Practical No. 1

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

Software and Hardware Requirements: 1. Intel i3,3.3GHz,4gb ram.

2.Linux based Ubuntu Operating System.

3.Python distribution.

4.Gedit text editor.

5.Terminal (Command line interface).

Theory:

Hashing:

Hashing is a technique or process of mapping keys, and values into the hash table by using a hash function. It is done for faster access to elements. The efficiency of mapping depends on the efficiency of the hash function used.

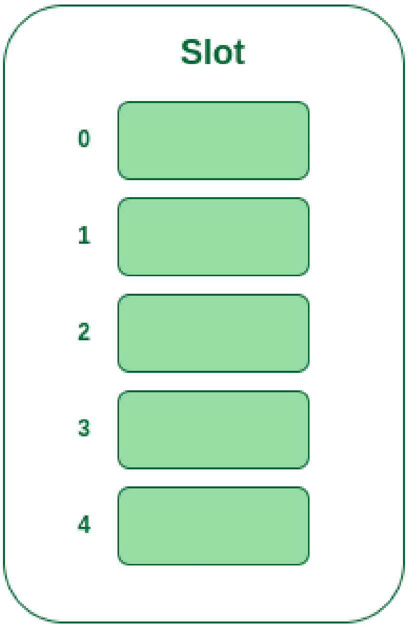
Linear Probing:

In linear probing, the hash table is searched sequentially that starts from the original location of the hash. If in case the location that we get is already occupied, then we check for the next location.

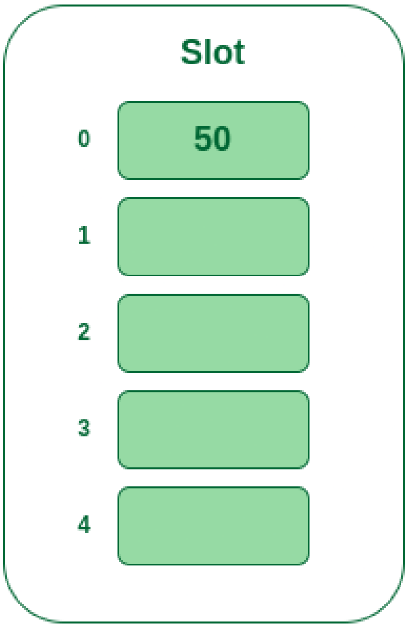
The function used for rehashing is as follows: rehash(key) = (n+1)%table-size.

Example: Let us consider a simple hash function as "key mod 5" and a sequence of keys that are to be inserted are 50, 70, 76, 93.

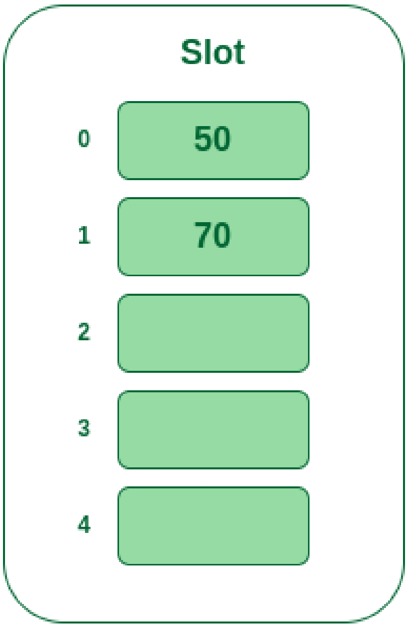
- **Step1:** First draw the empty hash table which will have a possible range of hash values from 0 to 4 according to the hash function provided.
- **Step 2:** Now insert all the keys in the hash table one by one. The first key is 50. It will map to slot number 0 because 50%5=0. So insert it into slot number 0.
- **Step 3:** The next key is 70. It will map to slot number 0 because 70%5=0 but 50 is already at slot number 0 so, search for the next empty slot and insert it.
- **Step 4:** The next key is 76. It will map to slot number 1 because 76%5=1 but 70 is already at slot number 1 so, search for the next empty slot and insert it.
- **Step 5:** The next key is 93 It will map to slot number 3 because 93%5=3, So insert it into slot number 3.



