

МОСКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Факультет Информационных технологий  
Кафедра СМАРТ-технологии

направление подготовки

09.03.02 «Информационные системы и технологии»

ЛАБОРАТОРНАЯ РАБОТА № 5

Дисциплина: Теория принятия решений

Тема: Разработать ПО решения задачи о рюкзаке с  
использованием метода динамического программирования

Выполнил: студент группы 211-723

Сергеев С. О.

30.11.2022

(Дата)

(Подпись)

Проверил: \_д.т.н. профессор Истомина Т. В.  
(Фамилия И.О., степень, звание)

(Оценка)

\_\_\_\_\_  
(Дата)

\_\_\_\_\_  
(Подпись)

Замечания:

\_\_\_\_\_  
\_\_\_\_\_

Москва

2022

# Отчет по лабораторной работе №5

**Цель:** выделить основные методы решения задачи о загрузке, классифицировать и сравнить эти методы.

**Задача:** реализовать алгоритмы решения классической задачи о рюкзаке.

## Ход решения

### 1. Метод полного перебора

Название метода говорит само за себя. Чтобы получить решение нужно перебрать все возможные варианты загрузки. И таких вариантов при  $n$ -предметах будет  $2^n$ .

Программная реализация:

```
#include <iostream>
#include <vector>
#include <iomanip>
#include <cmath>
using namespace std;

struct Thing
{
    int weight;
    int price;
};

int bruteForce(vector<Thing> _spisok, int _maxWeight, int& bestWeight)
{
    int maxPrice = 0;
    //Будем отмечать предмет как выбранный
    bool* used = new bool[_spisok.size()];
    for (int i = 0; i < _spisok.size(); i++)
        used[i] = false;
    //Обход комбинаций. Если в рюкзаке n предметов, то будет  $2^n$  комбинаций
    for (int i = 0; i < pow(2, _spisok.size()); i++)
    {
        int tempWeight = 0;
        int tempPrice = 0;
        int j = 0;
        //Отмечаем предмет
        while (used[j] && j < _spisok.size())
        {
            used[j] = false;
            j++;
        }
        used[j] = true;
        //Поиск стоимости
        for (int k = 0; k < _spisok.size(); k++)
        {
            if (used[k])
            {
                tempWeight += _spisok[k].weight;
                tempPrice += _spisok[k].price;
            }
        }
    }
}
```

```

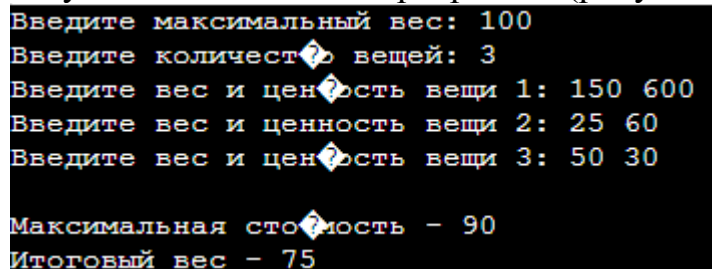
        if (tempPrice > maxPrice && tempWeight <= _maxWeight)
        {
            bestWeight = tempWeight;
            maxPrice = tempPrice;
        }
    }
    return maxPrice;
}

int main()
{
    setlocale(LC_ALL, "rus");
    int maxWeight = 0;
    int bestWeight = 0;
    int count;
    cout << "Введите максимальный вес: ";
    cin >> maxWeight;
    cout << "Введите количество вещей: ";
    cin >> count;
    vector<Thing> spisok(count);
    for (int i = 0; i < count; i++)
    {
        cout << "Введите вес и ценность вещи " << i + 1 << ": ";
        cin >> spisok[i].weight >> spisok[i].price;
    }
    cout << "\nМаксимальная стоимость - " << bruteForce(spisok, maxWeight, bestWeight) <<
endl;
    cout << "Итоговый вес - " << bestWeight << endl;

    return 0;
}

```

Результат выполнения программы (рисунок 1).



```

Введите максимальный вес: 100
Введите количество вещей: 3
Введите вес и ценность вещи 1: 150 600
Введите вес и ценность вещи 2: 25 60
Введите вес и ценность вещи 3: 50 30

Максимальная стоимость - 90
Итоговый вес - 75

```

Рисунок 1

## 2. Метод ветвей и границ

По существу, данный метод – это вариация полного перебора, с исключениями заведомо не оптимальных решений. Для полного перебора можно построить дерево решений. Если у нас есть какое-то оптимальное решение  $P$ , мы пытаемся улучшить его, но если на рассматриваемой в текущий момент ветви решение заведомо хуже, чем  $P$  то следует остановить поиск и выбрать другую ветвь для рассмотрения.

