**Name:** Moin Tamboli

**Roll no.:** COTC64

**Practical No. 3**

**Source Code:**

# -------------------------#Selection Sort:----------------------

def selection\_sort(arr):

for i in range(len(arr)):

min\_index=i

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

if arr[j]<arr[min\_index]:

min\_index=j

arr[i], arr[min\_index]=arr[min\_index], arr[i]

return arr

a1=[20,10,5,7,9,13]

selection\_sort(a1)

# -------------------#Prims algorithm------------------

import heapq

def prim(graph, start):

mst=[]

visited=set([start])

edges=[(cost,start,to) for to, cost in graph[start].items()]

heapq.heapify(edges)

while edges :

cost, frm, to = heapq.heappop(edges)

if to not in visited:

visited.add(to)

mst.append((frm,to, cost))

for to\_next, cost2 in graph [to].items():

if to\_next not in visited:

heapq.heappush(edges,(cost2, to, to\_next))

return mst

graph={

'A':{'B':2, 'C':3},

'B':{'A':2, 'C':1, 'D':1},

'C':{'A':3, 'B':1, 'D':4},

'D':{'B':1, 'C':4}

}

print(prim(graph, 'A'))

# -----------------------#Krushkals Algorithm:----------------------------

def krushkal(graph):

mst=[]

edges=[(cost, frm, to) for frm, to\_dict in graph.items() for to, cost in to\_dict.items()]

edges.sort()

parent={node: node for node in graph}

def find\_root(node):

if parent[node]==node:

return node

parent[node]=find\_root(parent[node])

return parent[node]

for cost, frm, to in edges:

root1=find\_root(frm)

root2=find\_root(to)

if root1 != root2:

parent[root1]=root2

mst.append((frm, to, cost))

return mst

graph={

'A':{'B':2, 'C':3},

'B':{'A':2, 'C':1, 'D':1},

'C':{'A':3, 'B':1, 'D':4},

'D':{'B':1, 'C':4}

}

print(krushkal(graph))

# -----------------------#Job scheduling Algorithm:----------------------------

def printJobScheduling(arr, t):

# length of array

n = len(arr)

# Sort all jobs according to

# decreasing order of profit

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]

# To keep track of free time slots

result = [False] \* t

# To store result (Sequence of jobs)

job = ['-1'] \* t

# Iterate through all given jobs

for i in range(len(arr)):

# Find a free slot for this job

# (Note that we start from the

# last possible slot)

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

# Free slot found

if result[j] is False:

result[j] = True

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

break

# print the sequence

print(job)

# Driver's Code

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

arr = [['a', 2, 100], # Job Array

['b', 1, 19],

['c', 2, 27],

['d', 1, 25],

['e', 3, 15]]

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

# Function Call

printJobScheduling(arr, 3)

**Output:**

[5, 7, 9, 10, 13, 20]

[('A', 'B', 2), ('B', 'C', 1), ('B', 'D', 1)]

[('B', 'C', 1), ('B', 'D', 1), ('A', 'B', 2)]

Following is maximum profit sequence of jobs

['c', 'a', 'e']