Department Of Computer Engineering

Data Structure and Algorithm Lab Manual

SUBMITTED TO THE DEPARTMENT OF COMPUTER ENGINEERING

AISSMS IOIT

SE COMPTER ENGINEERING

SUBMITTED BY

Harsh A Shah

ERP No.- 59

Teams No.-25

TABLE OF CONTENTS

EXPERIMENT – 1

AIM:

Consider telephone book database of N clients. Make use of a hash table implementation to quickly look up client‘s telephone number. Make use of two collision handling techniques and compare them using number of comparisons required to find a set of telephone numbers.

OBJECTIVE:

1) To understand working of hash table.
2) To implement program using Chaining method and Addressing method.

THEORY:

Hash Table:

The Hash table data structure stores elements in key-value pairs where

- Key- unique integer that is used for indexing the values
- Value - data that are associated with keys.

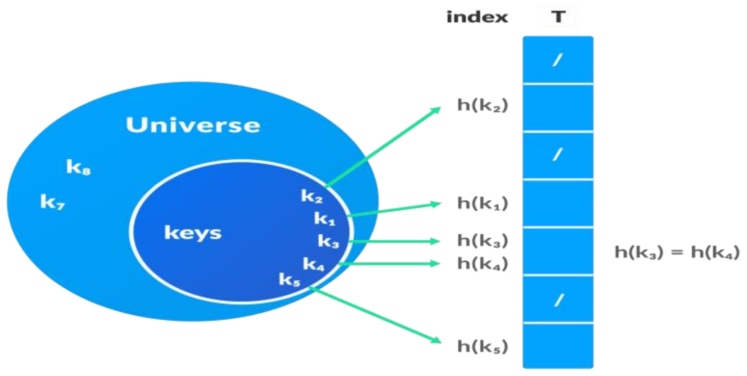
Hashing (Hash Function):

In a hash table, a new index is processed using the keys. And, the element corresponding to that key is stored in the index. This process is called hashing.

Let k be a key and *h(x)* be a hash function.

Here, *h(k)* will give us a new index to store the element linked with *k*.

Hashing is a technique of mapping a large set of arbitrary data to tabular indexes using a hash function. It is a method for representing dictionaries for large datasets.



Hash Collision:

When the hash function generates the same index for multiple keys, there will be a conflict (what value to be stored in that index). This is called a hash collision.

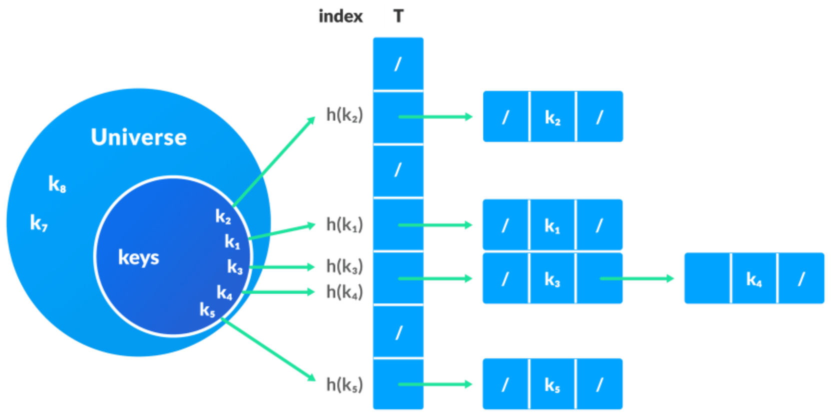
We can resolve the hash collision using one of the following techniques.

1. Collision resolution by chaining
2. Open Addressing: Linear/Quadratic Probing and Double Hashing

Collision resolution by chaining

In chaining, if a hash function produces the same index for multiple elements, these elements are stored in the same index by using a doubly-linked list.

If *j* is the slot for multiple elements, it contains a pointer to the head of the list of elements. If no element is present, *j* contains *NIL*.



Open Addressing

Unlike chaining, open addressing doesn't store multiple elements into the same slot. Here, each slot is either filled with a single key or left *NIL*.

Different techniques used in open addressing are:

•***Linear Probing***

In linear probing, collision is resolved by checking the next slot.

h(k, i) = (h'(k) + i) % m

where*i = {0, 1, ….}*, *h'(k)* is a new hash function

If a collision occurs at *h(k, 0)*, then *h(k, 1)* is checked. In this way, the value of *i* is incremented linearly.

The problem with linear probing is that a cluster of adjacent slots is filled. When inserting a new element, the entire cluster must be traversed. This adds to the time required to perform operations on the hash table.

• ***Quadratic Probing***

It works similar to linear probing but the spacing between the slots is increased (greater than one) by using the following relation.

h(k, i) = (h'(k) + c1i + c2i2) % m

where,*c1* and *c2* are positive auxiliary constants,*i = {0, 1, ….}*

• ***Double hashing***

If a collision occurs after applying a hash function *h(k)*, then another hash function is calculated for finding the next slot.

h(k, i) = (h1(k) + ih2(k)) % m

Applications of Hash Table:

Hash tables are implemented where

1. constant time lookup and insertion is required
2. cryptographic applicationsindexing data is required

ALGORITHM:

```
1. Start / Run
2. selecting Chaining method .
3. Declare an array of a linked list with the hash table size.
4.Initialize an array of a linked list to NULL.
/// inserting :
5. Find hash key.
6. If chain[key] == NULL
    Make chain[key] points to the key node.
7.Otherwise(collision),
    Insert the key node at the end of the chain[key].
/// searching :
8. Get the value
9. Compute the hash key.
10. Search the value in the entire chain. i.e. chain[key].
11. If found, print "Search Found"
12. Otherwise, print "Search Not Found"
///Removing :
13. Get the value
14. Compute the key.
15. Using linked list deletion algorithm, delete the element from the chain[key].
   Linked List Deletion Algorithm: Deleting a node in the linked list
16. If unable to delete, print "Value Not Found"
17.selecting linear probing (addressing):
/// inserting
18.use hash function to find index for a record
19.If that spot is already in use, we use next available spot in a "higher" index.
20. Treat the hash table as if it is round, if you hit the end of the hash table, go back to
the front
///searching
21.use hash function to find index of where an item should be.
```

```
22.If it isn't there search records that records after that hash location (remember to
treat table as cicular) until either it found, or until an empty record is found. If there is
an empty spot in the table before record is found,
it means that the the record is not there.
/// removing :
23.determine the hash index of the record
24.find record and remove it making the spot empty
25. Stop
```

PROGRAM:

```
class hashing_chaining:
arr = []

def __init__(self, num):
self.MAX = num
self.arr = [[] for iin range(self.MAX)]

def get_hash(self, key):
    h = 0
for char in key:
        h += ord(char)
return h % self.MAX

def __setitem__(self, key, val):
    h = self.get_hash(key)
    found = False

    for idx, elemin enumerate(self.arr[h]):
if len(elem) == 2 and elem[0] == key:
self.arr[h][idx] = (key, val)
        found = True
        break

    if not found:
self.arr[h].append((key, val))  # not exist
```

```
def __getitem__(self, key):
        h = self.get_hash(key)
for elein self.arr[h]:
if ele[0] == key:
return ele[1]
return None

    def __delitem__(self, key):
        h = self.get_hash(key)
for index, key_valin enumerate(self.arr[h]):
if key_val[0] == key:
del self.arr[h][index]
print('Item Deleted ')
break
        else:
print('Item Not Found ')


class hashing_addressing:
def __init__(self, num):
self.MAX = num
self.arr = [None for iin range(self.MAX)]
print(len(self.arr))

def get_hash(self, key):
        hash = 0
for char in key:
            hash += ord(char)
return hash % self.MAX

def __getitem__(self, key):
count_jump = 0
h = self.get_hash(key)
if self.arr[h] is None:
return
prob_range = self.get_prob_range(h)
for prob_indexin prob_range:
count_jump += 1
```

```
element = self.arr[prob_index]
if element is None:
return
          if element[0] == key:
return element[1], count_jump

def __setitem__(self, key, val):
       h = self.get_hash(key)
if self.arr[h] is None:
self.arr[h] = (key, val)
else:
new_h = self.find_slot(key, h)
self.arr[new_h] = (key, val)
# print(self.arr)

def get_prob_range(self, index):
return [*range(index, len(self.arr))] + [*range(0, index)]

def find_slot(self, key, index):
prob_range = self.get_prob_range(index)
for prob_indexin prob_range:
if self.arr[prob_index] is None:
return prob_index
if self.arr[prob_index][0] == key:
return prob_index
raise Exception("Hashmap full")

def __delitem__(self, key):
       h = self.get_hash(key)
prob_range = self.get_prob_range(h)
for prob_indexin prob_range:
if self.arr[prob_index] is None:
print('Item Not Found ')
break  # item not found so return. You can also throw exception
if self.arr[prob_index][0] == key:
self.arr[prob_index] = None
print('Item Deleted ')
break
```

```
# print(self.arr)


if __name__ == '__main__':
try:
while True:
        userinput1 = int(input(
'\nWelcome to Hashing Table \nEnter which hashing table type to apply\n1.CHAINING
\n2.ADDRESSING \n3.EXIT\n==>'))

if userinput1 == 1:
          size1 = int(input("ENTER SIZE OF TABLE :"))
ch = hashing_chaining(size1)
while True:
            userinput2 = int(input("\n1.INSERT \n2.SEARCH\n3.Display HASHING
TABLE\n4.DELETE\n5.EXIT\n>>>> "))
if userinput2 == 1:
              numb = int(input("ENTER NUMBER OF CLIENT:"))
for iin range(numb):
                name1 = input("ENTER NAME OF CLIENT:")
                number1 = int(input("ENTER PHONE NUMBER OF CLIENT:"))
ch[name1] = number1
elifuserinput2 == 2:
              name1 = input("ENTER NAME OF CLIENT:")
print(ch[name1])
elifuserinput2 == 3:
print(ch.arr)
elifuserinput2 == 4:
              name1 = input("ENTER NAME OF CLIENT:")
del ch[name1]

elifuserinput2 == 5:
break
            else:
print("WRONG INPUT")

elifuserinput1 == 2:
          size2 = int(input("ENTER SIZE OF TABLE :"))
```

```
            ah = hashing_addressing(size2)
while True:
               userinput2 = int(
input("\n1.INSERT\n2.SEARCH\n3.Display HASHING TABLE\n4.DELETE\n5.EXIT\n>>>> "))

if userinput2 == 1:
                  numb = int(input("ENTER NUMBER OF CLIENT:"))
for iin range(numb):
                     name1 = input("ENTER NAME OF CLIENT:")
                     number1 = int(input("ENTER PHONE NUMBER OF CLIENT:"))
                     ah[name1] = number1

elifuserinput2 == 2:
                  name1 = input("ENTER NAME OF CLIENT:")
                  element, count = ah[name1]
print('Number is :', element, '\tRequired number of jumps:', count)

elifuserinput2 == 3:
print(ah.arr)

elifuserinput2 == 4:
                  name1 = input("ENTER NAME OF CLIENT:")
del ah[name1]

elifuserinput2 == 5:
break
               else:
print("WRONG INPUT")
elifuserinput1 == 3:
break
         else:
print('Wrong Input')
except Exception as e:
print('\nWrong input :', e)
```

OUTPUT:

```
Welcome to Hashing Table

Enter which hashing table type to apply

1.CHAINING

2.ADDRESSING

3.EXIT

==>1

ENTER SIZE OF TABLE :10


1.INSERT

2.SEARCH

3.Display HASHING TABLE

4.DELETE

5.EXIT

>>>> 1

ENTER NUMBER OF CLIENT:7

ENTER NAME OF CLIENT:Kaustubh

ENTER PHONE NUMBER OF CLIENT:9168100204

ENTER NAME OF CLIENT:Harsh

ENTER PHONE NUMBER OF CLIENT:9874563210

ENTER NAME OF CLIENT:Onasvee

ENTER PHONE NUMBER OF CLIENT:7410852963

ENTER NAME OF CLIENT:Akash

ENTER PHONE NUMBER OF CLIENT:3214056789
ENTER NAME OF CLIENT:KK1

ENTER PHONE NUMBER OF CLIENT:9875632104

ENTER NAME OF CLIENT:Lonewolf

ENTER PHONE NUMBER OF CLIENT:7531598240

ENTER NAME OF CLIENT:Orion

ENTER PHONE NUMBER OF CLIENT:9517538264


1.INSERT

2.SEARCH

3.Display HASHING TABLE

4.DELETE

5.EXIT

>>>> 2

ENTER NAME OF CLIENT:KK1

9875632104


1.INSERT

2.SEARCH

3.Display HASHING TABLE

4.DELETE

5.EXIT

>>>> 2

ENTER NAME OF CLIENT:Lonewolf

7531598240
```

```
1.INSERT

2.SEARCH

3.Display HASHING TABLE

4.DELETE

5.EXIT

>>>> 2

ENTER NAME OF CLIENT:Orion

9517538264


1.INSERT

2.SEARCH

3.Display HASHING TABLE

4.DELETE

5.EXIT

>>>> 2

ENTER NAME OF CLIENT:acbd

None


1.INSERT

2.SEARCH

3.Display HASHING TABLE

4.DELETE

5.EXIT
```

```
>>>> 3

[[], [('Onasvee', 7410852963)], [('Harsh', 9874563210)], [], [], [], [], [], [('Akash',
3214056789), ('Lonewolf', 7531598240)], [('Kaustubh', 9168100204), ('KK1', 9875632104),
('Orion', 9517538264)]]


1.INSERT

2.SEARCH

3.Display HASHING TABLE

4.DELETE

5.EXIT

>>>> 4

ENTER NAME OF CLIENT:KK1

Item Deleted


1.INSERT

2.SEARCH

3.Display HASHING TABLE

4.DELETE

5.EXIT

>>>> 3

[[], [('Onasvee', 7410852963)], [('Harsh', 9874563210)], [], [], [], [], [], [('Akash',
3214056789), ('Lonewolf', 7531598240)], [('Kaustubh', 9168100204), ('Orion',
9517538264)]]


1.INSERT

2.SEARCH
```

```
3.Display HASHING TABLE

4.DELETE

5.EXIT

>>>> 4

ENTER NAME OF CLIENT:Lonewolf

Item Deleted


1.INSERT

2.SEARCH

3.Display HASHING TABLE

4.DELETE

5.EXIT

>>>> 3

[[], [('Onasvee', 7410852963)], [('Harsh', 9874563210)], [], [], [], [], [], [('Akash', 3214056789)], [('Kaustubh', 9168100204), ('Orion', 9517538264)]]


1.INSERT

2.SEARCH

3.Display HASHING TABLE

4.DELETE

5.EXIT

>>>> 5


Welcome to Hashing Table
```

```
Enter which hashing table type to apply

1.CHAINING

2.ADDRESSING

3.EXIT

==>2

ENTER SIZE OF TABLE :10

10


1.INSERT

2.SEARCH

3.Display HASHING TABLE

4.DELETE

5.EXIT

>>>> 1

ENTER NUMBER OF CLIENT:7

ENTER NAME OF CLIENT:Kaustubh

ENTER PHONE NUMBER OF CLIENT:9168100204

ENTER NAME OF CLIENT:Harsh

ENTER PHONE NUMBER OF CLIENT:7894561230

ENTER NAME OF CLIENT:Onasvee

ENTER PHONE NUMBER OF CLIENT:7410852963

ENTER NAME OF CLIENT:Akash

ENTER PHONE NUMBER OF CLIENT:9517538264

ENTER NAME OF CLIENT:KK1
ENTER PHONE NUMBER OF CLIENT:9875632144

ENTER NAME OF CLIENT:Lonewolf

ENTER PHONE NUMBER OF CLIENT:8527413330

ENTER NAME OF CLIENT:Orion

ENTER PHONE NUMBER OF CLIENT:2583571593


1.INSERT

2.SEARCH

3.Display HASHING TABLE

4.DELETE

5.EXIT

>>>> 3

[('KK1', 9875632144), ('Onasvee', 7410852963), ('Harsh', 7894561230), ('Lonewolf',
8527413330), ('Orion', 2583571593), None, None, None, ('Akash', 9517538264),
('Kaustubh', 9168100204)]


1.INSERT

2.SEARCH

3.Display HASHING TABLE

4.DELETE

5.EXIT

>>>> 2

ENTER NAME OF CLIENT:Kaustubh

Number is : 9168100204   Required number of jumps: 1
```

```
1.INSERT

2.SEARCH

3.Display HASHING TABLE

4.DELETE

5.EXIT

>>>> 2

ENTER NAME OF CLIENT:Lonewolf

Number is : 8527413330  Required number of jumps: 6


1.INSERT

2.SEARCH

3.Display HASHING TABLE

4.DELETE

5.EXIT

>>>> 2

ENTER NAME OF CLIENT:KK1

Number is : 9875632144  Required number of jumps: 2


1.INSERT

2.SEARCH

3.Display HASHING TABLE

4.DELETE

5.EXIT

>>>> 2
```

```
ENTER NAME OF CLIENT:abcd

None


1.INSERT

2.SEARCH

3.Display HASHING TABLE

4.DELETE

5.EXIT

>>>> 5

Process finished with exit code 0
```

ANALYSIS:

Worst-Case Time Complexity (Linear Probing)
Find: O(n)
Insert: O(n)
Remove: O(n)

Average-Case Time Complexity (Linear Probing)
Find: O(1)
Insert: O(1)
Remove: O(1)

Best-Case Time Complexity (Linear Probing)
Find: O(1) — No collisions
Insert: O(1) — No collisions
Remove: O(1) — No collisions

Space Complexity (Linear Probing)
O(n)
Worst-Case Time Complexity (Separate Chaining)
Find: O(n) — If all the keys mapped to the same index (assuming Linked List)

Insert: O(n) — If all the keys mapped to the same index (assuming Linked List) and we check for duplicates
Remove: O(n) — If all the keys mapped to the same index (assuming Linked List)

Average-Case Time Complexity (Separate Chaining)
Find: O(1)
Insert: O(1)
Remove: O(1)

Best-Case Time Complexity (Separate Chaining)
Find: O(1) — No collisions
Insert: O(1) — No collisions
Remove: O(1) — No collisions

Space Complexity (Separate Chaining)
O(n) — Hash Tables typically have a capacity that is at most some constant multiplied by n
(the constant is predetermined), and each of our n nodes occupies O(1) space

CONCLUSION: Hence, we have studied and implemented hashing, We have studied different collision handling techniques on our telephone book database and have successfully implemented them.

EXPERIMENT – 2

AIM:

To create ADT that implement the "set" concept.
a. Add (newElement) -Place a value into the set
b. Remove (element) Remove the value
c. Contains (element) Return true if element is in collection
d. Size () Return number of values in collection Iterator () Return an iterator used to loop over collection
e. Intersection of two sets
f. Union of two sets
g. Difference between two sets
h.Subset

OBJECTIVE:

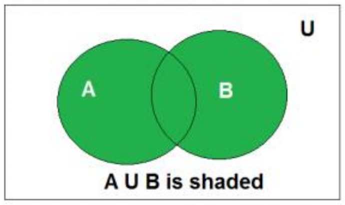
1. To understand the concept of Set.
2. To understand different operations on set.
3. To implement set in python.

THEORY:

A **Set** is an unordered collection of objects, known as elements or members of the set.
An element 'a' belong to a set A can be written as '$a \in A$', '$a \notin A$' denotes that a is not an element of the set A.

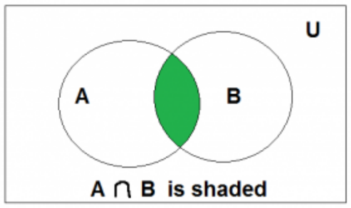
Representation of a Set
A set can be represented by various methods. 3 common methods used for representing set:
1. Statement form.
2. Roaster form or tabular form method.
3. Set Builder method.

Union
Union of the sets A and B, denoted by $A \cup B$, is the set of distinct elements that belong to set A or set B, or both.



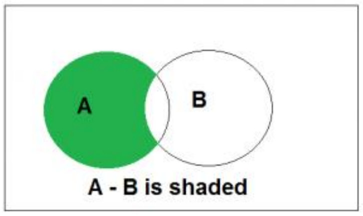
Venn diagram of A ∪ B

Intersection
The intersection of the sets A and B, denoted by A ∩ B, is the set of elements that belong to both A and B i.e. set of the common elements in A and B.



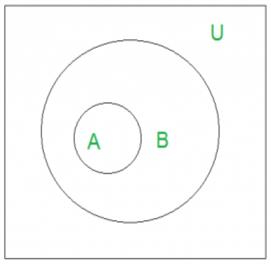
Venn diagram of A ∩ B

Set Difference
The difference between sets is denoted by 'A – B', which is the set containing elements that are in A but not in B. i.e., all elements of A except the element of B.



Above is the Venn Diagram of A-B.

Subset
A set A is said to be **subset** of another set B if and only if every element of set A is also a part of other set B.
Denoted by '⊆'.
'A ⊆ B ' denotes A is a subset of B.
To prove A is the subset of B, we need to simply show that if x belongs to A then x also belongs to B.
To prove A is not a subset of B, we need to find out one element which is part of set A but not belong to set B.



‘U’ denotes the universal set.
Above Venn Diagram shows that A is a subset of B.

ALGORITHM:

STEP1: Create a class Set which will create an empty set.

STEP 2: Accept elements from user and append them into the set.

STEP 3: If user wants to add an element, then accept the element and append it to the existing set.

STEP 4: If user wants to remove an element, then accept the element from the user and delete it from the set if it is present.

STEP 5: If user wants to search an element in the set, then accept the element and display “Element is present” if it is present in the set.

STEP 6: If user wants to see the elements in the set, then calculate the total elements present in the set and display the length.

