1. word break

```
def word(s, word_dict):
    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_dict:
                dp[i] = True
                break
    return dp[len(s)]


s = input("Enter the string: ")
word_list = input("Enter the words for the dictionary, separated by spaces: ").split()
word_dict = set(word_list)
print(word_dict)
print(word(s,word_dict))
```

2. assembly line scheduling using 3 lines

```
def assembly_line_scheduling(a, t, e, x):
    num_stations = len(a[0])
    T1 = [0] * num_stations
    T2 = [0] * num_stations
    T3 = [0] * num_stations

    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, num_stations):
        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[-1] + x[0], T2[-1] + x[1], T3[-1] + x[2])

    return final_time

a = [[5,6,7],[8,5,6],[6,7,8]]
t = [[0,3,4],[0,4,5],[0,5,6]]
e = [1, 1, 1]
x = [1, 1, 1]

print(assembly_line_scheduling(a, t, e, x))
```

3. all minimum spanning tree

```
class Graph:
    def _init_(self, vertices):
        self.V = vertices
        self.graph = []
        self.adj = [[] for _ in range(vertices)]

    def add_edge(self, u, v, w):
        self.graph.append([u,v,w])
        self.adj[u].append((v, w))
        self.adj[v].append((u, w))

    def kruskal(self):
        result, parent, rank = [], [], []
        self.graph.sort(key=lambda item: item[2])
        for node in range(self.V):
            parent.append(node)
            rank.append(0)
        def find(parent, i):
            if parent[i] == i:
                return i
            return find(parent, parent[i])
        def union(parent, rank, x, y):
            xroot = find(parent, x)
            yroot = find(parent, y)
            if rank[xroot] < rank[yroot]:
                parent[xroot] = yroot
            elif rank[xroot] > rank[yroot]:
                parent[yroot] = xroot
            else:
                parent[yroot] = xroot
                rank[xroot] += 1
        for u, v, w in self.graph:
            x, y = find(parent, u), find(parent, v)
            if x != y:
                result.append([u,v,w])
                union(parent, rank, x, y)
        return result

    def prim(self):
        key, parent, mstSet = [float('inf')] * self.V, [None] * self.V, [False] * self.V
        key[0], parent[0] = 0, -1
        for _ in range(self.V):
            u = min((key[v], v) for v in range(self.V) if not mstSet[v])[1]
            mstSet[u] = True
            for v, w in self.adj[u]:
                if not mstSet[v] and w < key[v]:
                    key[v], parent[v] = w, u
        return [(parent[i], i, key[i]) for i in range(1, self.V)]

    def boruvka(self):
        parent, rank, cheapest = list(range(self.V)), [0] * self.V, [-1] * self.V
        numTrees, MSTweight, result = self.V, 0, []
        def find(parent, i):
            if parent[i] == i:
                return i
            return find(parent, parent[i])
        def union(parent, rank, x, y):
            xroot, yroot = find(parent, x), find(parent, y)
            if rank[xroot] < rank[yroot]:
                parent[xroot] = yroot
            elif rank[xroot] > rank[yroot]:
                parent[yroot] = xroot
            else:
                parent[yroot] = xroot
                rank[xroot] += 1
        while numTrees > 1:
            for u, v, w in self.graph:
                set1, set2 = find(parent, u), find(parent, v)
                if set1 != set2:
                    if cheapest[set1] == -1 or cheapest[set1][2] > w:
                        cheapest[set1] = [u, v, w]
                    if cheapest[set2] == -1 or cheapest[set2][2] > w:
                        cheapest[set2] = [u, v, w]
            for node in range(self.V):
                if cheapest[node] != -1:
                    u, v, w = cheapest[node]
                    set1, set2 = find(parent, u), find(parent, v)
                    if set1 != set2:
                        MSTweight += w
                        union(parent, rank, set1, set2)
                        result.append([u,v,w])
                        numTrees -= 1
            cheapest = [-1] * self.V
        return result

g = Graph(4)
g.add_edge(0,1,10)
g.add_edge(0, 2, 6)
g.add_edge(0, 3, 5)
g.add_edge(1, 3, 15)
g.add_edge(2, 3, 4)

print("Kruskal's MST:", g.kruskal())
print("Prim's MST:", g.prim())
print("Borůvka's MST:", g.boruvka())
```