Тверской Государственный технический университет

Кафедра: Программного обеспечения и вычислительной

техники

Лабораторная работа № 4

Выполнил: Студент второго курса

Группы Б.ПИН.РИС 18.05

Федотов Ярослав Всеволодович

Тверь 2019

**Задача об одномерном рюкзаке**

1. Реализовать решение задачи с перебором вариантов.
2. Реализовать алгоритм с временной сложностью для «задачи о ранце».
3. Реализовать алгоритм с временной сложностью и сложностью по памяти для измененных условий задачи.
4. Реализовать решение на с. 235. Определить реккурентное выражение для решения.
5. Решить типовую задачу, похожую на «задачу о ранце» методом динамического программирования (используя изученную задачу о рюкзаке):

Дан брус длиной . Его можно разрезать на бруски типов. Длины брусков и их стоимости известны. Требуется найти такой способ разрезания бруса, при котором суммарная стоимость полученных брусков будет максимальной.

**Решение задач:**

1. Реализовать решение задачи с перебором вариантов.

for i in range(ItemsCount):# Начало формирование результата

                    for j in range(int(WeightItems[i]), ItemsWeight):

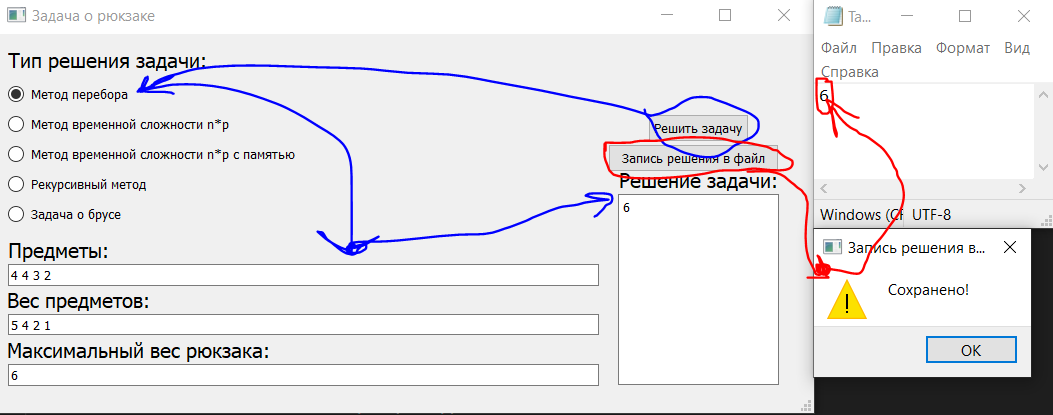
                        WeightItems[j] = NumbersItems[j]# Приравниваем текущий вес к текущему элементу

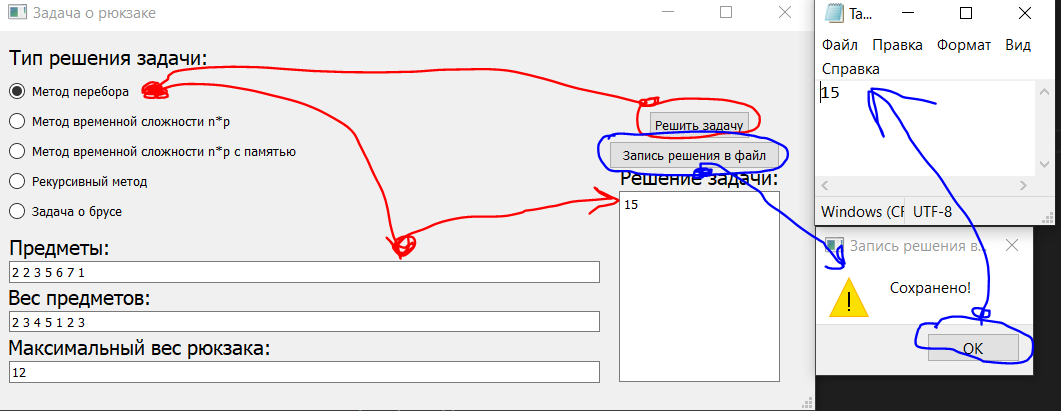
                        if(int(NumbersItems[j - int(WeightItems[i])])+int(WeightItems[i]) >int(WeightItems[j])): #Если текущий элемент + его текущий вес больше текущего веса, то

