# 01背包问题

## the python code:

class Item():

def \_\_init\_\_(self,weight,value):

self.value=value

self.weight=weight

# test data:

items\_list=[Item(2,3),Item(3,4),Item(1,5),Item(5,6)]

# items\_list=[Item(2,3),Item(3,4),Item(1,5),Item(6,26)]

def knapsack01(n, w,items\_list):

"""[summary]

Args:

n (int): the number of the items to be choose （originally)

w (int): the max weight the knapsack could maintain

items\_list (list): list of items

Returns:

list: two dimesion list (the last element is the max value of the knapsack)

"""

k = [[0 for j in range(w+1)] for i in range(n+1)]

for i in range(1,n+1):

for j in range(1,w+1):

item=items\_list[i-1]

if j<item.weight:

k[i][j]=k[i-1][j]

else:

k[i][j]=max(k[i-1][j],k[i-1][j-item.weight]+item.value)

return k

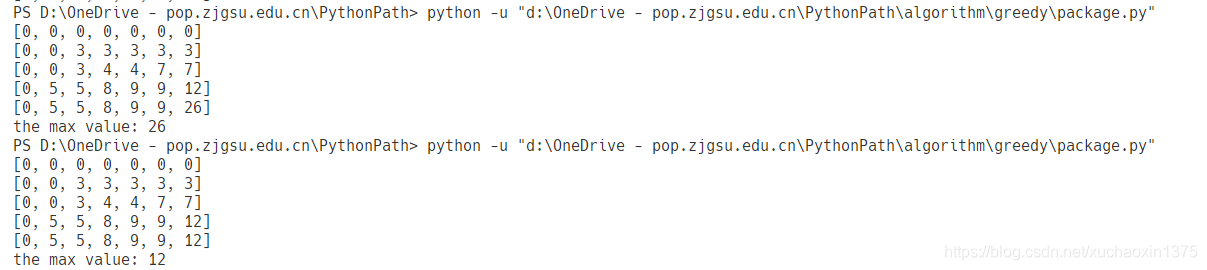
k=knapsack01(4,6,items\_list)

for line in k:

print(line)

print("the max value:",k[-1][-1])

## the executed result:



# Kruskal:

## python code：

""" 输入连通图的顶点的编号(顶点数书可以计算,也可以直接给出)和边(边数可以统计,也可以直接给出):

输出最小生成树所包含的各条边(用边上的两个顶点来表示)

输入连通网的边数：(可选)

6 10

输入连通网的顶点：(如果没有特殊要求,可以根据输入的点数直接默认是从1开始为顶点编号)

1

2

3

4

5

6

输入各边的起始点和终点及权重：(边上的顶点和之前的编号体系必须相对应,输入0结束边的录入)

1,2,6

1,3,1

1,4,5

2,3,5

2,5,3

3,4,5

3,5,6

3,6,4

4,6,2

5,6,6

0

"""

class Edge():

def \_\_init\_\_(self, start, end, weight):

self.start = start

self.end = end

self.weight = weight

class Node():

def \_\_init\_\_(self, number, sign):

self.number = number

self.sign = sign

def set\_sign(self, new\_sign):

self.sign = new\_sign

# get the nodes number(you can custom the number regularity,there use the default simple number system)

nodes = [Node(i, i) for i in range(1, 7)]

# get the edges parameters to instantiate the edge nodes ,put the edges to the list edges;

edges = []

while(True):

line = input("input node:")

if line == "0":

break

edge\_param = line.split(",")

start, end, weight = int(edge\_param[0]), int(

edge\_param[1]), int(edge\_param[2])

edges.append(Edge(start, end, weight))

#test: print(edges,node\_numbers)

# the number of the edges we need is the number\_of\_nodes-1:

number\_of\_nodes = len(nodes)

# sort the edges' weights

edges.sort(key=lambda edge: edge.weight)

#test: print([ edge.weight for edge in edges])

# count:count and judge whether we have complete the MST(Minimum Spanning Tree)

count = 0

# save the edges whice will be the edge of the MST

result\_edges = []

# select number\_of\_edges-1 edges to consist of the optimal solution

for edge in edges:

sign\_start = nodes[edge.start-1].sign

sign\_end = nodes[edge.end-1].sign

# the core code:judge the cricuit:

# by overlay the start node sign with the end node sign

# print(edge.end)

if sign\_start != sign\_end:

result\_edges.append(edge)

# traverse the all nodes:

for node in nodes:

# node there:

if node.sign == sign\_end:

node.sign = sign\_start

count += 1

if count == number\_of\_nodes-1:

break

else:

pass

# print(result\_edges)

print("the edges consist of the MST by Kruskal method:")

for edge in result\_edges:

edge\_tuple = edge.start, edge.end

print(edge\_tuple, "weight:", edge.weight)

## the result:

