☒ problem No:

☒ problem Name: write a program for insertion sorting in an array.

☒ Theory: Insertion sort is a simple sorting algorithm that works by building a sorted sub-array, one element at a time. It is an in-place comparison-based algorithm.

☒ Algorithm:

(i) [Initialization counter] Set J := N

(ii) Repeat step (iii) and (iv) while J >= K

(iii) Set LA[J+1] := LA[J]

(iv) Set J := J-1

(v) Set LA[K] := ITEM.

(vi) Set N := N+1

(vii) Exit.

python Code:

```
def insertion_sort(arr):
    for i in range(1, len(arr)):
        key = arr[i]
        j = i-1
        while j >=0 and key < arr[j]:
            arr[j+1] = arr[j]
            j--
        arr[j+1] = key
    return arr

arr = [3,1,4,1,5,9,2,6,5,3,5]
sorted_arr = insertion_sort(arr)
print(sorted_arr)
```

output:

```
[1,1,2,3,3,4,5,5,5,6,9]
```

☒ problem No:

☒ problem Name: Write a program for deleting element in an array.

☒ Theory: Deletion sort is also known as selection Sort. Deletion Sort is another simple Sorting algorithm works by repeatedly finding the smallest element from an unsorted portion of the array and swapping it with the first element of the unsorted portion. The algorithm divides the input array into two parts one is sorted sublist and another is unsorted sublist.

☒ Algorithm:

(i) Set ITEM := LA[K]

(ii) Repeat for J = K to N-1:

[move J+1st element upward in LA] set LA[J] := LA[J+1]

~~Set N := N-1~~

[End of loop]

(iii) Set N := N-1.

(iv) Exit.

python Code:

```
def deletion_sort(arr):
    n = len(arr)
    for i in range(n):
        min_idx = i
        for j in range(i+1, n):
            if arr[j] < arr[min_idx]:
                min_idx = j
        arr[i], arr[min_idx] = arr[min_idx], arr[i]
    return arr

arr = [64, 25, 12, 22, 11]
Sorted_arr = deletion_sort(arr)
print(Sorted_arr)
```

output:

```
[11, 12, 22, 25, 64]
```

Problem Name: Write a program to sort an array using bubble sort algorithm.

Theory: We take an unsorted array for our example. Bubble sort takes $O(n^2)$ times.

14, 33, 27, 35, 10

Bubble Sort starts with very first two elements comparing them to check which one is greater

[14], [33], 27, 35, 10

In this case value 33 is greater than 14 So it is already in sorted locations. Next we compare 33 with 27.

14, [33], [27] 35, 10

We find 27 is smaller than 33. These two values must be swapped.

14, 27, [33], [35] 10

Similarly we know then that 35 is greater than 33. So its also swapped. Finally 10 is smaller ~~wit~~ than all other values. So it also compares ~~att~~ with all values of array. So finally we find an sorted array

Algorithm: (Bubble Sort) BUBBLE (DATA, N)

1. Repeat step 2 and 3 for K=1 to N-1
2. Set PTR := 1
3. Repeat while PTR ≤ N-K:
   - (a) If DATA[PTR] > DATA[PTR+1], Then:
     Interchange DATA[PTR] and DATA[PTR+1]
   - (b) Set PTR := PTR+1.
4. Exit.

python Source code:

```
from array import*
def bubbleSort(arr):
    n = len(arr)
    for i in range(0,n):
    for j in range(0, n-i-1):
        if(arr[j] > arr[j+1]:
            arr[j], arr[j+1] = arr[j+1], arr[j]
#main function
while True:
    n = int(input("Enter the number: "))
    arr = array('i',[])
    print("Enter the elements:")
```

```
for i in range(n):
    x = int(input())
    arr.append(x)
bubbleSort(arr)
print("After Bubble Sorting:")
for i in range(0, len(arr)):
    print(arr[i], end=" ")
```

Input:

Enter the number: 5

Enter the array elements: 14 33 27 35 10

Output:

After bubble Sorting:

10, 14 27 33 35

Problem Name:
Write a program to sort an array using merge sort algorithm.

Theory: Suppose an array A with n elements A[1], A[2], ..., A[N] is in memory. The merge sort algorithm which sorts A will first be described by means of a specific example.

Example: Suppose the array A containg 14 elements as follows:

66, 33, 40, 22, 55, 88, 60, 11, 80, 20, 50, 44, 77, 30

The merge sort algorithm will start at the beginning of the array A and merge pairs of sorted sub-arrays as follows:

Step 1. merge each pair of elements to obtain the following list of sorted pairs:

33, 66 22, 40 55, 88 11, 60 20, 80, 44, 50 30, 77

Step 2: merge each pairs to obtain the following sorted quadruplets:

22, 33, 40, 66 11, 55, 60, 88 20, 44, 50, 80 30, 77

Step 3: merge each array pair of pairs to obtain the following to sorted sub array.

11, 22, 33, 40, 55, 60, 66, 88 20, 30, 40, 50, 77, 80

Step-4: merge the two sorted sub-array to obtain the single sorted array

11, 20, 22, 30, 33, 40, 44, 50, 55, 60, 66, 77, 80, 88

The original array A is sorted.

Algorithm: MERGESORT (A, N)

This algorithm sorts the N-elements array A using an auxiliary array B.

1. Set L := 1
2. Repeat steps 3, to 6 while L < N:
3. call MERGEPASS (A, N, L, B).
4. call MERGEPASS (B, N, 2*L, A).
5. Set L := 4*L
   [End of step 2 loop]
6. Exit.

python Source code:

```
from array import *
def mergesort(arr):
    if len(arr) > 1:
        # base case
        mid = len(arr)//2
        low = arr[:mid]
        high = arr[mid:]
        merge_sorting(low)
        merge_sorting(high)
        i = j = k = 0
    while i < len(low) and j < len(high):
        if low[i] < high[j]:
            arr[k] = low[i]
            i += 1
        else:
            arr[k] = high[j]
            j += 1
            k += 1
    while i < len(low):
        arr[k] = low[i]
        i += 1
    while j < len(high)
        arr[k] = high[j]
        j += 1
        k += 1
```

[Diffe... vs ... da[I...

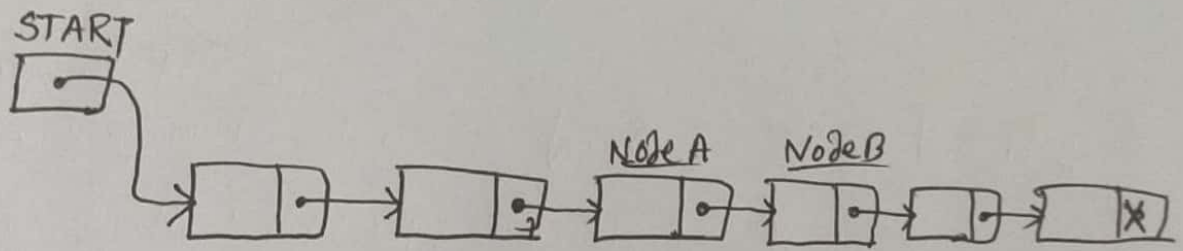
```
# printing function
def printList(arr):
    for i in range(len(arr)):
        print(arr[i], end=" ")
    print()
# main function.
while True:
    n = int(input("Enter the number:"))
    arr = array("i",[])
    print("Enter the array elements:")
    for i in range(0,n):
        x = int(input())
        arr.append(x)
print("Given an array:")
printList(arr)
mergeSort(arr)
print("After sorted array:")
printList(arr)
```

Input: Enter the number: 5
Enter the array Elements: 10 40 20 50 30

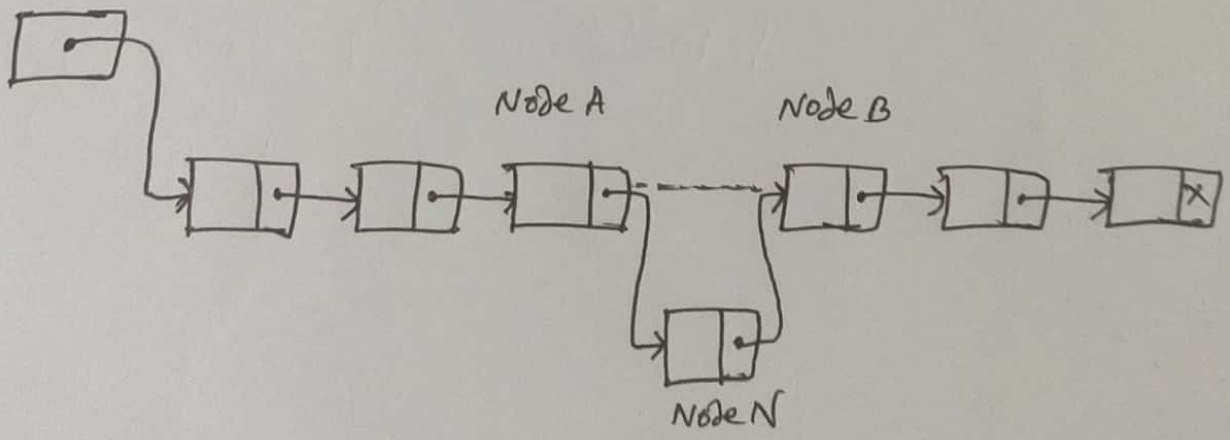
Output: Given an array: 10 40 20 50 30
After sorted array: 10 20 30 40 50

✠ problem Name: Write a program to insert a node into a linked list.

✠ Theory: Let LIST be a linked list with Successive nodes A and B as ~~fig~~ pictured in fig (a). Suppose A node N is to be inserted into the list between nodes A and B. The Schematic diagram of such an insertion appears in fig (b).



(a) Before insertion



(b) After insertion.

Algorithm: INSLOC(INFO, LINK, START, AVAIL, LOC, ITEM)

1. [overflow?] If Avail = Null Then: write: overflow and exit
2. Set new := Avail and Avail := Link[Avail]
3. Set Info[New] := ITEM.
4. If Loc = Null, Then:
   Set Link[New] := Link[Loc] and Link[Loc] := New
5. Exit.

python Source code:

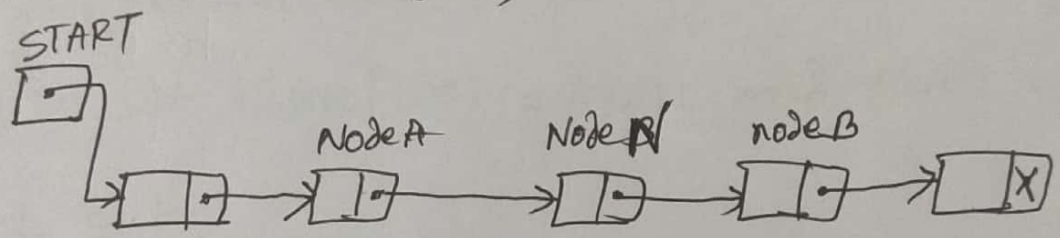
```
class Node:
  def ...init...(self, data):
    Self.data = data
    Self.next = None
class LinkedList:
    def ...init...(self):
        Self.head = None.
  def insert(self, data):
    new node = Node(data)
  if self.head is None:
      Self.head = new-node
  else: last_node = self.head.
    while last_node.next is not None:
        last_node = last_node.next.
        lastnode.next = new node
      def print_list(self):
          current_node = self.head.
          while current_node:
          print(current_node.data)
           current_node = current_node.next.
```

output:

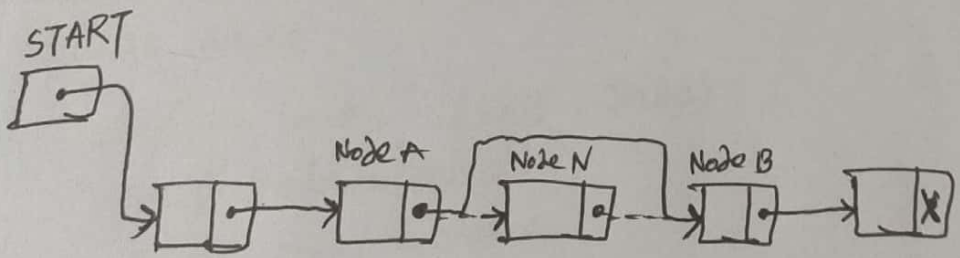
```
1
2
3
```

Problem Name: Write a program to delete a node from a linked list.

Theory: Let list be a linked list with a node N between nodes A and B, as pictured in Fig (a). Suppose node N is to be deleted from the linked list. The schematic diagram of such a deletion appears in Fig (b).



(a) Before deletion



(b) After deletion

Algorithm: DEL(INFO, LINK, START, AVAIL, LOC, LOCP).

