LAB-13

Word break

CODE:

```
def wordBreak(s, wordDict):
    word_set = set(wordDict)
    dp = [False] * (len(s) + 1)
    dp[0] = True
    for i in range(1, len(s) + 1):
        for j in range(i):
            if dp[j] and s[j:i] in word_set:
                dp[i] = True
                break
    return dp[len(s)]
s = "leetcode"
wordDict = ["leet", "code"]
print(wordBreak(s, wordDict))
```

OUTPUT:

```
>>>
    = RESTART: C:/Users/bored/AppData/Local/Programs/Python/Python312/word break usi
    ng dp.py
    True
>>>
```

Assembly line scheduling with 3 lines

CODE:

```
def assembly_line_scheduling(line1, line2, line3, t1, t2, t3, t4, t5, e1, e2, e3, x1, x2, x3):
    n = len(line1)
    f = [[0] * (n + 1) for _ in range(4)]
    g = [[0] * (n + 1) for _ in range(4)]
    h = [[0] * (n + 1) for _ in range(4)]
    f[1][1] = e1 + line1[0]
    g[1][1] = e2 + line2[0]
    h[1][1] = e3 + line3[0]
    for j in range(2, n + 1):
```

```
        f[1][j] = line1[j-1] + min(f[1][j-1], g[1][j-1] + t1[j-2], h[1][j-1] + t2[j-2])
        g[1][j] = line2[j-1] + min(g[1][j-1], f[1][j-1] + t1[j-2], h[1][j-1] + t3[j-2])
        h[1][j] = line3[j-1] + min(h[1][j-1], f[1][j-1] + t2[j-2], g[1][j-1] + t3[j-2])
        f[2][j] = line1[j-1] + min(f[2][j-1], g[2][j-1] + t4[j-2], h[2][j-1] + t5[j-2])
        g[2][j] = line2[j-1] + min(g[2][j-1], f[2][j-1] + t4[j-2], h[2][j-1] + t2[j-2])
        h[2][j] = line3[j-1] + min(h[2][j-1], f[2][j-1] + t5[j-2], g[2][j-1] + t2[j-2])
        f[3][j] = line1[j-1] + min(f[3][j-1], g[3][j-1] + t3[j-2], h[3][j-1] + t4[j-2])
        g[3][j] = line2[j-1] + min(g[3][j-1], f[3][j-1] + t3[j-2], h[3][j-1] + t5[j-2])
        h[3][j] = line3[j-1] + min(h[3][j-1], f[3][j-1] + t4[j-2], g[3][j-1] + t5[j-2])
    f_exit = f[1][n] + x1
    g_exit = g[2][n] + x2
    h_exit = h[3][n] + x3
    min_time = min(f_exit, g_exit, h_exit)
    return min_time
line1 = [7, 9, 3, 4]
line2 = [8, 5, 6, 4]
line3 = [3, 6, 7, 2]
t1 = [2, 3, 1]
t2 = [2, 1, 2]
t3 = [3, 2, 1]
t4 = [1, 2, 1]
t5 = [2, 1, 3]
e1 = 2
e2 = 3
e3 = 1
x1 = 3
x2 = 2
x3 = 3
min_time = assembly_line_scheduling(line1, line2, line3, t1, t2, t3, t4, t5, e1, e2, e3, x1, x2, x3)
print("Minimum assembly time:", min_time)
```

OUTPUT:

```
>>>
= RESTART: C:/Users/bored/AppData/Local/Programs/Python/Python312/assembly line
scheduling with 3 lines using dp.py
Minimum assembly time: 16
>>>
```

