1.COIN CHANGE USING GREEDY ALGORITHM

CODE:

def findMin(V):

deno=[1,2,5,10,20,50,100,500,1000]

n=len(deno)

ans=[]

i=n-1

while(i>=0):

while(V>=deno[i]):

V-=deno[i]

ans.append(deno[i])

i-=1

for i in range(len(ans)):

print(ans[i],end=" ")

if \_\_name\_\_=='\_\_main\_\_':

n=93

print("Following is minimal number","of change for",n,":",end="")

findMin(n)

2.KNAPSACK PROBLEM USING GREEDY ALGORITHM

CODE:

class Item:

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

self.profit = profit

self.weight = weight

def fractionalKnapsack(W, arr):

arr.sort(key=lambda x: (x.profit/x.weight), reverse=True)

finalvalue = 0.0

for item in arr:

if item.weight <= W:

W -= item.weight

finalvalue += item.profit

else:

finalvalue += item.profit \* W / item.weight

break

return finalvalue

if \_\_name\_\_ == "\_\_main\_\_":

W = 50

arr = [Item(60, 10), Item(100, 20), Item(120, 30)]

max\_val = fractionalKnapsack(W, arr)

print(max\_val)

3.JOB SEQUENCE USING GREEDY

CODE;

def printJobScheduling(arr, t):

n = len(arr)

for i in range(n):

for j in range(n - 1 - i):

if arr[j][2] < arr[j + 1][2]:

arr[j], arr[j + 1] = arr[j + 1], arr[j]

result = [False] \* t

job = ['-1'] \* t

for i in range(len(arr)):

for j in range(min(t - 1, arr[i][1] - 1), -1, -1):

if result[j] is False:

result[j] = True

job[j] = arr[i][0]

break

print(job)

if \_\_name\_\_ == '\_\_main\_\_':

arr = [['a', 2, 100],

['b', 1, 19],

['c', 2, 27],

['d', 1, 25],

['e', 3, 15]]

print("Following is maximum profit sequence of jobs")

printJobScheduling(arr, 3)

4.HUFFMAN CODING USING GREEDY

CODE:

import heapq

class node:

def \_\_init\_\_(self, freq, symbol, left=None, right=None):

self.freq = freq

self.symbol = symbol

self.left = left

self.right = right

self.huff = ''

def \_\_lt\_\_(self, nxt):

return self.freq < nxt.freq

def printNodes(node, val=''):

newVal = val + str(node.huff)

if(node.left):

printNodes(node.left, newVal)

if(node.right):

printNodes(node.right, newVal)

if(not node.left and not node.right):

print(f"{node.symbol} -> {newVal}")

chars = ['a', 'b', 'c', 'd', 'e', 'f']

freq = [5, 9, 12, 13, 16, 45]

nodes = []

for x in range(len(chars)):

heapq.heappush(nodes, node(freq[x], chars[x]))

while len(nodes) > 1:

left = heapq.heappop(nodes)

right = heapq.heappop(nodes)

left.huff = 0

right.huff = 1

newNode = node(left.freq+right.freq, left.symbol+right.symbol, left, right)

heapq.heappush(nodes, newNode)

printNodes(nodes[0])

5.DIJKARTS ALGORITHM USING GREEDY

CODE:

import sys

class Graph():

def \_\_init\_\_(self, vertices):

self.V = vertices

self.graph = [[0 for column in range(vertices)]

for row in range(vertices)]

def printSolution(self, dist):

print("Vertex \tDistance from Source")

for node in range(self.V):

print(node, "\t", dist[node])

def minDistance(self, dist, sptSet):

min = sys.maxsize

for u in range(self.V):

if dist[u] < min and sptSet[u] == False:

min = dist[u]

min\_index = u

return min\_index

def dijkstra(self, src):

dist = [sys.maxsize] \* self.V

dist[src] = 0

sptSet = [False] \* self.V

for cout in range(self.V):

x = self.minDistance(dist, sptSet)

sptSet[x] = True

for y in range(self.V):

if self.graph[x][y] > 0 and sptSet[y] == False and \

dist[y] > dist[x] + self.graph[x][y]:

dist[y] = dist[x] + self.graph[x][y]

self.printSolution(dist)

if \_\_name\_\_ == "\_\_main\_\_":

g = Graph(9)

g.graph = [[0, 4, 0, 0, 0, 0, 0, 8, 0],

[4, 0, 8, 0, 0, 0, 0, 11, 0],

[0, 8, 0, 7, 0, 4, 0, 0, 2],

[0, 0, 7, 0, 9, 14, 0, 0, 0],

[0, 0, 0, 9, 0, 10, 0, 0, 0],

[0, 0, 4, 14, 10, 0, 2, 0, 0],

[0, 0, 0, 0, 0, 2, 0, 1, 6],

[8, 11, 0, 0, 0, 0, 1, 0, 7],

[0, 0, 2, 0, 0, 0, 6, 7, 0]

]

g.dijkstra(0)

