Министерство науки и высшего образования Российской Федерации
ФГАОУ ВО «Волгоградский государственный университет»
Институт Математики и информационных технологий
Кафедра компьютерных наук и экспериментальной математики

Научно-исследовательская работа
Задача полного рюкзака и его применение

Выполнил(а):
студент гр. МОСб-221
Дживанян Давид Арташесович

Проверил:
Старший преподаватель
Чебаненко Никита Алексеевич

Волгоград, 2024

Оглавление

Введение

Актуальность темы и её обоснование.

В жизни современного человека, где развитие технологий и объем информации растут с огромной скоростью, задача полного рюкзака становится всё более актуальной. Задача полного рюкзака сводится к выбору из заданных предметов такой набор, который можно поместить в рюкзак с ограниченной вместимостью, а суммарная стоимость была максимальна. Эта задача имеет множество практических применений, что делает её изучение актуальным и важным для различных отраслей экономики и науки. Таких как: распределение ресурсов, логистика, планирование производства, шифрование и многие другие.

Цель работы.

Целью данной работы является исследование задачи полного рюкзака и анализ её применения в различных сферах деятельности. Для достижения этой цели необходимо решить следующие основные задачи:

1. Изучить теоретические основы задачи полного рюкзака, математическую формулировку и методы решения.
2. Рассмотреть практические примеры применения задачи полного рюкзака в логистике, планировании производства и других областях.
3. Проанализировать эффективность различных методов решения задачи полного рюкзака на конкретных примерах.

Практическая значимость работы.

Результаты данного исследования могут быть полезны для специалистов в области логистики, планирования производства, распределения ресурсов и других областей, где требуется оптимизация процессов. Они могут помочь в

разработке более эффективных алгоритмов и методов решения задачи полного рюкзака, что приведёт к улучшению качества принимаемых решений и повышению эффективности работы предприятий.

Глава 1. Общие сведения

1.1 Постановка задачи в общем виде

Пусть $M_0, M_1, ..., M_{n-1}$ - n конечных линейно упорядоченных множеств, G - совокупность ограничений (условий), ставящих в соответствие векторам вида

$$v = (v_0, v_1, ..., v_k)^T \quad (v_j \in \notin M_j;\ j = 0, 1, ... k;\ k<=n-1),$$

булево значение G (v) = истина или ложь.

Векторы $v = (v_0, v_1, ..., v_k)^T$, для которых G (v) = истина, назовем частичными решениями.

Пусть, далее, существует конкретное правило P , в соответствии с которым некоторые из частичных решений могут объявляться полными решениями.

Тогда возможна постановка следующих поисковых задач:

1. Найти все полные решения или установить отсутствие таковых.
2. Найти хотя бы одно полное решение или установить его отсутствие.

Часто задание правила P на частичных решениях имеет вид:

$$P(v_0, v_1, ..., v_k) = \begin{cases} полное\ решение, & при\ k = n-1 \\ неполное\ решение, & при\ k < n-1 \end{cases}$$

Общий метод решения таких задач состоит в последовательном покомпонентном наращивании вектора v слева направо, начиная с v0, и последующих испытаниях его ограничениями G и правилом P.

В итоге постановка задачи сводится к тому, что:
Имеется:

1. Набор векторов o1, o2, …, oN
2. Набор весов w1, w2, …, wN
3. Набор стоимостей s1, s2, …, sN

Необходимо упаковать рюкзак объемом W, чтобы стоимость его S была максимальной.

1.2 Варианты решения

Есть несколько основных способов решения задачи полного рюкзака:

1. Метод полного перебора.
2. Метод ветвей и границ.
3. Динамическое программирование.
4. Жадный алгоритм.

Метод полного перебора.

Это один из основных методов решения задачи о рюкзаке. Он заключается в последовательном рассмотрении всех возможных вариантов заполнения рюкзака предметами и выборе среди них оптимального по заданному критерию.

Алгоритм метода полного перебора.

1. Создать массив A[1, 2, …, n], где каждый элемент массива будет содержать информацию о весе и стоимости предмета.
2. Составить все возможные комбинации предметов. Для каждого предмета принять решение: взять его в рюкзак или не брать. Поэтому

общее количество возможных комбинаций равно 2 в степени n, где n - количество предметов.

3. Для каждой комбинации предметов проверить, не превышает ли суммарный вес предметов в рюкзаке емкость рюкзака.
4. Если вес комбинации предметов не превышает вместимость рюкзака, то вычислить сумму цен предметов в этой комбинации.
5. После перебора всех возможных комбинаций, выбирать комбинацию с максимальной суммой цен, которая не превышает вместимость рюкзака.
6. Вывести эту комбинацию как оптимальное решение задачи.

Метод ветвей и границ.

Это эффективный алгоритм решения задачи полного рюкзака, который позволяет найти приближенно оптимальное решение с меньшим количеством перебора комбинаций, чем метод полного перебора.

Алгоритм метода ветвей и границ.

1. Начинаем с пустого рюкзака и устанавливаем нижнюю границу равной 0.
2. Создаем узел дерева решений, представляющий оставшиеся предметы и текущую загрузку рюкзака.
3. Разделяем узел на два подузла:

   В первом подузле добавляем следующий предмет в рюкзак и пересчитываем общий вес и стоимость рюкзака.

   Во втором подузле исключаем предмет из рассмотрения и переходим к следующему шагу.
4. Проверяем, превысила ли текущая загрузка рюкзака вместимость. Если да, то этот узел не ведет к оптимальному решению и отбрасываем его.

5. Вычисляем оценку верхней границы для каждого узла, используя оставшиеся предметы и их ценность в соответствии с оставшимся местом в рюкзаке.
6. Если полученная оценка верхней границы узла меньше текущего лучшего решения, отбрасываем данный узел.
7. Повторяем шаги 3-7 для оставшихся узлов дерева решений, пока не пройдем все возможные комбинации.
8. Выбираем комбинацию с максимальной ценностью как оптимальное решение задачи.

Динамическое программирование.

Это алгоритмический подход к решению задач оптимизации, который разбивает большую задачу на более мелкие подзадачи, решая их последовательно и запоминая результаты для повторного использования.

Алгоритм метода динамического программирования.

1. Создаем двумерный массив dp размерностью (n+1) на (W+1), где n - количество предметов, W - вместимость рюкзака.
2. Инициализируем первую строку и первый столбец массива dp нулями, так как для нулевого предмета или нулевой емкости рюкзака цена всегда равна нулю.
3. Для каждого предмета i от 1 до n и для каждой вместимости рюкзака w от 1 до W выполняем следующие действия:

   Если вес предмета i меньше или равен w, то dp[i][w] равно максимуму из dp[i-1][w] (не включаем предмет) и dp[i-1][w-weight[i]] + cost[i] (включаем предмет).