STEP 7: If user wants intersection or union of set, then accept another set from user and display the intersection or union accordingly.

STEP 8: If user wants to calculate the difference of two set, then accept another set and display the difference.

STEP 9: If user wants to check if a set is a subset, then accept the set from the user and check if it is a subset or not.

STEP 10: Repeat steps 2-9 if until user wants to exit.

STEP 11: STOP.

PROGRAM

Menu.py

```
"""

Name: Harsh Shah

ERP No: 59

SE Comp-1

To create ADT that implement the "set" concept.

a. Add (newElement) -Place a value into the set

b. Remove (element) Remove the value

c. Contains (element) Return true if element is in collection

d. Size () Return number of values in collection Iterator () Return an iterator used to loop over collection

e. Intersection of two sets

f. Union of two sets

g. Difference between two sets

h.Subset

"""


from SetOperations import Set

def createSet():

    n=int(input("Enter number of Elements in set: "))

    s = Set(n)

    return s

choice = 0
```

```
print("Create Set A: ")

s1 = createSet()

print(str(s1))

while choice != 10:

    print("|-------------------|")

    print("| Menu              |")

    print("| 1.Add             |")

    print("| 2.Remove          |")

    print("| 3.Contains        |")

    print("| 4.Size            |")

    print("| 5.Intersection    |")

    print("| 6.Union           |")

    print("| 7.Difference      |")

    print("| 8.Subset          |")

    print("| 9.Proper Subset   |")

    print("| 10.Exit           |")

    print("|-------------------|")

    choice = int(input("Enter Choice: "))

    if choice==1:

        e = int(input("Enter Number to Add: "))

        s1.add(e)

        print(str(s1))

    elif choice==2:

        e = int(input("Enter Number to Remove: "))
        s1.remove(e)
        print(str(s1))
    elif choice==3:
        e = int(input("Enter Number to Search: "))
        if e in s1:
            print("Number Present in Set: ")
        else:
            print("Number is not Present in Set: ")
        print(str(s1))
    elif choice==4:
        print("Set Contains {} elements".format(len(s1)))
    elif choice==5:
        print("Create a Set B for doing Intersection Operation")
        s2 = createSet()
        s3 = s1.intersect(s2)
        print("Set A = "+str(s1))
        print("Set B = "+str(s2))
        print("Intersection = "+str(s3))
    elif choice==6:
        print("Create a Set B for doing Union Operation")
        s2 = createSet()
        s3 = s1.union(s2)
        print("Set A = "+str(s1))
        print("Set B = "+str(s2))
        print("Union = "+str(s3))
    elif choice==7:
        print("Create a Set B for calculating Set Difference")
        s2 = createSet()
        s3 = s1.difference(s2)
        print("Set A = "+str(s1))
        print("Set B = "+str(s2))
        print("Difference = "+str(s3))

    elif choice==8:
        print("Create a Set B for checking Subset or not")
        s2 = createSet()
        isSubset = s1.isSubsetOf(s2)
        print("Set A = "+str(s1))
        print("Set B = "+str(s2))
        if isSubset:
            print("Set B is the Subset of Set A")
        else:
            print("Set B is not a Subset of Set A")

    elif choice==9:
        print("Create a Set B for checking ProperSubset or not")
        s2 = createSet()
        isProperSubset = s1.isProperSubset(s2)
        print("Set A = "+str(s1))

        print("Set B = "+str(s2))

        if isProperSubset:

            print("Set B is the Proper Subset of Set A")

        else:

            print("Set B is not a Proper Subset of Set A")


    elif choice==10:

        break;


    elif choice<1 or choice>10:

        print("Please Enter Valid Choice")
```

SetOperations.py

```
class Set :

    # Creates an empty set instance.

    def __init__( self, initElementsCount ):

        self._s = []

        for i in range(initElementsCount) :

            e = int(input("Enter Element {}: ".format(i+1)))

            self.add(e)



    def get_set(self):
```

```
    return self._s


def __str__(self):
    string = "\n{ "
    for i in range(len(self.get_set())):
        string = string + str(self.get_set()[i])
        if i != len(self.get_set())-1:
            string = string + " , "
    string = string + " }\n"
    return string


# Returns the number of items in the set.
def __len__( self ):
    return len( self._s )


# Determines if an element is in the set.
def __contains__( self, e ):
    return e in self._s


# Determines if the set is empty.
def isEmpty( self ):
    return len(self._s) == 0


# Adds a new unique element to the set.
```

```
 def add( self, e ):
     if e not in self :
         self._s.append( e )

# Removes an e from the set.
 def remove( self, e ):
     if e in self.get_set():
         self.get_set().remove(e)

# Determines if this set is equal to setB.
 def __eq__( self, setB ):
     if len( self ) != len( setB ) :
         return False
     else :
         return self.isSubsetOf( setB )

# Determines if this set is a subset of setB.
 def isSubsetOf( self, setB ):
  for e in setB.get_set() :
      if e not in self.get_set() :
          return False
  return True

# Determines if this set is a proper subset of setB.
```

```
 def isProperSubset( self, setB ):
     if self.isSubsetOf(setB) and not setB.isSubsetOf(self):
         return True
     return False
# Creates a new set from the union of this set and setB.
 def union( self, setB ):
  newSet = self
  for e in setB :
      if e not in self.get_set() :
          newSet.add(e)
  return newSet
# Creates a new set from the intersection: self set and setB.
 def intersect( self, setB ):
     newSet = Set(0)
     for i in range(len(self.get_set())) :
         for j in range(len(setB.get_set())) :
             if self.get_set()[i] == setB.get_set()[j] :
                 newSet.add(self.get_set()[i])
     return newSet
 # Creates a new set from the difference: self set and setB.
 def difference( self, setB ):
     newSet = Set(0)
     for e in self.get_set() :
         if e not in setB.get_set():
             newSet.add(e)
     return newSet
  # Creates the iterator for traversing the list of items
 def __iter__( self ):
     return iter(self._s)
```

OUTPUT:

```
Create Set A:


Enter number of Elements in set: 5


Enter Element 1: 1


Enter Element 2: 2


Enter Element 3: 3


Enter Element 4: 4


Enter Element 5: 5


{ 1 , 2 , 3 , 4 , 5 }


|-------------------|
| Menu              |
| 1.Add             |
| 2.Remove          |
| 3.Contains        |
| 4.Size            |
| 5.Intersection    |
| 6.Union           |
| 7.Difference      |
| 8.Subset          |
| 9.Proper Subset   |
| 10.Exit           |
|-------------------|

Enter Choice: 1

Enter Number to Add: 6

{ 1 , 2 , 3 , 4 , 5 , 6 }

|-------------------|
| Menu              |
| 1.Add             |
| 2.Remove          |
| 3.Contains        |
| 4.Size          |

| 5.Intersection    |

| 6.Union         |

| 7.Difference      |

| 8.Subset        |

| 9.Proper Subset   |

| 10.Exit         |

|-------------------|


Enter Choice: 2


Enter Number to Remove: 6


{ 1 , 2 , 3 , 4 , 5 }


|-------------------|

| Menu            |

| 1.Add           |

| 2.Remove        |

| 3.Contains       |

| 4.Size          |

| 5.Intersection    |

| 6.Union         |

| 7.Difference      |
| 8.Subset         |

| 9.Proper Subset   |

| 10.Exit          |

|-------------------|


Enter Choice: 3


Enter Number to Search: 4

Number Present in Set:


{ 1 , 2 , 3 , 4 , 5 }


|-------------------|

| Menu             |

| 1.Add            |

| 2.Remove         |

| 3.Contains       |

| 4.Size           |

| 5.Intersection    |

| 6.Union          |

| 7.Difference      |

| 8.Subset         |

| 9.Proper Subset   |

| 10.Exit          |
|-------------------|


Enter Choice: 4

Set Contains 5 elements

|-------------------|

| Menu              |

| 1.Add             |

| 2.Remove          |

| 3.Contains        |

| 4.Size            |

| 5.Intersection    |

| 6.Union           |

| 7.Difference      |

| 8.Subset          |

| 9.Proper Subset   |

| 10.Exit           |

|-------------------|


Enter Choice: 5

Create a Set B for doing Intersection Operation


Enter number of Elements in set: 3


Enter Element 1: 5
Enter Element 2: 6


Enter Element 3: 7
Set A =
{ 1 , 2 , 3 , 4 , 5 }


Set B =
{ 5 , 6 , 7 }


Intersection =
{ 5 }


|-------------------|
| Menu              |
| 1.Add             |
| 2.Remove          |
| 3.Contains        |
| 4.Size            |
| 5.Intersection    |
| 6.Union           |
| 7.Difference      |
| 8.Subset          |
| 9.Proper Subset   |
| 10.Exit          |
|-------------------|


Enter Choice: 6
Create a Set B for doing Union Operation


Enter number of Elements in set: 3


Enter Element 1: 6


Enter Element 2: 7


Enter Element 3: 8
Set A =
{ 1 , 2 , 3 , 4 , 5 , 6 , 7 , 8 }


Set B =
{ 6 , 7 , 8 }


Union =
{ 1 , 2 , 3 , 4 , 5 , 6 , 7 , 8 }


|-------------------|
| Menu              |
| 1.Add            |

| 2.Remove         |

| 3.Contains       |

| 4.Size           |

| 5.Intersection   |

| 6.Union          |

| 7.Difference     |

| 8.Subset         |

| 9.Proper Subset  |

| 10.Exit          |

|------------------|


Enter Choice: 7

Create a Set B for calculating Set Difference


Enter number of Elements in set: 3


Enter Element 1: 6


Enter Element 2: 7


Enter Element 3: 8

Set A =

{ 1 , 2 , 3 , 4 , 5 , 6 , 7 , 8 }
```

```
Set B =
{ 6 , 7 , 8 }


Difference =
{ 1 , 2 , 3 , 4 , 5 }


|-------------------|
| Menu              |
| 1.Add             |
| 2.Remove          |
| 3.Contains        |
| 4.Size            |
| 5.Intersection    |
| 6.Union           |
| 7.Difference      |
| 8.Subset          |
| 9.Proper Subset   |
| 10.Exit           |
|-------------------|


Enter Choice: 8
Create a Set B for checking Subset or not
```

```
Enter number of Elements in set: 2


Enter Element 1: 1


Enter Element 2: 2

Set A =

{ 1 , 2 , 3 , 4 , 5 , 6 , 7 , 8 }


Set B =

{ 1 , 2 }


Set B is the Subset of Set A

|-------------------|

| Menu              |

| 1.Add             |

| 2.Remove          |

| 3.Contains        |

| 4.Size            |

| 5.Intersection    |

| 6.Union           |

| 7.Difference      |

| 8.Subset          |

| 9.Proper Subset   |

| 10.Exit           |
|-------------------|

Enter Choice: 9
Create a Set B for checking ProperSubset or not

Enter number of Elements in set: 2

Enter Element 1: 1

Enter Element 2: 2
Set A =
{ 1 , 2 , 3 , 4 , 5 , 6 , 7 , 8 }

Set B =
{ 1 , 2 }

Set B is the Proper Subset of Set A
|-------------------|
| Menu              |
| 1.Add             |
| 2.Remove          |
| 3.Contains        |
| 4.Size            |
| 5.Intersection    |
| 6.Union          |
| 7.Difference      |
| 8.Subset         |
| 9.Proper Subset   |
| 10.Exit          |
|-------------------|

Enter Choice: 10
```

ANALYSIS:

We require O(n) time to calculate the length of the set, add element in the set and remove element from the set.

We require $O(n^2)$ time to find union, intersection, difference and subset of two sets.

CONCLUSION: Thus we have created an ADT to implement Set operations. We have created a set and have performed different operations on it.

EXPERIMENT – 3

AIM:

A book consists of chapters, chapters consist of sections and sections consist of sub sections. Construct a tree and print the nodes. Find the time and space complexity of the program.

OBJECTIVE:

1. To learn the basics of tree data structure in C++ and apply it .

THEORY:

Trees:

Unlike Arrays, Linked Lists, Stack and queues, which are linear data structures, trees are hierarchical data structures.

Trees are non-linear hierarchical data structures. A tree is a collection of nodes connected to each other by means of “edges” which are either directed or undirected. One of the nodes is designated as “Root node” and the remaining nodes are called child nodes or the leaf nodes of the root node. In general, each node can have as many children but only one parent node.

Trees In C++

Tree with its various parts.

- **Root node:** This is the topmost node in the tree hierarchy. In the above diagram, Node A is the root node. Note that the root node doesn’t have any parent.
- **Leaf node:** It is the Bottom most node in a tree hierarchy. Leaf nodes are the nodes that do not have any child nodes. They are also known as external nodes. Nodes E, F, G, H and C in the above tree are all leaf nodes.
- **Subtree:** Subtree represents various descendants of a node when the root is not null. A tree usually consists of a root node and one or more subtrees. In the above diagram, (B-E, B-F) and (D-G, D-H) are subtrees.
- **Parent node:** Any node except the root node that has a child node and an edge upward towards the parent.
- **Ancestor Node:** It is any predecessor node on a path from the root to that node. Note that the root does not have any ancestors. In the above diagram, A and B are the ancestors of E.
- **Key:** It represents the value of a node.

- **Level:** Represents the generation of a node. A root node is always at level 1. Child nodes of the root are at level 2, grandchildren of the root are at level 3 and so on. In general, each node is at a level higher than its parent.
- **Path:** The path is a sequence of consecutive edges. In the above diagram, the path to E is A=>B->E.
- **Degree:** Degree of a node indicates the number of children that a node has. In the above diagram, the degree of B and D is 2 each whereas the degree of C is 0.

Tree types in c++:

- General tree
- Forest
- Binary tree
- Binary Search tree
- Expression tree

Binary Tree:

A Binary tree is a widely used tree data structure. When each node of a tree has at most two child nodes then the tree is called a Binary tree.

So a typical binary tree will have the following components:

- A left subtree
- A root node
- A right subtree

Binary Tree Types:

- Full Binary Tree
- Complete Binary tree
- Perfect Binary tree
- Balanced Binary

ALGORITHM:

```
1.  Start
    //Insertion

2.  If root is NULL then create root node and return
3.  If root exists then compare the data with node.data
        while until insertion position is located.

4.  If data is greater than node.data
        Goto right subtree

5.  Else. Goto left subtree
6.  End while, insert data


    //Searching

7.  If root.data is equal to search.data, return root
8.  Else, while data not found
9.  If data is greater than node.data, goto right subtree
10. Else ,goto left subtree
11. If data found, return node
12. End while
13. Return data not found


    //Display

14. Repeat until all nodes are traversed −
    Step 1 − Recursively traverse left subtree.

    Step 2 − Recursively traverse right subtree.

    Step 3 − Visit root node.

15. Stop
```

PROGRAM:

```
/*

Group B 1

Name: Harsh Shah
```

SE COMP 1

ERP No :59

A book consists of chapters, chapters consist of sections and sections consist of subsections.

Construct a tree and print the nodes.

```
*/
#include<iostream>
#include<stdio.h>
#include<queue>
using namespace std;
class Tree
{
typedef struct node
	{
		char data[10];
		struct node *left;
		struct node * right;
	}btree;
public:
btree *New,*root;
Tree();
void create();
void insert(btree *root,btree *New);
void display();
};
```

```
Tree::Tree()

{

root=NULL;

}

void Tree::create()

{

New=new btree;

New->left=New->right=NULL;

cout<<"\n\tEnter the Data: ";

cin>>New->data;

if(root==NULL)

          {

                    root=New;

          }

else

{

                    insert(root,New);

}

}

void Tree::insert(btree *root,btree *New)

{

char ans;

cout<<"\n\t"<<New->data<<" Want to Insert at "<<root->data<<" at Left(L) OR Right(R)";

          cin>>ans;
```

```
if(ans=='L'||ans=='l')

{

            if(root->left==NULL)

                  root->left=New;

else

insert(root->left,New);

         }

else

{

            if(root->right==NULL)

                  root->right=New;

else

insert(root->right,New);

         }

}

void Tree::display()

{

         int i=1;

         if(root==NULL){

                  cout<<"\n NULL Tree";

                  return;

         }

         queue<btree *> q;

         q.push(root);
```

```
		cout<<"\n\tLevelwise(BFS) Traversal\n";
		while(q.empty()==false)
		{
			btree *node=q.front();
			if(i==1)
				cout<<node->data<<"\n";
			if(i==2)
				cout<<node->data<<"\t";
			if(i==3)
				cout<<node->data<<"\n";
			if(i==4||i==5||i==6||i==7)
				cout<<node->data<<"\t";


			i++;
			q.pop();
			if(node->left!=NULL)
				q.push(node->left);
			if(node->right!=NULL)
				q.push(node->right);
		}
	}
	int main()
	{
	Tree tr;
```

```
int i=0;

do

{

if(i==0)

{

                    cout<<"\n\tEnter Chapter Name";

                    tr.create();

                    i++;

}

if(i==1||i==2)

{

                    cout<<"\n\tEnter Section Name";

tr.create();

i++;

}

if(i==3||i==4||i==5||i==6)

{

                    cout<<"\n\tEnter Sub-Section Name";

tr.create();

i++;

}

            if(i==7)

{

cout<<"\n tree is:";
```

```
                    tr.display();
break;
}
}while(1);
}
```

OUTPUT:

```
      Enter Chapter Name
      Enter the Data: Chapter

      Enter Section Name
      Enter the Data: Section1

      Section1 Want to Insert at Chapter at Left(L) OR Right(R)L

      Enter Section Name
      Enter the Data: Section2

      Section2 Want to Insert at Chapter at Left(L) OR Right(R)R

      Enter Sub-Section Name
      Enter the Data: sub1
```

```
sub1 Want to Insert at Chapter at Left(L) OR Right(R)l


sub1 Want to Insert at Section1 at Left(L) OR Right(R)l


Enter Sub-Section Name

Enter the Data: sub2


sub2 Want to Insert at Chapter at Left(L) OR Right(R)l


sub2 Want to Insert at Section1 at Left(L) OR Right(R)r


Enter Sub-Section Name

Enter the Data: sub3


sub3 Want to Insert at Chapter at Left(L) OR Right(R)r


sub3 Want to Insert at Section2 at Left(L) OR Right(R)l


Enter Sub-Section Name

Enter the Data: sub4


sub4 Want to Insert at Chapter at Left(L) OR Right(R)r


sub4 Want to Insert at Section2 at Left(L) OR Right(R)r
```

```
 tree is:
       Levelwise(BFS) Traversal
Chapter
Section1       Section2
sub1    sub2    sub3    sub4
--------------------------------
Process exited after 101.7 seconds with return value 0
Press any key to continue . . .
```

ANALYSIS:

Time Complexity:

1. Creating node: O(1)
2. Inserting Node : O(1)
3. Displaying Tree : O(n) {n=number of node tree have}

Space Complexity:

Space complexity: O(h)   {h= height of tree}

O(n)   (in worst case)

CONCLUSION: Hence, Trees are a non-linear hierarchical data structure that is used in many applications in the software field. Unlike linear data structures that have only one way to traverse the list, we can traverse trees in a variety of ways.

EXPERIMENT – 4

AIM:

Beginning with an empty binary search tree, Construct binary search tree by inserting the values in the order given. After constructing a binary tree –

1)Insert new node

2) Find number of nodes from longest path from root

3) Minimum data value found in the tree

4) change a tree so that the roles of the left & right pointers are swapped at every node

5) search a value.

OBJECTIVE:

1. To learn the basics of tree data structure in C++ and apply it.

THEORY:

Introduction to Tree:

Definition:

A tree T is a set of nodes storing elements such that the nodes have a parent-child relationship that satisfies the following

• if T is not empty, T has a special tree called the root that has no parent

• each node v of T different than the root has a unique parent node w; each node with parent w is a child of w

Recursive definition

• T is either empty

• or consists of a node r (the root) and a possibly empty set of trees whose roots are the children of r

Tree is a widely-used data structure that emulates a tree structure with a set of linked nodes. The tree graphical is represented most commonly as on Picture 1. The circles are the nodes and the edges are the links between them.

Trees are usually used to store and represent data in some hierarchical order. The data are stored in the nodes, from which the tree is consisted of.

A node may contain a value or a condition or represent a separate data structure or a tree of its own. Each node in a tree has zero or more child nodes, which are one level lower in the tree hierarchy (by convention, trees grow down, not up as they do in nature). A node that has a child is called the child's parent node (or ancestor node, or superior). A node has at most one parent. A node that has no child is called a leaf, and that node is of course at the bottom most Level of the tree. The height of a node is the length of the longest path to a leaf from that node.

The height of the root is the height of the tree. In other words, the "height" of tree is the "number

of levels" in the tree. Or more formal, the height of a tree is defined as follows:

1. The height of a tree with no elements is 0

2. The height of a tree with 1 element is 1

3. The height of a tree with > 1 element is equal to 1 + the height of its tallest subtree.

The depth of a node is the length of the path to its root (i.e., its root path). Every child node is always one level lower than his parent.

The topmost node in a tree is called the root node. Being the topmost node, the root node will not have parents. It is the node at which operations on the tree commonly begin (although some algorithms begin with the leaf nodes and work up ending at the root). All other nodes can be reached from it by following edges or links. (In the formal definition, a path from a root to a Node, for each different node is always unique). In diagrams, it is typically drawn at the top. In some trees, such as heaps, the root node has special properties.

A subtree is a portion of a tree data structure that can be viewed as a complete tree in itself. Any node in a tree T, together with all the nodes below his height, that are reachable from the node, comprise a subtree of T. The subtree corresponding to the root node is the entire tree; the subtree corresponding to any other node is called a proper subtree (in analogy to the term proper subset).

Every node in a tree can be seen as the root node of the subtree rooted at that node. An internal node or inner node is any node of a tree that has child nodes and is thus not a leaf node. There are two basic types of trees. In an unordered tree, a tree is a tree in a purely structural sense — that is to say, given a node, there is no order for the children of that node.

A tree on which an order is imposed — for example, by assigning different natural numbers to each child of each node — is called an ordered tree, and data structures built on them are called ordered tree data structures. Ordered trees are by far the most common form of tree data structure. Binary search trees are one kind of ordered tree.

Important Terms

Following are the important terms with respect to tree.

- Path − Path refers to the sequence of nodes along the edges of a tree.
- Root − The node at the top of the tree is called root. There is only one root per tree and one path from the root node to any node.
- Parent − Any node except the root node has one edge upward to a node called parent.
- Child − The node below a given node connected by its edge downward is called its child node.
- Leaf − The node which does not have any child node is called the leaf node.
- Subtree − Subtree represents the descendants of a node.
- Visiting − Visiting refers to checking the value of a node when control is on the node.
- Traversing − Traversing means passing through nodes in a specific order.
- Levels − Level of a node represents the generation of a node. If the root node is at level 0, then its next child node is at level 1, its grandchild is at level 2, and so on.
- keys − Key represents a value of a node based on which a search operation is to be carried out for a node.

Advantages of trees

Trees are so useful and frequently used, because they have some very serious advantages:

- Trees reflect structural relationships in the data

- Trees are used to represent hierarchies
- Trees provide an efficient insertion and searching
- Trees are very flexible data, allowing to move subtrees around with minimum effort.

ALGORITHM:

```
1. Start
   //Insertion

2. If root is NULL then create root node and return
3. If root exists then compare the data with node.data
       while until insertion position is located.

4. If data is greater than node.data
       Goto right subtree

5. Else. Goto left subtree
6. End while, insert data


   //Searching

7. If root.data is equal to search.data, return root
8. Else, while data not found
9. If data is greater than node.data, goto right subtree
10. Else ,goto left subtree
11. If data found, return node
12. End while
13. Return data not found.
    //Deletion

14. Starting at root, find the deepest and rightmost node in binary tree and node which we
    want to delete.
15. Replace the deepest rightmost node’s data with node to be deleted.
16. Then delete the deepest rightmost node.
    //Display

17. Repeat until all nodes are traversed −
    Step 1 − Recursively traverse left subtree.

    Step 2 − Recursively traverse right subtree.

    Step 3 − Visit root node.
```

//Minimum Value

18. Just traverse the node from root to left recursively until left is NULL. The node whose left is NULL is the node with minimum value.
    //Maximum Value
19. Just traverse the node from root to right recursively until right is NULL. The node whose right is NULL is the node with maximum value.
20. Stop

PROGRAM:

```
/*

Group B 2

Name: Harsh Shah

Class: SE comp 1

ERP No: 59

Beginning with an empty binary search tree, Construct binary search tree by inserting the
values in the order given. After constructing a binary tree –

I)Insert new node,

ii) Find number of nodes from longest path from root,

iii) Minimum data value found in the tree,

iv) change a tree so that the roles of the left & right pointers are swapped at every node,

v) search a value.

*/

#include<iostream>

usingnamespace std;

#defineCOUNT 10

//Building a BST by structure tree
```

```
structtree

{

int key,count,depth;

structtree *left,*right;

};

//Inserting a node in the tree. The function would not add duplicate nodes. But the count of
how many times the

//node appears in the tree is stored.

structtree *insert(structtree *root,intvalue)

{

if(root==NULL)

{

root=(structtree *)malloc(sizeof(structtree));

root->key=value;

root->count=1;

root->left=NULL;

root->right=NULL;

returnroot;

}

//Adding the count when found a duplicate node

elseif (value == root->key)

{

(root->count)++;

returnroot;
```

```
}

else

{

//Insert in right substree

if(root->key <value)

root->right=insert(root->right,value);

else

{

//Insert in left subtree

if(root->key >value)

root->left=insert(root->left,value);

}

}

returnroot;

}


//Deleting a node from tree

structtree *deletion(structtree *root,intvalue)

{

if(root == NULL)

returnroot;


if(root->key <value)

{
```

```
//The node to be deleted is in right subtree

root->right=deletion(root->right,value);

returnroot;

}

elseif(root->key >value)

{

//The node to be deleted is in the left subtree

root->left=deletion(root->left,value);

returnroot;

}

else

{

//Decreasing the count of deleted node

if (root->count > 1)

{

(root->count)--;

returnroot;

}

if(root->left == NULL)

{

structtree *temp=root->right;

deleteroot;

return temp;

}
```

```
elseif(root->right == NULL)

{

structtree *temp=root->left;

deleteroot;

return temp;

}


//Finding the sucessor

else

{

structtree *succparent=root->right;

structtree *succ=root->right;

while(succ->left != NULL)

{

succparent=succ;

succ=succ->left;

}

succparent->left=succ->right;

root->key=succ->key;

delete succ;

returnroot;

}

}
```

```
}

//Minimum Value in the tree
structtree *minimumval(structtree *root)
{
structtree *current=root;
while(current->left !=NULL)
current=current->left;
return current;
}

//Maximum Value in the tree
structtree *maximumval(structtree *root)
{
structtree *current=root;
while(current->right !=NULL)
current=current->right;
return current;
}

//Searching a node in the tree
structtree *search(structtree *root, intvalue)
{
if (root == NULL || root->key == value)
```

```
returnroot;

//Search in right subtree

if (root->key <value)
return search(root->right, value);

//Search in left subtree

return search(root->left, value);
}



//Finding depth of a node
int depth(structtree *root,intvalue, intlevel)
{

if(root == NULL)
return 0;

if(root->key == value)
returnlevel;
int temp = depth(root->left,value,level+1);
if(temp != 0)
```

```
return temp;


temp=depth(root->right,value,level+1);
return temp;
}


void print2DUtil(structtree *root, intlevel)
{
// Base case
if (root == NULL)
return;
// Increase distance between levels
level += COUNT;
// Process right child first
print2DUtil(root->right, level);
// Print current node after space
// count
cout<<endl;
for (int i = COUNT; i <level; i++)
cout<<"";
cout<<root->key<<"\n";
// Process left child
print2DUtil(root->left, level);
}
```

```
// Wrapper over print2DUtil()

void print2D(structtree *root)

{

// Pass initial space count as 0

print2DUtil(root, 0);

}

int main()

{

int choice=0,value=0,value1=0;

structtree *root=NULL,*searchh=NULL,*position=NULL;

do

{

//Menu for selecting the operation to perform

cout<<"\n\n";

cout<<"\n Binary Search Tree !!!!";

cout<<"\n [1] Insertion ";

cout<<"\n [2] Deletion ";

cout<<"\n [3] Search ";

cout<<"\n [4] Minimum Value ";

cout<<"\n [5] Maximum Value ";

cout<<"\n [6] Display ";

cout<<"\n [0] Exit ";

cout<<"\n Enter the choice: ";

cin>>choice;
```

```
switch(choice)

{

case 1:

cout<<"\n Insertion..!";

cout<<"\n Enter the element to be inserted: ";

cin>>value;

root=insert(root,value);

root->depth=depth(root,value,0);

cout<<"\n Depth of node "<<value<<" is: "<<root->depth;

break;

case 2:

cout<<"\n Deletion..!";

cout<<"\n Enter the element to be deleted: ";

cin>>value;

if(search(root,value) == NULL)

cout<<"\n Deletion Unsuccessful. Value does not exists in the tree!";

else

{

root->depth=depth(root,value,0);

cout<<"\n Depth of node "<<value<<" is: "<<root->depth;

root=deletion(root,value);

}

break;

case 3:
```

```
cout<<"\n Search..!";
cout<<"\n Enter the element to be searched: ";
cin>>value;
searchh=search(root,value);
if(searchh == NULL)
cout<<"\n Key not found!";
else
{
root->depth=depth(root,value,0);
cout<<"\n"<<" Key "<<searchh->key<<" Found!";
cout<<"\n Depth of node "<<value<<" is: "<<root->depth;
}
break;
case 4:
cout<<"\n Minimum Value..!";
if(root == NULL)
cout<<"\n No minimum values in empty tree";
else
{
value=minimumval(root)->key;
root->depth=depth(root,value,0);
cout<<"\n Smallest value in the tree: "<<value;
cout<<"\n Depth of node "<<value<<" is: "<<root->depth;
}
```

```
break;
case 5:
cout<<"\n Maximum Value..!";
if(root == NULL)
cout<<"\n No maximum values in empty tree";
else
{
value=maximumval(root)->key;
root->depth=depth(root,value,0);
cout<<"\n Largest value in the tree: "<<value;
cout<<"\n Depth of node "<<value<<" is: "<<root->depth;
}
break;
case 6:{
        cout<<"\n BST Display";
        print2D(root);
                                break;
                        }
case 0:
cout<<"\n Exiting..!";
break;
default:
cout<<"\n Invalid Choice!";
break;
```

```
}
}while(choice!=0);
return 0;
}
```

OUTPUT:

```
Binary Search Tree !!!!
 [1] Insertion
 [2] Deletion
 [3] Search
 [4] Minimum Value
 [5] Maximum Value
 [6] Display
 [0] Exit
 Enter the choice: 1

 Insertion..!
 Enter the element to be inserted: 20
 Depth of node 20 is: 0



 Binary Search Tree !!!!
 [1] Insertion
 [2] Deletion
```

```
[3] Search

[4] Minimum Value

[5] Maximum Value

[6] Display

[0] Exit

Enter the choice: 1


Insertion..!

Enter the element to be inserted: 10

Depth of node 10 is: 1



Binary Search Tree !!!!

[1] Insertion

[2] Deletion

[3] Search

[4] Minimum Value

[5] Maximum Value

[6] Display

[0] Exit

Enter the choice: 1


Insertion..!

Enter the element to be inserted: 30
```

```
Depth of node 30 is: 1



Binary Search Tree !!!!

[1] Insertion

[2] Deletion

[3] Search

[4] Minimum Value

[5] Maximum Value

[6] Display

[0] Exit

Enter the choice: 1

Insertion..!

Enter the element to be inserted: 5

Depth of node 5 is: 2



Binary Search Tree !!!!

[1] Insertion

[2] Deletion

[3] Search

[4] Minimum Value

[5] Maximum Value

[6] Display
```

```
[0] Exit

Enter the choice: 1


Insertion..!

Enter the element to be inserted: 25

Depth of node 25 is: 2




Binary Search Tree !!!!

[1] Insertion

[2] Deletion

[3] Search

[4] Minimum Value

[5] Maximum Value

[6] Display

[0] Exit

Enter the choice: 1


Insertion..!

Enter the element to be inserted: 67

Depth of node 67 is: 2




Binary Search Tree !!!!
```

```
[1] Insertion

[2] Deletion

[3] Search

[4] Minimum Value

[5] Maximum Value

[6] Display

[0] Exit

Enter the choice: 2


Deletion..!

Enter the element to be deleted: 25

Depth of node 25 is: 2


Binary Search Tree !!!!

[1] Insertion

[2] Deletion

[3] Search

[4] Minimum Value

[5] Maximum Value

[6] Display

[0] Exit

Enter the choice: 3


Search..!
```

```
Enter the element to be searched: 5

Key 5 Found!

Depth of node 5 is: 2



Binary Search Tree !!!!

[1] Insertion

[2] Deletion

[3] Search

[4] Minimum Value

[5] Maximum Value

[6] Display

[0] Exit

Enter the choice: 4


Minimum Value..!

Smallest value in the tree: 5

Depth of node 5 is: 2



Binary Search Tree !!!!

[1] Insertion

[2] Deletion

[3] Search
```

```
[4] Minimum Value

[5] Maximum Value

[6] Display

[0] Exit

Enter the choice: 5


Maximum Value..!

Largest value in the tree: 67

Depth of node 67 is: 2
```

ANALYSIS:

Time Complexity:

1. Creating node: O(1)
2. Inserting Node : O(1)
3. Displaying Tree : O(n) {n=number of node tree have}

Space Complexity:

Space complexity: O(h) {h= height of tree}

O(n) (in worst case)

CONCLUSION: Hence, Trees are a non-linear hierarchical data structure that is used in many applications in the software field. Unlike linear data structures that have only one way to traverse the list, we can traverse trees in a variety of ways.

EXPERIMENT – 5

AIM:

Construct an expression tree from the given prefix expression eg. +--a*bc/def and traverse it using post order traversal (non recursive) and then delete the entire tree.

OBJECTIVES:

1. To understand concept of Tree & Binary Tree.
2. To analyse the working of various Tree operations.

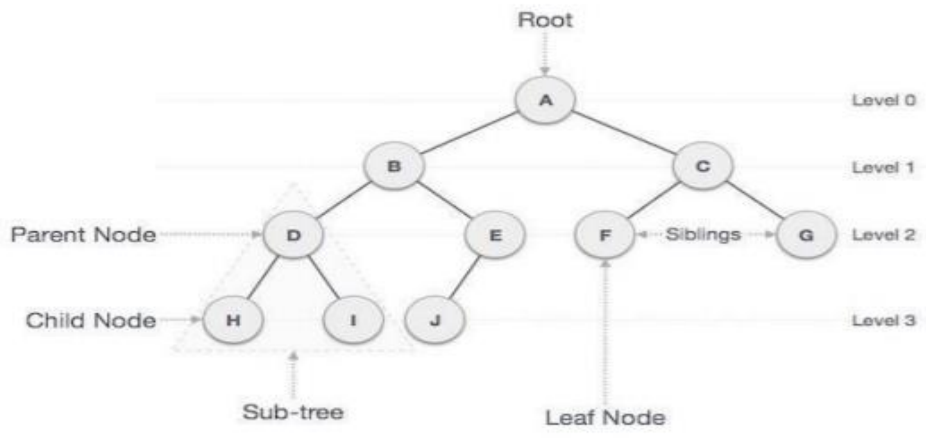
THEORY:

Tree

Tree represents the nodes connected by edges also a class of graphs that is acyclic is termed as Trees. Let us now discuss an important class of graphs called trees and its associated terminology.Trees are useful in describing any structure that involves hierarchy. Familiar examples of such structures are family trees, the hierarchy of positions in an organization, and so on.

Binary Tree

A binary tree is made of nodes, where each node contains a "left" reference, a "right" reference, and a data element. The topmost node in the tree is called the root.Every node (excluding a root) in a tree is connected by a directed edge from exactly one other node. This node is called a parent. On the other hand, each node can be connected to arbitrary number of nodes, called children. Nodes with no children are called leaves, or external nodes. Nodes which are not leaves are called internal nodes. Nodes with the same parent are called siblings.



Insert Operation

The very first insertion creates the tree. Afterwards, whenever an element is to be inserted, first locate its proper location. Start searching from the root node, then if the data is less than the key value, search for the empty location in the left subtree and insert the data. Otherwise, search for the empty location in the right subtree and insert the data.

Traversals

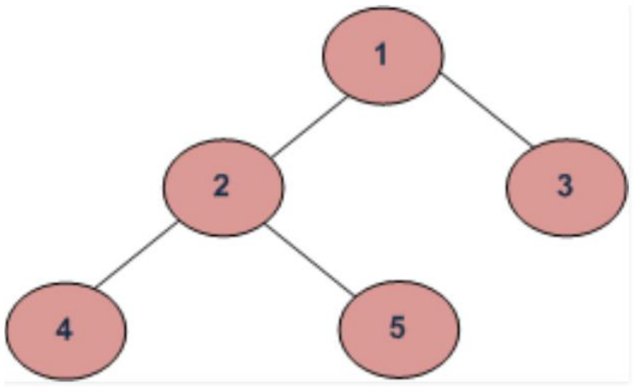
A traversal is a process that visits all the nodes in the tree. Since a tree is a non-linear data structure, there is no unique traversal. We will consider several traversal algorithms with we group in the following two kinds

- depth-first traversal
- breadth-first traversal

There are three different types of depth-first traversals, :

- Pre-order traversal - visit the parent first and then left and right children;
- In Order traversal - visit the left child, then the parent and the right child;
- Post-order traversal - visit left child, then the right child and then the parent;

There is only one kind of breadth-first traversal the level order traversal. This traversal visits nodes by levels from top to bottom and from left to right. As an example consider the following tree and its four traversals:



Depth First Traversals:
(a) Inorder (Left, Root, Right) : 4 2 5 1 3
(b) Preorder (Root, Left, Right) : 1 2 4 5 3
(c) Postorder (Left, Right, Root) : 4 5 2 3 1

ALGORITHM:

1. Algorithm to insert a node :
   Step 1 -
   Search for the node whose child node is to be inserted. This is a node at some level i, and a node is to be inserted at the level i +1 as either its left child or right child. This is the node after which the insertion is to be made.
   Step 2 -
   Link a new node to the node that becomes its parent node, that is, either the L child or the R child.

2. Algorithm to traverse a tree :
   In order traversal
   Until all nodes are traversed −
   Step 1 − Recursively traverse left subtree.
   Step 2 − Visit root node.
   Step 3 − Recursively traverse right subtree.

3. Pre-order traversal
   Until all nodes are traversed −
   Step 1 − Visit root node.
   Step 2 − Recursively traverse left subtree.
   Step 3 − Recursively traverse right subtree

4. Post-order traversal
   Until all nodes are traversed −
   Step 1 − Recursively traverse left subtree.
   Step 2 − Recursively traverse right subtree.
   Step 3 − Visit root node

5. Algorithm to copy one tree into another tree:
   Step 1 – if (Root == Null) Then return Null
   Step 2 –Tmp = new TreeNode
   Step 3 – Tmp->Lchild = TreeCopy(Root- >Lchild);
   Step 4 – Tmp->Rchild = TreeCopy(Root->Rchild);
   Step 5 – Tmp-Data = Root->Data;
   Then return

PROGRAM:

```
/*
Group B 3
Name : Harsh Shah
ERP no: 59
Construct an expression tree from the given prefix expression eg. +--a*bc/def and traverse it
using post order traversal (non recursive) and then delete the entire tree.
*/
#include<iostream>
#include<string.h>
#include<conio.h>
using namespace std;

struct node
{
	char data;
	node *left;
	node *right;
};

class tree
{	char prefix[20];
	public: node *top;
```

```
                    void expression(char []);

                    void display(node *);

                    void non_rec_postorder(node *);

                    void del(node *);


};


class stack1

{

          node *data[30];

          int top;

          public:

          stack1()

          {

                    top=-1;

          }

                    int empty()

                      {

                              if(top==-1)

                                        return 1;

                              return 0;

                      }

          void push(node *p)

                      {
```

```
                    data[++top]=p;
                }
        node *pop()
         {
                return(data[top--]);
         }
};

void tree::expression(char prefix[])
{char c;
stack1 s;
node *t1,*t2;
int len,i;
len=strlen(prefix);

        for(i=len-1;i>=0;i--){
                top=new node;
                top->left=NULL;
                top->right=NULL;

                if(isalpha(prefix[i]))
                {
                        top->data=prefix[i];
                        s.push(top);
                }
                else if(prefix[i]=='+'||prefix[i]=='*'||prefix[i]=='-'||prefix[i]=='/')
                {
                        t2=s.pop();
                        t1=s.pop();
                        top->data=prefix[i];
                        top->left=t2;
                        top->right=t1;
                        s.push(top);
}
        }
        top=s.pop();
}
void tree::display(node * root)
{
        if(root!=NULL)
        {       cout<<root->data;
                display(root->left);
                display(root->right);
         }
}
void tree::non_rec_postorder(node *top)
{
        stack1 s1,s2;    /*stack s1 is being used for flag . A NULL data implies that the right
subtree has not been visited */

        node *T=top;

        cout<<"\n";

        s1.push(T);

        while(!s1.empty())

        {

                T=s1.pop();

                s2.push(T);

                if(T->left!=NULL)

                s1.push(T->left);

                if(T->right!=NULL)

                s1.push(T->right);

        }

        while(!s2.empty())

        {

                top=s2.pop();

                cout<<top->data;

        }

}


void tree::del(node* node)

{

if (node == NULL) return;
```

```
/* first delete both subtrees */
del(node->left);
del(node->right);
/* then delete the node */
cout<<" Deleting node:"<<node->data;
delete(node);
}

int main()
{
	char expr[20];
	tree t;


	cout<<"Enter prefix  Expression: ";
	cin>>expr;
	cout<<"Prefix Expression is:"<<expr;
	cout<<"\n";
	t.expression(expr);
	cout<<"Postorder Traversal expression is:";
	t.non_rec_postorder(t.top);
}
```

OUTPUT:

CASE 1:

```
Enter prefix  Expression: +--a*bc/def

Prefix Expression is:+--a*bc/def

Postorder Traversal expression is:

abc*-de/-f+

--------------------------------

Process exited after 41.87 seconds with return value 0

Press any key to continue . . .
```

CASE 2:

```
Enter prefix  Expression: *-a/bc-/akl

Prefix Expression is:*-a/bc-/akl

Postorder Traversal expression is:

abc/-ak/l-*

--------------------------------

Process exited after 18.41 seconds with return value 0

Press any key to continue . . .
```

ANALYSIS:

Time Complexity:

1 Creating node: O(1)
2 Inserting Node : O(1)
3 Displaying Tree : O(n) {n=number of node tree have}

Space Complexity:

Space complexity: O(h) {h= height of tree}
O(n) (in worst case)

CONCLUSION: Thus we have studied the implementation of various Binary tree operations

EXPERIMENT – 6

AIM:

Represent a given graph using adjacency matrix/list to perform DFS and using adjacency list to perform BFS. Use the map of the area around the college as the graph. Identify the prominent land marks as nodes and DFS & BFS on that.

OBJECTIVE:

1. To understand DFS and BFS.
2. To implement program to represent graph using adjacency matrix and list.

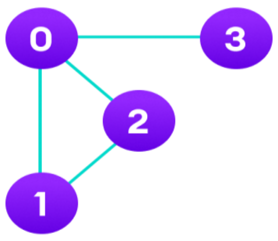
THEORY:

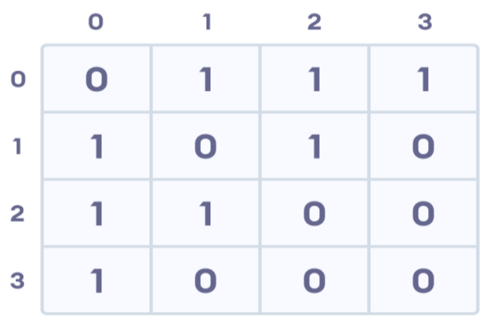
Adjacency Matrix:

An adjacency matrix is a square matrix used to represent a finite graph. The elements of the matrix indicate whether pairs of vertices are adjacent or not in the graph.

Adjacency matrix representation:

The size of the matrix is VxV where V is the number of vertices in the graph and the value of an entry Aij is either 1 or 0 depending on whether there is an edge from vertex i to vertex j.





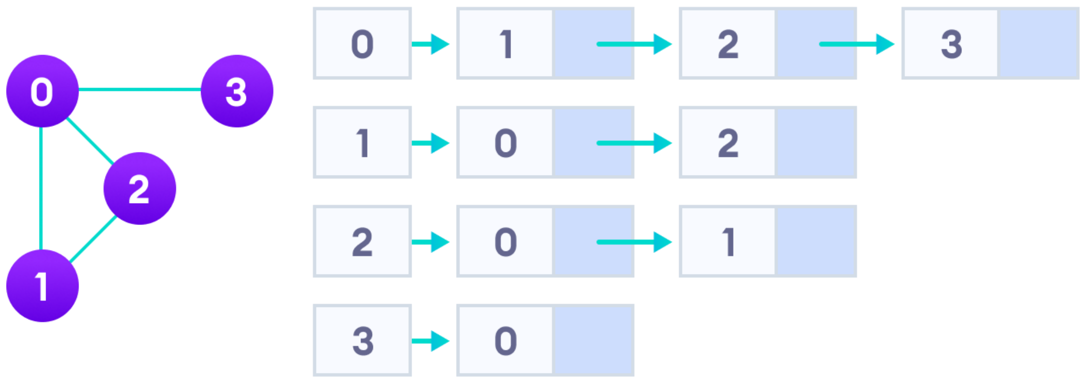
	0	1	2	3
0	0	1	1	1
1	1	0	1	0
2	1	1	0	0
3	1	0	0	0

Adjacency List:

An adjacency list represents a graph as an array of linked lists.

The index of the array represents a vertex and each element in its linked list represents the other vertices that form an edge with the vertex.

Adjacency List representation:



Depth First Search (DFS):

Depth first Search or Depth first traversal is a recursive algorithm for searching all the vertices of a graph or tree data structure.

A standard DFS implementation puts each vertex of the graph into one of two categories:

1. Visited
2. Not Visited

The purpose of the algorithm is to mark each vertex as visited while avoiding cycles.

Application of DFS :

1. For finding the path.
2. To test if the graph is bipartite.
3. For finding the strongly connected components of a graph.
4. For detecting cycles in a graph.

Breadth First Search (BFS):

Breadth First Traversal or Breadth First Search is a recursive algorithm for searching all the vertices of a graph or tree data structure.

A standard BFS implementation puts each vertex of the graph into one of two categories:

1. Visited
2. Not Visited

The purpose of the algorithm is to mark each vertex as visited while avoiding cycles.

ALGORITHM:

1) Start
2) Create a matrix of size n*n where every element is 0 representing there is no edge in the graph.
3) Now, for every edge of the graph between the vertices i and j set mat[i][j] = 1.
    a. //DFS using adjacency matrix
4) After the adjacency matrix has been created and filled, call the recursive function for the source i.e. vertex 0 that will recursively call the same function for all the vertices adjacent to it.
5) Also, keep an array to keep track of the visited vertices i.e. visited[i] = true represents that vertex i has been been visited before and the DFS function for some already visited node need not be called.
    a. //BFS using adjacency matrix x
6) After the adjacency matrix has been created and filled,
7) Visit the adjacent unvisited vertex. Mark it as visited. Display it. Insert it in a queue.
8) If no adjacent vertex is found, remove the first vertex from the queue.
9) Repeat Rule 1 and Rule 2 until the queue is empty.
10) Stop

PROGRAM:

```
#include <iostream>
#include <unordered_map>
#include <deque>
#include <stack>
#include <queue>
#include <list>
using namespace std;
```

```
class AdjacencyMatrix
{
   int vertices, edges;
   int **matrix = NULL;

public:
AdjacencyMatrix(int vertices, int edges)
   {
      this->vertices = vertices;
      this->edges = edges;

      matrix = new (nothrow) int *[vertices];
      if (!matrix)
         throw bad_alloc();

      for (int i = 0; i< vertices; i++)
      {
         matrix[i] = new (nothrow) int[vertices];
         if (!matrix[i])
            throw bad_alloc();
      }
   }
   ~AdjacencyMatrix()
   {
delete[] matrix;
   }
   void create();
   void display();
   void DepthFirstSearch(deque<bool>&);
};

void AdjacencyMatrix::create()
{
   for (int i = 0; i< vertices; i++)
   {
      for (int j = 0; j < vertices; j++)
      {
            matrix[i][j] = 0;
        }
    }

    int i, j;
    for (int k = 0; k < edges; k++)
    {
cout<< "\nEnter The Pair of Vertices For Edge " << k + 1 <<" : ";
cin>>i>> j;
        matrix[i][j] = 1;
        matrix[j][i] = 1;
    }
}

void AdjacencyMatrix::display()
{
    for (int i = 0; i< vertices; i++)
    {
        for (int j = 0; j < vertices; j++)
        {
cout<< " " << matrix[i][j];
        }
cout<<endl;
    }
}

void AdjacencyMatrix::DepthFirstSearch(deque<bool>&visited)
{
    stack<int>st;
st.push(0);

    while (!st.empty())
    {
        int vertex = st.top();
st.pop();

        if (!visited[vertex])
        {
```

```
cout<< vertex << " ";
            visited[vertex] = 1;
            for (int i = 0; i< vertices; i++)
            {
                if (matrix[vertex][i] == 1 and !visited[i])
                {
st.push(i);
                }
            }
        }
    }
cout<<endl;
}

class AdjacencyList
{
    int vertices, edges;
unordered_map<int, list<int>> *adjacencylist;

public:
AdjacencyList(int vertices, int edges)
    {
        this->vertices = vertices;
        this->edges = edges;
adjacencylist = new (nothrow) unordered_map<int, list<int>>;
        if (!adjacencylist)
            throw bad_alloc();
    }
    ~AdjacencyList()
    {
delete[] adjacencylist;
    }
    void create();
    void BreadthFirstSearch(deque<bool>&);
};

void AdjacencyList::create()
{
    int i, j;
    for (int k = 0; k < edges; k++)
    {
cout<< "\nEnter The Pair of Vertices For Edge " << k + 1 <<" : ";
cin>>i>> j;
        (*adjacencylist)[i].push_back(j);
    }
}

void AdjacencyList::BreadthFirstSearch(deque<bool>&visited)
{
    queue<int> que;
que.push(0);

    while (!que.empty())
    {
        int vertex = que.front();
cout<< vertex << " ";
        visited[vertex] = true;
que.pop();

        list<int>::iterator iter;
        for (iter = (*adjacencylist)[vertex].begin(); iter != (*adjacencylist)[vertex].end();
iter++)
        {
            if (!visited[*iter])
            {
                visited[*iter] = true;
que.push(*iter);
            }
        }
    }
cout<<endl;
}

int main()
{
    int vertices, edges;
```

```
   int userInput;
AdjacencyMatrix *adjm = NULL;
AdjacencyList *adjl = NULL;

   while (true)
   {
cout<< "\n1. Create Graph Using Adjacency Matrix\n2. Display Adjacency Matrix\n3.
Create Graph Using Adjacency List\n4. Depth First Traversal\n5. Breadth First
Traversal\n6. Exit\n>>>> ";
cin>>userInput;

       switch (userInput)
       {
       case 1:
cout<< "\nEnter No. of Vertices: ";
cin>> vertices;
cout<< "\nEnter No. of Edges: ";
cin>> edges;
adjm = new (nothrow) AdjacencyMatrix(vertices, edges);
adjm->create();
           break;

       case 2:
           if (adjm == NULL)
           {
cout<< "\nGraph is Empty!\n";
               break;
           }
cout<< "\nAdjacency Matrix: "
<< "\n";
adjm->display();
           break;

       case 3:
cout<< "\nEnter No. of Vertices: ";
cin>> vertices;
cout<< "\nEnter No. of Edges: ";
cin>> edges;
adjl = new (nothrow) AdjacencyList(vertices, edges);
adjl->create();
           break;

       case 4:
       {
           if (adjm == NULL)
           {
cout<< "\nGraph is Empty!\n";
               break;
           }
           deque<bool>visited(vertices, false);
cout<< "\nDepth First Search Traversal: ";
adjm->DepthFirstSearch(visited);
           break;
       }
       case 5:
       {
           if (adjl == NULL)
           {
cout<< "\nGraph is Empty!\n";
               break;
           }
           deque<bool>visited(vertices, false);
cout<< "\nBreadth First Search Traversal: ";
adjl->BreadthFirstSearch(visited);
           break;
       }
       case 6:
exit(0);
           break;

       default:
cout<< "\nPlease Enter Correct Input!\n";
           break;
       }
   }
   delete adjm;
   delete adjl;
   return 0;
}
```

OUTPUT:

```
// 1. Create Graph Using Adjacency Matrix
// 2. Display Adjacency Matrix
// 3. Create Graph Using Adjacency List
// 4. Depth First Traversal
// 5. Breadth First Traversal
// 6. Exit
// >>>> 2

// Graph is Empty!

// 1. Create Graph Using Adjacency Matrix
// 2. Display Adjacency Matrix
// 3. Create Graph Using Adjacency List
// 4. Depth First Traversal
// 5. Breadth First Traversal
// 6. Exit
// >>>> 4

// Graph is Empty!

// 1. Create Graph Using Adjacency Matrix
// 2. Display Adjacency Matrix
// 3. Create Graph Using Adjacency List
// 4. Depth First Traversal
// 5. Breadth First Traversal
// 6. Exit
// >>>> 5

// Graph is Empty!

// 1. Create Graph Using Adjacency Matrix
// 2. Display Adjacency Matrix
// 3. Create Graph Using Adjacency List
// 4. Depth First Traversal
// 5. Breadth First Traversal
// 6. Exit
// >>>> 1

// Enter No. of Vertices: 4

// Enter No. of Edges: 6

// Enter The Pair of Vertices For Edge 1 : 0 1

// Enter The Pair of Vertices For Edge 2 : 0 2

// Enter The Pair of Vertices For Edge 3 : 0 3

// Enter The Pair of Vertices For Edge 4 : 1 2

// Enter The Pair of Vertices For Edge 5 : 1 3

// Enter The Pair of Vertices For Edge 6 : 2 3

// 1. Create Graph Using Adjacency Matrix
// 2. Display Adjacency Matrix
// 3. Create Graph Using Adjacency List
// 4. Depth First Traversal
// 5. Breadth First Traversal
// 6. Exit
// >>>> 2

// Adjacency Matrix:
//  0 1 1 1
//  1 0 1 1
//  1 1 0 1
//  1 1 1 0

// 1. Create Graph Using Adjacency Matrix
// 2. Display Adjacency Matrix
// 3. Create Graph Using Adjacency List
// 4. Depth First Traversal
// 5. Breadth First Traversal
// 6. Exit
// >>>> 4

// Depth First Search Traversal: 0 3 2 1

// 1. Create Graph Using Adjacency Matrix
// 2. Display Adjacency Matrix
// 3. Create Graph Using Adjacency List
// 4. Depth First Traversal
// 5. Breadth First Traversal
// 6. Exit
// >>>> 5

// Graph is Empty!

// 1. Create Graph Using Adjacency Matrix
// 2. Display Adjacency Matrix
// 3. Create Graph Using Adjacency List
// 4. Depth First Traversal
// 5. Breadth First Traversal
// 6. Exit
// >>>> 3

// Enter No. of Vertices: 4

// Enter No. of Edges: 6

// Enter The Pair of Vertices For Edge 1 : 0 1

// Enter The Pair of Vertices For Edge 2 : 0 2

// Enter The Pair of Vertices For Edge 3 : 0 3

// Enter The Pair of Vertices For Edge 4 : 1 2
// Enter The Pair of Vertices For Edge 5 : 1 3

// Enter The Pair of Vertices For Edge 6 : 2 3

// 1. Create Graph Using Adjacency Matrix
// 2. Display Adjacency Matrix
// 3. Create Graph Using Adjacency List
// 4. Depth First Traversal
// 5. Breadth First Traversal
// 6. Exit
// >>>> 5

// Breadth First Search Traversal: 0 1 2 3

// 1. Create Graph Using Adjacency Matrix
// 2. Display Adjacency Matrix
// 3. Create Graph Using Adjacency List
// 4. Depth First Traversal
// 5. Breadth First Traversal
// 6. Exit
// >>>> 6
```

ANALYSIS:

Operations	Adjacency Matrix	Adjacency List
Storage Space	This representation makes use of VxV matrix, so space required in worst case is **O(|V|^2)**.	In this representation, for every vertex we store its neighbours. In the worst case, if a graph is connected O(V) is required for a vertex and O(E) is required for storing neighbours corresponding to every vertex .Thus, overall space complexity is O(|V|+|E|).
Adding a vertex	In order to add a new vertex to VxV matrix the storage must be increases to (|V|+1)^2. To achieve this we need to copy the whole	There are two pointers in adjacency list first points to the front node and the other one points to the rear node.Thus insertion of a vertex can be done directly

Operations	Adjacency Matrix	Adjacency List
	matrix. Therefore the complexity is **O(|V|²)**.	in **O(1) time.**
Adding an edge	To add an edge say from i to j, matrix[i][j] = 1 which requires **O(1)** time.	Similar to insertion of vertex here also two pointers are used pointing to the rear and front of the list. Thus, an edge can be inserted in **O(1)**time.
Removing a vertex	In order to remove a vertex from V*V matrix the storage must be decreased to $\|V\|^2$ from $(\|V\|+1)^2$. To achieve this we need to copy the whole matrix. Therefore the complexity is **O(|V|²)**.	In order to remove a vertex, we need to search for the vertex which will require O(|V|) time in worst case, after this we need to traverse the edges and in worst case it will require O(|E|) time.Hence, total time complexity is **O(|V|+|E|)**.
Removing an edge	To remove an edge say from i to j, matrix[i][j] = 0 which requires **O(1)** time.	To remove an edge traversing through the edges is required and in worst case we need to traverse through all the edges.Thus, the time complexity is **O(|E|)**.
Querying	In order to find for an existing edge the content of matrix needs to be checked. Given two vertices say i and j matrix[i][j] can be checked in **O(1)** time.	In an adjacency list every vertex is associated with a list of adjacent vertices. For a given graph, in order to check for an edge we need to check for vertices adjacent to given vertex. A vertex can have at most O(|V|) neighbours and in worst can we would have to check for every adjacent vertex. Therefore, time complexity is **O(|V|)** .

CONCLUSION: Hence,we have studied the concept of graphs. We have studied the implementation of graph using adjacency matrix. We have successfully implemented BFS and DFS on the given graph.

EXPERIMENT – 7

AIM:

There are flight paths between cities. If there is a flight between city A and city B then there isan edge between the cities. The cost of the edge can be the time that flight take to reach city Bfrom A, or the amount of fuel used for the journey. Represent this as a graph. The node can berepresented by airport name or name of the city. Use adjacency list representation of thegraph or use adjacency matrix representation of the graph. Check whether the graph isconnected or not. Justify the storage representation used.

OBJECTIVE:

1) To understand graph using Adjacency matrix.
2) To understand graph using Adjacency list representation

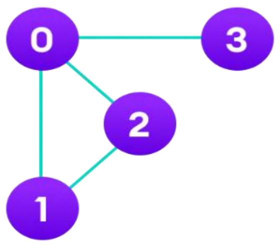
THEORY

Graph Data Structure:

A graph data structure is a collection of nodes that have data and are connected to other nodes.

More precisely, a graph is a data structure (V, E) that consists of

1. A collection of vertices V.
2. A collection of edges E, represented as ordered pairs of vertices (u,v).



In the graph,

```
V = {0, 1, 2, 3}
```

```
E = {(0,1), (0,2), (0,3), (1,2)}

G = {V, E}
```

Graph Terminology:

1. *Adjacency*: A vertex is said to be adjacent to another vertex if there is an edge connecting them. Vertices 2 and 3 are not adjacent because there is no edge between them.
2. *Path*: A sequence of edges that allows you to go from vertex A to vertex B is called a path. 0-1, 1-2 and 0-2 are paths from vertex 0 to vertex 2.
3. *Directed Graph*: A graph in which an edge (u,v) doesn't necessarily mean that there is an edge (v, u) as well. The edges in such a graph are represented by arrows to show the direction of the edge.

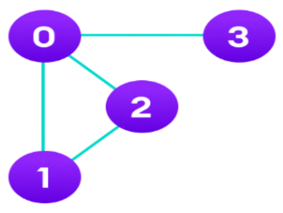
Graph Representation:

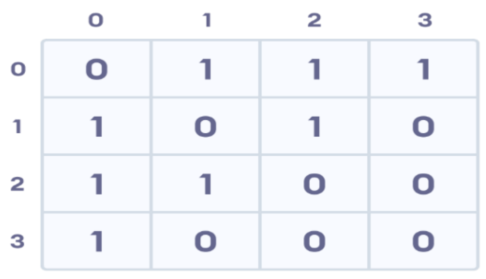
Graphs are commonly represented in two ways:

•Adjacency Matrix:

An adjacency matrix is a 2D array of V x V vertices. Each row and column represent a vertex.

If the value of any element a[i][j] is 1, it represents that there is an edge connecting vertex i and vertex j.



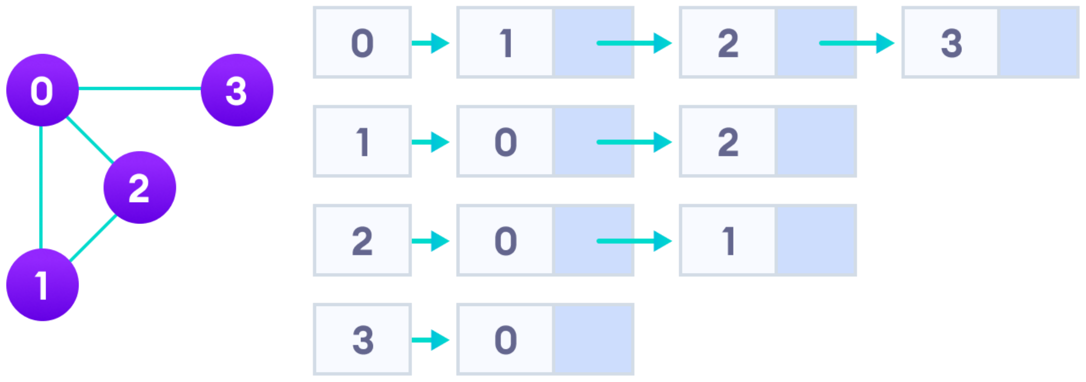


	0	1	2	3
0	0	1	1	1
1	1	0	1	0
2	1	1	0	0
3	1	0	0	0

•Adjacency List:

An adjacency list represents a graph as an array of linked lists.

The index of the array represents a vertex and each element in its linked list represents the other vertices that form an edge with the vertex.



Graph Operations:

The most common graph operations are:

1. Check if the element is present in the graph
2. Graph Traversal
3. Add elements(vertex, edges) to graph
4. Finding the path from one vertex to another

ALGORITHM:

1) Start
2) Create a 2D array(sizeNxN)
3) Initialize all value of this matrix to zero.
4) For each edge(flight hour) between two cities in arr[][](say **X and Y**), Update value at **Adj[X][Y]** and **Adj[Y][X]** to 1, denotes that there is an edge between X and Y.
    a. //DFS using adjacency matrix

5) Now, for every edge of the graph between the vertices i and j set mat[i][j] = number of hour .
6) After the adjacency matrix has been created and filled, call the recursive function for the source i.e. vertex 0 that will recursively call the same function for all the vertices adjacent to it.
7) Also, keep an array to keep track of the visited vertices i.e. visited[i] = true represents that vertex i has been been visited before and the DFS function for some already visited node need not be called.
    a. //BFS using adjacency matrix
8) Create a matrix of size n*n where every element is 0 representing there is no edge in the graph.
9) Now, for every edge of the graph between the vertices i and j set mat[i][j] = 1.
10) After the adjacency matrix has been created and filled,
11) Visit the adjacent unvisited vertex. Mark it as visited. Display it. Insert it in a queue.
12) If no adjacent vertex is found, remove the first vertex from the queue.
13) Repeat Rule 1 and Rule 2 until the queue is empty.
14) Stop

PROGRAM:

```
#include<iostream>
#include<stdlib.h>
#include<string.h>
using namespace std;
struct node
{   string vertex;
    int time;
    node *next;
};
class adjmatlist
{    int m[10][10],n,i,j; char ch;  string v[20];   node *head[20];  node *temp=NULL;

     public:
```

```
adjmatlist()
{     for(i=0;i<20;i++)
{   head[i]=NULL;  }
    }
    void getgraph();
    void adjlist();

    void displaym();
    void displaya();
};
void adjmatlist::getgraph()
{
cout<<"\n enter no. of cities(max. 20)";
cin>>n;
cout<<"\n enter name of cities";
  for(i=0;i<n;i++)
cin>>v[i];
  for(i=0;i<n;i++)
  {
    for(j=0;j<n;j++)
{  cout<<"\n if path is present between city "<<v[i]<<" and "<<v[j]<<" then press enter y
otherwise n";
cin>>ch;
      if(ch=='y')
      {
cout<<"\n enter time required to reach city "<<v[j]<<" from "<<v[i]<<" in minutes";
cin>>m[i][j];
      }
      else if(ch=='n')
{  m[i][j]=0;  }
      else
{ cout<<"\n unknown entry";  }
    }
  }
adjlist();

}
void adjmatlist::adjlist()
```

```
{  cout<<"\n ****";
      for(i=0;i<n;i++)
{  node *p=new(struct node);
         p->next=NULL;
         p->vertex=v[i];
         head[i]=p;   cout<<"\n"<<head[i]->vertex;
      }

      for(i=0;i<n;i++)
{  for(j=0;j<n;j++)
         {
                 if(m[i][j]!=0)
                 {
                       node *p=new(struct node);
                       p->vertex=v[j];
                       p->time=m[i][j];
                       p->next=NULL;
                       if(head[i]->next==NULL)
{  head[i]->next=p;   }
                       else
{  temp=head[i];
                       while(temp->next!=NULL)
{   temp=temp->next;  }
                           temp->next=p;
                       }

                 }

         }
      }

}
void adjmatlist::displaym()
{  cout<<"\n";
     for(j=0;j<n;j++)
{  cout<<"\t"<<v[j];  }

     for(i=0;i<n;i++)
```

```
{  cout<<"\n "<<v[i];
       for(j=0;j<n;j++)
{  cout<<"\t"<<m[i][j];
       }
cout<<"\n";
    }
}
void adjmatlist::displaya()
{
cout<<"\n adjacency list is";

      for(i=0;i<n;i++)
      {



                     if(head[i]==NULL)
{  cout<<"\n adjacency list not present";  break;   }
                     else
                     {
cout<<"\n"<<head[i]->vertex;
                     temp=head[i]->next;
                     while(temp!=NULL)
{  cout<<"-> "<<temp->vertex;
                        temp=temp->next;  }

                     }





      }

cout<<"\n path and time required to reach cities is";

      for(i=0;i<n;i++)
      {
```

```
                    if(head[i]==NULL)
{  cout<<"\n adjacency list not present";  break;   }
                    else
                    {

                    temp=head[i]->next;
                    while(temp!=NULL)
{  cout<<"\n"<<head[i]->vertex;
cout<<"-> "<<temp->vertex<<"\n   [time required: "<<temp->time<<" min ]";
                     temp=temp->next;  }

                    }




       }
}
int main()
{  int m;
adjmatlist a;

while(1)
  {
cout<<"\n\n enter the choice";
cout<<"\n 1.enter graph";
cout<<"\n 2.display adjacency matrix for cities";
cout<<"\n 3.display adjacency list for cities";
cout<<"\n 4.exit";
cin>>m;

       switch(m)
{            case 1: a.getgraph();
                          break;
                 case 2: a.displaym();
                         break;

                     case 3: a.displaya();
```

```
                        break;
                    case 4: exit(0);

                    default:  cout<<"\n unknown choice";
        }
    }
    return 0;
}
```

OUTPUT:

```
Enter the choice
 1.Enter graph
 2.Display adjacency matrix for cities
 3.Display adjacency list for cities
 4.Exit
1
 Enter no. of cities(max. 20)7

 Enter name of cities
Jalgaon
Pune
Mumbai
Delhi
Bangalore
Indore
Kolkata

 if path is present between city Jalgaon and Jalgaon then press enter y otherwise n
n

 if path is present between city Jalgaon and Pune then press enter y otherwise n
y

 enter time required to reach city Pune from Jalgaon in minutes
120
 if path is present between city Jalgaon and Mumbai then press enter y otherwise n
```

```
y

 enter time required to reach city Mumbai from Jalgaon in minutes
180

 if path is present between city Jalgaon and Delhi then press enter y otherwise n
y

 enter time required to reach city Delhi from Jalgaon in minutes
360

 if path is present between city Jalgaon and Banglore then press enter y otherwise n
y

 enter time required to reach city Banglore from Jalgaon in minutes
240

 if path is present between city Jalgaon and Indore then press enter y otherwise n
y

 enter time required to reach city Indore from Jalgaon in minutes
60

 if path is present between city Jalgaon and Kolkata then press enter y otherwise n
y

 enter time required to reach city Kolkata from Jalgaon in minutes
420

 if path is present between city Pune and Jalgaon then press enter y otherwise n
y

 enter time required to reach city Jalgaon from Pune in minutes
120

 if path is present between city Pune and Pune then press enter y otherwise n
n
```

```
 if path is present between city Pune and Mumbai then press enter y otherwise n
y

 enter time required to reach city Mumbai from Pune in minutes
60

 if path is present between city Pune and Delhi then press enter y otherwise n
y

 enter time required to reach city Delhi from Pune in minutes
420

 if path is present between city Pune and Banglore then press enter y otherwise n
y

 enter time required to reach city Banglore from Pune in minutes
180

 if path is present between city Pune and Indore then press enter y otherwise n
y

 enter time required to reach city Indore from Pune in minutes
180

 if path is present between city Pune and Kolkata then press enter y otherwise n
y

 enter time required to reach city Kolkata from Pune in minutes
480

 if path is present between city Mumbai and Jalgaon then press enter y otherwise n
y

 enter time required to reach city Jalgaon from Mumbai in minutes
180

 if path is present between city Mumbai and Pune then press enter y otherwise n
y
```

```
 enter time required to reach city Pune from Mumbai in minutes
60

 if path is present between city Mumbai and Mumbai then press enter y otherwise n
n

 if path is present between city Mumbai and Delhi then press enter y otherwise n
y

 enter time required to reach city Delhi from Mumbai in minutes
420

 if path is present between city Mumbai and Banglore then press enter y otherwise n
y

 enter time required to reach city Banglore from Mumbai in minutes
240

 if path is present between city Mumbai and Indore then press enter y otherwise n
y

 enter time required to reach city Indore from Mumbai in minutes
240

 if path is present between city Mumbai and Kolkata then press enter y otherwise n
y

 enter time required to reach city Kolkata from Mumbai in minutes
540

 if path is present between city Delhi and Jalgaon then press enter y otherwise n
y

 enter time required to reach city Jalgaon from Delhi in minutes
360

 if path is present between city Delhi and Pune then press enter y otherwise n
```

```
y

 enter time required to reach city Pune from Delhi in minutes
420

 if path is present between city Delhi and Mumbai then press enter y otherwise n
y

 enter time required to reach city Mumbai from Delhi in minutes
420

 if path is present between city Delhi and Delhi then press enter y otherwise n
n

 if path is present between city Delhi and Banglore then press enter y otherwise n
y

 enter time required to reach city Banglore from Delhi in minutes
540

 if path is present between city Delhi and Indore then press enter y otherwise n
y

 enter time required to reach city Indore from Delhi in minutes
420

 if path is present between city Delhi and Kolkata then press enter y otherwise n
y

 enter time required to reach city Kolkata from Delhi in minutes
300

 if path is present between city Banglore and Jalgaon then press enter y otherwise n
y

 enter time required to reach city Jalgaon from Banglore in minutes
240
```

```
 if path is present between city Banglore and Pune then press enter y otherwise n
y

 enter time required to reach city Pune from Banglore in minutes
180

 if path is present between city Banglore and Mumbai then press enter y otherwise n
y

 enter time required to reach city Mumbai from Banglore in minutes
240

 if path is present between city Banglore and Delhi then press enter y otherwise n
y

 enter time required to reach city Delhi from Banglore in minutes
540

 if path is present between city Banglore and Banglore then press enter y otherwise n
n

 if path is present between city Banglore and Indore then press enter y otherwise n
y

 enter time required to reach city Indore from Banglore in minutes
300

 if path is present between city Banglore and Kolkata then press enter y otherwise n
y

 enter time required to reach city Kolkata from Banglore in minutes
600

 if path is present between city Indore and Jalgaon then press enter y otherwise n
y

 enter time required to reach city Jalgaon from Indore in minutes
60
```

```
 if path is present between city Indore and Pune then press enter y otherwise n
y

 enter time required to reach city Pune from Indore in minutes
180

 if path is present between city Indore and Mumbai then press enter y otherwise n
y

 enter time required to reach city Mumbai from Indore in minutes
240

 if path is present between city Indore and Delhi then press enter y otherwise n
y

 enter time required to reach city Delhi from Indore in minutes
420

 if path is present between city Indore and Banglore then press enter y otherwise n
y

 enter time required to reach city Banglore from Indore in minutes
360

 if path is present between city Indore and Indore then press enter y otherwise n
n

 if path is present between city Indore and Kolkata then press enter y otherwise n
y

 enter time required to reach city Kolkata from Indore in minutes
420

 if path is present between city Kolkata and Jalgaon then press enter y otherwise n
y

 enter time required to reach city Jalgaon from Kolkata in minutes
```

```
420

 if path is present between city Kolkata and Pune then press enter y otherwise n
y

 enter time required to reach city Pune from Kolkata in minutes
480

 if path is present between city Kolkata and Mumbai then press enter y otherwise n
y

 enter time required to reach city Mumbai from Kolkata in minutes
540

 if path is present between city Kolkata and Delhi then press enter y otherwise n
y

 enter time required to reach city Delhi from Kolkata in minutes
300

 if path is present between city Kolkata and Banglore then press enter y otherwise n
y

 enter time required to reach city Banglore from Kolkata in minutes
600

 if path is present between city Kolkata and Indore then press enter y otherwise n
y

 enter time required to reach city Indore from Kolkata in minutes
420

 if path is present between city Kolkata and Kolkata then press enter y otherwise n
n

 ****
Jalgaon
Pune
```

```
Mumbai
Delhi
Bangalore
Indore
Kolkata

 Enter the choice
 1.Enter graph
 2.Display adjacency matrix for cities
 3.Display adjacency list for cities
 4.Exit
2

 Jalgaon Pune Mumbai Delhi   Bangalore IndoreKolkata
 Jalgaon       0      120     180     360     240     60      420

 Pune   120     0      60      420     180     180     480

 Mumbai 180     60      0      420     240     240     540

 Delhi  360     420     420     0      540     420     300

Bangalore 240     180     240     540     0      300     600

 Indore 60      180     240     420     360     0      420

 Kolkata   420     480     540     300     600     420     0


Enter the choice
 1.Enter graph
 2.Display adjacency matrix for cities
 3.Display adjacency list for cities
 4.Exit3

Adjacency list is
Jalgaon-> Pune-> Mumbai-> Delhi->Bangalore-> Indore-> Kolkata
Pune-> Jalgaon-> Mumbai-> Delhi->Bangalore-> Indore-> Kolkata
```

```
Mumbai-> Jalgaon-> Pune-> Delhi->Bangalore-> Indore-> Kolkata
Delhi-> Jalgaon-> Pune-> Mumbai->Bangalore-> Indore-> Kolkata
Bangalore-> Jalgaon-> Pune-> Mumbai-> Delhi-> Indore-> Kolkata
Indore-> Jalgaon-> Pune-> Mumbai-> Delhi->Bangalore-> Kolkata
Kolkata-> Jalgaon-> Pune-> Mumbai-> Delhi->Bangalore-> Indore
 path and time required to reach cities is
Jalgaon-> Pune
   [time required: 120 min ]
Jalgaon-> Mumbai
   [time required: 180 min ]
Jalgaon-> Delhi
   [time required: 360 min ]
Jalgaon->Bangalore
   [time required: 240 min ]
Jalgaon-> Indore
   [time required: 60 min ]
Jalgaon-> Kolkata
   [time required: 420 min ]
Pune-> Jalgaon
   [time required: 120 min ]
Pune-> Mumbai
   [time required: 60 min ]
Pune-> Delhi
   [time required: 420 min ]
Pune->Bangalore
   [time required: 180 min ]
Pune-> Indore
   [time required: 180 min ]
Pune-> Kolkata
   [time required: 480 min ]
Mumbai-> Jalgaon
   [time required: 180 min ]
Mumbai-> Pune
   [time required: 60 min ]
Mumbai-> Delhi
   [time required: 420 min ]
Mumbai->Bangalore
   [time required: 240 min ]
```

```
Mumbai-> Indore
  [time required: 240 min ]
Mumbai-> Kolkata
  [time required: 540 min ]
Delhi-> Jalgaon
  [time required: 360 min ]
Delhi-> Pune
  [time required: 420 min ]
Delhi-> Mumbai
  [time required: 420 min ]
Delhi->Bangalore
  [time required: 540 min ]
Delhi-> Indore
  [time required: 420 min ]
Delhi-> Kolkata
  [time required: 300 min ]
Bangalore-> Jalgaon
  [time required: 240 min ]
Bangalore-> Pune
  [time required: 180 min ]
Bangalore-> Mumbai
  [time required: 240 min ]
Bangalore-> Delhi
  [time required: 540 min ]
Bangalore-> Indore
  [time required: 300 min ]
Bangalore-> Kolkata
  [time required: 600 min ]
Indore-> Jalgaon
  [time required: 60 min ]
Indore-> Pune
  [time required: 180 min ]
Indore-> Mumbai
  [time required: 240 min ]
Indore-> Delhi
  [time required: 420 min ]
Indore->Banglore
  [time required: 360 min ]
```

```
Indore-> Kolkata
  [time required: 420 min ]
Kolkata-> Jalgaon
  [time required: 420 min ]
Kolkata-> Pune
  [time required: 480 min ]
Kolkata-> Mumbai
  [time required: 540 min ]
Kolkata-> Delhi
  [time required: 300 min ]
Kolkata->Bangalore
  [time required: 600 min ]
Kolkata-> Indore
  [time required: 420 min ]

Enter the choice
 1.Enter graph
 2.Display adjacency matrix for cities
 3.Display adjacency list for cities
 4.Exit
4
```

ANALYSIS:

Operations	Adjacency Matrix	Adjacency List
Storage Space	This representation makes use of VxV matrix, so space required in worst case is **O(|V|²)**.	In this representation, for every vertex we store its neighbours. In the worst case, if a graph is connected O(V) is required for a vertex and O(E) is required for storing neighbours corresponding to every vertex .Thus, overall space complexity is O(|V|+|E|).
Adding a vertex	In order to add a new vertex to VxV matrix the storage must be increases to $(\|V\|+1)^2$. To achieve this we need to copy the whole matrix. Therefore the complexity	There are two pointers in adjacency list first points to the front node and the other one points to the rear node.Thus insertion of a vertex can be done directly in **O(1) time.**

Operations	Adjacency Matrix	Adjacency List
	is **O(|V|^2)**.	
Adding an edge	To add an edge say from i to j, matrix[i][j] = 1 which requires **O(1)** time.	Similar to insertion of vertex here also two pointers are used pointing to the rear and front of the list. Thus, an edge can be inserted in **O(1)**time.
Removing a vertex	In order to remove a vertex from V*V matrix the storage must be decreased to |V|^2 from (|V|+1)^2. To achieve this we need to copy the whole matrix. Therefore the complexity is **O(|V|^2)**.	In order to remove a vertex, we need to search for the vertex which will require O(|V|) time in worst case, after this we need to traverse the edges and in worst case it will require O(|E|) time.Hence, total time complexity is **O(|V|+|E|)**.
Removing an edge	To remove an edge say from i to j, matrix[i][j] = 0 which requires **O(1)** time.	To remove an edge traversing through the edges is required and in worst case we need to traverse through all the edges.Thus, the time complexity is **O(|E|)**.
Querying	In order to find for an existing edge the content of matrix needs to be checked. Given two vertices say i and j matrix[i][j] can be checked in **O(1)** time.	In an adjacency list every vertex is associated with a list of adjacent vertices. For a given graph, in order to check for an edge we need to check for vertices adjacent to given vertex. A vertex can have at most O(|V|) neighbours and in worst can we would have to check for every adjacent vertex. Therefore, time complexity is **O(|V|)** .

CONCLUSION: Hence,we have implemented graph using adjacency list.We have also checked whether a graph is connected or not.

EXPERIMENT – 8

AIM:

You have a business with several offices; you want to lease phone lines to connect them up with each other; and the phone company charges different amounts of money to connect different pairs of cities. You want a set of lines that connects all your offices with a minimum total cost. Solve the problem by suggesting appropriate data structures.

OBJECTIVE:

1. To study and implement concepts of graphs.
2. To study the shortest path algorithm.

THEORY:

In graph theory, the shortest path problem is the problem of finding a path between two vertices (or nodes) in a graph such that the sum of the weights of its constituent edges is minimized.

The problem of finding the shortest path between two intersections on a road map may be modeled as a special case of the shortest path problem in graphs, where the vertices correspond to intersections and the edges correspond to road segments, each weighted by the length of the segment.

The most important algorithms for solving this problem are:

- Dijkstra's algorithm solves the single-source shortest path problem with non-negative edge weight.
- Bellman–Ford algorithm solves the single-source problem if edge weights may be negative.
- A* search algorithm solves for single-pair shortest path using heuristics to try to speed up the search.
- Floyd–Warshall algorithm solves all pairs shortest paths.
- Johnson's algorithm solves all pairs shortest paths, and may be faster than Floyd–Warshall on sparse graphs.
- Viterbi algorithm solves the shortest stochastic path problem with an additional probabilistic weight on each node.

Dijkstra's Shortest Path Algorithm

Dijkstra's algorithm has many variants but the most common one is to find the shortest paths from the source vertex to all other vertices in the graph.

Algorithm Steps:

- Set all vertices distances = infinity except for the source vertex, set the source distance equal to 0.

- Push the source vertex in a min-priority queue in the form (distance , vertex), as the comparison in the min-priority queue will be according to vertices distances.
- Pop the vertex with the minimum distance from the priority queue (at first the popped vertex = source).
- Update the distances of the connected vertices to the popped vertex in case of "current vertex distance + edge weight < next vertex distance", then push the vertex with the new distance to the priority queue.
- If the popped vertex is visited before, just continue without using it.
- Apply the same algorithm again until the priority queue is empty.

Floyd Warshall's Algorithm

Floyd Warshall's Algorithm is used to find the shortest paths between between all pairs of vertices in a graph, where each edge in the graph has a weight which is positive or negative. The biggest advantage of using this algorithm is that all the shortest distances between any 2 vertices could be calculated in $O(V^3)$, where V is the number of vertices in a graph.

The Algorithm Steps:

For a graph with N vertices:

- Initialize the shortest paths between any 2 vertices with Infinity.
- Find all pair shortest paths that use 0 intermediate vertices, then find the shortest paths that use 1 intermediate vertex and so on.. until using all N vertices as intermediate nodes.
- Minimize the shortest paths between any 2 pairs in the previous operation.
- For any 2 vertices (i,j) , one should actually minimize the distances between this pair using the first K nodes, so the shortest path will be: min(dist[i][k]+dist[k][j],dist[i][j]).

dist[i][k] represents the shortest path that only uses the first K vertices, dist[k][j] represents the shortest path between the pair k,j. As the shortest path will be a concatenation of the shortest path from i to k, then from k to j.

ALGORITHM:

1) Start

2) Take inputs about Vertex

3) Give connections between vertex

4) Define the path distances between vertex

5) Push the source vertex in a min-priority queue in the form (distance , vertex), as the comparison in the min-priority queue will be according to vertices distances.

6) Pop the vertex with the minimum distance from the priority queue (at first the popped vertex = source).

7) Update the distances of the connected vertices to the popped vertex in case of "current vertex distance + edge weight < next vertex distance", then push the vertex with the new distance to the priority queue.

8) If the popped vertex is visited before, just continue without using it.

9) Apply the same algorithm again until the priority queue is empty.

10) Stop.

PROGRAM:

```
/*

Name: Harsh Shah

Class: SE Comp-1

ERP No: 59


You have a business with several offices; you want to lease phone lines

to connect them up with each other; and the phone company charges different

amounts of money to connect different pairs of cities. You want a set of

lines that connects all your offices with a minimum total cost. Solve the

problem by suggesting appropriate data structures.

*/
```

```
#include <iostream>
        using namespace std;


        int main() {
                int n, i, j, k, row, col, mincost=0, min;
                char op;
                cout<<"Enter no. of vertices: ";
                cin>>n;
                int cost[n][n];
                int visit[n];
                for(i=0; i<n; i++)
                        visit[i] = 0;
                for(i=0; i<n; i++)
                        for(int j=0; j<n; j++)
                                cost[i][j] = -1;
                for(i=0; i<n; i++)
                {
                        for(j=i+1; j<n; j++)
                        {
                                cout<<"Do you want an edge between "<<i+1<<" and "<<j+1<<":
";
                                //use 'i' & 'j' if your vertices start from 0
                                cin>>op;
                                if(op=='y' || op=='Y')

                                {

                                        cout<<"Enter weight: ";

                                        cin>>cost[i][j];

                                        cost[j][i] = cost[i][j];

                                }

                        }

                }

                visit[0] = 1;

                for(k=0; k<n-1; k++)

                {

                        min = 999;

                        for(i=0; i<n; i++)

                        {

                                for(j=0; j<n; j++)

                                {

                                        if(visit[i] == 1 && visit[j] == 0)

                                        {

                                                if(cost[i][j] != -1 && min>cost[i][j])

                                                {

                                                        min = cost[i][j];

                                                        row = i;

                                                        col = j;
                                                }
                                        }
                                }
                        }
                        mincost += min;
                        visit[col] = 1;
                        cost[row][col] = cost[col][row] = -1;
                        cout<<row+1<<"->"<<col+1<<endl;
                        //use 'row' & 'col' if your vertices start from 0
                }
                cout<<"\nMin. Cost: "<<mincost;
                return 0;
        }
```

OUTPUT:

```
Enter no. of vertices: 5

Do you want an edge between 1 and 2: n

Do you want an edge between 1 and 3: y

Enter weight: 20

Do you want an edge between 1 and 4: y

Enter weight: 40

Do you want an edge between 1 and 5: y

Enter weight: 60
```

```
Do you want an edge between 2 and 3: n

Do you want an edge between 2 and 4: y

Enter weight: 80

Do you want an edge between 2 and 5: y

Enter weight: 100

Do you want an edge between 3 and 4: y

Enter weight: 120

Do you want an edge between 3 and 5: n

Do you want an edge between 4 and 5: n

1->3

1->4

1->5

4->2


Min. Cost: 200

--------------------------------

Process exited after 48.83 seconds with return value 0

Press any key to continue . . .
```

ANALYSIS:

- Worst case time complexity: **Θ(E+V log V)**
- Average case time complexity: **Θ(E+V log V)**
- Best case time complexity: **Θ(E+V log V)**
- Space complexity: **Θ(V)**

Time Complexity of Dijkstra's Algorithm is O(V2) but with min-priority queue it drops down to O(V+ElogV).

However, if we have to find the shortest path between all pairs of vertices, both of the above methods would be expensive in terms of time. Discussed below is another alogorithm designed for this case.

CONCLUSION: Hence, we have studied graph and successfully implemented the shortest path algorithm on our given graph.

EXPERIMENT – 9

AIM:

Given sequence k = k1 <k2 < … <kn of n sorted keys, with a search probability pi for each key ki . Build the Binary search tree that has the least search cost given the access probability for each key?

OBJECTIVE:

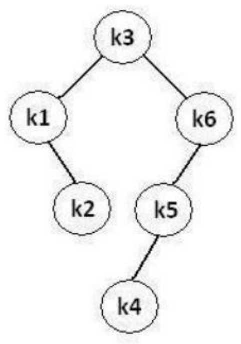
1. To understand concept of OBST.
2. To understand concept & features like extended binary search tree.

THEORY:

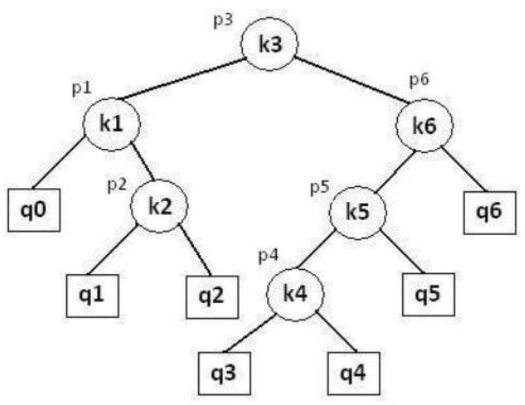
An optimal binary search tree is a binary search tree for which the nodes are arranged on levels such that the tree cost is minimum.

For the purpose of a better presentation of optimal binary search trees, we will consider “extended binary search trees”, which have the keys stored at their internal nodes. Suppose “n” keys k1, k2, … k n are stored at the internal nodes of a binary search tree. It is assumed that the keys are given in sorted order, so that k1< k2 < … < kn.

An extended binary search tree is obtained from the binary search tree by adding successor nodes to each of its terminal nodes as indicated in the following figure by squares:



Binary search tree



Extended binary search tree

In the extended tree:

- The squares represent terminal nodes. These terminal nodes represent unsuccessful searches of the tree for key values. The searches did not end successfully, that is, because they represent key values that are not actually stored in the tree;
- The round nodes represent internal nodes; these are the actual keys stored in the tree;
- Assuming that the relative frequency with which each key value is accessed is known, weights can be assigned to each node of the extended tree (p1 … p6). They represent the relative frequencies of searches terminating at each node, that is, they mark the successful searches.
- If the user searches a particular key in the tree, 2 cases can occur:
- 1 – the key is found, so the corresponding weight „p‟ is incremented;
- 2 – the key is not found, so the corresponding „q‟ value is incremented.

GENERALIZATION:
The terminal node in the extended tree that is the left successor of k1 can be interpreted as representing all key values that are not stored and are less than k1. Similarly, the terminal node in the extended tree that is the right successor of kn, represents all key values not stored in the tree that are greater than kn. The terminal node that is successes between ki and ki-1 in an inorder traversal represent all key values not stored that lie between ki and ki - 1.

ALGORITHM:

STEP 1: Create a class for OBST and declare all the arrays required.

STEP 2: Initialize front and rear to -1.

STEP 3: Accept the number of nodes and the data to be present in each node from the user and add it to an array.

STEP 4: Accept the possibilities for successful and unsuccessful searches and store them in two different arrays.

STEP 5: Create a method to calculate the OBST and store the elements according to their order in a 2-D array.

STEP 6: Display the root node of the optimal binary search tree.

STEP 7: Display the left and right nodes of the OBST.

STEP 8: STOP

PROGRAM:

```
/*

Name: Harsh Shah

Class: SE Comp-1

ERP N0: 59


Given sequence k = k1 <k2 < … <kn of n sorted keys, with a search

probability pi for each key ki.Build the Binary search tree that has

the least search cost given the access probability for each key?

*/


#include<iostream>

# define SIZE 10

using namespace std;

class Optimal

{

	private:

		int p[SIZE];

		int q[SIZE];

		int a[SIZE];

		int w [SIZE][SIZE];

		int c [SIZE][SIZE];

		int r [SIZE][SIZE];

		int n;
```

```
                int front, rear, queue[20];

        public:
                Optimal();
                void get_data();
                int Min_Value(int,int);
                void OBST();
                void build_tree();
};

Optimal::Optimal()
{
        front=rear=-1;
}

void Optimal::get_data()
{
        int i;
        cout<<"\n Optimal Binary Search Tree \n";
        cout<<"\n Enter the number of nodes: ";
        cin>>n;
        cout<<"\n Enter the data as ....\n";
        for(i=1;i<=n;i++)
        {
```

```
		cout<<"\n a["<<i<<"]";
		cin>>a[i];
	}

	cout<<"\n The probabilities for successful search ....\n";
	for(i=1;i<=n;i++)
	{
		cout<<"p["<<i<<"]";
		cin>>p[i];
	}

	cout<<"\n The probabilities for unsuccessful search ....\n";
	for(i=0;i<=n;i++)
	{
		cout<<"q["<<i<<"]";
		cin>>q[i];
	}
}

int Optimal::Min_Value(int i, int j)
{
	int m,k;
	int minimum=32000;
	for(m=r[i][j-1];m<=r[i+1][j];m++)
```

```
        {
                if((c[i][m-1]+c[m][j])<minimum)
                {
                        minimum=c[i][m-1]+c[m][j];
                        k=m;
                }
        }
        return k;
}

void Optimal::OBST()
{
        int i,j,k,m;
        for(i=0;i<n;i++)
        {
                w[i][j]=q[i];
                r[i][i]=c[i][i]=0;
                w[i][i+1]=q[i]+q[i+1]+p[i+1];
                r[i][i+1]=i+1;
                c[i][i+1]=q[i]+q[i+1]+p[i+1];
        }
        w[n][n]=q[n];
        r[n][n]=c[n][n]=0;
        for(m=2;m<=n;m++)

        {

                for(i=0;i<=n-m;i++)

                {

                        j=i+m;

                        w[i][j]=w[i][j-1]+p[j]+q[j];

                        k=Min_Value(i,j);

                        c[i][j]=w[i][j]+c[i][k-1]+c[k][j];

                        r[i][j]=k;

                }

        }

}


void Optimal::build_tree()

{

        int i,j,k;

        cout<<"The Optimal Binary Search Tree for the given node is: \n";

        cout<<"\n The Root of this OBST is:: "<<r[0][n];

        cout<<"\n The cost of this OBST is:: "<<c[0][n];

        cout<<"\n\n\tNODE\tLEFT CHILD\tRIGHT CHILD";

        cout<<"\n----------------------------------------------------"<<endl;

        queue[++rear]=0;

        queue[++rear]=n;

        while(front!=rear)
```

```
{
        i=queue[++front];
        j=queue[++front];
        k=r[i][j];
        cout<<"\n\t"<<k;
        if(r[i][k-1]!=0)
        {
                cout<<"                    "<<r[i][k-1];
                queue[++rear]=i;
                queue[++rear]=k-1;
        }
        else
        {
                cout<<"                    -";
        }
        if(r[k][j]!=0)
        {
                cout<<"                    "<<r[k][j];
                queue[++rear]=k;
                queue[++rear]=j;
        }
        else
        {
                cout<<"                    -";
                }
        }
        cout<<endl;
}
int main()
{
        Optimal obj;
        obj.get_data();
        obj.OBST();
        obj.build_tree();
        return 0;
}
```

OUTPUT:

```
 Optimal Binary Search Tree


 Enter the number of nodes: 4


 Enter the data as ....


 a[1] 1
 a[2] 2
 a[3] 3
 a[4] 4


 The probabilities for successful search ....
p[1] 3
p[2] 3
p[3] 1
p[4] 1


 The probabilities for unsuccessful search ....
q[0] 2
q[1] 3
q[2] 1
q[3] 1
q[4] 1
The Optimal Binary Search Tree for the given node is:


 The Root of this OBST is:: 2
 The cost of this OBST is:: 32


        NODE    LEFT CHILD      RIGHT CHILD
----------------------------------------------------


        2           1            3
        1           -            -
```

```
3       -       4

4       -       -


--------------------------------

Process exited after 44.79 seconds with return value 0

Press any key to continue . . .
```

COMPLEXITY:

Time Complexity: The algorithm requires O (n^3) time, since three nested for loops are used. Each of these loops takes on at most n values.

Space Complexity: The algorithm requires O(n^3) space.

CONCLUSION: Thus, we also have studied OBST and implemented it. We have built the Binary search tree that has the least search cost given the access probability for each key.

EXPERIMENT – 10

AIM:

Consider a scenario for Hospital to cater services to different kinds of patients as Serious (top priority), b) non-serious (medium priority), c) General Checkup (Least priority). Implement the priority queue to cater services to the patients.

OBJECTIVE:

1. To study the concept of priority queue.
2. To implement priority queue and perform operations like addition, removal, etc.

THEORY:

A priority queue is an abstract data type that behaves similarly to the normal queue except that each element has some priority, i.e., the element with the highest priority would come first in a priority queue. The priority of the elements in a priority queue will determine the order in which elements are removed from the priority queue.

The priority queue supports only comparable elements, which means that the elements are either arranged in an ascending or descending order.

For example, suppose we have some values like 1, 3, 4, 8, 14, 22 inserted in a priority queue with an ordering imposed on the values is from least to the greatest. Therefore, the 1 number would be having the highest priority while 22 will be having the lowest priority.

Priority Queue is an extension of queue with following properties.

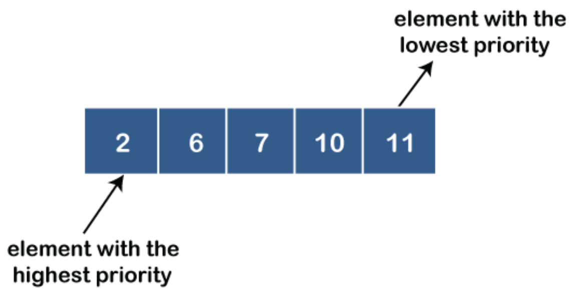
- Every item has a priority associated with it.
- An element with high priority is dequeued before an element with low priority.
- If two elements have the same priority, they are served according to their order in the queue.

A typical priority queue supports following operations.
insert(item, priority): Inserts an item with given priority.
getHighestPriority(): Returns the highest priority item.
deleteHighestPriority(): Removes the highest priority item.

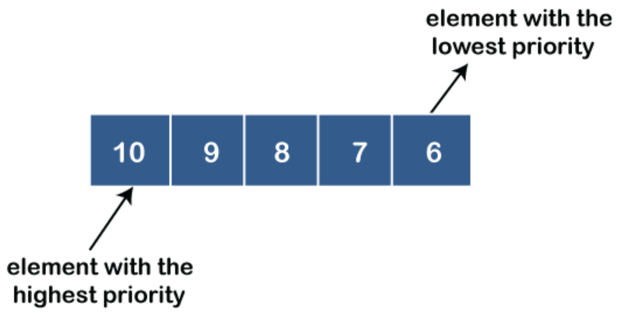
Types of Priority Queue

There are two types of priority queue:

- **Ascending order priority queue:** In ascending order priority queue, a lower priority number is given as a higher priority in a priority. For example, we take the numbers from 1 to 5 arranged in an ascending order like 1,2,3,4,5; therefore, the smallest number, i.e., 1 is given as the highest priority in a priority queue.


element with the lowest priority
2
6
7
10
11
element with the highest priority

- **Descending order priority queue:** In descending order priority queue, a higher priority number is given as a higher priority in a priority. For example, we take the numbers from 1 to 5 arranged in descending order like 5, 4, 3, 2, 1; therefore, the largest number, i.e., 5 is given as the highest priority in a priority queue.


element with the lowest priority
10
9
8
7
6
element with the highest priority

Applications of Priority Queue:
1) CPU Scheduling
2) Graph algorithms like Dijkstra's shortest path algorithm, Prim's Minimum Spanning Tree, etc
3) All queue applications where priority is involved.

ALGORITHM:

STEP 1: Initialize an array to store name of the patients.
STEP 2: Initialize another array to store priority of the patients.
STEP 3: Initialize front and rear to -1.
STEP 4: Display menu to user.
STEP 5: If user wants to insert in queue, then accept name of the patient and its priority from the user.
STEP 6: If the queue is not full, insert the element into the queue.
STEP 7: If user wants to remove an element from the queue, then delete the element with highest priority from the queue.
STEP 8: If user wants to check the patient's list, then display the queue along with the priority of each patient.
STEP 9: Repeat steps 4-8 until user wants to exit.
STEP 10: STOP

PROGRAM:

```
/*
Name: Harsh Shah
Class: SE Comp-1
ERP No:59

Consider a scenario for Hospital to cater services to
different kinds of patients as Serious (top priority),
b) non-serious (medium priority), c) General Checkup (Least priority).
Implement the priority queue to cater services to the patients.
*/

#include<iostream>
#include<string>

#define N 20

#define SERIOUS 10
#define NONSERIOUS 5
#define CHECKUP 1

using namespace std;
string Q[N];
int Pr[N];
int r = -1,f = -1;
void enqueue(string data,int p)//Enqueue function to insert data and its priority in queue
{
    int i;
    if((f==0)&&(r==N-1)) //Check if Queue is full
        cout<<"Queue is full/n";
    else {
        if(f==-1) { //if Queue is empty
            f = r = 0;
            Q[r] = data;
            Pr[r] = p;

        }
        else if(r == N-1) { //if there there is some elemets in Queue
            for(i=f;i<=r;i++) {
          Q[i-f] = Q[i];
          Pr[i-f] = Pr[i];
          r = r-f;
          f = 0;
          for(i = r;i>f;i--) {
                    if(p>Pr[i]) {
                        Q[i+1] = Q[i];
                        Pr[i+1] = Pr[i];
                    }
                    else break;

                    Q[i+1] = data;
                    Pr[i+1] = p;
                    r++;
                }
            }
        }
        else {
            for(i = r;i>=f;i--) {
                if(p>Pr[i]) {
                    Q[i+1] = Q[i];
                    Pr[i+1] = Pr[i];
                }
                else break;
            }
            Q[i+1] = data;
            Pr[i+1] = p;
            r++;
        }
        }
}
void print() { //print the data of Queue
        int i;
        for(i=f;i<=r;i++) {
                cout << "Patient's Name - "<<Q[i];
                switch(Pr[i]) {
                        case 1:
                                cout << " Priority - 'Checkup' " << endl;
                        break;
                        case 5:
                                cout << " Priority - 'Non-serious' " << endl;
                        break;
                        case 10:
                                cout << " Priority - 'Serious' " << endl;
                        break;
                        default:
                                cout << "Priority not found" << endl;
                }
        }
}

void dequeue() { //remove the data from front
        if(f == -1) {
                cout<<"Queue is Empty";
        }
        else {
        cout<<endl<<"Deleted Element ="<<Q[f]<<endl;
        cout<<"Its Priority = "<<Pr[f]<<endl;
                if(f==r) f = r = -1;
                else f++;
        }
}

int main() {
        string data;
        int opt,n,i,p;
        do {
        cout <<endl<< "1 For Insert the Data in Queue" << endl << "2 For show the Data in
Queue " << endl << "3 For Delete the data from the Queue"
                << endl << "0 For Exit"<< endl;
        cout<<"Enter Your Choice: ";
        cin >> opt;
                switch(opt) {
                        case 1:
                                cout << "Enter the number of patient: ";
                                cin >> n;
                                i = 0;
                                while(i < n) {
                                        cout << "Enter your name of the patient: ";
                                        cin.ignore();
                                        getline(cin,data);
                                        ifnotdoagain:
                                                cout << "Enter your Prioritys (0: serious, 1: non-
serious, 2: genral checkup): "l;
                                                cin >> p;
                                                switch(p) {
                                                        case 0:
                                                                enqueue(data,SERIOUS);
                                                        break;
                                                        case 1:
                                                                enqueue(data,NONSERIOUS);
                                                        break;
                                                        case 2:
                                                                enqueue(data,CHECKUP);
                                                        break;
                                                        default:
                                                                goto ifnotdoagain;
                                                }

                                        i++;
                                }
                        break;
                        case 2:
                                print();
                        break;
                        case 3:
                                 dequeue();
                        break;
                        case 0:
                                cout << "Bye Bye !" << endl;
                        break;
                        default:
                        cout<<"Incorrect Choice"<<endl;

                }
        }while(opt!=0);
    return 0;
```

```
}
```

OUTPUT:

```
1 For Insert the Data in Queue
2 For show the Data in Queue
3 For Delete the data from the Queue
0 For Exit
Enter Your Choice: 1
Enter the number of patient: 3
Enter your name of the patient: AAA
Enter your Prioritys (0: serious, 1: non-serious, 2: genral checkup): 0
Enter your name of the patient: BBB
Enter your Prioritys (0: serious, 1: non-serious, 2: genral checkup): 2
Enter your name of the patient: CCC
Enter your Prioritys (0: serious, 1: non-serious, 2: genral checkup): 1

1 For Insert the Data in Queue
2 For show the Data in Queue
3 For Delete the data from the Queue
0 For Exit
Enter Your Choice: 2
Patient's Name - AAA Priority - 'Serious'
Patient's Name - CCC Priority - 'Non-serious'
Patient's Name - BBB Priority - 'Checkup'

1 For Insert the Data in Queue
2 For show the Data in Queue
3 For Delete the data from the Queue
0 For Exit
Enter Your Choice: 3

Deleted Element =AAA
Its Priority = 10

1 For Insert the Data in Queue
2 For show the Data in Queue
3 For Delete the data from the Queue
0 For Exit
Enter Your Choice: 3

Deleted Element =CCC
Its Priority = 5
```

```
1 For Insert the Data in Queue
2 For show the Data in Queue
3 For Delete the data from the Queue
0 For Exit
Enter Your Choice: 2
Patient's Name - BBB Priority - 'Checkup'

1 For Insert the Data in Queue
2 For show the Data in Queue
3 For Delete the data from the Queue
0 For Exit
Enter Your Choice: 0
Bye Bye !

--------------------------------
Process exited after 38.5 seconds with return value 0
Press any key to continue . . .
```

ANALYSIS:

The insertion and deletion operation in priority queue is done in O(log n) time. While the searching in done in O(1) time.

CONCLUSION: Thus, we have studied and implemented the concept of priority queue. We have also performed operations like addition and deletion on the priority queue.

EXPERIMENT – 11

AIM:

Department maintains a student information. The file contains roll number, name, division and address. Allow user to add, delete information of student. Display information of particular employee. If record of student does not exist an appropriate message is displayed. If it is, then the system displays the student details. Use sequential file to main the data.

OBJECTIVE:

1. To understand concept of file organization in data structure.
2. To understand concept & features of sequential file organization

THEORY:

File organization refers to the relationship of the key of the record to the physical location of that record in the computer file. File organization may be either physical file or a logical file. A physical file is a physical unit, such as magnetic tape or a disk. A logical file on the other hand is a complete set of records for a specific application or purpose. A logical file may occupy a part of physical file or may extend over more than one physical file.

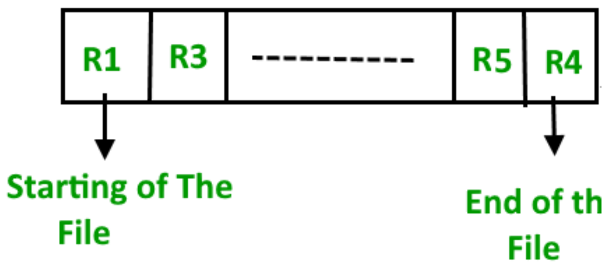
There are various methods of file organizations. These methods may be efficient for certain types of access/selection meanwhile it will turn inefficient for other selections. Hence it is up to the programmer to decide the best suited file organization method depending on his requirement.

Some of the file organizations are:

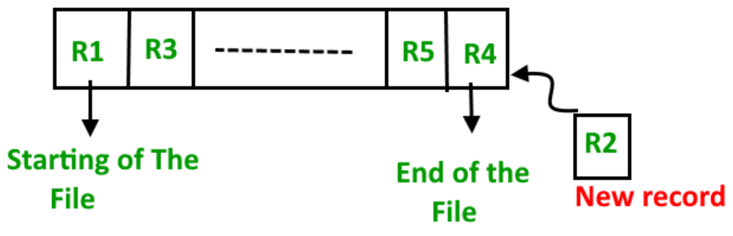
1. Sequential File Organization
2. Heap File Organization
3. 3. Hash/Direct File Organization
4. 4. Indexed Sequential Access Method
5. 5. B+ Tree File Organization
6. 6. Cluster File Organization

Sequential File Organization: The easiest method for file Organization is Sequential method. In this method the file are stored one after another in a sequential manner. There are two ways to implement this method:

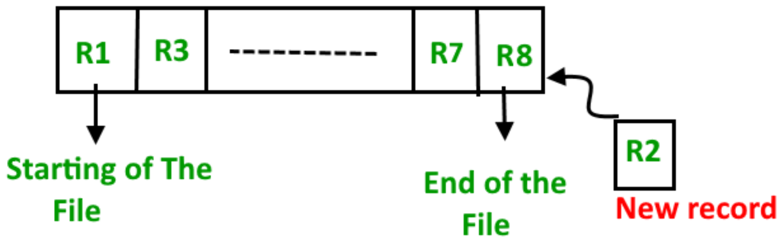
1. **Pile File Method –** This method is quite simple, in which we store the records in a sequence i.e one after other in the order in which they are inserted into the tables.



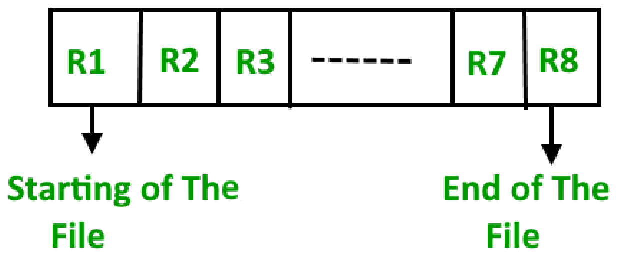
Insertion of new record –
Let the R1, R3 and so on upto R5 and R4 be four records in the sequence. Here, records are nothing but a row in any table. Suppose a new record R2 has to be inserted in the sequence, then it is simply placed at the end of the file.



2. **Sorted File Method –**In this method, As the name itself suggest whenever a new record has to be inserted, it is always inserted in a sorted (ascending or descending) manner. Sorting of records may be based on any primary key or any other key.



Insertion of new record –
Let us assume that there is a preexisting sorted sequence of four records R1, R3, and so on upto R7 and R8. Suppose a new record R2 has to be inserted in the sequence, then it will be inserted at the end of the file and then it will sort the sequence .



Pros and Cons of Sequential File Organization –

Pros –

- Fast and efficient method for huge amount of data.
- Simple design.
- Files can be easily stored in magnetic tapes i.e cheaper storage mechanism.

Cons –

- Time wastage as we cannot jump on a particular record that is required, but we have to move in a sequential manner which takes our time.
- Sorted file method is inefficient as it takes time and space for sorting records.

ALGORITHM:

STEP 1: Declare a class with the variables to store name, roll number, division and address of n students.

STEP 2: Open a file with .dat as extension to store the data of students.

STEP 3: Accept records of n students and store it in the dat file sequentially.

STEP 4: If user wants to display the file, then traverse the dat file and display each record along with all the details.

STEP 5: If user wants to delete a record from the file, then accept the roll number of the student from the user.

STEP 6: If the given roll number is present, then delete the record by initializing NULL values to all the fields of the particular record.

STEP 7: If the roll number is not present then display record not found.

STEP 8: If user wants to search a particular record from the file, then accept the roll number from the user.

STEP 9: Traverse the file and search for the given roll number.

STEP 10: If roll number is present, then display found, else display roll number not present.

Step 11: STOP

PROGRAM:

```
/*

Name: Harsh Shah

Class: SE Comp-1

Roll no: 59


Department maintains a student information. The file contains roll number,

name, division and address. Allow user to add, delete information of student.

Display information of particular employee. If record of student does not exist an

appropriate message is displayed. If it is, then the system displays the student

details. Use sequential file to main the data.

*/


#include<iostream>

#include<iomanip>

#include<fstream>
```

```
#include<cstring>

#include<stdlib.h>

using namespace std;


class STUDENT_CLASS

{

        typedef struct STUDENT

        {

                char name[10];

                int roll_no;

                char division;

                char address[50];

        }Rec;


        Rec Records;

        public:

                void Create();

                void Display();

                void Delete();

                int Search();

};


void STUDENT_CLASS::Create()

{
        char ch='y';

        fstream seqfile;

        seqfile.open("STUDENT.DAT",ios::app|ios::in|ios::out|ios::binary);

        do

        {

                cout<<"\n Enter Name: ";

                cin>>Records.name;

                cout<<"\n Enter roll no: ";

                cin>>Records.roll_no;

                cout<<"\n Enter Division: ";

                cin>>Records.division;

                cout<<"\n Enter address: ";

                cin>>Records.address;


                seqfile.write((char *)&Records,sizeof(Records));

                cout<<"\nDo you want to add more records?";

                cin>>ch;

        }while(ch=='y' || ch=='Y');

        seqfile.close();

}


void STUDENT_CLASS::Display()

{

        fstream seqfile;
```

```
          int n;

          seqfile.open("STUDENT.DAT",ios::in|ios::out|ios::binary);


          seqfile.seekg(0,ios::beg);

          cout<<"\n The Contents of file are ..."<<endl;


          while(seqfile.read((char *)&Records,sizeof(Records)))

          {

                    if(Records.roll_no!=-1)

                    {

                              cout<<"\n Nmae: "<<Records.name;

                              cout<<"\n Roll no: "<<Records.roll_no;

                              cout<<"\n Division: "<<Records.division;

                              cout<<"\n Address: "<<Records.address;

                              cout<<"\n";

                    }

          }

          int last_rec=seqfile.tellg();

          n=last_rec/sizeof(Rec);

          seqfile.close();

}


void STUDENT_CLASS::Delete()

{
          int pos;

          cout<<"\n For deletion";

          fstream seqfile;

          seqfile.open("STUDENT.DAT",ios::in|ios::out|ios::binary);

          seqfile.seekg(0,ios::beg);

          pos=Search();

          if(pos==-1)

          {

                    cout<<"\n The record is not present in the file";

                    return;

          }


          int offset=pos*sizeof(Rec);

          seqfile.seekp(offset);

          strcpy(Records.name,"");

          Records.roll_no=-1;

          Records.division='N';

          strcpy(Records.address,"");

          seqfile.write((char *)&Records,sizeof(Records))<<flush;

          seqfile.seekg(0);

          seqfile.close();

          cout<<"\n The record is Deleted !!!";

}
```

```
int STUDENT_CLASS::Search()

{

        fstream seqfile;

        int id,pos;

        cout<<"\n Enter the roll_no for searching the record: ";

        cin>>id;

        seqfile.open("STUDENT.DAT",ios::ate|ios::in|ios::out|ios::binary);

        seqfile.seekg(0);

        pos=-1;

        int i=0;

        while(seqfile.read((char *)&Records,sizeof(Records)))

        {

                if(id==Records.roll_no)

                {

                        pos=i;

                        break;

                }

                i++;

        }

        return pos;

}


int main()

{
```

```
STUDENT_CLASS List;
char ans='y';
int choice,key;

do
{
        cout<<"\n********Main Menu********"<<endl;
        cout<<"\n 1.Create";
        cout<<"\n 2.Display";
        cout<<"\n 3.Delete";
        cout<<"\n 4.Search";
        cout<<"\n 5.Exit";
        cout<<"\n Enter your choice: ";
        cin>>choice;
        switch(choice)
        {
                case 1:List.Create();
                        break;

                case 2:List.Display();
                        break;

                case 3:List.Delete();
                        break;
```

```
                    case 4:key=List.Search();
                    if(key<0)
                            cout<<"\n Record is not present in the file";
                    else
                            cout<<"\n Record is present in the file";
                    break;

                    case 7:exit(0);
            }
            cout<<"\n\t Do you want to go back to Main Menu?";
            cin>>ans;
    }while(ans=='y'||ans=='Y');
    return 0;
}
```

OUTPUT:

```
********Main Menu********

 1.Create
 2.Display
 3.Delete
 4.Search
```

```
5.Exit

Enter your choice: 1


Enter Name: Harsh


Enter roll no: 59


Enter Division: A


Enter address: Pune


Do you want to add more records?y


Enter Name: Kaustubh


Enter roll no: 37


Enter Division: A


Enter address: Jalgoan


Do you want to add more records?y


Enter Name: Onasvee
```

```
 Enter roll no: 16


 Enter Division: A


 Enter address: Pune


Do you want to add more records?n


         Do you want to go back to Main Menu?y


********Main Menu********


 1.Create

 2.Display

 3.Delete

 4.Search

 5.Exit

 Enter your choice: 2


 The Contents of file are ...


 Nmae: Harsh

 Roll no: 59
```

```
Division: A

Address: Pune


Nmae: Kaustubh

Roll no: 37

Division: A

Address: Jalgoan


Nmae: Onasvee

Roll no: 16

Division: A

Address: Pune


        Do you want to go back to Main Menu?y


********Main Menu********


1.Create

2.Display

3.Delete

4.Search

5.Exit

Enter your choice: 4
Enter the roll_no for searching the record: 59


Record is present in the file
        Do you want to go back to Main Menu?y

********Main Menu********

1.Create
2.Display
3.Delete
4.Search
5.Exit
Enter your choice: 3


For deletion
Enter the roll_no for searching the record: 59


The record is Deleted !!!
        Do you want to go back to Main Menu?y

********Main Menu********

1.Create
2.Display
```

```
3.Delete

4.Search

5.Exit

Enter your choice: 2


The Contents of file are ...


Nmae: Kaustubh

Roll no: 37

Division: A

Address: Jalgoan


Nmae: Onasvee

Roll no: 16

Division: A

Address: Pune


        Do you want to go back to Main Menu?n


--------------------------------

Process exited after 80.41 seconds with return value 0

Press any key to continue . . .
```

ANALYSIS:

Time Complexity: Creation, deletion, display and searching in the file is completed in O(n) time.

CONCLUSION: Thus, we have learned and implemented sequential file organization. We have created a file using sequential organization to store the details of students.

EXPERIMENT – 12

AIM:

Company maintains employee information as employee ID, name, designation and salary. Allow user to add, delete information of employee. Display information of particular employee. If employee does not exist an appropriate message is displayed. If it is, then the system displays the employee details. Use index sequential file to maintain the data.

OBJECTIVE:

1. To understand concept of file organization in data structure.
2. To understand concept & features of index sequential file organization

THEORY:

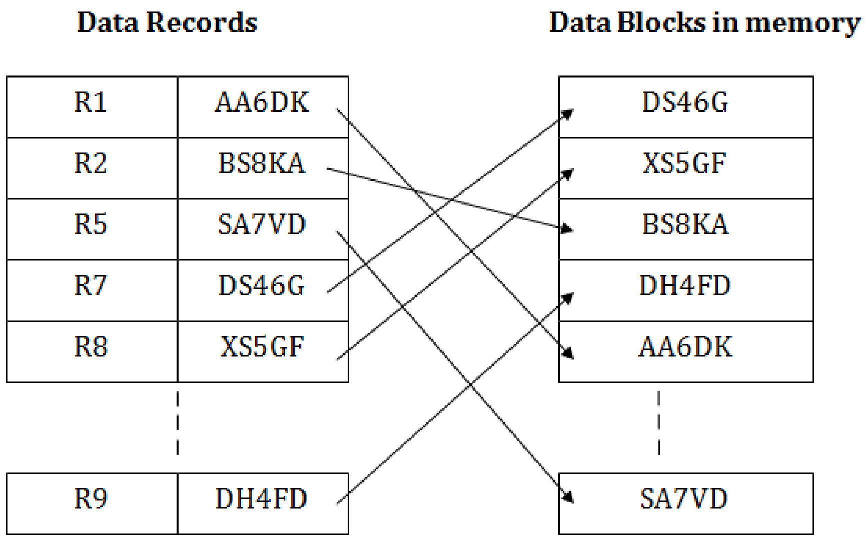
File Organization

File organization ensures that records are available for processing. It is used to determine an efficient file organization for each base relation.

For example, if we want to retrieve employee records in alphabetical order of name. Sorting the file by employee name is a good file organization. However, if we want to retrieve all employees whose marks are in a certain range, a file is ordered by employee name would not be a good file organization.

Indexed sequential access file organization

- Indexed sequential access file combines both sequential file and direct access file organization.
- In indexed sequential access file, records are stored randomly on a direct access device such as magnetic disk by a primary key.
- This file has multiple keys. These keys can be alphanumeric in which the records are ordered is called primary key.
- The data can be access either sequentially or randomly using the index. The index is stored in a file and read into memory when the file is opened.
- In index sequential file organization, a separate file for storing indexes of every record is maintained along with the master file.



Advantages of Indexed sequential access file organization

- In indexed sequential access file, sequential file and random file access is possible.
- It accesses the records very fast if the index table is properly organized.
- The records can be inserted in the middle of the file.
- It provides quick access for sequential and direct processing.
- It reduces the degree of the sequential search.

Disadvantages of Indexed sequential access file organization

- Indexed sequential access file requires unique keys and periodic reorganization.
- Indexed sequential access file takes longer time to search the index for the data access or retrieval.
- It requires more storage space.
- It is expensive because it requires special software.
- It is less efficient in the use of storage space as compared to other file organizations.

ALGORITHM:

STEP 1: Declare a class with the variables to store name, employee ID, designation and salary of n employees.

STEP 2: Open a file with .dat as extension to store the data of employees.

STEP 3: Accept records of n employees and store it in the dat file using index sequential file organisation.

STEP 4: If user wants to display the file, then traverse the dat file and display each record along with all the details.

STEP 5: If user wants to delete a record from the file, then accept the employee ID of the student from the user.

STEP 6: If the given employee ID is present, then delete the record by initializing NULL values to all the fields of the particular record.

STEP 7: If the employee ID is not present then display record not found.

STEP 8: If user wants to search a particular record from the file, then accept employee ID the from the user.

STEP 9: Traverse the file and search for the given employee ID.

STEP 10: If employee ID is present, then display found, else display employee ID not present.

Step 11: STOP

PROGRAM:

```
/*

Name: Harsh Shah

Class: SE Comp-1

Roll no: 59

Company maintains employee information as employee ID, name, designation and

salary. Allow user to add, delete information of employee. Display information

of particular employee. If employee does not exist an appropriate message is

displayed. If it is, then the system displays the employee details. Use index

sequential file to maintain the data.

*/
```

```
#include<iostream>

#include<iomanip>

#include<fstream>

#include<cstring>

#include<stdlib.h>

using namespace std;

class EMP_CLASS

{

        typedef struct EMPLOYEE

        {

                char name[10];

                int emp_id;

                int salary;

                char desig[50];

        }Rec;

        typedef struct INDEX

        {

                int emp_id;

                int position;

        }Ind_Rec;

        Rec Records;

        Ind_Rec Ind_Records;

        public:

                EMP_CLASS();
                void Create();
                void Display();
                void Delete();
                void Search();
};
EMP_CLASS::EMP_CLASS()
{
        strcpy(Records.name,"");
}
void EMP_CLASS::Create()
{
        int i;
        char ch='y';
        fstream seqfile;
        fstream indexfile;
        i=0;
        indexfile.open("IND.DAT",ios::app|ios::in|ios::out|ios::binary);
        seqfile.open("EMP.DAT",ios::app|ios::in|ios::out|ios::binary);
        do
        {
                cout<<"\n Enter Name: ";
                cin>>Records.name;
                cout<<"\n Enter EMP_ID: ";
                cin>>Records.emp_id;
                cout<<"\n Enter Salary: ";

                cin>>Records.salary;

                cout<<"\n Enter designation: ";

                cin>>Records.desig;


                seqfile.write((char *)&Records,sizeof(Records))<<flush;

                Ind_Records.emp_id=Records.emp_id;

                Ind_Records.position=i;

                indexfile.write((char *)&Ind_Records,sizeof(Ind_Records))<<flush;

                i++;

                cout<<"\nDo you want to add more records?";

                cin>>ch;

        }while(ch=='y' || ch=='Y');

        seqfile.close();

        indexfile.close();

}

void EMP_CLASS::Display()

{

        fstream seqfile;

        fstream indexfile;

        int i;

        seqfile.open("EMP.DAT",ios::in|ios::out|ios::binary);

        indexfile.open("IND.DAT",ios::in|ios::out|ios::binary);

        indexfile.seekg(0,ios::beg);
```

```
	seqfile.seekg(0,ios::beg);
	cout<<"\n The Contents of file are ..."<<endl;
	i=0;
	while(indexfile.read((char *)&Ind_Records,sizeof(Ind_Records)))
	{
		i=Ind_Records.position*sizeof(Rec);
		seqfile.seekg(i,ios::beg);
		seqfile.read((char *)&Records,sizeof(Records));
		if(Records.emp_id!=-1)
		{
			cout<<"\n Name: "<<Records.name;
			cout<<"\n Emp_ID: "<<Records.emp_id;
			cout<<"\n Salary: "<<Records.salary;
			cout<<"\n Designation: "<<Records.desig;
			cout<<"\n";
		}
	}
	seqfile.close();
	indexfile.close();
}
void EMP_CLASS::Delete()
{
	int id,pos;
	cout<<"\n For deletion";
```

```
cout<<"\n Enter the Emp_ID for deletion: ";

cin>>id;

fstream seqfile;

fstream indexfile;

seqfile.open("EMP.DAT",ios::in|ios::out|ios::binary);

indexfile.open("IND.DAT",ios::in|ios::out|ios::binary);

seqfile.seekg(0,ios::beg);

indexfile.seekg(0,ios::beg);

pos=-1;


while(indexfile.read((char *)&Ind_Records,sizeof(Ind_Records)))

{

        if(id==Ind_Records.emp_id)

        {

                pos=Ind_Records.position;

                Ind_Records.emp_id=-1;

                break;

        }

}

if(pos==-1)

{

        cout<<"\n The record is not present in the file";

        return;

}
```

```
        int offset=pos*sizeof(Rec);

        seqfile.seekp(offset);

        strcpy(Records.name,"");

        Records.emp_id=-1;

        Records.salary=-1;

        strcpy(Records.desig,"");

        seqfile.write((char *)&Records,sizeof(Records))<<flush;


        offset=pos*sizeof(Ind_Rec);

        indexfile.seekp(offset);

        Ind_Records.emp_id=-1;

        Ind_Records.position=pos;

        indexfile.write((char *)&Ind_Records,sizeof(Ind_Records))<<flush;

        seqfile.seekg(0);

        indexfile.close();

        seqfile.close();

        cout<<"\n The record is Deleted !!!";

}

void EMP_CLASS::Search()

{

        fstream seqfile;

        fstream indexfile;

        int id,pos,offset;
```

```
cout<<"\n Enter the emp_id for searching the record: ";
cin>>id;
indexfile.open("IND.DAT",ios::in|ios::binary);
pos=-1;

while(indexfile.read((char *)&Ind_Records,sizeof(Ind_Records)))
{
        if(id==Ind_Records.emp_id)
        {
                pos=Ind_Records.position;
                break;
        }
}
if(pos==-1)
{
        cout<<"\n Record is not present in the file";
        return;
}
offset=pos*sizeof(Records);
seqfile.open("EMP.DAT",ios::in|ios::binary);
seqfile.seekp(offset,ios::beg);
seqfile.read((char *)&Records,sizeof(Records));
if(Records.emp_id==-1)
{
```

```
                cout<<"\n Record is not present in the file";
                return;
        }
        else
        {
                cout<<"\n Name: "<<Records.name;
                cout<<"\n Emp_ID: "<<Records.emp_id;
                cout<<"\n Salary: "<<Records.salary;
                cout<<"\n Designation: "<<Records.desig;
        }
        seqfile.close();
        indexfile.close();
}
int main()
{
        EMP_CLASS List;
        char ans='y';
        int choice;

        do
        {
                cout<<"\n********Main Menu********"<<endl;
                cout<<"\n 1.Create";
                cout<<"\n 2.Display";
```

```
            cout<<"\n 3.Delete";

            cout<<"\n 4.Search";

            cout<<"\n 5.Exit";

            cout<<"\n Enter your choice: ";

            cin>>choice;

            switch(choice)

            {

                        case 1:List.Create();

                                    break;


                        case 2:List.Display();

                                    break;


                        case 3:List.Delete();

                                    break;


                        case 4:List.Search();

                                    break;


                        case 7:exit(0);

            }

            cout<<"\n\t Do you want to go back to Main Menu?";

            cin>>ans;

}while(ans=='y'||ans=='Y');
```

```
        return 0;
}
```

OUTPUT:

```
********Main Menu********

 1.Create
 2.Display
 3.Delete
 4.Search
 5.Exit
 Enter your choice: 1

 Enter Name: Harsh

 Enter EMP_ID: 123

 Enter Salary: 1200000

 Enter designation: Manager

Do you want to add more records?y
```

```
 Enter Name: AAA

 Enter EMP_ID: 124

 Enter Salary: 800000

 Enter designation: Assistant

Do you want to add more records?y

 Enter Name: BBB

 Enter EMP_ID: 127

 Enter Salary: 750000

 Enter designation: HR

Do you want to add more records?n

        Do you want to go back to Main Menu?y

********Main Menu********
```

```
1.Create

2.Display

3.Delete

4.Search

5.Exit

Enter your choice: 2


The Contents of file are ...


Name: Harsh

Emp_ID: 123

Salary: 1200000

Designation: Manager


Name: AAA

Emp_ID: 124

Salary: 800000

Designation: Assistant


Name: BBB

Emp_ID: 127

Salary: 750000

Designation: HR
```

```
        Do you want to go back to Main Menu?y

********Main Menu********


 1.Create

 2.Display

 3.Delete

 4.Search

 5.Exit

 Enter your choice: 4


 Enter the emp_id for searching the record: 123


 Name: Harsh

 Emp_ID: 123

 Salary: 1200000

 Designation: Manager

        Do you want to go back to Main Menu?y

********Main Menu********


 1.Create

 2.Display

 3.Delete
```

```
4.Search

5.Exit

Enter your choice: 3


For deletion

Enter the Emp_ID for deletion: 124


The record is Deleted !!!

        Do you want to go back to Main Menu?y


********Main Menu********


1.Create

2.Display

3.Delete

4.Search

5.Exit

Enter your choice: 2


The Contents of file are ...


Name: Harsh

Emp_ID: 123

Salary: 1200000
```

```
Designation: Manager


Name: BBB

Emp_ID: 127

Salary: 750000

Designation: HR


        Do you want to go back to Main Menu?n


--------------------------------

Process exited after 107 seconds with return value 0

Press any key to continue . . .
```

ANALYSIS:

Time Complexity: Creation, deletion, display and searching in the file is completed in O(n) time.

CONCLUSION: Thus, we have learned and implemented index sequential file organization. We have created a file using index sequential organization to store the details of employees.

EXPERIMENT – 13

(MINI PROJECT)

AIM:

Design a mini project to implement a Smart text editor.

OBJECTIVE:

3. To create a smart text editor in python.
4. To learn about Tkinter module in python.

THEORY:

Text Editor

A text editor is a type of computer program that edits plain text. Such programs are sometimes known as "notepad" software, following the naming of Microsoft Notepad. Text editors are provided with operating systems and software development packages, and can be used to change files such as configuration files, documentation files and programming language source code. Some features of text editor are:

- Find and replace – Text editors provide extensive facilities for searching and replacing text, either on groups of files or interactively. Advanced editors can use regular expressions to search and edit text or code.
- Cut, copy, and paste – most text editors provide methods to duplicate and move text within the file, or between files.
- Text formatting – Text editors often provide basic visual formatting features like line wrap, auto-indentation, bullet list formatting using ASCII characters, comment formatting, syntax highlighting and so on. These are typically only for display and do not insert formatting codes into the file itself.
- Undo and redo – As with word processors, text editors provide a way to undo and redo the last edit, or more. Often—especially with older text editors—there is only one level of edit history remembered and successively issuing the undo command will only "toggle" the last change. Modern or more complex editors usually provide a multiple-level history such that issuing the undo command repeatedly will revert the document to successively older edits. A separate redo command will cycle the edits "forward" toward the most recent changes.

The number of changes remembered depends upon the editor and is often configurable by the user.

Tkinter Programming

Tkinter is the standard GUI library for Python. Python when combined with Tkinter provides a fast and easy way to create GUI applications. Tkinter provides a powerful object-oriented interface to the Tk GUI toolkit.

Creating a GUI application using Tkinter is an easy task. All you need to do is perform the following steps −

- Import the *Tkinter* module.
- Create the GUI application main window.
- Add one or more of the above-mentioned widgets to the GUI application.
- Enter the main event loop to take action against each event triggered by the user.

Standard attributes

Let us take a look at how some of their common attributes.such as sizes, colors and fonts are specified.

- Dimensions
- Colors
- Fonts
- Anchors
- Relief styles
- Bitmaps
- Cursors

PROGRAM:

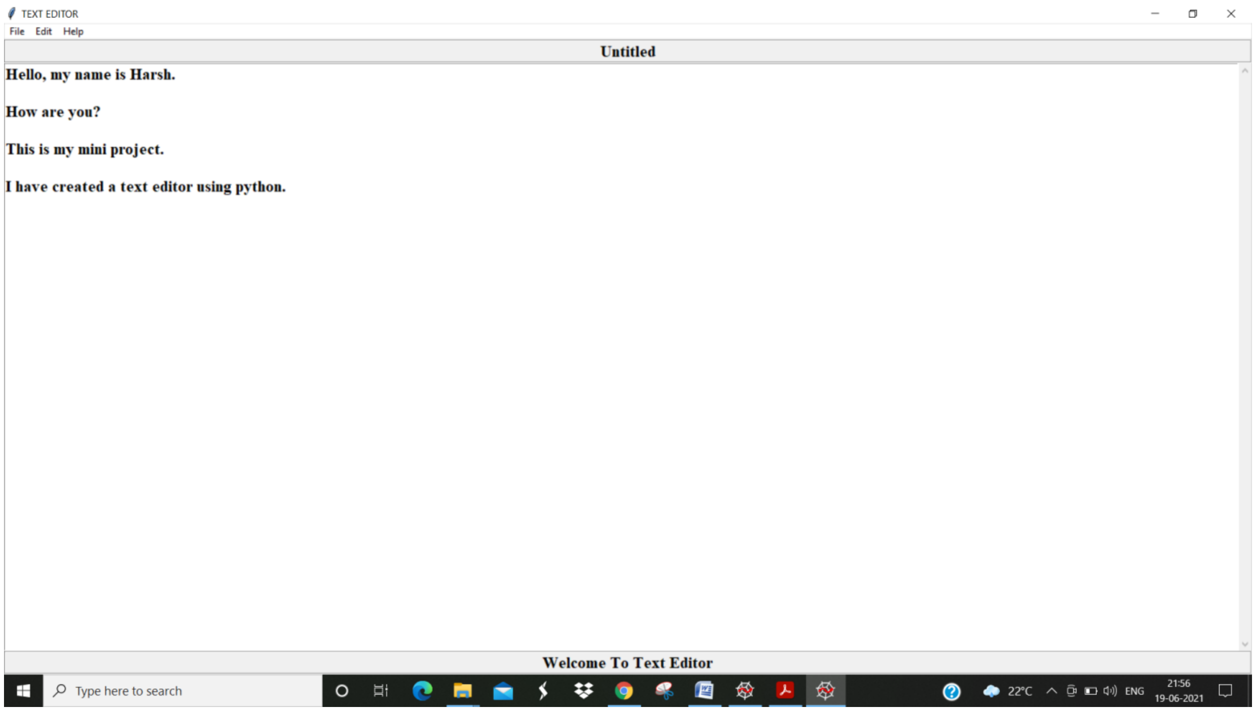
```
# Importing Required libraries & Modules

from tkinter import *

from tkinter import messagebox

from tkinter import filedialog

# Defining TextEditor Class
```

```
class TextEditor:

 # Defining Constructor

 def __init__(self,root):

   # Assigning root

   self.root = root

   # Title of the window

   self.root.title("TEXT EDITOR")

   # Window Geometry

   self.root.geometry("1200x700+200+150")

   # Initializing filename

   self.filename = None

   # Declaring Title variable

   self.title = StringVar()

   # Declaring Status variable

   self.status = StringVar()

   # Creating Titlebar

   self.titlebar = Label(self.root,textvariable=self.title,font=("times new
roman",15,"bold"),bd=2,relief=GROOVE)

   # Packing Titlebar to root window

   self.titlebar.pack(side=TOP,fill=BOTH)

   # Calling Settitle Function

   self.settitle()

   # Creating Statusbar
   self.statusbar = Label(self.root,textvariable=self.status,font=("times new roman",15,"bold"),bd=2,relief=GROOVE)

   # Packing status bar to root window

   self.statusbar.pack(side=BOTTOM,fill=BOTH)

   # Initializing Status

   self.status.set("Welcome To Text Editor")

   # Creating Menubar

   self.menubar = Menu(self.root,font=("times new roman",15,"bold"),activebackground="skyblue")

   # Configuring menubar on root window

   self.root.config(menu=self.menubar)

   # Creating File Menu

   self.filemenu = Menu(self.menubar,font=("times new roman",12,"bold"),activebackground="skyblue",tearoff=0)

   # Adding New file Command

   self.filemenu.add_command(label="New",accelerator="Ctrl+N",command=self.newfile)

   # Adding Open file Command

   self.filemenu.add_command(label="Open",accelerator="Ctrl+O",command=self.openfile)

   # Adding Save File Command

   self.filemenu.add_command(label="Save",accelerator="Ctrl+S",command=self.savefile)

   # Adding Save As file Command

   self.filemenu.add_command(label="Save As",accelerator="Ctrl+A",command=self.saveasfile)

   # Adding Seprator

   self.filemenu.add_separator()
   # Adding Exit window Command

   self.filemenu.add_command(label="Exit",accelerator="Ctrl+E",command=self.exit)

   # Cascading filemenu to menubar

   self.menubar.add_cascade(label="File", menu=self.filemenu)

   # Creating Edit Menu

   self.editmenu = Menu(self.menubar,font=("times new
roman",12,"bold"),activebackground="skyblue",tearoff=0)

   # Adding Cut text Command

   self.editmenu.add_command(label="Cut",accelerator="Ctrl+X",command=self.cut)

   # Adding Copy text Command

   self.editmenu.add_command(label="Copy",accelerator="Ctrl+C",command=self.copy)

   # Adding Paste text command

   self.editmenu.add_command(label="Paste",accelerator="Ctrl+V",command=self.paste)

   # Adding Seprator

   self.editmenu.add_separator()

   # Adding Undo text Command

   self.editmenu.add_command(label="Undo",accelerator="Ctrl+U",command=self.undo)

   # Cascading editmenu to menubar

   self.menubar.add_cascade(label="Edit", menu=self.editmenu)

   # Creating Help Menu

   self.helpmenu = Menu(self.menubar,font=("times new
roman",12,"bold"),activebackground="skyblue",tearoff=0)

   # Adding About Command

   self.helpmenu.add_command(label="About",command=self.infoabout)
```

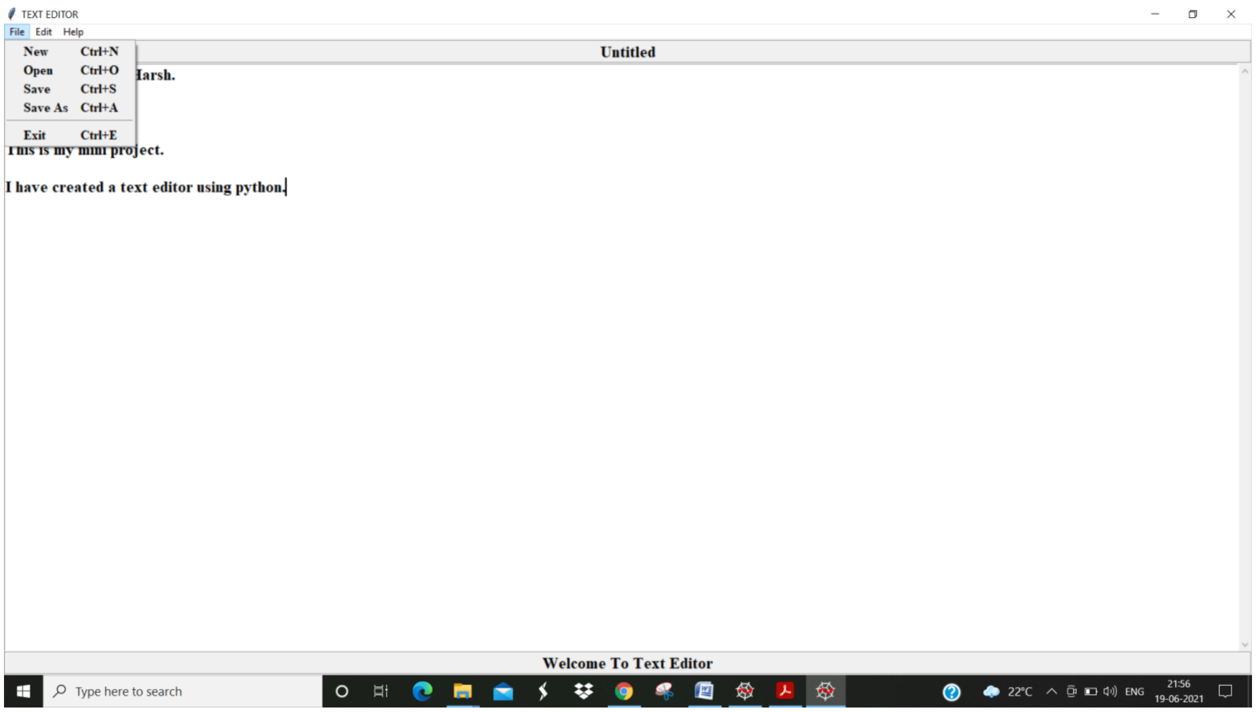
```
  # Cascading helpmenu to menubar

  self.menubar.add_cascade(label="Help", menu=self.helpmenu)

  # Creating Scrollbar

  scrol_y = Scrollbar(self.root,orient=VERTICAL)

  # Creating Text Area

  self.txtarea = Text(self.root,yscrollcommand=scrol_y.set,font=("times new roman",15,"bold"),state="normal",relief=GROOVE)

  # Packing scrollbar to root window

  scrol_y.pack(side=RIGHT,fill=Y)

  # Adding Scrollbar to text area

  scrol_y.config(command=self.txtarea.yview)

  # Packing Text Area to root window

  self.txtarea.pack(fill=BOTH,expand=1)

  # Calling shortcuts funtion

  self.shortcuts()

 # Defining settitle function

 def settitle(self):

  # Checking if Filename is not None

  if self.filename:

    # Updating Title as filename

    self.title.set(self.filename)

  else:

    # Updating Title as Untitled
```

```
      self.title.set("Untitled")

  # Defining New file Function

  def newfile(self,*args):

    # Clearing the Text Area

    self.txtarea.delete("1.0",END)

    # Updating filename as None

    self.filename = None

    # Calling settitle funtion

    self.settitle()

    # updating status

    self.status.set("New File Created")

  # Defining Open File Funtion

  def openfile(self,*args):

    # Exception handling

    try:

      # Asking for file to open

      self.filename = filedialog.askopenfilename(title = "Select file",filetypes = (("All
Files","*.*"),("Text Files","*.txt"),("Python Files","*.py")))

      # checking if filename not none

      if self.filename:

        # opening file in readmode

        infile = open(self.filename,"r")

        # Clearing text area
        self.txtarea.delete("1.0",END)
        # Inserting data Line by line into text area
        for line in infile:
          self.txtarea.insert(END,line)
        # Closing the file
        infile.close()
        # Calling Set title
        self.settitle()
        # Updating Status
        self.status.set("Opened Successfully")
    except Exception as e:
      messagebox.showerror("Exception",e)
  # Defining Save File Funtion
  def savefile(self,*args):
    # Exception handling
    try:
      # checking if filename not none
      if self.filename:
        # Reading the data from text area
        data = self.txtarea.get("1.0",END)
        # opening File in write mode
        outfile = open(self.filename,"w")
        # Writing Data into file
        outfile.write(data)

        # Closing File

        outfile.close()

        # Calling Set title

        self.settitle()

        # Updating Status

        self.status.set("Saved Successfully")

      else:

        self.saveasfile()

    except Exception as e:

      messagebox.showerror("Exception",e)

  # Defining Save As File Funtion

  def saveasfile(self,*args):

    # Exception handling

    try:

      # Asking for file name and type to save

      untitledfile = filedialog.asksaveasfilename(title = "Save file As",defaultextension=".txt",initialfile = "Untitled.txt",filetypes = (("All Files","*.*"),("Text Files","*.txt"),("Python Files","*.py")))

      # Reading the data from text area

      data = self.txtarea.get("1.0",END)

      # opening File in write mode

      outfile = open(untitledfile,"w")

      # Writing Data into file
      outfile.write(data)

      # Closing File

      outfile.close()

      # Updating filename as Untitled

      self.filename = untitledfile

      # Calling Set title

      self.settitle()

      # Updating Status

      self.status.set("Saved Successfully")

    except Exception as e:

      messagebox.showerror("Exception",e)

  # Defining Exit Funtion

  def exit(self,*args):

    op = messagebox.askyesno("WARNING","Your Unsaved Data May be Lost!!")

    if op>0:

      self.root.destroy()

    else:

      return

  # Defining Cut Funtion

  def cut(self,*args):

    self.txtarea.event_generate("<<Cut>>")

  # Defining Copy Funtion

  def copy(self,*args):
      self.txtarea.event_generate("<<Copy>>")
  # Defining Paste Funtion
  def paste(self,*args):
    self.txtarea.event_generate("<<Paste>>")
  # Defining Undo Funtion
  def undo(self,*args):
    # Exception handling
    try:
      # checking if filename not none
      if self.filename:
        # Clearing Text Area
        self.txtarea.delete("1.0",END)
        # opening File in read mode
        infile = open(self.filename,"r")
        # Inserting data Line by line into text area
        for line in infile:
          self.txtarea.insert(END,line)
        # Closing File
        infile.close()
        # Calling Set title
        self.settitle()
        # Updating Status
        self.status.set("Undone Successfully")
      else:
        # Clearing Text Area
        self.txtarea.delete("1.0",END)
        # Updating filename as None
        self.filename = None
        # Calling Set title
        self.settitle()
        # Updating Status
        self.status.set("Undone Successfully")
    except Exception as e:
      messagebox.showerror("Exception",e)
  # Defining About Funtion
  def infoabout(self):
    messagebox.showinfo("About Text Editor","A Simple Text Editor\nCreated using Python.")
  # Defining shortcuts Funtion
  def shortcuts(self):
    # Binding Ctrl+n to newfile funtion
    self.txtarea.bind("<Control-n>",self.newfile)
    # Binding Ctrl+o to openfile funtion
    self.txtarea.bind("<Control-o>",self.openfile)
    # Binding Ctrl+s to savefile funtion
    self.txtarea.bind("<Control-s>",self.savefile)
    # Binding Ctrl+a to saveasfile funtion
    self.txtarea.bind("<Control-a>",self.saveasfile)
    # Binding Ctrl+e to exit funtion
    self.txtarea.bind("<Control-e>",self.exit)
    # Binding Ctrl+x to cut funtion
    self.txtarea.bind("<Control-x>",self.cut)
    # Binding Ctrl+c to copy funtion
    self.txtarea.bind("<Control-c>",self.copy)
    # Binding Ctrl+v to paste funtion
    self.txtarea.bind("<Control-v>",self.paste)
    # Binding Ctrl+u to undo funtion
    self.txtarea.bind("<Control-u>",self.undo)
# Creating TK Container
root = Tk()
# Passing Root to TextEditor Class
TextEditor(root)
# Root Window Looping
root.mainloop()
```

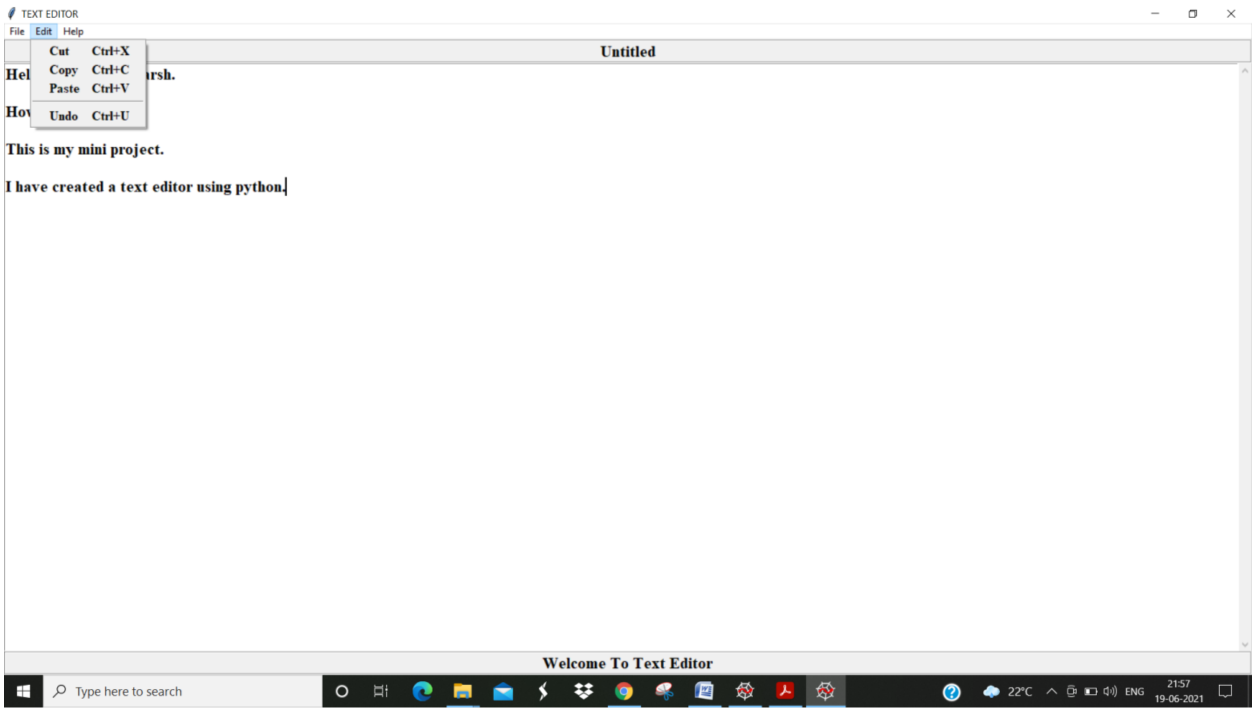
OUTPUT:



```
Hello, my name is Harsh.

How are you?

This is my mini project.

I have created a text editor using python.
```



```
Harsh.

This is my mini project.

I have created a text editor using python.
```



Hel rsh.

Hov

This is my mini project.

I have created a text editor using python.

CONCLUSION: Hence, we have successfully created a smart text editor in python.