ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

Отчёт по лабораторной работе № 8

« Задача о рюкзаке »

Выполнил работу

Шишмарёв Андрей

Академическая группа № С3100

Принято

Вершинин Владислав

Санкт-Петербург

2025

**Введение**

Описание: решить задачу о рюкзаке тремя способами: полный перебор, жадные алгоритмы, динамическое программирование.

Цель работы: освоить различные методы решения задачи о рюкзаке, чтобы изучить принцип их работы

Задачи:

1. Ознакомиться с методами, предложенными для решения данной задачи – полный перебор, жадный алгоритм, динамическое программирование.
2. Изучить особенности их реализации на языке С++
3. Реализовать данные алгоритмы
4. Провести анализ полученных результатов
5. Подготовить отчёт

Ограничения:

* + - 1. рекурсия для полного перебора
      2. сортировка и цикл для жадного алгоритма
      3. динамическое программирование с массивами/таблицами

**Реализация**

### **Полный перебор**

Метод полного перебора заключается в том, чтобы рассмотреть все возможные комбинации предметов и выбрать ту, которая максимальную стоимость, при этом «укладываясь» в ограничение по весу рюкзака.

Для этого используется рекурсивная функция, которая принимает на вход текущий вес рюкзака, массив со стоимостью и весом предметов, количество доступных предметов. Затем алгоритм «рассматривает» базовый случай – когда отсутствуют предметы или максимальный вес рюкзака установлен равным 0. Тогда функция возвращает 0.

Затем для каждого предмета проверяется, можно ли добавить его в рюкзак, при этом соблюдая ограничение по весу. Если предмет не помещается, то функция переходит к следующим предметам. Если же предмет помещается, то функция вызывается дважды – без добавления данного предмета и с его добавлением (учитывая вес и стоимость). После этого возвращается максимальная стоимость из двух случаев (предмет добавлен / предмет не добавлен).

Сложность:

* Лучший случай: O(2^n) – первый предмет сразу соответствует условию.
* Худший случай: O(2^n) – алгоритм также проверяет все возможные комбинации предметов.
* Средний случай: O(2^n).

Память:

* Основной массив: O(n).
* Переменные: O(1).

### **Жадный алгоритм**

Метод жадного алгоритма заключается в том, что он принимает решения на каждом шаге, основываясь только на локально наиболее выгодных выборах с расчётом, что эти локальные выборы приведут к оптимальному решению в целом.

Сначала для каждого предмета вычисляется его удельная стоимость (отношение стоимости к весу). Затем все предметы сортируются по убыванию удельной стоимости, после чего алгоритм начинает добавлять предметы в рюкзак на основании этой сортировки, начиная с самого ценного, пока не достигнет ограничения по весу. Если предмет не помещается в рюкзак, но рюкзак ещё не полон, то предмет просто пропускается.

Жадный алгоритм не гарантирует оптимальное решение для задачи о рюкзаке в случае, где все предметы являются цельными. Однако для задачи о рюкзаке с дробными предметами жадный алгоритм даёт оптимальное решение.

Сложность:

* Лучший случай: O(n \* log(n)).
* Худший случай: O(n \* log(n)).
* Средний случай: O(n \* log(n)).

Память:

* Основной массив: O(n).
* Переменные: O(1).

### **Динамическое программирование**

Метод динамического программирования заключается в том, что задача разбивается на подзадачи. В задаче о рюкзаке подзадачи формируются на основе выбора или отказа от каждого предмета.

Для начала создаётся двумерный массив (таблица), где строки соответствуют предметам, а столбцы – возможным весам рюкзака. Каждый элемент таблицы dp[i][w] будет хранить максимальную ценность, которую можно получить, используя первые i предметов и имея максимальный вес w. Для каждого предмета принимается решение, добавлять его в рюкзак. Если предмет добавляется, то его стоимость соответственно добавляется к максимальной стоимости, которую можно получить с оставшимся весом. Чтобы избежать повторных вычислений, результаты для каждой комбинации предметов и весов сохраняются. После заполнения таблицы, максимальная ценность для полного рюкзака будет находиться в правом нижнем углу таблицы.

