**Problem Statement –** Implement Greedy search algorithm for any of the following application:

I. Selection Sort

П. Minimum Spanning Tree

III.Single-Source Shortest Path Problem

IV. Job Scheduling Problem

V. Prim's Minimal Spanning Tree Algorithm

VI. Kruskal's Minimal Spanning Tree Algorithm

VII. Dijkstra's Minimal Spanning Tree Algorithm

import sys, heapq

from collections import defaultdict

from math import inf

def selectionSort(A):

U = A.copy()

for i in range(len(A)):

min\_idx = i

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

if A[min\_idx] > A[j]:

min\_idx = j

A[i], A[min\_idx] = A[min\_idx], A[i]

print(f'Selection Sort:\nUnsorted array: {U}\nSorted array: {A}')

def jobScheduling(arr):

n = len(arr)

arr.sort(key=lambda x: x[1])

result = []

maxHeap = []

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

if i == 0:

slots\_available = arr[i][1]

else:

slots\_available = arr[i][1] - arr[i - 1][1]

heapq.heappush(maxHeap, (-arr[i][2], arr[i][1], arr[i][0]))

while slots\_available and maxHeap:

profit, deadline, job\_id = heapq.heappop(maxHeap)

slots\_available -= 1

result.append([job\_id, deadline])

result.sort(key=lambda x: x[1])

print(f'\n\nJob Scheduling Problem:\nFollowing is maximum profit sequence of jobs: {result}')

class PGraph:

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

self.V = vertices

self.graph = graph

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

print(f'\n\nPrim’s Minimum Spanning Tree:\nEdge \tWeight')

minimumCost = 0

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

print(f'{parent[i]} -- {i} == {self.graph[i][parent[i]]}')

minimumCost += self.graph[i][parent[i]]

print(f'Minimum cost = {minimumCost}')

class KGraph:

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

self.V = vertices

self.graph = []

for i in range(self.V):

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

if graph[i][j] != 0:

self.graph.append([i, j, graph[i][j]])

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 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(f'\n\nKruskal’s Minimum Spanning Tree:\nEdge \tWeight')

for u, v, weight in result:

minimumCost += weight

print(f'{u} -- {v} == {weight}')

print(f'Minimum cost = {minimumCost}')

def dijkstra(inp: list, source: str, dest: str) -> int:

graph = defaultdict(list)

weight = {}

path = [dest]

for i in range(len(inp)):

for j in range(i, len(inp)):

if inp[i][j] != 0:

s, d, c = i, j, inp[i][j]

graph[s].append(d)

weight[f'{s} {d}'] = int(c)

graph[d].append(s)

weight[f'{d} {s}'] = int(c)

Q = list(graph.keys())

A, d, p = [], {}, defaultdict(list)

for v in Q:

d[v] = inf

p[v] = []

d[source] = 0

while Q:

u = min(Q, key=lambda x: d[x])

A.append(u)

Q.remove(u)

for v in set(graph[u]).intersection(Q):

alt = d[u] + weight[f'{u} {v}']

if d[v] > alt:

d[v] = alt

p[v].append(u)

if u == dest:

break

key = dest

while p[key]:

path.append(p[key][-1])

key = p[key][-1]

path.reverse()

print(f'\n\nDijkstra Single-Source Shortest Path:\nPath: {path}\nMinimum Cost: {d[dest]}\n\n')

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

A = [64, 25, 12, 22, 11]

selectionSort(A)

A = [

['A', 2, 100],

['B', 1, 19],

['C', 2, 27],

['D', 1, 25],

['E', 3, 15]

]

jobScheduling(A)

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 = PGraph(9, graph)

g.primMST()

g = KGraph(9, graph)

g.KruskalMST()

dijkstra(graph, 0, 4)

Output:-

Selection Sort:

Unsorted array: [64, 25, 12, 22, 11]

Sorted array: [11, 12, 22, 25, 64]

Job Scheduling Problem:

Following is maximum profit sequence of jobs: [['A', 2], ['C', 2], ['E', 3]]

Prim’s Minimum Spanning Tree:

Edge Weight

0 -- 1 == 4

1 -- 2 == 8

2 -- 3 == 7

3 -- 4 == 9

2 -- 5 == 4

5 -- 6 == 2

6 -- 7 == 1

2 -- 8 == 2

Minimum cost = 37

Kruskal’s Minimum Spanning Tree:

Edge Weight

6 -- 7 == 1

2 -- 8 == 2

5 -- 6 == 2

0 -- 1 == 4

2 -- 5 == 4

2 -- 3 == 7

0 -- 7 == 8

3 -- 4 == 9

Minimum cost = 37

Dijkstra Single-Source Shortest Path:

Path: [0, 7, 6, 5, 4]

Minimum Cost: 21