

|  |  |
| --- | --- |
| **Министерство образования и науки**  **Российской Федерации**  **Государственное образовательное учреждение высшего образования «МОСКОВСКИЙ АВТОМОБИЛЬНО-ДОРОЖНЫЙ**  **ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**  **(МАДИ)»** |  |

**Кафедра «Высшая математика»**

**Отчет по дисциплине «Структуры и алгоритмы обработки данных»**

**Практическая работа №4**

**«Задача о заполнении рюкзака»**

**Выполнил:**

Учебная группа 1бПМ\_\_\_\_\_\_\_\_\_  
ФИО Греча К. П. \_\_\_\_\_

**Принял:**

Должность Старший преподаватель\_

Звание \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

ФИО \_ Кутейников И. А.\_\_\_\_\_

«\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2024г.

Цель:

В задаче о рюкзаке есть набор предметов. Каждый предмет имеет

название, вес и ценность. Требуется сложить вещи с максимальной

стоимостью в рюкзак, имеющий ограничение по весу.

Реализовать программу решающую задачу о заполнении рюкзака с

помощью:

1 Рекурсивного метода;

2 Метода динамического программирования;

3 Жадного алгоритма.

Для жадного алгоритма реализовать стратегии:

- заполняем в первую очередь предметами с максимальным

соотношением цена/вес (четные варианты)

В программе должен присутствовать класс «Предмет», обладающий

полями: название, вес, цена; и класс «Рюкзак», обладающим

Алгоритм рекурсивного метода:

static KnapsackResult knapsackRec(int[] w, int[] v, int n, int W) {  
 if (n <= 0) {  
 return new KnapsackResult(0, new ArrayList<>());  
 } else if (w[n - 1] > W) {  
 return *knapsackRec*(w, v, n - 1, W);  
 } else {  
 KnapsackResult withoutItem = *knapsackRec*(w, v, n - 1, W);  
  
 KnapsackResult withItem = *knapsackRec*(w, v, n - 1, W - w[n - 1]);  
 withItem.maxValue += v[n - 1];  
 withItem.items.add(n - 1);  
  
 if (withItem.maxValue > withoutItem.maxValue) {  
 return withItem;  
 } else {  
 return withoutItem;  
 }  
 }  
 // сложность Big O(2^N)  
}

Жадный алгоритм:

static int knapsackGreedy(int[] w, int[] v, int W, List<Integer> itemsSelected) {  
 int n = w.length;  
 double[] k = new double[n];  
 for (int i = 0; i < n; i++) {  
 k[i] = (double) v[i] / w[i];  
 }  
  
 // сортировка пузырьком для стоимости  
 for (int i = 0; i < n - 1; i++) {  
 for (int j = 0; j < n - i - 1; j++) {  
 if (k[j] < k[j + 1]) {  
 // Сортировка удельной стоимости  
 double tempCost = k[j];  
 k[j] = k[j + 1];  
 k[j + 1] = tempCost;  
 // Со сортировкой удельной стоимости сортируем веса  
 int tempWeight = w[j];  
 w[j] = w[j + 1];  
 w[j + 1] = tempWeight;  
 // А также значения  
 int tempValue = v[j];  
 v[j] = v[j + 1];  
 v[j + 1] = tempValue;  
 }  
 }  
 }  
  
 int currentWeight = 0; // Текущий вес в рюкзаке  
 int currentValue = 0; // Текущая стоимость  
  
 // Добавление предметов в рюкзак  
 for (int i = 0; i < n; i++) {  
 if (currentWeight + w[i] <= W) {  
 // Добавляем целиком, если есть место  
 currentWeight += w[i];  
 currentValue += v[i];  
 itemsSelected.add(i); // Добавляем индекс предмета в список выбранных  
 } else {  
 // Заканчиваем, если вес следующего предмета не помещается  
 break;  
 }  
 }  
 System.*out*.println(currentWeight+" - теккущий вес рюкзака");  
 return currentValue;  
}

Реализация динамическим программированием:

static class KnapsackResult {  
 int maxValue;  
 List<Integer> items;  
 public KnapsackResult(int maxValue, List<Integer> items) {  
 this.maxValue = maxValue;  
 this.items = items;  
 }  
}  
  
static KnapsackResult knapsackDP(int[] w, int[] v, int n, int W) {  
 int[][] m = new int[n + 1][W + 1];  
 for (int j = 0; j <= W; j++) {  
 m[0][j] = 0;  
 }  
 for (int i = 1; i <= n; i++) {  
 for (int j = 1; j <= W; j++) {  
 if (w[i - 1] > j)  
 m[i][j] = m[i - 1][j];  
 else  
 m[i][j] = Math.*max*(m[i - 1][j], m[i - 1][j - w[i - 1]] + v[i - 1]);  
 }  
 }  
 // сложность Big O(W\*n)  
  
 // нахождения предметов, которые составляют макс. стоимость  
 List<Integer> items = new ArrayList<>();  
 List<Integer> q=new ArrayList<>();  
 for (int i = n, k = W; i > 0 && k > 0; i--) {  
 if (m[i][k] != m[i - 1][k]) {  
 items.add(i - 1); // добавляем индекс предмета  
 k = k - w[i - 1]; // вычитаем вес предмета из оставшегося веса  
 q.add(w[i-1]);  
 }  
 }  
 return new KnapsackResult(m[n][W], items);  
}

**Результат:**

**Ввод:**

1. Заполнение списка предметов из файла

2. Добавление предмета

3. Изменение предмета

4. Удаление предмета

5. Задание максимального веса рюкзака

6. Просмотр содержимого рюкзака

7. Выбор способа решения задачи

8. Сравнение способов решения

Выберите пункт меню: 1

Введите имя файла: **C:/Programming/lab4.txt**

Список предметов заполнен из файла

Данные файла:

Кроссовки,2,50

Ручка,1,2

Очки,2,15

Форма,4,20

Кошелек,1,35

Ключи,5,10

Выберите пункт меню: 5

Введите максимальный вес рюкзака:

**11**

**Вывод:**

1. Заполнение списка предметов из файла

2. Добавление предмета

3. Изменение предмета

4. Удаление предмета

5. Задание максимального веса рюкзака

6. Просмотр содержимого рюкзака

7. Выбор способа решения задачи

8. Сравнение способов решения

Выберите пункт меню: 8

Решение рекурсивным методом: **122**

Время выполнения : **971900**

Решение методом динамического программирования: **122**

Время выполнения : **57900**

**10** - текущий вес рюкзака

Решение жадным методом: **122**

Время выполнения : **730600**

**Заключение:**

В процессе выполнения лабораторной работы были реализованы различные методы решение задачи с рюкзаком, а именно: метод динамического программирования, жадный алгоритм, рекурсивный метод. Также было реализовано меню с возможность записи данных из файла.