LAB-10

Date:20/6/2024

1. word break

Coding:

```
1 usage
def wbreak(s,wdict):
    wset = set(wdict)
    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 wset:
                dp[i] = True
                break
    return dp
s = "moonaskedwhy"
wdict= ["moon","asked", "why"]
print(wbreak(s,wdict))
```

Output:

```
[True, False, False, False, True, False, False, False, False, True, False, False, True]

Process finished with exit code 0
```

2.assembly line

Coding:

```
def assemblyLineScheduling(a, t, e, x):
    n = len(a[0])
    T1 = [0] * n
    T2 = [0] * n
    T3 = [0] * n
    T1[0] = e[0] + a[0][0]
    T2[0] = e[1] + a[1][0]
    T3[0] = e[2] + a[2][0]
    for i in range(1, n):
        T1[i] = min(T1[i-1] + a[0][i], T2[i-1] + t[1][i] + a[0][i], T3[i-1] + t[2][i] + a[0][i])
        T2[i] = min(T2[i-1] + a[1][i], T1[i-1] + t[0][i] + a[1][i], T3[i-1] + t[2][i] + a[1][i])
        T3[i] = min(T3[i-1] + a[2][i], T1[i-1] + t[0][i] + a[2][i], T2[i-1] + t[1][i] + a[2][i])
    final_time = min(T1[n-1] + x[0], T2[n-1] + x[1], T3[n-1] + x[2])
    return final_time
a = [[1,1,1,1],
     [2,1,2,1],
```

```
    a = [[1,1,1,1],
         [2,1,2,1],
         [3,2,1,2]]
    t = [[0,2,1,3],
         [0,3,5,6],
         [0,4,3,1]]
    e = [10,10,10]
    x = [18, 7, 11]
    print(assemblyLineScheduling(a,t,e,x))
```

Output:

```
23

Process finished with exit code 0
```

3.mst using 3 algorithms:

Coding:

```
import heapq

class UnionFind:
   def __init__(self, n):  # Fix: __init__ instead of _init_
      self.parent = list(range(n))
      self.rank = [1] * n

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

```
    def union(self, u, v):
        rootU = self.find(u)
        rootV = self.find(v)
        if rootU != rootV:
            if self.rank[rootU] > self.rank[rootV]:
                self.parent[rootV] = rootU
            elif self.rank[rootU] < self.rank[rootV]:
                self.parent[rootU] = rootV
            else:
                self.parent[rootV] = rootU
                self.rank[rootU] += 1

def prims_algorithm(graph, start_vertex):
    mst = []
    visited = set()
    min_heap = [(0, start_vertex, None)]  # (weight, vertex, parent)
    total_cost = 0
    while min_heap:
        weight, u, parent = heapq.heappop(min_heap)
        if u not in visited:
            visited.add(u)
            if parent is not None:
                mst.append((parent, u, weight))
                total_cost += weight
            for v, w in graph[u]:
                if v not in visited:
                    heapq.heappush(min_heap, (w, v, u))
    return mst, total_cost

def kruskals_algorithm(graph, num_vertices):
    edges = []
    for u in graph:
        for v, w in graph[u]:
            edges.append((w, u, v))
    edges.sort()
    uf = UnionFind(num_vertices)
    mst = []
    total_cost = 0
    for weight, u, v in edges:
        if uf.find(u) != uf.find(v):
            uf.union(u, v)
            mst.append((u, v, weight))
            total_cost += weight
    return mst, total_cost

def boruvkas_algorithm(graph, num_vertices):
    uf = UnionFind(num_vertices)
    mst = []
    total_cost = 0
    num_components = num_vertices
    while num_components > 1:
        cheapest = [-1] * num_vertices
        for u in graph:
            for v, w in graph[u]:
                u_root = uf.find(u)
                v_root = uf.find(v)
                if u_root != v_root:
```

```
            if cheapest[u_root] == -1 or cheapest[u_root][0] > w:
                cheapest[u_root] = (w, u, v)
            if cheapest[v_root] == -1 or cheapest[v_root][0] > w:
                cheapest[v_root] = (w, v, u)
    for u in range(num_vertices):
        if cheapest[u]!= -1:
            w, u, v = cheapest[u]
            if uf.find(u)!= uf.find(v):
                uf.union(u, v)
                mst.append((u, v, w))
                total_cost += w
                num_components -= 1
    return mst, total_cost

graph = {
    0: [(1, 10), (2, 6), (3, 5)],
    1: [(0, 10), (3, 15)],
    2: [(0, 6), (3, 4)],
    3: [(0, 5), (1, 15), (2, 4)]
}
num_vertices = len(graph)
mst_prims, cost_prims = prims_algorithm(graph, 0)
print("Prim's Algorithm MST:", mst_prims)
print("Prim's Algorithm Cost:", cost_prims)
mst_kruskals, cost_kruskals = kruskals_algorithm(graph, num_vertices)
print("Kruskal's Algorithm MST:", mst_kruskals)
print("Kruskal's Algorithm Cost:", cost_kruskals)
mst_boruvkas, cost_boruvkas = boruvkas_algorithm(graph, num_vertices)
print("Boruvka's Algorithm MST:", mst_boruvkas)
print("Boruvka's Algorithm Cost:", cost_boruvkas)
```

Output:

```
Prim's Algorithm MST: [(0, 3, 5), (3, 2, 4), (0, 1, 10)]
Prim's Algorithm Cost: 19
Kruskal's Algorithm MST: [(2, 3, 4), (0, 3, 5), (0, 1, 10)]
Kruskal's Algorithm Cost: 19
Boruvka's Algorithm MST: [(0, 3, 5), (1, 0, 10), (2, 3, 4)]
Boruvka's Algorithm Cost: 19

Process finished with exit code 0
```