка в третьей строке четвёртого столбца.

Третья ячейка четвёртого столбца заполняется значением *min*(*a*3, *b*4) = *min*(10, 20) = 10.

Запас, оставшийся в ячейке *a*3 будет равнятся *a*3 – *min*(*a*3, *b*4) = 10 – 10 = 0.

Потребность в ячейке *b*4 будет равнятся *b*4 – *min*(*a*3, *b*4) = 20 – 10 = 10.

В таблице 2.7 показан результат выполнения пятого шага алгоритма.

Таблица 2.7 – Результат выполнения пятого шага алгоритма

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Запас *ai* | Потребность, *bj* | | | |
| 0 | 0 | 0 | 10 |
| 0 | 201 | 2 | 3 | 4 |
| 0 | 304 | 2 | 3 | 1 |
| 0 | 1 | 15­2 | 151 | 102 |
| 10 | 3 | 4 | 5 | 3 |

Шаг 6. Так как запасы поставщиков во третьей строке исчерпаны, а потребности поставщиков в четвёртом столбце полностью не удовлетворены, следующей заполняется ячейка в четвёртой строке четвёртого столбца.

Четвёртая ячейка четвёртого столбца заполняется значением *min*(*a*4, *b*4) = *min*(10, 10) = 0.

Запас, оставшийся в ячейке *a*4 будет равнятся *a*4 – *min*(*a*4, *b*4) = 10 – 10 = 0.

Потребность в ячейке *b*4 будет равнятся *b*4 – *min*(*a*4, *b*4) = 10 – 10 = 0.

В таблице 2.8 показан результат выполнения шестого шага алгоритма.

Таблица 2.8 – Результат выполнения шестого шага алгоритма

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Запас *ai* | Потребность, *bj* | | | |
| 0 | 0 | 0 | 0 |
| 0 | 201 | 2 | 3 | 4 |
| 0 | 304 | 2 | 3 | 1 |
| 0 | 1 | 15­2 | 151 | 102 |
| 0 | 3 | 4 | 5 | 103 |

Стоимость доставки продукции для полученного плана вычисляется по формуле , где *n* – число поставщиков, *m* ­– число потребителей, d – значение элемента матрицы перевозок, x – значение элемента из матрицы удельных стоимостей транспортировки.

Полученная стоимость доставки продукции равняется.

Таким образом транспортная задача была пошагово решена методом северо-западного угла.

**2.2 Метод динамического программирования в многокритериальных**

**системах**

В многокритериальных системах метод динамического программирования может быть модифицирован для учета нескольких критериев оптимизации одновременно. Одним из подходов является использование двумерных или трехмерных массивов для хранения оптимальных значений для каждого подзадания с учетом различных критериев.

Как правило для каждого критерия создается отдельное измерение в массиве динамического программирования. Каждая ячейка массива представляет собой оптимальное значение для соответствующего подзадания и критерия. При заполнении массива, значения обновляются с учетом всех критериев.

Одна из распространенных модификаций метода динамического программирования в многокритериальных системах – это построение множества Парето-оптимальных решений.

Множество Парето-оптимальных решений состоит из решений, которые не могут быть улучшены по одному критерию без ухудшения других. Для этого используется сравнение значений оптимальных решений в различных ячейках массива по каждому критерию.

Получив заполненный массив динамического программирования, можно проследить путь от последней ячейки к первой, чтобы получить оптимальное решение, учитывающее все критерии.

Таким образом, модифицированный метод динамического программирования позволяет находить набор Парето-оптимальных решений, предоставляя полезную информацию для принятия решений в многокритериальных системах.

Таблица 2.9 – Характерастики задачи о рюкзаке

|  |  |
| --- | --- |
| Характеристика | Описание |
| Цель | Выбрать вещи с максимальной ценностью, которые вместе не превышают стоимость услуги курьера за один час и размер рюкзака. |
| Исходные данные | Входными данными для задачи о рюкзаке являются предметы, которые необходимо поместить в рюкзак. Предмет имеет три параметра: стоимость (*cost*), размер (*size*) вещи, стоимость доставки (*tariff*). |
| Ограничения | Размер рюкзака (*max*\_*size*) является положительным вещественным числом и представляет собой максимальную вместимость рюкзака.  Размер рюкзака (*max*\_*tariff*) является положительным вещественным числом и представляет собой максимальную стоимость доставки.  Максимальная ставка в час курьера (*max\_cost*) является положительным вещественным числом и представляет собой максимально возможную ставку курьера. |
| Результат | Список предметов с максимальной суммарной стоимостью, которые в сумме не превышают стоимость услуги курьера за один час и размер рюкзака. |

Когда решается задача о многокритериальном рюкзаке, где требуется оптимизировать несколько критериев одновременно, метод динамического программирования предоставляет эффективный подход. Суть метода заключается в заполнении трехмерного массива пошагово, где каждая ячейка содержит оптимальное значение для соответствующего подзадания. Затем, проследив путь от последней ячейки к первой, можно получить оптимальное решение, учитывающее все критерии и ограничения на вместимость рюкзака.

На рисунке 2.2 показана графическая схема метода динамического программирования для задачи о рюкзаке.

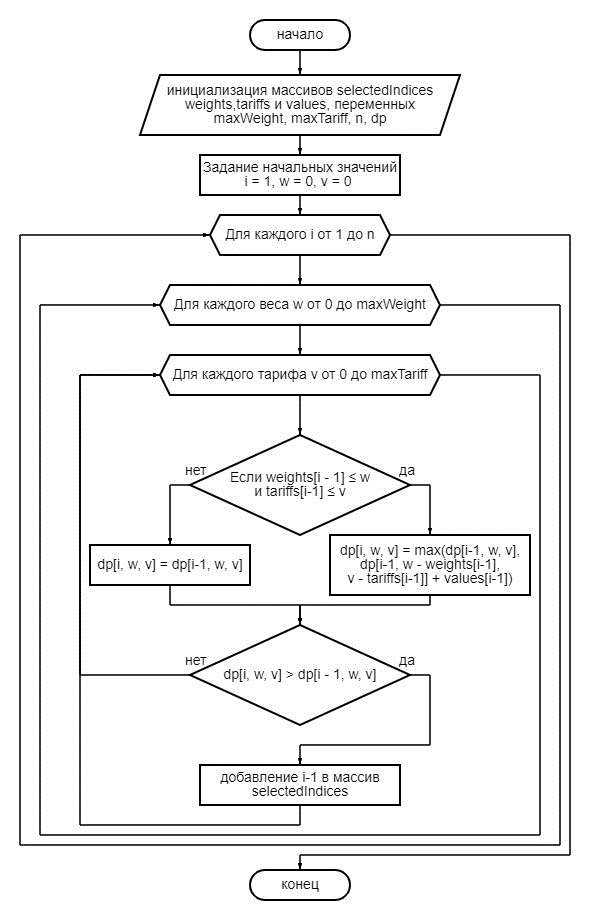


Рисунок 2.2 – Графическая схема метода динамического программирования для задачи о рюкзаке