   Если вес предмета i больше w, то dp[i][w] равно dp[i-1][w] (не включаем предмет).

4. После завершения процесса заполнения массива dp, находим максимальную цену, которую можно получить, и следим за тем, какие предметы были включены для достижения этой максимальной цены.
5. Восстанавливаем оптимальное решение, начиная с dp[n][W] и двигаясь обратно по массиву dp, чтобы определить, какие предметы были включены.

Жадный алгоритм.

Это метод решения задачи, при котором на каждом шаге выбирается локально оптимальное решение, с тем чтобы получить глобально оптимальное решение задачи.

Алгоритм метода жадного алгоритма.

1. Вычисляем отношение цены к весу для каждого предмета.
2. Сортируем предметы по убыванию этого отношения.
3. Идем по предметам в отсортированном порядке и пытаемся включить их в рюкзак в порядке убывания отношения цены к весу.
4. Вывести эту комбинацию как оптимальное решение задачи.

1.3 Другие вариации задачи.

У задачи полного рюкзака есть множество вариаций с различными дополнительными условиями, которые значительно расширяют область применения задач.

Некоторые вариации задачи полного рюкзака:

Ограниченный рюкзак.

Вид классической задачи, где любой предмет может быть взят несколько раз.

Неограниченный рюкзак.

Вид ограниченного рюкзака, где любой предмет может быть выбран неограниченное количество раз.

Непрерывный рюкзак.

Вид классической задачи, в котором возможно брать любую дробную часть от предмета. Удельная стоимость предмета при этом сохраняется.

Задача о суммах подмножеств.

Упрощённый вид классической задачи, в которой стоимость предмета совпадает с его весом.

Задача о размене.

Часто задачу формулируют как: дать сдачу наименьшим количеством монет.

Имеются N бесконечных типов предметов. Нужно наполнить рюкзак предметами с фиксированным весом W.

Задача об упаковке.

Есть N рюкзаков вместимости W и столько же предметов. Нужно распределить все предметы, задействовав как можно меньше рюкзаков.

Задача о назначении.

Самый общий вид задачи. Каждый предмет имеет разные характеристики в зависимости от рюкзака. Есть N предметов и M рюкзаков, где $M \leqslant N$. У каждого рюкзака есть своя вместимость Wi, у j предмета pij стоимость и вес, при помещении его в i рюкзак, равны pij и wij соответственно.

1.4 Области применения

Задача о полном рюкзаке применяется во многих областях, где необходимо оптимизировать распределение ресурсов или выбор наилучшего набора предметов при ограниченной вместимости. Зачастую области применения задачи полного рюкзака включают в себя такие аспекты как:

1. Логистика. Оптимальное распределение груза в транспортных средствах или контейнерах с учетом веса и стоимости предметов.
2. Финансы: Выбор инвестиционного портфеля или оптимизация расходов при ограниченных финансовых ресурсах.
3. Производство: Распределение и выбор сырья, материалов или компонентов для производства товаров с целью максимизации прибыли.
4. Распределение ресурсов: Оптимизация использования ограниченных ресурсов, таких как энергия, время или материалы.
5. Телекоммуникации: Выбор оптимальных комбинаций оборудования или услуг для максимизации производительности или минимизации затрат.

6. Биоинформатика: Анализ данных о геноме, для выбора наиболее важных генов или хромосом при ограниченных ресурсах.
7. Стратегическое планирование: Выбор оптимальных действий или решений при наличии ограничений и целевых критериев.

Глава 2. Техническая часть

2.1 Инструментарий

Для решения поставленной задачи я использую язык программирования высокого уровня JavaScript, язык гипертекстовой разметки HTML5, каскадную таблицу стилей CSS3. Эти 3 инструмента позволят не только эффективно решить задачу полного рюкзака, а также наглядно показать результат работы. Также я использую среду разработки PhpStorm, которая предоставляет все инструменты для комфортной работы с используемыми технологиями.

2.1.1 Язык программирования высокого уровня JavaScript

JavaScript — это высокоуровневый язык программирования, который чаще всего используется для создания интерактивных веб-приложений и сайтов. Он работает на стороне клиента в браузере и позволяет создавать динамические элементы.

Основные особенности JavaScript:

1. Интерпретируемый язык. Код выполняется браузером без предварительной компиляции. Это значительно упрощает разработку.
2. Объектно-ориентированный язык. В JavaScript есть поддержка объектов, классов, наследования и других концепций объектно-ориентированного программирования.
3. Динамическая типизация. Переменные не имеют фиксированного типа данных, что значительно упрощает написание кода.
4. Асинхронное выполнение. JavaScript является однопоточным языком, но он поддерживает асинхронное выполнение задач через функции обратного вызова или промисов.

JavaScript используется вместе с HTML и CSS для создания веб-страниц. JavaScript имеет множество библиотек и фреймворков, которые упрощают

разработку и расширяют возможности языка. Но в рамках реализации данной задачи я буду использовать стандартную современную версию языка без сторонних библиотек.

2.1.2 Язык гипертекстовой разметки HTML5

HTML (HyperText Markup Language) — это язык разметки, который используется для создания структуры и содержания веб-страниц. С помощью HTML можно определить заголовки, абзацы, списки, таблицы, ссылки и другие элементы на странице.

2.1.3 Каскадная таблица стилей CSS3

CSS (Cascading Style Sheets) — это язык стилей, который используется для описания внешнего вида веб-страниц. С помощью CSS можно задать цвета, шрифты, отступы, поля, размеры и другие параметры элементов на странице.

2.1.4 Система контроля версий Git

Git — это распределённая система контроля версий, которая позволяет разработчикам отслеживать изменения в исходном коде проекта и работать над ним совместно. Git используется для управления версиями программного обеспечения и других текстовых файлов. Git является одним из самых популярных инструментов для контроля версий в мире разработки программного обеспечения. Он используется многими крупными компаниями и проектами с открытым исходным кодом.

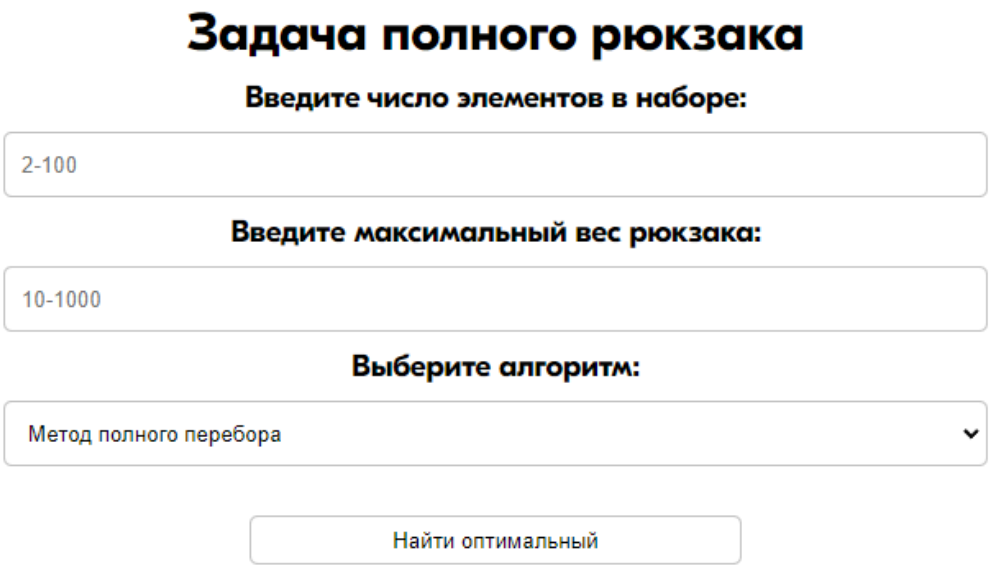
Так же я буду пользоваться сервисом GitHub, который предоставляет облачное хранилище и возможность выгрузить проект в сеть на их домен.

2.2 Общий подход реализации

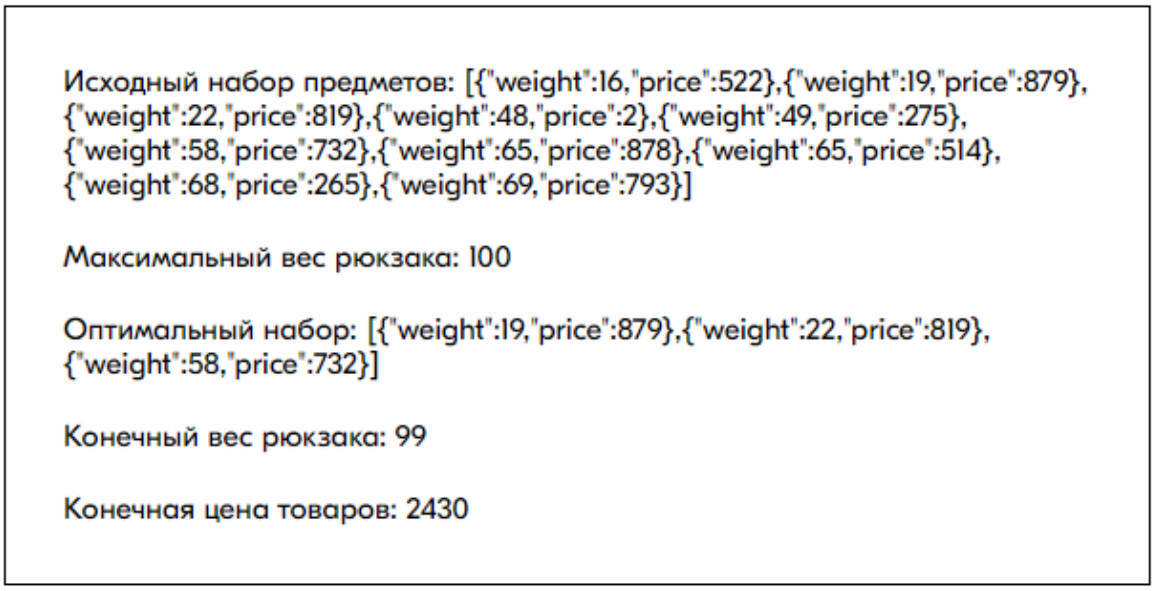
Для наглядной демонстрации методов решения задачи полного рюкзака и дальнейшего сравнения алгоритмов я создал страницу, на которой можно выбрать количество элементов N, максимальный вес рюкзака W и алгоритм, с помощью которого будет находиться оптимальное решение.

Программа будет показывать исходный набор элементов, конечный набор, который в итоге попал в рюкзак.

Скриншот 1 — Поле ввода входных данных.



Скриншот 2 — Пример вывода результата.



2.3 Метод полного перебора

По заданному выше алгоритму я написал функцию goingThrough (перебор), которая принимает в себя 2 параметра: array (массив) и capacity (int).
Внутри функции рекурсивная вспомогательная функция getAllSubarrays, которая рекурсивно перебирает все возможные подмассивы исходного массива.

Функция удаляет один элемент и вызывает сама себя дважды с удалённым элементом и без него. Как только в функция приходит пустой массив, она сравнивает результирующий массив с текущим. Если текущий помещается в рюкзак и его цена больше результирующего, то она встает на место результирующего массива.

Листинг 1 — Реализация функции полного перебора.

```
function goingThrough(arr, capacity) : any[]  {
  let result : any[]  = [];
  1+ usages
  function getAllSubarrays(arr, currentSubarray : any[]  = []) : void  {
    if (arr.length === 0) {
      if (backpackWeight(currentSubarray) <= capacity && backpackPrice(currentSubarray) > backpackPrice(result)) {
        result = [...currentSubarray];
      }

      return;
    }

    const firstElement = arr[0];
    const restOfArray = arr.slice(1);

    // Создаем новый подмассив без первого элемента
    getAllSubarrays(restOfArray, currentSubarray);

    // Создаем новый подмассив, включающий первый элемент
    getAllSubarrays(restOfArray,  currentSubarray: [...currentSubarray, firstElement]);
  }
  getAllSubarrays(arr);

  return result;
}
```

Таким образом функция перебирает все подмассивы и находит самый оптимальный.

Сложность алгоритма.

В каждом рекурсивном вызове мы имеем два варианта:

1. Включить текущий элемент в подмассив.
2. Исключить текущий элемент из подмассива.

Это означает, что для каждого элемента в массиве, у нас есть 2 возможных решения (включить или исключить). Таким образом, для массива длины n, мы будем иметь 2^n возможных подмассивов.

Каждый рекурсивный вызов выполняет постоянное число операций (создание нового массива, вызов функции), поэтому общая сложность алгоритма будет O(2^n).

Это довольно высокая сложность, и она может быть проблемой для больших массивов.

2.4 Метод ветвей и границ

По описанному выше алгоритму я написал функцию branch, которая принимает 2 параметра array (массив) - массив всех предметов и capacity (int) максимальный вес рюкзака.

Для удобства из массива с предметами я создаю 2 массива: 1 массив с весом каждого предмета, 2 массив с ценностями каждого предмета.

Я создал функцию dfs, которая принимает 3 параметра. Вес текущего элемента, его стоимость и текущий массив с элементами, которые в рюкзаке.

Внутри функции используется рекурсивный Deep First Search (DFS) подход для построения дерева решений. Каждый раз, когда мы исследуем новый узел, мы вычисляем верхнюю границу для оставшихся предметов с помощью функции calcUpperBound. Если верхняя граница меньше, чем текущее лучшее решение, мы можем отбросить этот узел и вернуться.

Листинг 2 — Реализация метода ветвей и границ.

```
function tree(array, capacity) : any[]  {
  const n = array.length;
  let bestValue : number  = 0;
  let bestSolution : any[]  = [];
  let weights = array.map((e) => e.weight);
  let values = array.map((e) : number  => e.price);

  1+ usages
  function dfs(currentWeight, currentValue, currentSolution) : void  {
    // Если текущий вес превышает максимальную грузоподъемность, возвращаемся
    if (currentWeight > capacity) {
      return;
    }

    // Обновляем лучшее решение, если текущее значение больше
    if (currentValue > bestValue) {
      bestValue = currentValue;
      bestSolution = currentSolution.slice();
    }

    // Пробуем добавить каждый оставшийся предмет
    for (let i : number  = 0; i < n; i++) {

      if (!currentSolution.includes(i)) {

        // Добавляем предмет
        currentSolution.push(i);

        dfs( currentWeight: currentWeight + weights[i],  currentValue: currentValue + values[i], currentSolution);
        currentSolution.pop(); // Откатываемся

        // Если текущее значение + оценка верхней границы меньше, чем лучшее решение, возвращаемся
        if (currentValue + calcUpperBound(currentWeight, i, values, weights) < bestValue) {
          return;
        }
      }
    }
  }
```

Таким образом в конце мы возвращаем лучшее найденное решение.

Сложность алгоритма.

Сложность алгоритма ветвей и границ для решения задачи полного рюкзака зависит от нескольких факторов:

1. Количество предметов (n)
2. Вместимость рюкзака (capacity)
3. Эффективность отбрасывания неперспективных ветвей

В худшем случае, когда алгоритм не может отбросить ни одну не перспективную ветвь, временная сложность будет экспоненциальной, то есть O(2^n).

Однако, благодаря механизму отбрасывания неперспективных ветвей, основанному на вычислении верхней границы, сложность алгоритма, как правило, ниже. Точную сложность сложно оценить, так как она сильно зависит от структуры данных и характеристик конкретной задачи.

Оценочно, сложность алгоритма ветвей и границ для решения задачи полного рюкзака можно представить как O(n * 2^n), где n - количество предметов. Это связано с тем, что для каждого предмета мы рассматриваем два варианта (включить или не включить) и выполняем некоторые дополнительные операции, такие как вычисление верхней границы.

Однако, на практике, благодаря эффективному отбрасыванию неперспективных ветвей, сложность может быть значительно ниже, особенно для задач с небольшим количеством предметов или сравнительно небольшой емкостью рюкзака.

2.5 Динамическое программирование

По заданному выше алгоритму я написал функцию dynamic, которая считает оптимальный набор рюкзака по алгоритму динамического программирования. Функция принимает в себя массив и максимальный вес.
строит матрицу N + 1 на W + 1 и заполняем её нулями.

Затем для каждого предмета i от 1 до n и для каждой вместимости рюкзака w от 1 до W выполняет следующие действия:
Если вес предмета i меньше или равен w, то dp[i][w] равно максимуму из dp[i-1][w] (не включаем предмет) и dp[i-1][w-weight[i]] + cost[i] (включаем предмет).
Если вес предмета i больше w, то dp[i][w] равно dp[i-1][w] (не включаем предмет).

После завершения процесса заполнения массива dp, находим максимальную цену, которую можно получить, и следим за тем, какие предметы были включены для достижения этой максимальной цены.

Восстанавливаем оптимальное решение, начиная с dp[n][W] и двигаясь обратно по массиву dp, чтобы определить, какие предметы были включены.

Листинг 3 — Реализация метода динамического программирования.

```
function dynamic(array, capacity) : any[]  {
  const dp : (...)[]  = new Array( arrayLength: array.length + 1).fill( value: 0).map(() : any[]  => new Array( arrayLength: capacity + 1).fill( value: 0));
  for (let i : number  = 0; i <= array.length; i++) {
    for (let j : number  = 0; j <= capacity; j++) {
      dp[i][j] = 0
    }
  }

  for (let i : number  = 1; i <= array.length; i++) {
    for (let j : number  = 1; j <= capacity; j++) {
      if (array[i - 1].weight <= j) {
        dp[i][j] = Math.max(
          dp[i - 1][j],
          dp[i - 1][j - array[i - 1].weight] + array[i - 1].price
        );
      } else {
        dp[i][j] = dp[i - 1][j];
      }
    }
  }

  let i = array.length, j = capacity;
  const result : any[]  = [];
  while (i > 0 && j > 0) {
    if (dp[i][j] !== dp[i - 1][j]) {
      result.push(array[i - 1]);
      j -= array[i - 1].weight;
    }
    i--;
  }

  return result;
```

Таким образом алгоритм ищет оптимальный набор элементов.

Сложность алгоритма.

Создание двумерного массива dp размера (items.length + 1) x (capacity + 1): O(N*W), где n - количество предметов, c - вместимость рюкзака.

Заполнение таблицы dp: O(N*W), поскольку мы проходим по всем ячейкам таблицы.

Поиск максимального значения в последней строке таблицы: O(c).

Восстановление выбранных предметов: O(n), поскольку мы проходим по всем предметам в обратном порядке.

Таким образом, общая временная сложность алгоритма: O(N*W).

Это намного быстрее чем в представленных выше алгоритмах.

2.6 Жадный алгоритм

Это единственный метод решения из всех вышеперечисленных, который получает не оптимальный ответ, а приближенный. Но это компенсируется простотой и быстротой выполнения функции.

Я написал функцию, которая принимает в себя массив искомых элементов и максимальный вес рюкзака.
Функция считает удельную стоимость товара (цена / вес) и сортирует его по убыванию. Затем по порядку добавляет в результирующий массив.

Листинг 5 — Реализация метода жадного алгоритма.

```
function greedy(array, capacity) : any[]  {
  const result : any[]  = [];

  const items = array.map((item, index : number ) : {...}  => ({
    weight: item.weight,
    price: item.price,
    ratio: item.price /item.weight,
  })).sort((a, b) => b.ratio - a.ratio);

  let totalWeight : number  = 0;

  for (const item : any  of items) {
    if (totalWeight + item.weight <= capacity) {
      totalWeight += item.weight;
      result.push({weight: item.weight, price: item.price})
    } else {
      break;
    }
  }

  return result;
}
```

Таким образом мы находим приближенный к оптимальному результат.

