Министерство образования Республики Беларусь
Учреждение образования
«Брестский государственный технический университет»
Кафедра ИИТ

Лабораторная работа №6
По дисциплине « **Алгоритмы и структуры данных** »
Тема: **«Задача о рюкзаке»**

Выполнил:
Студент 2 курса
Группы ПО-11(2)
Сымоник И.А
Проверила:
Глущенко Т.А

Брест 2023

Цель работы: изучить алгоритм поиска с возвратом.

Ход работы

Задание 1. *Динамическим программированием* решить *классическую задачу о рюкзаке*. Выбор входных параметров провести самостоятельно.

Исходный код:

```
#include <Windows.h>
#include <vector>
#include <iostream>

const int backpackSize = 25;
std::vector<int> costs = { 5, 8, 14, 1,4 };
std::vector<int> weights = { 7,6,12,1,8};

std::vector<int> answer;

void GetResult(const std::vector<std::vector<int>> &matrix, int i, int j)
{
   if (matrix[i][j] == 0)
      return;

   if (matrix[i][j] == matrix[i - 1][j])
   {
      GetResult(matrix, i - 1, j);
   }
   else
   {
      GetResult(matrix, i - 1, j - weights[i-1]);
      answer.push_back(i);
   }
}

std::vector<std::vector<int>> backPackII(int m, const std::vector<int>& weights, const std::vector<int>& costs)
{

   std::vector<std::vector<int>> matrix(weights.size() + 1, std::vector<int>(m + 1, false));

   for (int i = 0; i < m + 1; i++)
      matrix[0][i] = 0;
   for (int i = 0; i < weights.size() + 1; i++)
      matrix[i][0] = 0;

   for (int i = 1; i < weights.size() + 1; i++)
   {
      for (int j = 1; j < m + 1; j++)
      {
         if (j - weights[i - 1] < 0)
         {
            matrix[i][j] = matrix[i - 1][j];
         }
         else
         {
```

```
            matrix[i][j] = max(matrix[i - 1][j],
                    matrix[i - 1][j - weights[i - 1]] + costs[i - 1]);
        }
    }
  }

  std::cout << "Таблица мемоизации" << std::endl;
  for (int i = 0; i < matrix.size(); i++)
  {
    for (int j = 0; j < matrix[i].size(); j++)
    {
      std::cout << matrix[i][j] << " ";
    }
    std::cout << std::endl;
  }


  return matrix;
}


int main()
{
  SetConsoleCP(1251);
  SetConsoleOutputCP(1251);

  auto matrix = backPackII(backpackSize, weights, costs);

  GetResult(matrix, weights.size(), backpackSize);

  std::cout << "Результат: ";
  for (const auto& i : answer)
  {
    std::cout << i << " ";
  }

}
```

Результат:

```
Таблица мемоизации
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5
0 0 0 0 0 0 8 8 8 8 8 8 8 13 13 13 13 13 13 13 13 13 13 13 13
0 0 0 0 0 0 8 8 8 8 8 8 14 14 14 14 14 14 22 22 22 22 22 22 22 27
0 1 1 1 1 1 8 9 9 9 9 9 14 15 15 15 15 15 22 23 23 23 23 23 23 27
0 1 1 1 1 1 8 9 9 9 9 9 14 15 15 15 15 15 22 23 23 23 23 23 23 27
Результат: 1 2 3
```

Контрольные вопросы:

1. Какие точные методы решения данной задачи вы знаете?

- Метод перебора (метод равномерного поиска, перебор по сетке) — простейший из методов поиска значений действительно-значных функций по какому-либо из критериев сравнения
- Метод ветвей и границ является вариацией метода полного перебора с той разницей, что исключаются заведомо неоптимальные ветви дерева полного перебора. Как и метод полного перебора, он позволяет найти оптимальное решение и поэтому относится к точным алгоритмам.

2. Чему равна временная сложность решения данной задачи *полным перебором?*

Так как для каждого предмета существует 2 варианта: предмет кладётся в рюкзак либо предмет не кладётся в рюкзак. Тогда перебор всех возможных вариантов имеет временную сложность $O(2^n)$

3. Чему равна временная сложность решения данной задачи *динамическим программированием?*

Сложность алгоритма при решении задачи динамическим программированием равна – O(N*W), где N – количество предметов, W – максимальный вес рюкзака.

4. Описать решение *задачи о рюкзаке* с приведенными ниже входными данными *жадным алгоритмом* и *методом динамического программирования*. Каким из методов мы находим *оптимальное решение?*
   Грузоподъемность рюкзака, *W = 4*. Даны *3* предмета с весом w_i и ценой p_i: $w_1 = 1, p_1 = 30;\ w_2 = 2, p_2 = 50;\ w_3 = 3, p_3 = 60;$

Жадный алгоритм для задачи о рюкзаке состоит в следующем:

Вариант 1:

- Выбрать предмет максимальной стоимости.

- Упорядочить предметы по «удельной стоимости» (стоимости деленной на вес), и набивать рюкзак наиболее «удельно-дорогими» предметами, пока они влезают.

Первым предметом будет 3, так как его стоимость наибольшая. Следующим предметом будет 1, так как его удельная стоимость равна 30 (у 2 удельная стоимость равна 25).

Вариант 2:

- Выбрать предмет максимальной удельной стоимости.

- Упорядочить предметы по «удельной стоимости» (стоимости деленной на вес), и набивать рюкзак наиболее «удельно-дорогими» предметами, пока они влезают.

Первым предметом будет 1, так как его стоимость наибольшая. Следующим предметом будет 2, так как его удельная стоимость равна 25 (у 3 удельная стоимость равна 20).

Первый вариант алгоритма выбрал более оптимальный результат.

Динамическое программирование:

Первым шагом метода динамического программирования для задачи является создание таблицы мемоизации, где столбцы соответствуют весу, а строки предметам.

	0	1	2	3	4
№0	0	0	0	0	0
№1	0	30	30	30	30
№2	0	30	50	80	80
№3	0	30	50	80	90

Ячейка высчитывается по формуле

S[i, j] = max (S[i−1, j], цена i-го предмета + S[i−1, j−вес i-го предмета]),

где i — номер строки, j — столбца.

5. Что такое *NP-полная* задача и является ли *задача о рюкзаке NP-полной* задачей.

NP-полная задача(недетерминированного завершения за полиномиальное время) — задача с ответом «да» или «нет» из класса NP, к которой можно свести любую другую задачу из этого класса за полиномиальное время. Примеры: 3Sat, задача о вершинном покрытии, задача коммивояжёра.

NP – класс задач, проверяемых за полиномиальное время. Примеры таких задач: задача о выполнимости булевой формулы, задача о вершинном покрытии, задача о клике и т.д.

Задача о рюкзаке является NP-полной.

Вывод: изучили алгоритм поиска с возвратом.