В задаче о многокритериальном рюкзаке, когда требуется оптимизировать несколько критериев одновременно, метод динамического программирования можно применить следующим образом:

1. определить список критериев, которые требуется оптимизировать;
2. создать трехмерный массив размером [*n*+1][*W*+1][*C*+1], где *n* – количество предметов, *W* – вместимость рюкзака по весу, *C* – стоимость доставки и *K* – количество критериев;
3. инициализировать первый слой массива нулями;
4. заполнить массив значениями; для каждого предмета *i* (от 1 до *n*) и каждой вместимости *w* (от 1 до *W*), вычислить оптимальное значение для каждого критерия *k* (от 0 до *K*-1):
   1. если вес предмета *i* превышает текущую вместимость *w*, то значение в ячейке [*i*][*w*][*k*] будет равно значению в ячейке [*i* – 1][*w*][*k*];
   2. если вес предмета *i* не превышает текущую вместимость *w*, то оптимальное значение в ячейке [*i*][*w*][*k*] изменено на значение в ячейке [*i* – 1][*w*][*k*] либо на значение полученное при добавлении текущего предмета к значению в ячейке [*i* – 1][*w* – вес предмета *i*][*k*];
5. получить оптимальное решение, проследив путь от последней ячейки к первой, начиная с ячейки [*n*][*W*][*C*]; выбирать предметы, которые были использованы для достижения каждого значения, двигаясь от ячейки [*n*][*W*][*C*] к ячейке [0][0][0].

Этот подход позволяет найти набор Парето-оптимальных решений, не ухудшающихся по одному критерию без ухудшения других, и предоставляет полезную информацию для принятия решений в многокритериальных системах.

# **3 ПРОГРАММНАЯ РЕАЛИЗАЦИЯ ЗАДАЧИ**

## **3.1 Архитектура приложения**

Монолитная архитектура *WFA* (*Windows Form Application*) представляет собой структурный подход к разработке приложений, в котором весь код и функциональность приложения объединены в единый монолитный модуль или компонент. В такой архитектуре отсутствует явное разделение на модули, слои или сервисы, что может привести к некоторым негативным последствиям [4].

Плюсы монолитной архитектуры *WFA* могут включать следующие факторы:

1. простота разработки: вся логика приложения находится в одном месте, что позволяет разработчикам быстрее создавать и изменять функциональность. В Windows Forms есть много инструментов и библиотек для быстрой разработки пользовательского интерфейса;
2. простота развертывания: приложение *WFA* может быть упаковано в один исполняемый файл, что упрощает его развертывание на различных компьютерах без необходимости установки дополнительных зависимостей;
3. производительность: использование монолитной архитектуры может обеспечить хорошую производительность приложения, поскольку нет накладных расходов на коммуникацию и сетевое взаимодействие между различными компонентами;
4. простота отладки: отладка монолитного приложения может быть более простой, поскольку все компоненты находятся в одном месте. Это может сократить время поиска и устранения ошибок.

Однако стоит учитывать, что при масштабировании больших и сложных систем монолитная архитектура может стать ограничивающим фактором. Изменение и поддержка отдельных компонентов может быть затруднительным, особенно при увеличении размера приложения и сложности его функциональности.

В монолитной архитектуре *WFA* приложений все компоненты приложения находятся в одном проекте. Это может привести к следующим проблемам:

1. сложность масштабирования: при увеличении размера и сложности приложения, монолитная архитектура может стать сложной для масштабирования и поддержки. Изменение или добавление новой функциональности может требовать модификации всего монолита, что может быть сложным и рискованным процессом;
2. отсутствие модульности: в монолитной архитектуре все компоненты находятся в одном месте, что ограничивает возможности повторного использования кода и модульности. Изменение или замена отдельных компонентов может быть сложным и требовать изменений во всем приложении;
3. ограниченная гибкость: монолитная архитектура может быть менее гибкой при внедрении новых технологий или языков программирования. Внесение изменений в технологический стек приложения может потребовать модификации всего монолита, что может быть затруднительным;
4. зависимость от платформы: при использовании *Windows Forms* и монолитной архитектуры *WFA* приложение становится зависимым от платформы *Windows*. Это ограничивает его переносимость на другие операционные системы;
5. сложность отладки и тестирования: в монолитной архитектуре отладка и тестирование могут быть сложными, особенно при наличии большого количества кода и взаимосвязанных компонентов. Ошибка в одной части приложения может повлиять на всю систему.

Важно отметить, что эти минусы не всегда являются проблемами и могут зависеть от конкретного проекта и его требований. Для некоторых небольших и простых приложений монолитная архитектура *WFA* может быть предпочтительной и достаточно эффективной.

## **3.2 Описание исходных данных**

***3.2.1***Формат используемого файла

В качестве исходных данных для данного проекта выбран формат текстового файла, известный своей простотой и гибкостью. Текстовый файл представляет собой последовательность символов, обычно организованных в виде строк. Он не имеет фиксированной структуры и может содержать любую информацию, которая может быть представлена в текстовой форме.

Преимущества использования текстового файла включают:

* простота чтения и записи: текстовые файлы легко читать и записывать с помощью различных программ или скриптов на разных платформах. Текстовый формат понятен для большинства пользователей и не требует специфического программного обеспечения;
* переносимость и совместимость: текстовые файлы могут быть использованы на разных операционных системах, таких как *Windows*, *macOS* и *Linux*, без необходимости внесения значительных изменений. Они также могут быть обработаны различными программами и инструментами, что позволяет легко обмениваться данными между разными системами;
* удобство визуализации и редактирования: текстовые файлы могут быть открыты в текстовых редакторах, что позволяет легко просматривать и редактировать содержимое. Это особенно полезно при работе с небольшими или средними объемами данных, когда не требуется специализированное программное обеспечение;
* гибкость и расширяемость: текстовые файлы позволяют гибко представлять данные в различных форматах, таких как разделители (например, запятые или табуляция) или структурированный формат (например, *JSON* или *XML*). Это позволяет адаптировать файл под конкретные требования и упрощает добавление дополнительной информации или расширение данных;
* возможность работы с большими объемами данных: текстовые файлы могут обрабатывать большие объемы данных, так как они не требуют значительных ресурсов для хранения и обработки. Это делает их удобным средством для работы с большими транспортными задачами, где может быть большое количество поставщиков, потребителей и других переменных.

***3.2.2***Исходные данные транспортной задачи

Исходные данные транспортной задачи включают следующие элементы:

* поставщики (источники): это набор поставщиков или источников ресурсов, которые могут поставлять определенное количество товаров или услуг. Каждый поставщик имеет определенную мощность или ограничение поставок;
* потребители (назначения): это набор потребителей ресурсов, которые требуют определенное количество товаров или услуг. Каждый потребитель имеет свою потребность или требование поставок;
* расстояния или стоимости перевозок: Для каждой пары поставщик-потребитель определены расстояния (или стоимости) перевозок. Это может быть матрица расстояний или набор заранее определенных стоимостей перевозок.