Сложность алгоритма.

Сортировка предметов по отношению ценности к весу эта операция выполняется с помощью стандартной функции сортировки, которая имеет среднюю сложность O(n*log (n)).

Перебор предметов в отсортированном порядке и их помещение в рюкзак: Эта часть алгоритма выполняется за линейное время, т.е. O(n), так как мы перебираем все предметы один раз.

Таким образом общая сложность алгоритма O(n*log (n)).

Глава 3. Применение задачи полного рюкзака

Задача о рюкзаке, также известная как задача о загрузке или задача полного рюкзака, является одной из классических задач комбинаторной оптимизации. Она имеет долгую историю и множество практических применений в различных областях, таких как логистика, планирование производства, распределение ресурсов и другие.

Задача о рюкзаке представляет собой одну из первых задач, для которых были разработаны точные методы решения. Это делает её важным этапом в развитии теории оптимизации и методов решения сложных задач.

В целом, задача о рюкзаке является примером того, как абстрактная математическая задача может иметь множество практических приложений и оказывать значительное влияние на развитие различных областей человеческой деятельности.

Важность задачи о рюкзаке заключается в том, что она помогает оптимизировать процессы и повысить эффективность работы предприятий в различных отраслях экономики и науки. Решение этой задачи позволяет определить оптимальный набор предметов, которые можно поместить в рюкзак с ограниченной вместимостью, при этом максимизируя общую ценность или полезность этих предметов.

3.1 Задача полного рюкзака в логистике.

Задача полного рюкзака широко применяется в логистике, в частности, при решении следующих задач:

1. Загрузка грузового транспорта.

При перевозке различных товаров и грузов необходимо эффективно использовать ограниченное пространство в грузовом автомобиле, самолете, корабле или контейнере. Задача полного рюкзака помогает определить, какие грузы необходимо загрузить, чтобы максимизировать общую стоимость перевозимого груза при ограничении по весу или объему.

2. Распределение складских площадей.

Компании, имеющие ограниченные складские помещения, сталкиваются с необходимостью эффективно размещать различные товары. Задача полного рюкзака может использоваться для определения оптимального расположения товаров на складе, учитывая их стоимость, объем и ограничения по доступному пространству.

3. Планирование поставок.

При организации цепочек поставок логистические компании должны решать, какие товары и в каких количествах следует перевозить, чтобы максимизировать прибыль. Задача полного рюкзака помогает определить, какие грузы следует включить в рейсы, ограниченные вместимостью транспортных средств.

4. Комплектование заказов.

В интернет-торговле и складской логистике необходимо эффективно собирать заказы, состоящие из множества различных товаров. Задача полного рюкзака может применяться для оптимального размещения товаров в тару (коробки, контейнеры) с учетом ограничений по весу, объему и стоимости.

5. Управление запасами.

Логистические компании должны определять, какие товары следует хранить на складе, чтобы максимизировать прибыль при ограничениях по складским площадям и бюджету. Задача полного рюкзака может использоваться для оптимизации ассортимента и объемов хранимых товаров.

3.2 Задача полного рюкзака в планирование производства

Также задача полного рюкзака широко используется в планировании производства.

В особенности при решении таких задачах как:

1. Распределение производственных ресурсов.

Если на производстве ограниченные производственные мощности, такие как станки, оборудование, площади, могут быть представлены как "рюкзак" с определенной вместимостью.

Или если на производстве есть различные виды продукции, которые можно производить, выступают в роли "предметов", которые нужно "упаковать" в этот "рюкзак" с учетом их требований к ресурсам и приоритетов (прибыльности, спроса, стратегических целей).

Также Задача полного рюкзака помогает определить оптимальный набор продуктов, которые следует производить, чтобы максимизировать общую выручку или прибыль при ограничениях по ресурсам.

2. Планирование производственных мощностей.

При расширении или модернизации производства необходимо определить, какое оборудование, линии и технологии следует установить.

Ещё задача полного рюкзака может использоваться для оптимизации инвестиций в новое оборудование, учитывая его производительность, стоимость, требования к площадям и другим ресурсам.

Это помогает спланировать необходимые производственные мощности, чтобы удовлетворить прогнозируемый спрос на продукцию

3. Планирование ассортимента продукции.

Ограниченные производственные возможности, спрос, доступность ресурсов представляются как "рюкзак". Различные виды продукции, которые могут быть произведены, выступают в роли "предметов".

Задача полного рюкзака помогает определить оптимальный ассортимент продукции, который максимизирует общую прибыль.

3.3 Задача полного рюкзака в сфере образования

В сфере образования эта задача может использоваться для составления учебных планов и расписания, формирования индивидуальных образовательных траекторий, распределения стипендий и грантов, а также отбора абитуриентов в учебные заведения.

При составлении учебных планов и расписания ограниченное количество учебных часов, аудиторий и преподавателей представляется как "рюкзак" с определенной вместимостью, а различные учебные дисциплины, курсы и занятия выступают в качестве "предметов", которые необходимо "уместить" в этот "рюкзак". Задача полного рюкзака помогает оптимально распределить учебные ресурсы, чтобы максимизировать количество предметов, которые могут быть изучены студентами.

При формировании индивидуальных образовательных траекторий ограниченное количество свободных учебных мест и возможностей финансирования представляется как "рюкзак", а различные образовательные программы, курсы по выбору и дополнительные занятия выступают в роли "предметов". Задача полного рюкзака используется для подбора оптимального набора курсов и дисциплин, которые максимально соответствуют интересам, способностям и ресурсам конкретного студента.

Задача полного рюкзака также применяется в распределении стипендий и грантов, где ограниченный бюджет на финансовую поддержку представляется как "рюкзак", а различные студенты, претендующие на

получение средств, выступают в роли "предметов". Решение данной задачи позволяет определить оптимальное распределение стипендий или грантов среди студентов, чтобы максимизировать общий образовательный эффект.

Кроме того, задача полного рюкзака используется в отборе абитуриентов в учебные заведения, где ограниченное количество мест на определенные направления подготовки представляется как "рюкзак", а абитуриенты, подавшие заявления, выступают в роли "предметов". В этом случае задача полного рюкзака применяется для отбора оптимального набора студентов, который максимизирует качество подготовки, соответствие профилю или другие критерии.

Таким образом, задача полного рюкзака является эффективным инструментом для оптимального распределения ограниченных образовательных ресурсов и принятия решений в сфере управления образовательными процессами.

3.3 Задача полного рюкзака в криптографии

Задача полного рюкзака в криптографии — это метод шифрования данных, основанный на сложности задачи о рюкзаке.

В криптографии задача полного рюкзака используется для создания криптосистемы с открытым ключом. В этой системе отправитель и получатель имеют разные ключи: открытый ключ, который известен всем, и закрытый ключ, который знает только получатель. Открытый ключ используется для шифрования сообщения, а закрытый ключ — для его расшифровки.

Идея использования задачи полного рюкзака для создания криптосистемы была предложена Ральфом Мерклом и Мартином Хеллманом в 1978 году. Они назвали свою систему «ранцевым шифрованием» (англ. «knapsack cryptosystem»).

Ранцевая криптосистема Меркла-Хеллмана работает следующим образом:

1. Отправитель генерирует два вектора A и B, где каждый элемент является случайным целым числом из определённого диапазона. Вектор A называется «секретным ключом», а вектор B — «открытым ключом».

2. Затем отправитель вычисляет «весовую функцию» f(x), которая представляет собой сумму произведений элементов векторов A и x. Функция f(x) должна быть трудно обратимой, то есть должно быть сложно найти такой вектор x, который бы удовлетворял уравнению f(x) = c, где c — заданное число.
3. После этого отправитель публикует открытый ключ B, а секретный ключ A сохраняет у себя.
4. Чтобы зашифровать сообщение m, отправитель разбивает его на блоки равной длины и преобразует каждый блок в целое число xi. Затем он вычисляет значения функции f(xi) для каждого блока и отправляет их получателю вместе с открытым ключом B.
5. Получатель, зная открытый ключ B, может легко вычислить значения функции f(xi). Однако без знания секретного ключа A он не сможет восстановить исходные сообщения m, так как задача о рюкзаке является NP-полной.

Несмотря на то что ранцевая криптосистема Меркла-Хеллмана была первой практической реализацией идеи открытого ключа, она оказалась уязвимой к атакам на основе подобранного открытого текста. Это связано с тем, что в некоторых вариантах задачи полного рюкзака можно подобрать открытый текст таким образом, чтобы получить информацию о секретном ключе. Например, если в задаче полного рюкзака используются только положительные целые числа, то можно попытаться найти такие значения x, которые будут давать небольшие значения функции f(x). Это может позволить злоумышленнику восстановить часть секретного ключа.

Заключение

В ходе данной научно-исследовательской работы был проведен всесторонний анализ задачи полного рюкзака, её различных вариаций, а также методов и алгоритмов для их решения. Кроме того, были изучены сферы применения этой классической задачи оптимизации.

Исследование показало, что задача полного рюкзака является фундаментальной задачей комбинаторной оптимизации, которая имеет широкий спектр применений в различных областях, включая менеджмент, логистику, финансы, а также сферу образования и информационной безопасности.

Были рассмотрены основные вариации задачи полного рюкзака. И основные способы решения задачи. Для каждой из этих вариаций были изучены

эффективные методы и алгоритмы решения, включая точные методы, такие как динамическое программирование, а также приближенные алгоритмы, например, жадный алгоритм.

Особое внимание было уделено практическим аспектам применения задачи полного рюкзака. Было проанализировано использование данной задачи в сфере образования, где она находит применение при составлении учебных планов, формировании индивидуальных образовательных траекторий, распределении стипендий и грантов, а также отборе абитуриентов. В сфере производства, где она применяется в логистике, хранении товаров и планах производства. Кроме того, был рассмотрен опыт применения задачи полного рюкзака в криптографии, где она используется для шифрования данных.

Полученные результаты свидетельствуют о важности и актуальности задачи полного рюкзака, а также о разнообразии её практических применений. Данная работа вносит вклад в понимание теоретических основ задачи полного рюкзака, а также демонстрирует её широкое использование в различных сферах человеческой деятельности.

Список литературы

1. Silvano Martello, Paolo Toth. Knapsack Problems: Algorithms and Computer Implementations — 1990. — 308c.
2. Семинар БИ «Ресурсно-эффективные алгоритмы» — Кафедра УРПО отделения программной инженерии — 2011-2012. — 26с.
3. David Pisinger. Algorithms for Knapsack Problems — Univ., Department of Computer Science — 1995. — 199c.
4. Левитин А. В. Алгоритмы. Введение в разработку и анализ — М.: Вильямс — 2006. — 576 с.
5. Б. Шнайер. Прикладная криптография. Протоколы, алгоритмы, исходные тексты на языке Си — 2-ое. — 2002. — 816 с.
6. Ковалёв, М. М. Дискретная оптимизация: Целочисленное программирование / М. М. Ковалёв. — 3-е. — 2011.— 192 с.
7. Методы оптимизации : пособие / Р. Габасов [и др.]. — Минск, 2011. — 472 с.

Приложение

Листинг разработанной программы

Листинг 1 — html разработанного интерактивного сайта.

```
<body>
<div class="main-container">
  <h1>Задача полного рюкзака</h1>
  <form class="form" id="form">
    <h3 class="title">Введите число элементов в наборе:</h3>
    <input class="input" name="count" type="number" value="10" placeholder="2-100" min="2" max="100" required>
    <h3 class="title">Введите максимальный вес рюкзака:</h3>
    <input class="input" name="weight" type="number" value="100" placeholder="10-1000" min="10" max="1000" required>
    <h3 class="title">Выберите алгоритм:</h3>
    <select class="select" name="algorithm" required>
      <option value="default">Метод полного перебора</option>
      <option value="tree">Метод ветвей и границ</option>
      <option value="dynamic">Динимическое программирование</option>
      <option value="greedy">Жадный алгоритм</option>
    </select>
    <button class="btn" type="submit">Найти оптимальный</button>
  </form>
  <div id="result-wrapper" class="hidden">
    <h3 id="method"></h3>
    <div>
      Исходный набор предметов: <span id="items"></span>
    </div>
    <div>
      Максимальный вес рюкзака: <span id="weight"></span>
    </div>
    <div>
      Оптимальный набор: <span id="result"></span>
    </div>
    <div>
      Конечный вес рюкзака: <span id="backpack-weight"></span>
    </div>
    <div>
      Конечная цена товаров: <span id="backpack-price"></span>
    </div>
  </div>
</div>
<script src="index.js"></script>
</body>
```

Листинг 2 — css разработанного интерактивного сайта.

```css
body, * {
  padding: 0;
  margin: 0;
  box-sizing: border-box;
}

.main-container {
  display: flex;
  flex-direction: column;
  align-items: center;
  justify-content: center;
  font-family: "GT Eesti Pro Display", sans-serif;
  padding: 20px;
}

.form {
  display: flex;
  flex-direction: column;
  align-items: center;
  width: 600px;
}

.input,
.select {
  height: 40px;
  padding: 10px;
  border: 1px solid #c4c4c4;
  border-radius: 5px;
  width: 100%;
}

.title {
  margin: 10px 0;
}

.btn {
  outline: none;
  background: none;
  border: 1px solid #c4c4c4;
  border-radius: 5px;
```