1. If LOCP = NULL, Then:
   Set START := LINK[START].

   Else:
   Set LINK[LOCP] := LINK[LOC].

   [End of if structure]

2. [Return deleted node to the avail list]
   Set LINK[LOC] := AVAIL and AVAIL := LOC.

3. Exit.

python Source code:

```
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None

class LinkedList:
    def __init__(self):
        self.head = None

    def push(self, new_data):
        new_node = Node(new_node)
        new_node.Next = self.head.
        self.head = new_node,
```

```
def deleteNode(self, key):
    temp = self.head
    if (temp is not None):
        if (temp.data == key):
            self.head = temp.next
            temp = None
            return
    while (temp is not None):
        if temp.data == key:
            break
        prev = temp
        temp = temp.next
    if (temp == None):
        return
    prev.next = temp.next
    temp = None
def printList(self):
    temp = self.head
    while (temp):
        print(" %d" %(temp.data)),
        temp = temp.Next
# Driver program
llist = LinkedList()
llist.push(7)
llist.push(1)
llist.push(3)
llist.push(2)
```

```
print("created linked list:")
    ~~list = li~~
    list.printList()
    list.deleteNode(1)
print ("linked list after deletion of 1:")
    list.printList()
```

Output:

Created linked list:

2
3
1
7

Linked list after deletion 1:

2
3
7

✠ problem name: Write a program to find an element using binary search algorithm.

✠ Theory: Suppose Data is an array which is sorted in increasing numerical order or — equivalently or alphabetically. Then there is an extremely efficient searching algorithm called binary search. which can be used to find the location LOC of a given ITEM of information in Data.

✠ Algorithm:

(Binary Search) BINARY (DATA, LB, UB, ITEM, LOC)

1. [Initialize Segment variables]
   Set BEG := LB, END := UB and MID = INT((BEG+END)/2)
2. Repeat Steps 3 and 4 while BEG ≤ END and DATA [MID] ≠ ITEM.
3. If ITEM < DATA [MID], Then:
   Set END := MID − 1.
   Else:
   Set BEG := MID + 1
4. Set MID := INT((BEG+END)/2).
   [End of Step ~~to~~ 2 Loop]
5. If DATA [MID] = ITEM, Then:
   Set LOC := MID
   Else: Set LOC := NULL.
6. Exit.

python Source code:

```
from array import*
def binarySearch(arr, n, x):

    low = 0
    high = n-1
    while low <= high:
      mid = low+(high-1)/2
    if arr[mid] = x:
        return mid
      elif arr[mid]> x
        high = mid-1
      else:
        low = mid+1
      return -1
# main function
  while True:
    n = int(input("Enter the number")
    arr1 = array('i', [])
    for i in range (0, n)
        x = int(input())
      arr1.append(x)
    n = int(input("Enter the search number:"))
    arr = sorted(arr1)
    result = binarySearch(arr, n, n)
    if result = = -1
    else: print("Element are not present.")
          print("Element are present", result)
```

※ Input:

Enter the number: 4

30 20 10 50

Enter the Search number: 50

※ Output:

Element are present at index 4.

Problem Name: Write a program to solve the following 0/1 Knapsack problem using dynamic programming approach, profits P = (11, 21, 31, 33), weight w = (2, 11, 22, 15), Knapsack capacity c = 40 and no. of items n = 4

Theory: In this item can not be broken which means thief should take the item as a whole or should leave it. That's why it is called 0/1 Knapsack problem.

- Each item is taken or not taken
- Can not taken fractional amount of an item taken.
- It can not be solved by the greedy approach because it is unable to fill the Knapsack to capacity.
- Greedy Approach doesn't ensure an optimal solution.

Algorithm: KNAPSACK (n, W)

1. for w = 0, W
2. do V[0, w)
3. for i = 0, n
4. do V[i, 0]
5. for w = 0, W
6. do if (wi ≤ W & vi + V[i-1, w-wi] > V[i-1, W])
7. Then V[i, W] ← vi + V[i-1, w-wi]
8. else V[i, w] ← V[i-1, W].

python source code:

```
def zero-one-Knapsack( capacity, weight, profits, n):
Array = [[0 for x in range (capacity+1)] for x
    in range (n+1)]

for i in range (n+1):
    for w in range (capacity +1):

        if i == 0 or w == 0:
            Array [i] [w] = 0
        elif weight [i-1] <= w:
            Array [i] [w] = max (profits [i-1] + Array [i-
                [w-weight [i-1]], Array [i-1] [W])
        else:
            Array [i] [w] = Array [i-1] [w]
        return Array [n] [capacity]
```

```
# matrix for create omothen matrix.
def Knapsack_matrix(capacity, weight, profits, n):
    matrix = [[0 for x in range(capacity+1)] for x in
                range(n+1)]
    #Table in bottom up manner.
    for i in range (n+1):
        for w in range (capacity+1):
            if i==0 or w==0:
            matrix[i][w]=0
            elif weight[i-1] <=w:
                matrix[i][w]=max(profits[i-1]+ matrix
                    [i-1][w- weight[i-1]], matrix[i-1][w])
            else:
                matrix[i][w] = matrix[i-1][w]
    print("The matrix for Zero-One Knapsack:")
    for i in range(len(matrix)):
        for j in range(len(matrix[i])):
            print(matrix[i][j], end=" ")
        print()
#function close
    profits = [int(item) for item in input
                ("Enter the profit value:").split()]
```

```
weight = [int(item) for item in input("Enter the weight
        value").split()]
c = int(input("Enter the capacity : "))
n = int(input("Enter no. of item : "))
Knapsack_matrix(c, weight, profits, n)
print("matrix maximum profit from this matrix
        using zero one Knapsack method :")
print(zero_one_Knapsack(c, weight, profits, n))
```

Output:

```
Enter the profit value: 11 21 31 33
Enter the weight value : 2 11 22 15
Enter the capacity : 40
Enter the no. of item : 4

maximum profit from this matrix using
zero one Knapsack method : 75
```

Problem Name: Write a program using Greedy method to solve this problem when no. of jobs $n=7$, profits $(P_1, P_2, P_3, \cdots P_7) = (3, 5, 20, 18, 1, 6, 30)$ and deadlines $(d_1, d_2, \cdots d_7) = (1, 3, 4, 3, 2, 1, 2)$.

Theory: Given an array of jobs where every job has deadline and associated profit if the job is finished before the deadline. It is also given that every job takes the single unit of time, so the minimum possible deadline for any job is 1. How to maximize total profit if only one job can be scheduled at a time.

Example:

Input: 7 jobs following deadlines and profits

JobID	Deadline	Profit
a	1	3
b	3	5
c	4	20
d	3	18
e	2	1
f	1	6
g	2	30

Algorithm:

step1: Sort all jobs in decreasing order of profit

step2: Iterate on jobs in decreasing order of profit. For each job, do the following:

(a) Find a time slot i, such that it is empty and i < deadline and i is greatest. put the job in this slot and mark this slot filled.

(b) If no such i exits, then ignore the job.

python source code:

```
def __init__(self, jobNO, deadline, profit):
    self.jobNO=jobNO
    self.deadline = deadline
    self.profit = profit

def Greedy_method(matrix, T):
    profit = 0
    slot = [-1] * T
    print("slot=", slot)
```

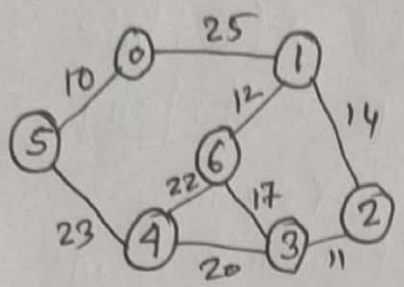
```
for pip in matrix:
    print(pip.jobNo, pip.deadline, pip.profit)
    for j in reversed(range(pip.deadline)):
        if j<T and slot[j]==-1:
            slot[j] = pip.jobNo
            profit += pip.profit
            break
print("The maximum profit sequence are :",
      list(filter(lambda var: var!=-1, slot)))
    return profit.
# function close and value of jobNo, deadline, profits
matrix = [pip(1,1,3), pip(2,3,5), pip(3,4,20),
          pip(4,3,18), pip(5,2,1), pip(6,1,6), pip(7,2,30)]
jobNo = 5
matrix.sort(key=lambda var: var.profit, reverse=True)
print("maximum profit is:", Greedy_method(matrix, jobNo))
```

output: slot = [-1,-1,-1,-1,-1,-1,-1]

The maximum profit sequence are : [6,7,4,3]

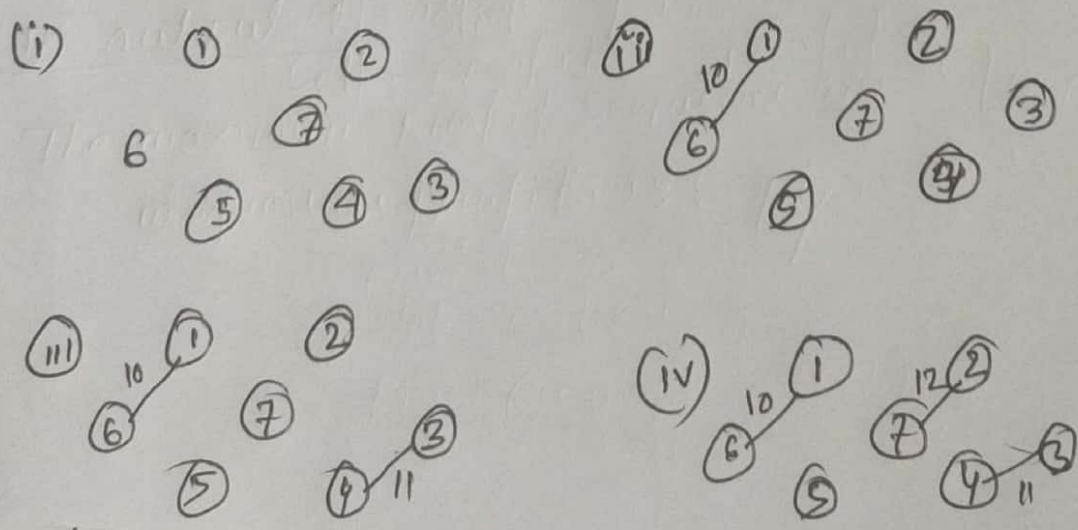
maximum profit is: 74

Problem Name: Write a program to implement Kruskal's algorithm and find the minimum spanning tree for the following graph.



Theory: Kruskal's algorithm always the minimum cost edge has to be selected. But it is not necessary that selected optimum edge is adjacent.

Example: First we will select all the vertices then an edge with optimal weight is selected from heap even though it is not adjacent to previously selected edge. Care should be taken for not forming circuit.



L + INF, INF, INF, INF, INF

```
Tree[k][0] = v1
Tree[k][1] = v2
    k++
    count++;
    Sum += a[pos].cost
union (i, j, parent); }
a[pos].cost INFINITY }
if (count = tot_nodes - 1) then {
    for i=0 to tot_nodes - 1 {
        write (tree[i][0], tree[i][1] } }
        write ("cost of spanning tree is", sum) {
```

python Source code:

```
from dataclass import dataclass, field

class Edge:
    src: int
    dst: int
    weight: int

class Graph:
    num_nodes: int
    edgelist: list
    parent: list = field(default_factory=list)
    rank: list = field(default_factory=list)
    mst: list = field(default_factory=list)
```

```
def Findparent(self, node):
    if self.parent[node] == node:
        return node
    return self.Findparent(self.parent[node])
    def KruskalMST(self):
    self.edgelist.sort(key=lamda Edge: Edge.weight)
    self.parent = [None] * self.num_nodes
    self.rank = [None] * self.num_nodes

for n in range(self.num_nodes):
    self.parent[n] = n
    self.rank[n] = 0

for edge in self.edgelist:
    root1 = self.Findparent(edge.src)
    root2 = self.Findparent(edge.dst)

if root1 != root2:
    self.mst.append(edge)
    if self.rank[root1] < self.rank[root2]:
        self.parent[root1] = root2

    else:
        self.parent[root2] = root1
        self.rank[root1] += 1

print("Edge of minimum Spanning Tree are in g1:")
    cost=0
```

```
for edge in self.mst:
    print("[" + str(edge.src) + "-" + str(edge.dst) + "](" 
        + str(edge.weight) + ")", end = ' ')
    cost += edge.weight
print("\n cost of minimum Sparning Tree: " + str(cost))

def main():
    num_nodes = 7
    a = Edge(0,1,1)
    b = Edge(0,2,2)
    c = Edge(0,3,1)
    d = Edge(0,4,1)
    e = Edge(0,5,2)
    f = Edge(0,6,1)
    g = Edge(1,2,2)
    h = Edge(1,6,2)
    i = Edge(2,3,1)
    j = Edge(3,4,2)
    k = Edge(4,5,2)
    l = Edge(5,6,1)
g2 = Graph(num_nodes, [a,b,c,d,e,f,g,h,i,j,k,l])
g2.KruskalMST()
if __name__ == "__main__":
    main()
```

Output: Edge of minimum spanning Tree are

in g1: [0-1] (1) [0-3] (1) [0-4] (1) [0-6] (1) [2-3] (1)
[5-6] (1)

cost of minimum Sparning Tree : 6

✠ problem Name: Write a program to find the shortest path from a directed weighted ~~graph~~ multistage graph using dynamic algorithm.

✠ Theory: A multi-stage graph $G=(V,E)$ which is a directed graph all the vertices are partitioned into the K stages where $K>2$. In multistage graph problem we ~~find~~ have to find the shortest path from source to sink. The cost of each path is calculated by using the weight given along the edge. The cost of each path from source to sink is the sum of the costs of edges on the path. In multi-stage graph problem we have to find the path from S to T. The multi-stage graph can be solved using forward and backward approach. Let us solve multistage problem for ~~both the~~ backward approach.

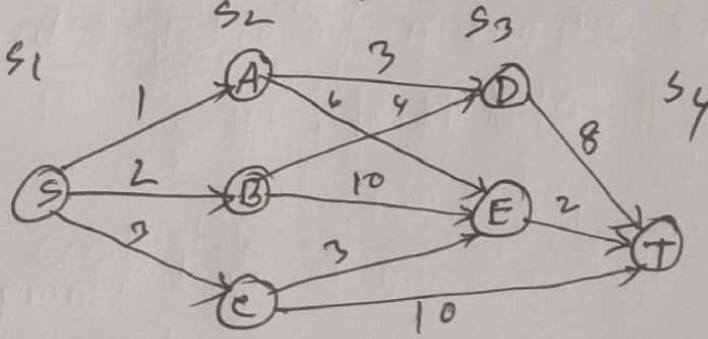


Fig: multistage graph.

Algorithm:

```
Backward_Gr(G, stages, n, p)
// input: The multistage graph G=(V,E)
// stages is for total no. of stages.
// n is total number of vertices of G.
// p is an array for restoring path.
// output: The path with minimum cost.

back_cost[i] = 0
for(i=0 to n-2) do {
   r = Get_min(i, n)
  back_cost[i] = back_cost[r] + c[r][i]
  d[i] = r }
  p[0] = 0
  p[stages-1] = n-1
  for (i = stages-1 to 1) do
     p[i] = d[p[i+1]];
```

Analysis clearly the above algorithm has a time complexity $\theta(|V|+|E|)$.

python source code:

```
def shortestDist(graph):
    global INF
    dist = [0] * N
    dist[N-1] = 0
    for i in range(N-2, -1, -1):
        dist[i] = INF
        for j in range(N):
            if graph[i][j] == INF:
                continue
            dist[i] = min(dist[i], graph[i][j]+dist[j])
    return dist[0]

#driver code
N=8
INF = 999999999
graph = [[INF, 1, 2, 5, INF, INF, INF],
         [INF, INF, INF, INF, 4, 11, INF, INF],
         [INF, INF, INF, INF, 9, 5, 16, INF],
         [INF, INF, INF, INF, INF, INF, 18],
         [INF, INF, INF, INF, INF, INF, 13],
         [INF, INF, INF, INF, INF, INF, 2]]
```

```
print("shortest path from source to sink is:")
print(shortestDist(graph))
```

output:

Shortest path from source to sink is: 9

✻ problem name: Write a program to find the all pair shortest path from a graph using Floyd warshall's Algorithm.

✻ Theory: When a weighted graph, represented by its weight matrix w then objective is to find the distance between every pair of nodes.
We will apply dynamic programming to solve the all pair of shortest path.

Step1: let, A[i,j] be the length of shortest path from node i to node j such that the label for every intermediate node will be <= K
we will compute A^K for K = 1 ... n for n nodes.

Step2: For solving all pair shortest path, the principle of optimality is used. That means any subpath of shortest path is a shortest path between the end nodes.

(i) path going from i to j via K,
(ii) path which is not going via K
select only shortest path from two case.

Step3: The shortest path can be calculate

using bottom up computation method. Following is recursion method.