Список значений объёмов поставок в программе представляет собой массив значений целочисленного типа.

Список значений объёмов потребностей поставок в программе представляет собой массив значений целочисленного типа.

Стоимости перевозок в программе представляют собой двумерный массив значений целочисленного типа.

***3.2.3***Исходные данные задачи о рюкзаке

Исходные данные задачи о рюкзаке включают следующие элементы:

* максимальный вес посылки: это ограничение на максимальный вес предметов, которые можно упаковать в рюкзак. Обычно это представлено числом или конкретным значением;
* стоимость ставки курьера в час: это ограничение на стоимость доставки. Обычно это представлено числом или конкретным значением;
* предметы: каждый предмет имеет свои характеристики, такие как вес, тариф доставки и стоимость. Обычно предметы представлены в виде списка или массива, где каждый элемент содержит информацию о предмете.

Максимальный весь посылки в программе представляет собой переменную целочисленного значения.

Стоимость ставки курьера в час в программе представляет собой переменную целочисленного значения.

Список предметов представляет собой коллекцию типа *List* объектов класс *Element*, ключающий в себя информацию о весе предмета, стоимости доставки, стоимости предмета в виде полей целочисленного значения.

**3.3 Структура программного комплекса**

Для разработки программного комплекса, предназначенного для решения поставленной задачи, был использован язык программирования высокого уровня *C*#. Графический интерфейс данного программного комплекса был реализован с использованием технологии *WFA*.

На рисунке 3.1 представлена структура разработанного решения.

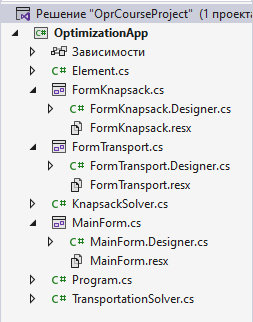


Рисунок 3.1 – Структура разработанного решения

Проект состоит из следующих классов:

– *Element*;

– *TransportationSolver*;

– *KnapsackSolver*;

– *FormKnapsack*;

– *FormTransport*;

– *MainForm*

– *Program*.

*Element* представляет собой предмет, который можно положжить в рюкзак. Он реализован в виде класс с тремя целочисленными свойствами *Weight* (вес), *Value* (стоимость) и *Tariff* (тариф курьера).

*TransportationSolver* представляет собой статический класс, который содержит статический метод *SolveTransportationProblem*, который принимает список потребностей и ресурсов и выполняет решение транспортной задачи методом северо-западного угла.

*KnapsackSolver* представляет собой статический класс, который содержит статические методы *SolveKnapsackProblem* и *GetSelectedItems*. Метод *SolveKnapsackProblem* выполняет решение задачи о рюкзаке методом динамического программирования. Метод *GetSelectedItems* осуществляет получение списка элементов в рюкзаке после решения задачи о рюкзаке.

*FormKnapsack* представляет собой класс, реализующий окно для ввода исходных данных и решения задачи о рюкзаке. В этом окне реализован интерфейс пользователя для отображения и взаимодействия с исходными данными задачи о рюкзаке, а также для для вывода результата решения задачи.

*FormTransport* представляет собой класс, реализующий окно для ввода исходных данных и решения транспортной задачи. В этом окне реализован интерфейс пользователя для отображения и взаимодействия с исходными данными транспортной задачи, а также для для вывода результата решения задачи.

*MainForm* представляет собой класс, реализующий главное окно приложения, в котором осуществляется выбор решаемой задачи. В этом окне пользователю предоставляется возможность выбрать, какую задачу он собирается решать: транспортную или о рюкзаке.

Класс *Program* представляет собой точку входа в приложение. Он содержит метод *Main*, который выполняется при запуске проекта. Метод *Main* инициализирует конфигурацию приложения и отображает главное окно приложения.

## **3.4 Функционал программного продукта**

В результате было разработано приложения *WFA* для решения задачи о рюкзаке и транспортной задачи.

Главное окно приложения представлено на рисунке 3.2.

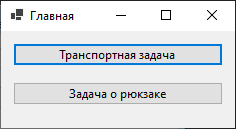


Рисунок 3.2 – Главное окно приложения

Главное окно содержит кнопки для перехода на окно транспортной задачи и задачи о рюкзаке.

При нажатии на кнопку «Транспортная задача» открывается окно для решения транспортной задачи.

При нажатии на кнопку «Задача о рюкзаке» открывается окно для решения задачи о рюкзаке.

Окно для решения транспортной задачи представлено на рисунке 3.3.

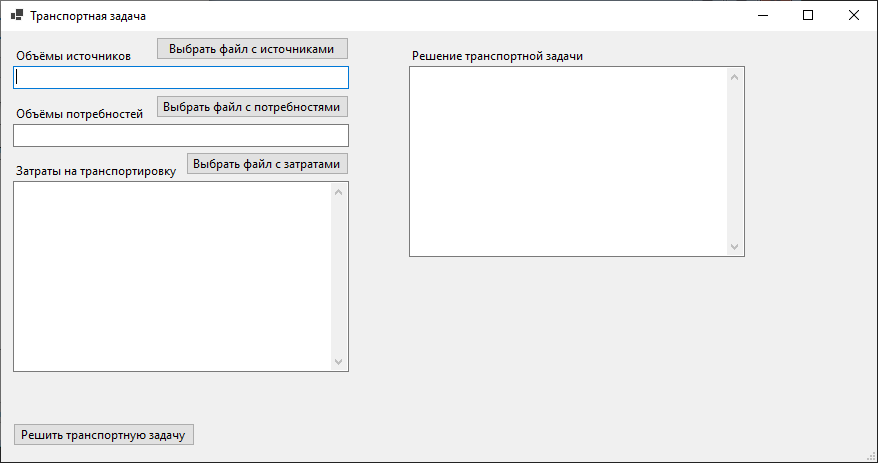


Рисунок 3.3 – Окно для решения транспортной задачи

Для решения транспортной задачи необходимо загрузить 3 файла. По нажатию на кнопку «Выбрать файл с затратами» необходимо выбрать файл с матрицей затрат.

По нажатию на кнопку «Выбрать файл с потребностями» необходимо выбрать файл с потребностями.

По нажатию на кнопку «Выбрать файл с источниками» необходимо выбрать файл с источниками.

После успешного открытия всех файлов все значения, необходимые для решения транспортной задачи, отобразятся на пользовательском интерфейсе.

По нажатию на кнопки для выбора файла открывается диалоговое окно для выбора файла. В нём пользователю предоставляется возможность выбрать файл с соответствующими данных, в зависимости от того, какая кнопка была нажата.