                            WeightItems[j] = int(NumbersItems[j-int(WeightItems[i])]) + int(WeightItems[i]) # Текущий вес будет равен текущему элементу – его вес + вес текущего элемента по столбцу.

                    for j in range(ItemsWeight):# Вывод результата

                        NumbersItems[j] = WeightItems[j]# Меняем местами вес и элемент

                self.ui.textResultEdit.setText(str(WeightItems[ItemsWeight - 1]))#Выводим результат



2.Реализовать алгоритм с временной сложностью для «задачи о ранце».

F = np.zeros((ItemsCount, ItemsWeight)) # Задаём исходную матрицу из размера элементов и их веса

                for k in range(ItemsCount): # Заполняем матрицу по элементам

                    F[k][k] = NumbersItems[k]

                for i in range(ItemsCount): # Начинаем поиск элемента по (NP)

                    for j in range(ItemsWeight): # Вложенный цикл

                        F[i][j] = F[i - 1][j] # Текущий элемент матрицы по [I,j]

Приравнием к предыдущему элементу по i и к текущему j

                        if (j >= int(WeightItems[i])) and ( # Если циклическая j будет больше или равна весу текущего элемента и если Элемент по i – 1 и j – вес текущего элемента + он же больше текущего элемента, то текущий элемент равен предыдущему по I – 1 и элементу по J- текущий вес элемента + он же.

                            F[i - 1][j - int(WeightItems[i])] + int(WeightItems[i])

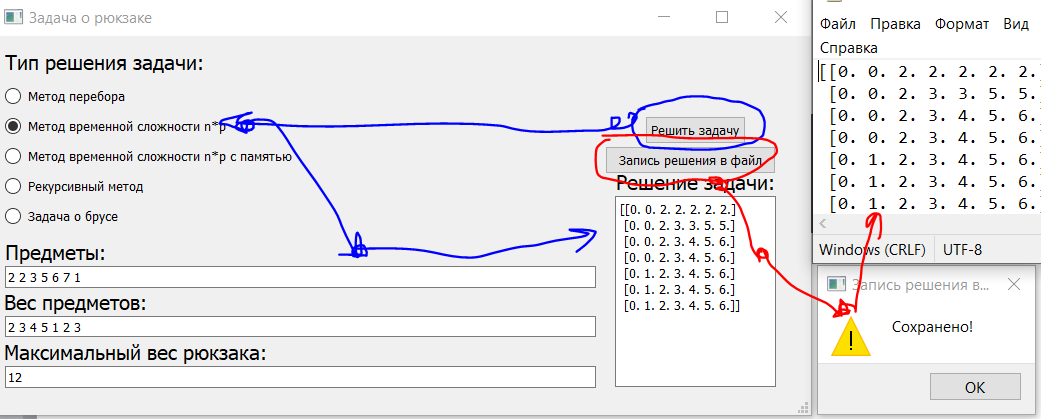
                            > F[i][j]

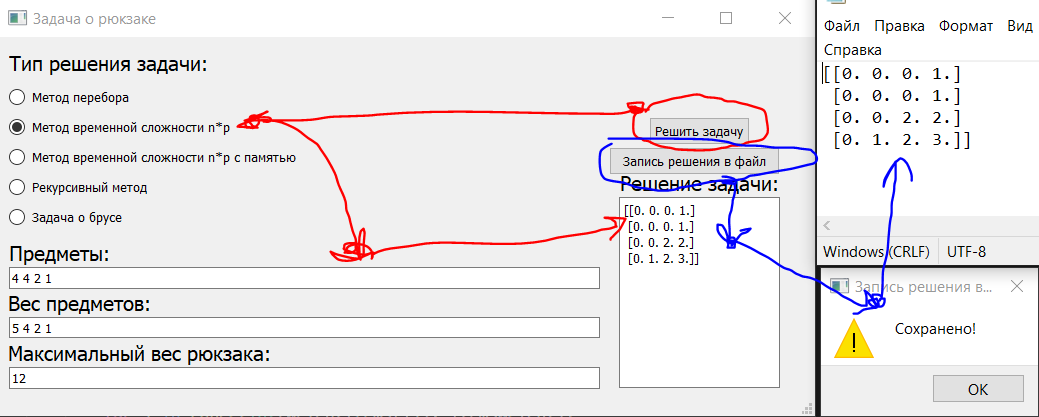
                        ):

                            F[i][j] = F[i - 1][j - int(WeightItems[i])] + int(

                                WeightItems[i]

                            )





3.Реализовать алгоритм с временной сложностью и сложностью по памяти для измененных условий задачи.

for i in range(ItemsCount):# Начало формирование результата

                    for j in range(int(WeightItems[i]), ItemsWeight):

                        WeightItems[j] = NumbersItems[j]# Приравниваем текущий вес к текущему элементу

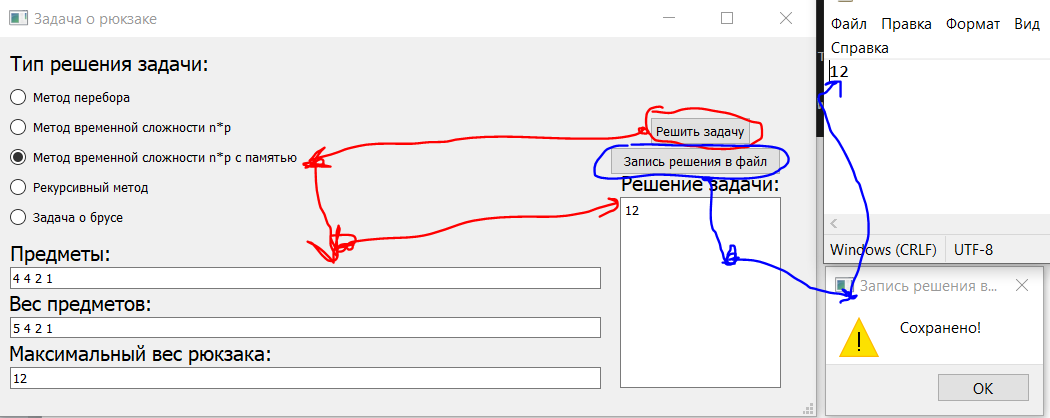
                        if(int(NumbersItems[j - int(WeightItems[i])])+int(WeightItems[i]) >int(WeightItems[j])): #Если текущий элемент + его текущий вес больше текущего веса, то

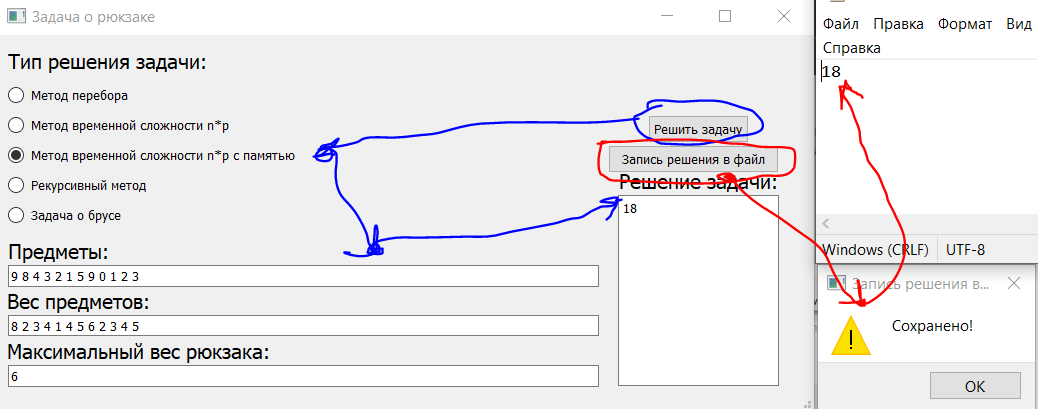
                            WeightItems[j] = int(NumbersItems[j-int(WeightItems[i])]) + int(WeightItems[i]) # Текущий вес будет равен текущему элементу – его вес + вес текущего элемента по столбцу.

