AI 3: GREEDY SEARCH ALGORITHMS

Assign 3A : SELECTION SORT ALGORITHM

```
def selectionSort( itemsList ):
    n = len( itemsList )
    for i in range( n - 1 ):
        minValueIndex = i

        for j in range( i + 1, n ):
            if itemsList[j] < itemsList[minValueIndex] :
                minValueIndex = j

        if minValueIndex != i :
            temp = itemsList[i]
            itemsList[i] = itemsList[minValueIndex]
            itemsList[minValueIndex] = temp

    return itemsList



el = [21,6,9,33,3]

print(selectionSort(el))
```

n Online Compiler

ain.py

Share Run

```python
# Online Python compiler (interpreter) to run Python online.
def selectionSort( itemsList ):
    n = len( itemsList )
    for i in range( n - 1 ):
        minValueIndex = i

        for j in range( i + 1, n ):
            if itemsList[j] < itemsList[minValueIndex] :
                minValueIndex = j

        if minValueIndex != i :
            temp = itemsList[i]
            itemsList[i] = itemsList[minValueIndex]
            itemsList[minValueIndex] = temp

    return itemsList


el = [21,6,9,33,3]

print(selectionSort(el))
```

Output

```
[3, 6, 9, 21, 33]

=== Code Execution Successful ===
```

AI 3: GREEDY SEARCH ALGORITHMS

Assign 3B

===

2. Single Source Shortest Path (Dijkstra's Algorithm)

==

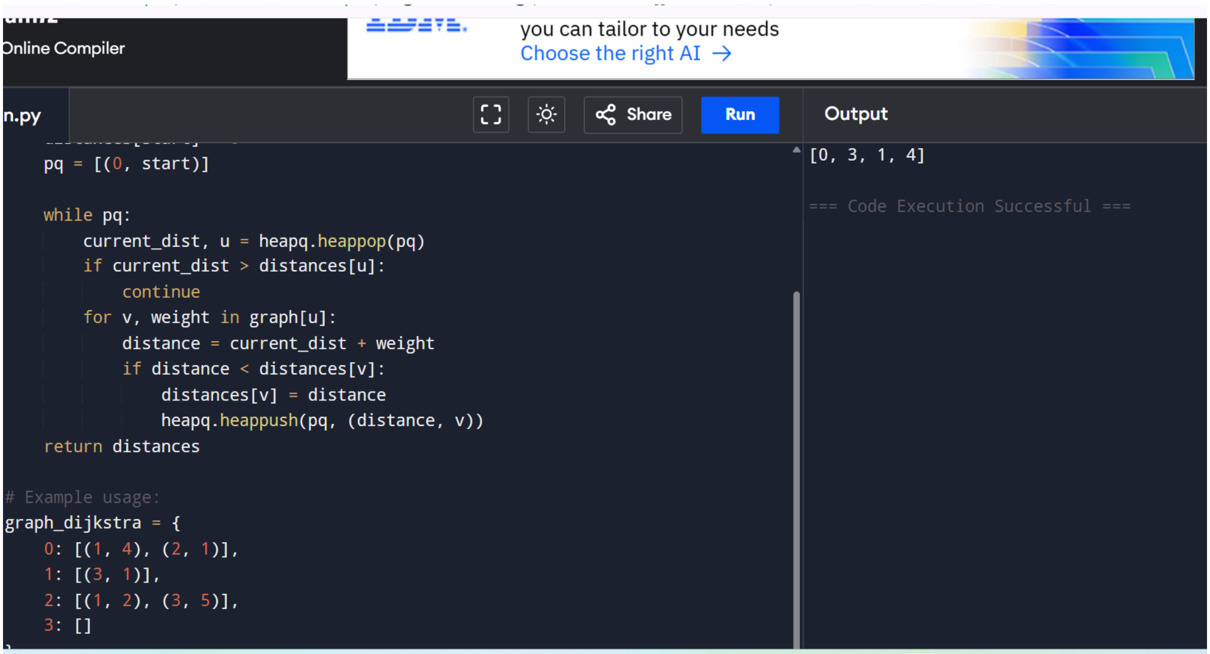
```
# Dijkstra's Algorithm
import heapq

def dijkstra(graph, start):
    n = len(graph)
    distances = [float('inf')] * n
    distances[start] = 0
    pq = [(0, start)]

    while pq:
        current_dist, u = heapq.heappop(pq)
        if current_dist > distances[u]:
            continue
        for v, weight in graph[u]:
            distance = current_dist + weight
            if distance < distances[v]:
                distances[v] = distance
                heapq.heappush(pq, (distance, v))
    return distances

# Example usage:
graph_dijkstra = {
    0: [(1, 4), (2, 1)],
    1: [(3, 1)],
```

```
    2: [(1, 2), (3, 5)],
    3: []
}
start_node = 0
distances = dijkstra(graph_dijkstra, start_node)
print(distances)
```



AI 3: GREEDY SEARCH ALGORITHMS

Assign 3C

===

3. Kruskal’s Minimum Spanning Tree

===

```
# Kruskal's Algorithm
class DisjointSet:
    def __init__(self, n):
        self.parent = list(range(n))

    def find(self, x):
        if self.parent[x] != x:
            self.parent[x] = self.find(self.parent[x])
        return self.parent[x]

    def union(self, x, y):
        x_root = self.find(x)
        y_root = self.find(y)
        if x_root != y_root:
            self.parent[x_root] = y_root
            return True
        return False

def kruskal(n, edges):
    edges.sort(key=lambda x: x[2])
    ds = DisjointSet(n)
    mst = []
    total_weight = 0

    for u, v, weight in edges:
```

```
        if ds.union(u, v):
            mst.append((u, v, weight))
            total_weight += weight

    return mst, total_weight


# Example usage:
edges_kruskal = [
    (0, 1, 10),
    (0, 2, 6),
    (0, 3, 5),
    (1, 3, 15),
    (2, 3, 4)
]
mst_k, weight_k = kruskal(4, edges_kruskal)
print(mst_k, weight_k)
```