Программная реализация:

```
#include <iostream>
#include <vector>
#include <iomanip>
#include <cmath>
using namespace std;

struct Thing
{
    int weight;
    int price;
};

int branchBound(vector <Thing> _spisok, int _maxWeight, int& bestWeight)
{
    int maxPrice = 0;
    int k = 0; // Будем отмечать предмет как выбранный
    bool* used = new bool[_spisok.size()];
    for (int i = 0; i < _spisok.size(); i++)
        used[i] = false;
    // Обход комбинаций. Если в рюкзаке n предметов, то будет 2^n комбинаций
    for (int i = 0; i < pow(2, _spisok.size()); i++)
    {
        int tempWeight = 0;
        int tempPrice = 0;
        int stopPoint = i;
        int j = 0;
        while (stopPoint)
        {
            used[j] = bool(stopPoint % 2);
            stopPoint /= 2;
            if (used[j])
            {
                tempWeight += _spisok[j].weight;
                tempPrice += _spisok[j].price;
                if (tempWeight >= _maxWeight)
                    break;
            }
            j++;
        }
        if (tempPrice > maxPrice && tempWeight <= _maxWeight) {
            bestWeight = tempWeight;
            maxPrice = tempPrice;
        }
    }
    return maxPrice;
}

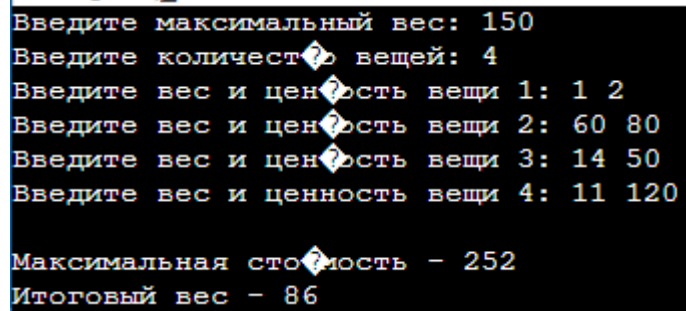
int main()
{
    setlocale(LC_ALL, "rus");
    int maxWeight = 0;
    int bestWeight = 0;
    int count;
    cout << "Введите максимальный вес: ";
```

```

cin >> maxWeight;
cout << "Введите количество вещей: ";
cin >> count;
vector<Thing> spisok(count);
for (int i = 0; i < count; i++)
{
    cout << "Введите вес и ценность вещи " << i + 1 << ": ";
    cin >> spisok[i].weight >> spisok[i].price;
}
cout << "\nМаксимальная стоимость - " << branchBound(spisok, maxWeight, bestWeight)
<< endl;
cout << "Итоговый вес - " << bestWeight << endl;
return 0;
}

```

Результат выполнения программы (рисунок 2)



```

Введите максимальный вес: 150
Введите количество вещей: 4
Введите вес и ценность вещи 1: 1 2
Введите вес и ценность вещи 2: 60 80
Введите вес и ценность вещи 3: 14 50
Введите вес и ценность вещи 4: 11 120

Максимальная стоимость - 252
Итоговый вес - 86

```

Рисунок 2

### 3. Жадный Алгоритм

В случае применения жадного алгоритма поступаем так: сортируем предметы по убыванию стоимости единицы каждого. Пытаемся поместить в рюкзак все что помещается, и одновременно наиболее дорогое по параметру  $P$ .

Программная реализация:

```
#include <iostream>
#include <vector>
#include <iomanip>
#include <cmath>
using namespace std;

struct Thing
{
    int weight;
    int price;
    double ratio;
};

void swap(int& a, int& b)
{
    int temp = a;
    a = b;
    b = temp;
}

int greedyAlgorithm(vector< Thing> _spisok, int _maxWeight, int& bestWeight)
{
    int maxPrice = 0; //Сортировка по убыванию стоимости единицы каждого
    for (int i = 0; i < _spisok.size() - 1; i++)
    {
        for (int j = 0; j < _spisok.size() - i - 1; j++)
        {
            if (_spisok[j].ratio < _spisok[j + 1].ratio)
            {
                swap(_spisok[j], _spisok[j + 1]);
            }
        }
    }
    //Помещаем в рюкзак
    for (int i = 0; i < _spisok.size(); i++)
    {
        if (bestWeight + _spisok[i].weight <= _maxWeight)
        {
            maxPrice += _spisok[i].price;
            bestWeight += _spisok[i].weight;
        }
        else
        {
            int wt = _maxWeight - bestWeight;
            maxPrice += (wt * _spisok[i].ratio);
            bestWeight += wt;
            break;
        }
    }
    return maxPrice;
}

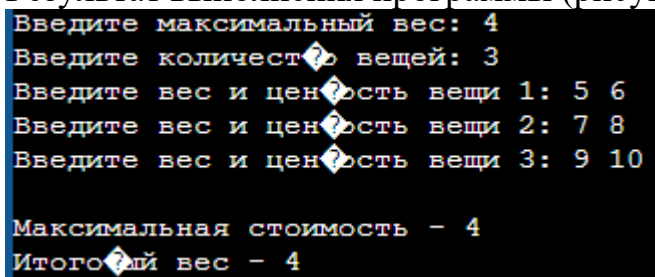
int main()
```

```

{
    setlocale(LC_ALL, "rus");
    int maxWeight = 0;
    int bestWeight = 0;
    int count;
    cout << "Введите максимальный вес: ";
    cin >> maxWeight;
    cout << "Введите количество вещей: ";
    cin >> count;
    vector<Thing> spisok(count);
    for (int i = 0; i < count; i++)
    {
        cout << "Введите вес и ценность вещи " << i + 1 << ": ";
        cin >> spisok[i].weight >> spisok[i].price;
        spisok[i].ratio = double(spisok[i].price) / double(spisok[i].weight);
    }
    cout << "\nМаксимальная стоимость - " << greedyAlgorithm(spisok, maxWeight,
bestWeight) << endl;
    cout << "Итоговый вес - " << bestWeight << endl;
    return 0;
}

```

Результат выполнения программы (рисунок 3)



```

Введите максимальный вес: 4
Введите количество вещей: 3
Введите вес и ценность вещи 1: 5 6
Введите вес и ценность вещи 2: 7 8
Введите вес и ценность вещи 3: 9 10

Максимальная стоимость - 4
Итоговый вес - 4

```

Рисунок 3

**Вывод:** в данной лабораторной работе были реализованы 3 основных метода решения задачи о рюкзаке (о загрузке).

Эти алгоритмы можно разделить на два типа: точные и приближенные. Точные – полный перебор, метод ветвей и границ (сокращение полного перебора). Приближенные алгоритмы – жадный алгоритм.

Предпочтение отдается точным методам, так как приближенные могут выдать неточный ответ.