In [ ]:
```
word break
```

In [16]:
```
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="goodday"
wdict=["go","good", "day"]
print(wbreak(s,wdict))
```

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

In [ ]:
```
assembly line scheduling
```

In [20]:
```
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] +
        T2[i]=min(T2[i-1] + a[1][i], T1[i-1] + t[0][i] + a[1][i], T3[i-1] +
        T3[i]=min(T3[i-1] + a[2][i], T1[i-1] + t[0][i] + a[2][i], T2[i-1] +
    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],
     [0,0,0,0]]
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))
```

```
19
```

In [22]:
```
class DisjointSet:
    def __init__(self, n):
        self.parent=list(range(n))
        self.rank=[0]*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):
        root_u=self.find(u)
        root_v=self.find(v)
        if root_u!=root_v:
            if self.rank[root_u]>self.rank[root_v]:
                self.parent[root_v]=root_u
            elif self.rank[root_u]<self.rank[root_v]:
                self.parent[root_u]=root_v
            else:
                self.parent[root_v]=root_u
                self.rank[root_u]+=1
def kruskal(n, edges):
    edges.sort(key=lambda x: x[2])
    ds=DisjointSet(n)
    mst_weight=0
    for u, v, weight in edges:
        if ds.find(u)!= ds.find(v):
            ds.union(u, v)
            mst_weight+= weight
    return mst_weight
import heapq
def prim(n, edges):
    adj={i: [] for i in range(n)}
    for u, v, weight in edges:
        adj[u].append((weight, v))
        adj[v].append((weight, u))
    mst_weight=0
    visited=[False *n
    min_heap=[(0, 0)]
    while min_heap:
        weight, u=heapq.heappop(min_heap)
        if visited[u]:
            continue
        visited[u]= True
        mst_weight+= weight
        for next_weight, v in adj[u]:
            if not visited[v]:
                heapq.heappush(min_heap, (next_weight, v))
    return mst_weight
def boruvka(n, edges):
    ds=DisjointSet(n)
    mst_weight=0
    num_components=n
    while num_components>1:
        cheapest=[-1]*n
        for u, v, weight in edges:
            set_u=ds.find(u)
            set_v=ds.find(v)
            if set_u!= set_v:
                if cheapest[set_u]==-1 or cheapest[set_u][2]>weight:
                    cheapest[set_u]=(u, v, weight)
                if cheapest[set_v]==-1 or cheapest[set_v][2]>weight:
                    cheapest[set_v]=(u, v, weight)
```

```
        for node in range(n):
            if cheapest[node]!=-1:
                u, v, weight=cheapest[node]
                if ds.find(u)!= ds.find(v):
                    ds.union(u, v)
                    mst_weight+=weight
                    num_components-= 1
    return mst_weight
n=4
edges=[
    (0, 1, 10),
    (0, 2, 6),
    (0, 3, 5),
    (1, 3, 15),
    (2, 3, 4)
]
print("Kruskal's MST weight:", kruskal(n, edges))
print("Prim's MST weight:", prim(n, edges))
print("Boruvka's MST weight:", boruvka(n, edges))
```

```
Kruskal's MST weight: 19
Prim's MST weight: 19
Boruvka's MST weight: 19
```

In [23]:
```
# Prim's Algorithm in Python
def prims(G):
    INF=9999999
    V=5
    selected=[0, 0, 0, 0, 0]
    no_edge=0
    selected[0]=True
    print("Edge : Weight")
    while (no_edge < V - 1):
        minimum=INF
        x=0
        y=0
        for i in range(V):
            if selected[i]:
                for j in range(V):
                    if ((not selected[j]) and G[i][j]):
                        if minimum>G[i][j]:
                            minimum=G[i][j]
                            x=i
                            y=j
        print(str(x) + " - " + str(y) + " : " + str(G[x][y]))
        selected[y]=True
        no_edge +=1
#kruskals algorithm
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
def kruskals(G):
    V=len(G)
    edges=[]
    for i in range(V):
        for j in range(i+1, V):
            if G[i][j] != 0:
                edges.append((G[i][j], i, j))
    edges.sort()
    parent=[]
    rank=[]
    for node in range(V):
        parent.append(node)
        rank.append(0)
    MST=[]
    edge_count= 0
    index=0
    while edge_count<V-1:
        w, u, v=edges[index]
        index=index+1
        x=find(parent, u)
        y=find(parent, v)
        if x!= y:
            edge_count = edge_count + 1
```

```
            MST.append((u, v, w))
            union(parent, rank, x, y)
    print("Edge : Weight")
    for u, v, weight in MST:
        print(str(u) + "-" + str(v) + " : " + str(weight))
#boruvkas algorithm
def findi(parent, i):
    if parent[i] == i:
        return i
    return findi(parent, parent[i])
def unioni(parent, rank, x, y):
    xroot = findi(parent, x)
    yroot = findi(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
def boruvkas(G):
    V = len(G)
    parent = []
    rank=[]
    cheapest=[]
    for node in range(V):
        parent.append(node)
        rank.append(0)
        cheapest.append([-1, float('inf')])
    numTrees=V
    print("Edge : Weight")
    while numTrees>1:
        for i in range(V):
            for j in range(V):
                if G[i][j]!= 0:
                    u=findi(parent, i)
                    v=findi(parent, j)
                    if u!=v:
                        if G[i][j]<cheapest[u][1]:
                            cheapest[u]=[j, G[i][j]]
                        if G[i][j]<cheapest[v][1]:
                            cheapest[v]=[i, G[i][j]]
        for i in range(V):
            if cheapest[i][0]!=-1:
                u=i
                v=cheapest[i][0]
                w=cheapest[i][1]
                set_u=findi(parent, u)
                set_v=findi(parent, v)
                if set_u!= set_v:
                    print(str(u) + " - " + str(v) + " : " + str(w))
                    unioni(parent, rank, set_u, set_v)
                    numTrees-= 1
        cheapest=[[-1, float('inf')] for _ in range(V)]
G = [[0, 9, 75, 0, 0],
     [9, 0, 95, 19, 42],
     [75, 95, 0, 51, 66],
     [0, 19, 51, 0, 31],
     [0, 42, 66, 31, 0]]
print("Prims Algorithm:")
prims(G)
```

```
print("\nkruskals Algorithm:")
kruskals(G)
print("\nBoruvkas Algorithm:")
boruvkas(G)
```

```
Prims Algorithm:
Edge : Weight
0 - 1 : 9
1 - 3 : 19
3 - 4 : 31
3 - 2 : 51

kruskals Algorithm:
Edge : Weight
0-1 : 9
1-3 : 19
3-4 : 31
2-3 : 51

Boruvkas Algorithm:
Edge : Weight
0 - 1 : 9
2 - 3 : 51
3 - 1 : 19
4 - 3 : 31
```

In [ ]: