**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:

**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:

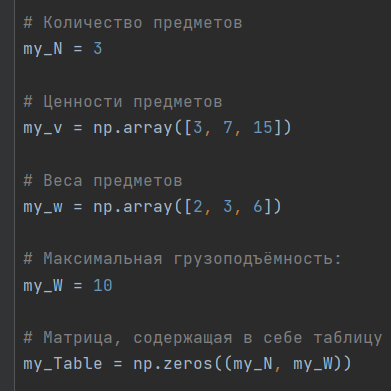
# Жуковский Павел, 3 курс, 12 группа  
# ИСО, Лабораторная Работа №4, Вариант 8  
  
import numpy as np  
  
"""  
Вариант 8  
Найти такие X1, X2, X3, что:  
3\*X1 + 7\*X2 + 15\*X3 --> max  
s.t. 2\*X1 + 3\*X2 + 6\*X3 <= 10  
X1, X2, X3 >= 0 - целые  
"""  
  
# Количество предметов  
my\_N = 3  
  
# 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:  


В коде она нам понадобится чуть позже.

Далее, инициализируем все переменные нашими данными:



В переменной **my\_N** храним наше количество переменных (в нашем случае три, т.к. есть только x1, x2, x3).

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

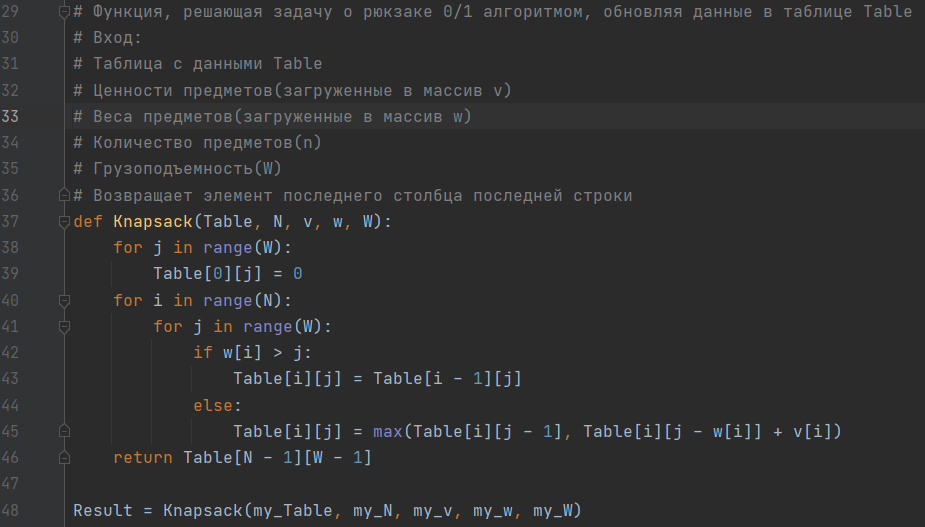
В переменной **my\_w** храним веса предметов (это коэффициенты во второй строке нашей задачи – [2, 3, 6])

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

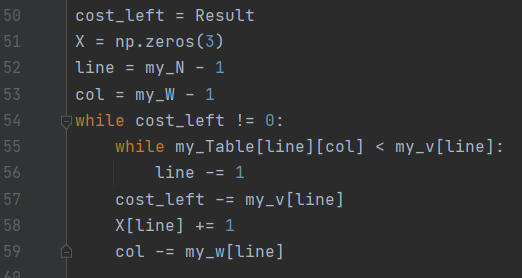
И на конец нам понадобится матрица **my\_Table** размера **my\_N** на **my\_W**, ну или в нашем случае 3x10, чтобы в неё во время работы алгоритма записывать данные. Здесь мы используем один из методов библиотеки numpy – np.zeros((N, M)), которые возвращает нам матрицу размера NxM, заполненную нулями.

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**:



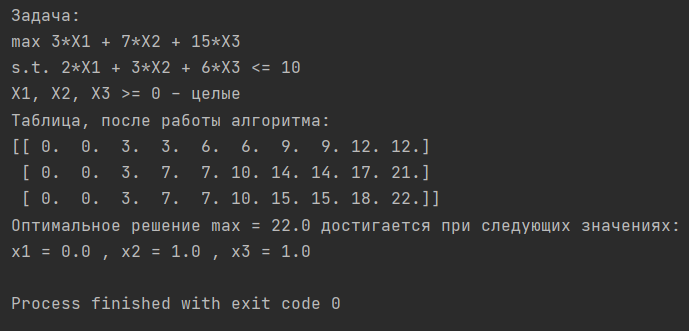
Следующий блок кода нужен для того, чтобы определить, какие именно переменные (x1, x2 или x3) и в каких количествах составляют наше максимальное значение:



Таким образом, внутри вектора X = [x1, x2, x3] будет получен ответ.

**Результаты**

В конце своей работы программа вывела следующие результаты:



**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: x1 = 0, x2 = 1, x3 = 1.

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