

|  |  |
| --- | --- |
| МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ  федеральное государственное бюджетное образовательное учреждение высшего образования  **«МОСКОВСКИЙ АВТОМОБИЛЬНО-ДОРОЖНЫЙ ГОСУДАРСТВЕННЫЙ**  **ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ (МАДИ)»** |  |

**Кафедра «Высшая математика»**

**Лабораторная работа № 4**

по дисциплине

«Структуры и алгоритмы обработки данных»

на тему:

**«**Задача о заполнении рюкзака**»**

**Выполнил:**

Учебная группа: 1бПМ   
ФИО: Кузнецов А.-С. О.

Подпись: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Руководитель лабораторной работы:**

Должность: старший преподаватель

Звание: б/з

ФИО: Кутейников И. А.

Подпись: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

«\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_20\_\_г.

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 весом

(нечетные варианты)

−

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 полями:

максимальный вес, текущий вес, список предметов в рюкзаке.

Реализовать меню:

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

Код:

class Item {  
 String name;  
 int weight;  
 int value;  
  
 public Item(String name, int weight, int value) {  
 this.name = name;  
 this.weight = weight;  
 this.value = value;  
 }  
  
 public String getName() {  
 return name;  
 }  
  
 public int getWeight() {  
 return weight;  
 }  
  
 public int getValue() {  
 return value;  
 }  
}  
  
class Knapsack {  
 int maxWeight;  
 int currentWeight;  
 List<Item> items;  
 List<Item> itemsForC;  
  
 public Knapsack(int maxWeight) {  
 this.maxWeight = maxWeight;  
 this.currentWeight = 0;  
 this.items = new ArrayList<>();  
 this.itemsForC = new ArrayList<>();  
 }  
  
 public void addItem(Item item) {  
 items.add(item);  
 currentWeight = currentWeight + item.getWeight();  
 System.*out*.println("Предмет добавлен: " + item.getName());  
 }  
  
 public void addItemForC(Item item) {  
 if (item.getWeight() <= maxWeight) {  
 itemsForC.add(item);  
 System.*out*.println("Предмет добавлен: " + item.getName());  
 } else {  
 System.*out*.println("Невозможно добавить предмет " + item.getName() + ", превышен максимальный вес рюкзака.");  
 }  
 }  
  
 public static void loadItemsFromFile(Knapsack knapsack, String filePath) {  
 File file = new File(filePath);  
 try (Scanner scanner = new Scanner(file)) {  
 while (scanner.hasNextLine()) {  
 String line = scanner.nextLine();  
 String[] parts = line.split(",");  
 if (parts.length == 3) {  
 String name = parts[0].trim();  
 int weight = Integer.*parseInt*(parts[1].trim());  
 int value = Integer.*parseInt*(parts[2].trim());  
 knapsack.addItemForC(new Item(name, weight, value));  
 }  
 }  
 System.*out*.println("Предметы успешно загружены из файла.");  
 } catch (FileNotFoundException e) {  
 System.*out*.println("Файл не найден: " + e.getMessage());  
 } catch (NumberFormatException e) {  
 System.*out*.println("Ошибка в формате данных: " + e.getMessage());  
 } catch (Exception e) {  
 System.*out*.println("Произошла ошибка при чтении файла: " + e.getMessage());  
 }  
 }  
  
 public void removeItem(String itemName) {  
 Item itemToRemove = null;  
 for (Item item : items) {  
 if (item.name.equalsIgnoreCase(itemName)) {  
 itemToRemove = item;  
 break;  
 }  
 }  
 if (itemToRemove != null) {  
 items.remove(itemToRemove);  
 currentWeight -= itemToRemove.weight;  
 }  
 }  
  
 public void updateItem(String itemName, int newWeight, int newValue) {  
 for (Item item : items) {  
 if (item.name.equalsIgnoreCase(itemName)) {  
 currentWeight -= item.weight;  
 item.weight = newWeight;  
 item.value = newValue;  
 currentWeight += newWeight;  
 break;  
 }  
 }  
 }  
  
 public void displayContents() {  
 if (items.isEmpty()) {  
 System.*out*.println("Рюкзак пуст.");  
 } else {  
 System.*out*.println("Содержимое рюкзака:");  
 for (Item item : items) {  
 System.*out*.println("Название: " + item.name + ", Вес: " + item.weight + ", Стоимость: " + item.value);  
 }  
 System.*out*.println("Текущий общий вес: " + currentWeight);  
 int n = (int) ((currentWeight \* 10) / maxWeight);  
 System.*out*.println("Заполненность рюкзака: " + n + "/10");  
 }  
 }  
 // Реализация методов решения задачи о рюкзаке  
  
 // Рекурсивный метод  
 public int recursiveKnapsack(int index, int weightLeft) {  
 if (index == itemsForC.size() || weightLeft == 0) {  
 return 0;  
 }  
 if (itemsForC.get(index).weight > weightLeft) {  
 return recursiveKnapsack(index + 1, weightLeft);  
 }  
 int withoutCurrentItem = recursiveKnapsack(index + 1, weightLeft);  
 int withCurrentItem = itemsForC.get(index).value + recursiveKnapsack(index + 1, weightLeft - itemsForC.get(index).weight);  
 return Math.*max*(withoutCurrentItem, withCurrentItem);  
 }  
  