Листинг 3 — рендеринг результирующего блока разработанного интерактивного сайта.

```
const form : HTMLElement  = document.getElementById( elementId: 'form');
const resultWrapper : HTMLElement  = document.getElementById( elementId: 'result-wrapper');

 Dzhivanyan D.A
form.addEventListener( type: 'submit',  listener: (e : SubmitEvent ) : void  => {
  e.preventDefault();
  let result : any[]  = [];
  let title;

  const count = form.elements.count.value;
  const weight = form.elements.weight.value;
  const algorithm = form.elements.algorithm.value;

  const items : any[]  = createRandomArray(count,  max: weight * 0.75);

  switch (algorithm) {
    case 'tree':
      result = branch(items, weight);
      title = 'Метод ветвей и границ'
      break;
    case 'dynamic':
      result = dynamic(items, weight);
      title = 'Динимическое программирование'
      break;
    case 'greedy':
      result = greedy(items, weight);
      title = 'Жадный алгоритм'
      break;
    default:
      result = goingThrough(items, weight);
      title = 'Метод полного перебора'
  }

  resultWrapper.querySelector( selectors: '#method').textContent = title;
  resultWrapper.querySelector( selectors: '#items').textContent = JSON.stringify(items);
  resultWrapper.querySelector( selectors: '#weight').textContent = weight;
  resultWrapper.querySelector( selectors: '#result').textContent = JSON.stringify(result);
  resultWrapper.querySelector( selectors: '#backpack-weight').textContent = backpackWeight(result);
  resultWrapper.querySelector( selectors: '#backpack-price').textContent = backpackPrice(result);
```

Листинг 4 — Реализация функции полного перебора.

```
function goingThrough(arr, capacity) : any[]  {
  let result : any[]  = [];
  1+ usages
  function getAllSubarrays(arr, currentSubarray : any[]  = []) : void  {
    if (arr.length === 0) {
      if (backpackWeight(currentSubarray) <= capacity && backpackPrice(currentSubarray) > backpackPrice(result)) {
        result = [...currentSubarray];
      }

      return;
    }

    const firstElement = arr[0];
    const restOfArray = arr.slice(1);

    // Создаем новый подмассив без первого элемента
    getAllSubarrays(restOfArray, currentSubarray);

    // Создаем новый подмассив, включающий первый элемент
    getAllSubarrays(restOfArray,  currentSubarray: [...currentSubarray, firstElement]);
  }
  getAllSubarrays(arr);

  return result;
}
```

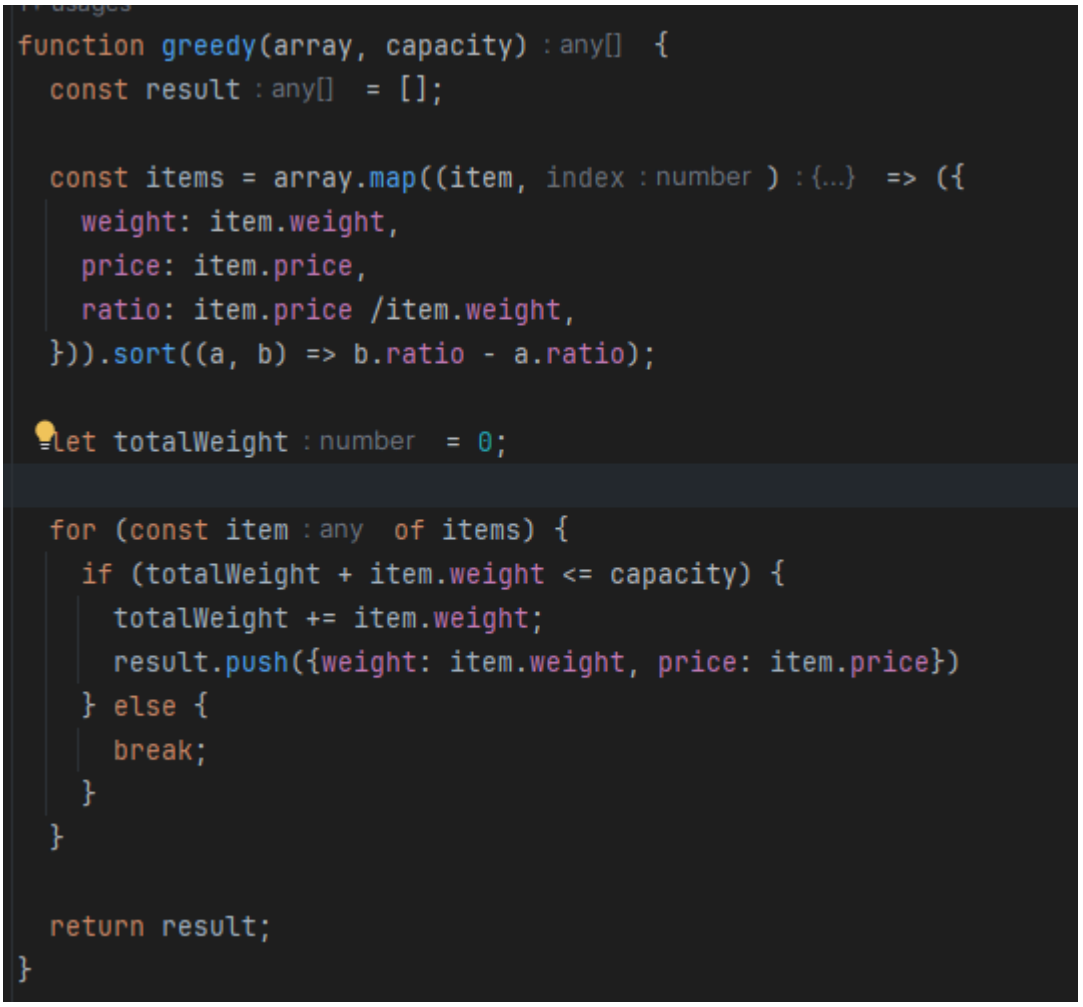
Листинг 5 — Реализация метода ветвей и границ.

```
function tree(array, capacity) : any[]  {
  const n = array.length;
  let bestValue : number  = 0;
  let bestSolution : any[]  = [];
  let weights = array.map((e) => e.weight);
  let values = array.map((e) : number  => e.price);

  1+ usages
  function dfs(currentWeight, currentValue, currentSolution) : void  {
    // Если текущий вес превышает максимальную грузоподъемность, возвращаемся
    if (currentWeight > capacity) {
      return;
    }

    // Обновляем лучшее решение, если текущее значение больше
    if (currentValue > bestValue) {
      bestValue = currentValue;
      bestSolution = currentSolution.slice();
    }

    // Пробуем добавить каждый оставшийся предмет
    for (let i : number  = 0; i < n; i++) {

      if (!currentSolution.includes(i)) {

        // Добавляем предмет
        currentSolution.push(i);

        dfs( currentWeight: currentWeight + weights[i],  currentValue: currentValue + values[i], currentSolution);
        currentSolution.pop(); // Откатываемся

        // Если текущее значение + оценка верхней границы меньше, чем лучшее решение, возвращаемся
        if (currentValue + calcUpperBound(currentWeight, i, values, weights) < bestValue) {
          return;
        }
      }
    }
  }
}
```

Листинг 3 — Реализация метода динамического программирования.

