INDEX

Sr. No.	Group	Title of Lab Assignment	CO	PO
1	A	Consider a 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 (Python)	CO1	PO1, PO2, PO3,PO4, PO12
2		Implement all the functions of a dictionary (ADT) using hashing and handle collisions using chaining with / without replacement. Data: Set of (key, value) pairs, Keys are mapped to values, Keys must be comparable, Keys must be unique Standard Operations: Insert(key, value), Find(key), Delete(key) (python)	CO1	PO1, PO2, PO3,PO4, PO12
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.	CO2	PO1, PO2, PO3,PO4, PO12
4	B	Beginning with an empty binary search tree, Construct binary search tree by inserting the values in the order given. After constructing a binary tree - ● Insert new node ● Find number of nodes in longest path from root ● Minimum data value found in the tree ● Change a tree so that the roles of the left and right pointers are swapped at every node ● Search a value	CO2	PO1, PO2, PO3,PO4, PO12
5		Construct an expression tree from the given prefix expression eg. +-- a*bc/def and traverse it using postorder traversal(non recursive) and then delete the entire tree.	CO2	PO1, PO2, PO3,PO4
6	C	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. Check whether the graph is connected or not. Justify the storage representation used.	CO3	PO1, PO2, PO4,
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	CO3	PO1, PO3,PO4, PO12

		your offices with a minimum total cost. Solve the problem by suggesting appropriate data structures.		
8	D	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?	CO4	PO1, PO2, PO3
9		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	CO4	PO1, PO3,PO4, PO12
10	E	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.	CO5	PO1, PO2, PO3,PO4, PO12
11	F	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 maintain the data.	CO6	PO1, PO3,PO4, PO12
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.	CO6	PO1, PO3,PO4, PO12
13	Content Beyond Syllabus	Design a mini project to implement snake and ladder game using Python.	CO1, CO2	PO1,PO2
14	V-Lab	Study of DFS on binary trees	CO2	PO1,PO2

GROUP: A

ASSIGNMENT NO: 1

Title: Implementation of hash Table.

Objective: To study the implementation of hash Table.

Problem Statement: Consider a 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 (Python)

Outcome: Explain the concepts of Hashing and apply it to solve searching problems

Theory:

In a mathematical sense, a map is a relation between two sets. We can define Map M as a set of pairs, where each pair is of the form (key, value), where for given a key, we can find a value using some kind of a “function” that maps keys to values. The key for a given object can be calculated using a function called a hash function. In its simplest form, we can think of an array as a Map where key is the index and value is the value at that index. For example, given an array A, if i is the key, then we can find the value by simply looking up A[i]. The idea of a hash table is more generalized and can be described as follows. The concept of a hash table is a generalized idea of an array where key does not have to be an integer. We can have a name as a key, or for that matter any object as the key. The trick is to find a hash function to compute an index so that an object can be stored at a specific location in a table such that it can easily be found. Example: Suppose we have a set of strings {“abc”, “def”, “ghi”} that we’d like to store in a table. Our objective here is to find or update them quickly from a table, actually in O(1). We are not concerned about ordering them or maintaining any order at all. Let us think of a simple schema to do this. Suppose we assign “a” = 1, “b”=2, … etc to all alphabetical characters. We can then simply compute a number for each of the strings by using the sum of the characters as follows. “abc” = 1 + 2 + 3=6, “def” = 4 + 5 + 6=15 , “ghi” = 7 + 8 + 9=24 If we assume that we have a table of size 5 to store these strings, we can compute the location of the string by taking the sum mod 5. So we will then store “abc” in 6 mod 5 = 1, “def” in 15 mod 5 = 0, and “ghi” in 24 mod 5 = 4 in locations 1, 0 and 4 as follows-

0	1	2	3	4
Def	Abc			ghi

Now the idea is that if we are given a string, we can immediately compute the location using a simple hash function, which is sum of the characters mod Table size. Using this hash value, we can search for the string. This seems to be great way to store a Dictionary. Therefore the idea of hashing seems to be a great way to store pairs of (key, value) in a table.

Implementation of a Simple Hash Table:

A hash table is stored in an array that can be used to store data of any type. In this case, we will define a generic table that can store nodes of any type. That is, an array of void*'s can be defined as follows-

```
 void* A[n];
```

The array needs to be initialized using

```
 for (i = 0; i < n ; i++)
A[i] = NULL;
```

Suppose we like to store strings in this table and be able to find them quickly. In order to find out where to store the strings, we need to find a value using a hash function. One possible hash function is

Given a string S = S1S2…. Sn

Define a hash function as

H(S) = H("S1S2…. Sn") = S1 + p S2 + p2 S3 + ….+ pn-1 Sn (1)

where each character is multiplied by a power of p, a prime number.

The above equation can be factored to make the computation more effective. Using the factored form, we can define a function hash code that computes the hash value for a string s as follows—

```
int hashcode(char* s){
int sum = s[strlen(s)-1], p = 101;
int i;
for (i=1;i< strlen(s);i++)
sum-s[strlen(s)-i-1]+p*sum;
return sum;
}
```

This allows any string to be placed in the table as follows. We assume a table of size 101. A[hashcode(s) %101] = s; // we assume that memory for s is already being allocated.

One problem with above method is that if any collisions occur, that is two strings with the same hash code,& we will lose one of the strings. Therefore we need to find a way to handle collisions in the table.

Collisions:

One problem with hashing is that it is possible that two strings can hash into the same location. This is called a collision. We can deal with collisions using many strategies, such as linear probing (looking for the next available location i+1, i+2, etc. from the hashed value i), quadratic probing (same as linear probing, except we look for available positions i+1 , i + 4, i + 9, etc from the hashed value i and separate chaining, the process of creating a linked list of values if they hashed into the same location.

Algorithm:

1: Create a hash table with maximum size.

2: Define Hash function.

3: Read the telephone no & user's information to insert it into hash table.

4: Insert telephone no value in the hash table.

5: If collision occurs, apply separate chaining method.

6: If user requires inserting another data, go to step4.

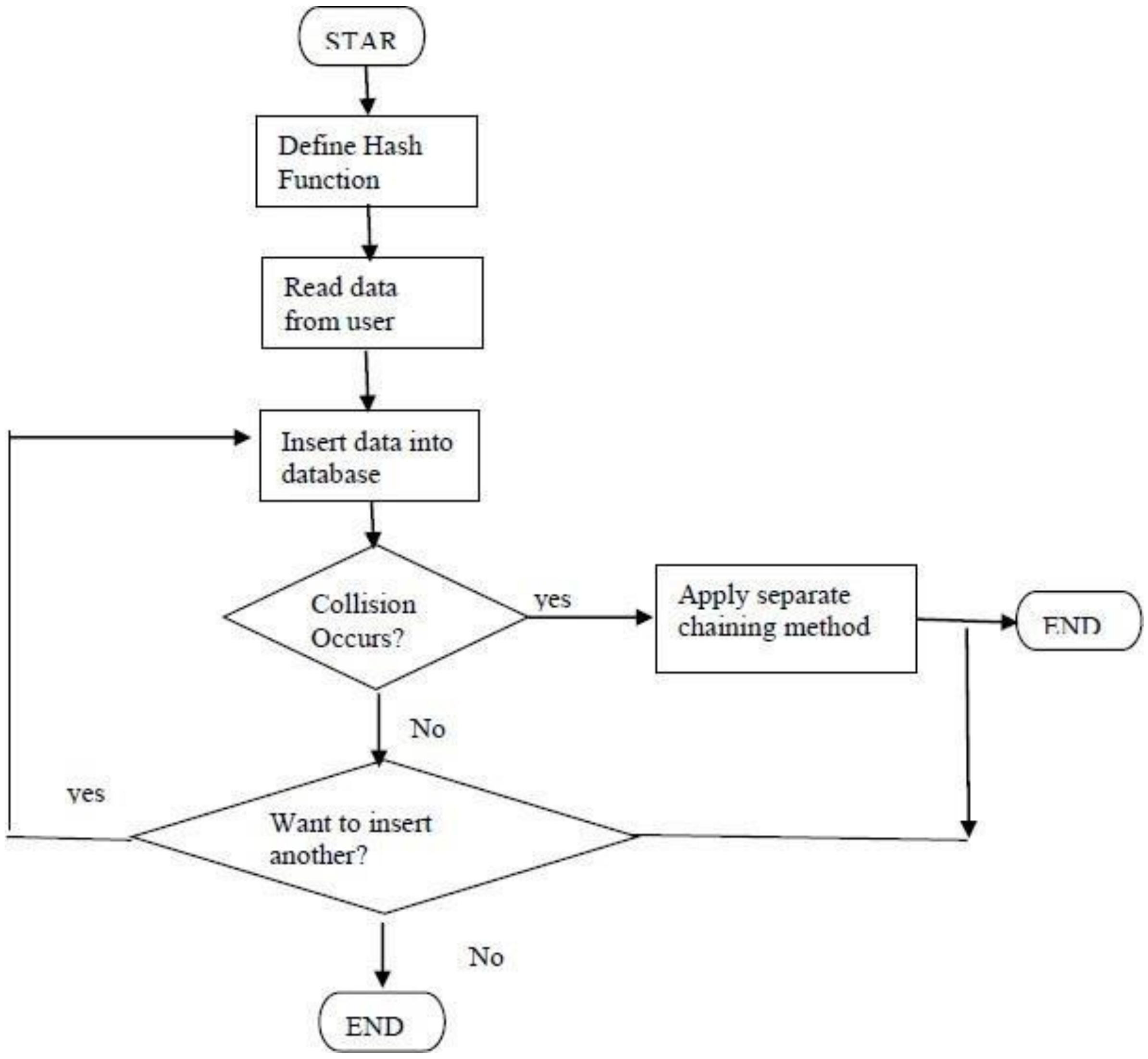
7: Display the all user's data.

8: Read the telephone no from user to be found.

9: find the data from hash table.

10: Display the data with minimum no. of comparisons.

Flowchart:



Test Cases:

Sr. No	Test ID	Steps	Input	Expected Result	Actual Result	Status (Pass/ Fail)
1	ID01	Insert telephone no in the hash table	Telephone no & the reference of hash table	Data to be inserted in the hash table	If key not found, segmentation fault	FAIL
2	ID02	Insert telephone no in the hash table	Telephone no & the reference of hash table	Data to be inserted in the hash table	Data is inserted	PASS
3	ID03	Find the telephone no in the hash table	Telephone no & the reference of hash table	Data found	If not present, not found	FAIL
4	ID04	Find the telephone no in the hash table	Telephone no & the reference of hash table	Data found	If present, not found	PASS

Conclusion:
Hence we have studied successfully the use of a hash table & implementing it.

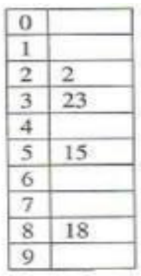
Q 1. A hash table of length 10 uses open addressing with hash function h(k)=k mod 10, and linear probing. After inserting 6 values into an empty hash

0	
1	
2	42
3	23
4	34
5	52
6	46
7	33
8	
9	

Q. 2 Which one of the following choices gives a possible order in which the key values could have been inserted in the table?
(A) 46, 42, 34, 52, 23, 33
(B) 34, 42, 23, 52, 33, 46
(C) 46, 34, 42, 23, 52, 33
(D) 42, 46, 33, 23, 34, 52
Ans: (C)
The sequence (A) doesn’t create the hash table as the element 52 appears before 23 in this sequence.
The sequence (B) doesn’t create the hash table as the element 33 appears before 46 in this sequence.
The sequence (C) creates the hash table as 42, 23 and 34 appear before 52 and 33, and 46 appears before 33.
The sequence (D) doesn’t create the hash table as the element 33 appears before 23 in this sequence.

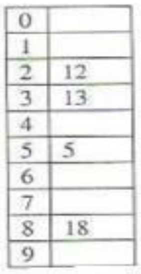
Q 2. How many different insertion sequences of the key values using the same hash function and linear probing will result in the hash table shown above?
(A) 10
(B) 20
(C) 30
(D) 40
Ans: (C)
Explanation: In a valid insertion sequence, the elements 42, 23 and 34 must appear before 52 and 33, and 46 must appear before 33.Total number of different sequences = 3! x 5 = 30. In the above expression, 3! is for elements 42, 23 and 34 as they can appear in any order, and 5 is for element 46 as it can appear at 5 different places.

Q 3. The keys 12, 18, 13, 2, 3, 23, 5 and 15 are inserted into an initially empty hash table of length 10 using open addressing with hash function h(k) = k mod 10 and linear probing. What is the resultant hash table?



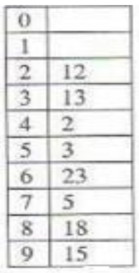
0	
1	
2	2
3	23
4	
5	15
6	
7	
8	18
9	

(A)



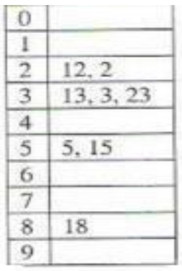
0	
1	
2	12
3	13
4	
5	5
6	
7	
8	18
9	

(B)



0	
1	
2	12
3	13
4	2
5	3
6	23
7	5
8	18
9	15

(C)



0	
1	
2	12, 2
3	13, 3, 23
4	
5	5, 15
6	
7	
8	18
9	

(D)

(A) A

(B) B

(C) C

(D) D

Ans: (C)

Explanation: To get the idea of open addressing concept, you can go through below lines from .Open addressing, or closed hashing, is a method of collision resolution in hash tables. With this method a hash collision is resolved by probing, or searching through alternate locations in the array (the probe sequence) until either the target record is found, or an unused array slot is found, which indicates that there is no such key in the table. Well known probe sequences include:

1. *linear probing* in which the interval between probes is fixed–often at 1.
2. *quadratic probing* in which the interval between probes increases linearly (hence, the indices are described by a quadratic function).
3. *double hashing* in which the interval between probes is fixed for each record but is computed by another hash function.

Q 4. What terminates a failed linear probe in a full hash table?

(A) The end of the array

(B) A deleted node

(C) A null entry

(D) A node with a non-matching key

(E) Revisiting the original hash index

Ans: A null entry will not appear in a full hash table. Seeing the end of the array isn't correct, since we need to examine all elements, including those that appear before our original hash index. A node with a non-matching key is what started our probe in the first place. The purpose of leaving a deleted node in the table is so that probing may proceed past it. Revisiting the original hash index means we've looked at every entry and determined the item doesn't appear in the table.

Q 5. If a hash table's array is re-sized to reduce collisions, what must be done to the elements that have already been inserted?
Ans: Since calculating a node's position in the hash table is a function of the node's key's hash code and the array size, all items must be reinserted.

Q 6. You need to store a large amount of data, but you don't know the exact number of elements. The elements must be searched quickly by some key. You want to waste no storage space. The elements to be added are in sorted order. What is the simplest data structure that meets your needs?
Ans: Hash tables provide fast searching, but they may waste storage space. A tree makes better use of memory. Since the keys are in a sorted order, it's likely a binary tree will end up looking like a linked list instead of a well-balanced tree. With a self-balancing tree, we can make sure searching goes faster.

Q 7. Is the hash table is ADT?

Ans: Hash table is a data structure, not an abstract data type.

Q 8. In a hash table that uses separate chaining to handle collisions, what, if any, restrictions should be placed on the table size?
Ans: With separate chaining, no probing is done. Each entry in the table is a linked list, and all items with the hash index appear in the list.

Q 9. Hashing is the best search method (constant running time) if we don't need to have the records sorted. In that case which method will you use separate chaining or open addressing?
Ans: If there is enough memory to keep a table twice larger than the number of the records - open addressing is the preferred method. Separate chaining is used when we don't know in advance the number of the records to be stored. It requires additional time for list processing; however, it is simpler to implement.

Q 10. What is Collision?

Ans: Since a hash function gets us a small number for a key which is a big integer or string, there is possibility that two keys result in same value. The situation where a newly inserted key maps to an already occupied slot in hash table is called collision and must be handled using some collision handling technique.

ASSIGNMENT NO: 02

Title: Perform different operations and techniques on Hash table

Objective: To understand the collision resolution techniques and various operations on hash table

Problem Statement: Implement all the functions of a dictionary (ADT) using hashing and handle collisions using chaining with / without replacement. Data: Set of (key, value) pairs, Keys are mapped to values, Keys must be comparable, Keys must be unique Standard Operations: Insert(key, value), Find(key), Delete(key) (python)

Outcome: Define class for Dictionary using Object Oriented features. Analyze working of hash function.

Theory:
Dictionary ADT Dictionary (map, association list) is a data structure, which is generally an association of unique keys with some values. One may bind a value to a key, delete a key (and naturally an associated value) and lookup for a value by the key. Values are not required to be unique. Simple usage example is an explanatory dictionary. In the example, words are keys and explanations are values.
Dictionary Operations:

- Dictionary create() creates empty dictionary
- boolean isEmpty(Dictionary d) tells whether the dictionary d is empty
- put(Dictionary d, Key k, Value v) associates key k with a value v; if key k already presents in the dictionary old value is replaced by v
- Value get(Dictionary d, Key k) returns a value, associated with key kor null, if dictionary contains no such key
- remove(Dictionary d, Key k) removes key k and associated value
- destroy(Dictionary d) destroys dictionary d Hash Table is a data structure which stores data in an associative manner.