После открытия файлов с исходными данными транспортной задачи данные будут отображены в полях окна приложения. Результат оборажения исходных данных транспортной задачи представлен на рисунке 3.4.

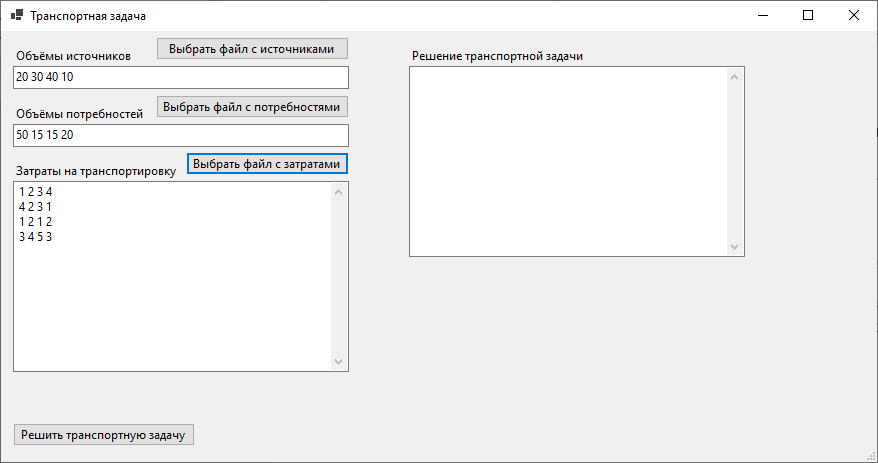


Рисунок 3.4 – Результат отображения исходных данных траспортной задачи

Далее по нажатию на кнопку «Решить транспортную задачу» будут проведены расчеты и результаты отобразяться в пользовательском интефейсе, показаном на рисунке 3.5.

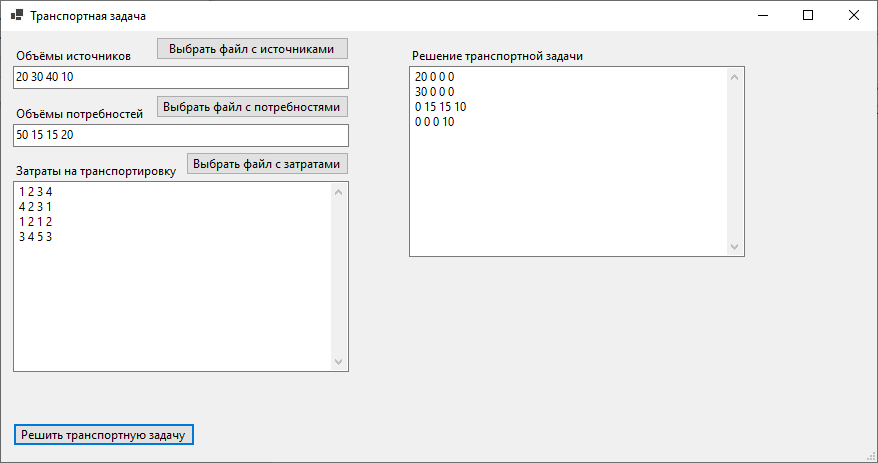


Рисунок 3.5 – Результат решения транспортной задачи

Для закрытия окна с транспортной задачей необходимо нажать крестик в правом верхнем углу главного окна программы.

Окно для решения задачи о рюкзаке показано на рисунке 3.6.

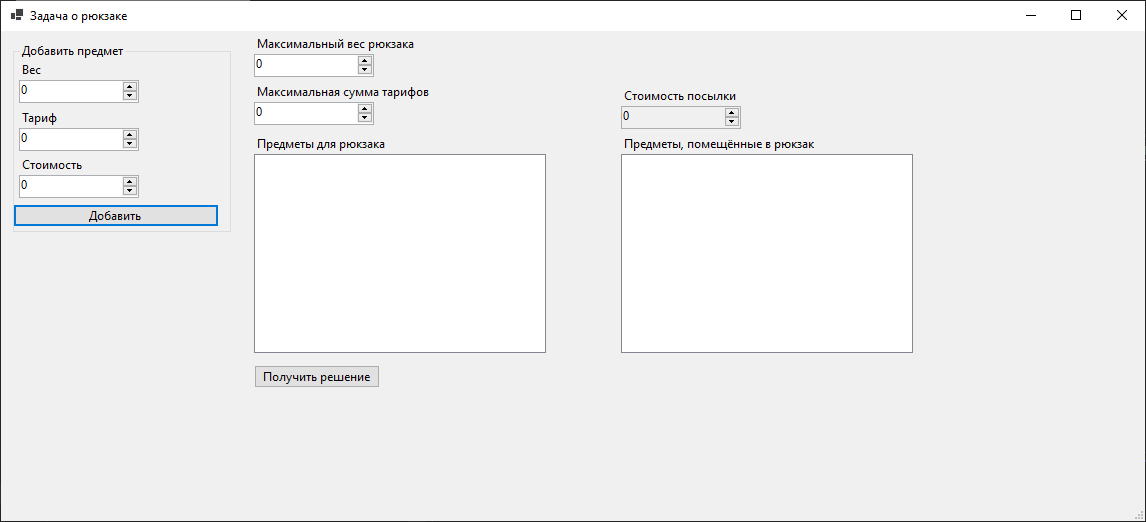


Рисунок 3.6 – Окно для решения задачи о рюкзаке

В окне для решения задачи о рюкзаке пользователю предоставляется возможность для добавления предметов, ввода веса рюкзака, суммы тарифов курьера.

На рисунке 3.7 показан результат ввода исходных данных задачи о рюкзаке.

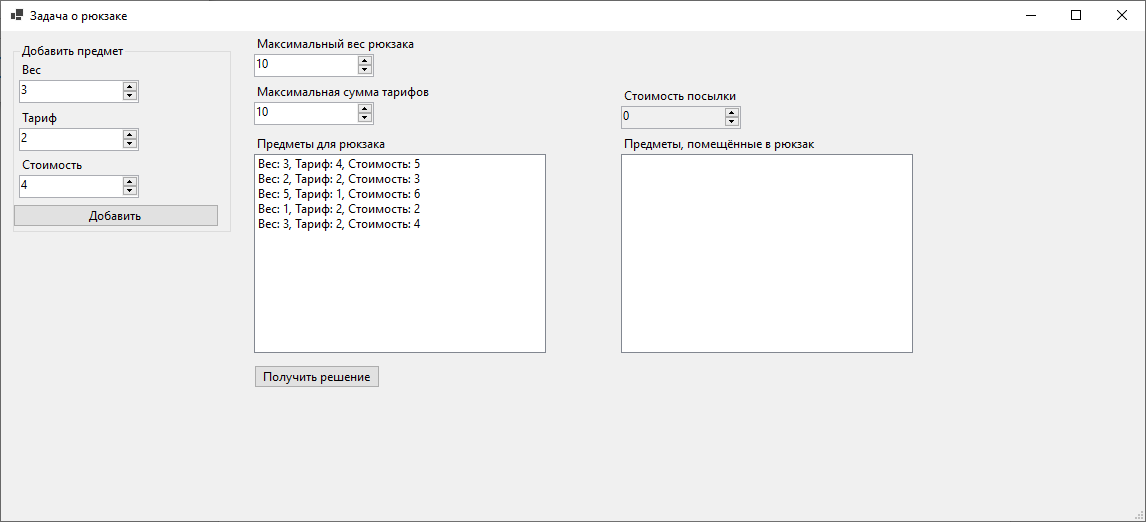


Рисунок 3.7 – Окно для решения задачи о рюкзаке

Далее по нажатию на кнопку «Получить решение» будут проведены расчеты и результаты отобразяться в пользовательском интефейсе.

Результат решения задачи о рюкзаке показан на рисунке 3.8.

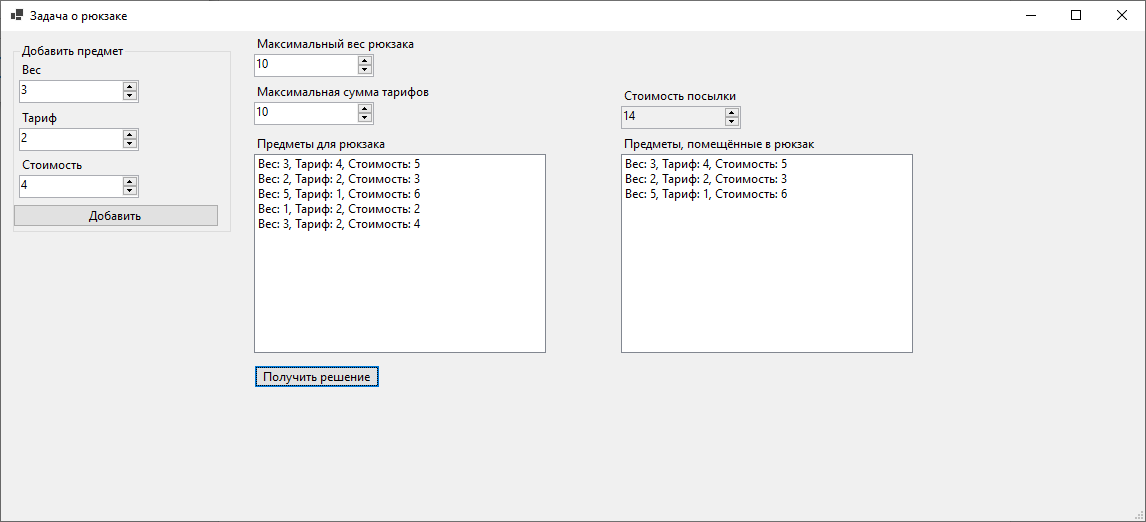


Рисунок 3.8 – Результат решения задачи о рюкзаке

На рисунке 3.8 можно увидеть, что в поле «Стоимость посылки» отобразилось значение, полученное путём сложения стоимостей предметов, помещённых в рюкзак.

# **4 ВЕРИФИКАЦИЯ ПРОГРАММНЫХ СРЕДСТВ**

## **4.1 Верификация транспортной задачи**

Для верефикации программного продукта было выполнено пошаговое программное решение транспортной задачи. Шаги

Входными данными является вектор поставщиков *am*, представляющий собой поставщиков, где *m* = 4, вектор потребителей *bn*, представляющий собой потребителей, где *n* = 4, матрица *Cnm*, представляющая собой стоимости перевозок груза от поставщика к потребителю. На рисунке 4.1 показаны исходные данные для транспортной задачи.

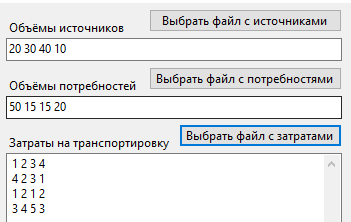


Рисунок 4.1 – Исходные данные для транспортной задачи

Данные на рисунке 4.1 соответствуют данным, представленным в таблице 2.2 пункта 2.1.2 второй главы.

Далее шаги решения будут соответствовать шагам решения задачи, представленным в пункте 2.1.2 второй главы.

Текущие поставщик и потребитель являются первыми в спиках поставщиков и потребителей.

Результат выполнения первого шага алгоритма представлен на рисунке 4.2.

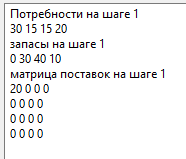


Рисунок 4.2 – Результат выполнения первого шага алгоритма

Поскольку потребности первого потребителя ещё не были полностью удовлетворены, текущий потребитель не меняется. Так как запасы первого поставщика были исчерпаны, текущий поставщик меняется на второго.

Результат выполнения второго шага алгоритма представлен на рисунке 4.3.

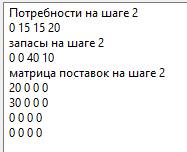


Рисунок 4.3 – Результат выполнения второго шага алгоритма

Поскольку потребности первого потребителя были полностью удовлетворены, текущий потребитель меняется на второго. Так как запасы второго поставщика были исчерпаны, текущий поставщик меняется на третьего.

Результат выполнения третьего шага алгоритма представлен на рисунке 4.4.

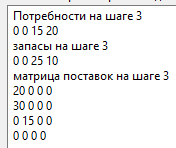


Рисунок 4.4 – Результат выполнения третьего шага алгоритма

Поскольку потребности второго потребителя были полностью удовлетворены, текущий потребитель меняется на третьего. Так как запасы третьего поставщика не были исчерпаны, текущий поставщик не меняется.

Результат выполнения четвёртого шага алгоритма представлен на рисунке 4.5.

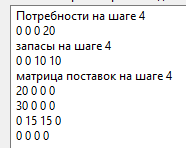


Рисунок 4.5 – Результат выполнения четвёртого шага алгоритма

Поскольку потребности третьего потребителя были полностью удовлетворены, текущий потребитель меняется на четвёртого. Так как запасы третьего поставщика не были исчерпаны, текущий поставщик не меняется.

Результат выполнения пятого шага алгоритма представлен на рисунке 4.6.

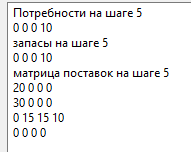


Рисунок 4.6 – Результат выполнения пятого шага алгоритма

Поскольку потребности четвёртого потребителя ещё не были полностью удовлетворены, текущий потребитель не меняется. Так как запасы третьего поставщика были исчерпаны текущий поставщик меняется на четвёртого.

Результат выполнения шестого шага алгоритма представлен на рисунке 4.7.

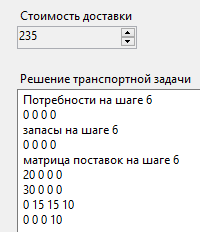


Рисунок 4.7 – Результат выполнения шестого шага алгоритма

В результате выполнения программы все запасы поставшиков были исчерпаны, а все потребности полностью удовлетворены.

Результаты выполнения программы, показанные на рисунках 4.1-4.7 соответствуют результатам вычислений, полученных в пункте 2.1.2 второй главы.

На рисунке 4.7 видно, что итоговая стоимость доставки соответствует стоимости доставки, полученной в пункте 2.1.2 второй главы.

Как видно из рисунков 4.1-4.7, алгоритм работает в соответствии с пошаговым его изложением, что свидетельствует о правильности работы программы.

## **4.2 Верификация задачи о рюкзаке**

Первым шагом метода динамическог программирования является инициализация исходных данных, как видно из рисунка 2.2.

Программная инициализация исходных данных задачи о рюкзаке показана на рисунке 4.8.

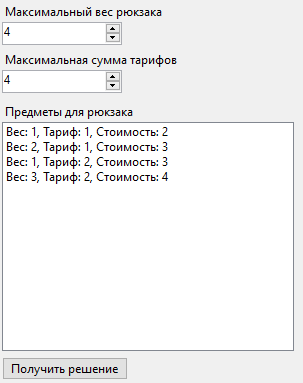


Рисунок 4.8 – Программная инициализация исходных данных задачи о рюкзаке

Следующим шагом алгоритма является выполнение цикла, в ходе которого предметы добавляются в рюкзак в зависимости от текущей максимальной массы рюкзака, установленой на данной итерации.

На рисунке 4.9 показан результат добавления предметов в рюкзак с максимальной массой, равной 1.

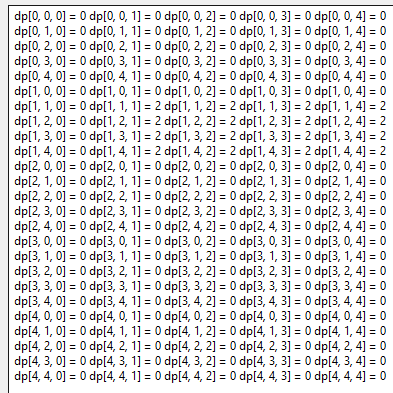


Рисунок 4.9 – Результат добавления предметов в рюкзак с максимальной

массой, равной 1

На каждой последующей итерации максимальная масса рюкзака будет увеличиваться на 1.

На следующей итерации действие повторяется для рюкзака с максимальной массой равной 2.

На рисунке 4.10 показан результат добавления предметов в рюкзак с максимальной массой, равной 2.

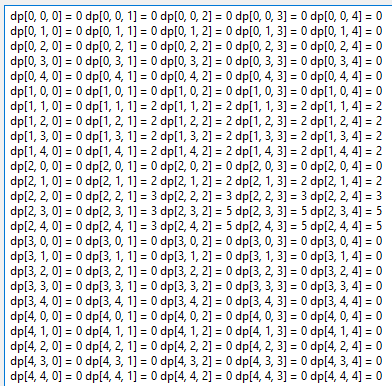


Рисунок 4.10 – Результат добавления предметов в рюкзак с максимальной массой, равной 2

На рисунке 4.11 показан результат добавления предметов в рюкзак с максимальной массой, равной 3.

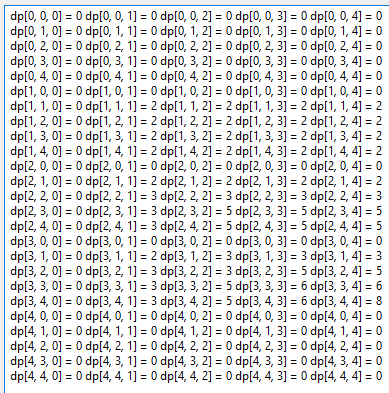


Рисунок 4.11 – Результат добавления предметов в рюкзак с максимальной массой, равной 3

На рисунке 4.12 показан результат добавления предметов в рюкзак с максимальной массой, равной 4.

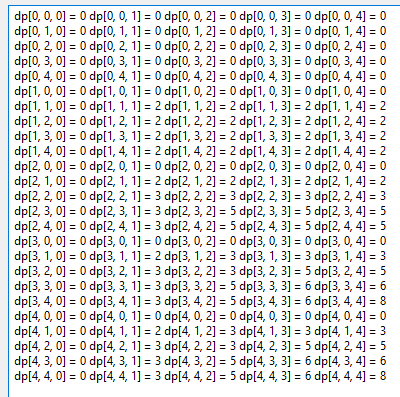


Рисунок 4.12 – Результат добавления предметов в рюкзак для максимальной массы, равной 4

На рисунке 4.13 показан результат оптимизации посылки в рюкзаке по стоимости предметов.

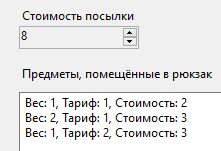


Рисунок 4.13 – Результат оптимизации посылки в рюкзаке по стоимости

предметов

Как видно на рисунках 4.8-4.13, алгоритм работает в соответствии со схемой, приведённой на рисунке 2.2, что свидетельствует о правильности его работы.

# **ЗАКЛЮЧЕНИЕ**

Разработка приложений для решения задач о рюкзаке и транспортной задачи имеет огромное значение в научно-исследовательской сфере. Эти задачи являются классическими примерами оптимизационных задач, которые находят применение в различных научных и инженерных областях. Их решение и алгоритмы, разработанные для них, имеют фундаментальное значение и способствуют развитию и прогрессу во многих областях деятельности.

Разработанное приложение предоставляет решение для задачи о рюкзаке, позволяющее эффективно выбирать оптимальное подмножество предметов, учитывая ограничения на их вес и стоимость. Это приложение имеет широкий спектр применения в различных областях, таких как управление ресурсами, финансы, логистика и снабжение. Его функциональность позволяет оптимизировать распределение ограниченных ресурсов с целью достижения максимальной ценности или прибыли. Таким образом, данное приложение является ценным инструментом, способствующим оптимизации и эффективному управлению важными бизнес-процессами.

Приложение, специально созданное для решения транспортных задач, обеспечивает оптимальное распределение ресурсов между различными источниками и потребителями, учитывая стоимость перевозок. Это приложение находит широкое применение в областях, таких как логистика, снабжение, планирование маршрутов, моделирование и оптимизация транспортных систем. Его функциональность позволяет эффективно управлять потоками материалов, товаров или информации. Таким образом, данное приложение представляет собой ценный инструмент для оптимизации и эффективного управления транспортными процессами и ресурсами.

Для реализации данного программного обеспечения был использован язык программирования *C*#. Для пользовательского интерфейса была использована платформа *WFA*.

В рамках данного курсового проекта были успешно достигнуты все поставленные цели и задачи, в результате чего было разработано программное обеспечение для решения задач о рюкзаке и транспортной задачи.

Таким образом, разработанные приложения для решения задач о рюкзаке и транспортной задачи играют важную роль в научном исследовании и предоставляют ценные инструменты для оптимизации и управления ресурсами в различных областях применения.

Курсовой проект выполнен самостоятельно, проверен в системе «Анти-плагиат». Процент оригинальности составляет 98,58%. Цитирования обозначены ссылками на публикации, указанные в «Списке использованных источников».

**Список использованных источников**

1. Kellerer, H. Knapsack Problems / H. Kellerer, U. Pferschy, D. Pisinger. – S-pringer-Verlag, 2003. – 535 с.
2. Рафгарден, Т. Совершенный алгоритм. Жадные алгоритмы и динамическое программирование / Т. Рафгарден. – СПб.: Питер, 2020. – 258 с.
3. Пахомов Б.И., C# для начинающих. – Спб.: БХВ-Петербург, 2014. – 432 с.
4. Петцольд Ч. Программирование для Microsoft Windows Froms. – М.: Русская редакция, СПб.: Питер, 2006.
5. Belton, V., Stewart T. Multiple Criteria Decision Analysis: An Integrated Approach.: Kluwer Academic Publishers, 2002. – 180 с.
6. Болотникова, О. В. Линейное программирование: транспорные и сетевые модели: учеб. пособие / О. В. Болотникова, Д. В. Тарасов, Р. В. Тарасов. – Пенза: Изд-во ПГУ, 2016. – 88 с.
7. Жадные алгоритмы. – Электрон. данные. – Режим доступа: https://algo-prog.ru/material/greedy\_simple.1. – Дата доступа: 01.12.2023.

# **ПРИЛОЖЕНИЕ А**

(обязательное)

**Листинг программы**

***Program.cs:***

namespace OprApp

{

internal static class Program

{

[STAThread]

static void Main()

{ApplicationConfiguration.Initialize();

Application.Run(new MainForm());

}

}

}

***Element.cs:***

namespace OprApp

{

public class Element

{

public int Weight { get; set; }

public int Cost { get; set; }

public int Tariff { get; set; }

public Element(int cost, int tariff, int weight)

{

Weight = weight;

Cost = cost;

Tariff = tariff;

}

public override string ToString()

{

return "Вес: " + Weight + ", Тариф: " + Tariff + ", Стоимость: " + Cost;

}

}

}

***KnapsackSolver.cs:***

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace OprApp

{

public static class KnapsackSolver

{

public static List<int[,,]> Dps = new List<int[,,]>();

static List<Element> GetSelectedItems(List<Element> elements, int[,,] dp, int[] weights, int[] volumes, int n, int w, int v)

{

List<int> selectedItems = new List<int>();

while (n > 0 && w > 0 && v > 0)

{

if (dp[n, w, v] != dp[n - 1, w, v])

{

selectedItems.Add(n - 1);

w -= weights[n - 1];

v -= volumes[n - 1];

}

n--;

}

selectedItems.Reverse();

return selectedItems.Select(i => elements[i]).ToList();

}

public static List<Element> SolveKnapsackProblem(List<Element> elements, int maxWeight, int maxTariffs)

{

int[] weights = elements.Select(e => e.Weight).ToArray();

int[] tariffs = elements.Select(e => e.Tariff).ToArray();

int[] values = elements.Select(e => e.Cost).ToArray();

int n = weights.Length;

int[,,] dp = new int[n + 1, maxWeight + 1, maxTariffs + 1];

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

{

for (int w = 0; w <= maxWeight; w++)

{

for (int v = 0; v <= maxTariffs; v++)

{

if (weights[i - 1] <= w && tariffs[i - 1] <= v)

{

dp[i, w, v] = Math.Max(dp[i - 1, w, v],

dp[i - 1, w - weights[i - 1], v - tariffs[i - 1]] + values[i - 1]);

}

else

{

dp[i, w, v] = dp[i - 1, w, v];

}

}

}

Dps.Add((int[,,])dp.Clone());

}

List<Element> selectedItems = GetSelectedItems(elements, dp, weights, tariffs, n, maxWeight, maxTariffs);

return selectedItems;

}

}

}

***TransportationSolver.cs:***

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace OprApp

{

public static class TransportationSolver

{

public static List<int[,]> PlansSteps = new List<int[,]>();

public static List<int[]> DemandsSteps = new List<int[]>();

public static List<int[]> SupplySteps = new List<int[]>();

public static int[,] Plan { get; set; }

public static int[,] Costs { get; set; }

public static int[,] SolveTransportationProblem(int[] supply, int[] demand, int[,] costs)

{

Costs = costs;

PlansSteps.Clear();

int m = supply.Length;

int n = demand.Length;

int[,] plan = new int[m, n];

int[] supplyRemaining = (int[])supply.Clone();

int[] demandRemaining = (int[])demand.Clone();

int i = 0;

int j = 0;

while (i < m && j < n)

{

int quantity = Math.Min(supplyRemaining[i], demandRemaining[j]);

plan[i, j] = quantity;

supplyRemaining[i] -= quantity;

demandRemaining[j] -= quantity;

PlansSteps.Add((int[,])plan.Clone());

DemandsSteps.Add((int[])demandRemaining.Clone());

SupplySteps.Add((int[])supplyRemaining.Clone());

if (supplyRemaining[i] == 0)

{

i++;

}

if (demandRemaining[j] == 0)

{

j++;

}

}

Plan = plan;

return plan;

}

public static int GetTransportCost()

{

int cost = 0;

for(int i = 0;i < Costs.GetLength(0);i++)

{

for(int j = 0;j < Costs.GetLength(1);j++)

{

cost += Costs[i, j] \* Plan[i,j];

}

}

return cost;

}

}

}

***MainForm.cs:***

namespace OprApp

{

public partial class MainForm : Form

{

public MainForm()

{

InitializeComponent();

}

private void buttonT\_Click(object sender, EventArgs e)

{

FormTransport formTransport = new FormTransport();

formTransport.ShowDialog();

}

private void buttonK\_Click(object sender, EventArgs e)

{

FormKnapsack formKnapsack = new FormKnapsack();

formKnapsack.ShowDialog();

}

}

}

***FormTransport.cs:***

using System.Text;

namespace OprApp

{

public partial class FormTransport : Form

{

public FormTransport()

{

InitializeComponent();

numericUpDownRes.ReadOnly = true;

}

private void buttonSolve\_Click(object sender, EventArgs e)

{

int[] sources = textBoxSources.Text.Split(new char[] { ' ' }).Select(s => int.Parse(s)).ToArray();

int[] demands = textBoxDemands.Text.Split(new char[] { ' ' }).Select(s => int.Parse(s)).ToArray();

int[][] costsArr = textBoxCosts.Text.Split(new char[] { '\n' }).Select(s => s.Split(' '))