Prims, kruskal's, boruvka's mst

CODE:

```
import heapq
from collections import defaultdict
def prims_algorithm(graph):
    num_nodes = len(graph)
    mst_set = set()
    min_heap = [(0, 0)]
    total_weight = 0
    mst_edges = []
    while len(mst_set) < num_nodes:
        weight, node = heapq.heappop(min_heap)
        if node in mst_set:
            continue
        mst_set.add(node)
        total_weight += weight
        for adjacent, edge_weight in enumerate(graph[node]):
            if edge_weight > 0 and adjacent not in mst_set:
                heapq.heappush(min_heap, (edge_weight, adjacent))
                if weight > 0:
                    mst_edges.append((node, adjacent, edge_weight))
    return total_weight, mst_edges
class UnionFind:
    def _init_(self, n):
        self.parent = list(range(n))
        self.rank = [0] * n
    def find(self, node):
        if self.parent[node] != node:
```

```
            self.parent[node] = self.find(self.parent[node])
        return self.parent[node]
    def union(self, node1, node2):
        root1 = self.find(node1)
        root2 = self.find(node2)
        if root1 != root2:
            if self.rank[root1] > self.rank[root2]:
                self.parent[root2] = root1
            elif self.rank[root1] < self.rank[root2]:
                self.parent[root1] = root2
            else:
                self.parent[root2] = root1
                self.rank[root1] += 1
def kruskals_algorithm(num_nodes, edges):
    uf = UnionFind(num_nodes)
    mst_edges = []
    total_weight = 0
    edges.sort(key=lambda x: x[2])
    for node1, node2, weight in edges:
        if uf.find(node1) != uf.find(node2):
            uf.union(node1, node2)
            mst_edges.append((node1, node2, weight))
            total_weight += weight
    return total_weight, mst_edges
def boruvkas_algorithm(num_nodes, edges):
    uf = UnionFind(num_nodes)
    mst_edges = []
    total_weight = 0
    num_components = num_nodes
    while num_components > 1:
        cheapest = [-1] * num_nodes
```

```
        for node1, node2, weight in edges:
            root1 = uf.find(node1)
            root2 = uf.find(node2)
            if root1 != root2:
                if cheapest[root1] == -1 or cheapest[root1][2] > weight:
                    cheapest[root1] = (node1, node2, weight)
                if cheapest[root2] == -1 or cheapest[root2][2] > weight:
                    cheapest[root2] = (node1, node2, weight)
        for node in range(num_nodes):
            if cheapest[node] != -1:
                node1, node2, weight = cheapest[node]
                if uf.find(node1) != uf.find(node2):
                    uf.union(node1, node2)
                    mst_edges.append((node1, node2, weight))
                    total_weight += weight
                    num_components -= 1
    return total_weight, mst_edges
graph_adj_matrix = [
    [0, 2, 0, 6, 0],
    [2, 0, 3, 8, 5],
    [0, 3, 0, 0, 7],
    [6, 8, 0, 0, 9],
    [0, 5, 7, 9, 0]
]
edges_list = [
    (0, 1, 2),
    (0, 3, 6),
    (1, 2, 3),
    (1, 3, 8),
    (1, 4, 5),
    (2, 4, 7),
```

```
    (3, 4, 9)
]
total_weight_prims, mst_edges_prims = prims_algorithm(graph_adj_matrix)
print("Prim's Algorithm:")
print(f"Total weight of MST: {total_weight_prims}")
print("Edges in the MST:")
for edge in mst_edges_prims:
    print(edge)
total_weight_kruskals, mst_edges_kruskals = kruskals_algorithm(len(graph_adj_matrix), edges_list)
print("\nKruskal's Algorithm:")
print(f"Total weight of MST: {total_weight_kruskals}")
print("Edges in the MST:")
for edge in mst_edges_kruskals:
    print(edge)
total_weight_boruvkas, mst_edges_boruvkas = boruvkas_algorithm(len(graph_adj_matrix),
edges_list)
print("\nBorůvka's Algorithm:")
print(f"Total weight of MST: {total_weight_boruvkas}")
print("Edges in the MST:")
for edge in mst_edges_boruvkas:
    print(edge)
```

OUTPUT:

```
>>>
= RESTART: C:/Users/king of lenovo/Documents/DAA LAB EXPERMENTS/all prms borucks
 and kruskals algorithm.py
Prim's Algorithm:
Total weight of MST: 16
Edges in the MST:
(1, 2, 3)
(1, 3, 8)
(1, 4, 5)
(2, 4, 7)
(4, 3, 9)

Kruskal's Algorithm:
Total weight of MST: 16
Edges in the MST:
(0, 1, 2)
(1, 2, 3)
(1, 4, 5)
(0, 3, 6)

Borůvka's Algorithm:
Total weight of MST: 16
Edges in the MST:
(0, 1, 2)
(1, 2, 3)
(0, 3, 6)
(1, 4, 5)
>>>
```