                    for j in range(ItemsWeight):# Вывод результата

                        NumbersItems[j] = WeightItems[j]# Меняем местами вес и элемент

                self.ui.textResultEdit.setText(str(WeightItems[ItemsWeight - 1]))#Выводим результат





4.Реализовать решение на с. 235. Определить реккурентное выражение для решения.

result = [] # Вывод результата по количеству взятых предметов

                weight\_limit = capacity # Максимальный вес рюкзака

                for j in range(ItemsCount): #Цикл заполнения вещами рюкзака

                    items = ( #

                        Item(float(NumbersItems[j]), float(WeightItems[j])),#1-й предмет

                        Item(float(NumbersItems[j]), float(WeightItems[j])),#2-й предмет

                        Item(float(NumbersItems[j]), float(WeightItems[j])),#3-й

предмет

                        Item(float(NumbersItems[j]), float(WeightItems[j])),#4-й

Предмет

У каждого предмета есть свой вес и значение

Большее значение предмета формируется в методе best\_value

@lru\_cache(maxsize=None)  # cache all calls

def best\_value(nitems, weight\_limit):

    if nitems == 0:  # no items

        return 0  # zero value

    elif items[nitems - 1].weight > weight\_limit:

        # new item is heavier than the current weight limit

        return best\_value(nitems - 1, weight\_limit)  # don't include new item

    else:

        return max(  # max of with and without the new item

            best\_value(nitems - 1, weight\_limit),  # without

            best\_value(nitems - 1, weight\_limit - items[nitems - 1].weight)

            + items[nitems - 1].value,

        )  # with the new item

                    )

                for i in reversed(range(len(items))):#Заполнение массива всеми предметами

                    if best\_value(i + 1, weight\_limit) > best\_value(i, weight\_limit):#Если индекс + 1 и макс вес рюкзака больше значение по индексу и макс весу

                        # better with the i-th item

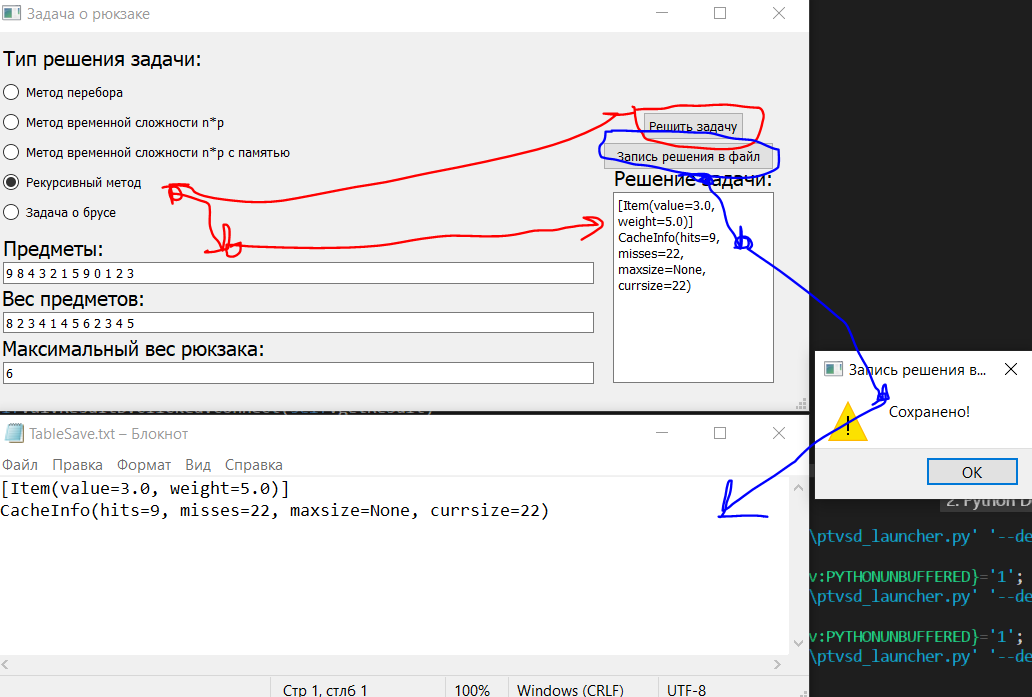
                        result.append(items[i])  # include it in the result то добавляем данный предмет в рюкзак