In a hash table, data is stored in an array format, where each data value has its own unique index value. Access of data becomes very fast if we know the index of the desired data. Thus, it becomes a data structure in which insertion and search operations are very fast irrespective of the size of the data. Hash Table uses an array as a storage medium and uses hash technique to generate an index where an element is to be inserted or is to be located from.

Hashing: Hashing is a technique to convert a range of key values into a range of indexes of an array. We're going to use modulo operator to get a range of key values. Consider an example of hash table of size 20, and the following items are to be stored. Item are in the (key,value) format.

Basic Operations of hash table 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.

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

```
    struct DataItem {
        int data; int key;
        };
```

Hash Method Define a hashing method to compute the hash code of the key of the data item. int hashCode(int key)

```
      {
            return key % SIZE;
            }
```

2. 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

```
       struct DataItem *search(int key)
       {
//get the hash int hashIndex = hashCode(key); //move in array until an empty while(hashArray[hashIndex]
!= NULL)
 {
if(hashArray[hashIndex]->key    ==    key)
return hashArray[hashIndex];
//go to next cell ++hashIndex;
//wrap around the table hashIndex %= SIZE;
}
return NULL; }
```

3. 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(int key,int data)

```
{ struct DataItem *item = (struct DataItem*) malloc(sizeof(struct DataItem));
item->data = data;
item->key = key;
//get the hash int hashIndex = 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; }
```

4.. 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.

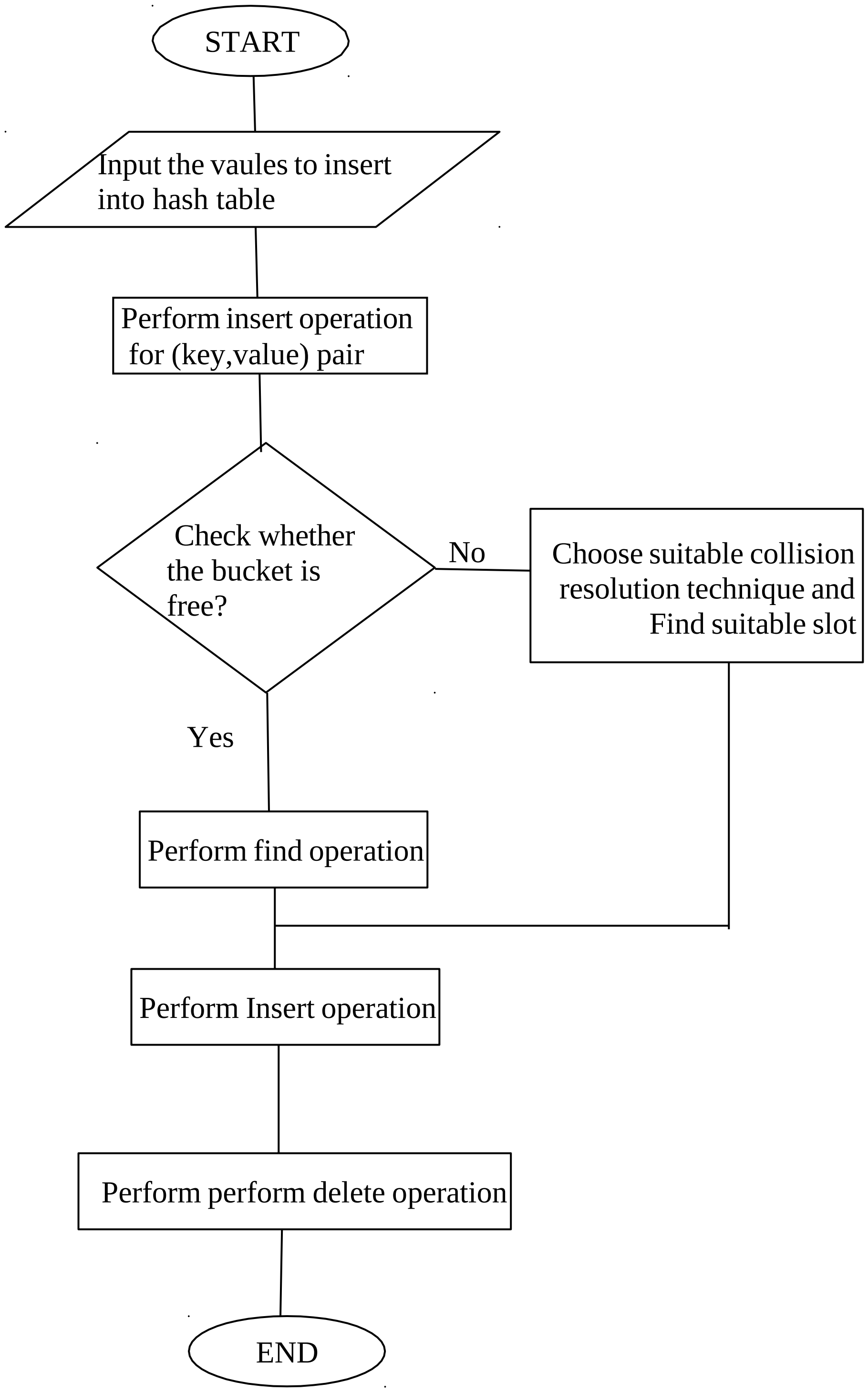
Example struct DataItem* delete(struct DataItem* item)

```
{ int key = item->key;
 //get the hash int hashIndex = hashCode(key);
//move in array until an empty
 while(hashArray[hashIndex] !=NULL)
{ if(hashArray[hashIndex]->key == key)
 { struct DataItem* 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; }

Input: No. of. elements with key and value pair.
Output: Create dictionary using hash table and search the elements in table.

Flowchart:


START
Input the vaules to insert into hash table
Perform insert operation for (key,value) pair
Check whether the bucket is free?
No
Choose suitable collision resolution technique and Find suitable slot
Yes
Perform find operation
Perform Insert operation
Perform perform delete operation
END

Test Cases:

Sr. No	Test ID	Steps	Input	Expected Result	Actual Result	Status (Pass/ Fail)
1	01	Insert (4, Asmita)	Read (4, Asmita)	The (4, Asmita) is inserted in it's home bucket	If bucket is free and (4, Asmita) is insered	Pass
2	02	Insert (4, Asmita)	Read (4, Asmita)	The record inserted in the hash table	If calculate d incorrect bucket	fail

Conclusion:
This program gives us the knowledge of dictionary(ADT).
OUTCOME Upon completion Students will be able to:
ELO1: Learn object-oriented Programming features.
ELO2: Understand & implement Dictionary (ADT) using hashing.

FAQ:

Q.1 How do you solve a collision hash?
One method for resolving collisions looks into the hash table and tries to find another open slot to hold the item that caused the collision. A simple way to do this is to start at the original hash value position and then move in a sequential manner through the slots until we encounter the first slot that is empty.

Q.2 What is hashing and collision?
a hash collision is a random match in hash values that occurs when a hashing algorithm produces the same hash value for two district pieces of data. The hashing process provides the security layer necessary for securing the transmission of a message to its recipient.

Q.3 What is hashing in data structure with example?
Hashing is a technique or process of mapping keys, values into the hash table by using a hash function. It is done for faster access to elements. For example, if the list of values is [11,12,13,14,15] it will be stored at positions {1,2,3,4,5} in the array or Hash table respectively.

Q.4 What happens in hashing method to overcome collision?
Collisions that occur during hashing need to be resolved. In order to tackle collisions, the hash table can be restructured where each hash location can accommodate more than one item that is each location is a "bucket" or an array itself. Another method is to design the hash table as an array of linked chains.

Q.5 What is a hash index?

What is a hash index? Basically, a hash index is an array of N buckets or slots, each one containing a pointer to a row. Hash indexes use a hash function F(K, N) in which given a key K and the number of buckets N , the function maps the key to the corresponding bucket of the hash index.

Q.6 What do you mean by hash table?

The hash table data structure is merely an array of some fixed size, containing the keys. A key is a string with an associated value. Each key is mapped into some number in the range 0 to tablesize-1 and placed in the appropriate cell.

Q.7 Define Hashing.

Hashing is the transformation of string of characters into a usually shorter fixed length value or key that represents the original string. Hashing is used to index and retrieve items in a database because it is faster to find the item using the short-hashed key than to find it using the original value

Q.8 What do you mean by separate chaining?

Separate chaining is a collision resolution technique to keep the list of all elements that hash to the same value. This is called separate chaining because each hash table element is a separate chain (linked list). Each linked list contains all the elements whose keys hash to the same index

Q.9 What do you mean by open addressing?

Open addressing is a collision resolving strategy in which, if collision occurs alternative cells are tried until an empty cell is found. The cells h0(x), h1(x), h2(x),…. are tried in succession, where hi(x)=(Hash(x)+F(i)) mod Table size with F(0)=0. The function F is the collision resolution strategy.

Q.10 Write the importance of hashing.

- Maps key with the corresponding value using hash function.
- Hash tables support the efficient addition of new entries and the time spent on searching for the required data is independent of the number of items stored.

ASSIGNMENT NO: 03

Title:

Printing the tree level wise.

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

Problem Statement:

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.

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 tree.

A node that has no child 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 formally, 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.

Every node in a tree can be seen as the root node of the subtree rooted at that node.

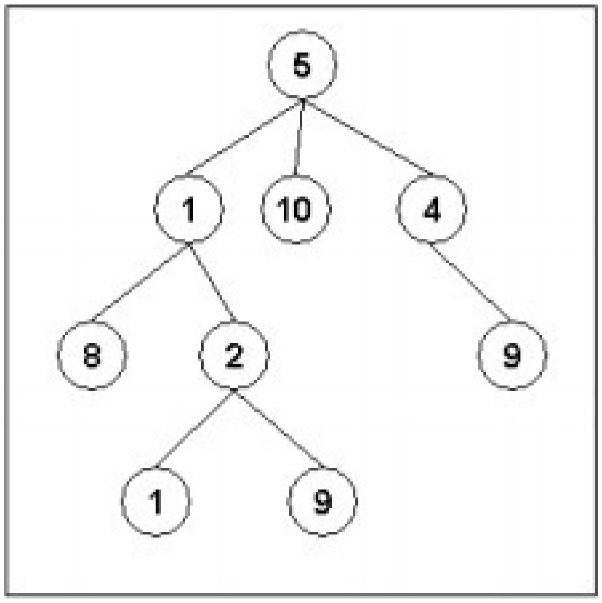


Fig1. An example of a tree

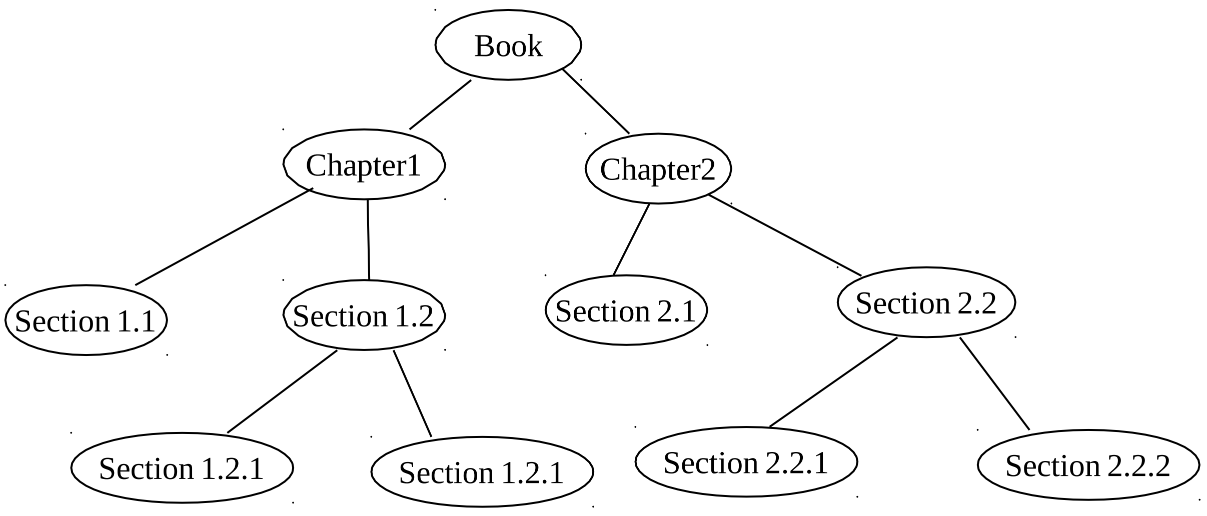
Important Terms

Following are the important terms with respect to tree.

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

Advantages of Trees

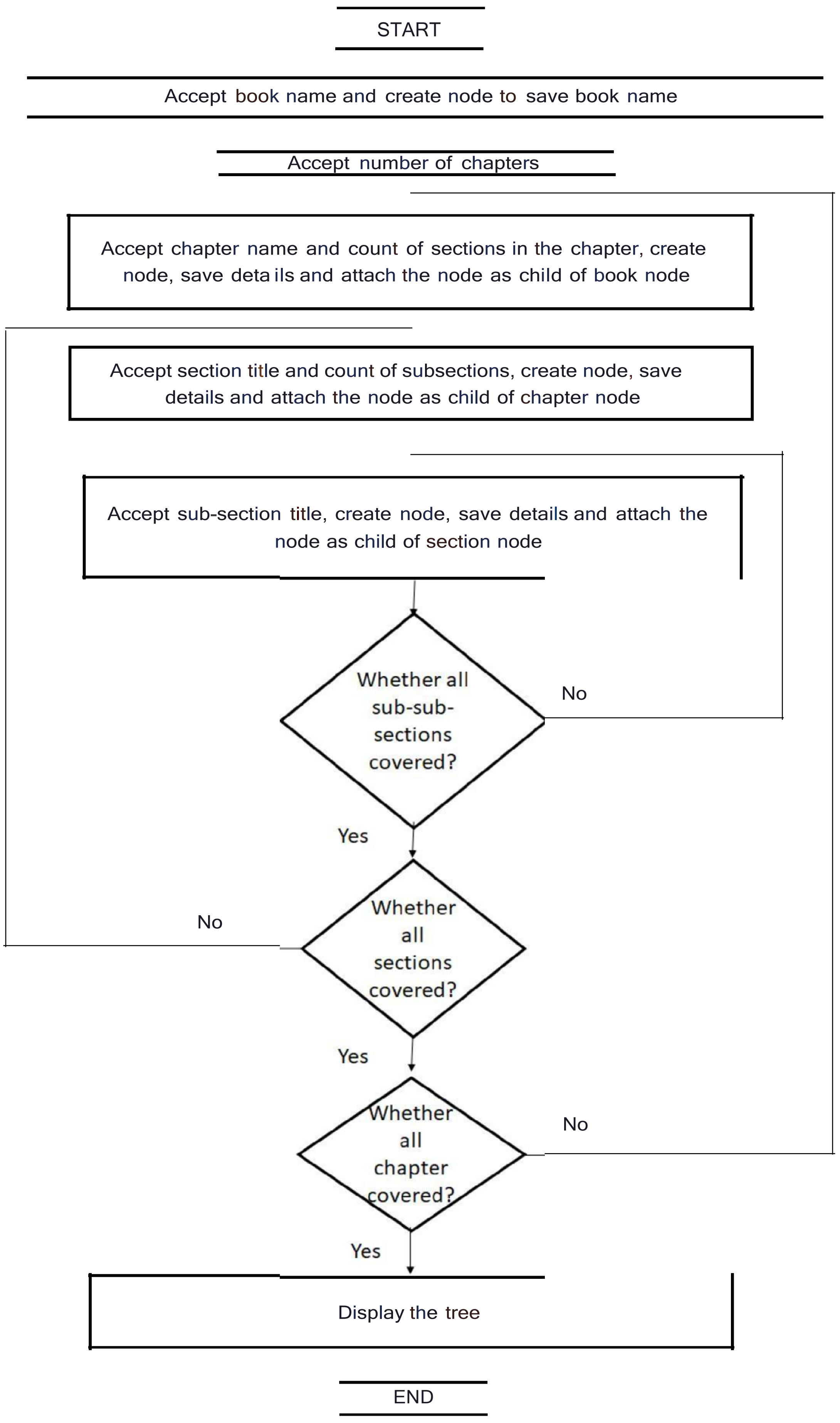
- Trees are so useful and frequently used, because they have some very serious advantages:
- Trees reflect structural relationships in the data
- Trees are used to represent hierarchies
- Trees provide an efficient insertion and searching
- Trees are very flexible data, allowing to move subtrees around with minimum effort for this assignment we are considering the tree as follows.



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

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

Flowchart:


START
Accept book name and create node to save book name
Accept number of chapters
Accept chapter name and count of sections in the chapter, create node, save details and attach the node as child of book node
Accept section title and count of subsections, create node, save details and attach the node as child of chapter node
Accept sub-section title, create node, save details and attach the node as child of section node
Whether all sub-sub-sections covered?
No
Yes
Whether all sections covered?
No
Yes
Whether all chapter covered?
No
Yes
Display the tree
END

Test-cases:

Sr. No.	Test ID	Steps	Input	Expected Result	Actual Result	Status (Pass/ Fail)
1	ID1	Enter count of chapters	Number of chapters is nonzero positive value	Nodes for chapters are created in the tree and attached as the children of book node	Book node and its children chapters are created	Pass
2	ID2	Enter count of sections	Number of sections is nonzero positive value	Nodes for sections are created in the tree and attached as the children of chapter node	chapter node and its children sections are created	Pass
3	ID3	Enter count of sub-sections	Number of sub-sections is nonzero positive value	Nodes for sub-sections are created in the tree and attached as the children of section node	Section node and its children sub-section nodes are created	Pass
4	ID1	Enter number of subsections as 0	Number of subsections: 0	No subsection node should be created	Number of subsections is zero hence no children node are introduced for section node	Pass

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

OUTCOME:
Upon completion Students will be able to:
ELO1: Learn object-oriented Programming features.
ELO2: Understand & implement tree data structure.

FAQs:

1. What is class, object and data structure?
A class can define a set of properties/fields that every instance/object of that class inherits. A data structure is a way to organize and store data. Technically a data structure is an object, but it's an object with the specific use for holding other objects (everything in Java is an object, even primitive types).

2. What is tree data structure?
A tree is a nonlinear data structure, compared to arrays, linked lists, stacks and queues which are linear data structures. A tree can be empty with no nodes or a tree is a structure consisting of one node called the root and zero or one or more subtrees.

3. Explain different types of tree?
The binary tree is the kind of tree in which most two children can be found for each parent......This is more popular than most other trees. When certain constraints and characteristics are applied in a Binary tree, a number of others such as AVL tree, BST (Binary Search Tree), RBT tree, etc. are also used.

4. What is difference between tree and Graph?
Graph is a non-linear data structure. **Tree** is a non-linear data structure. It is a collection of vertices/nodes and edges. It is a collection of nodes and edges.

5. How do you check if a given binary tree is a subtree of another binary tree?
Consider we have a binary tree T. We now want to check if a binary tree S is a subtree of T.To do this, first, try to check if you find a node in T that is also in S. Once you find this common node, check if the following nodes are also a part of S. If yes, we can safely say that S is a subtree of T

6. How do you find the distance between two nodes in a binary tree?
Consider two nodes n1 and n2 that are part of a binary tree. The distance between n1 and n2 is equal to the minimum number of edges that need to be traversed to reach from one node to the other. It is important to note that you traverse the shortest distance between the nodes.

7. What is the Red-Black tree data structure?
The Red-Black tree is a special type of self-balancing tree that has the following properties:

1. Each node has a colour either red or black.
2. The root is always black.
3. A red node cannot have a red parent or red child.
4. Every path from the root node to a NULL node has the same number of black nodes.

8. What is Binary Tree and its application?

binary trees are used in two very different ways: First, as a means of accessing nodes based on some value or label associated with each node. Binary trees labelled this way are used to implement binary search trees and binary heaps, and are used for efficient searching and sorting.

9. How trees are stored in memory?

Usually, it's stored as an adjacency list. Which is basically a linked list for every single node. So the linked list of a node u contains every node v such that (u,v) is a valid edge of the tree.But that's used when the nodes of the tree are less (since it requires more memory).

GROUP: B
ASSIGNMENT NO: 04

Title: Different operations on binary tree.

Objective: To study Different operations on binary tree.

Problem Statement: Beginning with an empty binary search tree, construct binary search tree by inserting the values in the order given. After constructing a binary tree -
i. Insert new node
ii. Find number of nodes in longest path from root
iii. Minimum data value found in the tree
iv. Change a tree so that the roles of the left and right pointers are swapped at every node
v. Search a value

Outcome: Classify the Tree data structures & apply it for problem solving

Theory:
A binary tree is composed of nodes connected by edges. Some binary tree is either empty or consists of a single root element with two distinct binary tree child elements known as the left subtree and the right subtree of a node. As the name binary suggests, a node in a binary tree has a maximum of children.
An important special kind of binary tree is the binary search tree (BST). In a BST, each node stores some information including a unique key value, and perhaps some associated data. A binary tree is a BST iff, for every node n in the tree:

- All keys in n's left subtree are less than the key in n, and
- all keys in n's right subtree are greater than the key in n.

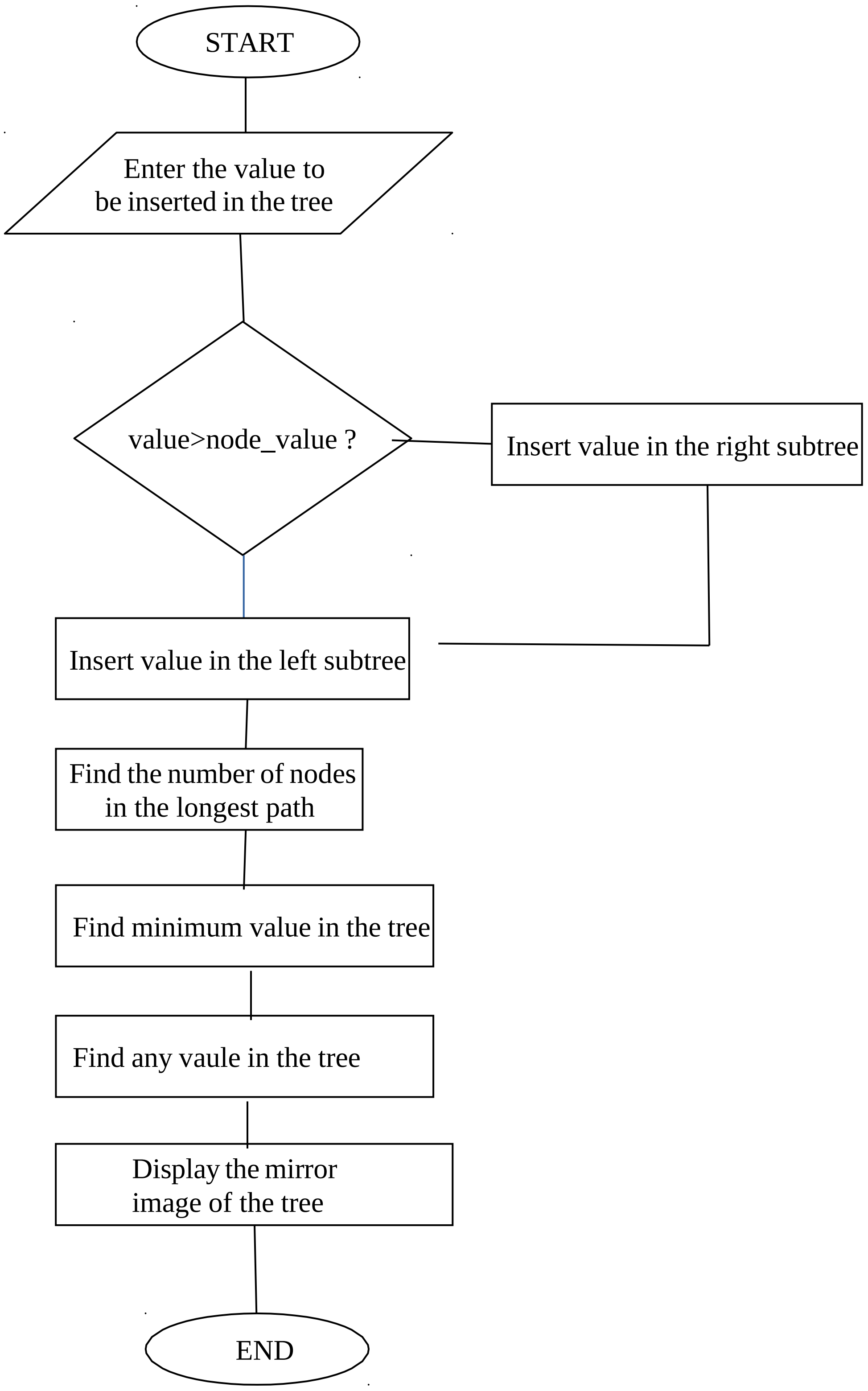
Algorithm:
Algorithm to insert a node in the binary tree:

```
1. Create a new BST node and assign values to it.
2. insert(node, key)
   i) If root == NULL,
   return the new node to the calling function.
   ii) if root=>data < key
   call the insert function with root=>right and assign the return value in root=>right.
   root->right = insert(root=>right,key)
   iii) if root=>data > key
   call the insert function with root->left and assign the return value in root=>left.
   root=>left = insert(root=>left,key)
3. Finally, return the original root pointer to the calling function.
```

Algorithm to find the minimum value in binary tree:

1. Define Node class which has three attributes namely: data, left and right. Here, left represents the left child of the node and right represents the right child of the node.
2. When a node is created, data will pass to data attribute of node and both left and right will be set to null.
3. Define another class which has an attribute root.
   A) Root represent root node of the tree and initialize it to null.

Flow-Chart:


START
Enter the value to be inserted in the tree
value>node_value ?
Insert value in the right subtree
Insert value in the left subtree
Find the number of nodes in the longest path
Find minimum value in the tree
Find any vaule in the tree
Display the mirror image of the tree
END

Test Cases:

Sr. No	Test ID	Steps	Input	Expected Result	Actual Result	Status (Pass/Fail)
1	01	Inserting a node in binary search tree	Read value from user	Value inserted either in left subtree or right subtree	Segmentation fault	Fail
2	02	Finding a value in the tree	Read value from user	According to the property of BST , it should disply the result	If inserted exactly reverse of the property	fail
3	03	Finding a value in the tree	Read value from user	According to the property of BST , it should disply the result	If inserted exactly according to the property of BST	Pass

Conclusion:

Hence we have studied successfully how to delete any node from binary tree using operator overloading.

FAQs:

Q 1. List any four Operators that cannot be overloaded?

Ans: Following operators cannot be overloaded are Class member access operator (. , .*), Scope resolution operator (::), Size operator (sizeof) & Conditional operator (?:).

Q 2. What is a Copy Constructor?

Ans: A copy constructor is used to declare and initialize an object from another object. It takes a reference to an object of the same class as an argument
Eg: integer i2(i1);

would define the object i2 at the same time initialize it to the values of i1.

Another form of this statement is Eg: integer i2=i1;
The process of initializing through a copy constructor is known as copy initialization

Q 3. Give an example for a Copy Constructor.

Ans:

```
#include<iostream>
#include<conio.h>
using namespace std;
class Example
{
int a,b; // Variable Declaration
public:
        Example(int x,int y)     //Constructor with Argument
         {
          a=x; b=y;     // Assign Values In Constructor
          cout<<"\n Im Constructor";
        }
          void Display()
        {
          cout<<"\n Values :"<<a<<"\t"<<b;
         }
}
```

Q 4. What is the need for Overloading an operator?

Ans: To define a new relation task to an operator, we must specify what it means in relation to the class to which the operator is applied. This is done with the help of a special function called operator function. It allows the developer to program using notation closer to the target domain and allow user types to look like types built into the language. It provides the ability to tell the compiler how to perform a certain operation when its corresponding operator is used on one or more variables.

Q 5. What are the applications of binary tree?

Ans: Binary tree is used in data processing.

a. File index schemes.

b. Hierarchical database management system.

Q 6. List out the cases involved in deleting a node from a binary search tree.

Ans: Case 1: Node to be deleted is a Leaf node.
Case 2: Node to be deleted has one child.
Case 3: Node to be deleted has two children.

Q 7. A + (B-C)*D+(E*F), if the above arithmetic expression is represented using a binary tree, Find the number of non-leaf nodes in the tree.

Ans: Expression tree is a binary tree in which non – leaf nodes are operators and the leaf nodes are operands. In the above example, we have 5 operators. Therefore the number of non-leaf nodes in the tree is 5.

Q 8. Give various implementations of trees.

Ans: Linear implementation Linked list implementation.

Q 9. Define binary search tree. Why it is preferred rather than the sorted linear array and linked list?

Ans: Definition: Binary search tree is a binary tree in which key values of the left sub trees are lesser than the root value and the key values of the right sub tree are always greater than the root value. Reason-

- In linear array or linked list the values are arranged in the form of increasing or decreasing order. If we want to access any element means, we have to traverse the entire list.

- But if we use BST, the element to be accessed is greater or smaller than the root element means we can traverse either the right or left sub tree and can access the element irrespective of searching the entire tree.

Q 10. Define complete binary tree.

Ans: It is a complete binary tree only if all levels, except possibly the last level have the maximum number of nodes maximum. A complete binary tree of height ‘h’ has between 2 h and 2h+1 – 1 node.

ASSIGNMENT NO: 5

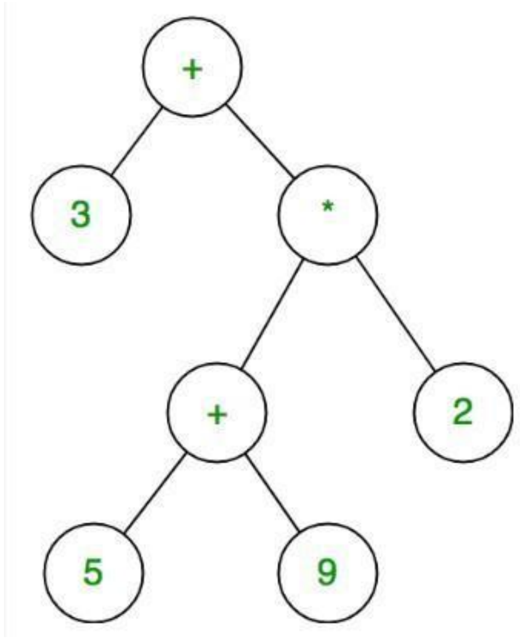
Title: Study of Expression tree.

Objective: Understand construction of expression tree from given prefix expression.
Understand Non Recursive post order traversal.

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

Outcome: Postorder expression for given prefix expression

Theory:
The expression tree is a binary tree in which each internal node corresponds to the operator and each leaf node corresponds to the operand so for example expression tree for 3 + ((5+9)*2) would be:



There are different types of expression formats:

- Prefix expression
- Infix expression and
- Postfix expression

Expression Tree is a special kind of binary tree with the following properties:
1. Each leaf is an operand. Examples: a, b, c, 6, 100
2. The root and internal nodes are operators. Examples: +, -, *, /, ^
3. Subtrees are subexpressions with the root being an operator.

Traversal Techniques:

There are 3 standard traversal techniques to represent the 3 different expression formats.

- **Inorder Traversal:**
    - We can produce an infix expression by recursively printing out
    - the left expression,
    - the root, and
    - the right expression.

- **Postorder Traversal:**
- The postfix expression can be evaluated by recursively printing out
- the left expression,
- the right expression and
- then the root
-
- **Preorder Traversal:**

1) We can also evaluate prefix expression by recursively printing out:
2) the root,
3) the left expressoion and
4) the right expression.

Algorithm:
Construction of Expression Tree:

1. Read the Prefix expression in reverse order (from right to left)
2. If the symbol is an operand, create one node tree and then push it onto the Stack
3. If the symbol is an operator, then pop two pointers from the Stack namely T1 &T2 and form a new tree with root as operator T1 & T2 as a left and right child. A pointer to this new tree is pushed onto the stack
4. Repeat the above steps until end of Prefix expression.

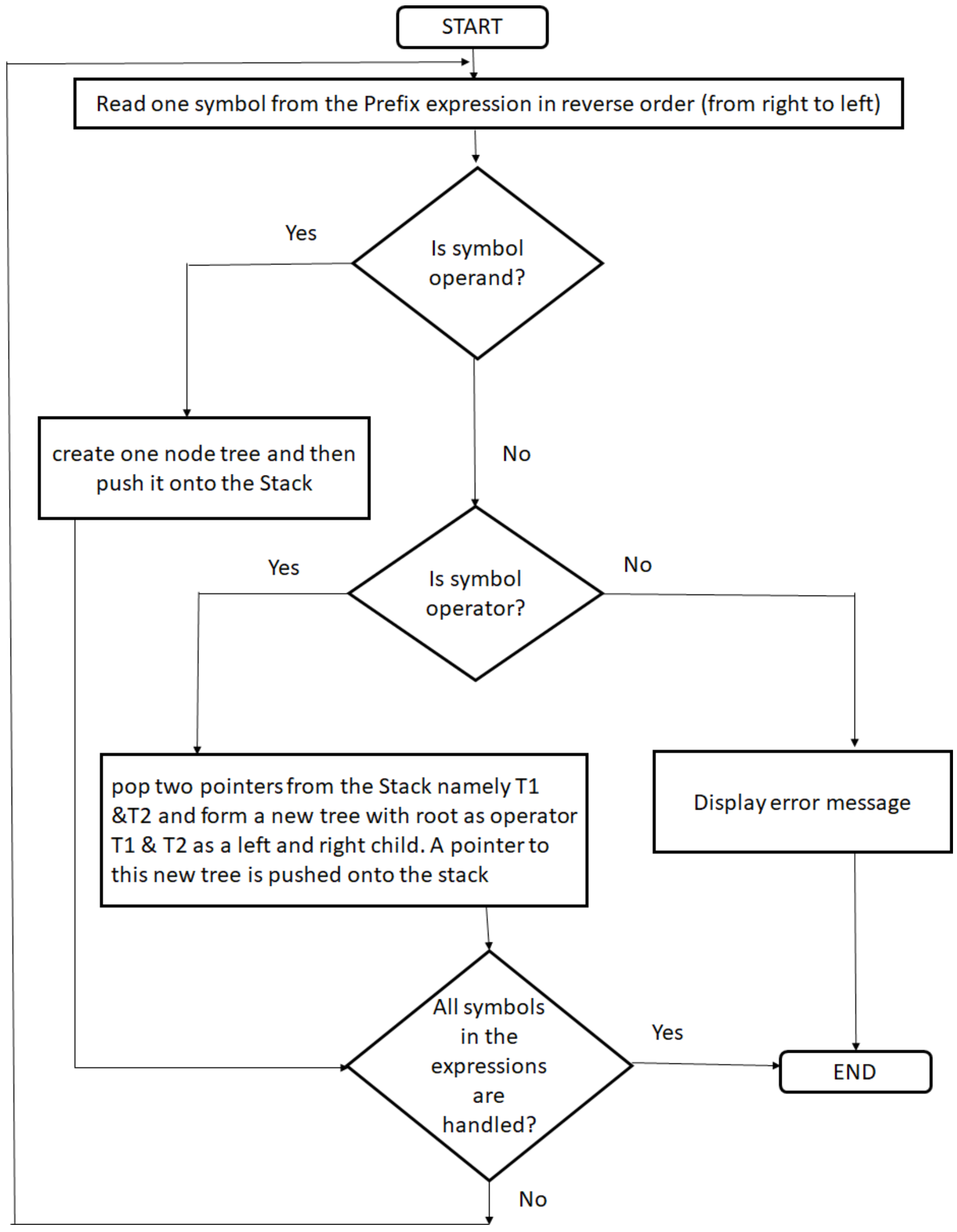
Post order Traversal –Non recursive:

Using two Stack S1 & S2

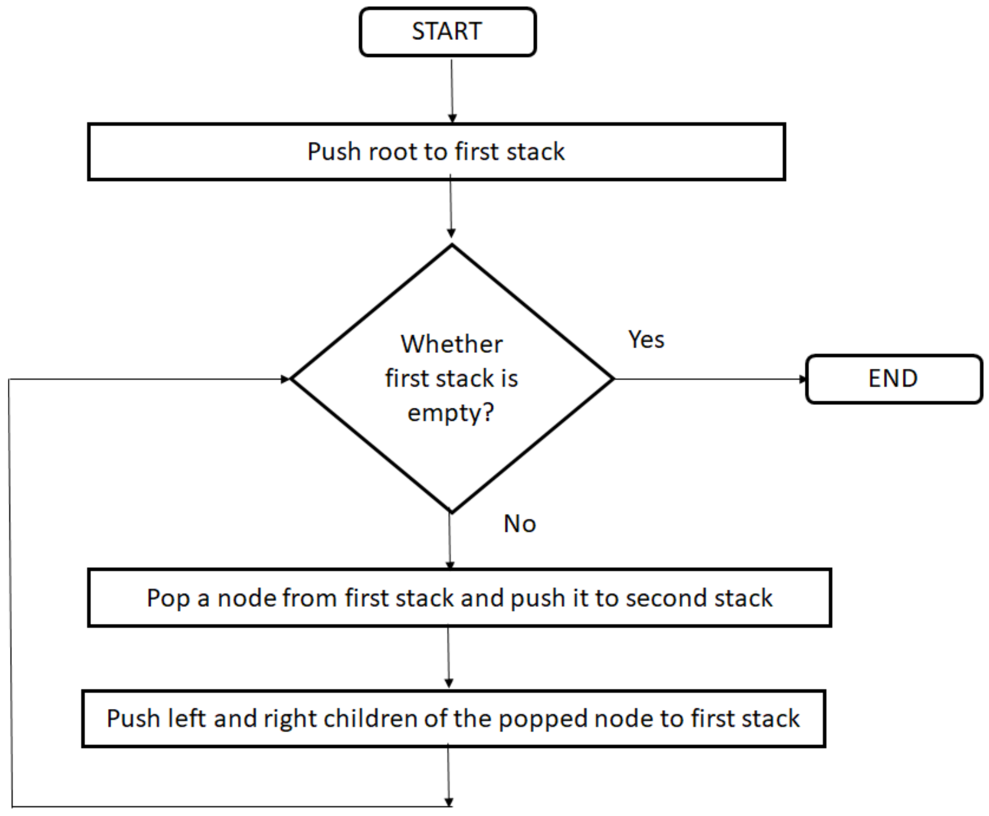
```
1. Push root to first stack.
2. Loop while first stack is not empty
   2.1 Pop a node from first stack and push it to second stack
   2.2 Push left and right children of the popped node to first stack
3. Print contents of second stack
```

Flowchart:

- **Construction of Expression Tree:**


START
Read one symbol from the Prefix expression in reverse order (from right to left)
Is symbol operand?
Yes
No
create one node tree and then push it onto the Stack
Is symbol operator?
Yes
No
pop two pointers from the Stack namely T1 &T2 and form a new tree with root as operator T1 & T2 as a left and right child. A pointer to this new tree is pushed onto the stack
Display error message
All symbols in the expressions are handled?
Yes
No
END

- **Post order Traversal –Non recursive:**


START
Push root to first stack
Whether first stack is empty?
Yes
END
No
Pop a node from first stack and push it to second stack
Push left and right children of the popped node to first stack

Test Cases:

Sr. No.	**Test ID**	**Steps**	**Input**	**Expected Result**	**Actual Result**	**Status (Pass/ Fail)**
1	ID1	Given an prefix expression the tree is constructed	Prefix expression	Respective tree	Tree constructed for the expression	Pass
2	ID2	Give Incomplete prefix expression	Incomplete prefix expression	Display error message on screen	Display error message on screen	Pass
3	ID3	Postorder traversal of the expression tree	Expression tree	Postorder traversal sequence of the expression tree	Postorder traversal sequence of the expression tree	Pass

FAQs:

Q1. What is expression tree?
Ans: Expression tree is a tree in which operands are present at leaf nodes and operators are present at internal nodes.

Q2. At which position operators present in Expression tree?
Ans: At internal nodes

Q3. At which position operands present in Expression tree?
Ans: At leaf node or external node

Q4. What is output for inorder traversal on Expression Tree?
Ans: Infix expression

Q5. What are infix , postfix and prefix expression?
Ans: Infix expression : operand1 operator operand2
Postfix expression : operand1 operand2 operator
Prefix expression: operatoroperand1 operator

Q6. What is use of stack in program?
Ans: The stack is used to push pointer to newly created node of expression tree.

Q7. Can STL used instead of user defined stack in program?
Ans: Yes We can use STL stack

Q8. What are applications of trees?
Ans: Expression tree, Decision tree, Gaming tree

Q9. What are applications for stack?
Ans:Expression conversion and evaluation
Decimal to binary conversion
Reversing a string

Q10. What is difference between tree and graph?
Ans: Graph has cycle and they are connected i.e we can move from any vertex to any vertex. Tree has no cycle.

GROUP: C

ASSIGNMENT NO: 6

Title: Study types of representation of the Graph

Objective: To implement Adjacency Matrix representation of graph

Problem Statement: 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.

Outcome:

Summarize the basic concepts and operations on Graph data structure & apply it for solving real life problems

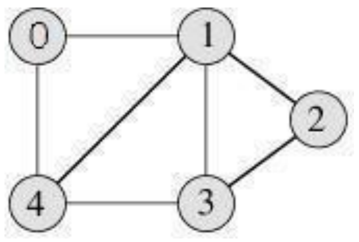
Theory:

Graph is a data structure that consists of following two components:

1. A finite set of vertices also called as nodes.
2. A finite set of ordered pair of the form (u, v) called as edge.

The pair is ordered because (u, v) is not same as (v, u) in case of directed graph(di-graph). The pair of form (u, v) indicates that there is an edge from vertex u to vertex v. The edges may contain weight/value/cost.

Following is an example undirected graph with 5 vertices.



Following two are the most commonly used representations of graph.

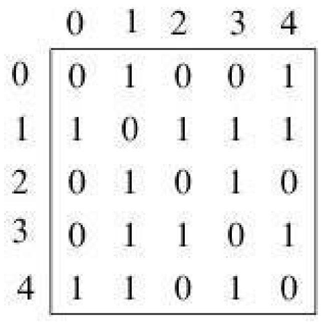
1. Adjacency Matrix
2. Adjacency List

There are other representations also like, Incidence Matrix and Incidence List. The choice of the graph representation is situation specific. It totally depends on the type of operations to be performed and ease of use.

Adjacency Matrix:

Adjacency Matrix is a 2D array of size V x V where V is the number of vertices in a graph. Let the 2D array be adj[][], a slot adj[i][j] = 1 indicates that there is an edge from vertex i to vertex j. Adjacency matrix for undirected graph is always symmetric. Adjacency Matrix is also used to represent weighted graphs. If adj[i][j] = w, then there is an edge from vertex i to vertex j with weight w.

The adjacency matrix for the above example graph is:



$$\begin{array}{c|ccccc|}
 & 0 & 1 & 2 & 3 & 4 \\
\hline
0 & 0 & 1 & 0 & 0 & 1 \\
1 & 1 & 0 & 1 & 1 & 1 \\
2 & 0 & 1 & 0 & 1 & 0 \\
3 & 0 & 1 & 1 & 0 & 1 \\
4 & 1 & 1 & 0 & 1 & 0 \\
\hline
\end{array}$$

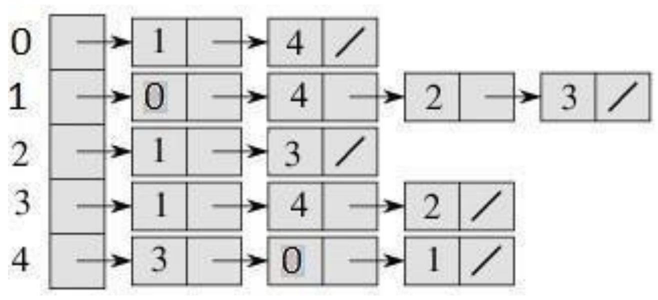
Adjacency Matrix Representation of the above graph

Pros: Representation is easier to implement and follow. Removing an edge takes O(1) time. Queries like whether there is an edge from vertex ‘u’ to vertex ‘v’ are efficient and can be done O(1).

Cons: Consumes more space O(V^2). Even if the graph is sparse(contains less number of edges), it consumes the same space. Adding a vertex is O(V^2) time.

Adjacency List:

An array of linked lists is used. Size of the array is equal to number of vertices. Let the array be array[ ]. An entry array[i] represents the linked list of vertices adjacent to the i^{th} vertex. This representation can also be used to represent a weighted graph. The weights of edges can be stored in nodes of linked lists. Following is adjacency list representation of the above graph.



Adjacency List Representation of the above Graph

Applications:

Graphs are used to represent many real life applications: Graphs are used to represent networks. The networks may include paths in a city or telephone network or circuit network. Graphs are also used in social networks like linkedIn, facebook. For example, in facebook, each person is represented with a vertex(or node). Each node is a structure and contains information like person id, name, gender and locale.

Algorithm:

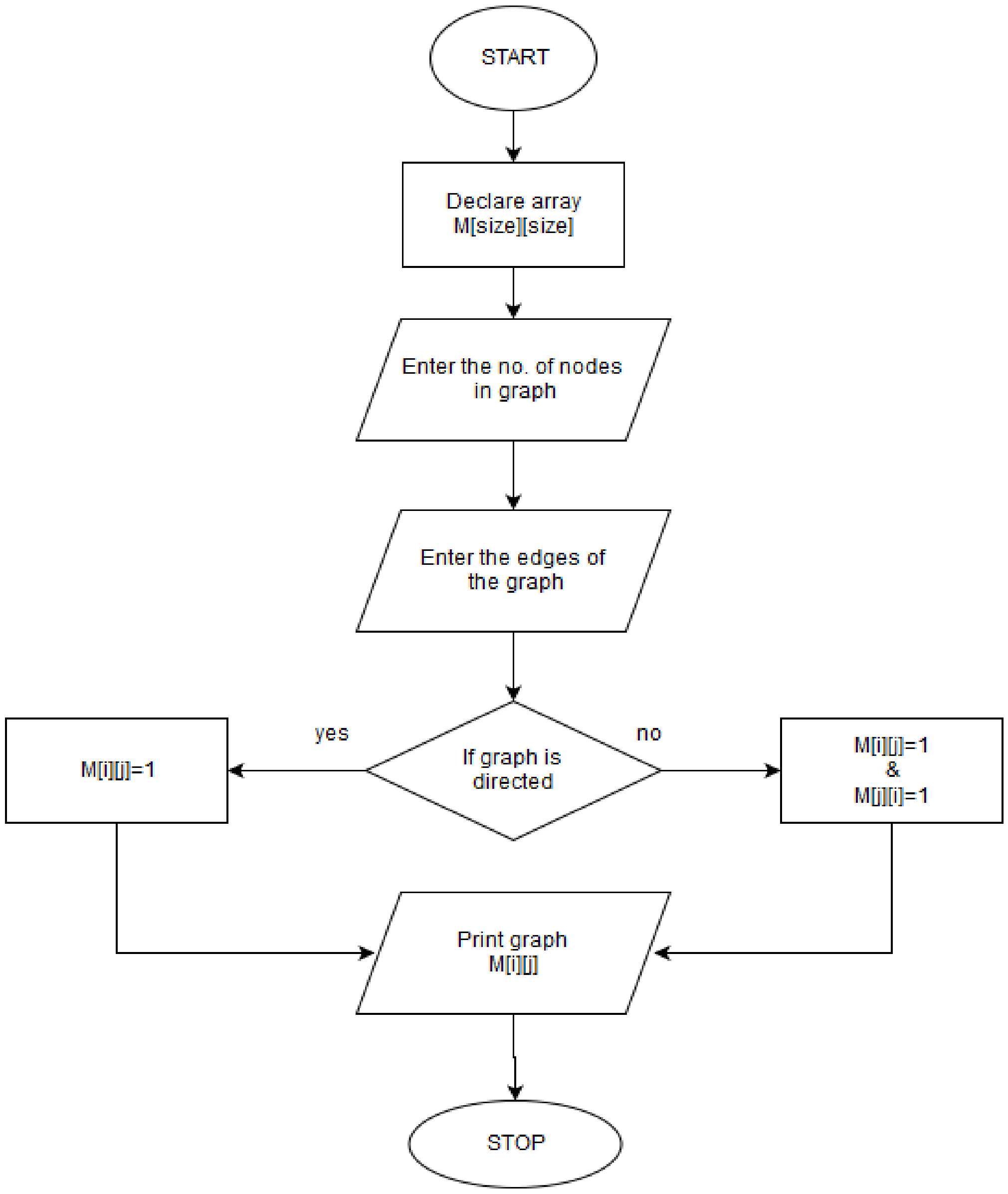
Graph creation using Adjacency Matrix

1. Declare an array of M[size][size] which will store the graph.
2. Enter how many nodes you want in a graph.
3. Enter the edges of the graph by two vertices each, say V_i , V_j indicates some edge
4. If the graph is directed set M[i][j]=1. If graph is undirected set M[i][j] =1 and M[j][i] =1 as well.
5. When all the edges of the desired graph is entered print the graph M[i][j].

Graph creation using Adjacency list

1. Declare node structure for creating adjacency list.
2. Initialize an array of nodes. This array will act as head nodes. The index of head[ ] will be the starting vertex.
3. The create function will create the adjacency list.

Flow Chart:


START
Declare array M[size][size]
Enter the no. of nodes in graph
Enter the edges of the graph
If graph is directed
yes
no
M[i][j]=1
M[i][j]=1 & M[j][i]=1
Print graph M[i][j]
STOP

Test cases:

Sr. No.	Test ID	Steps	Input	Expected Result	Actual Result	Status (Pass/ Fail)
1	ID1	Enter the matrix values	All 0’s	Graph is not connected	If for any vertex there shows a adjacent vertex	Fail

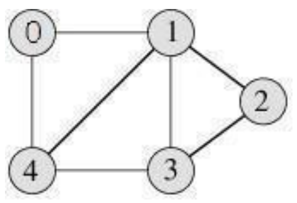
2	ID2	Enter the matrix values	All 0’s	Graph is not connected	No Adjacent vertex shown	Pass
3	ID3	Enter the matrix values	Some non zero values and remaining zero’s	Some edges are present	No adjacent vertex present	Fail
4	ID4	Print the graph using DFS	{0,1,1,0,1,0,0,1, 1,0,0,1,0,1,1,0}	1,2,4,3	1,2,4,3 or 1,3,4,2	Pass
5	ID5	Print the graph using DFS	{0,1,1,0,1,0,0,1, 1,0,0,1,0,1,1,0}	1,2,4,3	1,3,2,4	Fail

Conclusion: Graph representation using adjacency matrix is implemented successfully**.**
FAQs :

Q1. What are the advantages of adjacency list representation over adjacency matrix representation of a graph?
Ans : In adjacency list representation, space is saved for sparse graphs. DFS and BSF can be done in O(V + E) time for adjacency list representation. These operations take O(V^2) time in adjacency matrix representation. Here is V and E are number of vertices and edges respectively. Adding a vertex in adjacency list representation is easier than adjacency matrix representation.

Q2. Define graph
Ans : A graph G, consists of 2 sets V & E. V is a finite non-empty set of vertices. E is a set of pairs of vertices, these pairs are called edges.

Q3. What is adjacency matrix?
Adjacency Matrix is a 2D array of size V x V where V is the number of vertices in a graph. Let the 2D array be adj[][], a slot adj[i][j] = 1 indicates that there is an edge from vertex i to vertex j.

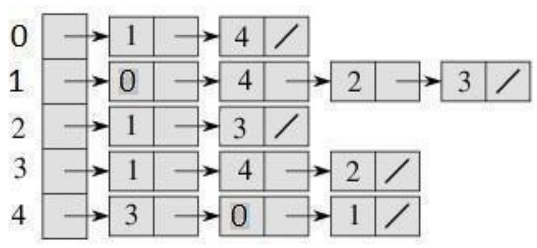


$$\begin{array}{c|ccccc|} & 0 & 1 & 2 & 3 & 4 \\ \hline 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 & 1 & 1 \\ 2 & 0 & 1 & 0 & 1 & 0 \\ 3 & 0 & 1 & 1 & 0 & 1 \\ 4 & 1 & 1 & 0 & 1 & 0 \\ \hline \end{array}$$

Print the graph using DFS
{0,1,1,0,1,0,0,1,1,0,0,1,0,1,1,0}
Adjacency matrix for undirected graph is always symmetric. Adjacency Matrix is also used to represent weighted graphs. If adj[i][j] = w, then there is an edge from vertex i to vertex j with weight w adjacency matrix for above graph

Q4. What is adjacency list?

An array of linked lists is used. Size of the array is equal to number of vertices. Let the array be array[]. An entry array[i] represents the linked list of vertices adjacent to the ith vertex. This representation can also be used to represent a weighted graph. The weights of edges can be stored in nodes of linked lists. Following is adjacency list representation of the above graph.



Q5. For an undirected graph with n vertices and e edges, the sum of the degree of each vertex is equal to ______

Ans : 2e

Q6. The maximum degree of any vertex in a simple graph with n vertices is _____

Ans : n–1

Q7. What are the advantages and disadvantages of Adjacency matrix

Ans :

- fast to tell whether edge exists between any two vertices *i* and *j* (and to get its weight)
- consumes a lot of memory on sparse graphs (ones with few edges) redundant information for undirected graphs

Q8. What are the advantages and disadvantages of Adjacency list

Ans : advantages

- new nodes can be added easily
- new nodes can be connected with existing nodes easily
- "who are my neighbors" easily
  answered disadvantages:
- determining whether an edge exists between two nodes: O(average degree)

Q9. Which algorithm can be used to most efficiently determine the presence of a cycle in a given graph ?

Ans : Depth First Search

Q10. Given two vertices in a graph s and t, which of the two traversals (BFS and DFS) can be used to find if there is path from s to t?

Ans :BFS and DFS

ASSIGNMENT NO: 7

Title : Study of minimum spanning tree

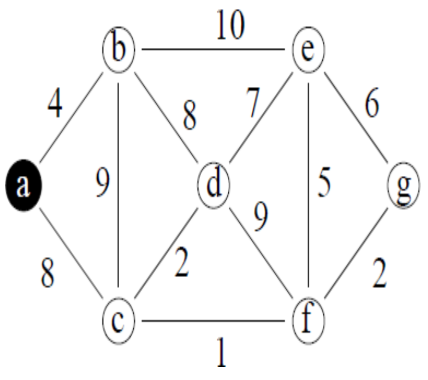
Objective : To implement Minimum Spanning Tree

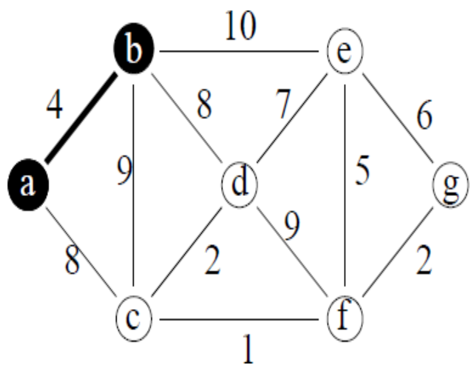
Problem Statement : You have a business with several offices; you want to lease phone lines to connect them up with each other; and the phone company charges different amounts of money to connect different pairs of cities. You want a set of lines that connects all your offices with a minimum total cost.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

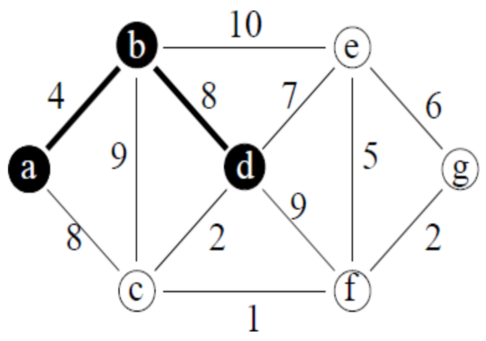
Outcomes : Explain the concepts of Hashing and apply it to solve searching problems

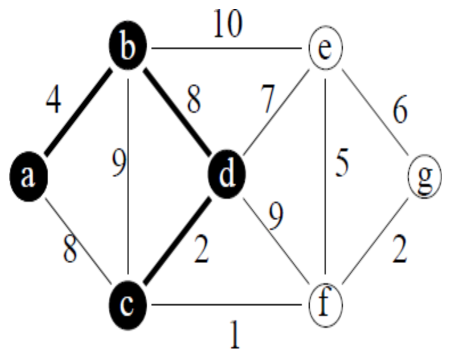
Theory :

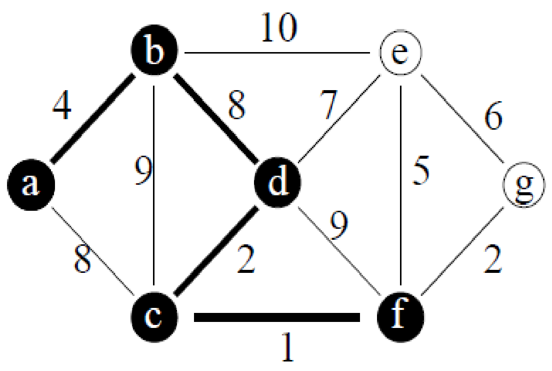
Spanning Trees: A sub-graph T of a undirected graph G=(V,E) is a spanning tree of G if it is a tree and contains every vertex of G. Minimum Spanning Tree in an undirected connected weighted graph is a spanning tree of minimum weight (among all spanning trees).
Prim's algorithm is a greedy algorithm that finds a minimum spanning tree for a connected weighted undirected graph. This means it finds a subset of the edges that forms a tree that includes every vertex, where the total weight of all the edges in the tree is minimized. Start form any arbitrary **vertex** Find the edge that has minimum weight form **all** known vertices Stop when the tree covers all vertices









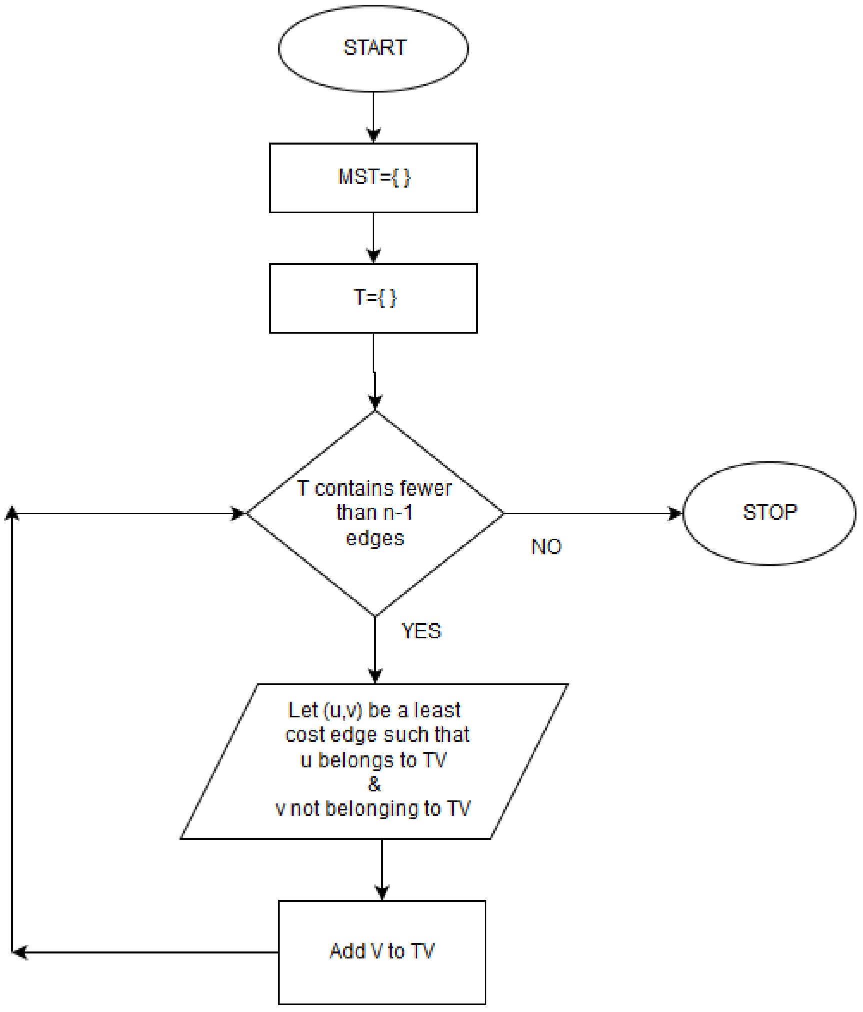


Algortithm:

Prim's Algorithm:

```
MST= {0}                    // start with vertex 0
For (T=0; T contains fewer than n-1 edges; add(u,v) to T)
        {
Let (u,v) be a least cost edge such that u ϵ TV & v not belonging to TV;
Add V to TV
        }
```

Flow Chart :


START
MST={ }
T={ }
T contains fewer than n-1 edges
NO
STOP
YES
Let (u,v) be a least cost edge such that u belongs to TV & v not belonging to TV
Add V to TV

Test Cases:

Sr. No.	Test ID	Steps	Input	Expected Result	Actual Result	Status (Pass/ Fail)
1	ID1	Take no. of vertices and edges. for e.g. 5 and 8	Enter two vertices and the cost of edge between them	An adjacency matrix representation of graph	An adjacency matrix representation of graph	Pass
2	ID2	Vertices with no	Cost[i][j]=infin	Cost between	Some are not	Fail

		edge between them are assigned some value infinity	ity	all the vertices having no edge between them should be made infinity	made	
3	ID3	Enter the cost of edges present	1 2 10 1 3 5 1 4 8 2 4 3 2 5 7 3 4 4 3 5 2 4 5 1	1 3 5 4 2	1 3 5 4 2	Pass
4	ID4	Enter the cost of edges present	1 2 10 1 3 5 1 4 8 2 4 3 2 5 7 3 4 4 3 5 2 4 5 1	1 3 5 4 2	1 3 4 2 5	Fail
5	ID5	While finding the MST	if all vertices are not visited	Visited [1 to 5]=1	If Visited[5]=0	Fail

Conclusion: Minimum Spanning Tree assignment has implemented successfully.

FAQs:

Q 1.What is MST?
Ans: MST is minimum spanning tree. It is a spanning tree with minimum weight. Spanning tree is tree that contains all vertices of a given graph and contains subset of edges present in given graph.

Q 2. Which algorithms are available for finding MST for given graph?
Ans: Prim's algorithm
Kruskal's algorithm

Q 3.Which algorithm you have used in program?
Ans: Prim's algorithm is used to find MST of given graph.

Q 4. What is constructor, types of constructor
Ans: Constructor is a special member function used to initialize the class objects.
Types
Default constructor: Constructor with no arguments.
Parameterized constructor: Constructor taking arguments
Copy constructor: Used to initialize the object from previously created objects.

Q 5. Have you used constructor is assignment? If Yes, what is its type?
Ans: Yes . Default constructor.

Q 6. Why visited array is used in program?
Ans: Visited array is used to keep track of vertices which are already visited and not to be visited again in algorithm.

Q 7. How many edges does a minimum spanning tree has?
Ans: A minimum spanning tree has (V 1) edges where V is the number of vertices in the given graph.

Q 8. How does Prim's Algorithm Work?
Ans : The idea behind Prim's algorithm is simple, a spanning tree means all vertices must be connected. So the two disjoint subsets of vertices must be connected to make a Spanning Tree. And they must be connected with the minimum weight edge to make it a Minimum Spanning Tree.
Q 9. Does the minimum spanning tree of a graph give the shortest distance between any 2 specified nodes?
Ans: No. The Minimal spanning tree assures that the total weight of the tree is kept at its minimum. But it doesn't mean that the distance between any two nodes involved in the minimum-spanning tree is minimum.

Q 10. What is a spanning Tree?
Ans: A spanning tree is a tree associated with a network. All the nodes of the graph appear on the tree once. A minimum spanning tree is a spanning tree organized so that the total edge weight between nodes is minimized.

Group: D
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:

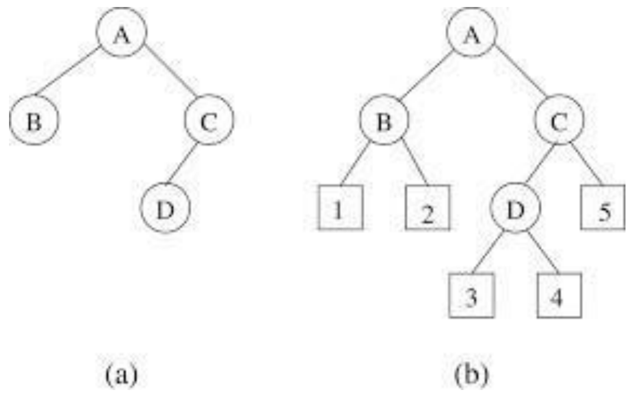
✓ Define class for Extended binary search tree using Object Oriented features.
✓ Analyze working of functions.

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:



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
for i = 0 to n do
C (i, i) ←0
W (i, i) ←q(i)
for m = 0 to n do
for i = 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)
f i = k then
*build a new leaf node N" labeled (i, i)
else
*N" ←CONSTRUCT(R, i, k)
*N" is the left child of node
Nif 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.

Input:

- No. of Element
- key values
- Key Probability

Output: Create binary search tree having optimal searching cost.

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

OUTCOME:Upon completion Students will be able to:
ELO1: Learn object oriented Programming features.
ELO2: Understand & implement extended binary search tree.

FAQS:

Q 1.What is BST?
Ans: It is binary search tree. BST is binary tree in which left sub-tree contains data smaller than node and right sub-tree contains data greater than node.

Q 2. What is difference in Binary tree and BST?
Ans: Binary is tree in which number of child nodes is restricted to 0, 1, 2
BST is binary tree in which left sub-tree contains data smaller than node and right sub-tree contains data greater than node.

Q 3. What is friend class?
Ans: When we create a friend class then all the member functions of the friend class also become the friend of the other class.

Q4. What is constructor , types of constructor
Ans: Constructor is a special member function used to initialize the class objects.
Types
Default constructor: Constructor with no arguments.
Parameterized constructor: Constructor taking arguments
Copy constructor: Used to initialize the object from previously created objects.

Q5. Have you used constructor is assignment?
Ans: Yes . Parameterized constructor is used in assignment

Q6. Which value is returned by strcmp( ) function?

Ans: Strcmp( ) function returns zero (0) if both strings are equal.

Strcmp( ) function returns value greater than zero (0) if first string is greater than (means alphabetical order) second.

Strcmp( ) function returns value less than zero (0) if first string is less than (means alphabetical order) second.

Q7. What is use of strcpy function()?

Ans: It is used to copy one string into another string.

Q8. What are traversals done on tree to display data?

Ans: Inorder(LVR), preorder(VLR) and post order(LRV) traversal

Q9. Inorder traversal will display the data of dictionary in which order/sequence?

Ans: Inorder traversal will display the data of dictionary in ascending order.

Q10. Out of two classes in program , which class object is created and which class pointer is created?

Ans: Out of two classes in program, tree class object is created and node class pointer is created

ASSIGNMENT NO: 9

Title: Study of AVL trees

Objective: To implement AVL Trees

Problem Statement: 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**.**

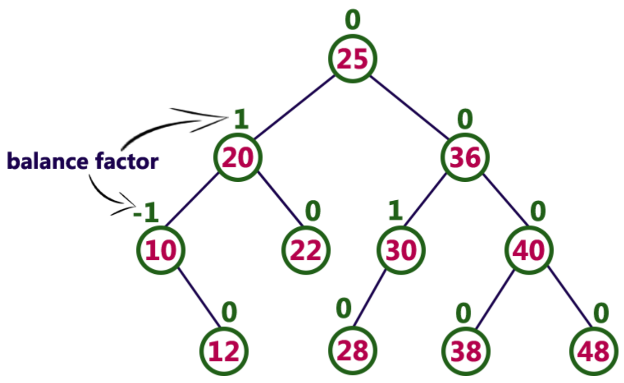
Outcome: Describe the concept of symbol tables with OBST and AVL trees

Theory:

An **AVL tree** (Adelson-Velsky and Landis' tree, named after the inventors) is a self-balancing binary search tree. In an AVL tree, the heights of the two child subtrees of any node differ by at most one;
Balance factor = heightOfLeftSubtree – heightOfRightSubtree

if at any time they differ by more than one, rebalancing is done to restore this property. Lookup, insertion, and deletion all take O(log *n*) time in both the average and worst cases, where *n* is the number of nodes in the tree prior to the operation. Insertions and deletions may require the tree to be rebalanced by one or more tree rotations.

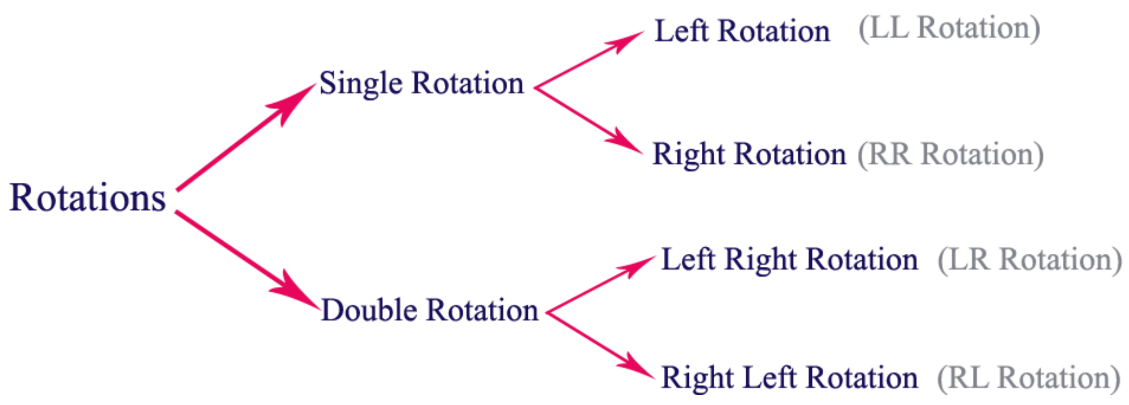
Example AVL tree



AVL Tree Rotations:

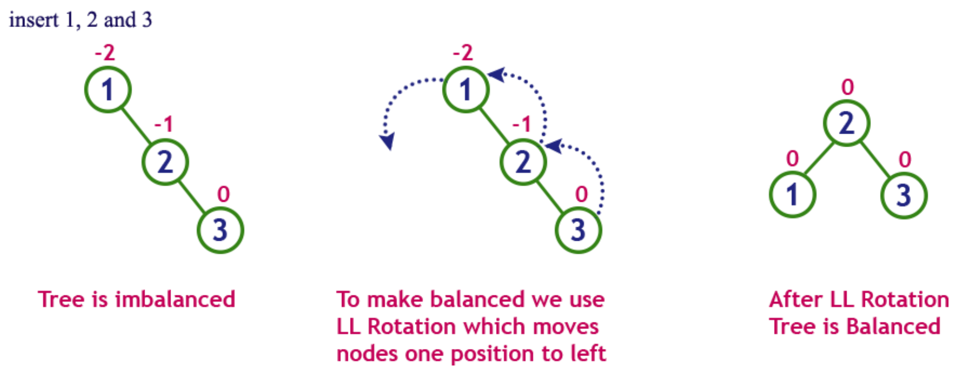
In AVL tree, after performing every operation like insertion and deletion we need to check the balance factor of every node in the tree. If every node satisfies the balance factor condition then we conclude the operation otherwise we must make it balanced. We use rotation operations to make the tree balanced whenever the tree is becoming imbalanced due to any operation.
Rotation operations are used to make a tree balanced. Rotation is the process of moving the nodes to either left or right to make tree balanced.

There are **four** rotations and they are classified into **two** types.


Rotations
Single Rotation
Double Rotation
Left Rotation (LL Rotation)
Right Rotation (RR Rotation)
Left Right Rotation (LR Rotation)
Right Left Rotation (RL Rotation)

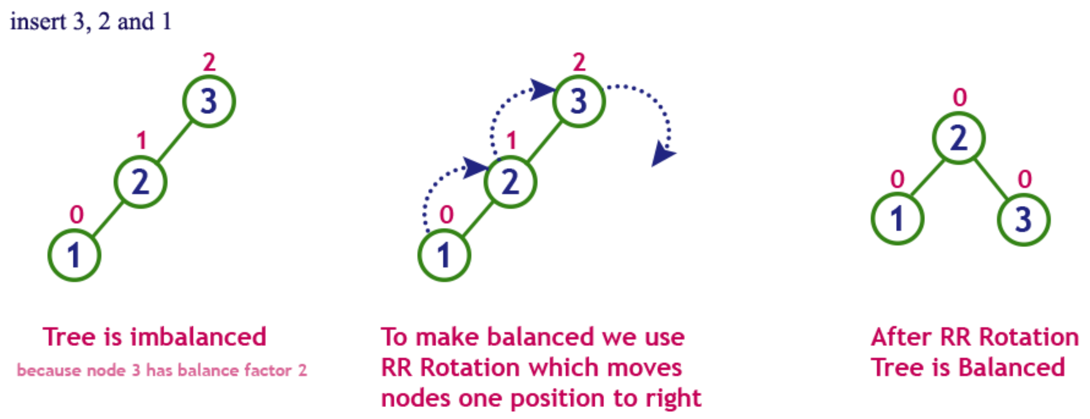
Single Left Rotation (LL Rotation)

In LL Rotation every node moves one position to left from the current position.


insert 1, 2 and 3
-2
1
-1
2
0
3
Tree is imbalanced
To make balanced we use LL Rotation which moves nodes one position to left
After LL Rotation Tree is Balanced

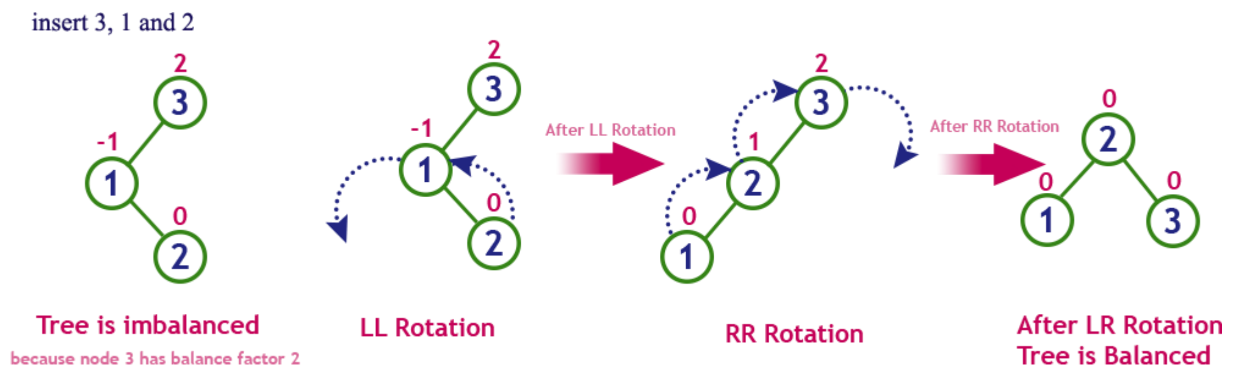
Single Right Rotation (RR Rotation)

In RR Rotation every node moves one position to right from the current position.


insert 3, 2 and 1
Tree is imbalanced
because node 3 has balance factor 2
To make balanced we use RR Rotation which moves nodes one position to right
After RR Rotation Tree is Balanced

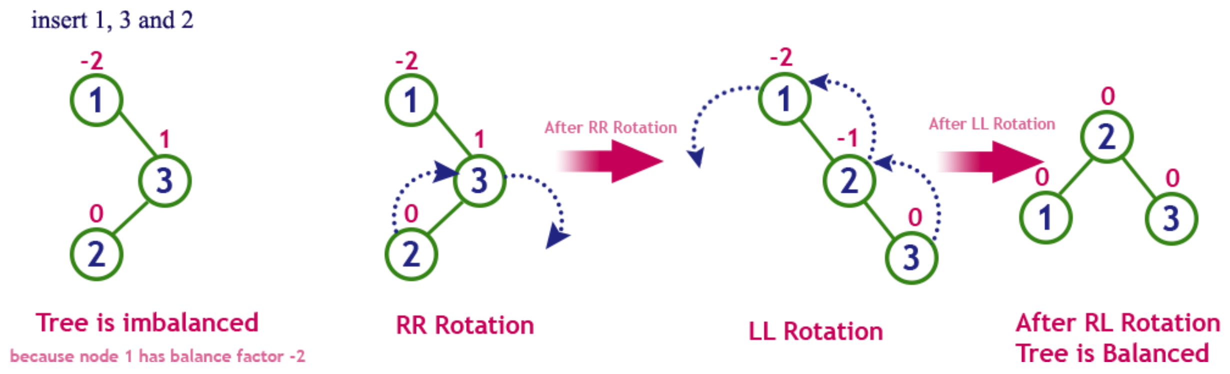
Left Right Rotation (LR Rotation)

The LR Rotation is combination of single left rotation followed by single right rotation. In LR Rotation, first every node moves one position to left then one position to right from the current position.



Right Left Rotation (RL Rotation)

The RL Rotation is combination of single right rotation followed by single left rotation. In RL Rotation, first every node moves one position to right then one position to left from the current position.



The following operations are performed on an AVL tree...

1. Search
2. Insertion
3. Deletion

Algorithm :

Insertion Operation in AVL Tree

In an AVL tree, the insertion operation is performed with **O(log n)** time complexity. In AVL Tree, new node is always inserted as a leaf node. The insertion operation is performed as follows...

Step 1: Insert the new element into the tree using Binary Search Tree insertion logic.

Step 2: After insertion, check the **Balance Factor** of every node.

Step 3: If the **Balance Factor** of every node is **0 or 1 or -1** then go for next operation.

Step 4: If the **Balance Factor** of any node is other than **0 or 1 or -1** then tree is said to be imbalanced. Then perform the suitable **Rotation** to make it balanced. And go for next operation.

Search Operation in AVL Tree

In an AVL tree, the search operation is performed with **O(log n)** time complexity. The search operation is performed similar to Binary search tree search operation. We use the following steps to search an element in AVL tree…

Step 1: Read the search element from the user

Step 2: Compare, the search element with the value of root node in the tree.

Step 3: If both are matching, then display "Given node found!!!" and terminate the function

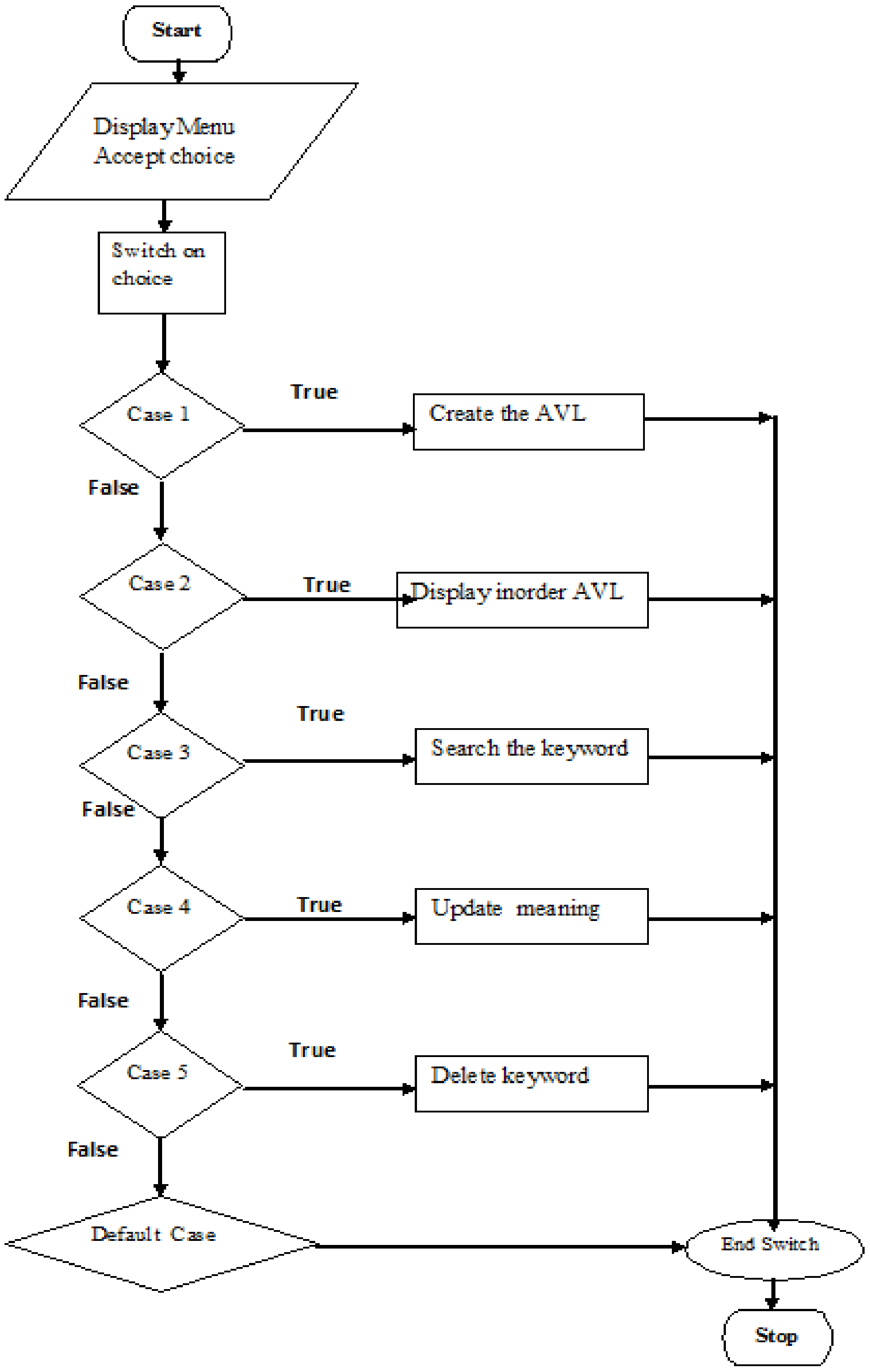
Step 4: If both are not matching, then check whether search element is smaller or larger than that node value.

Step 5: If search element is smaller, then continue the search process in left subtree.
Step 6: If search element is larger, then continue the search process in right subtree.
Step 7: Repeat the same until we found exact element or we completed with a leaf node
Step 8: If we reach to the node with search value, then display "Element is found" and terminate the function.
Step 9: If we reach to a leaf node and it is also not matching, then display "Element not found" and terminate the function.

Deletion Operation in AVL Tree

In an AVL Tree, the deletion operation is similar to deletion operation in BST. But after every deletion operation we need to check with the Balance Factor condition. If the tree is balanced after deletion then go for next operation otherwise perform the suitable rotation to make the tree Balanced.

Flowchart:


Start
Display Menu
Accept choice
Switch on choice
Case 1
True
Create the AVL
False
Case 2
True
Display inorder AVL
False
Case 3
True
Search the keyword
False
Case 4
True
Update meaning
False
Case 5
True
Delete keyword
False
Default Case
End Switch
Stop

Test Cases:

Sr. NO	Test ID	Steps	Input	Expected Result	Actual Result	Statu s (Pass/ Fail)
1	ID1	Enter the data and display AVL tree	Happy Good Mood Zoo Wild Animals Cat Pet Animal	Sorted list of data to be displayed	Cat Pet Animal Happy Good Mood Zoo Wild Animals	Pass
2	ID2	Enter the data and search present data	Happy Good Mood Zoo Wild Animals Cat Pet Animal	Required data is preseTextnt	Message displayed data is not present	Fail
3	ID3	Enter the data and search non existing data	Happy Good Mood Zoo Wild Animals Cat Pet Animal	Required data is absent	Message displayed data is present	Fail
4	ID4	Search in empty AVL	No Input	AVL Tree is empty	Message displayed AVL Tree is empty	Pass
5	ID5	Enter the data , display AVL tree and delete node	Happy Good Mood Zoo Wild Animals Cat Pet Animal	Happy Good Mood Zoo Wild Animals	Cat Pet Animal Happy Good Mood Zoo Wild Animals	Fail

Conclusion: Dictionary assignment has implemented successfully using binary search tree.

FAQs:

Q 1. What is AVL tree?

Ans: An AVL tree is a self-balancing binary search tree. In an AVL tree, the heights of the two child subtrees of any node differ by at most one; if at any time they differ by more than one, re-balancing is done to restore this property.

Q 2. What does AVL stand for?

Ans: AVL stands for Adelson-Velsky and Landis' tree , names of the inventors

Q 3. Is AVL a BST?

Ans: Yes. AVL is binary search tree.

Q 4. What is balance factor in AVL?
Ans: Balance factor of AVL can be either -1, 0,1

Q 5. What are single rotations ?
Ans: LL & RR

Q 5. What are double rotations ?
Ans: LR & RL

Q 6. What is time complexity of using AVL tree?
Ans: For insert, search and delete operations on AVL tree, time complexity is O(log n) where n is the number of nodes in the tree.

Q 7. What is height of AVL tree?
Ans: The height of an AVL tree is always O(logn) where n is the number of nodes in the tree.

Q 8. How to calculate Balance factor?
Ans: BalanceFactor(N) := –Height(LeftSubtree(N)) + Height(RightSubtree(N))

Q 9. What are applications of A VL Trees
Ans: AVL trees are applied in the following situations:
There are few insertion and deletion operations , Short search time is needed Input data is sorted or nearly sorted

Q 10. Compare AVL tree with Red-Black Tree.
Ans: AVL trees are more rigidly balanced and hence provide faster look-ups. Thus for a look-up intensive task use an AVL tree. For an insert intensive task, use a Red-Black tree.

GROUP:D
ASSIGNMENT NO:10

Title: Study Heap data structure

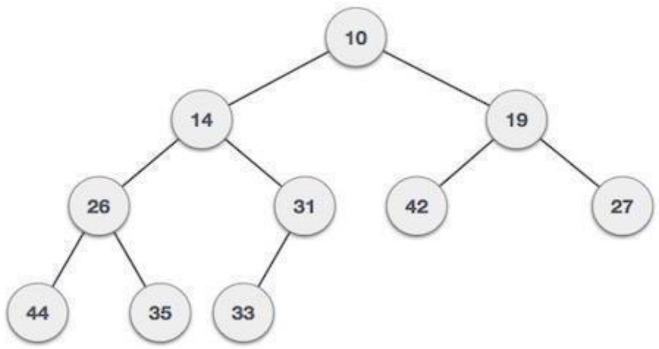
Objective: To understand Max heap and Min heap**.**

Problem Statement: 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.

Outcome: Demonstrate and apply knowledge of Indexing and Multiway Trees

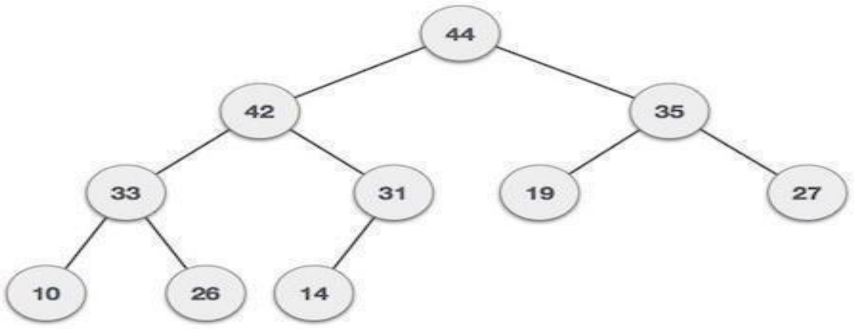
Theory:
Heap: Heap is a special case of balanced binary tree data structure where the root-node key is compared with its children and arranged accordingly. If **α** has child node **β** then −
key(α) ≥ key(β)
As the value of parent is greater than that of child, this property generates **Max Heap**. Based on this criteria, a heap can be of two types −

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

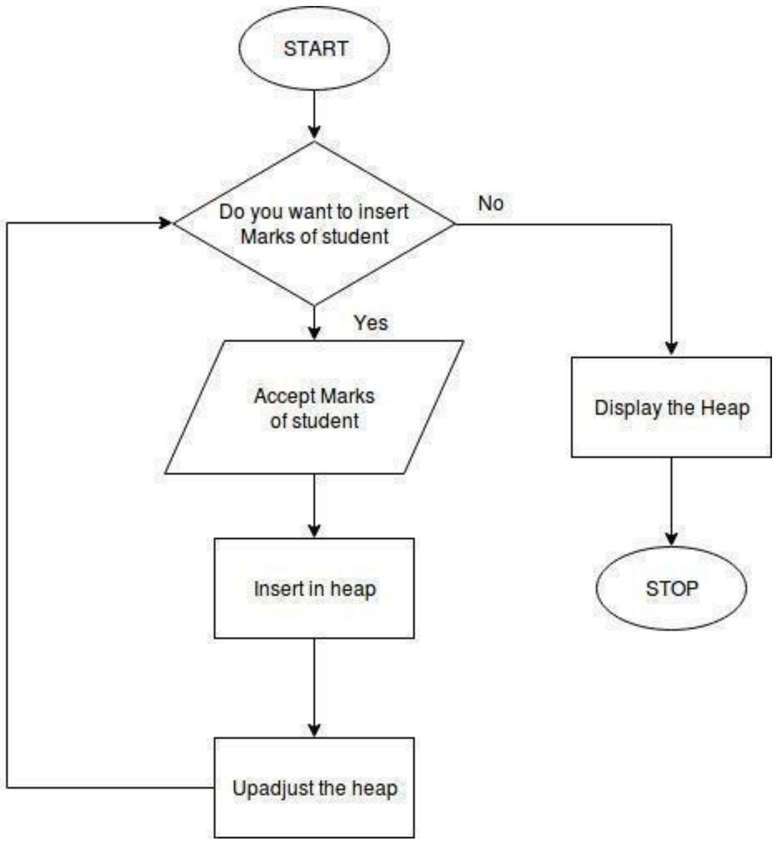


Max-Heap − Where the value of the root node is greater than or equal to either of its children.
Implementation
Algorithm :

1. Accept the marks for student.
2. Insert the marks in heap.
3. Up-adjust the heap.
4. Repeat the steps 1 to 4 if marks of more students to be entered.
5. Display the maximum marks.


44
42
35
33
31
19
27
10
26
14

Flowchart:


START
Do you want to insert Marks of student
No
Yes
Accept Marks of student
Display the Heap
Insert in heap
STOP
Upadjust the heap

Test Cases:

Sr. NO	Test ID	Steps	Input	Expected Result	Actual Result	Statu s (Pass/ Fail)
1	ID1	Construct a Max heap and display maximum marks.	10 20 30	Maximum marks to be displayed 30	Maximum marks is displayed as 30	Pass
2	ID2	Construct a Min heap and display minimum marks.	40 20 10	Minimum marks to be displayed 10	Minimum marks is displayed as 10	Pass
3	ID3	Construct a Max heap and display maximum marks.	50 10 90	Maximum marks to be displayed 90	Maximum marks is displayed as 500	Fail
4	ID4	Construct a Min heap and display minimum marks.	70 10 30	Minimum marks to be displayed 10	Minimum marks is displayed as 30	Fail

Conclusion: Max Heap and Min heap are implemented Successfully**.**
FAQs:
Q 1. What is Heap?
Ans: Heap is complete binary tree

Q 2. How data is stored in Heap?
Ans: For ith node, parent(i) at i/2 , left child(i) at 2i & right child(i) at 2i+1

Q 3. What are operations on Heap?
Ans: Create heap() , Insert data(), Upadjust(), DeleteMax(), DownAdjust(), FindMin(), FindMax(), Display()

Q 4. What is Max Heap?
Ans: A **max-heap** is a complete binary tree in which the value in each internal node is greater than or equal to the values in the children of that node.

Q 5. What is Min Heap?

Ans: A **min-heap** is a complete binary tree in which the value in each internal node is smaller than or equal to the values in the children of that node.

Q 6. At Which position maximum data is present in Max heap?

Ans: At Root position.

Q 7. At Which position minimum data is present in Max heap?

Ans: At Root position.

Q 8. What are applications of Heap?

Ans: Heapsort

Selection algorithms: A heap allows access to the min or max element in constant time, and other selections (such as median or kth-element) can be done in sub-linear time on data that is in a heap

Graph algorithms: By using heaps as internal traversal data structures, run time will be reduced by polynomial order. Examples of such problems are Prim's minimal-spanning-tree algorithm and Dijkstra's shortest-path algorithm.

Priority Queue: A priority queue is an abstract concept like "a list" or "a map"; just as a list can be implemented with a linked list or an array, a priority queue can be implemented with a heap or a variety of other methods.

Order statistics: The Heap data structure can be used to efficiently find the kth smallest (or largest) element in an array.

GROUP: F
ASSIGNMENT NO: 11

Title: Study of Sequential files

Objective: To Study Sequential file system.

Problem Statement: 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 maintain the data.

Outcomes:
To use effective and efficient data structures in solving various Computer Engineering domain problems.

Theory :

Many real life problems use large volume of data and in such cases we require to use some devices such as floppy disk or hard disk to store the data. The data in these devices is stored using the concept of **files .** A file is a collection of related data stored in a particular area on the disk. Programs can be designed to perform the read and write operations on these files.

A program involves either or both of following types of data communication.

1. Data transfer between the console and the program
2. Data transfer between the program and a disk file.

The I/O system of C++ contains a set of classes that define the file handling methods.

These include **ifstream, ofstream** and **fstream .** These classes are derived from **fstream base** and from corresponding **stream** classes. These classes are declared in **fstream.h** header file. We must include this file in the program that uses file.

Details of file stream classes

Fstreambase	**Serves as a base for fstream, ifstream** and **ofstream** class. Contains **close( )** and **open( )**
Ifstream	Provides input operations. Contains **open( )** with default input mode.Inherits the

functions **get(), getline( ), read( ), seekg( ) and tellg( )** functions from **istream.**

Ofstream	Provides output operations. Contains **open( )** with default output mode. Inherits the functions **put( ), write( )seekp( ) and tellp( )** functions from **ostream.**
Fstream	Provides support for i/p and o/p operations.Contains **open(**) without default mode. Inherits all the functions from **istream** and **ostream** classes through **iostream.**
Filebuf	To set file buffers to read and write

A file can be opened in two ways:

- Using the constructor function of the class : This method is useful when we use only one file in the stream.

- Using the member function **open( )** of the class. : This method is used when we want to manage multiple files using one stream.

Opening Files Using Constructor

Filename is used to initialize the file stream object. There are 2 steps :

Create a file stream object to manage the stream using appropriate class
Initialize the file object with the desired filename.
Ex. Ofstream outfile("result");

Open : File Modes

The general form of the function **open( )** with two arguments is :

stream-object.open("filename", mode);

The second argument mode specifies the purpose for which the file is opened. The prototype of the class member functions contain default values for the second argument .

The default values are as :
ios :: in for ifstream functions open for reading only
ios:: out for ofstream functions open for writing only

The file mode parameter can take one (or more) of constants defined in the class **ios()**.

Parameter	**Meaning**
ios :: app	Append to end-of –file.
ios::ate	Go to end-of-file on opening.
ios::in	Open file for reading only
ios::nocreate	Open fails if file does not exist
ios::noreplace	Open fails if file already exists
ios::out	Open file for writing only
ios::trunc	Delete the contents of the file if it exists
ios::binary	Binary file

File Pointers and Their Manipulations

Each file has two associated pointers known as file pointers. One of them is called input pointer(or get pointer) and the other is called the output pointer(or put pointer).

We can use these pointers to move through the files while reading or writing . The input pointer is used for reading the contents of a given location and the output pointer is used for writing to a given location.

Each time an input or output operation takes place , the appropriate pointer is automatically adjusted

Default actions

When we open a file in read-only mode , the input pointer is automatically set at the beginning so that we can read the file from the start.

When we open a file in write-only mode, the existing contents are deleted and the output pointer is set at the beginning of the file. This enables to write to the file from the start

When we want to open an existing file to add more data , the file is opened in 'append' mode . This moves the output pointer to the end of file.

The file stream classes support the functions to manage movements of the file pointers seekg( ) Moves get pointer to a specified location.

seekp( ) Moves put pointer to a specified location.

tellg( ) Gives the current position of the get pointer.

tellp( )	Gives the current position of the put pointer.

Seek functions seekg( ) and seekp( ) can also be used with two arguments:

Seekg(offset, refposition);
Seekp(offset, refposition);

The parameter offset represents the number of bytes the file pointer is to be moved from the location specified by the parameter refposition.

The refposition takes one of following three constants defined in the **ios** class:

ios::beg	start of the file
ios::cur	**c**urrent position of the pointer
ios::end	End of the file

The **seekg()** function moves associated file's 'get' pointer while the **seekp()** function moves the associated file's 'put' pointer.

Sequential Input and Output Operations
The file stream classes support a number of member functions for performing the input and output operations on files.

The first pair of functions , **put( )** and **get( )** are designed for handling a single character at a time

Second pair of functions, **write( )** and **read( )** are designed to write and read a blocks of binary data.

Put( ) and get( ) Functions
The function **put(**) writes a single character to the associated stream
The function **get( )** reads a single character from the associated stream

Write( ) and read( ) Functions
The functions write( ) and read( ) , unlike the functions put( ) and get( ) handle the data in binary form. The binary format is more accurate for storing the numbers as they are stored in the exact internal representation.

The binary input and output functions takes form as:

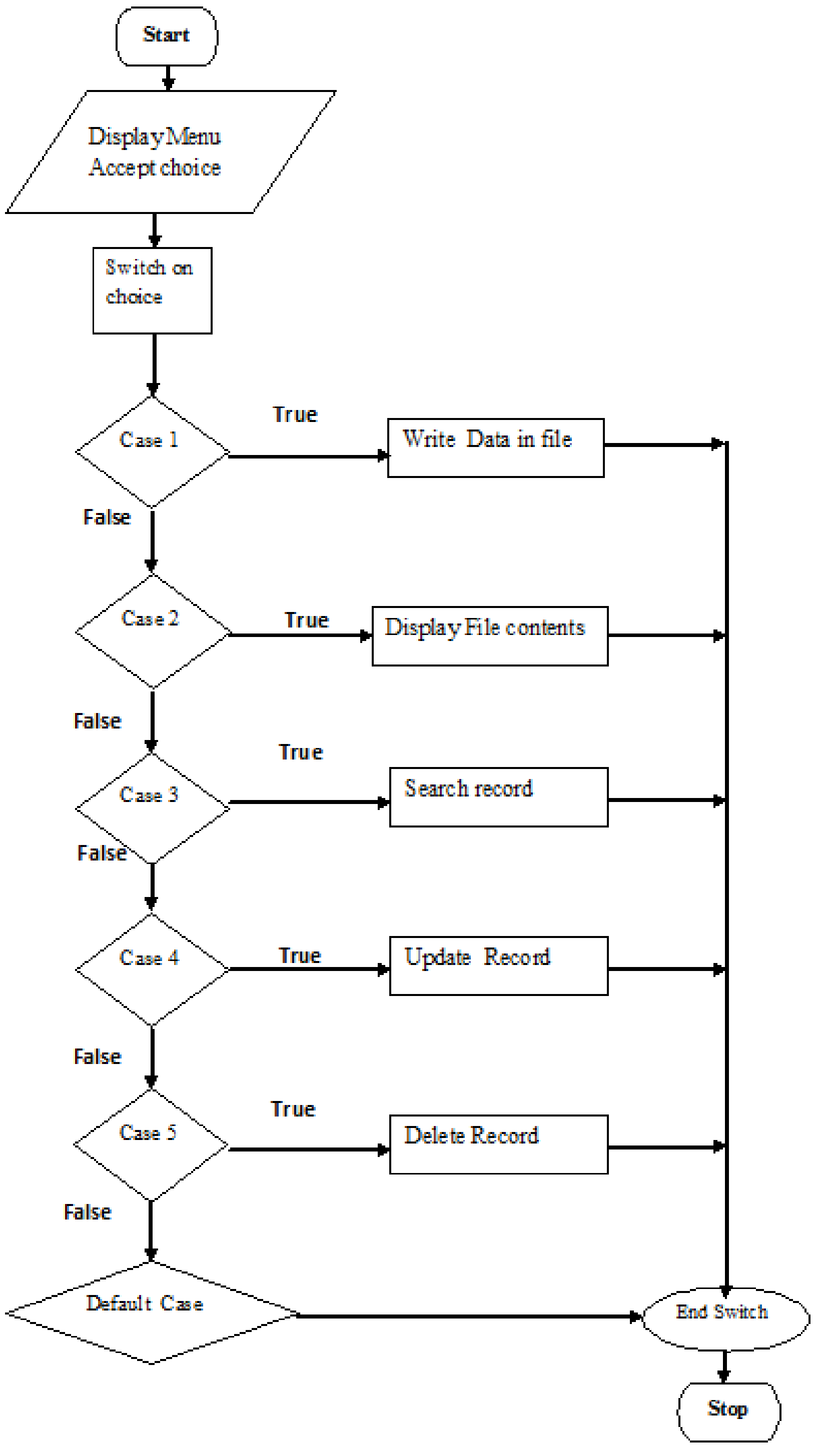
```
infile.read ((char *) &V, sizeof (V) );
outfile.write ((char *) &V, sizeof (V) );
```

The functions takes two arguments . The first is the address of the variable V and the second is the length of variable in bytes.

Algorithm :

1. Open the file
2. Write the student record in file.
3. Display all contents of filename
4. Delete the required student record from file and d display the contents
5. Search for specific student record and display respective record.

Flowchart:


Start
Display Menu
Accept choice
Switch on choice
Case 1
True
Write Data in file
False
Case 2
True
Display File contents
False
True
Case 3
Search record
False
Case 4
True
Update Record
False
True
Case 5
Delete Record
False
Default Case
End Switch
Stop

Test Cases:

Sr. No.	Test ID	Steps	Input	Expected Result	Actual Result	Statu s (Pass/ Fail)
1	ID1	Open the file in ios::out mode	12 Amit I Kothrud	Data to be written in file	Data is written in file	Pass
2	ID2	Open the file in ios::in mode	No input	Data from file to be displayed	Data from file is displayed	Pass
3	ID3	Open the file in ios::in mode, search record	Roll No 12	Respective student record to be displayed	Different student record is played	Fail
4	ID4	Open the file in ios::out mode, delete record and display	Roll No 10	All contents of file except deleted record to be displayed	All contents of file except deleted record are displayed	Pass
5	ID5	Open the file in ios::out mode, append record and display	15 Rahul I Karvenagar	Newly added record to be displayed at end	New record is displayed as last record	Pass

Conclusion: Sequential Files are implemented successfully for Student database system.

FAQs:

Q 1. What is file?

Ans: A file is a collection of related data stored in a particular area on the disk

Q 2. What are Sequential Input and Output Operations?

Ans: Put() and get() , write( ) and read( )

Q 3. What are character input output functions?

Ans: Put() and get()

Q 4. What are file pointers?

Ans: Each file has two associated pointers known as file pointers. One of them is called input pointer(or get pointer) and the other is called the output pointer(or put pointer).

Q 5. What are file opening modes?

Ans:

Parameter	**Meaning**
ios :: app	Append to end-of –file.
ios::ate	Go to end-of-file on opening.
ios::in	Open file for reading only
ios::nocreate	Open fails if file does not exist
ios::noreplace	Open fails if file already exists
ios::out	Open file for writing only
ios::trunc	Delete the contents of the file if it exists
ios::binary	Binary file

Q 6. What is syntax of read and write functions?

Ans: The binary input and output functions takes form as:
infile.**read** ((char *) &V, sizeof (V));
outfile.**write** ((char *) &V, sizeof (V));

Q 7. What are functions for file pointers movement?

Ans:

seekg( )	Moves get pointer to a specified location.
seekp( )	Moves put pointer to a specified location.
tellg( )	Gives the current position of the get pointer.
tellp( )	Gives the current position of the put pointer.

Q 8. In which header file , stream classes are declared?

Ans: The stream classes are declared in **fstream.h** header file.

Q 9. What is a stream :

Ans: Stream is sequence of bytes.

Q10. What are stream classes?

Ans: ofstream: Stream class to write on files
ifstream: Stream class to read from files
fstream: Stream class to both read and write from/to files.

Assignment No: 12

Title: Study of Use index sequential file

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

Problem Statement: 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

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.

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 - return 0 to exit form the program execution.

Approach:
1. For storing the data of the employee, create a user define datatype which will store the information

regarding Employee. Below is the declaration of the data type:
2. struct employee
{ 3. string name;
4. long int Employee_id;
5. string designation;
6. int salary;
7. };

Building the Employee's table:
For building the employee table the idea is to use the array of the above struct datatype which will use to store the information regarding employee.
For storing information at index i the data is stored as:
struct employee emp[10];
emp[i].name = "GeeksforGeeks"
emp[i].code = "12345"
emp[i].designation = "Organisation"
emp[i].exp = 10 emp[i].age = 10

Deleting in the record: Since we are using array to store the data, therefore to delete the data at any index shift all the data at that index by 1 and delete the last data of the array by decreasing the size of array by 1

Searching in the record: For searching in the record based on any parameter, the idea is to traverse the data and if at any index the value parameters matches with the record stored, print all the information of that employee.

Conclusion: Index Sequential Files are implemented successfully for Student database system.

FAQs:

Q 1. What are the advantages of indexed sequential file organization?
Ans: Following are the advantages of In indexed sequential file:
5) sequential file and random file access is possible.
6) It accesses the records very fast if the index table is properly organized.
7) The records can be inserted in the middle of the file.
8) It provides quick access for sequential and direct processing.
9) It reduces the degree of the sequential search.

Q 2. What are the dis advantages of indexed sequential file organization?
Ans: Following are the advantages of In indexed sequential file:
1) Indexed sequential access file requires unique keys and periodic reorganization.
2) Indexed sequential access file takes longer time to search the index for the data access or retrieval.
3) It requires more storage space.
4) It is expensive because it requires special software.
5) It is less efficient in the use of storage space as compared to other file organizations.

Q 3. What are the objectives of file organization?

Ans: The objectives of File organizations are as follows:

1) It contains an optimal selection of records, i.e., records can be selected as fast as possible.
2) To perform insert, delete or update transaction on the records should be quick and easy.
3) The duplicate records cannot be induced as a result of insert, update or delete.
4) For the minimal cost of storage, records should be stored efficiently

Q 4. What are the types of file organization?

Ans: File organization contains various methods. These particular methods have pros and cons on the basis of access or selection. In the file organization, the programmer decides the best-suited file organization method according to his requirement. Types of file organization are as follows:

- Sequential file organization
- Heap file organization
- Hash file organization
- B+ file organization
- Indexed sequential access method (ISAM)
- cluster file organization

Q 5. What are the advantages of direct access file organization?

Ans: Advantages are as follows:

1) Direct access file helps in online transaction processing system (OLTP) like online railway reservation system.
2) In direct access file, sorting of the records are not required.
3) It accesses the desired records immediately.
4) It updates several files quickly.
5) It has better control over record allocation.

Q.6. What are the disadvantages of direct access files?

Ans: Disadvantages are as follows:

1. Direct access file does not provide back up facility.
2. It is expensive.
3. It has less storage space as compared to sequential file

Q.7. What are the advantages of sequential access file?

Ans: Advantages are as follows:

1. It is simple to program and easy to design.
2. Sequential file is best use if storage space.

Q.8. What are the dis advantages of sequential access file?

Ans: Dis-advantages are as follows:

1. Sequential file is time consuming process.
2. It has high data redundancy.

3. Random searching is not possible

Q.9. What are the important features of files?

Ans: important features are as follows:

1. **File Activity :**File activity specifies percent of actual records which proceed in a single run.
2. **File Volatility:** File volatility addresses the properties of record changes. It helps to increase the efficiency of disk design than tape.
3. **File organization:** File organization ensures that records are available for processing. It is used to determine an efficient file organization for each base relation. For example, if we want to retrieve employee records in alphabetical order of name. Sorting the file by employee name is a good file organization. However, if we want to retrieve all employees whose marks are in a certain range, a file is ordered by employee name would not be a good file organization.

Q.10. What is cluster file organization?

Ans: When the two or more records are stored in the same file, it is known as clusters. These files will have two or more tables in the same data block, and key attributes which are used to map these tables together are stored only once. This method reduces the cost of searching for various records in different files. The cluster file organization is used when there is a frequent need for joining the tables with the same condition. These joins will give only a few records from both tables. In the given example, we are retrieving the record for only particular departments. This method can't be used to retrieve the record for the entire department.

Q.11. What are the types of cluster file organization?

Ans: Cluster file organization is of two types:

1. Indexed Clusters: In indexed cluster, records are grouped based on the cluster key and stored together. The above EMPLOYEE and DEPARTMENT relationship is an example of an indexed cluster. Here, all the records are grouped based on the cluster key- DEP_ID and all the records are grouped.
2. Hash Clusters: It is similar to the indexed cluster. In hash cluster, instead of storing the records based on the cluster key, we generate the value of the hash key for the cluster key and store the records with the same hash key value.

Q.10. What are the advantages of cluster file organization?

Ans: The advantages are as follows:

1. cluster file organization is used when there is a frequent request for joining the tables with same joining condition.
2. It provides the efficient result when there is a 1:M mapping between the tables.

Assignment No: 13

Title: To apply BFS and queue combined

Objective: To understand the application of BFS.

Problem Statement: Design a mini project to implement snake and ladder game using Python.

Outcome: apply the graph traversals

Theory: The idea is to consider the snakes and ladders board as a directed graph and run Breadth–first search (BFS) from the starting node, vertex 0, as per game rules. We construct a directed graph, keeping in mind the following conditions:

1. For any vertex in graph `v`, we have an edge from `v` to `v+1`, `v+2`, `v+3`, `v+4`, `v+5`, `v+6` as we can reach any of these nodes in one throw of dice from node `v`.
2. If any of these neighbors of `v` has a ladder or snake, which takes us to position `x`, then `x` becomes the neighbor instead of the base of the ladder or head of the snake.

Now the problem is reduced to finding the shortest path between two nodes in a directed graph problem. We represent the snakes and ladders board using a map.

Output: Attach program and output of the project

Assignment No: 14

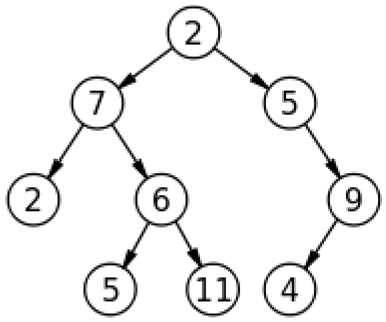
Title: Study DFS traversal on binary trees.

Objective: To understand DFS traversal on binary trees**.**

Problem Statement: Implementation of DFS traversal on binary trees using Python classes.

Outcome: Classify the Tree data structures & apply it for problem solving

Theory:
Binary Tree: A binary tree is a tree data structure in which each node has at most two children, which are referred to as the left child and the right child.
Example of binary tree



Tree Traversals (Inorder, Preorder and Postorder)

Unlike linear data structures (Array, Linked List, Queues, Stacks, etc) which have only one logical way to traverse them, trees can be traversed in different ways. Following are the generally used ways for traversing trees.

Inorder Traversal:
Algorithm Inorder(tree)
1. Traverse the left subtree, i.e., call Inorder(left-subtree)
2. Visit the root.
3. Traverse the right subtree, i.e., call Inorder(right-subtree)

Preorder Traversal:

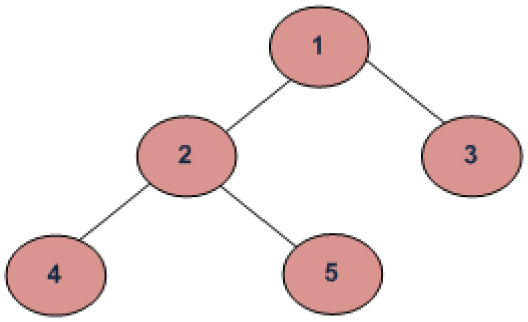
Algorithm Preorder(tree)
1. Visit the root.
2. Traverse the left subtree, i.e., call Preorder(left-subtree)
3. Traverse the right subtree, i.e., call Preorder(right-subtree)

Postorder Traversal:

Algorithm Postorder(tree)

1. Traverse the left subtree, i.e., call Postorder(left-subtree)
2. Traverse the right subtree, i.e., call Postorder(right-subtree)
3. Visit the root.

Example of Traversal



Depth First Traversals:

(a) Inorder (Left, Root, Right) : 4 2 5 1 3

(b) Preorder (Root, Left, Right) : 1 2 4 5 3Ans: S

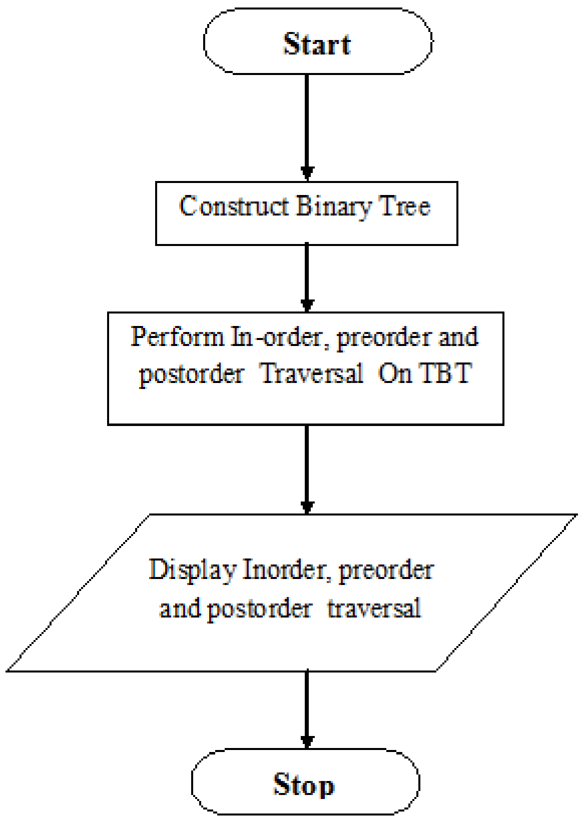
(c) Postorder (Left, Right, Root) : 4 5 2 3 1

Breadth First or Level Order Traversal : 1 2 3 4 5

Algorithm :

1. Define class node including constructor
2. Define class tree including constructor
3. Display inorder , preorder and postroder traversals on constructed tree

Flowchart:


Start
Construct Binary Tree
Perform In-order, preorder and postorder Traversal On TBT
Display Inorder, preorder and postorder traversal
Stop

Test Cases:

Sr. NO	Test ID	Steps	Input	Expected Result	Actual Result	Statu s (Pass/ Fail)
1	ID1	Construct a tree and display inorder traversal on it	10 20 30	Inorder traversal to be displayed	Inorder traversal is displayed 20 10 30	Pass
2	ID2	Construct a tree and display preorder traversal on it	10 20 30	Preorder traversal to be displayed	Preorder traversal is displayed 10 20 30	Pass

Conclusion: Successfully Tree traversals are implemented**.**

FAQs:

Q1. What is BFS (with example)

Ans: BFS is breadth first traversal. Example

Tree1

