Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Российский химико-технологический университет имени Д.И. Менделеева»

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №10

Выполнил студент группы КС-36: Золотухин Андрей Александрович
Ссылка на репозиторий: https://github.com/MUCTR-IKT-CPP/ZolotukhinAA_36_ALG
Принял: Крашенников Роман Сергеевич
Дата сдачи: 12.05.2025

Москва
2025

Оглавление

Описание задачи

Идея **динамического программирования** в том, что мы разбиваем задачу на малые подзадачи, спускаясь вниз, далее, находим самую минимальную подзадачу, решаем её, затем, имея решение, мы создаём подзадачи, которые включают в своё решение решённую более меньшую подзадачу, снова получае их решения, сохраняем их, и движемся далее вверх.

По итогу динамическое программирование требует следующий условий для решений какой-либо задачи:

- перекрывающиеся подзадачи;
- оптимальная подструктура;
- возможность запоминания решения часто встречающихся подзадач.

Описание метода/модели

Выполнение задачи

Задача реализована на языке *Java*.

```java
package com.dp;

import java.util.Random;

public class Main
{
  public static void main( String[] args )
  {
    Random random = new Random();
    int n = random.nextInt(20) + 1;
    int[] w = new int[n + 1];
    int total = 0;

    System.out.println("Generated stones: " + n);
    System.out.print("Weights of stones: ");

    for (int i = 1; i <= n; i++) {
    w[i] = random.nextInt(10000) + 1;
    total += w[i];
    System.out.print(w[i] + " ");
    }

    System.out.println("\nTotal weight: " + total);
    int minDiff = solveKnapsack(w, total);
    System.out.println("Min diff: " + minDiff);

    // Scanner sc = new Scanner(System.in);
    // int n = sc.nextInt();
    // int[] w = new int[n + 1];
    // int total = 0;

    // for (int i = 1; i <= n; i++) {
    //     w[i] = sc.nextInt();
    //     total += w[i];
    // }

    // System.out.println(solveKnapsack(w, total));
  }

  private static int solveKnapsack(int[] w, int total) {
    int W = total / 2;
    int[][] dp = new int[2][W + 1];

    for (int j = 0; j <= W; j++) {
      dp[0][j] = 0;
    }

    for (int i = 1; i < w.length; i++) {
      int curr = i % 2;
      int prev = 1 - curr;

      for (int j = 0; j <= W; j++) {
```

```
        if (w[i] <= j) {
          dp[curr][j] = Math.max(dp[prev][j], dp[prev][j - w[i]] + w[i]);
        } else {
          dp[curr][j] = dp[prev][j];
        }
      }
    }

    int maxSum = dp[(w.length - 1) % 2][W];
    return total - 2 * maxSum;
  }
}
```