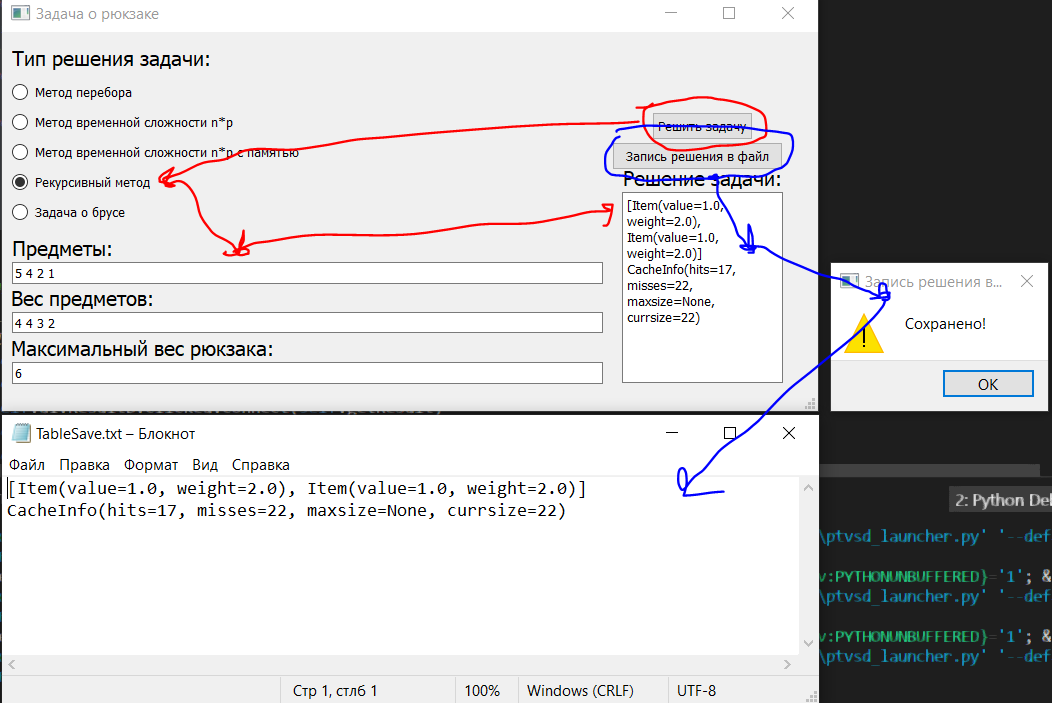
                        weight\_limit -= items[i].weight# и отнимаем массу так как рюзкак заполняется

                self.ui.textResultEdit.setText(#Выводим какие предметы мы положили в рюкзак ,а так также cache best\_value для значений использованных при счёте(вес предмета)

                    str(result) + "\n" + str(best\_value.cache\_info())

                )





5.Решить типовую задачу, похожую на «задачу о ранце» методом динамического программирования (используя изученную задачу о рюкзаке):

Дан брус длиной . Его можно разрезать на бруски типов. Длины брусков и их стоимости известны. Требуется найти такой способ разрезания бруса, при котором суммарная стоимость полученных брусков будет максимальной.

for i in range(ItemsCount):# Начало формирование результата

                    for j in range(int(WeightItems[i]), ItemsWeight):

