Жуковский Павел 3 курс 12 группа

Лабораторная работа №4

Вариант 8

Описание задачи

Задача 8.

Решите следующую задачу о рюкзаке, записав результаты рекурсии динамического программирования в таблицу. Напишите алгоритм обратного хода. Задача:

$$\{\ 3x_1 + 7x_2 + 15x_3\ \} \to max$$

$$s.t.\ 2x_1 + 3x_2 + 6x_3 \leq 10$$

$$x_1, x_2, x_3 \geq 0 - \text{целые}$$

Решение

Данную задачу можно решать достаточно известным алгоритмом к задаче «о рюкзаке». Существует алгоритм «0/1 рюкзака», который предполагает, что каждого из N ресурсов у нас только по одному, однако мы реализуем более общий алгоритм, которые предполагает, что каждого экземпляра у нас может быть сколь угодно много, лишь бы выполнялись ограничения (в нашем случае, не более 10 вся сумма и все параметры – положительные целые числа).

Реализация

Данную задачу я решил реализовать на языке программирования Python (хотя, на самом деле выбор языка для данной задачи несущественен). Листинг программы следующий:

```
# Жуковский Павел, 3 курс, 12 группа
# ИСО, Лабораторная Работа №4, Вариант 8

import numpy as np

"""
Вариант 8
Найти такие X1, X2, X3, что:
3*X1 + 7*X2 + 15*X3 --> max
```

```
s.t. 2*X1 + 3*X2 + 6*X3 <= 10
X1, X2, X3 >= 0 - целые
"""

# Количество предметов
my_N = 3

# Ценности предметов
my_v = np.array([3, 7, 15])

# Веса предметов
my_w = np.array([2, 3, 6])

# Максимальная грузоподъёмность:
my_W = 10

# Матрица, содержащая в себе таблицу
my_Table = np.zeros((my_N, my_W))

# Функция, решающая задачу о рюкзаке 0/1 алгоритмом, обновляя данные в таблице
Table
# Вход:
# Таблица с данными Table
# Ценности предметов(загруженные в массив v)
# Веса предметов(загруженные в массив w)
# Количество предметов(n)
# Грузоподъемность(W)
# Возвращает элемент последнего столбца последней строки
def Knapsack(Table, N, v, w, W):
    for j in range(W):
        Table[0][j] = 0
    for i in range(N):
        for j in range(W):
            if w[i] > j:
                Table[i][j] = Table[i - 1][j]
            else:
                Table[i][j] = max(Table[i][j - 1], Table[i][j - w[i]] + v[i])
    return Table[N - 1][W - 1]

Result = Knapsack(my_Table, my_N, my_v, my_w, my_W)

cost_left = Result
X = np.zeros(3)
line = my_N - 1
col = my_W - 1
while cost_left != 0:
    while my_Table[line][col] < my_v[line]:
        line -= 1
    cost_left -= my_v[line]
    X[line] += 1
    col -= my_w[line]

print("Задача:")
print("max " + str(my_v[0]) + "*X1 + " + str(my_v[1]) + "*X2 + " + str(my_v[2])
+ "*X3")
print("s.t. " + str(my_w[0]) + "*X1 + " + str(my_w[1]) + "*X2 + " + str(my_w[2])
+ "*X3 <=", my_W)
print("X1, X2, X3 >= 0 - целые")
print("Таблица, после работы алгоритма:")
```

```
print(my_Table)
print("Оптимальное решение max =", Result, "достигается при следующих значениях:")
print("x1 =", X[0], ", x2 =", X[1], ", x3 =", X[2])
```

В общем-то, почти на каждый блок кода есть комментарии, но ещё раз объясним каждое действие.

Для начала, нам понадобится библиотека numpy, чтобы было проще работать с массивами и матрицами:

```
import numpy as np
```

В коде она нам понадобится чуть позже.