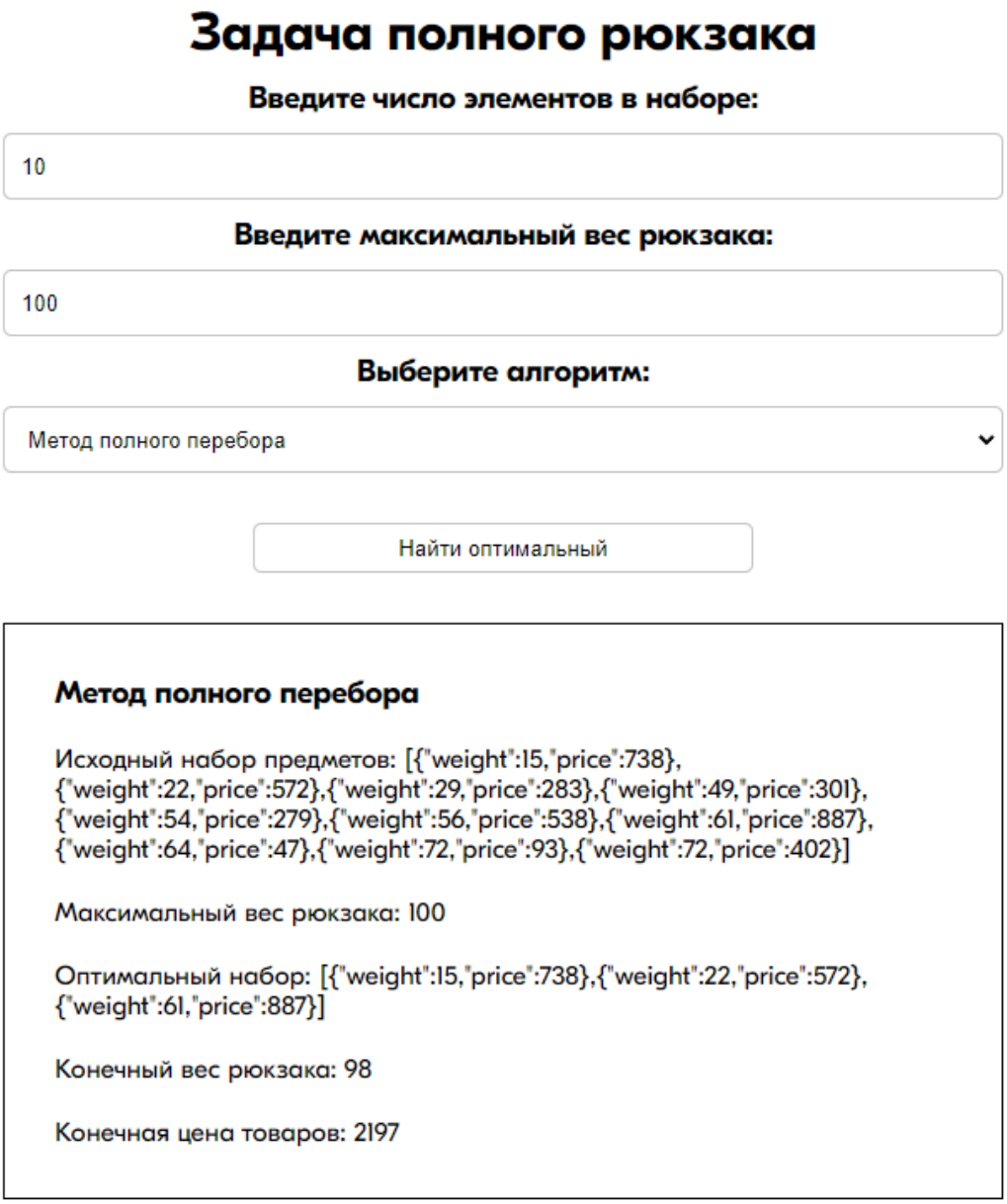
```
function dynamic(array, capacity) : any[]  {
  const dp : (...)[]  = new Array( arrayLength: array.length + 1).fill( value: 0).map(() : any[]  => new Array( arrayLength: capacity + 1).fill( value: 0));
  for (let i : number  = 0; i <= array.length; i++) {
    for (let j : number  = 0; j <= capacity; j++) {
      dp[i][j] = 0
    }
  }

  for (let i : number  = 1; i <= array.length; i++) {
    for (let j : number  = 1; j <= capacity; j++) {
      if (array[i - 1].weight <= j) {
        dp[i][j] = Math.max(
          dp[i - 1][j],
          dp[i - 1][j - array[i - 1].weight] + array[i - 1].price
        );
      } else {
        dp[i][j] = dp[i - 1][j];
      }
    }
  }

  let i = array.length, j = capacity;
  const result : any[]  = [];
  while (i > 0 && j > 0) {
    if (dp[i][j] !== dp[i - 1][j]) {
      result.push(array[i - 1]);
      j -= array[i - 1].weight;
    }
    i--;
  }

  return result;
```

Листинг 5 — Реализация метода жадного алгоритма.



```
function greedy(array, capacity) : any[]  {
  const result : any[]  = [];

  const items = array.map((item, index : number ) : {...}  => ({
    weight: item.weight,
    price: item.price,
    ratio: item.price /item.weight,
  })).sort((a, b) => b.ratio - a.ratio);

  let totalWeight : number  = 0;

  for (const item : any  of items) {
    if (totalWeight + item.weight <= capacity) {
      totalWeight += item.weight;
      result.push({weight: item.weight, price: item.price})
    } else {
      break;
    }
  }

  return result;
}
```

Листинг 6 — Визуальное отображение.



Ссылка на страницу опубликованная на хостинге: https://david-dzhivanyan.github.io/First-semester-R-D/

Ссылка на git репозиторий с кодом всей программы: https://github.com/David-Dzhivanyan/First-semester-R-D