LAB-11

Assembly Line Scheduling

CODE:

```
def fun(a, t, cl, cs, x1, x2, n):
        if cs == n - 1:
                if cl == 0:
                        return x1
                else:
                        return x2
        same = fun(a, t, cl, cs + 1, x1, x2, n) + a[cl][cs + 1]
        diff = fun(a, t, not cl, cs + 1, x1, x2, n) + a[not cl][cs + 1] + t[cl][cs + 1]
        return min(same, diff)
n = 4
a = [[4, 5, 3, 2], [2, 10, 1, 4]]
t = [[0, 7, 4, 5], [0, 9, 2, 8]]
e1 = 10
e2 = 12
x1 = 18
x2 = 7
x = fun(a, t, 0, 0, x1, x2, n) + e1 + a[0][0]
y = fun(a, t, 1, 0, x1, x2, n) + e2 + a[1][0]
print(min(x, y))
```

OUTPUT:

```
>>>
= RESTART: C:/Users/bored/AppData/Local/Programs/Python/Python312/assembly line
scheduling using dp.py
35
>>>
```

Knapsack problem and Memory

CODE:

```
def knapsack(wt, val, W, n):
        if n == 0 or W == 0:
                return 0
```

```
        if t[n][W] != -1:
                return t[n][W]
        if wt[n-1] <= W:
                t[n][W] = max(
                        val[n-1] + knapsack(
                                wt, val, W-wt[n-1], n-1),
                        knapsack(wt, val, W, n-1))
                return t[n][W]
        elif wt[n-1] > W:
                t[n][W] = knapsack(wt, val, W, n-1)
                return t[n][W]
if __name__ == '__main__':
        profit = [60, 100, 120]
        weight = [10, 20, 30]
        W = 50
        n = len(profit)
        t = [[-1 for i in range(W + 1)] for j in range(n + 1)]
        print(knapsack(weight, profit, W, n))
```

OUTPUT:

```
>>>
    = RESTART: C:/Users/bored/AppData/Local/Programs/Python/Python312/Knapsack probl
    em and Memory using dp.py
    220
>>>
```

Bellman-Ford Algorithm

CODE:

```
def bellman_ford(graph, source):
    distances = {vertex: float('inf') for vertex in graph}
    distances[source] = 0
    for _ in range(len(graph) - 1):
        for u in graph:
            for v, weight in graph[u].items():
```

```
            if distances[u] != float('inf') and distances[u] + weight < distances[v]:
                distances[v] = distances[u] + weight
    for u in graph:
        for v, weight in graph[u].items():
            if distances[u] != float('inf') and distances[u] + weight < distances[v]:
                raise ValueError("Graph contains negative weight cycle")
    return distances
graph = {
    'A': {'B': -1, 'C': 4},
    'B': {'C': 3, 'D': 2, 'E': 2},
    'C': {},
    'D': {'B': 1, 'C': 5},
    'E': {'D': -3}
}
source = 'A'
shortest_distances = bellman_ford(graph, source)
print(shortest_distances)
```

OUTPUT:

```
>>>
    = RESTART: C:/Users/bored/AppData/Local/Programs/Python/Python312/bellman ford.p
    y
    {'A': 0, 'B': -1, 'C': 2, 'D': -2, 'E': 1}
>>>
```

Warshall's & Floyd's Algorithm

CODE:

```
nV = 4
INF = 999
def floyd_warshall(G):
    distance = list(map(lambda i: list(map(lambda j: j, i)), G))
    for k in range(nV):
        for i in range(nV):
            for j in range(nV):
```

```
            distance[i][j] = min(distance[i][j], distance[i][k] + distance[k][j])
    print_solution(distance)
def print_solution(distance):
    for i in range(nV):
        for j in range(nV):
            if(distance[i][j] == INF):
                print("INF", end=" ")
            else:
                print(distance[i][j], end="  ")
        print(" ")
G = [[0, 3, INF, 5],
        [2, 0, INF, 4],
        [INF, 1, 0, INF],
        [INF, INF, 2, 0]]
floyd_warshall(G)
```

OUTPUT:

```
>>>
    = RESTART: C:/Users/bored/AppData/Local/Programs/Python/Python312/Warshall's & F
    loyd's Algorithm.py
    0   3   7   5
    2   0   6   4
    3   1   0   5
    5   3   2   0
>>>
```