LABORATORY MANUAL

DATA STRUCTURES AND ALGORITHMS LABORATORY (210256)

Class: Second Year

Course: 2019

DEPARTMENT OF COMPUTER ENGINEERING,

NBN SINHGAD SCHOOL OF ENGINEERING, PUNE-41

LIST OF ASSIGNMENTS

Sr. No.	Name of the Assignments
1	Consider telephone book database of N clients. Make use of a hash table implementation to quickly look up client's telephone number. Make useof two collision handling techniques and compare them using number of comparisons required to find a set of telephone numbers
2	To create ADT that implement the "set" concept. a. Add (new Element) -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
3	A book consists of chapters, chapters consist of sections and sections consist of subsections. Construct a tree and print the nodes. Find the time and space requirements of your method.
4	A Dictionary stores keywords and its meanings. Provide facility for adding new keywords, deleting keywords, updating values of any entry. Provide facility to display whole data sorted in ascending/ Descending order. Also find how many maximum comparisons may require for finding any keyword. Use Binary Search Tree for implementation.
5	Convert given binary tree into threaded binary tree. Analyze time and space complexity of the algorithm.
6	There are flight paths between cities. If there is a flight between city A and city B then there is an edge between the cities. The cost of the edge can be the time that flight takes to reach city B from A, or the amount of fuel used for the journey. Represent this as a graph. The node can be represented by airport name or name of the city. Use adjacency list representation of the graph or use adjacency matrix representation of the graph. Justify the storage representation used
7	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.
8	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?
9	A Dictionary stores keywords and its meanings. Provide facility for adding new keywords, deleting keywords, updating values of any entry. Provide facility to display whole data sorted in ascending/ Descending order. Also find how many maximum comparisons may require for finding any keyword. Use Height balance tree and find the complexity for finding a keyword
10	Read the marks obtained by students of second year in an online examination of particular subject. Find out maximum and minimum marks obtained in that subject. Use heap data structure. Analyze the algorithm

11	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.
12	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.
13	**MINI Project**

ASSIGNMENT NO. –0 1

TITLE: _____

NAME OF THE STUDENT:_____

ROLL NO: _____

DATE OF PERFORMANCE:

DATE OF SUBMISSION:

SIGNATURE:

ASSIGNMENT NO.-01

Title: 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

Objectives:

1. To understand concept of Hashing
2. To understand to find record quickly using hash function.
3. To understand concept & features of object oriented programming.

Learning Objectives

- ✓ To understand concept of hashing.
- ✓ To understand operations like insert and search record in the database.

Learning Outcome

- ✓ Learn object oriented Programming features
- ✓ Understand & implement concept of hash table **.**

Theory:

Hash tables are an efficient implementation of a keyed array data structure, a structure sometimes known as an associative array or map. If you're working in C++, you can take advantage of the STL map container for keyed arrays implemented using binary trees, but this article will give you some of the theory behind how a hash table works.

Keyed Arrays vs. Indexed Arrays

One of the biggest drawbacks to a language like C is that there are no keyed arrays. In a normal C array (also called an indexed array), the only way to access an element would be through its index number. To find element 50 of an array named "employees" you have to access it like this:

1employees[50];

In a keyed array, however, you would be able to associate each element with a "key," which can be anything from a name to a product model number. So, if you have a keyed array of employee records, you could access the record of employee "John Brown" like this:

1employees["Brown, John"];

One basic form of a keyed array is called the hash table. In a hash table, a key is used to find an element instead of an index number. Since the hash table has to be coded using an indexed array, there has to be some way of transforming a key to an index number. That way is called the hashing function.

Hashing Functions

A hashing function can be just about anything. How the hashing function is actually coded depends on the situation, but generally the hashing function should return a value based on a key and the size of the array the hashing table is built on. Also, one important thing that is sometimes overlooked is that a hashing function has to return the same value every time it is given the same key.

Let's say you wanted to organize a list of about 200 addresses by people's last names. A hash table would be ideal for this sort of thing, so that you can access the records with the people's last names as the keys.

First, we have to determine the size of the array we're using. Let's use a 260 element array so that there can be an average of about 10 element spaces per letter of the alphabet.>

Now, we have to make a hashing function. First, let's create a relationship between letters and numbers:

A --> 0

B --> 1

C --> 2

D --> 3

and so on until Z --> 25.

The easiest way to organize the hash table would be based on the first letter of the last name.

Since we have 260 elements, we can multiply the first letter of the last name by 10. So, when a key like "Smith" is given, the key would be transformed to the index 180 (S is the 19 letter of the alphabet, so S --> 18, and 18 * 10 = 180).

Since we use a simple function to generate an index number quickly, and we use the fact that the index number can be used to access an element directly, a hash table's access time is quite small. A linked list of keys and elements wouldn't be nearly as fast, since you would have to search through every single key-element pair.

Basic Operations

Following are the basic primary operations of a hash table.

- ☐ **Search** − Searches an element in a hash table.
- ☐ **Insert** − inserts an element in a hash table.
- ☐ **delete** − Deletes an element from a hash table.

DataItem

Define a data item having some data and key, based on which the search is to be conducted in a hash table.

```
structDataItem
{
  int data;
  int key;
};
```

Hash Method

Define a hashing method to compute the hash code of the key of the data item.

```
inthashCode(int key){

  return key % SIZE;
}
```

Search Operation

Whenever an element is to be searched, compute the hash code of the key passed and locate the element using that hash code as index in the array. Use linear probing to get the element ahead if the element is not found at the computed hash code.

Example

```
structDataItem *search(int key)
{
  //get the hash
  inthashIndex = hashCode(key);
```

```
  //move in array until an empty
  while(hashArray[hashIndex] != NULL) {

    if(hashArray[hashIndex]->key == key)
      returnhashArray[hashIndex];

    //go to next cell
    ++hashIndex;
    hashIndex %= SIZE;
  }
  return NULL;
}
```

Insert Operation

Whenever an element is to be inserted, compute the hash code of the key passed and locate the index using that hash code as an index in the array. Use linear probing for empty location, if an element is found at the computed hash code.

Example
```
void insert(intkey,int data)
{
  structDataItem *item = (structDataItem*) malloc(sizeof(structDataItem));
  item->data = data;
  item->key = key;
  //get the hash
  inthashIndex = hashCode(key);
  //move in array until an empty or deleted cell
  while(hashArray[hashIndex] != NULL
    &&hashArray[hashIndex]->key != -1) { //go to next cell

    ++hashIndex;
    //wrap around the table
    hashIndex %= SIZE;
  }
  hashArray[hashIndex] = item;
}
```

Delete Operation

Whenever an element is to be deleted, compute the hash code of the key passed and locate the index using that hash code as an index in the array. Use linear probing to get the element ahead if an element is not found at the computed hash code. When found, store a dummy item there to keep the performance of the hash table intact.

Example

```
structDataItem* delete(structDataItem* item) {
  int key = item->key;
  //get the hash
  inthashIndex = hashCode(key);
  //move in array until an empty
  while(hashArray[hashIndex] !=NULL) {
    if(hashArray[hashIndex]->key == key) {
      structDataItem* temp = hashArray[hashIndex];
      //assign a dummy item at deleted position
      hashArray[hashIndex] = dummyItem;
      return temp;
    }
    //go to next cell
    ++hashIndex;
    //wrap around the table
    hashIndex %= SIZE;
  }
  return NULL;
}
```

Expected Output

Conclusion: In this way we have implemented Hash table for quick lookup using C++.

Assignment Questions:

1.What is Hash Function?

2.what is Good Hash function?

3.How many ways are there to implement hash function?

4.What are the Collision Resolution Strategies?

5.What is the Hash Table Overflow?

ASSIGNMENT NO. – 02

TITLE: _____

 NAME OF THE STUDENT:_____

ROLL NO: _____

DATE OF PERFORMANCE:

DATE OF SUBMISSION:

SIGNATURE:

ASSIGNMENT NO.-02

Title:

To create ADT that implement the "set" concept. a. Add (new Element) -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

Objectives:

1. To understand concept of set and set ADT.
2. To understand concept python language.

Learning Objectives:
  - ✓ To understand concept of set.
  - ✓ To understand concept ADT.
  - ✓ To understand python language

Learning Outcome:
  - Define classes to show the concept of cohesion and coupling.
  - Analyze working of functions.

Theory:

SETS:

Sets are a type of abstract data type that allows you to store a list of non-repeated values. Their name derives from the mathematical concept of finite sets.

Unlike an array, sets are unordered and unindexed. You can think about sets as a room full of people you know. They can move around the room, changing order, without altering the set of people in that room. Plus, there are no duplicate people (unless you know someone who has cloned themselves). These are the two properties of a set: the data is unordered and it is not duplicated.

Sets have the most impact in mathematical set theory. These theories are used in many kinds of proofs, structures, and abstract algebra. Creating relations from different sets and codomains are also an important applications of sets.

In computer science, set theory is useful if you need to collect data and do not care about their multiplcity or their order. As we've seen on this page, hash tables and sets are very related. In databases, especially for relational databases, sets are very useful. There are many commands that finds unions, intersections, and differences of different tables and sets of data.

The set has four basic operations.

Function Name*	Provided Functionality
insert(i)	Adds *i* to the set
remove(i)	Removes *i* from the set
size()	Returns the size of the set
contains(i)	Returns whether or not the set contains *i*

Sometimes, operations are implemented that allow interactions between two sets

Function Name*	Provided Functionality
union(S, T)	Returns the union of set *S* and set *T*
intersection(S, T)	Returns the intersection of set *S* and set *T*
difference(S, T)	Returns the difference of set *S* and set *T*
subset(S, T)	Returns whether or not set *S* is a subset of set *T*

Sample Python Implementation Using a Dictionary

The only way to adequately implement a set in python is by using a dictionary, or a hash table. This is because the dictionary is the only primitive data structure whose elements are unordered. This require a few more lines of code, but it keeps the sets' principles intact.

Note: Python also has its own Set primitive, but we want to implement our own to show how it works

```
1 class Set:
2     def __init__(self, data=None):
3         self.data = {}
4         if data != None:
5             if len(data) != len(set(data)):
6                 data = set(data)
7             for d in data:
8                 self.data[d] = d
9     def insert(self, i):
10        if i in self.data.keys():
11            return 'Already in set'
12        self.data[i] = i
13    def remove(self, i):
14        if i not in self.data.keys():
15            return 'Not in set'
16        self.data.pop(i)
17    def size(self):
18        return len(self.data.keys())
19    def contains(self, i):
20        if i in self.data.keys():
21            return True
22        return False
23    def union(self, otherSet):
```

```
24          setData = list(set(self.data.keys()) | set(otherSet.data.keys()))
25          unionSet = Set(setData)
26          return unionSet
27      def intersection(self, otherSet):
28          setData = list(set(self.data.keys()) & set(otherSet.data.keys()))
29          intersectionSet = Set(setData)
30          return intersectionSet
31      def difference(self, otherSet):
32          setData = list(set(self.data.keys()) ^ set(otherSet.data.keys()))
33          differenceSet = Set(setData)
34          return differenceSet
35      def subset(self, otherSet):
36          if set(self.data.keys()) < set(otherSet.data.keys()):
37              return True
38          return False
39      def __repr__(self):
40        return '['+', '.join(str(x) for x in self.data.keys())+']'
```

This code is a little more complex than other data types like the associative array, so let's break it down.

On line 2, we instantiate our class using a python dictionary as our data structure. We use a dictionary to keep the set unordered (dictionaries are unordered because they are implemented using a hash table).

The functions `insert`, `remove`, `size`, and `contains` all simply search through the dictionary's list of keys to determine how to proceed with the operation.

The functions `union`, `intersection`, `difference`, and `subset` all utilize difference pythonic set operators.

For example, in the `union` function, we first turn both Set's data into a `set()`. Then we say we'll take data values in either set and put them in a new set. We do that using the OR operator. This looks like a `|`. Then, we change the `set()` into a `list()` because that is what our class takes as an argument. Then we create a new instance of our `Set`, and we return it. There are many other operators such as `&` and `<`.

Built-in Set Functionality in Python

Now let's look at the built-in set primitive in Python. This is *not* the same implementation that used above. This is data structure provided to you by Python itself. It is very similar to our implementation, but it is far more efficient and has many more functions.

Here are the basic functions:

```
1 >>> set1 = set(['Alex', 'Brittany', 'Joyce'])
2 >>> set1.add('John')
3 >>> set1
4 set(['John', 'Alex', 'Brittany', 'Joyce'])
```

```
5 >>> set1.remove('Alex')
6 >>> set1
7 set(['John', 'Brittany', 'Joyce'])
```

Here are functions that relate sets to one another:

```
1 >>> odds = set([1, 3, 5, 7, 9])
2 >>> evens = set([0, 2, 4, 6, 8])
3 >>> odds | evens                  #Performs a union
4 set([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
5 >>> odds & evens                  #Performs an intersection
6 set([])
7 >>> evens - odds                  #Performs a difference
8 set([0, 8, 2, 4, 6])
```

There are many more operations that Python's set can use.

Software Required: python39, 64 bit Fedora, pycharm, jupyter notebook

Expected Output

Create Set A
Enter number of Elements in set 3
Enter Element 1: 11
Enter Element 2: 12
Enter Element 3: 13

{ 11 , 12 , 13 }

```
|-----------------|
| 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 4
Enter Element 1: 11
Enter Element 2: 1

Enter Element 3: 12
Enter Element 4: 15
Set A =
{ 11 , 12 , 13 , 1 , 15 }

Set B =
{ 11 , 1 , 12 , 15 }

Union =
{ 11 , 12 , 13 , 1 , 15 }

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

Conclusion: This program gives us the knowledge of SET ADT

Questions asked in university exam.

1. What is Sets? Explain its operations..
2. What is Set as ADT?

DATA STRUCTURES AND ALGORITHMS – LABORATORY MANUAL

ASSIGNMENT NO.-03

Title:

ASSIGNMENT NO. – 03

TITLE: _____

 NAME OF THE STUDENT:_____

ROLL NO: _____

DATE OF PERFORMANCE:

DATE OF SUBMISSION:

SIGNATURE:

A book consists of chapters, chapters consist of sections and sections consist of subsections. Construct a tree and print the nodes. Find the time and space requirements of your method.

Objectives:
1. To understand concept of tree data structure
2. To understand concept & features of object oriented programming.

Learning Objectives:
- ✓ To understand concept of class
- ✓ To understand concept & features of object oriented programming.
- ✓ To understand concept of tree data structure.

Learning Outcome:
- Define class for structures using Object Oriented features.
- Analyze tree data structure.

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 graphicaly is represented most commonly as on Picture 1. The circles are the nodes and the edges are the links between them.

Trees are usualy used to store and represent data in some hierarhical 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 childs is called a leaf, and that node is of course at the bottommost 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 formaly, 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.
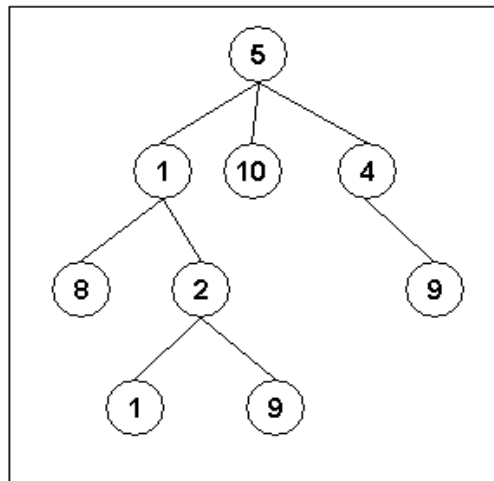
Fig. An example of a tree

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 minumum effort

For this assignment we are considering the tree as follows.
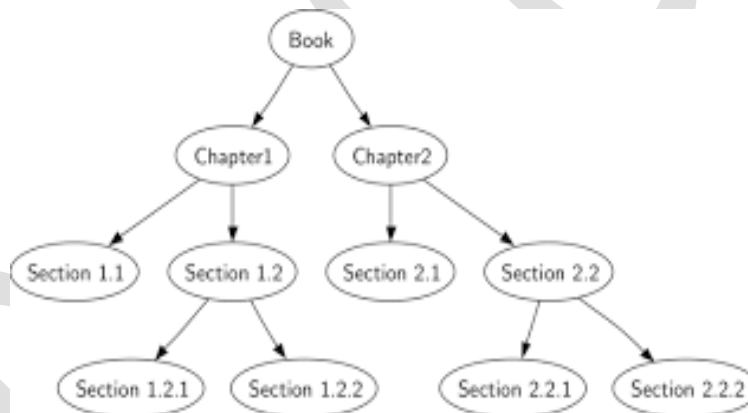
Software Required: g++ / gcc compiler- / 64 bit Fedora, eclipse IDE

Input: Book name & its number of sections and subsections along with name.

Output: Formation of tree structure for book and its sections.

Conclusion: This program gives us the knowledge tree data structure.

Questions asked in university exam.

1. What is class, object and data structure?
2. What is tree data structure?
3. Explain different types of tree?

CODE:

Function for inserting data in tree:

```
function insert()
{
 intchap,sec,subsec;
 char p;
 bnode *cnode,*snode,*par,*spar;
cout<<"Enter book NAme"<<endl;
cin>>p;
cnode = new bnode;
cnode->data=p;
cnode->right=NULL;
cnode->left=NULL;
root=cnode;
cout<<"Enter Number of Chapter?";
cin>>chap;
par=cnode;
for(inti=0;i<chap;i++)
{
   cout<<"enter name of chapter";
   cin>>p;
   cnode = new bnode;
   cnode->data=p;
   cnode->left=NULL;
   cnode->right=NULL;
   if(i==0)
   {
   par->left=cnode;
   }
   else
   {
   par->right=cnode;
   }
   spar=cnode;
   cout<<"enter How many section";
   cin>>sec;
   for(int j=0;j<sec;j++)
   {
   cout<<"enter name of section";
   cin>>p;
   snode = new bnode;
   snode->data=p;
   snode->left=NULL;
   snode->right=NULL;
```

```
if(j==0)
{
spar->left=snode;
}
else
{
spar->right=snode;
}
spar=snode;
}
par=cnode;
}
}
```

Function to display data in tree:

```
function displaytree()
{
structbnode *q[20];
structbnode *ptr,*bptr;
int front =-1,rear=-1;
front++;
rear++;
q[rear]=root;
q[++rear]=NULL;
while(front<=rear)
{
    ptr=q[front++];
    if(ptr==NULL)
       cout<<endl;
    if(ptr!=NULL)
    {
       cout<<" "<<ptr->data;
       do
       {
       if(ptr->left!=NULL)
       {
       q[++rear]=ptr->left;
       }
       if(ptr->right!=NULL)
       {
          ptr=ptr->right;
          cout<<" "<<ptr->data;
```

```
        if(ptr->left!=NULL &&ptr->right==NULL)
        {
        q[++rear]=ptr->left;
        }
    }
    }while(ptr->right!=NULL);
    q[++rear]=NULL;

  }
}

}
```

==============================OUTPUT============================
=
/* OUTPUT
[jspm@localhost ~]$ ./a.out

Menu
1.Create Tree
2.Display Tree
Enter Choice1
Enter book NAme
A
Enter Number of Chapter?3
enter name of chapterB
enter How many section2
enter name of sectionC
enter name of sectionD
enter name of chapterE
enter How many section2
enter name of sectionF
enter name of sectionG
enter name of chapterH
enter How many section3
enter name of sectionI
enter name of sectionJ
enter name of sectionK
do u want to continue?(1 for continue)1

Menu
1.Create Tree
2.Display Tree
Enter Choice2

A
B E H
C D F G I J K
do u want to continue?(1 for continue)0 */

<u>ASSIGNMENT NO. – 04</u>

TITLE: _____

NAME OF THE STUDENT:_____

ROLL NO: _____

DATE OF PERFORMANCE:

DATE OF SUBMISSION:

SIGNATURE:

ASSIGNMENT NO.-04

TITLE:
A Dictionary stores keywords & its meanings. Provide facility for adding new keywords, deleting keywords, & updating values of any entry. Also provide facility to display whole data sorted in ascending/ Descending order, Also find how many maximum comparisons may require for finding any keyword. Make use of appropriate data structures.

Objectives:
1. To understand concept of Tree data structure
2. To understand concept & features of object oriented programming.

Learning Objectives:
- ✓ To understand concept of Tree
- ✓ To understand concept of array, structure.

Learning Outcome:
☐ Implementation of sorting, searching on dictionary.
☐ Analyze working of functions

THEORY:
Binary search tree (BST), which may sometimes also be called an ordered or sorted binary tree, is a node-basedbinary treedata structure which has the following properties The left subtree of a node contains only nodes with keys less than the node's key. The right subtree of a node contains only nodes with keys greater than the node's key. Both the left and right subtrees must also be binary search trees. Generally, the information represented by each node is a record rather than a single data element. However, for sequencing purposes, nodes are compared according to their keys rather than any part of their associated records.
The major advantage of binary search trees over other data structures is that the related sorting algorithms and search algorithms such as in-order traversal can be very efficient. Binary search trees are a fundamental data structure used to construct more abstract data structures such as sets, multisets, and associative arrays.

ALGORITHMS:
Insertion: Insertion begins as a search would begin; if the root is not equal to the value, we search the left or right subtrees as before. Eventually, we will reach an external node and add the value as its right or left child, depending on the node's value. In other words, we examine the root and recursively insert the new node to the left subtree if the new value is less than the root, or the right subtree if the new value is greater than or equal to the root. Here's how a typical binary search tree insertion might be performed in C++:

```
/* Inserts the node pointed to by "newNode" into the subtree rooted at "treeNode" */
void InsertNode(Node* &treeNode, Node *newNode)
{
```

```
    if (treeNode == NULL)
treeNode = newNode;
    else if (newNode->key <treeNode->key)
InsertNode(treeNode->left, newNode);
    else
InsertNode(treeNode->right, newNode);
 }
```

Deletion: There are three possible cases to consider:
Deleting a leaf (node with no children): Deleting a leaf is easy, as we can simply remove it from the tree.
Deleting a node with one child: Remove the node and replace it with its child.
Deleting a node with two children: Call the node to be deleted N. Do not delete N. Instead, choose either its in-order successor node or its in-order predecessor node, R. Replace the value of N with the value of R, then delete R.
As with all binary trees, a node's in-order successor is the left-most child of its right sub tree, and a node's in-order predecessor is the right-most child of its left sub tree. In either case, this node will have zero or one children. Delete it according to one of the two simpler cases above.
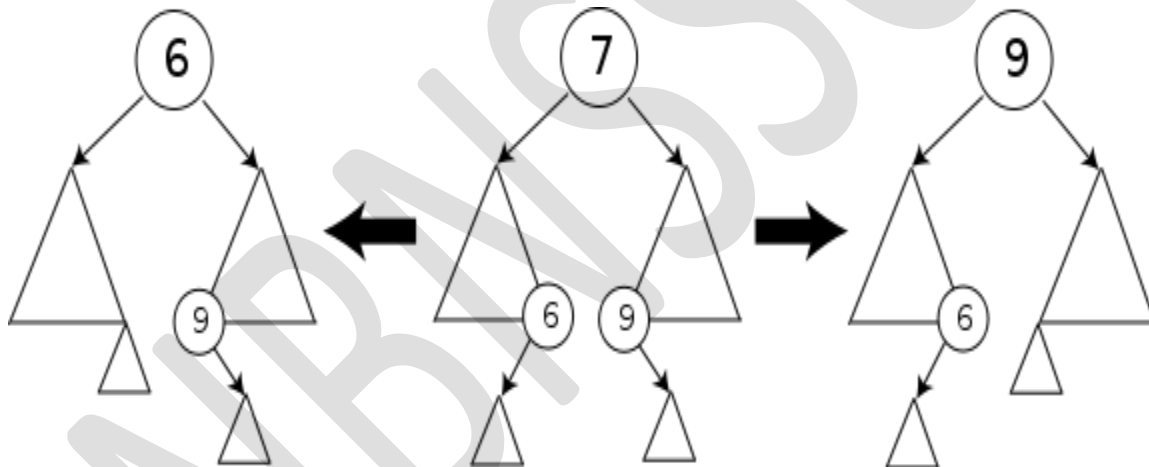
Deleting a node with two children from a binary search tree. The triangles represent sub trees of arbitrary size, each with its leftmost and rightmost child nodes at the bottom two vertices. Consistently using the in-order successor or the in-order predecessor for every instance of the two-child case can lead to an unbalanced tree, so good implementations add inconsistency to this selection.

STRCMP()
strcmp - compare two strings
Usage:
   include<string.h>
intstrcmp(const char *s1, const char *s2);
Description: The strcmp() function shall compare the string pointed to by s1 to the string pointed to by s2.
The sign of a non-zero return value shall be determined by the sign of the difference between the

values of the first pair of bytes (both interpreted as type **unsigned char**) that differ in the strings being compared.

RETURN VALUE
Upon completion, strcmp() shall return an integer greater than, equal to, or less than 0, if the string pointed to by s1 is greater than, equal to, or less than the string pointed to by s2, respectively

ALGORITHM:
Create Function:
Step 1: Start.
Step 2: Take data from user & create new node n1.
Step 3: Store data in n1 & make left & right child "NULL".
Step 4: If root is "NULL" then assign root to n1.
Step 5: Else call insert function & pass root & n1.
Step 6: Stop.

Insert Function:
Step 1: Start.
Step 2: We have new node in passed argument; so check data in temp
is less than data in root. If yes then check if Left child of root is
        "NULL" or not. If yes, store temp in left child otherwise call
insert function again & pass left address of node & temp.
Step 3: If data in root is less than data in temp then check if right child
of root is "NULL". If yes then store temp in right child field of
        root & if not call insert function again & pass right child address
of node & temp.
Step 4: Stop.

Display Function:
Step 1: Start.
Step 2: If root is "NULL" then display tree is not created.
Step 3: Otherwise call inorder function & pass root.
Step 4: Stop.

Inorder Function:
Step 1: Start.
Step 2: Taking root from inorder function, check if it is "NULL" or not.
Step 3: Again call inorder function & pass left child address of node
Step 4: Display data of root.
Step 5: Again call inorder function & pass right child address of node.
Step 6: Stop.

Search Function:
Step 1: Start.
Step 2: If root is "NULL" then display tree is not created.
Step 3: Read the key to be searched in a variable, say k.
Step 4: Search for the key in the root, left and right side of the root.
Step 5: Display appropriate message for successful and unsuccessful search.
Step 6: Stop

Update Function:
Step 1: Start.
Step 2: If root is "NULL" then display tree is not created.
Step 3: Read the key to be Updated in a variable, say k.
Step 4: Search for the key in the root, left and right side of the root.
Step 5: If key is not present, display appropriate message.
Step 6: If key is present, update its meaning.
Step 7: Stop

Delete Function:
Step 1: Start.
Step 2: If root is "NULL" then display tree is not created.
Step 3: Read the key to be Deleted in a variable, say k.
Step 4: Search for the key in the root, left and right side of the root.
Step 5: If key is not present, display appropriate message.
Step 6: If key is present, Delete the key.
Step 7: Stop

Main Function:
Step 1: Start.
Step 2: Create necessary class objects & declare necessary variables.
Step 3: Print menu as below & take choice from user:
Create
Display
Search
Update
Delete
Exit
Step 4: If choice is 1, call create function.
Step 5: If choice is 2, call display function.
Step 6: If choice is 3, call Search function.
Step 7: If choice is 4, call Update function.
Step 8: If choice is 5, call Delete function.
Step 9: If choice is 6, then exit.
Step 10: Ask user whether he wants to continue or not.
Step 11: If yes then go to step 3.

Step 12: Stop.

Mathematical Modeling:

Let I be a set of Inputs $\{n, i_n\}$ where n is no. of keywords of dictionary & i_n is an individual node values which stores keyword & its meaning.

Let O be a set of Outputs $\{C, Dk, N, D, S, U\}$ where

C is Create Dictionary function $= c(I) = L$ (c is function of create which gives output as a L set of keywords & its meaning of dictionary)

Dk is display keyword function $= dk(L) = i_n$ (dk is display function on L which gives output as i_n an individual node values stores keyword & its meaning)

N is Insert New keyword function $= n(L) = i_{n+1}$ (n is insert new keyword function on L which gives output as i_{n+1} an individual node values (stores keyword & its meaning) + 1)

D is Delete keyword function $= d(L) = i_{n-1}$ (d is delete keyword function on L which gives output as i_{n-1} an individual node values (stores keyword & its meaning) - 1)

S is Search keyword Function $= s(L) = a$ (s is search function on L which gives output as a where $a \in A = \{$Keyword Found, Keyword Not Found$\}$)

U is Update keyword Function $= u(L) = a$ (u is update function on L which gives output as a i_n an individual node updates meaning of keyword.

Software Required: g++ / gcc compiler- / 64 bit Fedora, eclipse IDE

INPUT: Enter data (numbers) to be stored in the binary search tree. Every node in the BST would contain 3 fields: data, left child pointer and right child pointer.

OUTPUT: Element should be inserted/deleted at the proper places in the BST.

Questions asked in university exam:

  1) Explain strcmp() with its return values?
  2) What is Binary Search Tree? Explain with an example?
  3) Explain various operations on BST?
  4) What is inorder traversal of BST?

CODE:

//class structure for the program

```
class dictionary
{
       int size;
       item keys[50];
```

```
public:
        dictionary(){size=0;}
        void create();
        int search(item x);
        void delet(item x);
        void insert(item x);
        void modify(item x);
        void sort();
        void display();
void dictionary::create()
{
        inti;
        cout<<"\n Enter no of keywords:";
        cin>>size;
        for(i=0;i<size;i++)
                keys[i].read();
}
```

//function to search the data

```
int dictionary::search(item x)
{
        inti;
        for(i=0;i<size;i++)
                if(keys[i].compare(x))
                        return i;
        return -1;
}
```

//function to delete the data

```
void dictionary::delet(item x)
{
        inti;
        intloc=search(x);
        if(loc==-1)
                cout<<"\n item not found";
        else
        {
                for(i=loc+1;i<size;i++)
                        keys[i-1]=keys[i];
                size--;
        }
}
```

//function to add the data

```
void dictionary::insert(item x)
{
        keys[size++]=x;
}
void dictionary::modify(item x)
{
        intloc=search(x);
        if(loc==-1)
                cout<<"\n item not found";
        else
        {
                cout<<"\n Enter new value:";
                keys[loc].read();
        }
}
```

// function to sort the data

```
void dictionary::sort()
{
        inti,j,n;
        item temp;
        n=size;
        for(i=0;i<n;i++)
                for(j=0;j<n-1;j++)
                        if(keys[j].larger(keys[j+1]))
                        {
                                temp=keys[j];
                                keys[j]=keys[j+1];
                                keys[j+1]=temp;
                        }
}
```
//function to display the contents

```
void dictionary::display()
{
        inti;
        cout<<"\n display asending:\n";
        for(i=0;i<size;i++)
                keys[i].display();
        cout<<"\n display desending:";
        for(i=size-1;i>=0;i--)
                keys[i].display();
}
```

OUTPUT

```
1)create
2)insert
3)delete
4)modify
5)searching
6)display asending/desending
7)quit
 Enter your choice:1

 Enter no of keywords:2

 Enter a keywords:apple

 Enter its meaning:fruits

 Enter a keywords:cricket

 Enter its meaning:game


1)create
2)insert
3)delete
4)modify
5)searching
6)display asending/desending
7)quit
 Enter your choice:2

 Enter a keywords:mango

 Enter its meaning:king


1)create
2)insert
3)delete
4)modify
5)searching
6)display asending/desending
7)quit
 Enter your choice:6

 display asending:
apple----------->>>>fruits
```

```
cricket----------->>>>game
mango----------->>>>king

 display desending:
mango----------->>>>king
cricket----------->>>>game
apple----------->>>>fruits


1)create
2)insert
3)delete
4)modify
5)searching
6)display asending/desending
7)quit
 Enter your choice:4

 Enter a key:mango

 Enter new value:

 Enter a keywords:maharashtra

 Enter its meaning:state


1)create
2)insert
3)delete
4)modify
5)searching
6)display asending/desending
7)quit
 Enter your choice:5

 Enter a key:cricket

cricket----------->>>>game


1)create
2)insert
3)delete
4)modify
5)searching
```

6)display asending/desending
7)quit
Enter your choice:3

Enter a key:apple

1)create
2)insert
3)delete
4)modify
5)searching
6)display asending/desending
7)quit
Enter your choice:7

ASSIGNMENT NO. – 05

TITLE: _____

NAME OF THE STUDENT:_____

ROLL NO: _____

DATE OF PERFORMANCE:

DATE OF SUBMISSION:

SIGNATURE:

ASSIGNMENT NO.-05

TITLE:
Convert given binary tree into threaded binary tree. Analyze time and space complexity of the algorithm.

OBJECTIVES:

1. To write simple program in C++.

2. To understand concept & features C++ programming.

3. To implement algorithms in C++.

Learning Objectives

- ✓ To understand concept of TBT algorithm.
- ✓ To implement concept of TBT algorithm.

Learning Outcome
After successfully completing this assignment, students will be able to
- ✓ Learn C++ Programming features
- ✓ Understand & implement TBT algorithm using C++**.**

RELEVANT THEORY:
THREADED BINARY SEARCH TREE:

Threaded Binary Search Tree
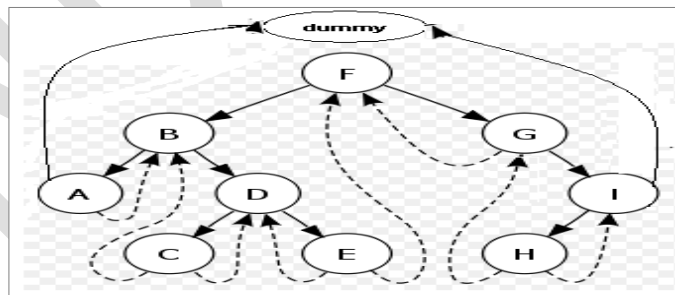
A threaded binary tree makes it possible to traverse the values in the binary tree via a linear traversal that is more rapid than a recursive in-order traversal.
It is also possible to discover the parent of a node from a threaded binary tree, without explicit use of parent pointers or a stack

Tree Traversals of Threaded Binary Tree:
 Inorder Traversals: A B C D E F F H I

Preorder Traversals: F   B   A   D   C   E   G   I   H
Postorder Traversals: A   C   E   D   B   H   I   G F

ALGORITHM:

Create Function:

Step 1: Start.
Step 2: Take data from user & create new node n1.
Step 3: Store data in n1 & make left & right child "NULL".
Step 4: If root is "NULL" then root is equal to n1. Create a dummy or head node. Head of left thread=1, right thread = 0,right = NULL, left field will store address of root; assign root of left field points to dummy & right field to dummy.
Step 5: Else call insert function & pass root & n1.
Step 6: Stop.

Insert Function:

Step 1: Start.
Step 2: We have new node in passed argument; so check data in temp is less than data in root. If yes then check if Left thread of root is or not. If yes, store temp in left child, store left thread equal to 1, temp of left field equal to root of left field. Temp of right field equal to root. otherwise call insert function again & pass left address of node & temp.
Step 3: If data in root is less than data in temp then check if right thread of root is 1 or not. If yes, store temp in right child, store right thread equal to 1,temp of right field equal to root of right field. Temp of left field equal to root. otherwise call insert function again & pass right address of node & temp.
Step 4: Stop.

Preorder Function:

Step 1: Start.
Step 2: If root is "NULL" then display tree is not created.
Step 3: Otherwise store root in temp.
Step 4: Display data of temp.
Step 5: if left thread of temp is equal to 1 then go to left of temp. repeat same step 4.
Step 6: if right thread is equal to 0 then then go to left of temp.
if temp is equal to dummy then go to step 9.otherwise repeat step 6.
Step 7: go to right of temp.
Step 8: if temp is equal to dummy then go to Step 9.
    otherwise repeat step 4.
Step 9: Stop.

Postorder Function:

Step 1: Start.

Step 2: If root is "NULL" then display tree is not created.

Step 3: Otherwise store root in temp.

Step 4: if left thread of temp is equal to 1 then go to left of temp. repeat same step 4.

Step 5: if right thread is equal to 0 then then go to left of temp.

if temp is equal to dummy then go to step 9.otherwise repeat step 6.

Step 6: go to right of temp.

Step 7: Display data of temp.

Step 8: if temp is equal to dummy then go to Step 9.
    otherwise repeat step 4.

Step 9: Stop.

Inorder Function:

Step 1: Start.

Step 2: If root is "NULL" then display tree is not created.

Step 3: Otherwise store root in temp.

Step 4: if left thread of temp is equal to 1 then go to left of temp. repeat same step 4.

Step 5: Display data of temp.

Step 6: if right thread is equal to 0 then then go to left of temp.

if temp is equal to dummy then go to step 9. Display data of temp. Repeat step 6.

Step 7: go to right of temp.

Step 8: if temp is equal to dummy then go to Step 9.

otherwise repeat step 4.

Step 9: Stop.

Main Function:

Step 1: Start.

Step 2: Declare necessary variables.

Step 3: Print menu as follow & take choice from user:
    1. Create threaded binary search tree
    2. Display preorder
    3. Display inorder
    4. Display postorder
    5. Exit

Step 4: If choice is 1, call create function.

Step 5: If choice is 2, call preorder function.

Step 6: If choice is 3, call inorder function.

Step 7: If choice is 4, call postorder function.

Step 8: If choice is 4, exit from function.

Step 9: Stop.

Mathematical Modelling:

Let I be a set of Inputs {n, i_n } where n is number of nodes & i_n is an individual node values which stores integer data.

Let O be a set of Outputs {C, Din, Dpre, Dpost } where
C is Create function = c(I) = L (c is function of create which gives output as a L set of nodes)
Din is display inorder function = in(L) = i_n (in is display function on L which gives output as inorder traversal of i_n an individual node values)
Dpre is display preorder function = pre(L) = i_n (pre is display function on L which gives output as preorder traversal of i_n an individual node values)
Dpost is display postorder function = post(L) = i_n (in is display function on L which gives output as postorder traversal of i_n an individual node values)

SOFTWARE REQUIRED: CPP compiler- / Fedora PC

Input:
Give integer numbers as data. And Read user choice.

Output:
    Conversion of binary tree into threaded binary tree.

Conclusion: Performed the operations on TBT.

OUTCOME

Upon completion Students will be able to:

ELO1: Learn Object oriented Programming features.
ELO2: Understand & implement TBT.

Questions asked in university exam:

1) What is Threaded Binary Tree?
2) What is difference in BT & TBT?
3) What is Advantages of TBT Tree?
4) What is use of thread in TBT?

ASSIGNMENT NO. – 06

TITLE: _____

NAME OF THE STUDENT:_____

ROLL NO: _____

DATE OF PERFORMANCE:

DATE OF SUBMISSION:

SIGNATURE:

ASSIGNMENT NO.-06

Title:
There are flight paths between cities. If there is a flight between city A and city B then there is an edge between the cities. The cost of the edge can be the time that flight take to reach city B from A, or the amount of fuel used for the journey. Represent this as a graph. The node can be represented by airport name or name of the city. Use adjacency list representation of the graph or use adjacency matrix representation of the graph. Justify the storage representation used.

Objectives:

1. To understand concept of Graph data structure
2. To understand concept of representation of graph.

Learning Objectives:
- ✓ To understand concept of Graph data structure
- ✓ To understand concept of representation of graph.

Learning Outcome:

- Define class for graph using Object Oriented features.
- Analyze working of functions.

Theory:
Graphs are the most general data structure. They are also commonly used data structures.

Graph definitions:

- A non-linear data structure consisting of nodes and links between nodes.

Undirected graph definition:

- An undirected graph is a set of nodes and a set of links between the nodes.
- Each node is called a **vertex**, each link is called an **edge**, and each edge connects two vertices.
- The order of the two connected vertices is unimportant.
- An undirected graph is a finite set of vertices together with a finite set of edges. Both sets might be empty, which is called the empty graph.
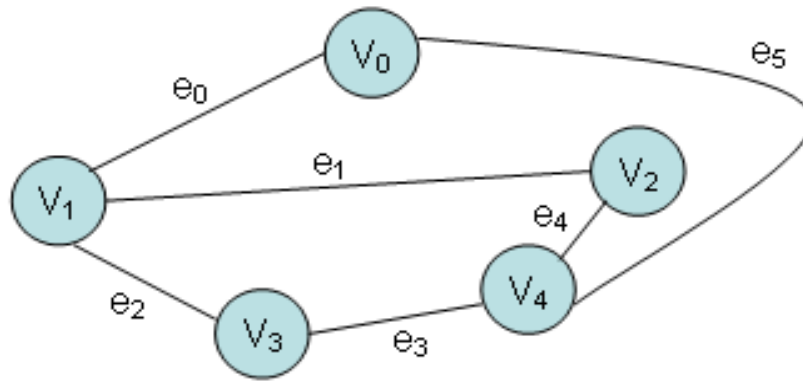
Graph Implementation:

Different kinds of graphs require different kinds of implementations, but the fundamental concepts of all graph implementations are similar. We'll look at several representations for one particular kind of graph: directed graphs in which loops are allowed.

Representing Graphs with an Adjacency Matrix
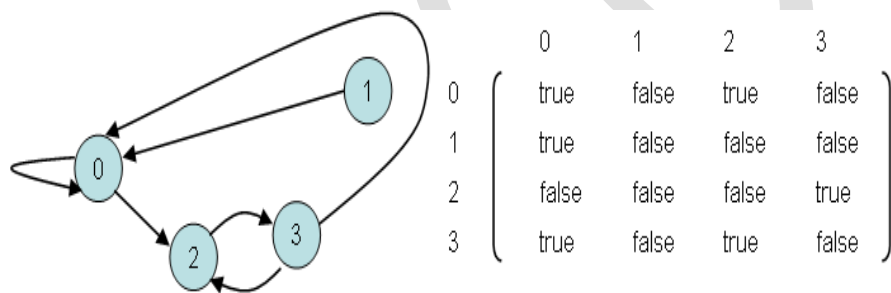
Fig: Graph and adjacency matrix

Definition:

- An adjacency matrix is a square grid of true/false values that represent the edges of a graph.
- If the graph contains n vertices, then the grid contains n rows and n columns.
- For two vertex numbers i and j, the component at row i and column j is true if there is an edge from vertex i to vertex j; otherwise, the component is false.

We can use a two-dimensional array to store an adjacency matrix:

boolean[][] adjacent = new boolean[4][4];

Once the adjacency matrix has been set, an application can examine locations of the matrix to determine which edges are present and which are missing.

Representing Graphs with Edge Lists
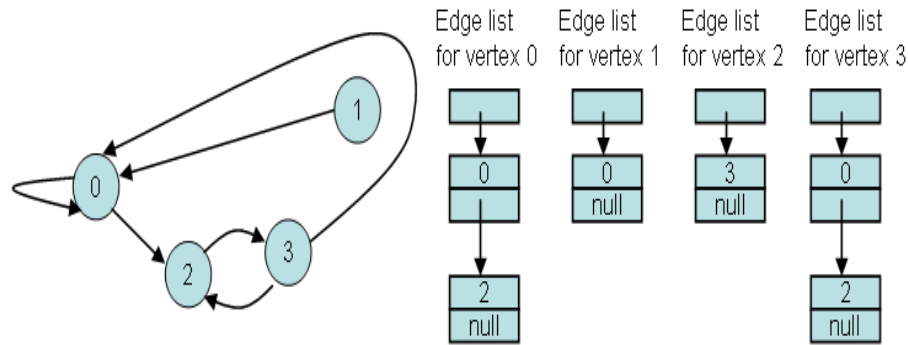
Fig: Graph and adjacency list for each node

Definition:

- A directed graph with n vertices can be represented by n different linked lists.
- List number i provides the connections for vertex i.
- For each entry j in list number i, there is an edge from i to j.

Loops and multiple edges could be allowed.

Representing Graphs with Edge Sets

To represent a graph with n vertices, we can declare an array of n sets of integers. For example:

IntSet[] connections = new IntSet[10]; // 10 vertices

A set such as connections[i] contains the vertex numbers of all the vertices to which vertex i is connected.

Software Required: g++ / gcc compiler- / 64 bit Fedora, eclipse IDE

Input: 1.Number of cities.
2.Time required to travel from one city to another.

Output: Create Adjacency matrix to represent path between various cities.

Conclusion: This program gives us the knowledge of adjacency matrix graph.

Questions asked in university exam.

1. What are different ways to represent the graph? Give suitable example.

2. What is time complexity of function to create adjacency matrix?

Expected Output

Menu
1.Create Graph using adjacency List
2.Display Graph
Enter Choice1
No of Cities ?4
No of Flights?3

please enter source city and destination citystarting from A upto number of cities like A,B,C,D
Edge no -> 1
Source city->A
Destination city-> B
cost->23

Edge no -> 2
Source city->B
Destination city-> C
cost->35

Edge no -> 3
Source city->C
Destination city-> D
cost->56

do u want to continue?(1 for continue)1

Menu
1.Create Graph using adjacency List
2.Display Graph
Enter Choice2

A--> B & cost23
|
B--> C & cost35
|
C--> D & cost56
|
D
do u want to continue?(1 for continue)0
*/

ASSIGNMENT NO. – 07

TITLE: _____

 NAME OF THE STUDENT:_____

ROLL NO: _____

DATE OF PERFORMANCE:

DATE OF SUBMISSION:

SIGNATURE:

ASSIGNMENT NO.-07

TITLE:
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.

Objectives:
1. To understand concept of graph data structure
2. To understand concept & features of object oriented programming.

Learning Objectives:
- ✓ To understand concept of class
- ✓ To understand concept & features of object oriented programming.
- ✓ To understand concept graph data structures

Learning Outcome:
- ✓ learn the working of minimum spanning tree
- ✓ Analyze working of functions

THEORY:

Spanning Tree: A spanning tree for a connected graph G is a tree T containing all the vertices of G.

.
     G   **T**
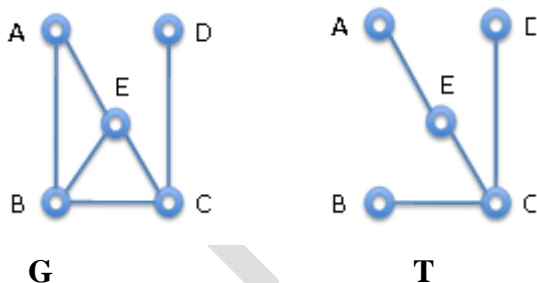
Minimum Spanning Tree: A spanning tree whose weight of edges is minimum is called as Minimum Spanning Tree.
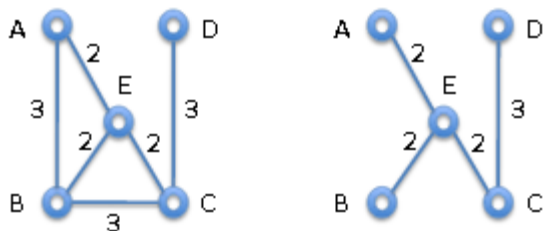
Two Methods to find Minimum Spanning Tree:

Prims Algorithm:

This algorithm is used to find minimum spanning tree and minimum cost of spanning tree.

Steps:: 1) This algorithm starts with start vertex.
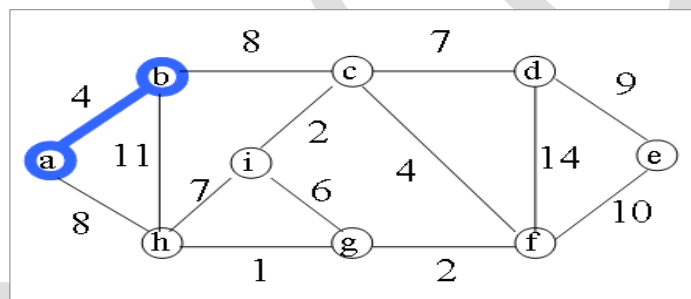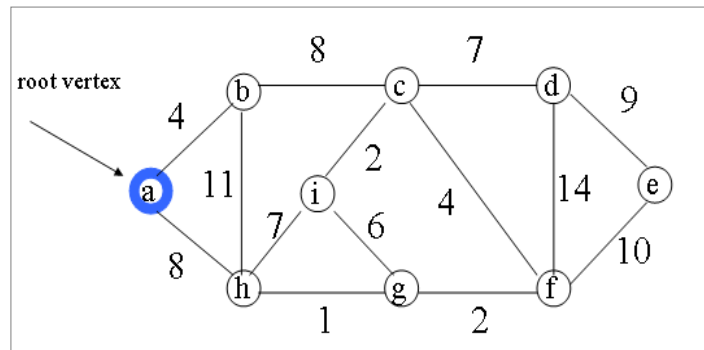
2) Then it finds edge which has minimum weight.
3) It continues the finding such that one vertex of edge is already visited & another vertex is not visited with minimum weight.
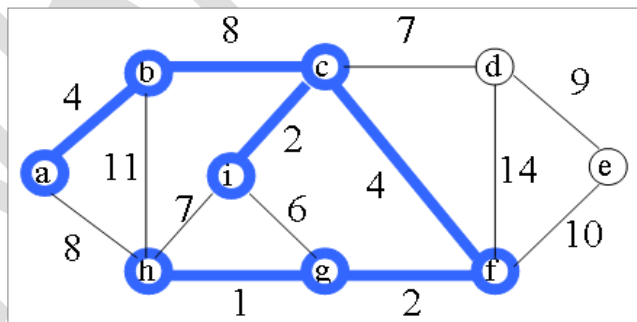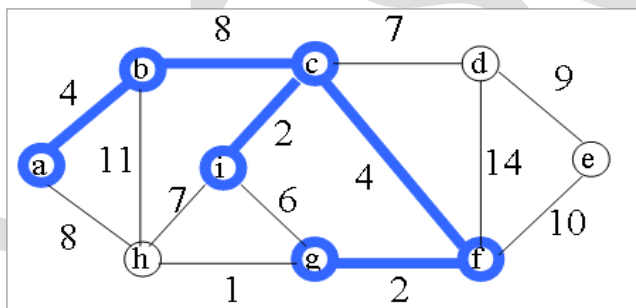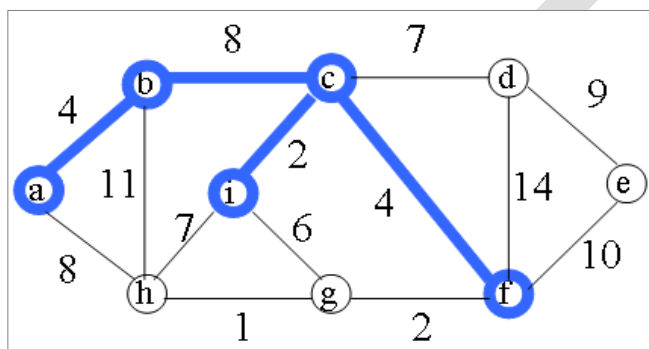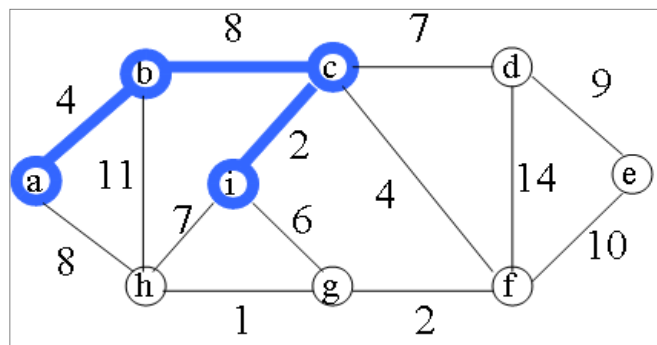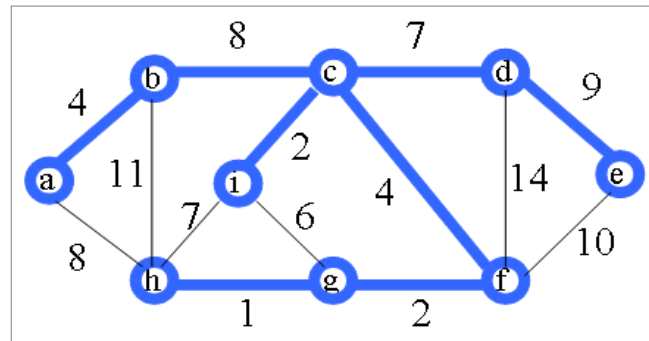
4) These steps are repeated to cover all the vertices of graph.
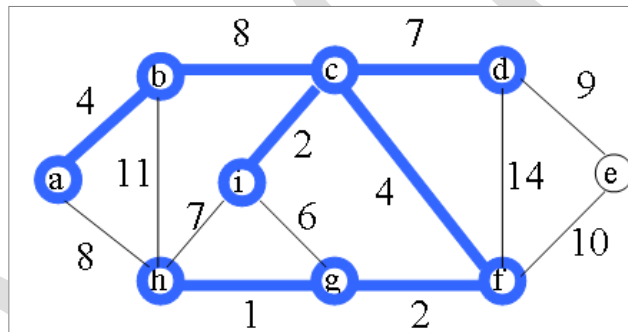
2. Kruskal's Algorithm
This algorithm is used to find minimum spanning tree and minimum cost of spanning tree.

Steps::   1) This algorithm starts with edge with minimum weight. AD and CE are the shortest edges, with length 5, and AD has been arbitrarily chosen, so it is highlighted.
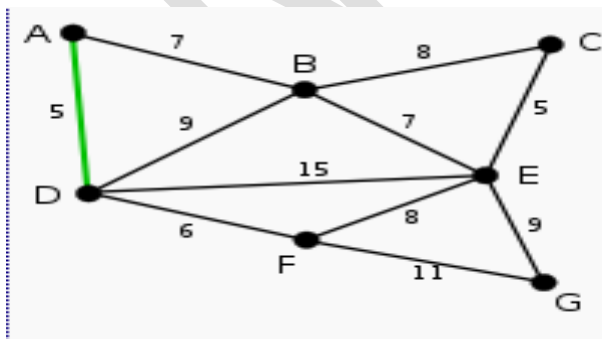
2) CE is now the shortest edge that does not form a cycle, with length 5, so it is highlighted as the second edge.
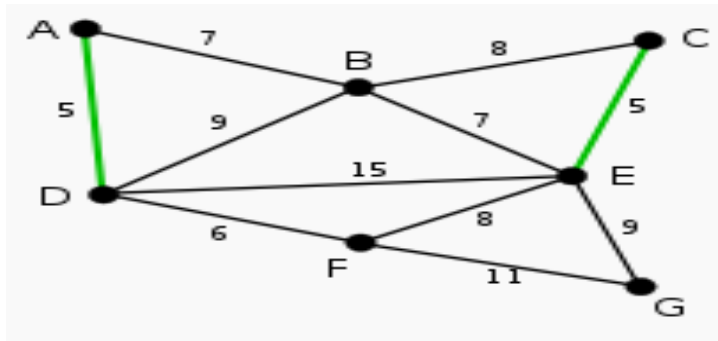
3) The next edge, DF with length 6, is highlighted using much the same method.
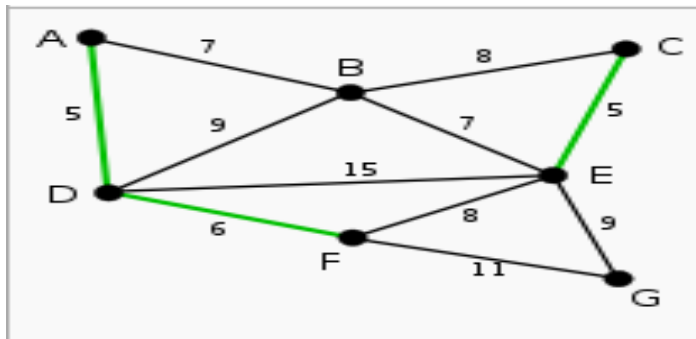
4) The next-shortest edges are AB and BE, both with length 7. AB is chosen arbitrarily, and is highlighted. The edge BD has been highlighted in red, because there already exists a path (in green) between B and D, so it would form a cycle (ABD) if it were chosen.
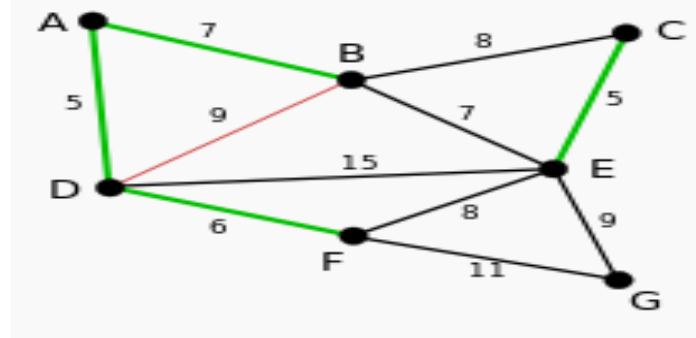
5) The process continues to highlight the next-smallest edge, BE with length 7. Many more edges are highlighted in red at this stage: BC because it would form the loop BCE, DE because it would form the loop DEBA, and FE because it would form FEBAD.
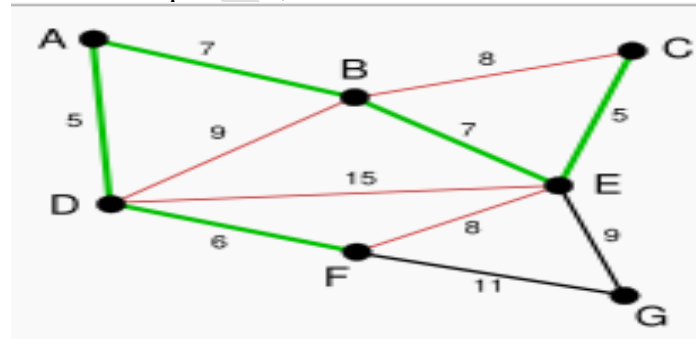
6) Finally, the process finishes with the edge **EG** of length 9, and the minimum spanning tree is

found.
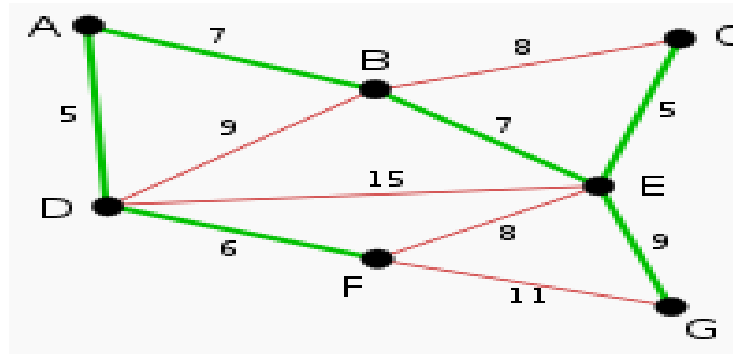
Application of Minimum Spanning Tree:
Minimum spanning trees are useful in constructing networks, by describing the way to connect a set of sites using the smallest total amount of wire.

SOFTWARE REQUIRED: C compiler- / Fedora PC

INPUT: number of vertices as offices and provide edge between vertices/offices as telephone lease lines(here leased line are nothing but edges and offices are nothing but vertices).

Output: The path which have minimum distance.

Conclusion: Prims Algorithms is successfully implemented to find out minimum distance.

Questions asked in university exam:

1. What is Spanning Tree?
2. What is Minimum Spanning Tree?
3. What is Prim's Algorithm?
4. What is Kruskal's Algorithm?
5. What are Applications of Minimum Spanning Tree?
6. How to decide whether to select Kruskal's algorithm or Prim's algorithm?

Expected Output

Menu
1.Create Graph for offices
2.Display Graph
3.min cost cities
Enter Choice1
No of offices ?4
No of leased line ?4

please enter source city and destination city of the edge(city no must be between 1 to4)
Edge no -> 1
Source city->1
Destination city-> 2
Cost of Conecting 2 city2

Edge no -> 2
Source city->2
Destination city-> 4
Cost of Conecting 2 city3

Edge no -> 3
Source city->3
Destination city-> 4
Cost of Conecting 2 city5

Edge no -> 4
Source city->1
Destination city-> 3
Cost of Conecting 2 city4
do u want to continue?(1 for continue)1

Menu
1.Create Graph for offices
2.Display Graph
3.min cost cities
Enter Choice2

0 2 4 0
2 0 0 3
4 0 0 5
0 3 5 0 do u want to continue?(1 for continue)1

Menu
1.Create Graph for offices
2.Display Graph
3.min cost cities
Enter Choice3

set of lines for connecting city with min. cost is
city1->city2
city2->city4
city1->city3
do u want to continue?(1 for continue)0

ASSIGNMENT NO. – 08

TITLE: _____

NAME OF THE STUDENT:_____

ROLL NO: _____

DATE OF PERFORMANCE:

DATE OF SUBMISSION:

SIGNATURE:

ASSIGNMENT NO.-08

Title:

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?

Objectives:

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

Learning Objectives:
- ✓ To understand concept of OBST.
- ✓ To understand concept & features like extended binary search tree.

Learning Outcome:

- ✓ Define class for Extended binary search tree using Object Oriented features.
- ✓ Analyze working of functions.

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:
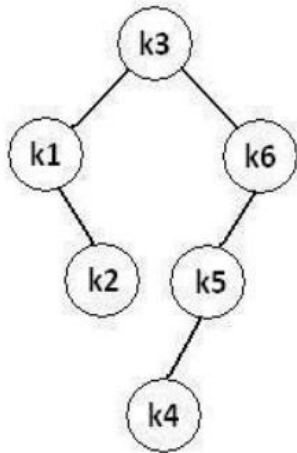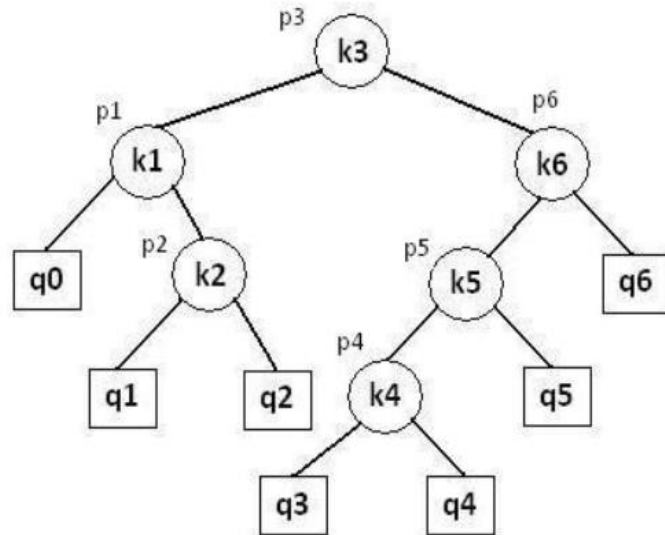
Binary search tree   Extended binary search tree

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.

ALGORITHMS

We have the following procedure for determining R(i, j) and C(i, j) with $0 <= i <= j <= n$:

PROCEDURE COMPUTE_ROOT(n, p, q; R, C)

begin

fori = 0 to n do

C (i, i) ←0

W (i, i) ←q(i)

for m = 0 to n do

fori = 0 to (n – m) do

j ←i + m

W (i, j) ←W (i, j – 1) + p (j) + q (j)

*find C (i, j) and R (i, j) which minimize the

tree cost

end

The following function builds an optimal binary sea

rch tree

FUNCTION CONSTRUCT(R, i, j)

begin

*build a new internal node N labeled (i, j)

k ←R (i, j)
fi = k then

*build a new leaf node N' labeled (i, i)

else

*N' ←CONSTRUCT(R, i, k)

*N' is the left child of node N

if k = (j – 1) then

*build a new leaf node N'' labeled (j, j)

else

*N'' ←CONSTRUCT(R, k + 1, j)

*N'' is the right child of node N

return N

end

COMPLEXITY ANALYSIS:

The algorithm requires O (n2) time and O (n2) storage. Therefore, as 'n' increases it will run out of storage even before it runs out of time. The storage needed can be reduced by almost half by implementing the two-dimensional arrays as one-dimensional arrays.

Software Required: g++ / gcc compiler- / 64 bit Fedora, eclipse IDE

Input: 1. No.of Element.
   2. key values
   3. Key Probability

Output: Create binary search tree having optimal searching cost.

Conclusion: This program gives us the knowledge OBST, Extended binary search tree.

Questions asked in university exam.

1. What is Binary Search Tree? Explain with an example?

2. Explain various operations on BST?

3. What is inorder traversal of BST?

4. What is a B tree?

5. What are threaded binary trees?

6. What is a B+ tree?

7. How do you find the depth of a binary tree?

8. Explain pre-order and in-order tree traversal.

9. Define threaded binary tree. Explain its common uses

10. Explain various operations on BST?

<div style="border:2px solid maroon; padding:20px;">

<u>ASSIGNMENT NO. – 09</u>

TITLE: _____

NAME OF THE STUDENT:_____

ROLL NO: _____

DATE OF PERFORMANCE:

DATE OF SUBMISSION:

SIGNATURE:

</div>

ASSIGNMENT NO.-09

Title:

A Dictionary stores keywords & its meanings. Provide facility for adding new keywords, deleting keywords, updating values of any entry. Provide facility to display whole data sorted in ascending/ Descending order. Also find how many maximum comparisons may require for finding any keyword. Use Height balance tree and find the complexity for finding a keyword.

Objectives:

1. To understand concept of height balanced tree data structure.
2. To understand procedure to create height balanced tree.

Learning Objectives:
- ✓ To understand concept of height balanced tree data structure.
- ✓ To understand procedure to create height balanced tree.

Learning Outcome:

- Define class for AVL using Object Oriented features.
- Analyze working of various operations on AVL Tree .

Theory:

An empty tree is height balanced tree if T is a nonempty binary tree with TL and TR as its left and right sub trees. The T is height balance if and only if Its balance factor is 0, 1, -1.

AVL (Adelson- Velskii and Landis) Tree: A balance binary search tree. The best search time, that is O (log N) search times. An AVL tree is defined to be a well-balanced binary search tree in which each of its nodes has the AVL property. The AVL property is that the heights of the left and right sub-trees of a node are either equal or if they differ only by 1.

What if the input to binary search tree comes in a sorted (ascending or descending) manner? It will then look like this −
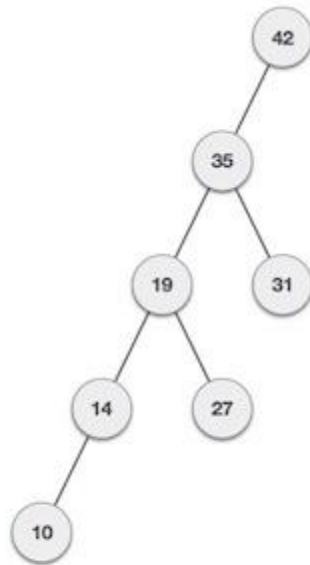
If input 'appears' non-increasing manner   If input 'appears' in non-decreasing manner
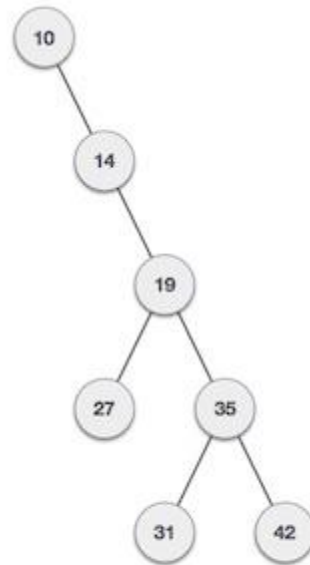
It is observed that BST's worst-case performance is closest to linear search algorithms, that is O(n). In real-time data, we cannot predict data pattern and their frequencies. So, a need arises to balance out the existing BST.

Named after their inventor **Adelson**, **Velski**&**Landis**, **AVL trees** are height balancing binary search tree. AVL tree checks the height of the left and the right sub-trees and assures that the difference is not more than 1. This difference is called the **Balance Factor**.

Here we see that the first tree is balanced and the next two trees are not balanced −
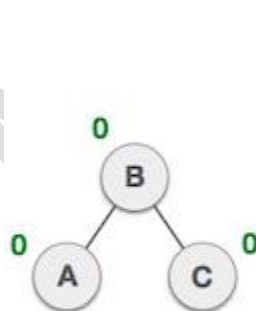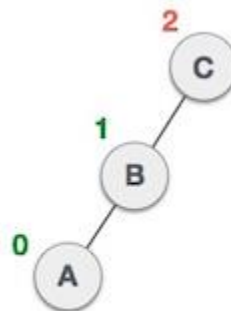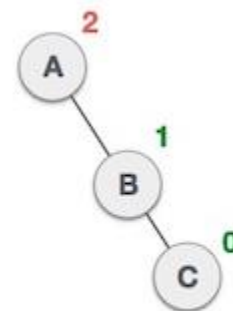
Balanced   Not balanced   Not balanced

In the second tree, the left subtree of **C** has height 2 and the right subtree has height 0, so the difference is 2. In the third tree, the right subtree of **A** has height 2 and the left is missing, so it is 0, and the difference is 2 again. AVL tree permits difference (balance factor) to be only 1.

BalanceFactor = height(left-sutree) − height(right-sutree)

If the difference in the height of left and right sub-trees is more than 1, the tree is balanced using some rotation techniques.

AVL Rotations

To balance itself, an AVL tree may perform the following four kinds of rotations −

- Left rotation
- Right rotation
- Left-Right rotation
- Right-Left rotation

The first two rotations are single rotations and the next two rotations are double rotations. To have an unbalanced tree, we at least need a tree of height 2. With this simple tree, let's understand them one by one.

Left Rotation

If a tree becomes unbalanced, when a node is inserted into the right subtree of the right subtree, then we perform a single left rotation −

Right unbalanced tree   Left Rotation   Balanced
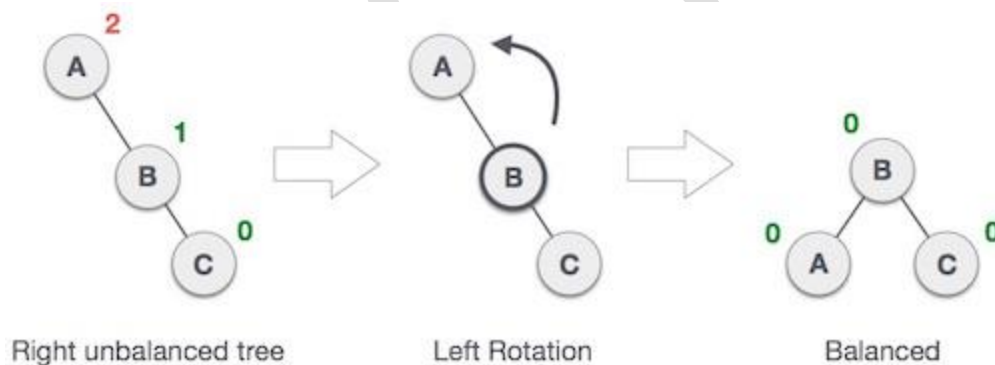
In our example, node **A** has become unbalanced as a node is inserted in the right subtree of A's right subtree. We perform the left rotation by making **A** the left-subtree of B.

Right Rotation

AVL tree may become unbalanced, if a node is inserted in the left subtree of the left subtree. The tree then needs a right rotation.
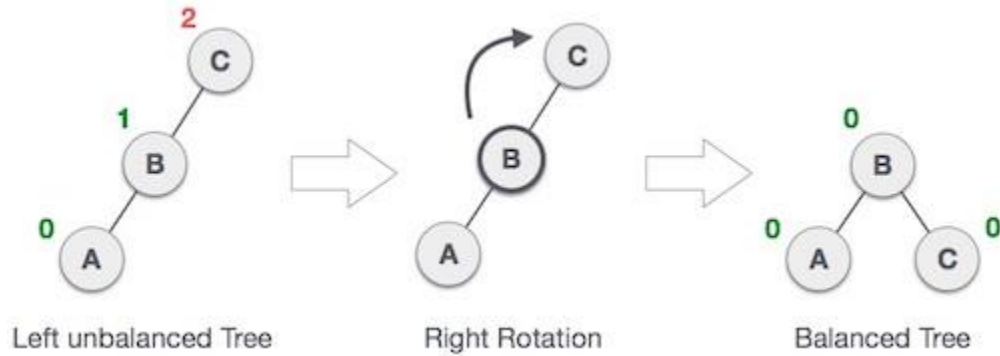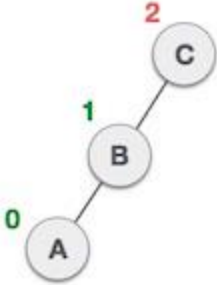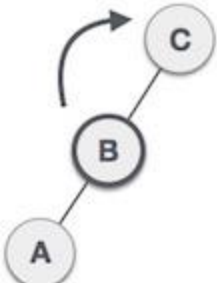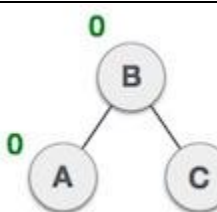
As depicted, the unbalanced node becomes the right child of its left child by performing a right rotation.
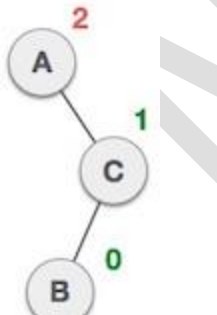
Left-Right Rotation

Double rotations are slightly complex version of already explained versions of rotations. To understand them better, we should take note of each action performed while rotation. Let's first check how to perform Left-Right rotation. A left-right rotation is a combination of left rotation followed by right rotation.
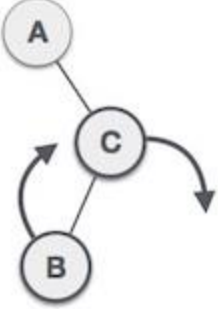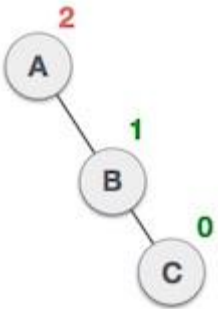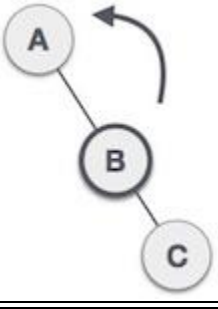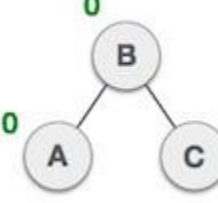
State	Action
	A node has been inserted into the right subtree of the left subtree. This makes **C** an unbalanced node. These scenarios cause AVL tree to perform left-right rotation.
	We first perform the left rotation on the left subtree of **C**. This makes **A**, the left subtree of **B**.

	Node **C** is still unbalanced, however now, it is because of the left-subtree of the left-subtree.
	We shall now right-rotate the tree, making **B** the new root node of this subtree. **C** now becomes the right subtree of its own left subtree.
	The tree is now balanced.

Right-Left Rotation

The second type of double rotation is Right-Left Rotation. It is a combination of right rotation followed by left rotation.

State	Action
	A node has been inserted into the left subtree of the right subtree. This makes **A**, an unbalanced node with balance factor 2.

	First, we perform the right rotation along **C** node, making **C** the right subtree of its own left subtree**B**. Now, **B** becomes the right subtree of **A**.
	Node **A** is still unbalanced because of the right subtree of its right subtree and requires a left rotation.
	A left rotation is performed by making **B** the new root node of the subtree. **A** becomes the left subtree of its right subtree**B**.
	The tree is now balanced.

Algorithm AVL TREE:
Insert:-
1. If P is NULL, then
      I. P = new node
      II. P ->element = x
      III. P ->left = NULL
      IV. P ->right = NULL
      V. P ->height = 0
2. else if x>1 => x<P ->element
      a.) insert(x, P ->left)
      b.) if height of P->left -height of P ->right =2

DEPARTMENT OF COMPUTER ENGINEERING   NBNSSOE, PUNE-41

        1. insert(x, P ->left)

        2. if height(P ->left) -height(P ->right) =2

            if x<P ->left ->element

                P =singlerotateleft(P)

            else

                P =doublerotateleft(P)

3. else

    if x<P ->element

        a.) insert(x, P -> right)

        b.) if height (P -> right) -height (P ->left) =2

            if(x<P ->right) ->element

                P =singlerotateright(P)

            else

                P =doublerotateright(P)

4. else

Print already exits

5. int m, n, d.

6. m = AVL height (P->left)

7. n = AVL height (P->right)

8. d = max(m, n)

9. P->height = d+1

10. Stop

RotateWithLeftChild(AvlNode k2)

- ➢ AvlNode k1 = k2.left;
- ➢ k2.left = k1.right;
- ➢ k1.right = k2;
- ➢ k2.height = max(height(k2.left), height(k2.right)) + 1;
- ➢ k1.height = max(height(k1.left), k2.height) + 1;
- ➢ return k1;

RotateWithRightChild(AvlNode k1)
- ➢ AvlNode k2 = k1.right;
- ➢ k1.right = k2.left;
- ➢ k2.left = k1;
- ➢ k1.height = max(height(k1.left), height(k1.right)) + 1;
- ➢ k2.height = max(height(k2.right), k1.height) + 1;
- ➢ return k2;

doubleWithLeftChild(AvlNode k3)
- ➢ k3.left = rotateWithRightChild(k3.left);
- ➢ return rotateWithLeftChild(k3);

doubleWithRightChild(AvlNode k1)

> ➢ k1.right = rotateWithLeftChild(k1.right);
> ➢ return rotateWithRightChild(k1);

Software Required: g++ / gcc compiler- / 64 bit Fedora, eclipse IDE

Input: Dictionary word and its meaning.

CODE:
Function to get max element:
```
intgetmax(int h1,int h2)
{
if(h1>h2)
return h1;
return h2;
}


avlnode * dict::rrotate(avlnode *y)
   {
cout<<endl<<"rotating right - "<< y->key;
avlnode *x=y->left;
avlnode *t=x->right;
x->right= y;
y->left=t;
y->ht=getmax(getht(y->left),getht(y->right))+1;
x->ht=getmax(getht(x->left),getht(x->right))+1;
return x;
   }
avlnode * dict::lrotate(avlnode *x)
   {
cout<<endl<<"rotating left - "<< x->key;

avlnode *y=x->right;
avlnode *t=y->left;
x->right= t;
y->left=x;
y->ht=getmax(getht(y->left),getht(y->right))+1;
x->ht=getmax(getht(x->left),getht(x->right))+1;
return y;
   }
```

OUTPUT:

[nbnssoe@localhost ~]$ ./a.out

Menu
1.Insert node
2.Inorder Display tree
Enter Choice1

enter key and meaning(single char)a
a

enter key and meaning(single char)b
b

height - 2
nodebal - -1 current key is b
enter key and meaning(single char)c
c

height - 2
nodebal - -1 current key is c
height - 3
nodebal - -2 current key is c
rotating left - a
enter key and meaning(single char)z
z

height - 2
nodebal - -1 current key is z
height - 3
nodebal - -1 current key is z
do u want to continue?(1 for continue)1

Menu
1.Insert node
2.Inorder Display tree
Enter Choice2
aa b b c c z z
do u want to continue?(1 for continue)0
[nbnssoe@localhost ~]$ */

Output: Allow Add, delete operations on dictionary and also display data in sorted order.

Conclusion: This program gives us the knowledge height balanced binary tree.

OUTCOME

Upon completion Students will be able to:

ELO1: Learn height balanced binary tree in data structure.

ELO2: Understand & implement rotations required to balance the tree.

Questions asked in university exam.

1. What is AVL tree?
2. In an AVL tree, at what condition the balancing is to be done
3. When would one want to use a balance binary search tree (AVL) rather than an array data structure

ASSIGNMENT NO. – 10

TITLE: _____

 NAME OF THE STUDENT:_____

ROLL NO: _____

DATE OF PERFORMANCE:

DATE OF SUBMISSION:

SIGNATURE:

ASSIGNMENT NO.10

Title:
Read the marks obtained by students of second year in an online examination of particular subject. Find out maximum and minimum marks obtained in a that subject. Use heap data structure. Analyze the algorithm.

Objectives:

1. To understand concept of heap
2. To understand concept & features like max heap, min heap.

Learning Objectives:
- ✓ To understand concept of heap
- ✓ To understand concept & features like max heap, min heap.

Learning Outcome:

- Define class for heap using Object Oriented features.
- Analyze working of functions.

Theory:

Theory:
Heap is a special case of balanced binary tree data structure where the root-node key is compared with its children and arranged accordingly. If **α** has child node **β** then −

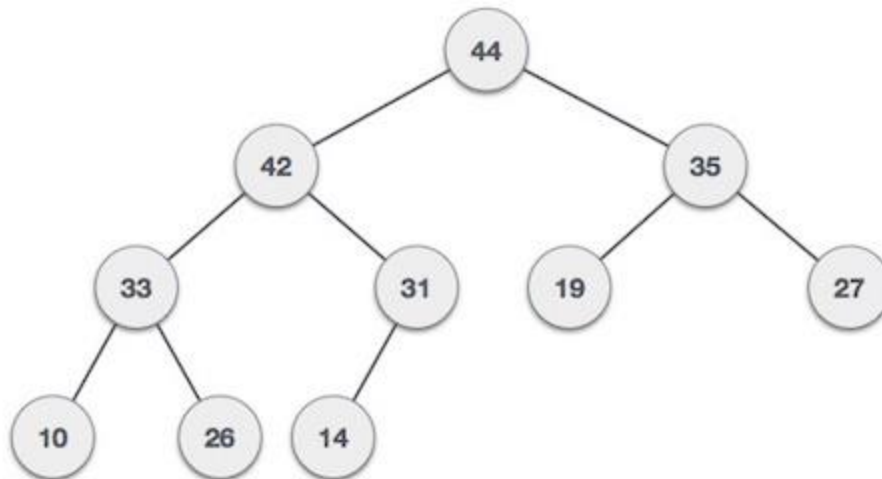$$\text{key}(\alpha) \geq \text{key}(\beta)$$

As the value of parent is greater than that of child, this property generates **Max Heap**. Based on this criteria, a heap can be of two types −

For Input → 35 33 42 10 14 19 27 44 26 31

Min-Heap − Where the value of the root node is less than or equal to either of its children.
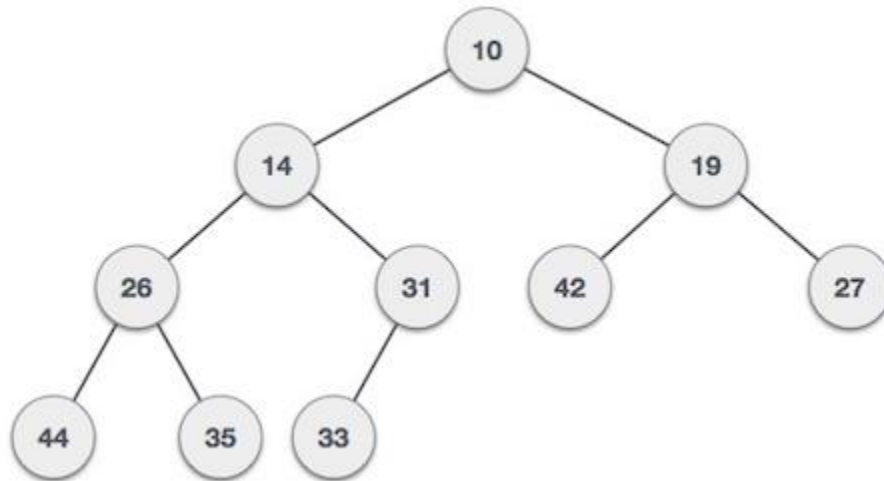
Max-Heap − Where the value of the root node is greater than or equal to either of its children.

Both trees are constructed using the same input and order of arrival.

Max Heap Construction Algorithm

We shall use the same example to demonstrate how a Max Heap is created. The procedure to create Min Heap is similar but we go for min values instead of max values.

We are going to derive an algorithm for max heap by inserting one element at a time. At any point of time, heap must maintain its property. While insertion, we also assume that we are inserting a node in an already heapified tree.

Step 1 − Create a new node at the end of heap.
Step 2 − Assign new value to the node.
Step 3 − Compare the value of this child node with its parent.
Step 4 − If value of parent is less than child, then swap them.
Step 5 − Repeat step 3 & 4 until Heap property holds.

Note − In Min Heap construction algorithm, we expect the value of the parent node to be less than that of the child node.

Let's understand Max Heap construction by an animated illustration. We consider the same input sample that we used earlier.

INPUT:35,33,42,10,14,19,27,44,16,31

Max Heap Deletion Algorithm

Let us derive an algorithm to delete from max heap. Deletion in Max (or Min) Heap always happens at the root to remove the Maximum (or minimum) value.
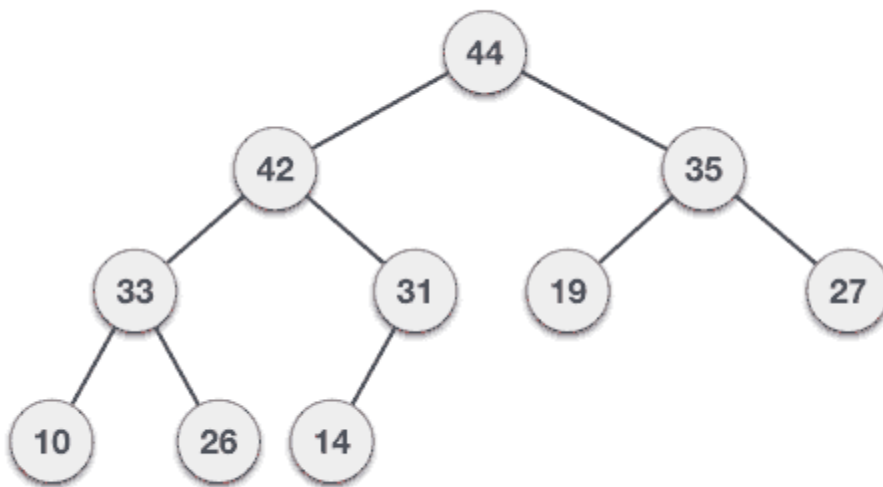
Step 1 − Remove root node.

Step 2 − Move the last element of last level to root.

Step 3 − Compare the value of this child node with its parent.

Step 4 − If value of parent is less than child, then swap them.

Step 5 − Repeat step 3 & 4 until Heap property holds.

Software Required: g++ / gcc compiler- / 64 bit Fedora, eclipse IDE

Input: Marks obtained by student..

CODE:

Function for inserting data in heap:

```
void  stud::insertheap(int tot)
{
for(inti=1;i<=tot;i++)
```

```
 {
cout<<"enter marks";
cin>>mks[i];
int j=i;
int par=j/2;
while(mks[j]<=mks[par] && j!=0)
    {
inttmp = mks[j];
mks[j]=mks[par];
mks[par]=tmp;
     j=par;
par=j/2;
    }
 }
}
```

Function to display max value in heap:

```
void stud::showmax(int tot)
{
int max1=mks[1];
for(inti=2;i<=tot;i++)
{
if(max1<mks[i])
 max1= mks[i];
}
cout<<"Max marks:"<<max1;
}
```

===================================OUTPUT================================
```
/*  OUTPUT
[jspm@localhost ~]$ ./a.out

Menu
1.Read marks of the student
2.Display  Min heap
3.Find Max Marks
4.Find Min Marks
Enter Choice1
how many students7
enter marks35
enter marks65
enter marks25
enter marks87
enter marks36
enter marks98
enter marks27
```

do u want to continue?(1 for continue)1

Menu
1.Read marks of the student
2.Display Min heap
3.Find Max Marks
4.Find Min Marks
Enter Choice2
   25

  36 27

 87 65 98 35
do u want to continue?(1 for continue)1

Menu
1.Read marks of the student
2.Display Min heap
3.Find Max Marks
4.Find Min Marks
Enter Choice3
Max marks:98
do u want to continue?(1 for continue)1

Menu
1.Read marks of the student
2.Display Min heap
3.Find Max Marks
4.Find Min Marks
Enter Choice4
Min marks:25
do u want to continue?(1 for continue)0
*/

Output: Find min and max marks ontained.

Conclusion: This program gives us the knowledge of heap and its types.

Questions asked in university exam.

1. What is Binary Heap?

2. Why is Binary Heap Preferred over BST for Priority Queue?

3. What is Binomial Heap?

4. What is Fibonacci Heap?

5. What is Heap Sort?

6. What is K'th Largest Element in an array?

ASSIGNMENT NO. – 11

TITLE: _____

NAME OF THE STUDENT:_____

ROLL NO: _____

DATE OF PERFORMANCE:

DATE OF SUBMISSION:

SIGNATURE:

-

ASSIGNMENT NO.-11

Title:

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.

Objectives:

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

Learning Objectives:
   ✓ To understand concept of file organization in data structure.
   ✓ To understand concept & features of sequential file organization.

Learning Outcome:

- Define class for sequential file using Object Oriented features.
- Analyze working of various operations on sequential file .

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. Hash/Direct File Organization
4. Indexed Sequential Access Method
5. B+ Tree File Organization

6. Cluster File Organization

Sequential File Organization:

It is one of the simple methods of file organization. Here each file/records are stored one after the other in a sequential manner. This can be achieved in two ways:

- Records are stored one after the other as they are inserted into the tables. This method is called pile file method. When a new record is inserted, it is placed at the end of the file. In the case of any modification or deletion of record, the record will be searched in the memory blocks. Once it is found, it will be marked for deleting and new block of record is entered.
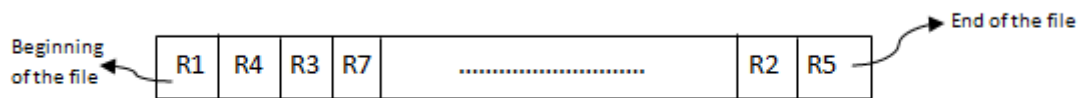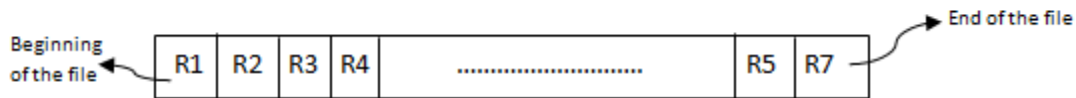

Inserting a new record:
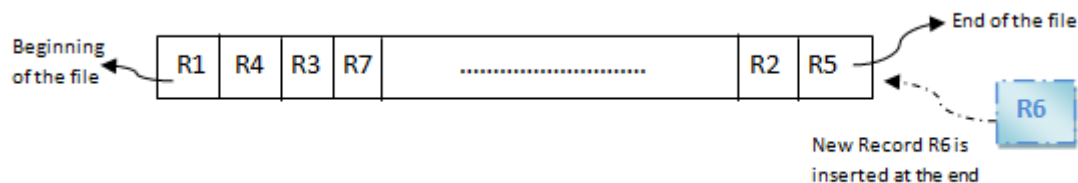
In the diagram above, R1, R2, R3 etc are the records. They contain all the attribute of a row. i.e.; when we say student record, it will have his id, name, address, course, DOB etc. Similarly R1, R2, R3 etc can be considered as one full set of attributes.
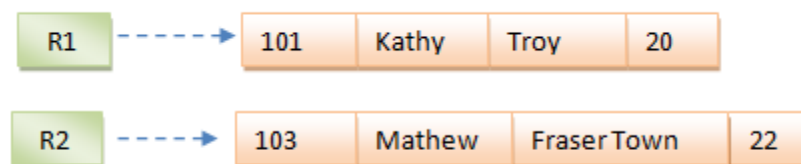

In the second method, records are sorted (either ascending or descending) each time they are inserted into the system. This method is called **sorted file method**. Sorting of records may be based on the primary key or on any other columns. Whenever a new record is inserted, it will be inserted at the end of the file and then it will sort – ascending or descending based on key value

and placed at the correct position. In the case of update, it will update the record and then sort the file to place the updated record in the right place. Same is the case with delete.

Inserting a new record:
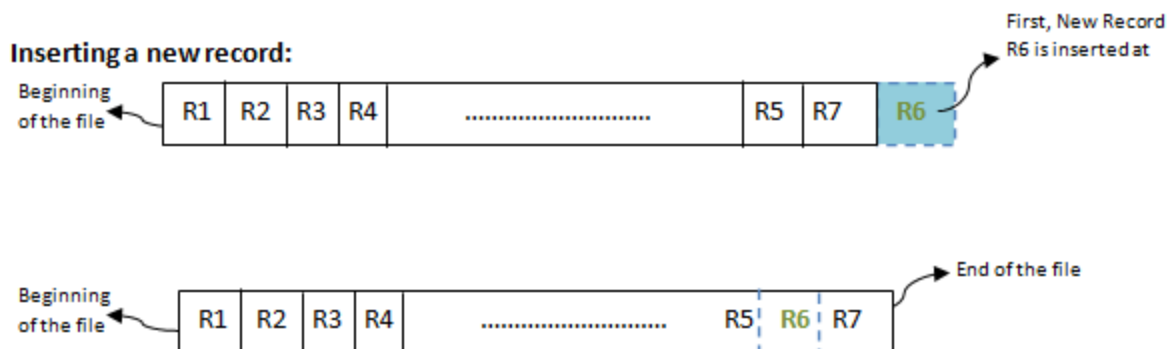
Advantages:

- Simple to understand.
- Easy to maintain and organize
- Loading a record requires only the record key.
- Relatively inexpensive I/O media and devices can be used.
- Easy to reconstruct the files.
- The proportion of file records to be processed is high.

Disadvantages:

- Entire file must be processed, to get specific information.
- Very low activity rate stored.
- Transactions must be stored and placed in sequence prior to processing.
- Data redundancy is high, as same data can be stored at different places with different keys.
- Impossible to handle random enquiries.

Software Required: g++ / gcc compiler- / 64 bit Fedora, eclipse IDE

Input: Details of student like roll no, name, address division etc.

Code:
Function To create insert data in File:
```
void STUD_CLASS::Create(char f[])
{
charch='y';
fstreamseqfile;
seqfile.open(f,ios::in|ios::out|ios::binary);
do
    {
cout<<"\n Enter Name: ";
cin>>Records.name;
cout<<"\n Enter roll no: ";
cin>>Records.rollno;
cout<<"\n Enter div: ";
cin>>Records.div;
cout<<"\n Enter Address ";
cin>>Records.Address;

      //then write the record containing this data in the file
seqfile.write((char*)&Records,sizeof(Records));
cout<<"\nDo you want to add more records?";
cin>>ch;
}while(ch=='y');
seqfile.close();
}
```

Function To display data from File:

```
void STUD_CLASS::Display(char f[])
{
fstreamseqfile;
intn,m,i;
seqfile.open(f,ios::in|ios::out|ios::binary);
   //positioning the pointer in the file at the beginning
seqfile.seekg(0,ios::beg);
cout<<"\n The Contents of file are ..."<<endl;
   //read the records sequentially
while(seqfile.read((char *)&Records,sizeof(Records)))
   {
```

```
if(Records.rollno!=-1)
     {
cout<<"\nName: "<<Records.name;
cout<<"\nRollNO: "<<Records.rollno;
cout<<"\nDivision: "<<Records.div;
cout<<"\nAddress: "<<Records.Address;
cout<<"\n";
     }
   }
intlast_rec=seqfile.tellg();//last record position
   //formula for computing total number of  objects in the file
//    n=last_rec/(sizeof(Rec));
//    cout<<"\n\n Total number of objects are "<<n<<"(considering logical deletion)";
seqfile.close();
}
```

OUTPUT:

/* output */

/*
please enter filename(make sure file is created in same folder)stud.dat

        Main Menu

1.Create
2.Display
3.Update
4.Delete
5.Append
6.Search
7.Exit
 Enter your choice 2

 The Contents of file are ...

Name: fdf
RollNO: 23
Division: a
Address: dcds

Name: fgf
RollNO: 45
Division: f
Address: fdfd

Do you want to go back to Main Menu?^[[6~^[[6~^[[6~^[[6~^[[6~^[[6~^[[6~
[nbnssoe@localhost ~]$ ./a.out

please enter filename(make sure file is created in same folder)stud.dat

       Main Menu

1.Create
2.Display
3.Update
4.Delete
5.Append
6.Search
7.Exit
 Enter your choice 6

 Enter the Roll no for searching the record 45

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

       Main Menu

1.Create
2.Display
3.Update
4.Delete
5.Append
6.Search
7.Exit
 Enter your choice 4

 For deletion,
 Enter the Roll no for searching the record 25

       Main Menu

1.Create
2.Display
3.Update
4.Delete
5.Append
6.Search
7.Exit
 Enter your choice 2

The Contents of file are ...

Name: fdf
RollNO: 23
Division: a
Address: dcds

Do you want to go back to Main Menu?*/

Output: If record of student does not exist an appropriate message is displayed otherwise the student details are displayed.

Conclusion: This program gives us the knowledge sequential file organization..

Questions asked in university exam.

1. What is file organization?
2. List three important factors of file organization.
3. Explain sequential file organization.
4. How record can be inserted in sequential file.

ASSIGNMENT NO. – 12

TITLE: _____

NAME OF THE STUDENT:_____

ROLL NO: _____

DATE OF PERFORMANCE:

DATE OF SUBMISSION:

SIGNATURE:

ASSIGNMENT NO.-12

Title:
       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:
To understand the concept and basic of index sequential file and its use in Data structure

Outcome:
To implement the concept and basic of index sequential file and to perform basic operation as adding

record, display all record, search record from index sequential file and its use in Data structure.

Software & Hardware Requirements:
64-bit Open source Linux or its derivative
Open Source C++ Programming tool like G++/GCC
Online mode: online GDB classroom, Google classroom

Theory:

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 have 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.}

Primitive operations on Index Sequential files:
• **Write (add, store):** User provides a new key and record, IS file inserts the new record and key.
• **Sequential Access (read next):** IS file returns the next record (in key order)
• **Random access (random read, fetch):** User provides key, IS file returns the record or "not there"
• **Rewrite (replace):** User provides an existing key and a new record, IS file replaces existing record
with new.
• **Delete:** User provides an existing key, IS file deletes existing record

Algorithm:
Step 1 - Include the required header files (iostream.h, conio.h, and windows.h for colors).
Step 2 - Create a class (employee) with the following members as public members.
emp_number, emp_name, emp_salary, as data members.
get_emp_details(), find_net_salary() and show_emp_details() as member functions.
Step 3 - Implement all the member functions with their respective code (Here, we have used scope
resolution operator ::).
Step 3 - Create a main() method.
Step 4 - Create an object (emp) of the above class inside the main() method.
Step 5 - Call the member functions get_emp_details() and show_emp_details().
Step 6 - returnn 0 to exit form the program execution.

Conclusion: This program gives knowledge of Index Sequential files.

Questions asked in university exam.

1. What is index sequential file?
2. What are the primitive operations on Index Sequential files ?