(67) Aim:
To count the number of good pairs in an array where nums[i] == nums[j] and i<j.

* Pseudocode
count=0
for i=0..n-1
for j=i+1..n-1
if (nums[i] == nums[j]) count++

* Input
nums=[1,2,3,1,1,3]

* output
4

---

(68) Aim.
To find the city that can reach the fewest cities within a given distance threshold using shortest paths

* pseudocode
init dist[n][n]=INF
for each edge (u,v,w): dist[u][v] = dist[v][u] = w
for i: dist[i][i]=0
for k,i,j : dist[i][j] = min (dist[i][j], dist[i][k] + dist[k][j])
Count reachable cities for each i: return city with min count (max index if tie)

* Input
n=4
edges = [[0,1,3], [1,2,1], [1,3,4], [2,3,1]]
distance threshold =4

* output
3

---

(69) Aim: To find the minimum time required for a signal to reach all nodes in a directed network from a given source

* pseudocode
init dist[i] =INF; dist[k]=0
for i=1..n-1:
for each edge (u,v,w):
dist[v] = min (dist[i])
if (ans ==INF) return -1
else return ans

* Input
times=[[2,1,1], [2,3,1], [3,4,1]]
n=4
k=2 * output =2

(9) Aim: To maximize the number of coins you can collect by optimally choosing piles when Alice always picks the largest and Bob picks the smallest

* pseudocode

```
sort (piles)
i = 0 ; j = n-1 ; ans = 0
while (i<j) {
  j-- ;
  ans += piles[j];
  j-- ; i++ ;
}
```

* Input
piles = [2,4,1,2,7,8]

* output
9

(10) Aim: To find the minimum number of coins to add so that all values from 1 to target are obtainable

* pseudocode

```
Sort (coins)
reach = 0, add = 0
for each coin:
while (coin > reach+1):
reach += reach+1
add++
reach += coin
add++
```

* Input
coins = [1,4,10]
target = 19

* output
2

(13) Aim:
To select a subset of non-overlapping jobs maximize total profit

* pseudocode

```
Sort jobs by end time
  dp[0] = 0
  for i = 1..n :
  j = find last Nonoverlap(i)
  dp[i] = max (dp[i])
  return dp[n]
```

*Input:
Starttime = [1,2,3,3]
endtime = [3,4,5,6]
profit = [50,10,40,70]
*output

120

(14) Aim
To find the shortest distances using Dijkstra algorithm
* pseudo code
dist (source) = 0
for i=0..n-1 : visited (i) = false
for count = 0..n-1:
u = minDistance(dist, visited)
visited [u] = true
if (!visited(v) && graph[u][v] != INF
dist (v) = min (dist (v), dist(u) + graph (u)(

* Input:
n = 5
graph = [[0,10,3, INF, INF],
[INF, 0, 1, 2, INF],
[INF, 4, 0, 8, 2],
[INF, INF, INF, 0, 7],
[INF, INF, INF, 9, 0],
]
Source = 0

* Output
[0,7,3,9,5)

(15) Aim:
To find the shortest distance from a source vertex to a target vertex in a weighted graph using Dijkstra's algorithm
* pseudocode
build adjacency List adj from edges d
(source) = 0
priority - queue pq
pq.push(source, 0)
while (!pq.empty):
u = pq.pop()
return dist (target)

*Input:
n=6
edges = [(0,1,7), (0,2,9), (0,5,14), (1,2,10), (1,3,15)
(2,3,11), (2,5,2), (3,4,6), (4,5,9)]
source=0
target=4
*output
20

(16) Aim
To construct a Huffman tree from characters and frequencies and generate prefix-tree
pseudocode
Create min Heap of nodes (char, freq)
while (heap.size > 1):
left = heap.pop()
right = heap.pop()
traverse tree (root, ""):
if leaf: print (char, code)
else:
traverse (left, code + "0")
traverse (right, code + "1")
*Input
n=4
characters = ['a', 'b', 'c', 'd']
frequencies = (5, 9, 12, 13)
*output
[('a', '110'), ('b', '10'), ('c', '0'). ('d', '111')]

(17) Aim
To decode a Huffman encoded string using the corresponding Huffman tree to retrieve the original message
*pseudocode
root = buildHuffmanTree (characters, frequen
node = root
if (bit == '0') node = node.left
* Input
n=4
characters = ['a', 'b', 'c', 'd']
*output

(19) Aim: To determine the minimum no. of required to Load ALL items using a greedy capacity

* pseudocode

Sort (weights decending)
Count=1
Current_Sum=0
for w in weights:
if (current-Sum+w <= max_capacity
Current_Sum+=w
else
Count++
Current_sum=w
return Count

* Input:
n=7
weights=[5,10,15,20,25,30,35]
max_capacity=50

* output
4

---

(20) Aim: To find the MST of a weighted graph using Kruskal's Algorithm

* pseudocode

Sort edges by weight
initialize parent[i]=i for union-findmst
weight=0
mst_edges=[]
for each edge (u,v,w):
if (find(u) != find(v)):
union(u,v)
mst-edges.push(edge)
mst-weight+=w
return mst-edges, mst-weight

* Input
n=4
m=5
edges = [(0,1,10), (0,2,6), (0,3,5), (1,3,15)(2,3,4)]

* output
edges in MST: [(2,3,4), (0,3,5) (0,1,10)]
Total weight of MST: 19

⑩ Ans:

To verify whether a given MST is unique and find another MST if it is not unique

* pseudocode

mst-weight = Sum(weights of given_mst)
for each edge not in MST:
try adding edge to MST
if (weight same & no cycle):
another_mst exists -> mst not unique
return is_unique, another_mst

* Input

n = 4
m = 5
edges = [(0,1,10), (0,2,6), (0,3,5), (1,3,15), (2,3,4)]
given mst = [(2,3,4), (0,3,5), (0,1,10)]

* output

- Is the given MST unique? True