Сложность:

* Лучший случай: O(n\*W) – проход по всем предметам (n) и для каждого предмета – по весам от 0 до W.
* Худший случай: O(n\*W).
* Средний случай: O(n\*W).

Память:

* Основной массив: O(n\*W).
* Переменные: O(1).

**Приложения**

#include <iostream>

#include <vector>

using namespace std;

// Рекурсивная функция для полного перебора

int rucksackRecursion(int W, vector<int>& weights, vector<int>& values, int n) {

    if (n == 0 || W == 0) {

        return 0; // Если нет предметов или вес рюкзака равен 0, возвращаем 0

    }

    // Если вес текущего предмета больше W, пропускаем его

    if (weights[n - 1] > W) {

        return rucksackRecursion(W, weights, values, n - 1);

    } else {

        // Возвращаем максимум из двух случаев:

        // 1. Не берем текущий предмет

        // 2. Берем текущий предмет

        return max(

            rucksackRecursion(W, weights, values, n - 1),

            values[n - 1] + rucksackRecursion(W - weights[n - 1], weights, values, n - 1)

        );

    }

}

int main() {

    vector<int> values = {60, 100, 120};    // Стоимости предметов

    vector<int> weights = {10, 20, 30};     // Вес предметов

    int W = 50;                             // Максимальный вес рюкзака

    int n = values.size();                  // Количество предметов

    int maxValue = rucksackRecursion(W, weights, values, n);

    cout << "Максимальная стоимость (при полном переборе): " << maxValue << endl;

    return 0;

}

#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

// Структура для хранения предмета

struct Item {

    int value;

    int weight;

    double ratio; // стоимость на единицу веса

    // Метод для вычисления соотношения стоимости к весу

    void calculateRatio() {

        ratio = (double)value / weight;

    }

};

// Функция для сравнения элементов при сортировке

bool compare(Item a, Item b) {

    return a.ratio > b.ratio; // Мы хотим, чтобы более высокая стоимость на единицу веса была первой

}

double rucksackGreedy(vector<Item>& items, int W) {

    sort(items.begin(), items.end(), compare); // Сортируем по соотношению стоимости к весу

    double totalValue = 0.0;

    for (auto& item : items) {

        if (W >= item.weight) {

            //Если в рюкзак вмещается весь предмет

            W -= item.weight;

            totalValue += item.value;

        } else {

            // Если не вмещается, берем дробную часть предмета

            totalValue += item.ratio \* W;

            break; // Рюкзак полон

        }

    }

    return totalValue;

}

int main() {

    vector<Item> items = { {60, 10}, {100, 20}, {120, 30} };

    for (auto& item : items) {

        item.calculateRatio();  // Рассчитываем соотношение

    }

    int W = 50; // Максимальный вес рюкзака

    double maxValue = rucksackGreedy(items, W);

    cout << "Максимальная стоимость (жадный алгоритм): " << maxValue << endl;

    return 0;

}

#include <iostream>

#include <vector>

using namespace std;

int rucksackDP(int W, vector<int>& weights, vector<int>& values, int n) {

    vector<vector<int>> dp(n + 1, vector<int>(W + 1, 0)); // Создаем таблицу для хранения результатов

    for (int i = 1; i <= n; i++) {

        for (int w = 0; w <= W; w++) {

            // Если текущий предмет не помещается в рюкзак

            if (weights[i - 1] <= w) {

                dp[i][w] = max(dp[i - 1][w], values[i - 1] + dp[i - 1][w - weights[i - 1]]);

            } else {

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

            }

        }

    }

    return dp[n][W]; // Возвращаем максимальную стоимость

}

int main() {

    vector<int> values = {60, 100, 120};  // Стоимости предметов

    vector<int> weights = {10, 20, 30};   // Вес предметов

    int W = 50;                           // Максимальный вес рюкзака

    int n = values.size();                // Количество предметов

    int maxValue = rucksackDP(W, weights, values, n);

    cout << "Максимальная стоимость (динамическое программирование): " << maxValue << endl;

    return 0;

}