Savitribai Phule Pune University

Second Year of Computer Engineering (2019 Course)

(With effect from Academic Year 2020-21)

Semester-III

Course Code	Course Name	Teaching Scheme (Hours/Week)			Examination Scheme and Marks						Credit Scheme			
		Lecture	Practical	Tutorial	Mid-Sem	End-Sem	Term work	Practical	Oral	Total	Lecture	Practical	Tutorial	Total
210241	Discrete Mathematics	03	-	-	30	70	-	-	-	100	03	--	-	03
210242	Fundamentals of Data Structures	03	-	-	30	70	-	-	-	100	03	-	-	03
210243	Object Oriented Programming (OOP)	03	-	-	30	70	-	-	-	100	03	-	-	03
210244	Computer Graphics	03	-	-	30	70	-	-	-	100	03	-	-	03
210245	Digital Electronics and Logic Design	03	-	-	30	70	-	-	-	100	03	-	-	03
210246	Data Structures Laboratory	-	04	-	-	-	25	50	-	75	-	02	-	02
210247	OOP and Computer Graphics Laboratory	-	04	-	-	-	25	25	-	50	-	02	-	02
210248	Digital Electronics Laboratory	-	02	-	-	-	25	-	-	25	-	01	-	01
210249	Business Communication Skills	-	02	-	-	-	25	-	-	25	-	01	-	01
210250	Humanity and Social Science	-	-	01	-	-	25	-	-	25	-	-	01	01
210251	Audit Course 3													
										Total Credit	15	06	01	22
	Total	**15**	**12**	**01**	**150**	**350**	**125**	**75**	**-**	**700**	**-**	**-**	**-**	**-**

Semester-IV

Course Code	Course Name	Teaching Scheme (Hours/Week)			Examination Scheme and Marks						Credit Scheme			
		Lecture	Practical	Tutorial	Mid-Sem	End-Sem	Term work	Practical	Oral	Total	Lecture	Practical	Tutorial	Total
207003	Engineering Mathematics III	03	-	01	30	70	25	-	-	125	03	--	01	04
210252	Data Structures and Algorithms	03	-	-	30	70	-	-	-	100	03	-	-	03
210253	Software Engineering	03	-	-	30	70	-	-	-	100	03	-	-	03
210254	Microprocessor	03	-	-	30	70	-	-	-	100	03	-	-	03
210255	Principles of Programming Languages	03	-	-	30	70	-	-	-	100	03	-	-	03
210256	Data Structures and Algorithms Laboratory	-	04	-	-	-	25	25	-	50	-	02	-	02
210257	Microprocessor Laboratory	-	02	-	-	-	25	-	25	50	-	01	-	01
210258	Project Based Learning II	-	04	-	-	-	50	-	-	50	-	02	-	02
210259	Code of Conduct	-	-	01	-	-	25	-	-	25	-	-	01	01
210260	Audit Course 4													
										Total Credit	15	05	02	22
	Total	**15**	**10**	**02**	**150**	**350**	**150**	**25**	**25**	**700**	**-**	**-**	**-**	**-**

Home

Savitribai Phule Pune University
Second Year of Computer Engineering (2019 Course)
210256: Data Structures and Algorithms Laboratory

Teaching Scheme	Credit Scheme	Examination Scheme and Marks
Practical: 04 Hours/Week	02	Term Work: 25 Marks Practical: 25 Marks

Companion Course : 210252: Data Structures and Algorithms, 210255:Priciples of Programming Languages

Course Objectives:

- To **understand** practical implementation and usage of non linear data structures for solving problems of different domain.
- To strengthen the ability to identify and **apply** the suitable data structure for the given real world problems.
- To **analyze** advanced data structures including hash table, dictionary, trees, graphs, sorting algorithms and file organization.

Course Outcomes:

On completion of the course, learner will be able to–

CO1: Understand the ADT/libraries, hash tables and dictionary to design algorithms for a specific problem.

CO2: Choose most appropriate data structures and **apply** algorithms for graphical solutions of the problems.

CO3: Apply and **analyze** non linear data structures to solve real world complex problems.

CO4: Apply and **analyze** algorithm design techniques for indexing, sorting, multi-way searching, file organization and compression.

CO5: Analyze the efficiency of most appropriate data structure for creating efficient solutions for engineering design situations.

Guidelines for Instructor's Manual

The instructor‘s manual is to be developed as a hands-on resource and reference. The instructor's manual need to include prologue (about University/program/ institute/ department/foreword/ preface), curriculum of course, conduction and Assessment guidelines, topics under consideration-concept, objectives, outcomes, set of typical applications/assignments/ guidelines, and references.

Guidelines for Student's Laboratory Journal

The laboratory assignments are to be submitted by student in the form of journal. Journal consists of prologue, Certificate, table of contents, and **handwritten write-up** of each assignment (Title, Objectives, Problem Statement, Outcomes, software and Hardware requirements, Date of Completion, Assessment grade/marks and assessor's sign, Theory- Concept in brief, algorithm, flowchart, test cases, Test Data Set(if applicable), mathematical model (if applicable), conclusion/analysis. **Program codes with sample output of all performed assignments are to be submitted as softcopy.**

As a conscious effort and little contribution towards Green IT and environment awareness, attaching printed papers as part of write-ups and program listing to journal may be avoided. Use of DVD containing students programs maintained by Laboratory In-charge is highly encouraged. For reference one or two journals may be maintained with program prints at Laboratory.

Guidelines for Laboratory / Term Work Assessment

Continuous assessment of laboratory work should be done based on overall performance and Laboratory assignments performance of student. Each Laboratory assignment assessment should be assigned grade/marks based on parameters with appropriate weightage. Suggested parameters for overall assessment as well as each Laboratory assignment assessment include- timely completion, performance, innovation, efficient codes, punctuality and neatness.

Guidelines for Laboratory Conduction

The instructor is expected to frame the assignments by understanding the prerequisites, technological aspects, utility and recent trends related to the topic. The assignment framing policy need to address the

average students and inclusive of an element to attract and promote the intelligent students. The instructor may set multiple sets of assignments and distribute among batches of students. It is appreciated if the assignments are based on real world problems/applications. Encourage students for appropriate use of Hungarian notation, proper indentation and comments. Use of open source software is to be encouraged.

In addition to these, instructor may assign one real life application in the form of a mini-project based on the concepts learned. Instructor may also set one assignment or mini-project that is suitable to respective branch beyond the scope of syllabus.

Set of suggested assignment list is provided in groups- A, B, C, D, E, F and G. Each student must perform at least 12 assignments(at least 02 from group A, 03 from group B, 02 from group C, 2 from group D, 01 from group E, 02 from group F.)

Operating System recommended :- 64-bit Open source Linux or its derivative

Programming tools recommended: - Open Source Python - Group A assignments, C++ Programming tool like G++/GCC

Guidelines for Practical Examination

Both internal and external examiners should jointly set problem statements. During practical assessment, the expert evaluator should give the maximum weightage to the satisfactory implementation of the problem statement. The supplementary and relevant questions may be asked at the time of evaluation to test the student's for advanced learning, understanding of the fundamentals, effective and efficient implementation. Consequently encouraging efforts, transparent evaluation and fair approach of the evaluator will not create any uncertainty or doubt in the minds of the students. Therefore adhering to these principles will consummate our team efforts to the promising start of the student's academics.

Virtual Laboratory:

- http://cse01-iiith.vlabs.ac.in/Courses%20Aligned.html?domain=Computer%20Science

Suggested List of Laboratory Experiments/Assignments

Sr. No	Group A
1	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
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)
3	For given set of elements create skip list. Find the element in the set that is closest to some given value. (note: Decide the level of element in the list Randomly with some upper limit)
4	To create ADT that implement the "set" concept. a. Add (new Element) -Place a value into the set , b. Remove (element) Remove the value c. Contains (element) Return true if element is in collection, d. Size () Return number of values in collection Iterator () Return an iterator used to loop over collection, e. Intersection of two sets , f. Union of two sets, g. Difference between two sets, h. Subset
	Group B
5	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.
6	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

7	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.
8	Read for the formulas in propositional calculus. Write a function that reads such a formula and creates its binary tree representation. What is the complexity of your function?
9	Convert given binary tree into threaded binary tree. Analyze time and space complexity of the algorithm.
10	Consider threading a binary tree using preorder threads rather than inorder threads. Design an algorithm for traversal without using stack and analyze its complexity. _
11	A Dictionary stores keywords and its meanings. Provide facility for adding new keywords, deleting keywords, updating values of any entry. Provide facility to display whole data sorted in ascending/ Descending order. Also find how many maximum comparisons may require for finding any keyword. Use Binary Search Tree for implementation.
12	Implement a file compression algorithm that uses binary tree. Your program should allow the user to compress and decompress messages containing alphabets using the standard Huffman algorithm for encoding and decoding.
	Group C
13	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.
14	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.
15	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.
16	Tour operator organizes guided bus trips across the Maharashtra. Tourists may have different preferences. Tour operator offers a choice from many different routes. Every day the bus moves from starting city S to another city F as chosen by client. On this way, the tourists can see the sights alongside the route travelled from S to F. Client may have preference to choose route. There is a restriction on the routes that the tourists may choose from, the bus has to take a short route from S to F or a route having one distance unit longer than the minimal distance. Two routes from S to F are considered different if there is at least one road from a city A to a city B which is part of one route, but not of the other route.
17	Consider the scheduling problem. n tasks to be scheduled on single processor. Let t1, ...,tn be durations required to execute on single processor is known. The tasks can be executed in any order but one task at a time. Design a greedy algorithm for this problem and find a schedule that minimizes the total time spent by all the tasks in the system. (The time spent by one is the sum of the waiting time of task and the time spent on its execution.)
	Group D
18	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?

Home

Home

19	A Dictionary stores keywords and its meanings. Provide facility for adding new keywords, deleting keywords, updating values of any entry. Provide facility to display whole data sorted in ascending/ Descending order. Also find how many maximum comparisons may require for finding any keyword. Use Height balance tree and find the complexity for finding a keyword
	Group E
20	Consider a scenario for Hospital to cater services to different kinds of patients as Serious (top priority), b) non-serious (medium priority), c) General Checkup (Least priority). Implement the priority queue to cater services to the patients.
21	Implement the Heap/Shell sort algorithm implemented in Java demonstrating heap/shell data structure with modularity of programming language
22	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.
	Group F
23	Department maintains a student information. The file contains roll number, name, division and address. Allow user to add, delete information of student. Display information of particular employee. If record of student does not exist an appropriate message is displayed. If it is, then the system displays the student details. Use sequential file to main the data.
24	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.
25	Implementation of a direct access file -Insertion and deletion of a record from a direct access file
26	Assume we have two input and two output tapes to perform the sorting. The internal memory can hold and sort m records at a time. Write a program in java for external sorting. Find out time complexity.
	Mini-Projects/ Case Study
27	Design a mini project using JAVA which will use the different data structure with or without Java collection library and show the use of specific data structure on the efficiency (performance) of the code.
28	Design a mini project to implement Snake and Ladders Game using Python.
29	Design a mini project to implement a Smart text editor.
30	Design a mini project for automated Term work assessment of student based on parameters like daily attendance, Unit Test / Prelim performance, Students achievements if any, Mock Practical.

@The CO-PO Mapping Matrix

PO	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12
CO1	1	2	2	-	-	-	-	-	-	-	-	-
CO2	-	2	2	-	-	-	-	-	-	-	-	-
CO3	-	2	2	1	-	-	-	-	-	-	-	-
CO4	1	2	1	1	-	-	-	-	-	-	-	-
CO5	1	1	2	2	-	-	-	-	-	-	-	-

ASSIGNMENT NO.1

Consider telephone book database of N clients. Make use of a hash table implementation to quicklylook up client_s telephone number

Learning Objectives:

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

Learning Outcome

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

Theory:

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

Keyed Arrays vs. Indexed Arrays

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

employees [50];

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

employees ["Brown, John"];

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

Hashing Functions

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

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

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

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

A --> 0
B --> 1
C --> 2
D --> 3
...
and so on until Z --> 25.

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

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

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

Basic Operations

Following are the basic primary operations of a hash table.