.Select(elem => elem.Select(s => int.Parse(s)).ToArray()).ToArray();

int[,] costs = JaggedArrayToTwoDimensionalArray(costsArr);

int[,] result = TransportationSolver.SolveTransportationProblem(sources, demands, costs);

textBoxResults.Text = MatrixToString(result);

numericUpDownRes.Value = TransportationSolver.GetTransportCost();

}

static int[,] JaggedArrayToTwoDimensionalArray(int[][] jaggedArray)

{

int rows = jaggedArray.Length;

int columns = jaggedArray[0].Length;

int[,] twoDimensionalArray = new int[rows, columns];

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

{

int[] row = jaggedArray[i];

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

{

twoDimensionalArray[i, j] = row[j];

}

}

return twoDimensionalArray;

}

static string MatrixToString(int[,] matrix)

{

int rows = matrix.GetLength(0);

int columns = matrix.GetLength(1);

StringBuilder sb = new StringBuilder();

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

{

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

{

sb.Append(matrix[i, j]);

if (j < columns - 1)

{

sb.Append(" ");

}

}

if (i < rows - 1)

{

sb.AppendLine();

}

}

return sb.ToString();

}

static string ArrayToString(int[] arr)

{

return string.Concat(arr.Select(x => x.ToString() + " ").ToArray());

}

private void buttonI\_Click(object sender, EventArgs e)

{

try

{

OpenFileDialog openFileDialog = new OpenFileDialog();

openFileDialog.Filter = "Text files(\*.txt)|\*.txt|All files(\*.\*)|\*.\*";

openFileDialog.ShowDialog(this);

string filename = openFileDialog.FileName;

string fileText = File.ReadAllText(filename);

if (!IsSpaceSeparatedArray(fileText))

{

MessageBox.Show("Неверный формат файла");

}

else

{

textBoxSources.Text = fileText;

MessageBox.Show("Файл успешно прочитан");

}

}

catch

{

MessageBox.Show("Ошибка при чтении файла");

}

}

private void buttonP\_Click(object sender, EventArgs e)

{

try

{

OpenFileDialog openFileDialog = new OpenFileDialog();

openFileDialog.Filter = "Text files(\*.txt)|\*.txt|All files(\*.\*)|\*.\*";

openFileDialog.ShowDialog(this);

string filename = openFileDialog.FileName;

string fileText = File.ReadAllText(filename);

if (!IsSpaceSeparatedArray(fileText))

{

MessageBox.Show("Неверный формат файла");

}

else

{

textBoxDemands.Text = fileText;

MessageBox.Show("Файл успешно прочитан");

}

}

catch

{

MessageBox.Show("Ошибка при чтении файла");

}

}

private void buttonZ\_Click(object sender, EventArgs e)

{

try

{

OpenFileDialog openFileDialog = new OpenFileDialog();

openFileDialog.Filter = "Text files(\*.txt)|\*.txt|All files(\*.\*)|\*.\*";

openFileDialog.ShowDialog(this);

string filename = openFileDialog.FileName;

string fileText = File.ReadAllText(filename);

if (!IsTwoDimensionalArray(fileText))

{

MessageBox.Show("Неверный формат файла");

}

else

{

textBoxCosts.Text = fileText;

MessageBox.Show("Файл успешно прочитан");

}

}

catch

{

MessageBox.Show("Ошибка при чтении файла");

}

}

static bool IsSpaceSeparatedArray(string input)

{

string[] elements = input.Split(' ');

foreach (string element in elements)

{

// Проверяем, является ли каждый элемент числом

if (!int.TryParse(element, out \_))

{

MessageBox.Show(element);

return false;

}

}

return true;

}

static bool IsTwoDimensionalArray(string input)

{

string[] rows = input.Split('\n');

foreach (string row in rows)

{

string[] elements = row.Trim().Split(' ');

foreach (string element in elements)

{

// Проверяем, является ли каждый элемент числом

if (!int.TryParse(element, out \_))

{

return false;

}

}

}

return true;

}

private void buttonPlan\_Click(object sender, EventArgs e)

{

textBoxResults.Text = MatrixToString(TransportationSolver.Plan);

}

private void buttonSteps\_Click(object sender, EventArgs e)

{

textBoxResults.Text = "";

for (int i = 0; i < TransportationSolver.PlansSteps.Count(); i++)

{

textBoxResults.Text += "Потребности на шаге " + (i + 1) + "\r\n";

textBoxResults.Text += ArrayToString(TransportationSolver.DemandsSteps[i]) + "\r\n";

textBoxResults.Text += "запасы на шаге " + (i + 1) + "\r\n";

textBoxResults.Text += ArrayToString(TransportationSolver.SupplySteps[i]) + "\r\n";

textBoxResults.Text += "матрица поставок на шаге " + (i+1) + "\r\n";

textBoxResults.Text += MatrixToString(TransportationSolver.PlansSteps[i]) + "\r\n";

textBoxResults.Text += "\r\n\r\n";

}

}

}

}

***FormKnapsack.cs:***

using System.Data;

namespace OprApp

{

public partial class FormKnapsack : Form

{

List<Element> elements;

public FormKnapsack()

{

InitializeComponent();

elements = new List<Element>();

numericUpDown6.ReadOnly = true;

}

private void buttonAddElem\_Click(object sender, EventArgs e)

{

Element element = new Element((int)numericUpDownC.Value, (int)numericUpDownT.Value, (int)numericUpDownW.Value);

elements.Add(element);

listBoxElements.Items.Add(element.ToString());

}

private void buttonSolve\_Click(object sender, EventArgs e)

{

List<Element> resultElements = KnapsackSolver.SolveKnapsackProblem(elements, (int)numericUpDown5.Value, (int)numericUpDown4.Value);

numericUpDown6.Value = resultElements.Select(re => re.Cost).Sum();

textBoxResults.Text = "";

foreach (var element in resultElements)

{

textBoxResults.Text += element.ToString() + "\r\n";

}

}

private void buttonSteps\_Click(object sender, EventArgs e)

{

textBoxResults.Text = "";

for (int i = 0; i < KnapsackSolver.Dps.Count(); i++)

{

textBoxResults.Text += dpToString(KnapsackSolver.Dps[i]) + "\r\n";

}

}

private static string dpToString(int[,,] dp)

{

string dpStr = "";

for (int i = 0; i < dp.GetLength(0); i++)

{

for (int w = 0; w < dp.GetLength(1); w++)

{

for (int v = 0; v < dp.GetLength(2); v++)

{

dpStr += $"dp[{i}, {w}, {v}] = {dp[i, w, v]} ";

if (v % 5 == 4)

{

dpStr += "\r\n";

}

}

}

}

return dpStr;

}

}

}

# **ПРИЛОЖЕНИЕ Б**

(обязательное)

**Схемы алгоритмов**