ADVANCED DATA STRUCTURES

LAB MANUAL

Course Code	**:**	**BCSB09**
Regulations	**:**	**IARE - R18**
Semester	**:**	**I**
Branch	**:**	**CSE**

Prepared by

Ms. S SWARAJYA LAXMI, ASSISTANT PROFESSOR

COMPUTER SCIENCE AND ENGINEERING

INSTITUTE OF AERONAUTICAL ENGINEERING
(Autonomous)
Dundigal, Hyderabad - 500 043

INSTITUTE OF AERONAUTICAL ENGINEERING

(Autonomous)

Dundigal, Hyderabad - 500 043

1. PROGRAM OUTCOMES:

PROGRAM OUTCOMES (POS)	
PO 1	Apply Analyze a problem, identify and define computing requirements, design and implement appropriate solutions.
PO 2	Solve complex heterogeneous data intensive analytical based problems of real time scenario using state of the art hardware/software tools.
PO 3	Demonstrate a degree of mastery in emerging areas of CSE/IT like IoT, AI, Data Analytics, Machine Learning, cyber security, etc.
PO 4	Write and present a substantial technical report/document.
PO 5	Independently carry out research/investigation and development work to solve practical problems.
PO 6	Function effectively on teams to establish goals, plan tasks, meet deadlines, manage risk and produce deliverables.
PO 7	Engage in life-long learning and professional development through self-study, continuing education, professional and doctoral level studies.

2. ATTAINMENT OF PROGRAM OUTCOMES:

WEEK NO	Experiment	Program Outcomes Attained
1.	DIVIDE AND CONQUER - 1	PO1, PO2, PO5
2.	DIVIDE AND CONQUER - 2	PO1, PO2, PO5
3.	IMPLEMENTATION OF STACK AND QUEUE	PO1, PO2, PO5
4.	HASHING TECHNIQUES	PO1, PO2, PO5
5.	APPLICATIONS OF STACK	PO1, PO2, PO5
6.	BINARY SEARCH TREE	PO1, PO2, PO3
7.	DISJOINT SET OPERATIONS	PO1, PO2, PO3
8.	GRAPH TRAVERSAL TECHNIQUES	PO1, PO2, PO3, PO4
9.	SHORTEST PATHS ALGORITHM	PO1, PO2, PO3, PO4
10.	MINIMUM COST SPANNING TREE	PO1, PO2, PO3, PO4, PO5
11.	TREE TRAVESRSALS	PO1, PO2, PO3, PO4, PO5
12.	ALL PAIRS SHORTEST PATHS	PO1, PO2, PO3, PO4, PO6

3. MAPPING COURSE OBJECTIVES LEADING TO THE ACHIEVEMENT OF PROGRAM OUTCOMES:

Course Objectives (COs)	Program Outcomes (POs)						
	PO1	PO2	PO3	PO4	PO5	PO6	PO7
I	√	√	√				
II	√	√		√	√		
III			√		√		
IV	√	√	√				

4 SYLLUBUS

ADVANCED DATA STRUCTURES LABORATORY

I Semester: CSE

Course Code	Category	Hours / Week			Credits	Maximum Marks		
		L	T	P	C	CIA	SEE	Total
BCSB09	Core	3	-	-	2	30	70	100
Contact Classes: Nil	Tutorial Classes: Nil	Practical Classes: 36				Total Classes: 36		

COURSE OBJECTIVES:

The course should enable the students to:

I. Implement linear and non linear data structures.

II. Analyze various algorithms based on their time complexity.

III. Choose appropriate data structure and algorithm design method for a specific application.

IV. Identify suitable data structure to solve various computing problems.

COURSE OUTCOMES (COs):

CO 1: Implement divide and conquer techniques to solve a given problem.

CO 2: Implement hashing techniques like linear probing, quadratic probing, random probing and double hashing/rehashing.

CO 3: Perform Stack operations to convert infix expression into post fix expression and evaluate the post fix expression.

CO 4: Differentiate graph traversal techniques Like Depth First Search, Breadth First Search.

CO 5: Identify shortest path to other vertices using various algorithms.

COURSE LEARNING OUTCOMES (CLOs):

1. Analyze time and space complexity of an algorithm for their performance analysis.
2. Understand arrays, single and doubly linked lists in linear data structure and tress , graphs in non-linear data structure.
3. Master a variety of advanced abstract data type (ADT) and their implementations
4. Understand dynamic data structures and relevant standard algorithms
5. Design and analyze and Concepts of heap, priority queue
6. Analyze probing methods like linear probing and quadratic probing
7. Understand and implement hash table and linear list representation
8. Understand the properties of binary tress and implement recursive and non-recursive traversals
9. Understand graphs terminology, representations and traversals in Graphs
10. Implement Depth First Search and Breath First Searching methods of non –linear data structures
11. Analyze Dijkstra’s algorithm for single source shortest path problem for minimum cost spanning trees
12. Implement binary search ADT for finding parent node, smallest and largest values in binary search
13. Understand and implement operations and applications of red-Black and splay Trees
14. Implement Huffman Coding and decoding for text compression.

LIST OF EXPERIMENTS

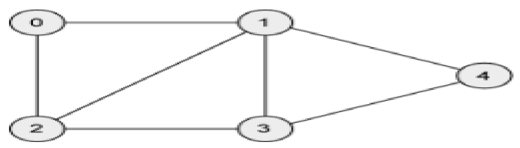
Week-1	DIVIDE AND CONQUER - 1

a. Implement Quick Sort on 1D array of Student structure (contains student name, student_roll_no,total_marks), with key as student_roll_no and count the number of swap performed.
b. Implement Merge Sort on 1D array of Student structure (contains student_name, student_roll_no,total_marks), with key as student_roll_no and count the number of swap performed.

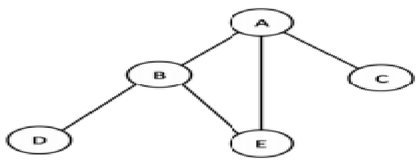
Week-2	DIVIDE AND CONQUER - 2
a. Design and analyze a divide and conquer algorithm for following maximum sub-array sum problem:given an array of integer's find a sub-array [a contagious portion of the array] which gives the maximum sum. b. Design a binary search on 1D array of Employee structure (contains employee_name, emp_no, emp_salary), with key as emp_no and count the number of comparison happened.	
Week-3	**IMPLEMENTATION OF STACK AND QUEUE**
a. Implement 3-stacks of size 'm' in an array of size 'n' with all the basic operations such as Is Empty(i),Push(i), Pop(i), IsFull(i) where 'i' denotes the stack number (1,2,3), Stacks are not overlapping each other. b. Design and implement Queue and its operations using Arrays	
Week-4	**HASHING TECHNIQUES**
Write a program to store k keys into an array of size n at the location computed using a hash function, loc =key % n, where k<=n and k takes values from [1 to m], m>n. To handle the collisions use the following collision resolution techniques a. Linear probing b. Quadratic probing c. Random probing d. Double hashing/rehashing	
Week-5	**APPLICATIONS OF STACK**
Write C programs for the following: a. Uses Stack operations to convert infix expression into post fix expression. b. Uses Stack operations for evaluating the post fix expression.	
Week-6	**BINARY SEARCH TREE**
Write a program for Binary Search Tree to implement following operations: a. Insertion b. Deletion i. Delete node with only child ii. Delete node with both children c. Finding an element d. Finding Min element e. Finding Max element f. Left child of the given node g. Right child of the given node h. Finding the number of nodes, leaves nodes, full nodes, ancestors, descendants.	
Week-7	**DISJOINT SET OPERATIONS**
a. Write a program to implement Make_Set, Find_Set and Union functions for Disjoint Set Data Structure for a given undirected graph G(V,E) using the linked list representation with simple implementation of Union operation. b. Write a program to implement Make_Set, Find_Set and Union functions for Disjoint Set Data Structure for a given undirected graph G(V,E) using the linked list representation with weighted-union heuristic approach.	

Week-8 | GRAPH TRAVERSAL TECHNIQUES

a. Print all the nodes reachable from a given starting node in a digraph using BFS method.

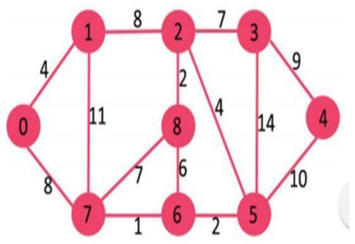


b. Check whether a given graph is connected or not using DFS method.



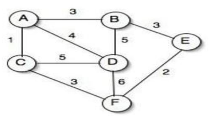
Week-9 | SHORTEST PATHS ALGORITHM

From a given vertex in a weighted connected graph, find shortest paths to other vertices using Dijkstra's algorithm.



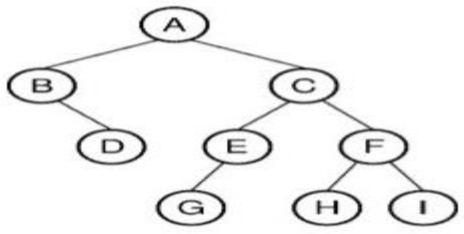
Week-10 | MINIMUM COST SPANNING TREE

Find Minimum Cost Spanning Tree of a given undirected graph using Kruskal's



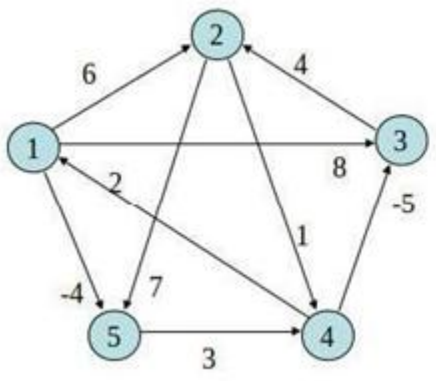
WeeK-11 | TREE TRAVESRSALS

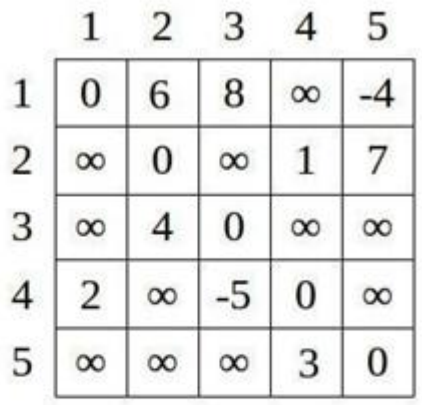
Perform various tree traversal algorithms for a given tree.



Week-12	ALL PAIRS SHORTEST PATHS

Implement All-Pairs Shortest Paths Problem using Floyd's algorithm.





	1	2	3	4	5
1	0	6	8	∞	-4
2	∞	0	∞	1	7
3	∞	4	0	∞	∞
4	2	∞	-5	0	∞
5	∞	∞	∞	3	0

Reference Books:

1. Kernighan Brian W, Dennis M. Ritchie, “The C Programming Language”, Prentice Hall of India, RePrint, 2008.
2. Balagurusamy E, “Programming in ANSIC”, Tata McGraw Hill, 6th Edition, 2008.
3. Gottfried Byron, “Schaum's Outline of Programming with C”, Tata McGraw Hill, 1st Edition, 2010.
4. Lipschutz Seymour, “Data Structures Schaum's Outlines Series”, Tata McGraw Hill, 3rd Edition, 2014.
5. Horowitz Ellis, Satraj Sahni, Susan Anderson, Freed, “Fundamentals of Data Structures in C”, W. H.Freeman Company, 2ndEdition, 2011.

Web References:

1. http://www.tutorialspoint.com/data_structures_algorithms
2. http://www.geeksforgeeks.org/data-structures/
3. http://www.studytonight.com/data-structures/
4. http://www.coursera.org/specializations/data-structures-algorithms

5. INDEX:

WEEK.NO	Experiment	Page Numbers
1.	DIVIDE AND CONQUER - 1	9
2.	DIVIDE AND CONQUER - 2	13
3.	IMPLEMENTATION OF STACK AND QUEUE	16
4.	HASHING TECHNIQUES	22
5.	APPLICATIONS OF STACK	25
6.	BINARY SEARCH TREE	27
7.	DISJOINT SET OPERATIONS	30
8.	GRAPH TRAVERSAL TECHNIQUES	32
9.	SHORTEST PATHS ALGORITHM	39
10.	MINIMUM COST SPANNING TREE	44
11.	TREE TRAVESRSALS	49
12.	ALL PAIRS SHORTEST PATHS	52

WEEK-1

DIVIDE AND CONQUER - 1

1.1 OBJECTIVE:

a. To implement Quick Sort on 1D array of Student structure (contains student name, student_roll_no,total_marks), with key as student_roll_no and count the number of swap performed.

b. To implement Merge Sort on 1D array of Student structure (contains student_name, student_roll_no,total_marks), with key as student_roll_no and count the number of swap performed.

1.2 RESOURCES:

Python 3.4

1.3 PROGRAM LOGIC:

To implement Quick Sort

```
# This function takes last element as pivot, places
# the pivot element at its correct position in sorted
# array, and places all smaller (smaller than pivot)
# to left of pivot and all greater elements to right
# of pivot
def partition(arr,low,high):
    i = ( low-1 )         # index of smaller element
    pivot = arr[high]     # pivot

    for j in range(low , high):

        # If current element is smaller than the pivot
        if   arr[j] < pivot:

            # increment index of smaller element
            i = i+1
            arr[i],arr[j] = arr[j],arr[i]

    arr[i+1],arr[high] = arr[high],arr[i+1]
    return ( i+1 )

# The main function that implements QuickSort
# arr[] --> Array to be sorted,
# low  --> Starting index,
# high  --> Ending index

# Function to do Quick sort
def quickSort(arr,low,high):
    if low < high:
        # pi is partitioning index, arr[p] is now
        # at right place
        pi = partition(arr,low,high)

        # Separately sort elements before
        # partition and after partition
        quickSort(arr, low, pi-1)
        quickSort(arr, pi+1, high)

# Driver code to test above
arr = [10, 7, 8, 9, 1, 5]
n = len(arr)
quickSort(arr,0,n-1)
print ("Sorted array is:")
for i in range(n):
    print ("%d" %arr[i])
```

To implement Merge Sort

```
# Python    program for implementation of MergeSort
def mergeSort(arr):
    if len(arr) >1:
        mid = len(arr)//2 #Finding the mid of the array
        L = arr[:mid] # Dividing the array elements
        R = arr[mid:] # into 2 halves

        mergeSort(L) # Sorting the first half
        mergeSort(R) # Sorting the second half

        i = j = k = 0

        # Copy data to temp arrays L[] and R[]
        while i < len(L) and j < len(R):
            if L[i] < R[j]:
                arr[k] = L[i]
                i+=1
            else:
                arr[k] = R[j]
                j+=1
            k+=1

        # Checking if any element was left
        while i < len(L):
            arr[k] = L[i]
            i+=1
            k+=1

        while j < len(R):
            arr[k] = R[j]
            j+=1
            k+=1

# Code to print the list
def printList(arr):
    for i in range(len(arr)):
        print(arr[i],end=" ")
    print()

# driver code to test the above code
if __name__ == '__main__':
    arr = [12, 11, 13, 5, 6, 7]
    print ("Given array is", end="\n")
    printList(arr)
    mergeSort(arr)
    print("Sorted array is: ", end="\n")
    printList(arr)
```

1.4 INPUT/OUTPUT

To implement Quick Sort

```
Case 1:
Enter the list of numbers: 5 2 8 10 3 0 4
Sorted list: [0, 2, 3, 4, 5, 8, 10]

Case 2:
Enter the list of numbers: 7 4 3 2 1
Sorted list: [1, 2, 3, 4, 7]

Case 3:
Enter the list of numbers: 2
Sorted list: [2]
```

To implement Merge Sort

```
Case 1:
Enter the list of numbers: 3 1 5 8 2 5 1 3
Sorted list: [1, 1, 2, 3, 3, 5, 5, 8]

Case 2:
Enter the list of numbers: 5 3 2 1 0
Sorted list: [0, 1, 2, 3, 5]

Case 3:
Enter the list of numbers: 1
Sorted list: [1]
```

1.5 PRE LAB VIVA QUESTIONS:

1. What is the classification of control structures?
2. What is the syntax of for loop?
3. What is the difference between for loop and while loop?

1.6 LAB ASSIGNMENT:

1. Write a Python program to Concatenate Two Strings
2. Write a Python program to Remove all Characters in a String Except Alphabets.
3. Write a Python program to Find Largest Element of an Array

1.7 POST LAB VIVA QUESTIONS:

1. What are conditional controls?
2. What is the syntax of break statement?
3. What are jumping statements?

WEEK - 2
DIVIDE AND CONQUER – 2

2.1 OBJECTIVE:

a. Design and analyze a divide and conquer algorithm for following maximum sub-array sum problem: given an array of integer's find a sub-array [a contagious portion of the array] which gives the maximum sum.

b. Design a binary search on 1D array of Employee structure (contains employee_name, emp_no, emp_salary), with key as emp_no and count the number of comparison happened.

2.2 RESOURCES:

Python 3.4

2.3 PROGRAM LOGIC:

a)Divide and conquer algorithm

```
# A Divide and Conquer based program
# for maximum subarray sum problem

# Find the maximum possible sum in
# arr[] auch that arr[m] is part of it
def maxCrossingSum(arr, l, m, h) :

    # Include elements on left of mid.
    sm = 0; left_sum = -10000

    for i in range(m, l-1, -1) :
        sm = sm + arr[i]

        if (sm > left_sum) :
            left_sum = sm

          # Include elements on right of mid
    sm = 0; right_sum = -1000
    for i in range(m + 1, h + 1) :
        sm = sm + arr[i]

        if (sm > right_sum) :
            right_sum = sm


    # Return sum of elements on left and right of mid
    return left_sum + right_sum;


# Returns sum of maxium sum subarray in aa[l..h]
```

```
def maxSubArraySum(arr, l, h) :

    # Base Case: Only one element
    if (l == h) :
        return arr[l]

    # Find middle point
    m = (l + h) // 2

    # Return maximum of following three possible cases
    # a) Maximum subarray sum in left half
    # b) Maximum subarray sum in right half
    # c) Maximum subarray sum such that the
    #     subarray crosses the midpoint
    return max(maxSubArraySum(arr, l, m),
               maxSubArraySum(arr, m+1, h),
               maxCrossingSum(arr, l, m, h))


# Driver Code
arr = [2, 3, 4, 5, 7]
n = len(arr)

max_sum = maxSubArraySum(arr, 0, n-1)
print("Maximum contiguous sum is ", max_sum)
```

b) binary search

```
# Python    Program for recursive binary search.

# Returns index of x in arr if present, else -1
def binarySearch (arr, l, r, x):

    # Check base case
    if r >= l:

        mid = l + (r - l)/2

        # If element is present at the middle itself
        if arr[mid] == x:
            return mid

        # If element is smaller than mid, then it
        # can only be present in left subarray
        elif arr[mid] > x:
            return binarySearch(arr, l, mid-1, x)

        # Else the element can only be present
        # in right subarray
        else:
            return binarySearch(arr, mid + 1, r, x)

    else:
        # Element is not present in the array
        return -1

# Test array
arr = [ 2, 3, 4, 10, 40 ]
x = 10

# Function call
result = binarySearch(arr, 0, len(arr)-1, x)

if result != -1:
    print "Element is present at index % d" % result
else:
    print "Element is not present in array"
```

2.4 INPUT/OUTPUT:

a) Divide and conquer algorithm

```
5
None
```

b) Binary search

```
Element is present at index 3
```

2.5 PRE LAB VIVA QUESTIONS:

1. What are vectors used for in Python?
2. What are maps in Python?
3. What is the use of standard template library in Python?

2.6 LAB ASSIGNMENT:

1. Write a Python program to Reverse a Sentence Using Recursion
2. Write a Python Program to Find G.C.D Using Recursion
3. Write a Python Program to Check Whether a Number can be Express as Sum of Two Prime Numbers

2.7 POST LAB VIVA QUESTIONS:

1. How STL is different from the Python Standard Library?
2. What are vectors used for in Python?
3. What are maps in Python?

WEEK- 3

IMPLEMENTATION OF STACK AND QUEUE

OBJECTIVE:

a. Implement 3-stacks of size ‘m’ in an array of size ‘n’ with all the basic operations such as Is Empty(i),Push(i), Pop(i), IsFull(i) where ‘i’ denotes the stack number (1,2,3), Stacks are not overlapping each other.

b. Design and implement Queue and its operations using Arrays.

3.2 RESOURCES:

Python 3.4

3.3 PROGRAM LOGIC:

a. Stack operations

```
class StackContainer(object):
    def __init__(self, stack_count=3, size=256):
        self.stack_count = stack_count
        self.stack_top = [None] * stack_count
        self.size = size
        # Create arena of doubly linked list
        self.arena = [{'prev': x-1, 'next': x+1} for x in range(self.size)]
        self.arena[0]['prev'] = None
        self.arena[self.size-1]['next'] = None
        self.arena_head = 0

    def _allocate(self):
        new_pos = self.arena_head
        free = self.arena[new_pos]
        next = free['next']
        if next:
            self.arena[next]['prev'] = None
            self.arena_head = next
        else:
            self.arena_head = None
        return new_pos

    def _dump(self, stack_num):
        assert 0 <= stack_num < self.stack_count
        curr = self.stack_top[stack_num]
        while curr is not None:
            d = self.arena[curr]
            print '\t', curr, d
            curr = d['prev']

    def _dump_all(self):
        print '-' * 30
        for i in range(self.stack_count):
            print "Stack %d" % i
            self._dump(i)

    def _dump_arena(self):
        print "Dump arena"
        curr = self.arena_head
        while curr is not None:
            d = self.arena[curr]
            print '\t', d
            curr = d['next']

    def push(self, stack_num, value):
        assert 0 <= stack_num < self.stack_count
        # Find space in arena for new value, update pointers
        new_pos = self._allocate()
        # Put value-to-push into a stack element
        d = {'value': value, 'prev': self.stack_top[stack_num], 'pos': new_pos}
        self.arena[new_pos] = d
        self.stack_top[stack_num] = new_pos

    def pop(self, stack_num):
        assert 0 <= stack_num < self.stack_count
        top = self.stack_top[stack_num]
        d = self.arena[top]
        assert d['pos'] == top
        self.stack_top[stack_num] = d['prev']
        arena_elem = {'prev': None, 'next': self.arena_head}
        # Link the current head to the new head
        head = self.arena[self.arena_head]
        head['prev'] = top
        # Set the curr_pos to be the new head
        self.arena[top] = arena_elem
        self.arena_head = top
        return d['value']

if __name__ == '__main__':
    sc = StackContainer(3, 10)
    sc._dump_arena()
    sc.push(0, 'First')
    sc._dump_all()
    sc.push(0, 'Second')
    sc.push(0, 'Third')
    sc._dump_all()
    sc.push(1, 'Fourth')
    sc._dump_all()
    print sc.pop(0)
    sc._dump_all()
    print sc.pop(1)
    sc._dump_all()
```

b)Implement Queue

```
# Class Queue to represent a queue
class Queue:

    # __init__ function
    def __init__(self, capacity):
        self.front = self.size = 0
        self.rear = capacity -1
        self.Q = [None]*capacity
        self.capacity = capacity

    # Queue is full when size becomes
    # equal to the capacity
    def isFull(self):
        return self.size == self.capacity

    # Queue is empty when size is 0
    def isEmpty(self):
        return self.size == 0

    # Function to add an item to the queue.
    # It changes rear and size
    def EnQueue(self, item):
        if self.isFull():
            print("Full")
            return
        self.rear = (self.rear + 1) % (self.capacity)
        self.Q[self.rear] = item
        self.size = self.size + 1
        print("%s enqueued to queue"  %str(item))

    # Function to remove an item from queue.
    # It changes front and size
    def DeQueue(self):
        if self.isEmpty():
            print("Empty")
            return

        print("%s dequeued from queue" %str(self.Q[self.front]))
        self.front = (self.front + 1) % (self.capacity)
        self.size = self.size -1

    # Function to get front of queue
    def que_front(self):
        if self.isEmpty():
            print("Queue is empty")
        print("Front item is", self.Q[self.front])

    # Function to get rear of queue
    def que_rear(self):
        if self.isEmpty():
            print("Queue is empty")
        print("Rear item is",  self.Q[self.rear])


# Driver Code
if __name__ == '__main__':

    queue = Queue(30)
    queue.EnQueue(10)
    queue.EnQueue(20)
    queue.EnQueue(30)
    queue.EnQueue(40)
    queue.DeQueue()
    queue.que_front()
    queue.que_rear()
```

3.4 INPUT/OUTPUT:

a)Stack operations

```
1) Push in stack
2) Pop from stack
3) Display stack
4) Exit
Enter choice:
1
Enter value to be pushed:
12
Enter choice:
1
Enter value to be pushed:
52
Enter choice:
1
Enter value to be pushed:
45
Enter choice:
1
Enter value to be pushed:
52
Stack Overflow
```

```
Enter choice:
2
The popped element is 45
Enter choice:
3
Stack elements are:52 12
Enter choice:
2
The popped element is 52
Enter choice:
3
Stack elements are:12
```

b) Queue operations

```
1) Insert element to queue
2) Delete element from queue
3) Display all the elements of queue
4) Exit
Enter your choice :
1
Insert the element in queue :
12
Enter your choice :
1
Insert the element in queue :
45
Enter your choice :
1
Insert the element in queue :
65
Enter your choice :
1
Queue Overflow
Enter your choice :
2
Element deleted from queue is: 0
Enter your choice:
52
Invalid choice
Enter your choice:
```

3.5 PRE-LAB VIVA QUESTIONS:

1. What are stacks and queues?
2. What is peek in stack?
3. What is a stack coding?
4. What are the real time applications of stack?

3.6 LAB ASSIGNMENT:

1. Write a Python program to Check Armstrong Number
2. Write a Python program to Find ASCII Value of a Character
3. Write a Python program to Check Leap Year

3.7 POST-LAB VIVA QUESTIONS:

1. What are the applications of stacks and queues?
2. Why is top in stack?
3. What does top do in stack?
4. What is stack with example?

WEEK-4

HASHING TECHNIQUES

4.1 OBJECTIVE:

To write a program to store k keys into an array of size n at the location computed using a hash function, loc =key % n, where k<=n and k takes values from [1 to m], m>n. To handle the collisions use the following collision resolution techniques

a. Linear probing

4.2 RESOURCES:

Python 3.4

4.3 PROGRAM LOGIC:

```
import random

def linear_probe(n, r_list):

   """    a linear probe checks every element in r_list

   """

   for ix in range(len(r_list)):

      if n == r_list[ix]:

         if test_print:

            print("ix = %s" % ix)

         return True

   return False

def quadratic_probe(n, r_list):

   """    a quadratic probe checks the r_list every n^2 element

   if not found it shifts the probing

   needs work to reach all index values!!!!

   """    # for test

   qix_list = []

   r_len = len(r_list)

   q_len = int(len(r_list)**0.5)
```

```
    shift = 0
    for shift in range(0, r_len):
        for ix in range(shift, q_len):
            qix = ix**2 + shift
            qix_list.append(qix)
            if test_print:
                print("ix=%s  shift=%s  qix=%s" % (ix, shift, qix))
            if r_list[qix] == n:
                if test_print:
                    print(sorted(qix_list))
                return True
    if test_print:
        print(sorted(qix_list))
    return False
    # for debugging set to True
test_print = True
# create a list of count random integers in the range low to high-1
low = 100  #1000
high = 120  #12000
count = 17  #9000
r_list = random.sample(range(low, high), count)
# test the first 10 elements
print(r_list[:10])
#print(len(r_list))
# now create a another randomm integer in the range low to high-1
```

```
n = random.randrange(low, high)

# testing ...

print("n = %s" % n)

# and probe the list for collision (does the value exist?)

if linear_probe(n, r_list):

    print("linear probe found collision")

if quadratic_probe(n, r_list):

    print("qudratic probe found collision")
```

4.4 INPUT/OUTPUT:

```
3
4
1
0
['foo', 1, None, 73, 93]
4
True
False
```

4.5 PRE LAB VIVA QUESTIONS:

1. How many parameters can a resize method take?
2. How do you define a set in Python?
3. How do you define a set?

4.6 LAB ASSIGNMENT

1. Write a Python program to Swap two numbers.
2. Write a Python program to find factorial of a number.
3. Write a Python program to find largest number among three numbers.

4.7 POST LAB VIVA QUESTIONS:

1. What is Auto Type Python?
2. Is set in Python sorted?
3. How do you clear a set in Python?

WEEK-5
APPLICATIONS OF STACK

5.1 OBJECTIVE:

Write C programs for the following:

a. Uses Stack operations to convert infix expression into post fix expression.

b. Uses Stack operations for evaluating the post fix expression.

5.2 RESOURCES:

Python 3.4

5.3 PROGRAM LOGIC:

a) convert infix expression into post fix expression

```
# Class to convert the expression
class Evaluate:

    # Constructor to initialize the class variables
    def __init__(self, capacity):
        self.top = -1
        self.capacity = capacity
        # This array is used a stack
        self.array = []

    # check if the stack is empty
    def isEmpty(self):
        return True if self.top == -1 else False

    # Return the value of the top of the stack
    def peek(self):
        return self.array[-1]

    # Pop the element from the stack
    def pop(self):
        if not self.isEmpty():
            self.top -= 1
            return self.array.pop()
        else:
            return "$"

    # Push the element to the stack
    def push(self, op):
        self.top += 1
        self.array.append(op)

      # The main function that converts given infix expression
    # to postfix expression
    def evaluatePostfix(self, exp):

        # Iterate over the expression for conversion
        for i in exp:

            # If the scanned character is an operand
            # (number here) push it to the stack
            if i.isdigit():
                self.push(i)

            # If the scanned character is an operator,
            # pop two elements from stack and apply it.
            else:
                val1 = self.pop()
                val2 = self.pop()
                self.push(str(eval(val2 + i + val1)))

        return int(self.pop())

 # Driver program to test above function
exp = "231*+9-"
obj = Evaluate(len(exp))
print "postfix evaluation: %d"%(obj.evaluatePostfix(exp))
```

5.4 INPUT/OUTPUT:

a) convert infix expression into post fix expression

```
abcd^e-fgh*+^*+i-
```

b)evaluating the post fix expression

```
Enter postfix expression, press right parenthesis ')' for end expression : 456*+)
Result of expression evaluation : 34
```

5.5 PRE-LAB VIVA QUESTIONS:

1. How do you sort a pair in Python?
2. Which sorting is best and why?
3. What is merge sort and how it works

5.6 LAB ASSIGNMENT:

1. Write a Python program to check whether a number is palindrome or not.
2. Write a Python program to create pyramid and pattern.
3. Write a Python program to find all roots of a quadratic equation

5.6 POST-LAB VIVA QUESTIONS:

1. What are the types of sorting?
2. What is sorting with example?
3. Which sort is best?

WEEK-6
BINARY SEARCH TREE

6.1 OBJECTIVE:

Write a program for Binary Search Tree to implement following operations:

a. Insertion
b. Deletion
   i. Delete node with only child
   ii. Delete node with both children
c. Finding an element
d. Finding Min element
e. Finding Max element
f. Left child of the given node
g. Right child of the given node
h. Finding the number of nodes, leaves nodes, full nodes, ancestors, descendants.

6.2 RESOURCES:

Python 3.4

6.3 PROGRAM LOGIC:

```
# A binary tree node has data, pointer to left child
# and a pointer to right child
class Node:
    def __init__(self,data):
        self.data = data
        self.left = None
        self.right = None

# Function to insert a new node in BST
def insert(root, data):

    # 1. If the tree is empty, return a new,
    # single node
    if not root:
        return Node(data)

    # 2. Otherwise, recur down the tree
    if data < root.data:
        root.left = insert(root.left, data)
    if data > root.data:
        root.right = insert(root.right, data)

    # return the (unchanged) node pointer
    return root

# Function to find the node with maximum value
# i.e. rightmost leaf node
def maxValue(root):
    current = root

    #loop down to find the rightmost leaf
    while(current.right):
        current = current.right
    return current.data

# Driver code
if __name__=='__main__':
    root=None
    root = insert(root,2)
    root = insert(root,1)
    root = insert(root,3)
    root = insert(root,6)
    root = insert(root,5)
    print("Maximum value in BST is {}".format(maxValue(root)))
```

6.4 INPUT/OUTPUT:

```
20
30
40
50
60
70
80
```

6.5 PRE-LAB VIVA QUESTIONS:

1. What is the index number of the last element of an array with 9 elements?
2. What is array data structure in Python?
3. What is Python language interview question?

6.6 LAB ASSIGNMENT:

1. Write a Python programs to find GCD of two numbers
2. Write a Python programs to addition of two matrix
3. Write a Python programs to reverse array element using function

6.7 POST-LAB VIVA QUESTIONS:

1. What is array in data structure with example?
2. What is array with example?
3. What is array and its types?

WEEK-7
DISJOINT SET OPERATIONS

7.1 OBJECTIVE:

a. Write a program to implement Make_Set, Find_Set and Union functions for Disjoint Set Data Structure for a given undirected graph G(V,E) using the linked list representation with simple implementation of Union operation.

7.2 RESOURCES:

Python 3.4

7.3 PROGRAM LOGIC:

```
 //finding root of an element.
int root(int Arr[ ],int i)
{
    while(Arr[ i ] != i)           //chase parent of current element until it reaches root.
    {
     i = Arr[ i ];
    }
    return i;
}

/*modified union function where we connect the elements by changing the root of one of the
element */

int union(int Arr[ ] ,int A ,int B)
{
    int root_A = root(Arr, A);
    int root_B = root(Arr, B);
    Arr[ root_A ] = root_B ;       //setting parent of root(A) as root(B).
}
bool find(int A,int B)
{
    if( root(A)==root(B) )       //if A and B have same root,means they are connected.
    return true;
    else
    return false;
}
```

7.4 INPUT/OUTPUT:

```
graph contains cycle
```

7.5 PRE-LAB VIVA QUESTIONS:

1. What is the difference between map and set in Python?
2. What is the use of maps in Python?
3. What is the difference between map and Multimap?

7.6 LAB ASSIGNMENT:

1. Write a Python program to Compare Two Strings.
2. Write a Python program to Calculate Area and Perimeter of Rectangle.
3. Write a Python program to swap two number with the help of third variable.

7.7 POST-LAB VIVA QUESTIONS:

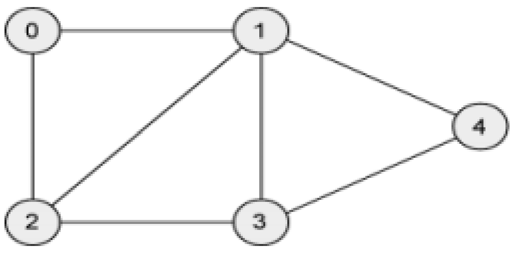
1. What is a multiset in Python ?
2. What is unordered set in Python?

WEEK-8

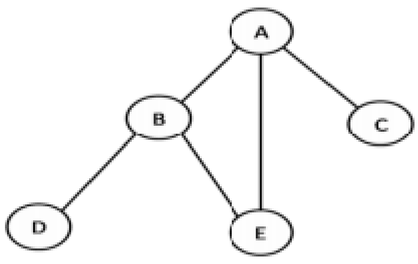
GRAPH TRAVERSAL TECHNIQUES

8.1 OBJECTIVE:

a. To print all the nodes reachable from a given starting node in a digraph using BFS method.



b. To check whether a given graph is connected or not using DFS method.



8.2 RESOURCES:

Python 3.4

8.3 PROGRAM LOGIC:

a) BFS method

```
class Graph:
    def __init__(self):
        # dictionary containing keys that map to the corresponding vertex object
        self.vertices = {}

    def add_vertex(self, key):
        """Add a vertex with the given key to the graph."""
        vertex = Vertex(key)
        self.vertices[key] = vertex

    def get_vertex(self, key):
        """Return vertex object with the corresponding key."""
        return self.vertices[key]

    def __contains__(self, key):
        return key in self.vertices

    def add_edge(self, src_key, dest_key, weight=1):
        """Add edge from src_key to dest_key with given weight."""
        self.vertices[src_key].add_neighbour(self.vertices[dest_key], weight)

    def does_edge_exist(self, src_key, dest_key):
        """Return True if there is an edge from src_key to dest_key."""
        return self.vertices[src_key].does_it_point_to(self.vertices[dest_key])

    def __iter__(self):
        return iter(self.vertices.values())


class Vertex:
    def __init__(self, key):
        self.key = key
        self.points_to = {}

    def get_key(self):
        """Return key corresponding to this vertex object."""
        return self.key

    def add_neighbour(self, dest, weight):
        """Make this vertex point to dest with given edge weight."""
        self.points_to[dest] = weight

    def get_neighbours(self):
        """Return all vertices pointed to by this vertex."""
        return self.points_to.keys()

    def get_weight(self, dest):
        """Get weight of edge from this vertex to dest."""
        return self.points_to[dest]

    def does_it_point_to(self, dest):
        """Return True if this vertex points to dest."""
        return dest in self.points_to


class Queue:
    def __init__(self):
        self.items = []

    def is_empty(self):
        return self.items == []

    def enqueue(self, data):
        self.items.append(data)
    def dequeue(self):
        return self.items.pop(0)

 def find_all_reachable_nodes(vertex):
    """Return set containing all vertices reachable from vertex."""
    visited = set()
    q = Queue()
    q.enqueue(vertex)
    visited.add(vertex)
    while not q.is_empty():
        current = q.dequeue()
        for dest in current.get_neighbours():
            if dest not in visited:
                visited.add(dest)
                q.enqueue(dest)
    return visited


g = Graph()
print('Menu')
print('add vertex <key>')
print('add edge <src> <dest>')
print('reachable <vertex key>')
print('display')
print('quit')

while True:
    do = input('What would you like to do? ').split()

    operation = do[0]
    if operation == 'add':
        suboperation = do[1]
        if suboperation == 'vertex':
            key = int(do[2])
            if key not in g:
                g.add_vertex(key)
            else:
                print('Vertex already exists.')
        elif suboperation == 'edge':
            src = int(do[2])
            dest = int(do[3])
            if src not in g:
                print('Vertex {} does not exist.'.format(src))
            elif dest not in g:
                print('Vertex {} does not exist.'.format(dest))
            else:
                if not g.does_edge_exist(src, dest):
                    g.add_edge(src, dest)
            else:
                print('Edge already exists.')

    elif operation == 'reachable':
        key = int(do[1])
        vertex = g.get_vertex(key)
        reachable = find_all_reachable_nodes(vertex)
        print('All nodes reachable from {}:'.format(key),
              [v.get_key() for v in reachable])

    elif operation == 'display':
        print('Vertices: ', end='')
        for v in g:
            print(v.get_key(), end=' ')
        print()

        print('Edges: ')
        for v in g:
            for dest in v.get_neighbours():
                w = v.get_weight(dest)
                print('(src={}, dest={}, weight={}) '.format(v.get_key(),
                                                             dest.get_key(), w))
        print()

    elif operation == 'quit':
        break
```

b)DFS method

```
# Python    program to check if a given directed graph is strongly
# connected or not

from collections import defaultdict

#This class represents a directed graph using adjacency list representation
class Graph:

    def __init__(self,vertices):
        self.V= vertices #No. of vertices
        self.graph = defaultdict(list) # default dictionary to store graph

    # function to add an edge to graph
    def addEdge(self,u,v):
        self.graph[u].append(v)


    #A function used by isSC() to perform DFS
```

```
def DFSUtil(self,v,visited):

    # Mark the current node as visited
    visited[v]= True

    #Recur for all the vertices adjacent to this vertex
    for i in self.graph[v]:
        if visited[i]==False:
            self.DFSUtil(i,visited)


# Function that returns reverse (or transpose) of this graph
def getTranspose(self):

    g = Graph(self.V)

    # Recur for all the vertices adjacent to this vertex
    for i in self.graph:
        for j in self.graph[i]:
            g.addEdge(j,i)

    return g

# The main function that returns true if graph is strongly connected
def isSC(self):

    # Step 1: Mark all the vertices as not visited (For first DFS)
    visited =[False]*(self.V)

    # Step 2: Do DFS traversal starting from first vertex.
    self.DFSUtil(0,visited)

    # If DFS traversal doesnt visit all vertices, then return false
    if any(i == False for i in visited):
        return False

    # Step 3: Create a reversed graph
    gr = self.getTranspose()

    # Step 4: Mark all the vertices as not visited (For second DFS)
    visited =[False]*(self.V)

    # Step 5: Do DFS for reversed graph starting from first vertex.
    # Staring Vertex must be same starting point of first DFS
    gr.DFSUtil(0,visited)

    # If all vertices are not visited in second DFS, then
    # return false
    if any(i == False for i in visited):
            return False
          return True
  # Create a graph given in the above diagram
g1 = Graph(5)
g1.addEdge(0, 1)
g1.addEdge(1, 2)
g1.addEdge(2, 3)
g1.addEdge(3, 0)
g1.addEdge(2, 4)
g1.addEdge(4, 2)
print "Yes" if g1.isSC() else "No"
  g2 = Graph(4)
g2.addEdge(0, 1)
g2.addEdge(1, 2)
g2.addEdge(2, 3)
print "Yes" if g2.isSC() else "No"
```

INPUT/OUTPUT:

BFS

```
Case 1:
Menu
add vertex <key>
add edge <src> <dest>
bfs <vertex key>
display
quit
```

DFS

```
Case 1:
Menu
add vertex <key>
add edge <src> <dest>
dfs <vertex key>
display
quit
```

8.4 PRE-LAB VIVA QUESTIONS:

1. What is unordered set in Python?
2. What is the difference between set and unordered set?
3. How do you check if an element is in a set Python?

8.5 LAB ASSIGNMENT:

1. Write a program in Python to find the greatest number in two numbers with the help of if/else.
2. Write a Python program for union and intersection of two sorted arrays
3. Write a Python program for bubble sort

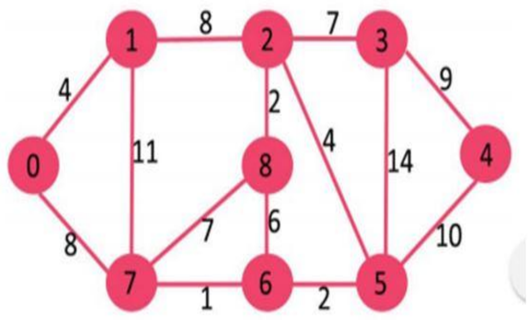
8.2 POST-LAB VIVA QUESTIONS:

1. What is the difference between ordered and unordered sets?
2. What is the difference between set and multiset in Python?
3. Explain whether set in Python sorted?

WEEK-9
SHORTEST PATHS ALGORITHM

9.1 OBJECTIVE:

From a given vertex in a weighted connected graph, find shortest paths to other vertices using Dijkstra's algorithm.



9.2 RESOURCES:

Python 3.4

9.3 PROGRAM LOGIC:

```
class Graph:
    def __init__(self):
        # dictionary containing keys that map to the corresponding vertex object
        self.vertices = {}

    def add_vertex(self, key):
        """Add a vertex with the given key to the graph."""
        vertex = Vertex(key)
        self.vertices[key] = vertex

    def get_vertex(self, key):
        """Return vertex object with the corresponding key."""
        return self.vertices[key]

    def __contains__(self, key):
        return key in self.vertices

    def add_edge(self, src_key, dest_key, weight=1):
        """Add edge from src_key to dest_key with given weight."""
        self.vertices[src_key].add_neighbour(self.vertices[dest_key], weight)

    def does_edge_exist(self, src_key, dest_key):
        """Return True if there is an edge from src_key to dest_key."""
        return self.vertices[src_key].does_it_point_to(self.vertices[dest_key])
    def __iter__(self):
        return iter(self.vertices.values())


class Vertex:
    def __init__(self, key):
        self.key = key
        self.points_to = {}

    def get_key(self):
        """Return key corresponding to this vertex object."""
        return self.key

    def add_neighbour(self, dest, weight):
        """Make this vertex point to dest with given edge weight."""
        self.points_to[dest] = weight

    def get_neighbours(self):
        """Return all vertices pointed to by this vertex."""
        return self.points_to.keys()

    def get_weight(self, dest):
        """Get weight of edge from this vertex to dest."""
        return self.points_to[dest]

    def does_it_point_to(self, dest):
        """Return True if this vertex points to dest."""
        return dest in self.points_to


def dijkstra(g, source):
    """Return distance where distance[v] is min distance from source to v.

    This will return a dictionary distance.

    g is a Graph object.
    source is a Vertex object in g.
    """
    unvisited = set(g)
    distance = dict.fromkeys(g, float('inf'))
    distance[source] = 0

    while unvisited != set():
        # find vertex with minimum distance
        closest = min(unvisited, key=lambda v: distance[v])

        # mark as visited
        unvisited.remove(closest)
        # update distances
        for neighbour in closest.get_neighbours():
            if neighbour in unvisited:
                new_distance = distance[closest] + closest.get_weight(neighbour)
                if distance[neighbour] > new_distance:
                    distance[neighbour] = new_distance

    return distance


g = Graph()
print('Undirected Graph')
print('Menu')
print('add vertex <key>')
print('add edge <src> <dest> <weight>')
print('shortest <source vertex key>')
print('display')
print('quit')

while True:
    do = input('What would you like to do? ').split()

    operation = do[0]
    if operation == 'add':
        suboperation = do[1]
        if suboperation == 'vertex':
            key = int(do[2])
            if key not in g:
                g.add_vertex(key)
            else:
                print('Vertex already exists.')
        elif suboperation == 'edge':
            src = int(do[2])
            dest = int(do[3])
            weight = int(do[4])
            if src not in g:
                print('Vertex {} does not exist.'.format(src))
            elif dest not in g:
                print('Vertex {} does not exist.'.format(dest))
            else:
                if not g.does_edge_exist(src, dest):
                    g.add_edge(src, dest, weight)
                    g.add_edge(dest, src, weight)
                else:
                    print('Edge already exists.')

    elif operation == 'shortest':
        key = int(do[1])
        source = g.get_vertex(key)
        distance = dijkstra(g, source)
        print('Distances from {}: '.format(key))
        for v in distance:
            print('Distance to {}: {}'.format(v.get_key(), distance[v]))
        print()

    elif operation == 'display':
        print('Vertices: ', end='')
        for v in g:
            print(v.get_key(), end=' ')
        print()

        print('Edges: ')
        for v in g:
            for dest in v.get_neighbours():
                w = v.get_weight(dest)
                print('(src={}, dest={}, weight={}) '.format(v.get_key(),
                                                             dest.get_key(), w))
        print()

    elif operation == 'quit':
        break
```

9.4 INPUT / OUTPUT :

```
Vertex   Distance from Source
0            0
1            4
2            12
3            19
4            21
5            11
6            9
7            8
8           14
```

9.5 PRE-LAB VIVA QUESTIONS:

1. How do you find the union and intersection of two arrays in Java?
2. How do you find the kth largest element in an unsorted array?
3. What is Union in array?

9.6 LAB ASSIGNMENT:

1. Write a Python program for insertion Sort?
2. Write a Python program to raise any number x to a positive power n?
3. Write a Python program to calculate square root of any number?

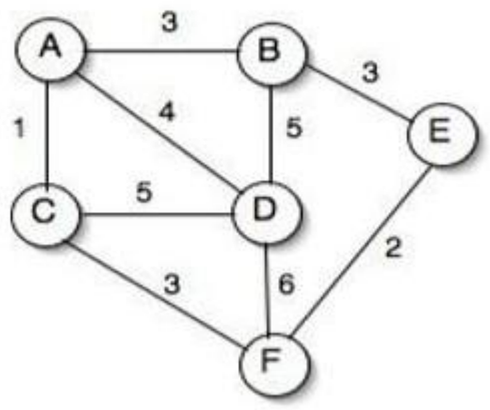
9.7 PRE-LAB VIVA QUESTIONS:

1. How do you find the intersection of two sets?
2. What are the examples of intersection of sets?

WEEK-10
MINIMUM COST SPANNING TREE

10.1 OBJECTIVE:

To find Minimum Cost Spanning Tree of a given undirected graph using Kruskal’s



10.2 RESOURCES:

Python 3.4

10.3 PROGRAM LOGIC:

```
    class Graph:
def __init__(self):
    # dictionary containing keys that map to the corresponding vertex object
    self.vertices = {}

def add_vertex(self, key):
    """Add a vertex with the given key to the graph."""
    vertex = Vertex(key)
    self.vertices[key] = vertex

def get_vertex(self, key):
    """Return vertex object with the corresponding key."""
    return self.vertices[key]

def __contains__(self, key):
    return key in self.vertices

def add_edge(self, src_key, dest_key, weight=1):
    """Add edge from src_key to dest_key with given weight."""
    self.vertices[src_key].add_neighbour(self.vertices[dest_key], weight)

def does_vertex_exist(self, key):
    return key in self.vertices

def does_edge_exist(self, src_key, dest_key):
        """Return True if there is an edge from src_key to dest_key."""
        return self.vertices[src_key].does_it_point_to(self.vertices[dest_key])

    def display(self):
        print('Vertices: ', end='')
        for v in self:
            print(v.get_key(), end=' ')
        print()

        print('Edges: ')
        for v in self:
            for dest in v.get_neighbours():
                w = v.get_weight(dest)
                print('(src={}, dest={}, weight={}) '.format(v.get_key(),
                                                     dest.get_key(), w))

    def __len__(self):
        return len(self.vertices)

    def __iter__(self):
        return iter(self.vertices.values())


class Vertex:
    def __init__(self, key):
        self.key = key
        self.points_to = {}

    def get_key(self):
        """Return key corresponding to this vertex object."""
        return self.key

    def add_neighbour(self, dest, weight):
        """Make this vertex point to dest with given edge weight."""
        self.points_to[dest] = weight

    def get_neighbours(self):
        """Return all vertices pointed to by this vertex."""
        return self.points_to.keys()

    def get_weight(self, dest):
        """Get weight of edge from this vertex to dest."""
        return self.points_to[dest]

    def does_it_point_to(self, dest):
        """Return True if this vertex points to dest."""
        return dest in self.points_to
```

```
def mst_krusal(g):
    """Return a minimum cost spanning tree of the connected graph g."""
    mst = Graph() # create new Graph object to hold the MST

    if len(g) == 1:
        u = next(iter(g)) # get the single vertex
        mst.add_vertex(u.get_key()) # add a copy of it to mst
        return mst

    # get all the edges in a list
    edges = []
    for v in g:
        for n in v.get_neighbours():
            # avoid adding two edges for each edge of the undirected graph
            if v.get_key() < n.get_key():
                edges.append((v, n))

    # sort edges
    edges.sort(key=lambda edge: edge[0].get_weight(edge[1]))

    # initially, each vertex is in its own component
    component = {}
    for i, v in enumerate(g):
        component[v] = i

    # next edge to try
    edge_index = 0

    # loop until mst has the same number of vertices as g
    while len(mst) < len(g):
        u, v = edges[edge_index]
        edge_index += 1

        # if adding edge (u, v) will not form a cycle
        if component[u] != component[v]:

            # add to mst
            if not mst.does_vertex_exist(u.get_key()):
                mst.add_vertex(u.get_key())
            if not mst.does_vertex_exist(v.get_key()):
                mst.add_vertex(v.get_key())
            mst.add_edge(u.get_key(), v.get_key(), u.get_weight(v))
            mst.add_edge(v.get_key(), u.get_key(), u.get_weight(v))

            # merge components of u and v
            for w in g:
                if component[w] == component[v]:
                    component[w] = component[u]
    return mst


g = Graph()
print('Undirected Graph')
print('Menu')
print('add vertex <key>')
print('add edge <src> <dest> <weight>')
print('mst')
print('display')
print('quit')

while True:
    do = input('What would you like to do? ').split()

    operation = do[0]
    if operation == 'add':
        suboperation = do[1]
        if suboperation == 'vertex':
            key = int(do[2])
            if key not in g:
                g.add_vertex(key)
            else:
                print('Vertex already exists.')
        elif suboperation == 'edge':
            src = int(do[2])
            dest = int(do[3])
            weight = int(do[4])
            if src not in g:
                print('Vertex {} does not exist.'.format(src))
            elif dest not in g:
                print('Vertex {} does not exist.'.format(dest))
            else:
                if not g.does_edge_exist(src, dest):
                    g.add_edge(src, dest, weight)
                    g.add_edge(dest, src, weight)
                else:
                    print('Edge already exists.')

    elif operation == 'mst':
        mst = mst_krusal(g)
        print('Minimum Spanning Tree:')
        mst.display()
        print()

    elif operation == 'display':
        g.display()
        print()
    elif operation == 'quit':
        break
```

10.4 INPUT / OUTPUT:

```
After sorting:
Weight   Src    Dest
1         7      6
2         8      2
2         6      5
4         0      1
4         2      5
6         8      6
7         2      3
7         7      8
8         0      7
8         1      2
9         3      4
10        5      4
11        1      7
14        3      5
```

10.5 PRE-LAB VIVA QUESTIONS:

1. How is linked list implemented?
2. What type of memory allocation is referred for linked lists?
3. What is the need for linked representation of lists?

10.6 LAB ASSIGNMENT:

1. Write a Python program to insert an element in an array
2. Write a Python program to do linear search in an array
3. Write a Python program to print three numbers in descending order

10.7 POST-LAB VIVA QUESTIONS:

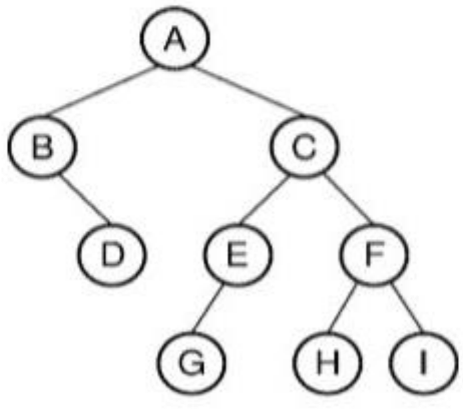
1. How many pointers are required to implement a simple linked list?
2. What are linked lists good for?
3. What are the advantages of linked list?

WEEK-11

TREE TRAVESRSALS

11.1 OBJECTIVE:

To perform various tree traversal algorithms for a given tree.



11.2 RESOURCES:

Python 3.4

11.3 PROGRAM LOGIC:

```
# Python    program to for tree traversals

# A class that represents an individual node in a
# Binary Tree
class Node:
    def __init__(self,key):
        self.left = None
        self.right = None
        self.val = key
# A function to do inorder tree traversal
def printInorder(root):

    if root:

        # First recur on left child
        printInorder(root.left)

        # then print the data of node
        print(root.val),

        # now recur on right child
        printInorder(root.right)
```

```
# A function to do postorder tree traversal
def printPostorder(root):

   if root:

        # First recur on left child
        printPostorder(root.left)

        # the recur on right child
        printPostorder(root.right)

        # now print the data of node
        print(root.val),


# A function to do preorder tree traversal
def printPreorder(root):

    if root:

        # First print the data of node
        print(root.val),

        # Then recur on left child
        printPreorder(root.left)

        # Finally recur on right child
        printPreorder(root.right)


# Driver code
root = Node(1)
root.left      = Node(2)
root.right     = Node(3)
root.left.right  = Node(4)
root.right.left = Node(5)
root.right.right = Node(6)
root.right.left.left = Node(7)
root.right.right.left = Node(8)
root.right.right.right = Node(9)
print("Preorder traversal of binary tree is")
printPreorder(root)

print("\nInorder traversal of binary tree is")
printInorder(root)

print("\nPostorder traversal of binary tree is")
printPostorder(root)
```

11.4 INPUT/OUTPUT:

```
Inorder traversal
5 ->12 ->6 ->1 ->9 ->
Preorder traversal
1 ->12 ->5 ->6 ->9 ->
Postorder traversal
5 ->6 ->12 ->9 ->1 ->
```

11.5 PRE-LAB VIVA QUESTIONS:

1. How do you find permutations?
2. What is permutations and combination?
3. Where permutation and combination is used?

11.6 LAB ASSIGNMENT:

1. Write a Python program to check whether a number is odd or even.
2. Write a Python program to print fibonacci series.
3. Write a Python program to Count no. of words in a string.

11.7 POST-LAB VIVA QUESTIONS:

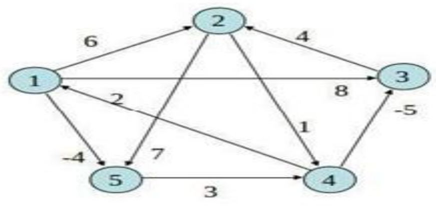
1. How do you tell if a question is a permutation or combination?
2. What is permutation formula?
3. What is permutation example?

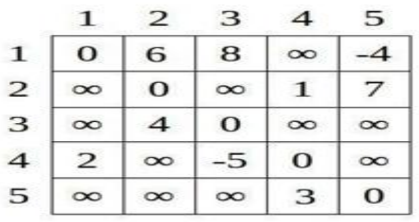
WEEK-12

ALL PAIRS SHORTEST PATHS

12.1 OBJECTIVE:

To implement All-Pairs Shortest Paths Problem using Floyd's algorithm.





	1	2	3	4	5
1	0	6	8	∞	-4
2	∞	0	∞	1	7
3	∞	4	0	∞	∞
4	2	∞	-5	0	∞
5	∞	∞	∞	3	0

12.2 RESOURCES:

Python 3.4

12.3 PROGRAM LOGIC:

```
class Graph:
    def __init__(self):
        # dictionary containing keys that map to the corresponding vertex object
        self.vertices = {}

    def add_vertex(self, key):
        """Add a vertex with the given key to the graph."""
        vertex = Vertex(key)
        self.vertices[key] = vertex

    def get_vertex(self, key):
        """Return vertex object with the corresponding key."""
        return self.vertices[key]

    def __contains__(self, key):
        return key in self.vertices

    def add_edge(self, src_key, dest_key, weight=1):
        """Add edge from src_key to dest_key with given weight."""
        self.vertices[src_key].add_neighbour(self.vertices[dest_key], weight)

    def does_edge_exist(self, src_key, dest_key):
        """Return True if there is an edge from src_key to dest_key."""
        return self.vertices[src_key].does_it_point_to(self.vertices[dest_key])

    def __len__(self):
        return len(self.vertices)

    def __iter__(self):
        return iter(self.vertices.values())
```

```
class Vertex:
    def __init__(self, key):
        self.key = key
        self.points_to = {}

    def get_key(self):
        """Return key corresponding to this vertex object."""
        return self.key

    def add_neighbour(self, dest, weight):
        """Make this vertex point to dest with given edge weight."""
        self.points_to[dest] = weight

    def get_neighbours(self):
        """Return all vertices pointed to by this vertex."""
        return self.points_to.keys()

    def get_weight(self, dest):
        """Get weight of edge from this vertex to dest."""
        return self.points_to[dest]

    def does_it_point_to(self, dest):
        """Return True if this vertex points to dest."""
        return dest in self.points_to


def floyd_warshall(g):
    """Return dictionaries distance and next_v.

    distance[u][v] is the shortest distance from vertex u to v.
    next_v[u][v] is the next vertex after vertex v in the shortest path from u
    to v. It is None if there is no path between them. next_v[u][u] should be
    None for all u.

    g is a Graph object which can have negative edge weights.
    """
    distance = {v:dict.fromkeys(g, float('inf')) for v in g}
    next_v = {v:dict.fromkeys(g, None) for v in g}

    for v in g:
        for n in v.get_neighbours():
            distance[v][n] = v.get_weight(n)
            next_v[v][n] = n

    for v in g:
         distance[v][v] = 0
         next_v[v][v] = None
    for p in g:
        for v in g:
            for w in g:
                if distance[v][w] > distance[v][p] + distance[p][w]:
                    distance[v][w] = distance[v][p] + distance[p][w]
                    next_v[v][w] = next_v[v][p]

    return distance, next_v


def print_path(next_v, u, v):
    """Print shortest path from vertex u to v.

    next_v is a dictionary where next_v[u][v] is the next vertex after vertex u
    in the shortest path from u to v. It is None if there is no path between
    them. next_v[u][u] should be None for all u.

    u and v are Vertex objects.
    """
    p = u
    while (next_v[p][v]):
        print('{} -> '.format(p.get_key()), end='')
        p = next_v[p][v]
    print('{} '.format(v.get_key()), end='')


g = Graph()
print('Menu')
print('add vertex <key>')
print('add edge <src> <dest> <weight>')
print('floyd-warshall')
print('display')
print('quit')

while True:
    do = input('What would you like to do? ').split()

    operation = do[0]
    if operation == 'add':
        suboperation = do[1]
        if suboperation == 'vertex':
            key = int(do[2])
            if key not in g:
                g.add_vertex(key)
            else:
                print('Vertex already exists.')
        elif suboperation == 'edge':
            src = int(do[2])
            dest = int(do[3])
            weight = int(do[4])
            if src not in g:
                print('Vertex {} does not exist.'.format(src))
            elif dest not in g:
                print('Vertex {} does not exist.'.format(dest))
            else:
                if not g.does_edge_exist(src, dest):
                    g.add_edge(src, dest, weight)
                else:
                    print('Edge already exists.')

    elif operation == 'floyd-warshall':
        distance, next_v = floyd_warshall(g)
        print('Shortest distances:')
        for start in g:
            for end in g:
                if next_v[start][end]:
                    print('From {} to {}: '.format(start.get_key(),
                                                   end.get_key()),
                          end = '')
                    print_path(next_v, start, end)
                    print('(distance {})'.format(distance[start][end]))

    elif operation == 'display':
        print('Vertices: ', end='')
        for v in g:
            print(v.get_key(), end=' ')
        print()

        print('Edges: ')
        for v in g:
            for dest in v.get_neighbours():
                w = v.get_weight(dest)
    print('(src={}, dest={}, weight={}) '.format(v.get_key(), dest.get_key(), w))
        print()

    elif operation == 'quit':
        break
```

12.4 INPUT/OUTPUT:

Shortest distance matrix

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

12.5 PRE-LAB VIVA QUESTIONS:

1. What is lexicographical order in string?
2. What does Lexicographically greater mean?

3. How do you compare two strings lexicographically?

12.6 LAB ASSIGNMENT:

1. Write a Python program to find length of a string.
2. Write a Python program to find sum of square of n natural numbers.
3. Write a Python program to find sum of digits of a number.

12.7 POST-LAB VIVA QUESTIONS:

1. What is lexicographic order of numbers?
2. What is lexicographic order example?
3. What does lexicographical mean?