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

DataItem

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

```
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;
}
```

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 ifthe 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;}
```

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;
}
```

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;
   }
```

Expected Output: Menu driver output for inserting phone number, display and look up function.

e.g

```
  Menu
1.Create Telephone book
2.Display
3. Look up
```

Program Code:

```
#include<iostream>
using namespace std;
 struct node{
 int value;
 node* next;
}*HashTable[10];
class hashing{
public:

 hashing(){

  for(int i=0 ; i<10 ; i++){
   HashTable[i]=NULL;
  }
 }


 int HashFunction(int value){
  return (value%10);
 }

 node* create_node(int x){
  node* temp=new node;
  temp->next=NULL;
  temp->value=x;
```

```
 return temp;
}

void display(){
 for(int i=0 ; i< 10; i++){
  node * temp=new node;
  temp=HashTable[i];
  cout<<"a["<<i<<"] : ";
  while(temp !=NULL){
   cout<<" ->"<<temp->value;
   temp=temp->next;
  }
  cout<<"\n";
 }
}


int searchElement(int value){
 bool flag = false;
            int hash_val = HashFunction(value);
            node* entry = HashTable[hash_val];
            cout<<"\nElement found at : ";
            while (entry != NULL)
     {
                if (entry->value==value)
         {
                    cout<<hash_val<<" : "<<entry->value<<endl;
                    flag = true;
                }
                entry = entry->next;
            }
            if (!flag)
                return -1;
}

void deleteElement(int value){
 int hash_val = HashFunction(value);
            node* entry = HashTable[hash_val];

            if (entry == NULL )
            {
             cout<<"No Element found ";
                return;
            }

            if(entry->value==value){
             HashTable[hash_val]=entry->next;
             return;
```

```
            }
            while ((entry->next)->value != value)
        {
                entry = entry->next;
            }
            entry->next=(entry->next)->next;


 }

 void insertElement(int value){
  int hash_val = HashFunction(value);
            // node* prev = NULL;
             //node* entry = HashTable[hash_val];
             node* temp=new node;
             node* head=new node;
             head = create_node(value);
             temp=HashTable[hash_val];
             if (temp == NULL)
                        {

                           HashTable[hash_val] =head;
                            }
             else{
              while (temp->next != NULL)

             {
                temp = temp->next;
             }

                    temp->next =head;

             }


  }
 };

 int main(){
  int ch;
  int data,search,del;
  hashing h;
  do{
   cout<<"\nTelephone : \n1.Insert \n2.Display \n3.Search \n4.Delete \n5.Exit";
   cin>>ch;
   switch(ch){
   case 1:cout<<"\nEnter phone no. to be inserted : ";
     cin>>data;
```

```
    h.insertElement(data);
    break;
  case 2:h.display();
    break;
  case 3:cout<<"\nEnter the no to be searched : ";
    cin>>search;

    if (h.searchElement(search) == -1)
          {
        cout<<"No element found at key ";
        continue;
          }
    break;
  case 4:cout<<"\nEnter the phno. to be deleted : ";
    cin>>del;
    h.deleteElement(del);
    cout<<"Phno. Deleted"<<endl;
    break;
  }
 }while(ch!=5);
 return 0;
}
```

Output:

```
Telephone :
1.Insert
2.Display
3.Search
4.Delete
5.Exit
1
Enter phone no. to be inserted : 34555

Telephone :
1.Insert
2.Display
3.Search
4.Delete
5.Exit
2
a[0] :
a[1] :
a[2] :
a[3] :
a[4] :
a[5] :  ->34555
a[6] :
```

a[7] :
a[8] :
a[9] :

Telephone :
1.Insert
2.Display
3.Search
4.Delete
5.Exit
3

Enter the no to be searched : 34555

Element found at : 5 : 34555

Telephone :
1.Insert
2.Display
3.Search
4.Delete
5.Exit
4

Enter the phno. to be deleted : 34555
Phno. Deleted

Telephone :
1.Insert
2.Display
3.Search
4.Delete
5.Exit

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

Assignment -2

Implement all the functions of a dictionary (ADT) using hashing.

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)

Learning Objectives:

- ✓ To understand Dictionary (ADT)
- ✓ To understand concept of hashing
- ✓ To understand concept & features like searching using hash function.

Learning 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 **k**or null, if dictionary contains no such key

- **remove(Dictionary d, Key k)**

  removes key **k** and associated value

- **destroy(Dictionary d)**

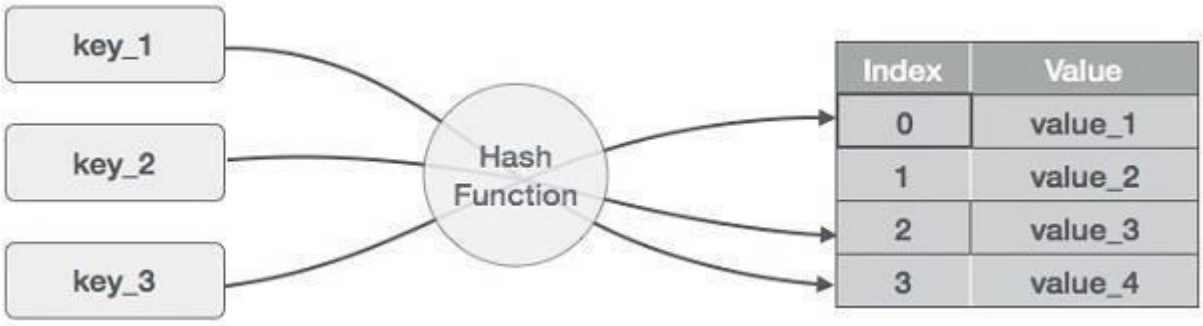
  destroys dictionary **d**

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. DataItem

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;
};
```

2. 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;
}
```

3. 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;
}
```

4. 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;
}
```

5. 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;
}
```

Expected Output: Create dictionary using hash table and search the elements in table.

Program code:

```
#include<iostream>
  #include<string.h>
  using namespace std;
  class HashFunction
   {
      typedef struct hash
   {
    long key;
    char name[10];
   }hash;
   hash h[10];
     public:
   HashFunction();
   void insert();
   void display();
   int find(long);
   void Delete(long);
   };

  HashFunction::HashFunction()
    {
   int i;
   for(i=0;i<10;i++)
     {
    h[i].key=-1;
    strcpy(h[i].name,"NULL");
     }
    }
```

```
void HashFunction::Delete(long k)
  {
 int index=find(k);
 if(index==-1)
   {
  cout<<"\n\tKey Not Found";
   }
 else
   {
  h[index].key=-1;
  strcpy(h[index].name,"NULL");
  cout<<"\n\tKey is Deleted";
   }


  }
int HashFunction::find(long k)
  {
 int i;
 for(i=0;i<10;i++)
   {
  if(h[i].key==k)
    {
   cout<<"\n\t"<<h[i].key<<" is Found at
"<<i<<" Location With Name "<<h[i].name;
   return i;
    }
   }
 if(i==10)
     {
  return -1;
   }

  }
void HashFunction::display()
  {
 int i;
 cout<<"\n\t\tKey\t\tName";
 for(i=0;i<10;i++)
    {

cout<<"\n\th["<<i<<"]\t"<<h[i].key<<"\t\t"<<
h[i].name;
   }
  }
void HashFunction::insert()
  {
 char ans,n[10],ntemp[10];
```

```
long k,temp;
int v,hi,cnt=0,flag=0,i;

do
  {
 if(cnt>=10)
   {
  cout<<"\n\tHash Table is FULL";
  break;
   }
 cout<<"\n\tEnter a Telephone No: ";
 cin>>k;
 cout<<"\n\tEnter a Client Name: ";
 cin>>n;
 hi=k%10;// hash function
 if(h[hi].key==-1)
   {
 h[hi].key=k;
 strcpy(h[hi].name,n);
  }
    else
   {

 if(h[hi].key%10!=hi)
   {
  temp=h[hi].key;
  strcpy(ntemp,h[hi].name);
  h[hi].key=k;
  strcpy(h[hi].name,n);
  for(i=hi+1;i<10;i++)
     {
   if(h[i].key==-1)
      {
    h[i].key=temp;
    strcpy(h[i].name,ntemp);
    flag=1;
    break;
       }
     }
  for(i=0;i<hi && flag==0;i++)
     {
   if(h[i].key==-1)
      {
    h[i].key=temp;
    strcpy(h[i].name,ntemp);
    break;
       }
         }
```

```
      }
   else
    {
    for(i=hi+1;i<10;i++)
       {
     if(h[i].key==-1)
        {
      h[i].key=k;
      strcpy(h[i].name,n);
      flag=1;
      break;
         }
       }
    for(i=0;i<hi && flag==0;i++)
       {
     if(h[i].key==-1)
        {
      h[i].key=k;
      strcpy(h[i].name,n);
      break;
         }
         }
     }

    }
      flag=0;
      cnt++;
      cout<<"\n\t..... Do You Want to Insert
More Key: y/n";
      cin>>ans;
   }while(ans=='y'||ans=='Y');

  }



int main()
  {
 long k;
 int ch,index;
 char ans;
 HashFunction obj;
 do
   {
  cout<<"\n\t***** Telephone (ADT) *****";
  cout<<"\n\t1. Insert\n\t2. Display\n\t3.
Find\n\t4. Delete\n\t5. Exit";
  cout<<"\n\t..... Enter Your Choice: ";
```

```
  cin>>ch;
  switch(ch)
    {
   case 1:  obj.insert();
     break;
   case 2: obj.display();
     break;
   case 3: cout<<"\n\tEnter a Key Which You
 Want to Search: ";
     cin>>k;
     index=obj.find(k);
     if(index==-1)
       {
      cout<<"\n\tKey Not Found";
       }
     break;
   case 4: cout<<"\n\tEnter a Key Which You
 Want to Delete: ";
     cin>>k;
     obj.Delete(k);
     break;
   case 5:
     break;
    }
  cout<<"\n\t..... Do You Want to Continue in
 Main Menu:y/n ";
  cin>>ans;
   }while(ans=='y'||ans=='Y');
  }
```

Output:

***** Telephone (ADT) *****

1. Insert
2. Display
3. Find
4. Delete
5. Exit

..... Enter Your Choice: 1

Enter a Telephone No: 7666

Enter a Client Name: abc

..... Do You Want to Insert More Key: y/n

n

..... Do You Want to Continue in Main Menu:y/n

y

***** Telephone (ADT) *****

1. Insert
2. Display
3. Find
4. Delete
5. Exit

..... Enter Your Choice: 2

	Key	Name
h[0]	-1	NULL
h[1]	-1	NULL
h[2]	-1	NULL
h[3]	-1	NULL
h[4]	-1	NULL
h[5]	-1	NULL
h[6]	7666	abc
h[7]	-1	NULL
h[8]	-1	NULL
h[9]	-1	NULL

..... Do You Want to Continue in Main Menu:y/n

y

***** Telephone (ADT) *****

1. Insert
2. Display
3. Find
4. Delete
5. Exit

..... Enter Your Choice: 3

```
Enter a Key Which You Want to Search: 7666

7666 is Found at 6 Location With Name abc
..... Do You Want to Continue in Main Menu:y/n
y

***** Telephone (ADT) *****
1. Insert
2. Display
3. Find
4. Delete
5. Exit
..... Enter Your Choice: 4

Enter a Key Which You Want to Delete: 7666

7666 is Found at 6 Location With Name abc
Key is Deleted
..... Do You Want to Continue in Main Menu:y/n
n
```

Conclusion: This program gives us the knowledge of dictionary (ADT).

Assignment -3

Title:

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.

Learning Objectives:

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

Learning Outcome:

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

Theory:

Introduction to Tree:

Definition:

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

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

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

Recursive definition

• T is either empty.

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

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

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

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

In some trees, such as heaps, the root node has special properties.

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

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

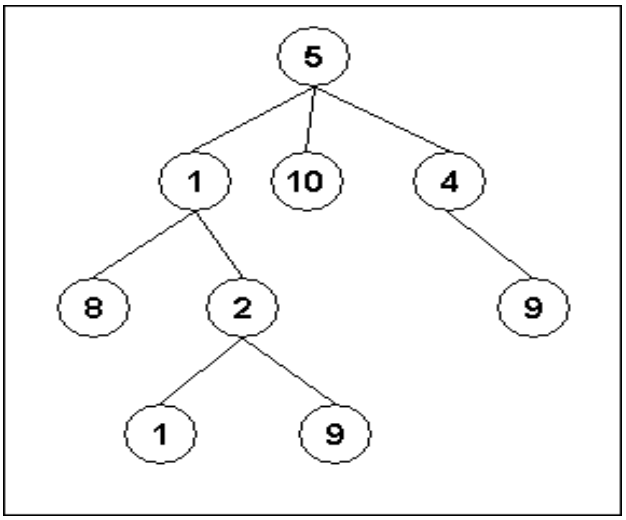


Fig1. An example of a tree

An internal node or inner node is any node of a tree that has child nodes and is thus not a leaf node.