6.CONTAINER LOADING USING GREEDY

CODE:

cont = [[ 0 for i in range(1000)]

for j in range(1000)]

def num\_of\_containers(n, x):

count = 0

cont[1][1] = x

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

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

if (cont[i][j] >= 1):

count += 1

cont[i + 1][j] += (cont[i][j] - 1) / 2

cont[i + 1][j + 1] += (cont[i][j] - 1) / 2

print(count)

n = 3

x = 5

num\_of\_containers(n, x)

7.KRUSKALS ALGORITHM USING GGREEDY

CODE:

class Graph:

def \_\_init\_\_(self, vertices):

self.V = vertices

self.graph = []

def addEdge(self, u, v, w):

self.graph.append([u, v, w])

def find(self, parent, i):

if parent[i] != i:

parent[i] = self.find(parent, parent[i])

return parent[i]

def union(self, parent, rank, x, y):

if rank[x] < rank[y]:

parent[x] = y

elif rank[x] > rank[y]:

parent[y] = x

else:

parent[y] = x

rank[x] += 1

def KruskalMST(self):

result = []

i = 0

e = 0

self.graph = sorted(self.graph,

key=lambda item: item[2])

parent = []

rank = []

for node in range(self.V):

parent.append(node)

rank.append(0)

while e < self.V - 1:

u, v, w = self.graph[i]

i = i + 1

x = self.find(parent, u)

y = self.find(parent, v)

if x != y:

e = e + 1

result.append([u, v, w])

self.union(parent, rank, x, y)

minimumCost = 0

print("Edges in the constructed MST")

for u, v, weight in result:

minimumCost += weight

print("%d -- %d == %d" % (u, v, weight))

print("Minimum Spanning Tree", minimumCost)

if \_\_name\_\_ == '\_\_main\_\_':

g = Graph(4)

g.addEdge(0, 1, 10)

g.addEdge(0, 2, 6)

g.addEdge(0, 3, 5)

g.addEdge(1, 3, 15)

g.addEdge(2, 3, 4)

g.KruskalMST()

9. PRIMS ALGORITHM USING GREEDY

CODE:

import sys

class Graph():

def \_\_init\_\_(self, vertices):

self.V = vertices

self.graph = [[0 for column in range(vertices)]

def printMST(self, parent):

print("Edge \tWeight")

for i in range(1, self.V):

print(parent[i], "-", i, "\t", self.graph[i][parent[i]])

def minKey(self, key, mstSet):

min = sys.maxsize

for v in range(self.V):

if key[v] < min and mstSet[v] == False:

min = key[v]

min\_index = v

return min\_index

def primMST(self):

key = [sys.maxsize] \* self.V

parent = [None] \* self.V

key[0] = 0

mstSet = [False] \* self.V

parent[0] = -1

for cout in range(self.V):

u = self.minKey(key, mstSet)

mstSet[u] = True

for v in range(self.V):

if self.graph[u][v] > 0 and mstSet[v] == False \

and key[v] > self.graph[u][v]:

key[v] = self.graph[u][v]

parent[v] = u

self.printMST(parent)

if \_\_name\_\_ == '\_\_main\_\_':

g = Graph(5)

g.graph = [[0, 2, 0, 6, 0],

[2, 0, 3, 8, 5],

[0, 3, 0, 0, 7],

[6, 8, 0, 0, 9],

[0, 5, 7, 9, 0]]

g.primMST()

9.BORUVKAS ALGORITHM USING GREEDY

CODE:

from collections import defaultdict

class Graph:

def \_\_init\_\_(self,vertices):

self.V= vertices

self.graph = []

def addEdge(self,u,v,w):

self.graph.append([u,v,w])

def find(self, parent, i):

if parent[i] == i:

return i

return self.find(parent, parent[i])

def union(self, parent, rank, x, y):

xroot = self.find(parent, x)

yroot = self.find(parent, y)

if rank[xroot] < rank[yroot]:

parent[xroot] = yroot

elif rank[xroot] > rank[yroot]:

parent[yroot] = xroot

else :

parent[yroot] = xroot

rank[xroot] += 1

def boruvkaMST(self):

parent = []; rank = [];

cheapest =[]

numTrees = self.V

MSTweight = 0

for node in range(self.V):

parent.append(node)

rank.append(0)

cheapest =[-1] \* self.V

while numTrees > 1:

for i in range(len(self.graph)):

u,v,w = self.graph[i]

set1 = self.find(parent, u)

set2 = self.find(parent ,v)

if set1 != set2:

if cheapest[set1] == -1 or cheapest[set1][2] > w :

cheapest[set1] = [u,v,w]

if cheapest[set2] == -1 or cheapest[set2][2] > w :

cheapest[set2] = [u,v,w]

for node in range(self.V):

if cheapest[node] != -1:

u,v,w = cheapest[node]

set1 = self.find(parent, u)

set2 = self.find(parent ,v)

if set1 != set2 :

MSTweight += w

self.union(parent, rank, set1, set2)

print ("Edge %d-%d with weight %d included in MST" % (u,v,w))

numTrees = numTrees - 1

cheapest =[-1] \* self.V

print ("Weight of MST is %d" % MSTweight)

g = Graph(4)

g.addEdge(0, 1, 10)

g.addEdge(0, 2, 6)

g.addEdge(0, 3, 5)

g.addEdge(1, 3, 15)

g.addEdge(2, 3, 4)

g.boruvkaMST()