```
      j        <-- level 0
     / \
    f      k     <-- level 1
  /   \      \
 a     h      z <-- level 2
  \
   d            <-- level 3
```

So, if we want to visit the elements level-by-level (and left-to-right, as usual), we would start at level 0 with **j**, then go to level 1 for **f** and **k**, then go to level 2 for **a, h** and **z**, and finally go to level 3 for **d**.

This level-by-level traversal is called a breadth-first traversal because we explore the breadth, i.e., full width of the tree at a given level, before going deeper.

Q2. What is DFS (with example)

Ans: DFS is Depth first Traversal

Tree1

```
      j    <-- root
     / \
    f      k
  /   \      \
 a     h      z
  \
   d
```

A preorder traversal would visit the elements in the order: j, f, a, d, h, k, z.

This type of traversal is called a depth-first traversal. Why? Because it tries to go deeper in the tree before exploring siblings. For example, the traversal visits all the descendants of f (i.e., keeps going deeper) before visiting f's sibling k (and any of k's descendants).

Q3. Which data structure is used in BFS

Ans: Queue

Q.4 What are the applications of BFS?

Ans: Most of the concepts in computer science and real world can be visualized and represented in terms of graph data structure. BFS is one such useful algorithm for solving these problems easily. The architecture of BFS is simple, accurate and robust. It is very seamless as it is guaranteed that the algorithm won't get caught in an infinite loop.

- **Shortest Path:** In an unweighted graph, the shortest path is the path with least number of edges. With BFS, we **always** reach a node from given source in shortest possible path. Example: Dijkstra's Algorithm.
- **GPS Navigation Systems:** BFS is used to find the neighboring locations from a given source location.
- **Finding Path:** We can use BFS to find whether a path exists between two nodes.
- **Finding nodes within a connected component:** BFS can be used to find all nodes reachable from a given node.
- **Social Networking Websites:** We can find number of people within a given distance 'k' from a person using BFS.
- **P2P Networks:** In P2P (Peer to Peer) Networks like BitTorrent, BFS is used to find all neighbor nodes from a given node.
- **Search Engine Crawlers:** The main idea behind crawlers is to start from source page and follow all links from that source to other pages and keep repeating the same. DFS can also be used here, but Breadth First Traversal has the advantage in limiting the depth or levels traversed.

Q. 5 What are the applications of DFS?

Ans: DFS algorithm works in the following way:

Most of the concepts in computer science can be visualized and represented in terms of graph structure.

- Solving maze-like puzzles with only one solution: DFS can be used to find all solutions to a maze problem by only considering nodes on the current path in the visited set.
- Topological Sorting:
    - This is mainly used for scheduling jobs from the given dependencies among jobs. DFS is highly preferred approach while finding solutions to the following type of problems using Topological Sort:
        - instruction/job scheduling
        - ordering of formula cell evaluation when recomputing formula values in spreadsheets
        - determining the order of compilation tasks to perform in makefiles
        - data serialization
        - resolving symbol dependencies in linkers
- Mapping Routes and Network Analysis
- Path Finding: DFS is used for finding path between two given nodes - source and destination - in a graph.
- Cycle detection in graphs

Q.6 Write an algorithm to check whether the graph is strongly connected or not?

Ans:

1. Start `DFS(G, v)` from a random vertex `v` of the graph `G`. If `DFS(G, v)` fails to reach every other vertex in the graph `G`, then there is some vertex `u`, such that there is no directed path from `v` to `u`. Thus, `G` is not strongly connected. If it does reach every vertex, then there is a directed path from `v` to every other vertex in the graph `G`.
2. Reverse the direction of all edges in the directed graph `G`.
3. Again, run a DFS starting from vertex `v`. If the DFS fails to reach every vertex, then there is some vertex `u`, such that in the original graph, there is no directed path from `u` to `v`. On the other hand, if it does reach every vertex, then there is a directed path from every vertex `u` to `v` in the original graph.

Q. 7 How can you say that the graph is strongly connected?

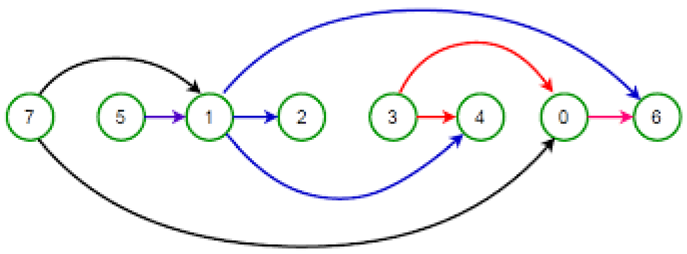
Ans: We can say that G is strongly connected if:

1. DFS(G, v) visits all vertices in the graph G, then there exists a path from v to every other vertex in G, and
2. There exists a path from every other vertex in G to v.

Q.8 What is topological sort using DAG?

Ans: Given a Directed Acyclic Graph (DAG), print it in topological order using topological sort algorithm. If the graph has more than one topological ordering, output any of them. Assume valid Directed Acyclic Graph (DAG).

A Topological sort or Topological ordering of a directed graph is a linear ordering of its vertices such that for every directed edge uv from vertex u to vertex v , u comes before v in the ordering. A topological ordering is possible if and only if the graph has no directed cycles, i.e. if the graph is DAG.



Topological Order

Q.9 How the non-recursive implementation of DFS differs from the non-recursive implementation of BFS?

Ans: it differs in following way:

- It uses a stack instead of a queue.
- The DFS should mark discovered only after popping the vertex, not before pushing it.
- It uses a reverse iterator instead of an iterator to produce the same results as recursive DFS.

Q.10 What are two ways to check whether graph is bipartite?

Ans: There are two ways to check whether the graph is bipartite:

1. A graph is bipartite if and only if it is 2–colorable.
2. A graph is bipartite if and only if it does not contain an odd cycle.