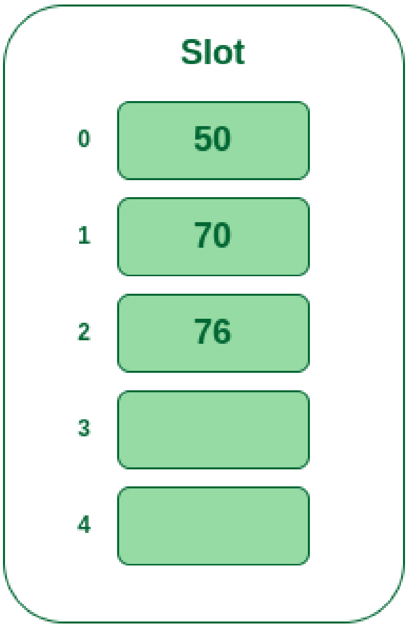
Step 1: Create empty hash table



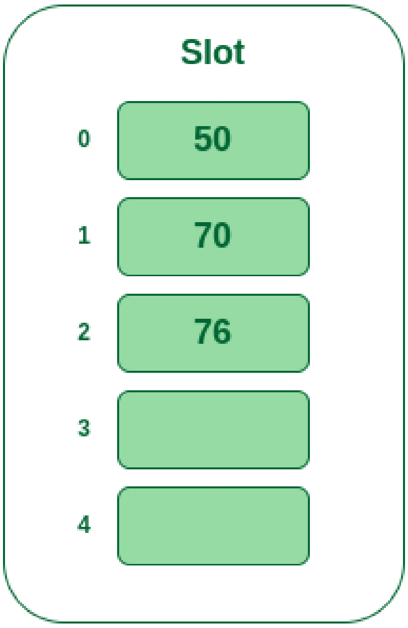
Step 2: Insert 50 into hash table



Step 3: Insert 70 into hash table



Step 4: Insert 76 into hash table



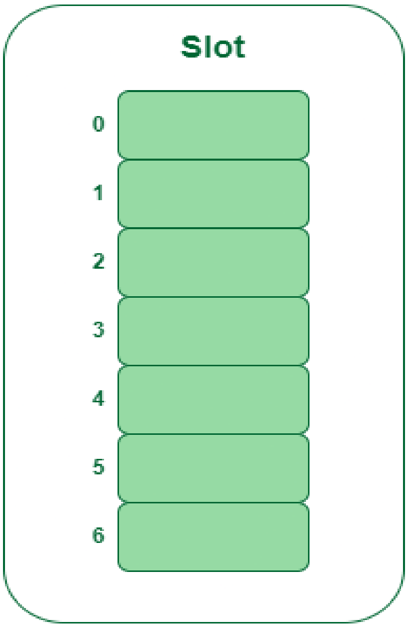
Step 5: Insert 76 into hash table

Quadratic Probing:

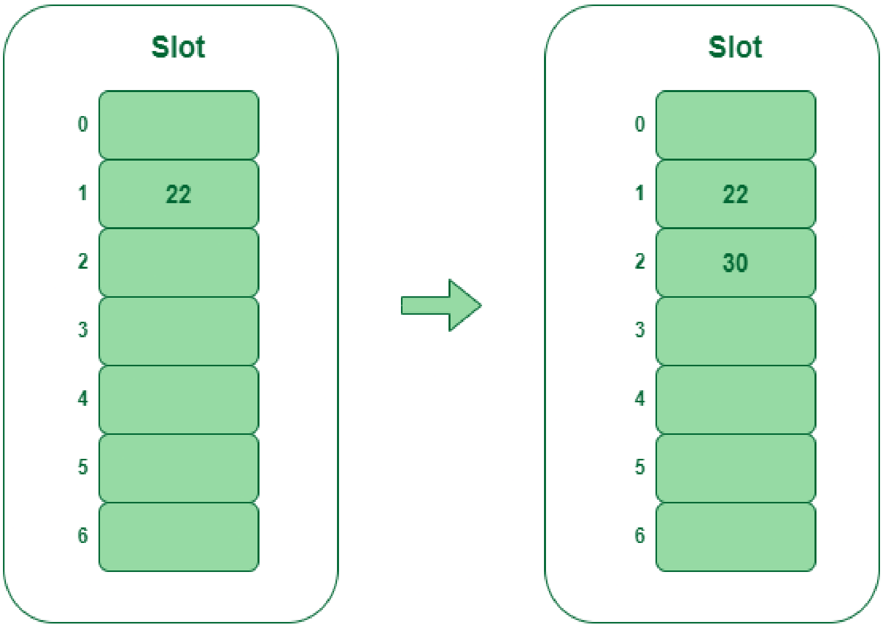
Quadratic probing is a method with the help of which we can solve the problem of clustering that was discussed above. This method is also known as the **mid-square** method. In this method, we look for the **i^2'th** slot in the **i^{th}** iteration. We always start from the original hash location. If only the location is occupied then we check the other slots.

Example: Let us consider table Size = 7, hash function as Hash(x) = x % 7 and collision resolution strategy to be $f(i) = i^2$. Insert = 22, 30, and 50.

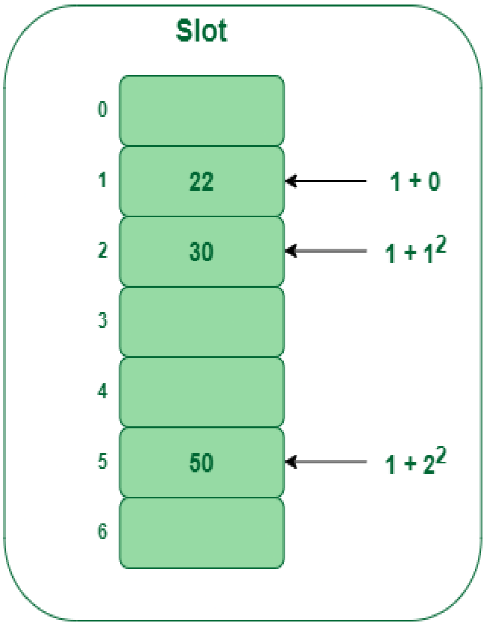
- **Step 1:** Create a table of size 7.
- **Step 2** – Insert 22 and 30
  - Hash(22) = 22 % 7 = 1, Since the cell at index 1 is empty, we can easily insert 22 at slot 1.
  - Hash(30) = 30 % 7 = 2, Since the cell at index 2 is empty, we can easily insert 30 at slot 2.
- **Step 3:** Inserting 50
  - Hash(50) = 50 % 7 = 1
  - In our hash table slot 1 is already occupied. So, we will search for slot $1+1^2$, i.e. 1+1 = 2,
  - Again slot 2 is found occupied, so we will search for cell $1+2^2$, i.e.1+4 = 5,
  - Now, cell 5 is not occupied so we will place 50 in slot 5.



Step 1: Create empty hash table



Step 2: Insert keys 22 and 30 in the hash table



Step 3: Insert 50 in the hash table

Conclusion: In this practical, we used a hash table implementation to quickly look up client‘s telephone number and Made use of two collision handling techniques and compared them using number of comparisons required to find a set of telephone numbers.

Practical No. 2

Aim: For given set of elements create skip list. Find the element in the set that is closest to some given value.

Software and Hardware Requirements: 1. Intel i3,3.3GHz,4gb ram.

2.Linux based Ubuntu Operating System.

3.Python distribution.

4.Gedit text editor.

5.Terminal (Command line interface).

Theory:

Skip List :

A skip list is a probabilistic data structure. The skip list is used to store a sorted list of elements or data with a linked list. It allows the process of the elements or data to view efficiently. In one single step, it skips several elements of the entire list, which is why it is known as a skip list.

Algorithm of insertion in Skiplist:

```
Insertion (L, Key)
local update [0...Max_Level + 1]
a = L → header
for i = L → level down to 0 do.
    while a → forward[i] → key  forward[i]
update[i] = a
a = a → forward[0]
lvl = random_Level()
if lvl > L → level then
for i = L → level + 1 to lvl do
    update[i] = L → header
    L → level = lvl
a = makeNode(lvl, Key, value)
for i = 0 to level do
    a → forward[i] = update[i] → forward[i]
update[i] → forward[i] = a
```

Algorithm of Searching in Skiplist:

```
Searching (L, SKey)
    a = L → header
    loop invariant: a → key level down to 0 do.
        while a → forward[i] → key forward[i]
    a = a → forward[0]
   if a → key = SKey then return a → value
   else return failure
```

Example 1: Create a skip list, we want to insert these following keys in the empty skip list.

1. 6 with level 1.
2. 29 with level 1.
3. 22 with level 4.
4. 9 with level 3.
5. 17 with level 1.
6. 4 with level 2.

Ans:

Step 1: Insert 6 with level 1

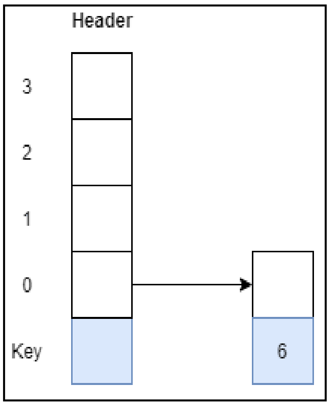
Step 2: Insert 29 with level 1

Step 3: Insert 22 with level 4

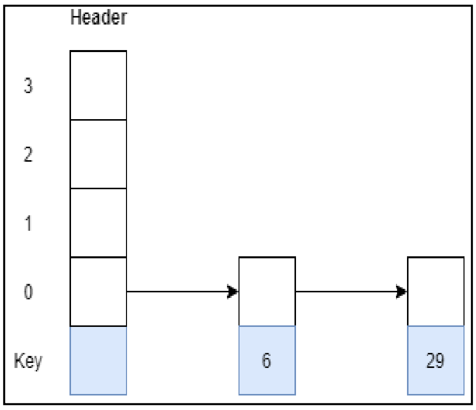
Step 4: Insert 9 with level 3

Step 5: Insert 17 with level 1

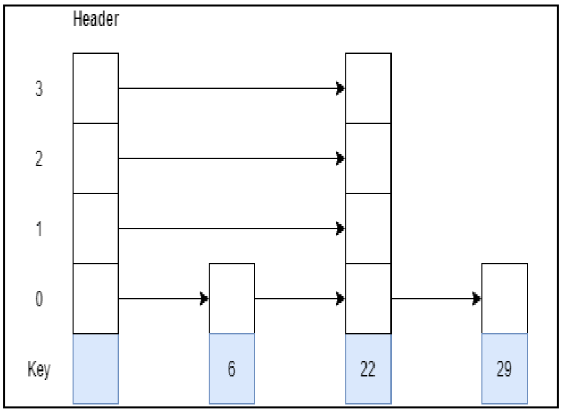
Step 6: Insert 4 with level 2



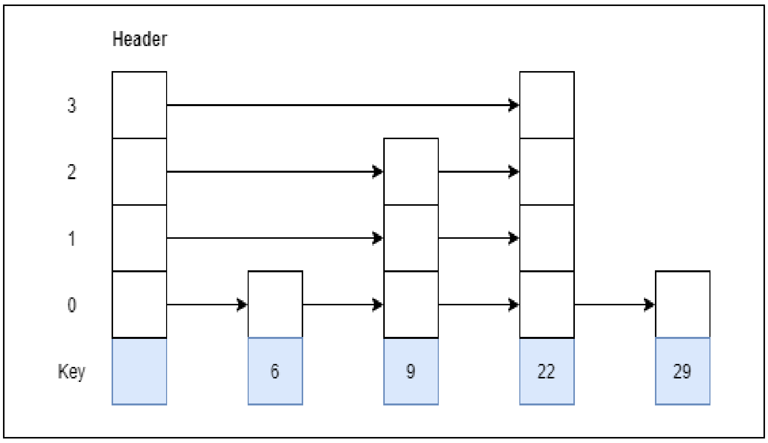
Step 1: Insert 6 with



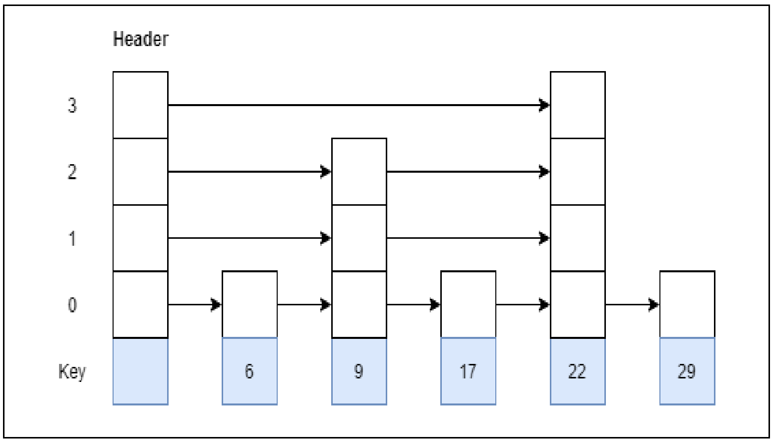
Step 2: Insert 29



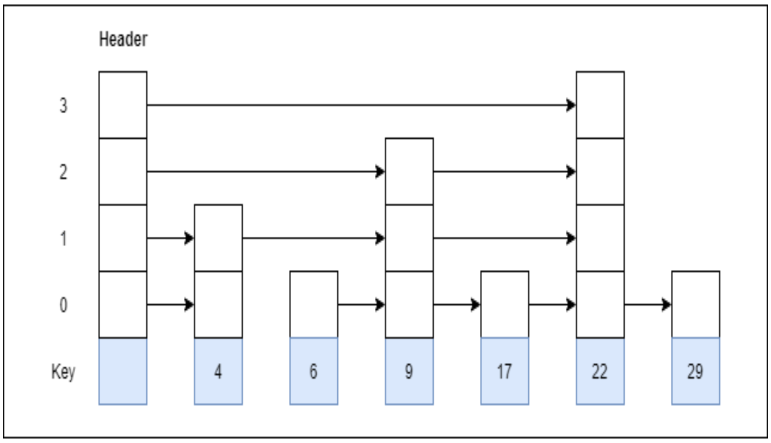
Step 3: Insert 22



Step 4: Insert 9 with



Step 5: Insert 17



Step 6: Insert 4 with

Conclusion: In this practical, we created a create skip list from given set of elements . And performed search operation on the elements.

Practical No. 3

Aim: 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.

Software and Hardware Requirements: 1. Intel i3,3.3GHz,4gb ram.

2.Linux based Ubuntu Operating System.

3.C++ Compiler.

4.Gedit text editor.

5.Terminal (Command line interface).

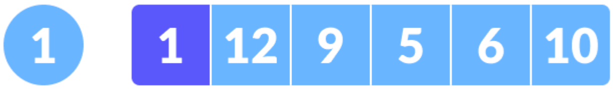
Theory:

Binary Tree:

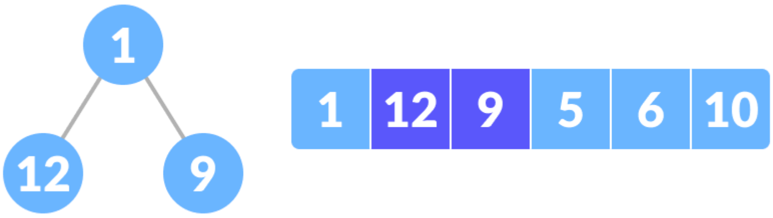
A binary tree is a tree data structure in which each node can have at most two children, which are referred to as the left child and the right child.

Algorithm to create Binary Tree

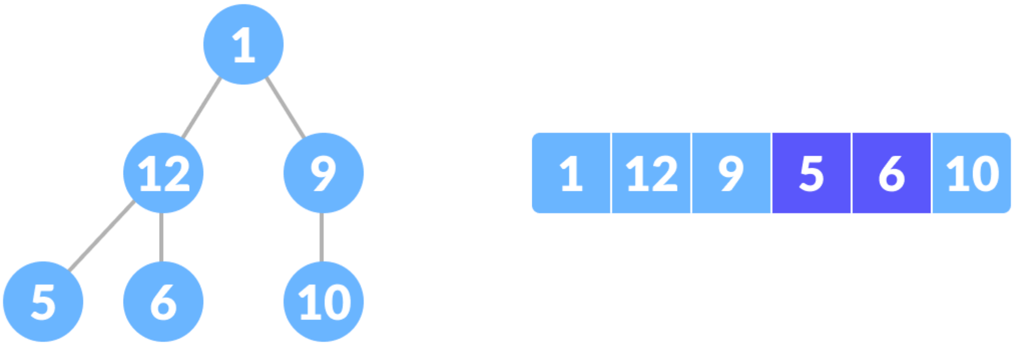
1. Select the first element of the list to be the root node. (no. of elements on level-I: 1)
2. Put the second element as a left child of the root node and the third element as the right child. (no. of elements on level-II)
3. Put the next two elements as children of the left node of the second level. Again, put the next two elements as children of the right node of the second level (no. of elements on level-III: 4) elements).
4. Keep repeating until you reach the last element.



Select the first element as root

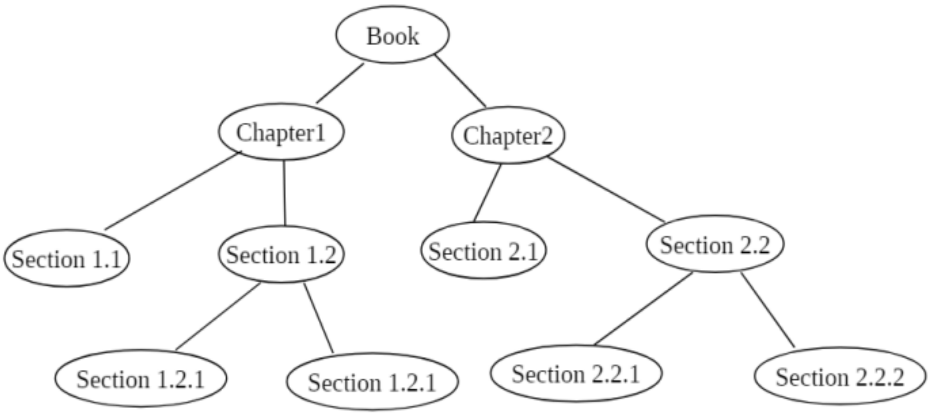


12 as a left child and 9 as a right child



5 as a left child and 6 as a right child

Binary Trees are very flexible data structure, allowing to move subtrees around with minimum effort for this practical we are considering the tree as follows.



Time Complexity: O(n) where n is the number of nodes in the n-ary tree.
Space Complexity: O(n)

Conclusion: In this practical , we constructed a Binary tree and printed its node.

Practical No. 5

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

Software and Hardware Requirements: 1. Intel i3,3.3GHz,4gb ram.

2.Linux based Ubuntu Operating System.

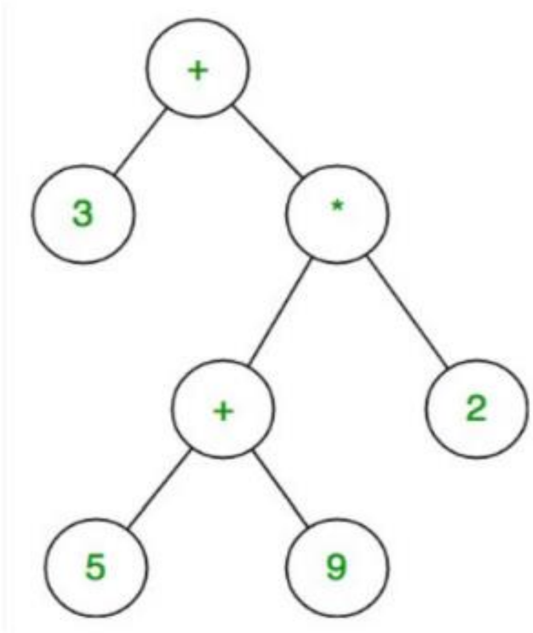
3.C++ Compiler.

4.Gedit text editor.

5.Terminal (Command line interface).

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 tree for 3+((5+9)*2) would be:



There are different types of expression formats:

1. Prefix expression
2. Infix expression
3. Postfix expression

Expression tree is a special kind of binary tree with the following properties:

1. Each leaf is an operand. Example:a,b,c,6,100
2. The root and internal nodes are operators. Example:+,-,*,/,^
3. Subtrees are subexpressions with the root being sn operator.

Traversal Techniques:

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

1. Inorder Traversal: We can produce an infix expression by recursively printing out the left expression, the root, and then the right expression.
2. Postorder Traversal: The postfix expression can be evaluated by recursively printing out the left expression, the right expression, and then the root.
3. Preorder Traversal: We can also evaluate prefix expression recursively by printing out the root, the left expression, and then the right expression.

Algorithm for 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 into the Stack
3. If the symbol is an operator, then pop two pointers from the stcak 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
    a. Pop a node from frist stack and push it to second stack.
    b. Push left and right children of the popped node to first stack
3. Print contents of second stack
```

Conclusion: In this practical, we Constructed an expression tree from the given prefix expression and traversed it using post order traversal (non recursive) and then deleted the entire tree.

Practical No. 6

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

Software and Hardware Requirements: 1. Intel i3,3.3GHz,4gb ram.

2.Linux based Ubuntu Operating System.

3.C++ Compiler.

4.Gedit text editor.

5.Terminal (Command line interface).

Theory:

Depth First Search

It is a recursive algorithm to search all the vertices of a tree data structure or a graph. The depth-first search (DFS) algorithm starts with the initial node of graph G and goes deeper until we find the goal node or the node with no children.

Because of the recursive nature, stack data structure can be used to implement the DFS algorithm. The process of implementing the DFS is similar to the BFS algorithm.

Algorithm

1. Step 1: SET STATUS = 1 (ready state) for each node in G
2. Step 2: Push the starting node A on the stack and set its STATUS = 2 (waiting state)
3. Step 3: Repeat Steps 4 and 5 until STACK is empty
4. Step 4: Pop the top node N. Process it and set its STATUS = 3 (processed state)
5. Step 5: Push on the stack all the neighbors of N that are in the ready state (whose STATUS = 1) and set their STATUS = 2 (waiting state)
6. [END OF LOOP]
7. Step 6: EXIT

Breadth First Search

It is a recursive algorithm to search all the vertices of a tree or graph data structure. BFS puts every vertex of the graph into two categories - visited and non-visited. It selects a single node in a graph and, after that, visits all the nodes adjacent to the selected node.

Algorithm

Step 1: SET STATUS = 1 (ready state) for each node in G

Step 2: Enqueue the starting node A and set its STATUS = 2 (waiting state)

Step 3: Repeat Steps 4 and 5 until QUEUE is empty

Step 4: Dequeue a node N. Process it and set its STATUS = 3 (processed state).

