Шибко Татьяна, 4 курс 12 группа

# Лабораторная работа 4

# Вариант 19

## Введение

В данной лабораторной работе мы решаем задачу о рюкзаке с неограниченным количеством предметов. Задача состоит в том, чтобы подобрать набор предметов, максимизирующих стоимость, при условии ограничения по весу рюкзака.

## Формулировка задачи

Мы решаем задачу вида:

$$\max 4x1 + 8x2 + 13x3$$
$$\text{s.t. } 3x1 + 4x2 + 5x3 \leq 10$$
$$x1, x2, x3 \geq 0 \text{ — целые.}$$

где:

- x1,x2,x3x1, x2, x3x1,x2,x3 — количество предметов каждого типа;
- 4, 8, 13 — стоимости предметов;
- 3, 4, 5 — веса предметов;
- Ограничение по весу рюкзака — 10.

## Математическая модель

Цель состоит в максимизации общей стоимости предметов, где каждый предмет может быть взят любое количество раз. Решение будет представлено с использованием алгоритма динамического программирования.

## Решение на Python

Алгоритм динамического программирования был реализован на Python для нахождения максимальной стоимости и количества каждого предмета, выбранного в оптимальном решении.

## Код на Python:

```
def knapsack_dp(values, weights, W):
    n = len(values)
    f = [0] * (W + 1)  # Таблица для хранения максимальной стоимости

    # Заполняем таблицу динамического программирования
    for i in range(n):
```

```
        for w in range(weights[i], W + 1):
            f[w] = max(f[w], f[w - weights[i]] + values[i])

    return f[W]

# Данные задачи
values = [4, 8, 13]  # Стоимость предметов
weights = [3, 4, 5]  # Масса предметов
W = 10  # Ограничение по массе

# Поиск максимальной стоимости
result = knapsack_dp(values, weights, W)
print(f"Оптимальная стоимость: {result}")
```

**Алгоритм обратного хода**

Для восстановления количества каждого предмета в решении используется следующий алгоритм:

```
def knapsack_solution(values, weights, W):
    n = len(values)
    f = [0] * (W + 1)
    p = [-1] * (W + 1)

    for i in range(n):
        for w in range(weights[i], W + 1):
            if f[w] < f[w - weights[i]] + values[i]:
                f[w] = f[w - weights[i]] + values[i]
                p[w] = i

    # Обратный ход для поиска количества предметов
    w = W
    solution = [0] * n
    while w > 0 and p[w] != -1:
        item = p[w]
        solution[item] += 1
        w -= weights[item]

    return solution

# Поиск количества предметов
optimal_solution = knapsack_solution(values, weights, W)
print(f"Количество предметов: {optimal_solution}")
```

**Решение на AMPL**

Для решения задачи в AMPL модель и данные были разделены на три файла: .mod, .dat и .run.

**Модель задачи (knapsack.mod):**

```
param W;  # Вместимость рюкзака

set ITEMS;

param weight{ITEMS};
param value{ITEMS};

# Переменные
var x{ITEMS} >= 0, integer;

# Целевая функция
maximize total_value:
   sum{i in ITEMS} value[i] * x[i];

# Ограничение на вес
subject to weight_constraint:
   sum{i in ITEMS} weight[i] * x[i] <= W;
```

**Данные (knapsack.dat):**

```
param W := 10;

set ITEMS := 1 2 3;

param weight :=
   1 3
   2 4
   3 5;

param value :=
   1 4
   2 8
   3 13;
```

**Файл запуска (knapsack.run):**

```
reset;
model knapsack.mod;
data knapsack.dat;
option solver cbc;
```

solve;
display x;

**Результаты**

- **Оптимальная стоимость:** 26
- **Количество предметов:** [0, 0, 2], то есть два предмета третьего типа, остальные не выбираются.

**AMPL**

```
ampl: include knapsack.run;
cbc 2.10.10: optimal solution; objective 26
0 simplex iterations
x [*] :=
1  0
2  0
3  2
;
```

**Python**

```
Оптимальная стоимость: 26
Количество предметов: [0, 0, 2]
```