Далее, инициализируем все переменные нашими данными:

```
# Количество предметов
my_N = 3

# Ценности предметов
my_v = np.array([3, 7, 15])

# Веса предметов
my_w = np.array([2, 3, 6])

# Максимальная грузоподъёмность:
my_W = 10

# Матрица, содержащая в себе таблицу
my_Table = np.zeros((my_N, my_W))
```

В переменной **my_N** храним наше количество переменных (в нашем случае три, т.к. есть только x_1, x_2, x_3).

В переменной **my_v** храним стоимость предметов (это коэффициенты в первой строке нашей задачи – [3, 7, 15])

В переменной **my_w** храним веса предметов (это коэффициенты во второй строке нашей задачи – [2, 3, 6])

В переменной **my_W** храним общую «грузоподъёмность», другими словами – то значение, которое нас ограничивает (в нашем случае, это 10)

И на конец нам понадобится матрица **my_Table** размера **my_N** на **my_W**, ну или в нашем случае 3x10, чтобы в неё во время работы алгоритма записывать данные. Здесь мы используем один из методов библиотеки numpy – np.zeros((N, M)), которые возвращает нам матрицу размера NxM, заполненную нулями.

Все переменные названы с префиксом «my_», чтобы не было похожих имён с параметрами функции (о ней далее), в которой будет реализован алгоритм.

Далее идёт функция, которой на вход мы передаём все данные, а она с помощью алгоритма заполняет матрицу **my_Table** и возвращает результат – максимальное возможное и в то же время допустимое значение в нашей задаче, которое мы записываем в переменную **Result**:

```python
# Функция, решающая задачу о рюкзаке 0/1 алгоритмом, обновляя данные в таблице Table
# Вход:
# Таблица с данными Table
# Ценности предметов(загруженные в массив v)
# Веса предметов(загруженные в массив w)
# Количество предметов(n)
# Грузоподъемность(W)
# Возвращает элемент последнего столбца последней строки
def Knapsack(Table, N, v, w, W):
    for j in range(W):
        Table[0][j] = 0
    for i in range(N):
        for j in range(W):
            if w[i] > j:
                Table[i][j] = Table[i - 1][j]
            else:
                Table[i][j] = max(Table[i][j - 1], Table[i][j - w[i]] + v[i])
    return Table[N - 1][W - 1]

Result = Knapsack(my_Table, my_N, my_v, my_w, my_W)
```

Следующий блок кода нужен для того, чтобы определить, какие именно переменные (x_1, x_2 или x_3) и в каких количествах составляют наше максимальное значение:

```python
    cost_left = Result
    X = np.zeros(3)
    line = my_N - 1
    col = my_W - 1
    while cost_left != 0:
        while my_Table[line][col] < my_v[line]:
            line -= 1
        cost_left -= my_v[line]
        X[line] += 1
        col -= my_w[line]
```

Таким образом, внутри вектора X = [x_1, x_2, x_3] будет получен ответ.

Результаты

В конце своей работы программа вывела следующие результаты:

```
Задача:
max 3*X1 + 7*X2 + 15*X3
s.t. 2*X1 + 3*X2 + 6*X3 <= 10
X1, X2, X3 >= 0 - целые
Таблица, после работы алгоритма:
[[ 0.  0.  3.  3.  6.  6.  9.  9. 12. 12.]
 [ 0.  0.  3.  7.  7. 10. 14. 14. 17. 21.]
 [ 0.  0.  3.  7.  7. 10. 15. 15. 18. 22.]]
Оптимальное решение max = 22.0 достигается при следующих значениях:
x1 = 0.0 , x2 = 1.0 , x3 = 1.0

Process finished with exit code 0
```

Выводы

Исходя из вывода программы можно сделать вывод, что максимальное возможное значение, которое «можно поместить в рюкзак»:

Result = 3*0 + 7*1 + 15*1 = **22**.

Оно достигается при: $x_1 = 0$, $x_2 = 1$, $x_3 = 1$.

Ограничения соблюдены: $2*0 + 3*1 + 6*1 = 9 \leq 10$