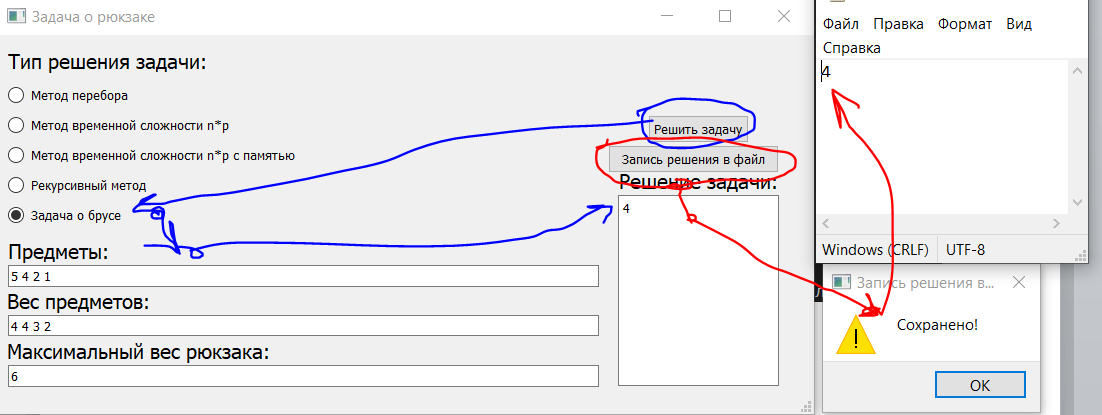
                        WeightItems[j] = NumbersItems[j]# Приравниваем текущий вес к текущему элементу

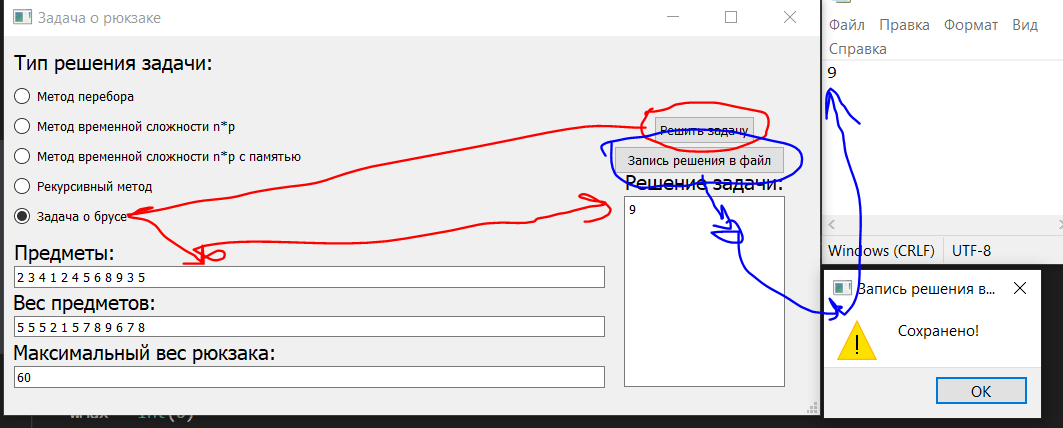
                        if(int(NumbersItems[j - int(WeightItems[i])])+int(WeightItems[i]) >int(WeightItems[j])): #Если текущий элемент + его текущий вес больше текущего веса, то

                            WeightItems[j] = int(NumbersItems[j-int(WeightItems[i])]) + int(WeightItems[i]) # Текущий вес будет равен текущему элементу – его вес + вес текущего элемента по столбцу.

                    for j in range(ItemsWeight):# Вывод результата

                        NumbersItems[j] = WeightItems[j]# Меняем местами вес и элемент

                self.ui.textResultEdit.setText(str(WeightItems[ItemsWeight - 1]))#Выводим результат



**Общий вывод:**

Данные задачи относятся к динамическому программированию, из-за того, для их решения нужно использовать циклические методы, рекурсии, также плюсом ко всему идёт всегда формирование динамических данных пользователем использование списков, массивов и специальных условий для их заполнения, а для этого опять нужно использовать методы, чтобы нагромождать код.