LAB-9

DATE:18/06/2024

1. Assembly Line Scheduling

Coding:

```
1 usage
def schedule(tasks, n, m):
    dp = [[0 for _ in range(m + 1)] for _ in range(n + 1)]
    for i in range(1, n + 1):
        for j in range(1, m + 1):
            if tasks[i - 1] <= j:
                dp[i][j] = max(tasks[i - 1] + dp[i - 1][j - tasks[i - 1]], dp[i - 1][j])
            else:
                dp[i][j] = dp[i - 1][j]
    return dp[n][m]
tasks = [1, 2, 3, 4, 5]
n = len(tasks)
m = 7
print(schedule(tasks, n, m))
```

Output:

```
7

Process finished with exit code 0
```

2. Knapsack problem and Memory

Coding:

```
1 usage
def knapsack(weights, values, capacity, n):
    memo = [[-1 for _ in range(capacity + 1)] for _ in range(n + 1)]
    def dp(i, w):
        if memo[i][w] != -1:
            return memo[i][w]
        if i == 0 or w == 0:
            result = 0
        elif weights[i - 1] > w:
            result = dp(i - 1, w)
        else:
            include = values[i - 1] + dp(i - 1, w - weights[i - 1])
            exclude = dp(i - 1, w)
            result = max(include, exclude)
        memo[i][w] = result
        return result
    return dp(n, capacity)
```

```
        return dp(n, capacity)
    weights = [1, 2, 4, 2, 5]
    values = [5, 3, 5, 3, 2]
    capacity = 10
    n = len(values)
    print(knapsack(weights, values, capacity, n))
```

Output:

```
16

Process finished with exit code 0
```

3. Bellman-Ford Algorithm

Coding:

```
def bellman_ford(graph, source):
    distance = {node: float('inf') for node in graph}
    distance[source] = 0
    for _ in range(len(graph) - 1):
        for node in graph:
            for neighbor, weight in graph[node].items():
                distance[neighbor] = min(distance[neighbor], distance[node] + weight)
    for node in graph:
        for neighbor, weight in graph[node].items():
            assert distance[neighbor] <= distance[node] + weight, "Negative-weight cycle detected"
    return distance
graph = { 'A': {'B': -1, 'C':  4}, 'B': {'C':  3, 'D':  2, 'E':  2}, 'C': {},
    'D': {'B':  1, 'C':  5},
    'E': {'D': -3}}
source = 'A'
print(bellman_ford(graph, source))
```

Output:

```
{'A': 0, 'B': -1, 'C': 2, 'D': -2, 'E': 1}

Process finished with exit code 0
```

4. Warshall's & Floyd's Algorithm

Coding:

```python
def all_pairs_shortest_path(graph):
    n = len(graph)
    dp = [[float('inf') for _ in range(n)] for _ in range(n)]
    for i in range(n):
        for j in range(n):
            dp[i][j] = graph[i][j]
    for k in range(n):
        for i in range(n):
            for j in range(n):
                dp[i][j] = min(dp[i][j], dp[i][k] + dp[k][j])
    return dp
graph = [
    [0, 5, float('inf'), 10],
    [float('inf'), 0, 3, float('inf')],
    [float('inf'), float('inf'), 0, 1],
    [float('inf'), float('inf'), float('inf'), 0]
]
result = all_pairs_shortest_path(graph)
for i in range(len(result)):
    for j in range(len(result[0])):
        if result[i][j] == float('inf'):
            print("*", end=" ")
        else:
            print(result[i][j], end=" ")
    print()
```

Output:

```
0 5 8 9
* 0 3 4
* * 0 1
* * * 0

Process finished with exit code 0
```