```
Step 5: Enqueue all the neighbours of N that are in the ready state (whose STATUS = 1) and set
their STATUS = 2
(waiting state)
[END OF LOOP]
Step 6: EXIT
```

Conclusion: In this practical, we have Represented a given graph using adjacency matrix/list to perform DFS.

Practical No. 7

Aim: 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.

Software and Hardware Requirements: 1. Intel i3,3.3GHz,4gb ram.

2.Linux based Ubuntu Operating System.

3.C++ Compiler.

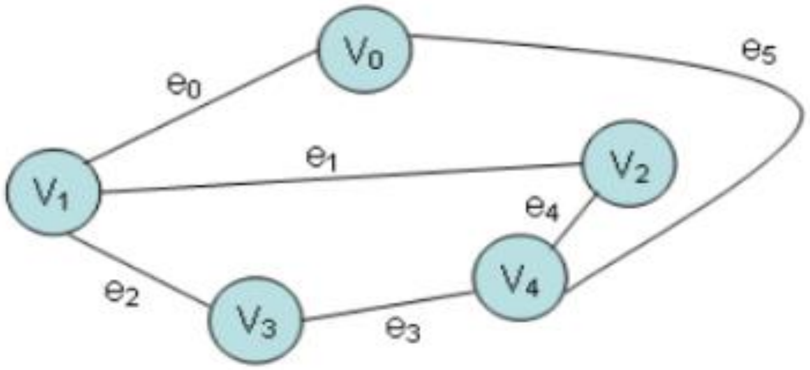
4.Gedit text editor.

5.Terminal (Command line interface).

Theory:

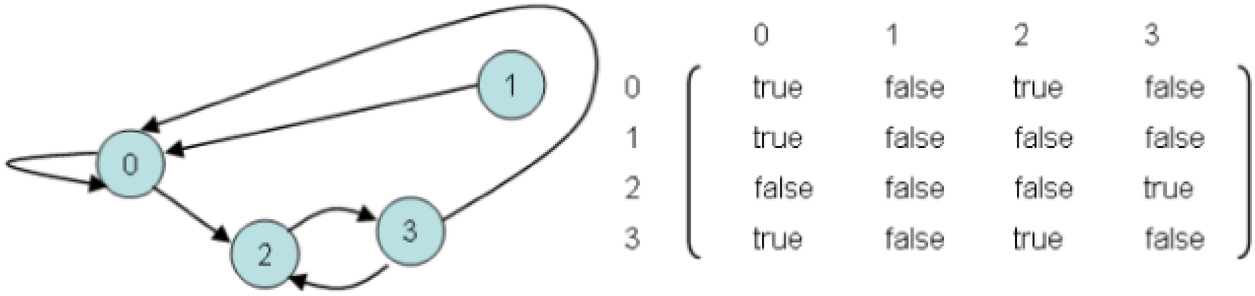
Graphs

Graphs are the most general data structure. They are also commonly used data structures. Graph definitions:
• A non-linear data structure consisting of nodes and links between nodes.
Undirected graph definition:
• An undirected graph is a set of nodes and a set of links between the nodes.
• Each node is called a vertex, each link is called an edge, and each edge connects two vertices.
• The order of the two connected vertices is unimportant.
• An undirected graph is a finite set of vertices together with a finite set of edges. Both sets might be empty, which is called the empty graph.



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

Representing Graphs with an Adjacency Matrix:



$$\begin{array}{c|cccc} & 0 & 1 & 2 & 3 \\ \hline 0 & \text{true} & \text{false} & \text{true} & \text{false} \\ 1 & \text{true} & \text{false} & \text{false} & \text{false} \\ 2 & \text{false} & \text{false} & \text{false} & \text{true} \\ 3 & \text{true} & \text{false} & \text{true} & \text{false} \end{array}$$

Fig: Graph and adjacency matrix

Definition:

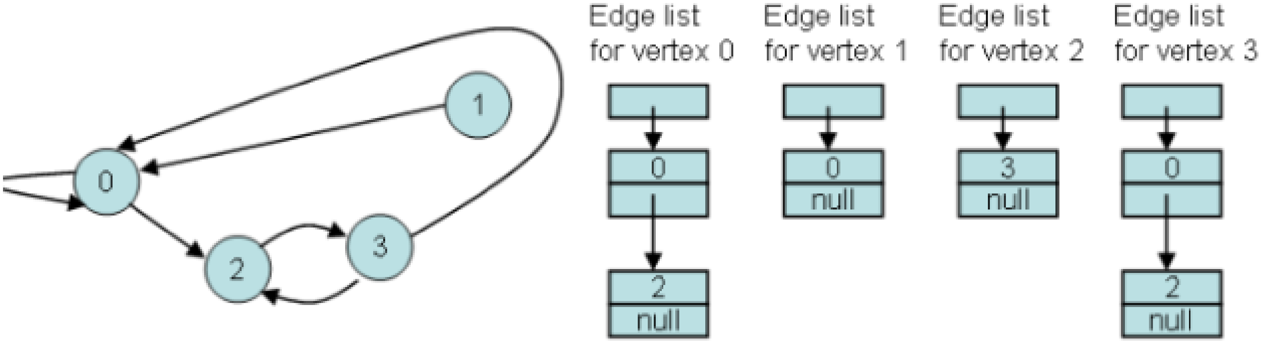
- An adjacency matrix is a square grid of true/false values that represent the edges of a graph.
- If the graph contains n vertices, then the grid contains n rows and n columns.
- For two vertex numbers i and j, the component at row i and column j is true if there is an edge from vertex i to vertex j;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

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

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

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

Representing Graphs with Edge Lists:



Definition:

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

Loops and multiple edges could be allowed.

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 Vi, V) 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].

Conclusion: In this practical, we have implemented graph using adjacency matrix.

Practical No. 8

Aim: Given sequence k = k1 < … < 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?

Software and Hardware Requirements: 1. Intel i3,3.3GHz,4gb ram.

2.Linux based Ubuntu Operating System.

3.C++ Compiler.

4.Gedit text editor.

5.Terminal (Command line interface).

Theory:

Binary Search Tree:

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.

Optimal Binary Search Tree:

An Optimal Binary Search Tree (OBST), also known as a Weighted Binary Search Tree, is a binary search tree that minimizes the expected search cost. In a binary search tree, the search cost is the number of comparisons required to search for a given key.

Algorithm for Optimal Binary Search Tree

```
Algorithm OBST(p, q, n)
// e[1…n+1, 0…n ] : Optimal sub tree
// w[1…n+1,  0…n] : Sum of probability
// root[1…n, 1…n] : Used to construct OBST

for i ← 1 to n + 1 do
  e[i, i – 1] ← qi – 1
  w[i, i – 1] ← qi – 1
end

for m ← 1 to n do
  for i ← 1 to n – m + 1 do
    j ← i + m – 1
    e[i, j] ← ∞
    w[i, j] ← w[i, j – 1] + pj + qj
    for r ← i to j do
      t ← e[i, r – 1] + e[r + 1, j] + w[i, j]
      if t < e[i, j] then
        e[i, j] ← t
        root[i, j] ← r
      end
    end
  end
end
return (e, root)
```

Conclusion: In this practical, we created optimal binary search tree that has the least search cost.

Practical No. 10

Aim: 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.

Software and Hardware Requirements: 1. Intel i3,3.3GHz,4gb ram.

2.Linux based Ubuntu Operating System.

3.C++ Compiler.

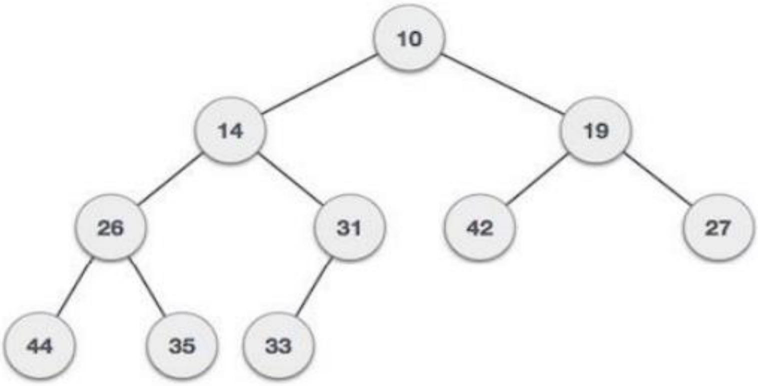
4.Gedit text editor.

5.Terminal (Command line interface).

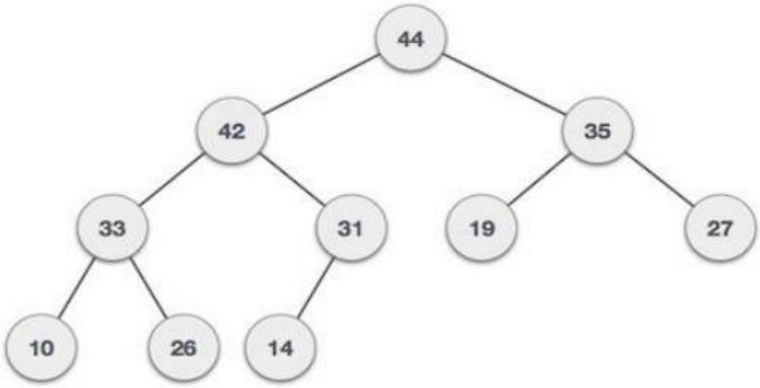
Theory:

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 −
For Input → 35 33 42 10 14 19 27 44 26 31
Min-Heap − Where the value of the root node is less than or equal to either of its children.



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



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

Heap Construction Algorithm :

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

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

Algorithm:
Step 1 − Create a new node at the end of heap.
Step 2 − Assign new value to the node.
Step 3 − Compare the value of this child node with its parent.
Step 4 − If value of parent is less than child, then swap them.
Step 5 − Repeat step 3 & 4 until Heap property holds.
Note − In Min Heap construction algorithm, we expect the value of the parent node to be less than that of the child node.

Heap Deletion Algorithm :

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

Algorithm:
Step 1 − Remove root node.
Step 2 − Move the last element of last level to root.
Step 3 − Compare the value of this child node with its parent.
Step 4 − If value of parent is less than child, then swap them.
Step 5 − Repeat step 3 & 4 until Heap property holds.

Complexity Analysis:

OPERATION	TIME COMPLEXITY		SPACE COMPLEXITY
Insertion	Best Case:	O(1)	O(1)
	Worst Case:	O(logN)	
	Average Case:	O(logN)	
Deletion	Best Case:	O(1)	O(1)
	Worst Case:	O(logN)	
	Average Case:	O(logN)	

Conclusion: In this practical, We Implemented Heap Data Structure to find out maximum and minimum marks obtained in a subject.

Practical No. 12

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

Software and Hardware Requirements: 1. Intel i3,3.3GHz,4gb ram.

2.Linux based Ubuntu Operating System.

3.C++ Compiler.

4.Gedit text editor.

5.Terminal (Command line interface).

Theory:

Indexed sequential access file organization

1. Indexed sequential access file combines both sequential file and direct access file organization.
2. In indexed sequential access file, records are stored randomly on a direct access device such as magnetic disk by a primary key.
3. This file have multiple keys. These keys can be alphanumeric in which the records are ordered is called primary key.
4. 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:

1. Write (add, store): User provides a new key and record, IS file inserts the new record and key.
2. 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"
3. Rewrite (replace): User provides an existing key and a new record, IS file replaces existing record with New.
4. 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 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:

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:

```
struct employee
{ string name;
  long int Employee_id;
  string designation;
  int salary;
};
```

Conclusion: In this practical, we understand the concept and basic of sequential file and its use in Data structure and implemented by creating a student information system