 // Метод динамического программирования  
 public int dynamicKnapsack() {  
 int n = itemsForC.size();  
 int[][] dp = new int[n + 1][maxWeight + 1];  
  
 for (int i = 1; i <= n; i++) {  
 for (int w = 1; w <= maxWeight; w++) {  
 if (itemsForC.get(i - 1).weight > w) {  
 dp[i][w] = dp[i - 1][w];  
 } else {  
 dp[i][w] = Math.max(dp[i - 1][w], itemsForC.get(i - 1).value + dp[i - 1][w - itemsForC.get(i - 1).weight]);  
 }  
 }  
 }  
  
 int maxValue = dp[n][maxWeight];  
 if (items.isEmpty()) {  
 List<String> selectedItems = new ArrayList<>();  
 int totalWeight = maxWeight;  
  
 for (int i = n; i > 0 && maxValue > 0; i--) {  
 if (maxValue != dp[i - 1][totalWeight]) {  
 selectedItems.add(itemsForC.get(i - 1).name);  
 items.add(itemsForC.get(i - 1));  
 currentWeight += itemsForC.get(i - 1).weight;  
 maxValue -= itemsForC.get(i - 1).value;  
 totalWeight -= itemsForC.get(i - 1).weight;  
 }  
 }  
 }  
 for (Item item : items) {  
 System.*out*.println("Добавлен: " + item.getName());  
 }  
 return dp[n][maxWeight];  
 }  
  
 // Жадный алгоритм  
 public int greedyKnapsack(boolean oddStrategy) {  
 List<Item> sortedItems = new ArrayList<>(itemsForC);  
 if (oddStrategy) {  
 sortedItems.sort((a, b) -> b.weight - a.weight);  
 } else {  
 sortedItems.sort((a, b) -> Double.*compare*((double) b.value / b.weight, (double) a.value / a.weight));  
 }  
  
 int totalValue = 0;  
 int currentWeight = 0;  
  
 for (Item item : sortedItems) {  
 if (currentWeight + item.weight <= maxWeight) {  
 totalValue += item.value;  
 currentWeight += item.weight;  
 }  
 }  
  
 return totalValue;  
 }  
}

…

Результат:

Меню:

1. Заполнить список предметов из файла

2. Добавить предмет

3. Изменить предмет

4. Удалить предмет

5. Установить максимальный вес рюкзака

6. Просмотреть содержимое рюкзака

7. Выбрать метод решения

8. Сравнить методы решения

9. Выйти

Введите ваш выбор: 1

Введите путь к файлу: C:\Users\Александр\IdeaProjects\LabaPv\input.txt

Невозможно добавить предмет лев, превышен максимальный вес рюкзака.

Невозможно добавить предмет тигр, превышен максимальный вес рюкзака.

Невозможно добавить предмет собак, превышен максимальный вес рюкзака.

Невозможно добавить предмет кот, превышен максимальный вес рюкзака.

Предметы успешно загружены из файла.

Введите ваш выбор: 5

Введите максимальный вес рюкзака: 100

Введите ваш выбор: 1

Введите путь к файлу: C:\Users\Александр\IdeaProjects\LabaPv\input.txt

Предмет добавлен: лев

Предмет добавлен: тигр

Предмет добавлен: собак

Предмет добавлен: кот

Предметы успешно загружены из файла.

Введите ваш выбор: 7

Выберите метод решения:

1. Рекурсивный метод

2. Метод динамического программирования

3. Жадный алгоритм

1

Добавлен: собак

Добавлен: тигр

Добавлен: лев

Максимальная стоимость: 1200

Время выполнения: 5 мс

Введите ваш выбор: 6

Содержимое рюкзака:

Название: собак, Вес: 70, Стоимость: 900

Название: тигр, Вес: 14, Стоимость: 200

Название: лев, Вес: 12, Стоимость: 100

Текущий общий вес: 96

Заполненность рюкзака: 9/10

Введите ваш выбор: 7

Выберите метод решения:

1. Рекурсивный метод

2. Метод динамического программирования

3. Жадный алгоритм

2

Добавлен: собак

Добавлен: тигр

Добавлен: лев

Максимальная стоимость: 1200

Время выполнения: 0 мс

Введите ваш выбор: 6

Содержимое рюкзака:

Название: собак, Вес: 70, Стоимость: 900

Название: тигр, Вес: 14, Стоимость: 200

Название: лев, Вес: 12, Стоимость: 100

Текущий общий вес: 96

Заполненность рюкзака: 9/10

Введите ваш выбор: 7

Выберите метод решения:

1. Рекурсивный метод

2. Метод динамического программирования

3. Жадный алгоритм

3

Добавлен: собак

Добавлен: тигр

Добавлен: лев

Выберите стратегию для жадного алгоритма:

Максимальная цена (стратегия с нечетным весом): 300

Время выполнения: 1 мс

Максимальная цена (стратегия с отношением цена/вес): 1200

Время выполнения: 0 мс

Введите ваш выбор: 6

Содержимое рюкзака:

Название: собак, Вес: 70, Стоимость: 900

Название: тигр, Вес: 14, Стоимость: 200

Название: лев, Вес: 12, Стоимость: 100

Текущий общий вес: 96

Заполненность рюкзака: 9/10

Вывод:

В ходе данной лабораторной работы была решена задача оптимального заполнения рюкзака при помощи различных алгоритмов. В частности, было получено понимание того, как работать с жадными алгоритмами, а также, как при помощи полученных знаний оптимизировать решения множества задач.