AI 3: GREEDY SEARCH ALGORITHMS

Assign 3D

===

4. Prim’s Minimum Spanning Tree

===

```
# Prim's Algorithm
def prim(graph):
    n = len(graph)
    visited = [False] * n
    min_heap = [(0, 0)]
    total_cost = 0
    mst = []

    while min_heap:
        weight, u = heapq.heappop(min_heap)
        if visited[u]:
            continue
        visited[u] = True
        total_cost += weight
        for v, w in graph[u]:
            if not visited[v]:
                heapq.heappush(min_heap, (w, v))
                mst.append((u, v, w))
    return mst, total_cost

# Example usage:
graph_prim = {
    0: [(1, 10), (2, 6), (3, 5)],
    1: [(0, 10), (3, 15)],
```

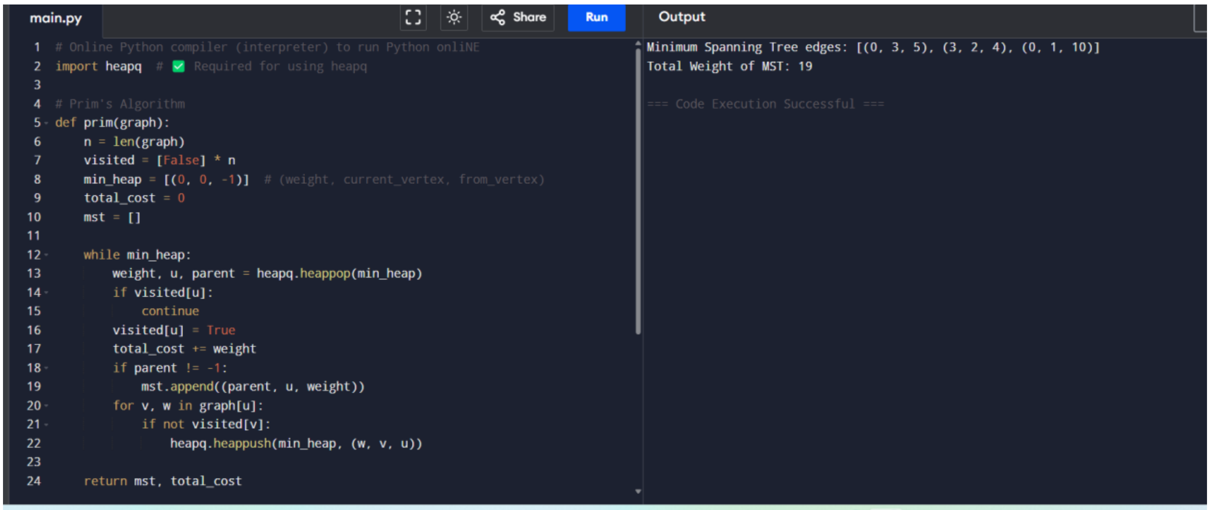
2: [(0, 6), (3, 4)],

3: [(0, 5), (1, 15), (2, 4)]

}

mst_p, weight_p = prim(graph_prim)

print(mst_p, weight_p)




```
# Online Python compiler (interpreter) to run Python onliNE
import heapq  # ✅ Required for using heapq

# Prim's Algorithm
def prim(graph):
    n = len(graph)
    visited = [False] * n
    min_heap = [(0, 0, -1)]  # (weight, current_vertex, from_vertex)
    total_cost = 0
    mst = []

    while min_heap:
        weight, u, parent = heapq.heappop(min_heap)
        if visited[u]:
            continue
        visited[u] = True
        total_cost += weight
        if parent != -1:
            mst.append((parent, u, weight))
        for v, w in graph[u]:
            if not visited[v]:
                heapq.heappush(min_heap, (w, v, u))

    return mst, total_cost
```

```
Minimum Spanning Tree edges: [(0, 3, 5), (3, 2, 4), (0, 1, 10)]
Total Weight of MST: 19

=== Code Execution Successful ===
```

AI 3: GREEDY SEARCH ALGORITHMS

Assign 3E

==

5. Job Scheduling Problem

==

```
# Job Scheduling Problem
class Job:
    def __init__(self, job_id, deadline, profit):
        self.id = job_id
        self.deadline = deadline
        self.profit = profit

def job_scheduling(jobs):
    jobs.sort(key=lambda x: x.profit, reverse=True)
    n = max(job.deadline for job in jobs)
    result = [None] * n
    slot = [False] * n
    total_profit = 0
    job_sequence = []

    for job in jobs:
        for j in range(min(n, job.deadline) - 1, -1, -1):
            if not slot[j]:
                slot[j] = True
                result[j] = job.id
                total_profit += job.profit
                job_sequence.append(job.id)
                break
```

```
    return job_sequence, total_profit


# Example usage:
jobs_list = [
    Job('J1', 2, 100),
    Job('J2', 1, 19),
    Job('J3', 2, 27),
    Job('J4', 1, 25),
    Job('J5', 3, 15)
]
seq, prof = job_scheduling(jobs_list)
print(seq, prof)
```