Initially: $A^0 = W[i,j]$

Next computations:

$$A^K(i,j) = \min \left\{ A^{K-1}_{(ij)} , A^{K-1}_{(iK)} + A^{K-1}_{(Kj)} \right\}$$

where $1 \leq K \leq n$

python source code:

```
# Number of vertices
    V=4
    INF = 999999
    def floydWarshall(graph):
    dist = list(map(lambda i: list(map(lambda j: j, i)),
                                    graph))

for K in range(V):
    for j in range(V):
        dist[i][j] = min(dist[i][j], dist[i][K] +
                                    dist[K][j])
        printSolution(dist)

def printSolution(dist):
print("Matrix shows the shortest distance between
        every pair of vertices")
```

```
for i in range (v):
    for j in range(v):
        if (dist[i][j] == INF):
            print ("%7s" %. ("INF"), end=" ")
        else:
            print ("%7d\t" %.(dist[i][j]), end=" ")
        if j==v-1:
            print()
#Driver code:
    if __name__ == "__main__"
# function call
    floywarshall (graph)
```

output:

matrix shows the shortest distance between every pair of vertices:

```
0     5     8     9
INF   0     3     4
INF   INF   0     1
INF   INF   INF   0
```

Problem Name: Write a program to solve the n-queens problem.

Theory: consider a nxn chessboard on which we have to place n queens so that no two queens attack each other by being in the same row or in the same column or on the same diagonal.

Algorithm:

```
Algorithm queen(n)
// input: total number of queens n.

    for column =1 to n do {
        if (place(row, column)) Then {
        board [row [column]] //no conflict so place queen

    if (row=n) Then
        print_board (n)
else: queen (row+1, n) { }
Algorithm place(row, column)
    for i=1 to row-1 do {
        if (board [i] = column) Then
                return 0;
        else if (abs(board [i] - column) = abs (i-row)) Then
                return 0; }
        return 1
```

Python Source code:

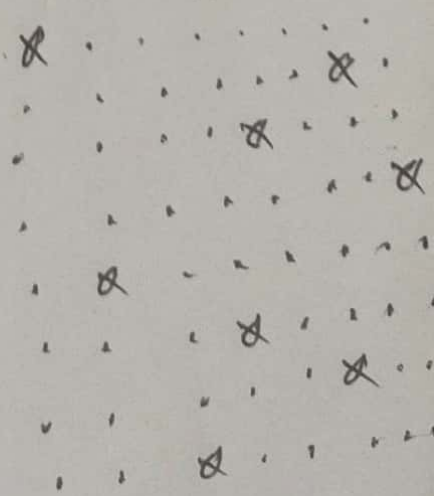
```
import copy
import random
def take_input():
while True:
    Try:
        n = int(input(' gnput size of queen? n = "))
            if n <= 3:
            print("Enter the value greater than 4 or
                                                equal")
                continue.
            return n.
    except ValueError:
        print("Invalid value entered, Enter again")

def get_board(n):
    board = ["."]*n
    for i in range(n):
        board[i] = ["."]*n
        return board.

def print_solution(solution, n):
    x = random.randint(0, len(solutions)-1)
    for row in solutions[x]:
    print(" ".join(row))
```

```
def solve(board, col, n)
    if col >= n:
        return
    for i in range(n):
        if is_safe(board, i, col, n):
            board[i][col] = "Q"
            if col == n-1:
                add_solution(board)
                board[i][col] = "."
                return
            solve(board, col+1, n)
            board[i][col] = "."

def is_safe(board, row, col, n):
    for j in range(col):
        if board[row][j] == "Q":
            return False
    i, j = row, col
    while i >= 0 and j >= 0:
        if board[i][j] == "Q":
            return False
        i, j = i-1, j-1
    x, y = row, col
    while x < n and y >= 0:
        if board[x][y] == "Q":
            return False
        x, y = x+1, y-1
```

```
        x = x+1
        y = y-1
    return True
def add_solution (board):
    global solutions
    saved_board = copy.deepcopy(board)
    solutions.append (saved_board)

n = take_input()
board = get_board (n)
solutions = []
solve (board, 0, n)
print ()
print("one of the solution is: \n")
print_solution (solutions, n)
print()
print ("Total number of solutions = ", len (solutions))
```

Output:

```
Input size of queen? n = 8
one of the solution is:
```



Total Number of solutions = 92