There are two basic types of trees. In an unordered tree, a tree is a tree in a purely structural sense — that is to say, given a node, there is no order for the children of that node. A tree on which an order is imposed — for example, by assigning different natural numbers to eachchild of each node — is called an ordered tree, and data structures built on them are called ordered tree data structures. Ordered trees are by far the most common form of tree datastructure. Binary search trees are one kind of ordered tree.

Important Terms

Following are the important terms with respect to tree.

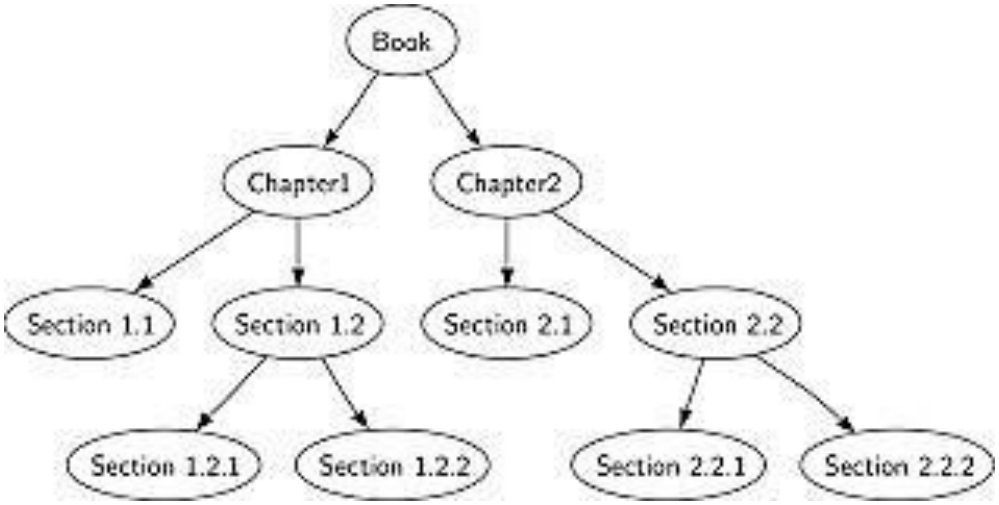
- **Path** − Path refers to the sequence of nodes along the edges of a tree.
- **Root** − The node at the top of the tree is called root. There is only one root per tree andone path from the root node to any node.
- **Parent** − Any node except the root node has one edge upward to a node called parent.
- **Child** − The node below a given node connected by its edge downward is called its childnode.
- **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 level0, 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 effortFor

  this assignment we are considering the tree as follows.



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

Program Code:

```
#include<iostream>
#include<stdlib.h>
#include<string.h>
using namespace std;
struct node
{   char name[20];
    node *next;
    node *down;
     int flag;
};
class Gll
{   char ch[20];    int n,i;
    node *head=NULL,*temp=NULL,*t1=NULL,*t2=NULL;
    public:
    node *create();
    void insertb();
    void insertc();
    void inserts();
    void insertss();
    void displayb();

};
node *Gll::create()
{
   node *p=new(struct node);
   p->next=NULL;
   p->down=NULL;
```

```
   p->flag=0;
   cout<<"\n enter the name";
   cin>>p->name;
   return p;
}
void Gll::insertb()
{

        if(head==NULL)
         {    t1=create();
              head=t1;
         }
         else
         {
           cout<<"\n book exist";
         }
}
void Gll::insertc()
{
     if(head==NULL)
        {
              cout<<"\n there is no book";
         }
         else
         {    cout<<"\n how many chapters you want to insert";
              cin>>n;
              for(i=0;i<n;i++)
              {
              t1=create();
              if(head->flag==0)
              {  head->down=t1;   head->flag=1;    }
              else
              {   temp=head;
                  temp=temp->down;
                 while(temp->next!=NULL)
                     temp=temp->next;
                  temp->next=t1;
              }
              }
        }


}
void Gll::inserts()
{

        if(head==NULL)
         {
```

```
            cout<<"\n there is no book";
        }
        else
        {    cout<<"\n Enter the name of chapter on which  you want to enter the section";
             cin>>ch;

             temp=head;
           if(temp->flag==0)
           {   cout<<"\n their are no chapters on in book";
           }
           else
           {    temp=temp->down;
            while(temp!=NULL)
             {
                 if(!strcmp(ch,temp->name))
                 {
                         cout<<"\n how many sections you want to enter";
                         cin>>n;
                         for(i=0;i<n;i++)
                         {

                                 t1=create();
                                    if(temp->flag==0)
                                    {      temp->down=t1;

                                           temp->flag=1;  cout<<"\n******";
                                           t2=temp->down;

                                    }
                                   else
                                    {
                                                cout<<"\n#####";
                                                 while(t2->next!=NULL)
                                                 {     t2=t2->next;         }
                                                       t2->next=t1;
                                      }
                          }
                            break;
                 }
                        temp=temp->next;
             }
            }
       }
}
void Gll::insertss()
{

       if(head==NULL)
```

```
{
      cout<<"\n there is no book";
 }
 else
 {    cout<<"\n Enter the name of chapter on which  you want to enter the section";
      cin>>ch;

      temp=head;
    if(temp->flag==0)
    {   cout<<"\n their are no chapters on in book";
    }
    else
    {    temp=temp->down;
     while(temp!=NULL)
      {
          if(!strcmp(ch,temp->name))
          {
            cout<<"\n enter name of section in which you want to enter the sub section";
            cin>>ch;

           if(temp->flag==0)
           {   cout<<"\n their are no sections ";   }
            else
            {       temp=temp->down;
                   while(temp!=NULL)
                   {
                      if(!strcmp(ch,temp->name))
                      {
                       cout<<"\n how many subsections you want to enter";
                         cin>>n;
        for(i=0;i<n;i++)
                    {

                          t1=create();
                             if(temp->flag==0)
                             {      temp->down=t1;

                                    temp->flag=1;  cout<<"\n******";
                                    t2=temp->down;

                             }
                            else
                             {
                                       cout<<"\n#####";
                                        while(t2->next!=NULL)
                                        {     t2=t2->next;         }
                                              t2->next=t1;
                               }
```

```
                        }
                        break;
                    }     temp=temp->next;
                   }
                }
              }

                  temp=temp->next;
            }
           }
      }
}
void Gll::displayb()
{

          if(head==NULL)
          {  cout<<"\n book not exist";
          }
          else
          {
           temp=head;

             cout<<"\n NAME OF BOOK:  "<<temp->name;
                if(temp->flag==1)
                {
                temp=temp->down;

                  while(temp!=NULL)
                  {     cout<<"\n\t\tNAME OF CHAPTER:  "<<temp->name;
                       t1=temp;
                       if(t1->flag==1)
                       {  t1=t1->down;
                          while(t1!=NULL)
                          {     cout<<"\n\t\t\t\tNAME OF SECTION:  "<<t1->name;
                               t2=t1;
                               if(t2->flag==1)
                               {  t2=t2->down;
                               while(t2!=NULL)
                               {     cout<<"\n\t\t\t\t\t\tNAME OF SUBSECTION:  "<<t2->name;
                               t2=t2->next;
                               }
                               }
                               t1=t1->next;
                          }
                       }
                        temp=temp->next;
                   }
```

```
                }
            }



}
int main()
{    Gll g;   int x;
      while(1)
     {    cout<<"\n\n enter your choice";
          cout<<"\n 1.insert book";
          cout<<"\n 2.insert chapter";
          cout<<"\n 3.insert section";
          cout<<"\n 4.insert subsection";
          cout<<"\n 5.display book";
          cout<<"\n 6.exit";
          cin>>x;
         switch(x)
         {   case 1:         g.insertb();
                                break;
             case 2:         g.insertc();
                                break;
             case 3:         g.inserts();
                                break;
             case 4:         g.insertss();
                                break;
             case 5:         g.displayb();
                                break;
             case 6:  exit(0);
         }
      }
      return 0;
}
```

Output:

```
enter your choice
 1.insert book
 2.insert chapter
 3.insert section
 4.insert subsection
 5.display book
 6.exit
1

 enter the name DSA
```

enter your choice
1.insert book
2.insert chapter
3.insert section
4.insert subsection
5.display book
6.exit
2

how many chapters you want to insert 1

enter the name Hashing

enter your choice
1.insert book
2.insert chapter
3.insert section
4.insert subsection
5.display book
6.exit
3

Enter the name of chapter on which you want to enter the section 1
enter your choice
1.insert book
2.insert chapter
3.insert section
4.insert subsection
5.display book
6.exit
4

Enter the name of chapter on which you want to enter the section 1

enter your choice
1.insert book
2.insert chapter
3.insert section
4.insert subsection
5.display book
6.exit
5

NAME OF BOOK: DSA
NAME OF CHAPTER: Hashing

enter your choice
1.insert book
2.insert chapter
3.insert section
4.insert subsection
5.display book
6.exit
6

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

Questions asked in university exam.

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

Assignment –4

Beginning with an empty binary search tree, Construct binary searchtree 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
iii. Minimum data value found in the tree
iv. Change a tree so that the roles of the left and right pointers are swapped atevery node
v. Search a value

Learning Objective:

- ✓ To understand the basic concept of Non-Linear Data Structure.
- ✓ –TREE‖ and its basicoperation in Data structure.

Learning Outcome:

- ✓ To implement the basic concept of Binary Search Tree to store a number in it.
- ✓ To perform basic Operation Insert, Delete and search, Traverse in tree in Data structure.

Theory –

Tree represents the nodes connected by edges. **Binary Tree** is a special data structure used for data storage purposes. A binary tree has a special condition that each node can have a maximum of two children. A binary tree has the benefits of both an ordered array and a linked list as searchis as quick as in a sorted array and insertion or deletion operation are as fast as in linked list.

Binary Search Tree Representation

Binary Search tree exhibits a special behavior. A node's left child must have a value less than its parent's value and the node's right child must have a value greater than its parent value.

A Binary Search Tree (BST) is a tree in which all the nodes follow the below-mentioned properties −

- The left sub-tree of a node has a key less than or equal to its parent node's key.
- The right sub-tree of a node has a key greater than or equal to its parent node's key.

Thus, BST divides all its sub-trees into two segments;the left sub-tree and the right sub-tree and can be defined as –

$$\text{left\_subtree (keys)} \leq \text{node (key)} \leq \text{right\_subtree (keys)}$$

Tree Node

Following Structure is used for Node creation

Struct node {Int data ;
Struct node *leftChild; Struct node *rightChild;
};

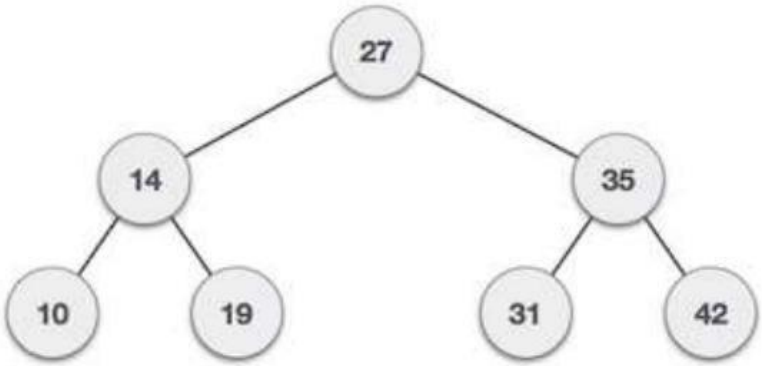


Fig: Binary

Search TreeBST Basic Operations

The basic operations that can be performed on a binary search tree data structure, are the following −

- **Insert**− Inserts an element in a tree/create a tree.
- **Search**− Searches an element in a tree.
- **Traversal**− A traversal is a systematic way to visit all nodes of T -Inorder, Preprder, Postorder,

a. pre-order: Root, Left, Right

Parent comes before children; overall root firstB.post-order: Left, Right, Root

Parent comes after children; overall root last

c. In Order: In-order: Left, Root, Right,

Insert Operation: Algorithm

```
If root is NULL
   then create root node
return

If root exists then
   compare the data with node.data

   while until insertion position is located

      If data is greater than node.data
         goto right subtree
      else
         goto left subtree
```

Search Operation:Algorithm

```
If root.data is equal to search.data
   return root
else
   while data not found

      If data is greater than node.data
         goto right subtree
      else
         goto left subtree

      If data found
         return node

   endwhile
```

Tree Traversal

In order traversal algorithm

```
Until all nodes are traversed -
Step 1 - Recursively traverse left subtree.
Step 2 - Visit root node.
Step 3 - Recursively traverse right subtree.
```

Pre order traversal Algorithm

```
Until all nodes are traversed -
Step 1 - Visit root node.
Step 2 - Recursively traverse left subtree.
Step 3 - Recursively traverse right subtree.
```

Post order traversal Algorithm

```
Until all nodes are traversed -
Step 1 - Recursively traverse left subtree.
Step 2 - Recursively traverse right subtree.
Step 3 - Visit root node.
```

Deleting in a BST

case 1: delete a node with zero child-
if x is left of its parent, set parent(x).left
= nullelse set parent(x).right = null
case 2: delete a node with one child
link parent(x) to the child of x.
case 3: delete a node with 2 children
Replace inorder successor to deleted node position.

Expected Output:

Formation of binary search tree structure with its basic Operation Insert, Delete and search, Traverse in tree.

Program Code:

```
#include<iostream>
#include<stdlib.h>
using namespace std;
struct node
{  int a;
   node *left,*right;
 };
class Bt
{
    node *root=NULL,*temp=NULL,*t1=NULL,*s=NULL, *t=NULL;
    int count;
    public:
    Bt(){  count=0;    }
    node *create();
    void insert();
    void del();
    node *delet(node*,int);
    void find();
    void search();
    void sw();
    void swap(node*);
    void height();
    int he(node*,int);
    void disp(node*);
    void display();
    node *findmin(node*);
```

```
};
node *Bt::create()
{
    node *p=new(struct node);
    p->left=NULL;
    p->right=NULL;
    cout<<"\n enter the data";
    cin>>p->a;
    return p;
}
void Bt::insert()
{
    temp=create();
    if(root==NULL)
    {      root=temp;       }
    else
    {    t1=root;
        while(t1!=NULL)
        {      s=t1;
               if((temp->a)>(t1->a))
               {     t1=t1->right;    }
               else
               {     t1=t1->left;     }
        }
           if((temp->a)>(s->a))
           {     s->right=temp;    }
           else
           {   s->left=temp;       }
     }
}
void Bt::find()
{
   if(root==NULL)
   {    cout<<"\n tree not exist";  }
   else
   {
       t1=root;

   while(t1->left!=NULL)
   {
            t1=t1->left;
   }

   cout<<"\n smallest no."<<t1->a;
   t1=root;

   while(t1->right!=NULL)
```

```
    {
            t1=t1->right;
    }
    cout<<"\n largest no."<<t1->a;
    }
}
void Bt::search()
{
     int m,f=0;
     if(root==NULL)
     {  cout<<"\n tree not exist";
     }
     else
     {
          cout<<"\n enter data to be searched";
          cin>>m;
       if(root->a==m)
       {   cout<<"\ndata found";   }
       else
       {    t1=root;
           while(t1->a!=m)
           {
              if((m)>(t1->a))
              {     t1=t1->right;  }
              else
              {   t1=t1->left;       }
              if(t1==NULL)
              {   cout<<"\n data not found";     f=1;
                 break;
              }
           }
         if(f==0)
         {   cout<<"\n data found";    }

       }
     }
}
 void Bt::sw()
 {
    if(root==NULL)
    {  cout<<"\n tree not exist";
    }
    else
    {
      swap(root);
    }
}
```

```
void Bt::swap(node *q)
{
  if(q->left!=NULL)
  swap(q->left);
  if(q->right!=NULL)
  swap(q->right);
  t=q->left;
  q->left=q->right;
  q->right=t;
}
void Bt::height()
{
   count=0;
   if(root==NULL)
   {  cout<<"\n tree not exist";
   }
   else
   {
   he(root,0); cout<<"\n height of the tree is"<<count;
   }

}
int Bt::he(node *q,int c)  // he is a function that will be used to calculate height of the tree. Can be called using root and counter intilized to 0
{
     c++;
    // cout<<"\n*"<<q->a<<"*"<<c<<"*\n";
        if(q->left!=NULL)
     {     he(q->left,c);
     }
        if(q->right!=NULL)
        {
             he(q->right,c);
        }
        if(count<c)
        {
             count=c;
        }


        return 0;

}
void Bt::del()
{    int x;
  cout<<"\n enter data to be deleted";
  cin>>x;
```

```
   delet(root,x);

}
node *Bt::delet(node *T,int x)
{
    if(T==NULL)
    {
      cout<<"\n element not found";
      return(T);
    }
    if(x<T->a)
    {
       T->left=delet(T->left,x);
       return (T);
    }
    if(x>T->a)
    {
        T->right=delet(T->right,x);
        return T;
    }
    if(T->left==NULL&&T->right==NULL)
    {
      temp=T;
      free(temp);
      return(NULL);
    }
    if(T->left==NULL)
    {
       temp=T;
       T=T->right;
       delete temp;
       return T;
    }
    if(T->right==NULL)
    {
      temp=T;
      T=T->left;
      delete temp;
      return T;
     }
     temp=findmin(T->right);
     T->a=temp->a;
     T->right=delet(T->right,temp->a);
     return T;
}
node *Bt::findmin(node *T)
{
```

```
   while(T->left!=NULL)
   {  T=T->left;    }
   return T;
}


void Bt::display()
{

   if(root==NULL)
   {  cout<<"\n tree not exist";
   }
   else
   {
     disp(root);
   }

}
void Bt::disp(node *q)
{
     cout<<"\n*"<<q->a;
        if(q->left!=NULL)
     {      disp(q->left);
     }
        if(q->right!=NULL)
        {
             disp(q->right);
        }

 }
int main()
{
    Bt b;  int x;     char ch;
    while(1)
    {
            cout<<"\n enter your choice";
            cout<<"\n 1.insert";
            cout<<"\n 2.find";
            cout<<"\n 3.search";
            cout<<"\n 4.swap";
            cout<<"\n 5.height";
            cout<<"\n 6.delete";
            cout<<"\n 7.display";
            cout<<"\n 8.exit";
            cin>>x;
            switch(x)
            {     case 1: b.insert();
```

```
                    break;
            case 2: b.find();
                    break;
            case 3: b.search();
                    break;

            case 4: b.sw();
                    break;
            case 5: b.height();
                    break;
            case 6: b.del();
                    break;
            case 7: b.display();
                    break;
            case 8:exit(0);
        }

    }

  return 0;
}
```

Output:

```
enter your choice
 1.insert
 2.find
 3.search
 4.swap
 5.height
 6.delete
 7.display
 8.exit
1

 enter the data 86

 enter your choice
 1.insert
 2.find
 3.search
 4.swap
 5.height
 6.delete
 7.display
 8.exit
```

1

enter the data 92

enter your choice
1.insert
2.find
3.search
4.swap
5.height
6.delete
7.display
8.exit
1

enter the data 48

enter your choice
1.insert
2.find
3.search
4.swap
5.height
6.delete
7.display
8.exit
1

enter the data 20

enter your choice
1.insert
2.find
3.search
4.swap
5.height
6.delete
7.display
8.exit
1

enter the data 77

enter your choice

1.insert
2.find
3.search
4.swap
5.height
6.delete
7.display
8.exit
2

smallest no.20
largest no.92
enter your choice
1.insert
2.find
3.search
4.swap
5.height
6.delete
7.display
8.exit
3

enter data to be searched 77

data found
enter your choice
1.insert
2.find
3.search
4.swap
5.height
6.delete
7.display
8.exit
5

height of the tree is3
enter your choice
1.insert
2.find
3.search
4.swap
5.height

```
 6.delete
 7.display
 8.exit
7

*86
*48
*20
*77
*92
 enter your choice
 1.insert
 2.find
 3.search
 4.swap
 5.height
 6.delete
 7.display
 8.exit
8
```

Conclusion: This program gives us the knowledge binary search tree data structure and its basic operations.

Assignment -5

Title:

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

Learning Objective:

To understand the basic concept of Non-Linear Data Structure –threaded binary tree‖ and its use in Data structure.

Learning Outcome:

To convert the Binary Tree into threaded binary tree and analyze its time and space complexity.

Theory –

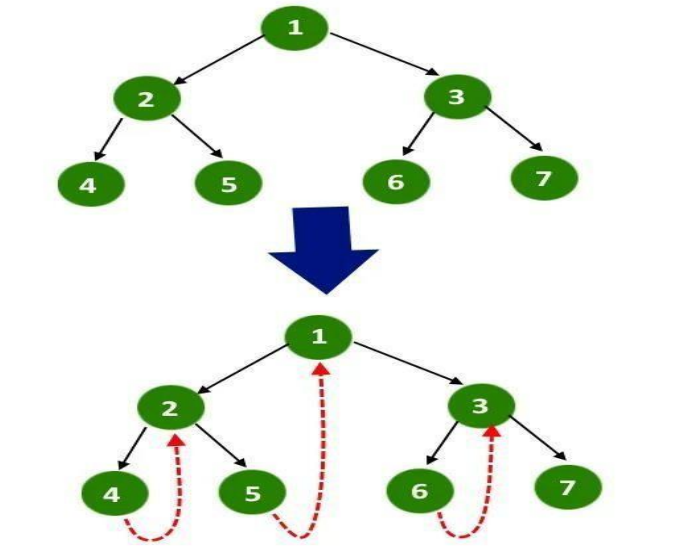
Threaded Binary Tree

Inorder traversal of a Binary tree is either be done using recursion or with the use of a auxiliary stack. The idea of threaded binary trees is to make inorder traversal faster and do it without stack and without recursion. A binary tree is made threaded by making all right child pointers that would normally be NULL point to the inorder successor of the node (if it exists).

There are two types of threaded binary tree.

Single Threaded:
Where a NULL right pointer is made to point to the inorder successor (if successor exists)

Double Threaded:
Where both left and right NULL pointers are made to point to inorder predecessor and inorder successor, respectively. The predecessor threads are useful for reverse inorder traversal and postorder traversal.
The threads are also useful for fast accessing ancestors of a node.
Following diagram shows an example Single Threaded Binary Tree. The dotted lines represent threads.



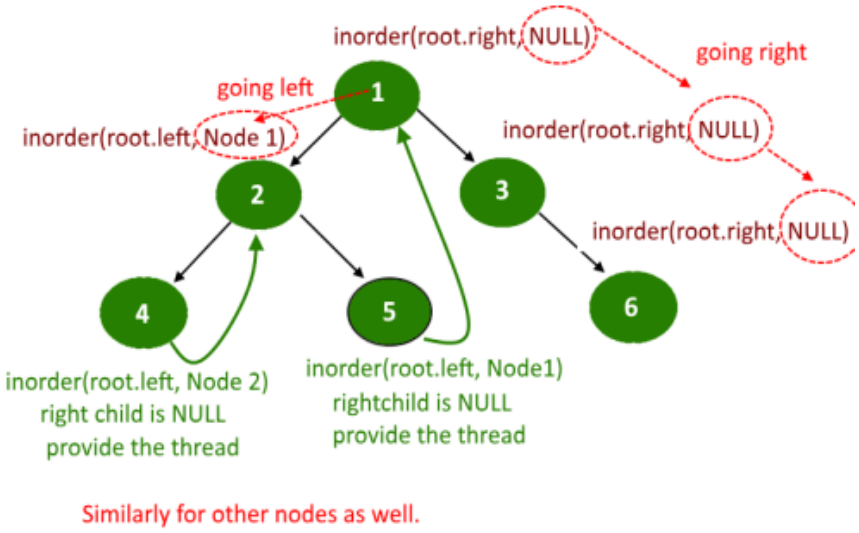
Convert Binary Tree into Threaded Binary Tree

C representation of a Threaded Node
Following is C representation of a single threaded node.

```
struct Node
{
int data;
Node *left, *right;bool rightThread;
}
```

Conversion of Binary Tree to Threaded Binary Tree

- ✓ Do the reverse inorder traversal, means visit right child first.
- ✓ In recursive call, pass additional parameter, the node visited previously.
- ✓ whenever you will find a node whose right pointer is NULL and previous visited node is not NULL then make the right of node points to previous visited node and mark the boolean right Threaded as true.
- ✓ Whenever making a recursive call to right subtree, do not change the previous visited not and when making a recursive call to left subtree then pass the actual previous visited node.



Expected Outcome:

The binary tree into threaded binary tree in one single traversal with no extra space required.

Program code:

```
#include<iostream>
#include<stdlib.h>
using namespace std;
struct node
{
    int data;
    node *left,*right;
    int lbit,rbit;
};
class tbt
{
  node *temp=NULL,*t1=NULL,*s=NULL,*head=NULL,*t=NULL;
  public:

  node *create();
  void insert();
  node *insuc(node*);
  node *inpre(node*);
  void dis();
  void display(node*);
  void thr();
  void thread(node*);
};
node *tbt::create()
{
  node *p=new(struct node);
```

```
  p->left=NULL;
  p->right=NULL;
  p->lbit=0;
  p->rbit=0;
  cout<<"\n enter the data";
  cin>>p->data;
  return p;
}
void tbt::insert()
{
   temp=create();
   if(head==NULL)
   {  node *p=new(struct node);
      head=p;
      head->left=temp;
      head->right=head;
      head->lbit=1;
      head->rbit=0;
      temp->left=head;
      temp->right=head;
      temp->lbit=0;
      temp->rbit=0;
   }
   else
   {     t1=head;
         t1=t1->left;

         while(t1!=NULL)
         {    s=t1;
            if(((temp->data)>(t1->data))&&t1->rbit==1)
            {   t1=t1->right;   }
            else if(((temp->data)<(t1->data))&&t1->lbit==1)
            {   t1=t1->left;     }
            else
            { break;       }
         }
         if(temp->data>s->data)
         {
           s->right=temp;
           s->rbit=1;
           temp->left=inpre(head->left);
           temp->right=insuc(head->left);
         }
         else
         {
           s->left=temp;
           s->lbit=1;
```

```
            temp->left=inpre(head->left);
            temp->right=insuc(head->left);
          }
     }
}
node *tbt::inpre(node *m)
{
   if(m->lbit==1)
   {
     inpre(m->left);
   }
   if(m->data==temp->data&&t==NULL)
   {  return head;       }
   if(m->data==temp->data)
   {    return t;         }
   t=m;
   if(m->rbit==1)
   { inpre(m->right);
   }

}
node *tbt::insuc(node *m)
{
   if(m->lbit==1)
   {  t=m;
     insuc(m->left);
   }

   if(m->data==temp->data&&t==NULL)
   {  return head;       }
   if(m->data==temp->data)
   {    return t;        }

   if(m->rbit==1)
   { insuc(m->right);
   }
}
void tbt::dis()
{   display(head->left);
}
void tbt::display(node *m)
{

      if(m->lbit==1)
      {    display(m->left);              }
      cout<<"\n"<<m->data;
      if(m->rbit==1)
```

```
      {    display(m->right);            }



}
void tbt::thr()
{  cout<<"\n thread are";
   thread(head->left);
}
void tbt::thread(node *m)
{

      if(m->lbit==1)
      {   thread(m->left);             }
      if(m->lbit==0||m->rbit==0)
      {
      cout<<"\n"<<m->data;
      }
      if(m->rbit==1)
      {    thread(m->right);            }
      }
int main()
{  tbt t;  int ch;
   while(1)
   {

   cout<<"\n enter the choice";
   cout<<"\n 1.insert data";
   cout<<"\n 2.display all data";
   cout<<"\n 3.display threaded node";
   cout<<"\n 4.exit";
   cin>>ch;
   switch(ch)
   {
     case 1:
              t.insert();
              break;
     case 2:
              t.dis();
              break;
     case 3:
              t.thr();
              break;
     case 4:  exit(0);

     default:
            cout<<"\n invalid entry";
```

```
    }
    }
   return 0;
}
```

Output:

```
enter the choice
 1.insert data
 2.display all data
 3.display threaded node
 4.exit
1

 enter the data 66

 enter the choice
 1.insert data
 2.display all data
 3.display threaded node
 4.exit
1

 enter the data 99

 enter the choice
 1.insert data
 2.display all data
 3.display threaded node
 4.exit
1

 enter the data 38

 enter the choice
 1.insert data
 2.display all data
 3.display threaded node
 4.exit
1

 enter the data 72

 enter the choice
 1.insert data
 2.display all data
 3.display threaded node
 4.exit
```

```
2

38
66
72
99
 enter the choice
 1.insert data
 2.display all data
 3.display threaded node
 4.exit
3

 thread are
38
72
99
 enter the choice
 1.insert data
 2.display all data
 3.display threaded node
 4.exit
4
```

Conclusion: Able to convert the Binary Tree into threaded binary tree and analyze its time and space complexity.

Assignment -6

Title:

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

Learning Objectives:

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

Learning Outcome:

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

Theory:

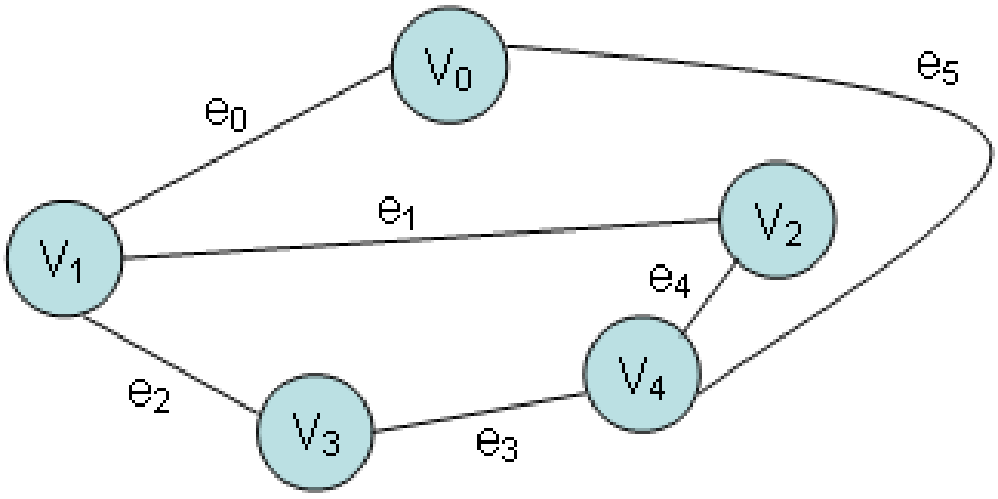
Graphs are the most general data structure. They are also commonly used data structures.

Graph definitions:

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

Undirected graph definition:

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



Graph Implementation:

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

Representing Graphs with an Adjacency Matrix

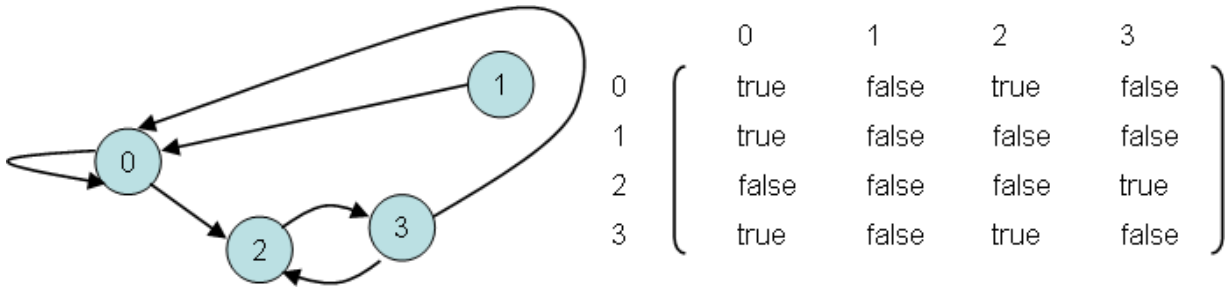


Fig: Graph and adjacency matrix

Definition:

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

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

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

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

Representing Graphs with Edge Lists

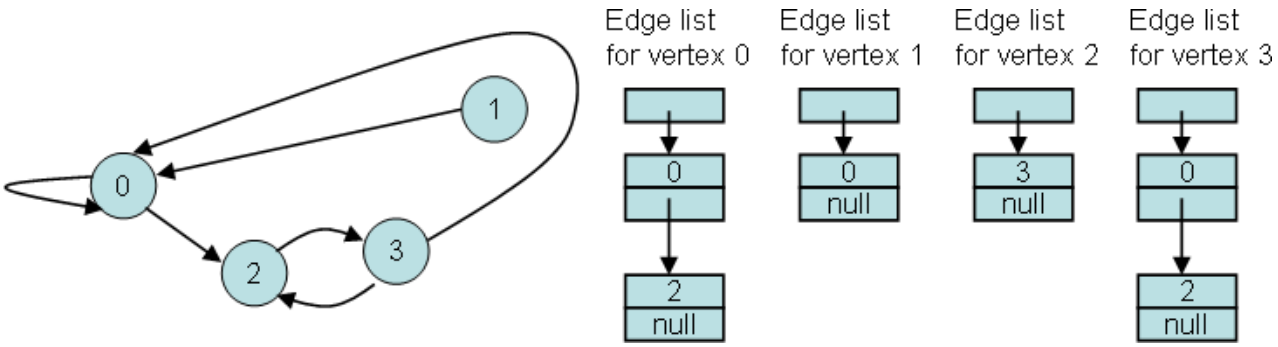


Fig: Graph and adjacency list for each node

Definition:

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

Loops and multiple edges could be allowed.

Representing Graphs with Edge Sets

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

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

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

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

Program code:

```
#include<iostream>
#include<stdlib.h>
#include<string.h>
using namespace std;
struct node
{   string vertex;
    int time;
```

```
   node *next;
};
class adjmatlist
{    int m[10][10],n,i,j; char ch;  string v[20];   node *head[20];  node *temp=NULL;

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

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

}
void adjmatlist::adjlist()
{      cout<<"\n ****";
       for(i=0;i<n;i++)
       {  node *p=new(struct node);
          p->next=NULL;
```

```
        p->vertex=v[i];
        head[i]=p;      cout<<"\n"<<head[i]->vertex;
      }

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

              }

        }
      }

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

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

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

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

                }




    }

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

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


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

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

                }




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

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

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

                     case 3: a.displaya();
                         break;
                     case 4: exit(0);

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

Output:

```
enter the choice
 1.enter graph
 2.display adjacency matrix for cities
 3.display adjacency list for cities
 4.exit
1

 enter no. of cities(max. 20) 3

 enter name of cities Sangli
Satara
Pune

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

 enter time required to reach city Sangli from Sangli in minutes 0

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

 enter time required to reach city Satara from Sangli in minutes 120
```

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

enter time required to reach city Pune from Sangli in minutes 250

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

enter time required to reach city Sangli from Satara in minutes 120

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

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

enter time required to reach city Pune from Satara in minutes 132

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

enter time required to reach city Sangli from Pune in minutes 250

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

enter time required to reach city Satara from Pune in minutes 132

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

enter time required to reach city Pune from Pune in minutes 0

****
Sangli
Satara
Pune

enter the choice
1.enter graph
2.display adjacency matrix for cities
3.display adjacency list for cities
4.exit
2

	Sangli	Satara	Pune
Sangli	0	120	250
Satara	120	0	132
Pune	250	132	0

enter the choice

```
 1.enter graph
 2.display adjacency matrix for cities
 3.display adjacency list for cities
 4.exit
3

 adjacency list is
Sangli-> Satara-> Pune
Satara-> Sangli-> Pune
Pune-> Sangli-> Satara
 path and time required to reach cities is
Sangli-> Satara
   [time required: 120 min ]
Sangli-> Pune
   [time required: 250 min ]
Satara-> Sangli
   [time required: 120 min ]
Satara-> Pune
   [time required: 132 min ]
Pune-> Sangli
   [time required: 250 min ]
Pune-> Satara
   [time required: 132 min ]

 enter the choice
 1.enter graph
 2.display adjacency matrix for cities
 3.display adjacency list for cities
 4.exit
4
```

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

Assignment -7

Title:

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

Learning Objective:

- ✓ To understand the concept and basic of spanning.
- ✓ To understand Graph in Data structure.

Learning Outcome:

- ✓ To implement the spanning
- ✓ To find the minimum distancebetween the vertices of Graph in Data structure.

Theory

Properties of a Greedy Algorithm:

- ✓ At each step, the best possible choice is taken and after that only the sub-problem is solved.
- ✓ Greedy algorithm might be depending on many choices. But it cannot ever be depending upon any choices of future and neither on sub-problems solutions.
- ✓ The method of greedy algorithm starts with a top and goes down, creating greedy choices in a series and then reduce each of the given problem to even smaller ones.

Minimum Spanning Tree:
A Minimum Spanning Tree (MST) is a kind of a sub graph of an undirected graph in which, the sub graph spans or includes all the nodes has a minimum total edge weight.

To solve the problem by a prim's algorithm, all we need is to find a spanning tree of minimum length, where a spanning tree is a tree that connects all the vertices together and a minimum spanning tree is a spanning tree of minimum length.

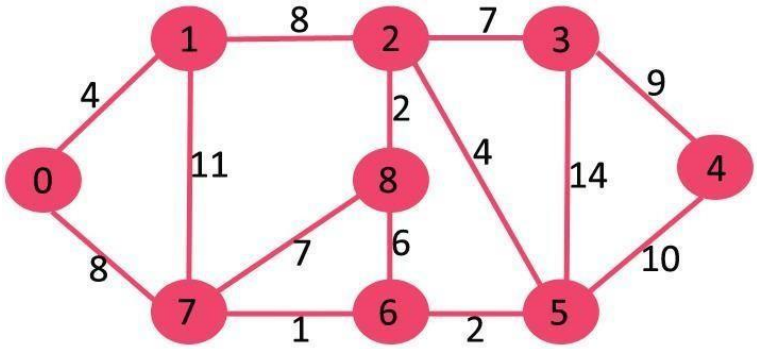
Properties of Prim's Algorithm:
Prim's Algorithm has the following properties:

1. The edges in the subset of some minimum spanning tree always form a single tree.

2. It grows the tree until it spans all the vertices of the graph.
3. An edge is added to the tree, at every step, that crosses a cut if its weight is the minimum of any edge crossing the cut, connecting it to a vertex of the graph.

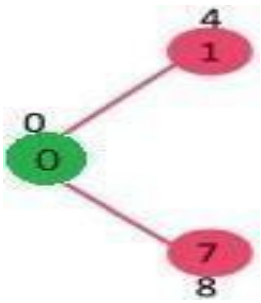
Algorithm:

1. Begin with any vertex which you think would be suitable and add it to the tree.
2. Find an edge that connects any vertex in the tree to any vertex that is not in the tree. Notethat, we don't have to form cycles.
3. Stop when n - 1 edges have been added to the tree



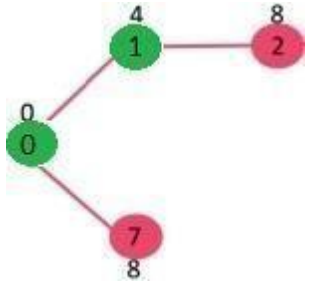
Example

- ✓ The set *mstSet* is initially empty and keys assigned to vertices are {0, INF, INF, INF, INF, INF, INF, INF} where INF indicates infinite.
- ✓ Pick the vertex with the minimum key value.
- ✓ The vertex 0 is picked, include it in *mstSet*. So *mstSet* becomes {0}.
- ✓ After including to *mstSet*, update key values of adjacent vertices.
- ✓ Adjacent vertices of 0 are 1 and 7.
- ✓ The key values of 1 and 7 are updated as 4 and 8.
- ✓ Following subgraph shows vertices and their key values, only the vertices with finite key values are shown.
- ✓ The vertices included in MST are shown in green color.

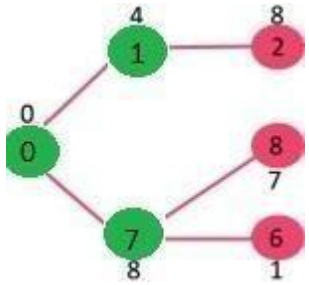


- ✓ Pick the vertex with minimum key value and not already included in MST (not in mstSET).
- ✓ The vertex 1 is picked and added to mstSet. So mstSet now becomes {0, 1}. Update the key values of adjacent vertices of 1.

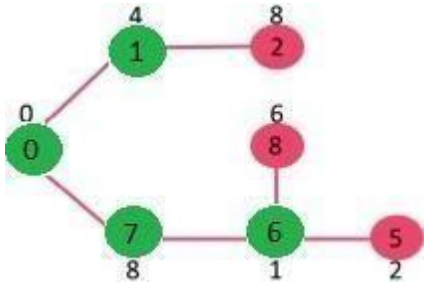
- ✓ The key value of vertex 2 becomes 8.



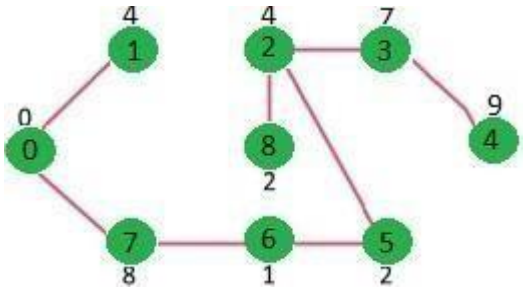
- ✓ Pick the vertex with minimum key value and not already included in MST (not in mstSET).
- ✓ Either pick vertex 7 or vertex 2, let vertex 7 is picked. So mstSet now becomes {0, 1, 7}.
- ✓ Update the key values of adjacent vertices of 7. The key value of vertex 6 and 8 becomes finite (1 and 7 respectively).



- ✓ Pick the vertex with minimum key value and not already included in MST (not in mstSET). Vertex 6 is picked. So mstSet now becomes {0, 1, 7, 6}.
- ✓ Update the key values of adjacent vertices of 6. The key value of vertex 5 and 8 are updated.



- ✓ Repeat the above steps until *mstSet* includes all vertices of given graph. Finally,
- ✓ Get the following graph.



Expected output:

Take the adjacency matrix as an input and the edges between the connected Cities should be displayed with weights.

Program code:

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

#define ROW 10
#define COL 10
#define infi 9999

class prims {
   int graph[ROW][COL], nodes;
public:
   void createGraph();
   void primsAlgo();
};

void prims::createGraph() {
   int i, j;
   cout << "Enter Total Offices: ";
   cin >> nodes;
   cout << "\nEnter Adjacency Matrix: \n";
   for (i = 0; i < nodes; i++) {
      for (j = i; j < nodes; j++) {
         cout << "Enter distance between " << i << " and " << j << endl;
         cin >> graph[i][j];
         graph[j][i] = graph[i][j];
      }
   }

   for (i = 0; i < nodes; i++) {
      for (j = 0; j < nodes; j++) {
         if (graph[i][j] == 0)
            graph[i][j] = infi;    //fill infinity where path is not present
      }
   }
}

void prims::primsAlgo() {
   int selected[ROW], i, j, ne=0;
   int zero = 0, one = 1, min = 0, x, y;
   int cost = 0;
   for (i = 0; i < nodes; i++)
      selected[i] = zero;
```

```
    selected[0] = one;        //starting vertex is always node-0

    while (ne < nodes - 1) {
        min = infi;

        for (i = 0; i < nodes; i++) {
            if (selected[i] == one) {
                for (j = 0; j < nodes; j++) {
                    if (selected[j] == zero) {
                        if (min > graph[i][j]) {
                            min = graph[i][j];
                            x = i;
                            y = j;
                        }
                    }
                }
            }
        }
        selected[y] = one;
        cout << "\n" << x << " --> " << y;
        cost += graph[x][y];
        ne++;
    }
    cout << "\nTotal cost is: " << cost << endl;
}

int main() {
    prims MST;
    cout << "\nPrims Algorithm to connect several offices\n";
    MST.createGraph();
    MST.primsAlgo();
}
```

Output:

```
Prims Algorithm to connect several offices
Enter Total Offices: 3

Enter Adjacency Matrix:
Enter distance between 0 and 0
 2 3
Enter distance between 0 and 1
Enter distance between 0 and 2
4 5
Enter distance between 1 and 1
Enter distance between 1 and 2
6 7
Enter distance between 2 and 2
```

0 --> 1
0 --> 2
Total cost is: 7

Conclusion:

Understood the concept and basic of spanning and to find the minimum distance between thevertices of Graph in Data structure.

Assignment -8

A Dictionary stores keywords & its meaning. 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.

Learning Objectives:

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

Learning Outcome:

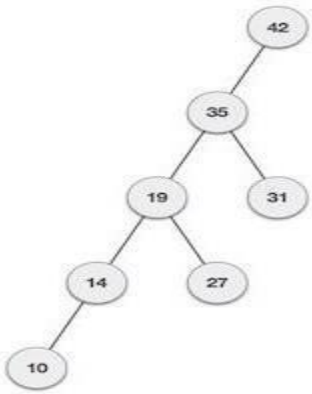
- Define class for AVL using Object Oriented features.
- Analyze working of various operations on AVL Tree.

Theory:

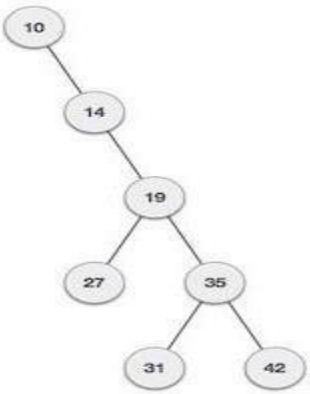
An empty tree is height balanced tree if T is a nonempty binary tree with TL and TR asits left and right sub trees. The T is height balance if and only if Its balance factor is 0, 1, -1.

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

What if the input to binary search tree comes in a sorted (ascending or descending)manner? It will then look like this



If input 'appears' non-increasing manner

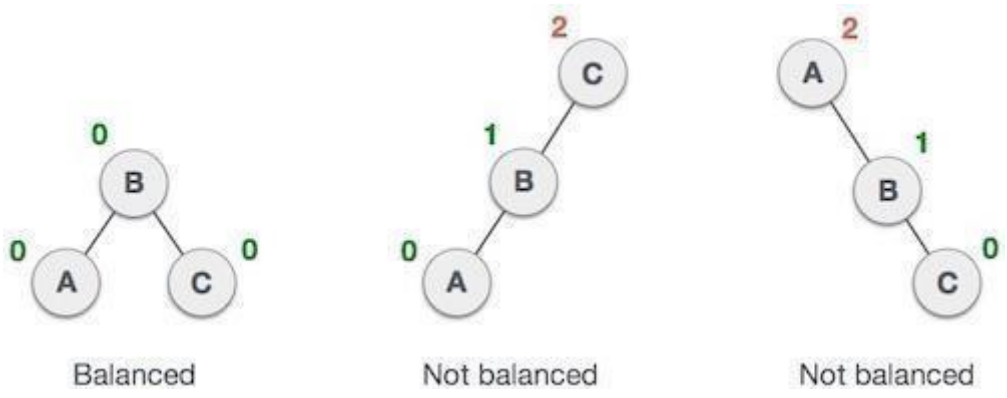


If input 'appears' in non-decreasing manner

It is observed that BST's worst-case performance is closest to linear search algorithms, that is O(n). In real-time data, we cannot predict data pattern and their frequencies. So, a need arises to balance out the existing BST.

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

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



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

$$\textbf{\textit{BalanceFactor}} = \text{height(left-sutree)} - \text{height(right-sutree)}$$

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

AVL Rotations

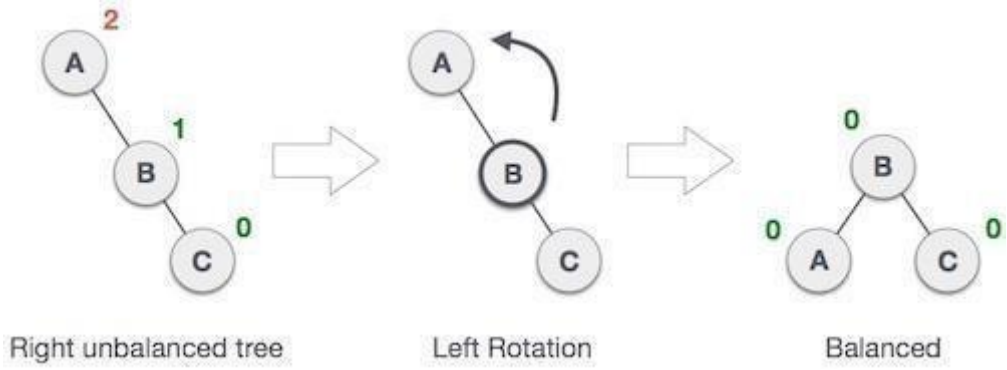
To balance itself, an AVL tree may perform the following four kinds of rotations −

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

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

Left Rotation

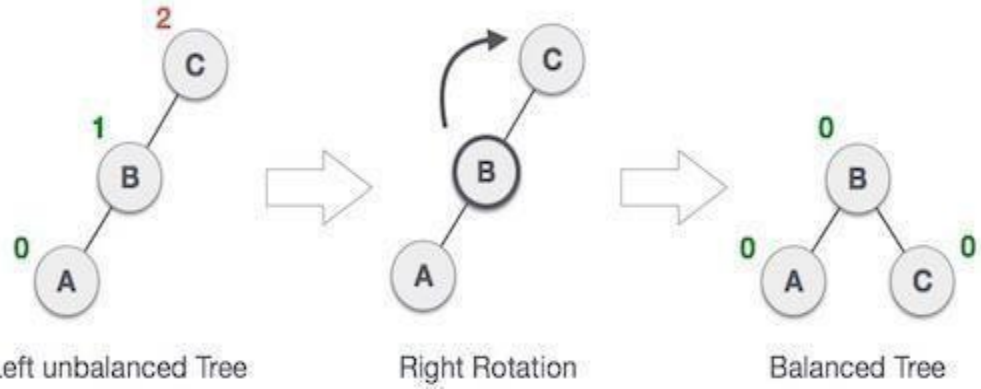
If a tree becomes unbalanced, when a node is inserted into the right subtree of the rightsubtree, then we perform a single left rotation −



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

Right Rotation

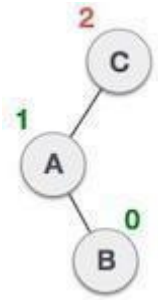
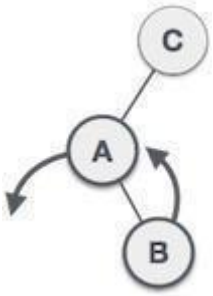
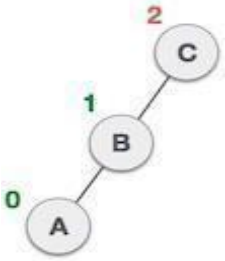
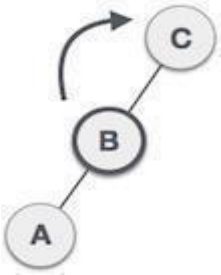
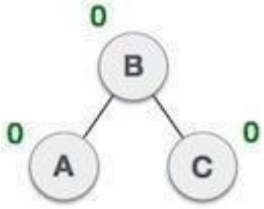
AVL tree may become unbalanced, if a node is inserted in the left subtree of the leftsubtree. The tree then needs a right rotation.



As depicted, the unbalanced node becomes the right child of its left child by performing a rightrotation.

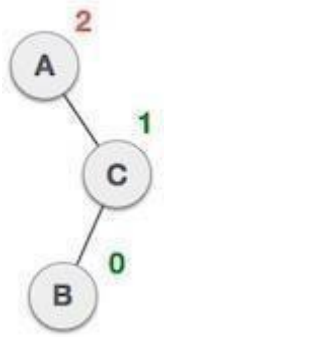
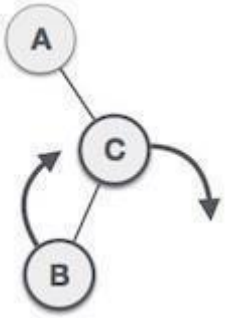
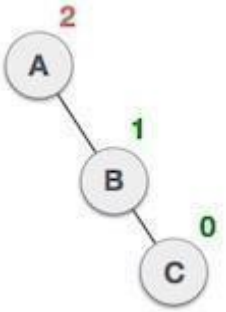
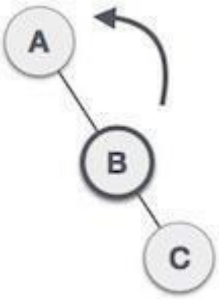
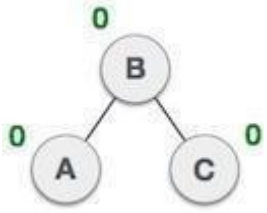
Left-Right Rotation

Double rotations are slightly complex version of already explained versions of rotations. To understand them better, we should take note of each action performed while rotation. Let's first check how to perform Left-Right rotation. A left-right rotation is a combination of left rotation followed by right rotation.

State	Action
	A node has been inserted into the right subtree of the left subtree. This makes **C** an unbalanced node. These scenarios cause AVL tree to perform left-right rotation.
	We first perform the left rotation on the left subtree of **C**. This makes **A**, the left subtree of **B**.
	Node C is still unbalanced, however now, it is because of the left- subtree of the left-subtree.
	We shall now right rotate the tree, making B the new root node of this subtree. C now becomes the right subtree of its own left subtree.
	The tree is now balanced.

Right-Left Rotation

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

State	Action
	A node has been inserted into the left subtree of the right subtree. This makes **A**, an unbalanced node with balance factor 2.
	First, we perform the right rotation along **C** node, making **C** the right subtree of its own left subtree **B**. Now, **B** becomes the right subtree of **A**.
	Node **A** is still unbalanced because of the right subtree of its right subtree and requires a left rotation.
	A left rotation is performed by making **B** the new root node of the subtree. **A** becomes the left subtree of its right subtree **B**.
	The tree is now balanced.

Algorithm AVL TREE:

Insert:-

1. If P is NULL, then
    I. P = new node
    II. P ->element = x
    III. P ->left = NULL
    IV. P ->right = NULL
    V. P ->height = 0

2. else if x>1 => x<P ->element
    a.) insert(x, P ->left)
    b.) if height of P->left -height of P ->right =2
        1. insert(x, P ->left)
        2. if height(P ->left) -height(P ->right) =2
            if x<P ->left ->element
                P =singlerotateleft(P)
            else
                P =doublerotateleft(P)
3. else
    if x<P ->element

        a.) insert(x, P -> right)
        b.) if height (P -> right) -height (P ->left) =2
            if(x<P ->right) ->element
                P =singlerotateright(P)
            else
                P =doublerotateright(P)
4. else
Print already exits
5. int m, n, d.

6. m = AVL height (P->left)
7. n = AVL height (P->right)
8. d = max(m, n)
9. P->height = d+1
10. Stop

RotateWithLeftChild(AvlNode k2)

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

RotateWithRightChild(AvlNode k1)

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

doubleWithLeftChild(AvlNode k3)

- k3.left = rotateWithRightChild(k3.left);
- return rotateWithLeftChild(k3);

doubleWithRightChild(AvlNode k1)

- k1.right = rotateWithLeftChild(k1.right);
- return rotateWithRightChild(k1);

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

Program code:

```
#include <iostream>
#include<string>
using namespace std;
class dictionary;
class node
{
 string word,meaning;
 node *left,*right;
public:
 friend class dictionary;
 node()
 {
  left=NULL;
  right=NULL;
```

```
 }
 node(string word, string meaning)
 {
  this->word=word;
  this->meaning=meaning;
  left=NULL;
  right=NULL;
 }
};

class dictionary
{
 node *root;
public:
 dictionary()
{
  root=NULL;
}
 void create();
 void inorder_rec(node *rnode);
 void postorder_rec(node *rnode);
 void inorder()
 {
  inorder_rec(root);
 }
 void postorder();

 bool insert(string word,string meaning);
 int search(string key);

};
int dictionary::search(string key)
{
 node *tmp=root;
 int count;
 if(tmp==NULL)
 {
  return -1;
 }
 if(root->word==key)
  return 1;
 while(tmp!=NULL)
 {

  if((tmp->word)>key)
  {
   tmp=tmp->left;
   count++;
  }
  else if((tmp->word)<key)
```

```
  {
   tmp=tmp->right;
   count++;
  }
  else if(tmp->word==key)
  {
   return ++count;
  }
 }
 return -1;

}
void dictionary::postorder()
{
 postorder_rec(root);
}
void dictionary::postorder_rec(node *rnode)
{
 if(rnode)
 {
  postorder_rec(rnode->right);
  cout<<" "<<rnode->word<<" : "<<rnode->meaning<<endl;
  postorder_rec(rnode->left);
 }
}
void dictionary::create()
{
 int n;
 string wordI,meaningI;
 cout<<"\nHow many Word to insert?:\n";
 cin>>n;
 for(int i=0;i<n;i++)
 {
  cout<<"\nENter Word: ";
  cin>>wordI;
  cout<<"\nEnter Meaning: ";
  cin>>meaningI;
  insert(wordI,meaningI);
 }
}
void dictionary::inorder_rec(node *rnode)
{
 if(rnode)
 {
  inorder_rec(rnode->left);
  cout<<" "<<rnode->word<<" : "<<rnode->meaning<<endl;
  inorder_rec(rnode->right);
 }
}
bool dictionary::insert(string word, string meaning)
{
```

```
 node *p=new node(word, meaning);
 if(root==NULL)
 {
  root=p;
  return true;
 }
 node *cur=root;
 node *par=root;
 while(cur!=NULL) //traversal
 {
  if(word>cur->word)
  {par=cur;
  cur=cur->right;
  }
  else if(word<cur->word)
  {
   par=cur;
   cur=cur->left;
  }
  else
  {
   cout<<"\nWord is already in the dictionary.";
   return false;
  }
 }
 if(word>par->word) //insertion of node
 {
  par->right=p;
  return true;
 }
 else
 {
  par->left=p;

  return true;
 }
}

int main() {
 string word;
 dictionary months;
 months.create();
 cout<<"Ascending order\n";
 months.inorder();

 cout<<"\nDescending order:\n";
 months.postorder();

 cout<<"\nEnter word to search: ";
 cin>>word;
 int comparisons=months.search(word);
```

```
 if(comparisons==-1)
 {
  cout<<"\nNot found word";
 }
 else
 {
  cout<<"\n "<<word<<" found in "<<comparisons<<" comparisons";
 }
 return 0;
}
```

Output:

```
How many Word to insert?:
3

ENter Word: Com

Enter Meaning: Computer

ENter Word: Mec

Enter Meaning: Mechanical

ENter Word: Civ

Enter Meaning: Civil
Ascending order
 Civ : Civil
 Com : Computer
 Mec : Mechanical

Descending order:
 Mec : Mechanical
 Com : Computer
 Civ : Civil

Enter word to search: Com

 Com found in 1 comparisons
```

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

Assignment -9

Title:

Implement the Heap/Shell sort algorithm implemented in Java demonstrating heap/shell data structure with modularity of programming language**.**

Learning Objectives:

- ✓ To understand concept of heap in data structure.
- ✓ To understand concept & features of java language.

Learning Outcome:

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

Theory:

Heap Sort:

Heap sort is a comparison-based sorting technique based on Binary Heap data structure. Itis similar to selection sort where we first find the maximum element and place themaximum element at the end. We repeat the same process for remaining element.

What is Binary Heap?

Let us first define a Complete Binary Tree. A complete binary tree is a binary tree in which every level, except possibly the last, is completely filled, and all nodes are as far left as possible (Source Wikipedia)

A Binary Heap is a Complete Binary Tree where items are stored in a special order such that value in a parent node is greater (or smaller) than the values in its two children nodes. The former is called as max heap and the latter is called min heap. The heap can be represented by binary tree or array.

Why array-based representation for Binary Heap?

Since a Binary Heap is a Complete Binary Tree, it can be easily represented as array and array based representation is space efficient. If the parent node is stored at index I, the left child can be calculated by 2 * I + 1 and right child by 2 * I + 2 (assuming the indexing starts at 0).

Heap Sort Algorithm for sorting in increasing order:

1. Build a max heap from the input data.

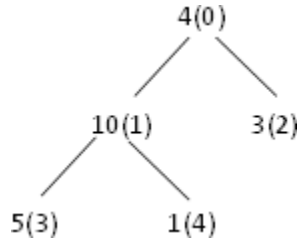
2. At this point, the largest item is stored at the root of the heap. Replace it with the last item ofthe heap followed by reducing the size of heap by 1. Finally, heapify the root of tree.
3. Repeat above steps until size of heap is greater than 1.

How to build the heap?

Heapify procedure can be applied to a node only if its children nodes are heapified. Sothe heapification must be performed in the bottom up order.

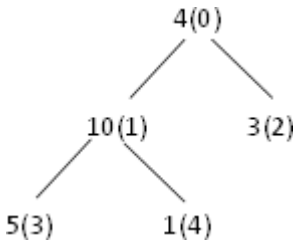
Lets understand with the help of an example:

Input data: 4, 10, 3, 5, 1



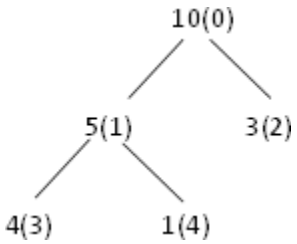
The numbers in bracket represent the indices in the arrayrepresentation of data.

Applying heapify procedure to index 1:



Applying heapify procedure to index 0:

The heapify procedure calls itself recursively to build heapin top-down manner.



Algorithm:

STEP 1: Logically, think the given array as Complete Binary Tree,
STEP 2: For sorting the array in ascending order, check whether the tree is satisfying Max-heap property at each node, (For descending order, Check whether the tree is satisfying Min-heap property) Here we will be sorting in Ascending order,
STEP 3: If the tree is satisfying Max-heap property, then largest item is stored at the root of the heap. (At this point we have found the largest element in array, Now if we place this element at the end(nth position) of the array then 1 item in array is at proper place.)

We will remove the largest element from the heap and put at its proper place(nth position) in array.

After removing the largest element, which element will take its place? We will put last element of the heap at the vacant place. After placing the last element at the root, The new tree formed may or may not satisfy max-heap property. So, If it is not satisfying max- heap property then first task is to make changes to the tree, So that it satisfies max-heap property.

(Heapify process: The process of making changes to tree so that it satisfies max-heap property is called heapify)

When tree satisfies max-heap property, again largest item is stored at the root of the heap. We will remove the largest element from the heap and put at its proper place(n-1 position) in array. Repeat step 3 until size of array is 1 (At this point all elements are sorted.)

Expected Output: Elements in sorted order.

Program Code:

```
import java.util.*;
class heapupdate1
{
   void heapdown(int a[],int n,int i)
   {
      int temp,j;
      while(2*i+1<n)
      {
         j=2*i+1;

         if(j+1 < n && a[j+1]> a[j])
            j=j+1;
         if(a[i]>a[j])
            break;
         else
         {
            temp=a[i];
            a[i]=a[j];
            a[j]=temp;
            i=j;
```

```
            }
        }
    }
    void heapsort(int a[],int n)
    {
        int i,temp;
        for(i=(n-1)/2;i>=0;i--)
            heapdown(a,n,i);

        while(n>0)
        {
            temp=a[0];
            a[0]=a[n-1];
            a[n-1]=temp;
            n--;
            heapdown(a,n,0);
        }
    }

public static void main (String args[])
{
        int a[]=new int[10];
        int n,i;
        Scanner in=new Scanner(System.in);

        System.out.println("Enter no. of elements");
        n =in.nextInt();
        System.out.println("Enter elements");
        for(i=0;i<n;i++)
        {
            a[i]=in.nextInt();
        }
        heapupdate1 ob=new heapupdate1();
        ob.heapsort(a,n);
        System.out.println("The sorted elements are->");
        for(i=0;i<n;i++)
        {
            System.out.println(""+a[i]);
        }


    }
}
```

Output:

```
Enter no. of elements
5
Enter elements
76
99
85
30
66
The sorted elements are->
30
66
76
85
99
```

Conclusion: This program gives us the knowledge of heap data structure.

Assignment-10

Title:

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

Learning Objectives:

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

Learning Outcome:

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

Theory:

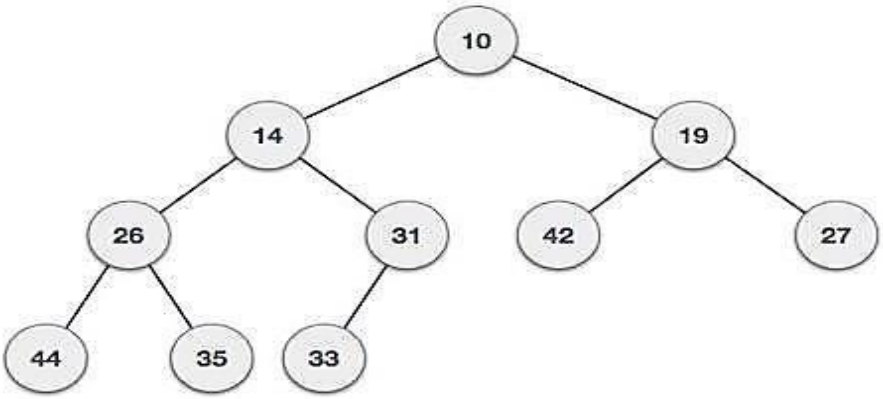
Heap is a special case of balanced binary tree data structure where the root-node key iscompared with its children and arranged accordingly. If **α** has child node **β** then –

$$key(\alpha) \geq key(\beta)$$

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

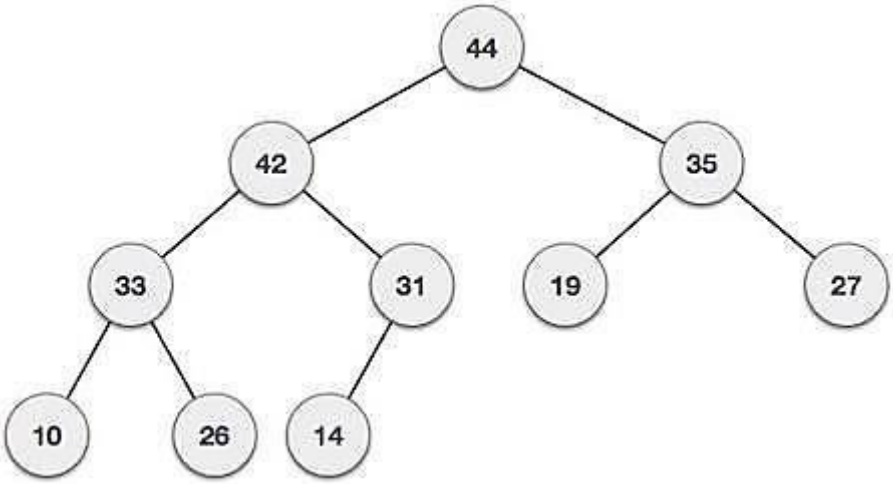
For Input → 35 33 42 10 14 19 27 44 26 3126 31

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



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

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



Max Heap Construction Algorithm

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

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

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

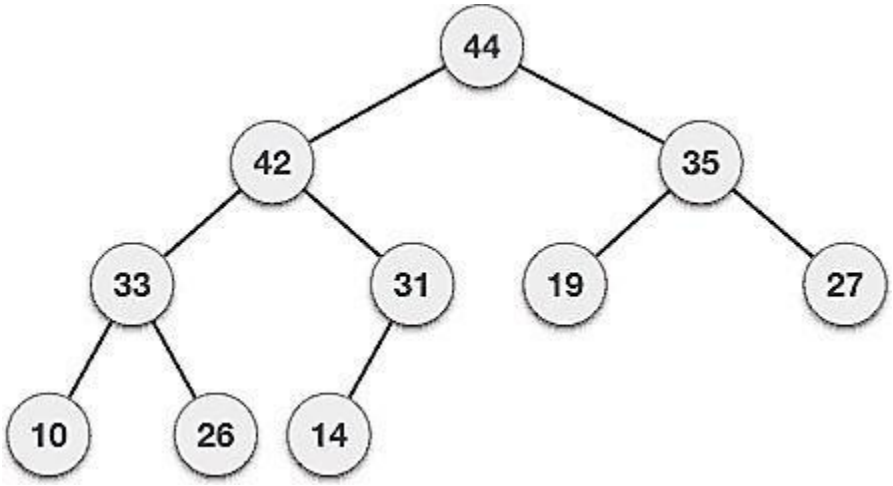
Note − In Min Heap construction algorithm, we expect the value of the parent node to be lessthan that of the child node.
Let's understand Max Heap construction by an animated illustration. We consider the same inputsample that we used earlier.

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

Max Heap Deletion Algorithm

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

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



Expected Output: Find min and max marks obtained.

Program code:

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

class hp
{
  int heap[20],heap1[20],x,n1,i;
  public:
  hp()
  { heap[0]=0;  heap1[0]=0;
  }
  void getdata();
  void insert1(int heap[],int);
  void upadjust1(int heap[],int);
  void insert2(int heap1[],int);
  void upadjust2(int heap1[],int);
  void minmax();
};
void hp::getdata()
{
  cout<<"\n enter the no. of students";
  cin>>n1;
```

```
  cout<<"\n enter the marks";
  for(i=0;i<n1;i++)
  {   cin>>x;
      insert1(heap,x);
      insert2(heap1,x);
  }
}
void hp::insert1(int heap[20],int x)
{
  int n;
  n=heap[0];
  heap[n+1]=x;
  heap[0]=n+1;

  upadjust1(heap,n+1);
}
void hp::upadjust1(int heap[20],int i)
{
   int temp;
   while(i>1&&heap[i]>heap[i/2])
   {
      temp=heap[i];
      heap[i]=heap[i/2];
      heap[i/2]=temp;
      i=i/2;
   }
}
void hp::insert2(int heap1[20],int x)
{
  int n;
  n=heap1[0];
  heap1[n+1]=x;
  heap1[0]=n+1;

  upadjust2(heap1,n+1);
}
void hp::upadjust2(int heap1[20],int i)
{
   int temp1;
   while(i>1&&heap1[i]<heap1[i/2])
   {
      temp1=heap1[i];
      heap1[i]=heap1[i/2];
      heap1[i/2]=temp1;
      i=i/2;
   }
}
void hp::minmax()
{
  cout<<"\n max marks"<<heap[1];
```

```
   cout<<"\n##";
   for(i=0;i<=n1;i++)
   {   cout<<"\n"<<heap[i];  }
   cout<<"\n min marks"<<heap1[1];
   cout<<"\n##";
   for(i=0;i<=n1;i++)
   {   cout<<"\n"<<heap1[i];  }
}
int main()
{
  hp h;
  h.getdata();
  h.minmax();
  return 0;
}
```

Output:

```
enter the no. of students 5

 enter the marks 94
78
89
75
87

 max marks94
##
5
94
87
89
75
78
 min marks75
##
5
75
78
89
94
87
```

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

Assignment-11

Title:

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

Learning Objectives:

- ✓ To understand concept of file organization in data structure.
- ✓ To understand concept & features of sequential file organization.

Learning Outcome:

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

Theory:

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

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

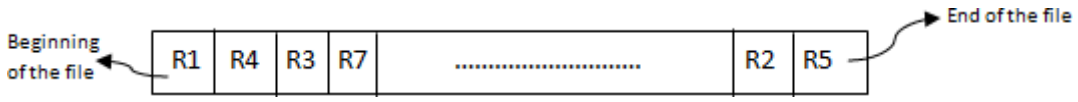
Some of the file organizations are:

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

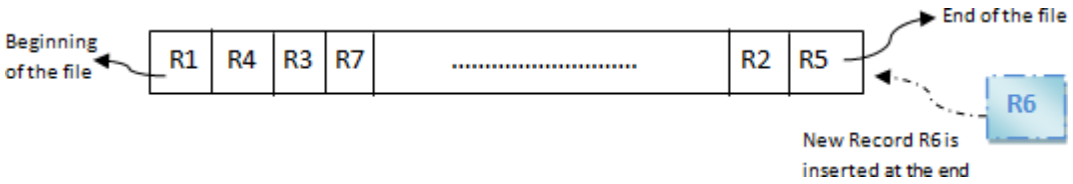
Sequential File Organization:

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

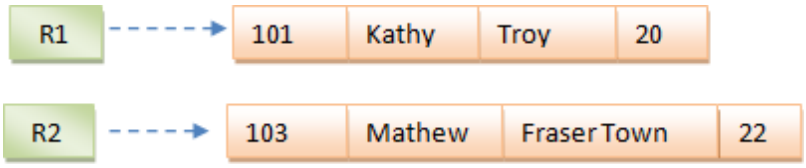
- Records are stored one after the other as they are inserted into the tables. This method is called pile file method. When a new record is inserted, it is placed at the end of the file.In the case of any modification or deletion of record, the record will be searched in the memory blocks. Once itis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



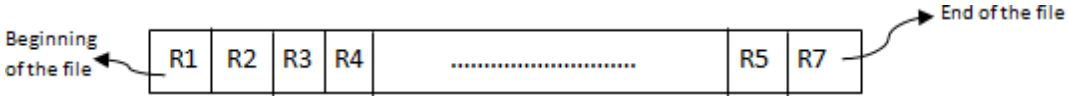
Inserting a new record:



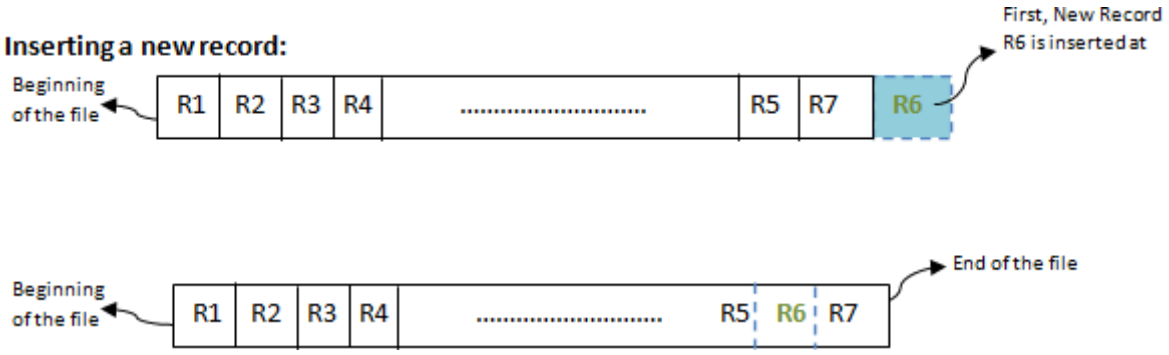
In the diagram above, R1, R2, R3 etc are the records. They contain all the attribute of a row. i.e.; when we say student record, it will have his id, name, address, course, DOB etc. Similarly R1, R2, R3 etc can be considered as one full set of attributes.



In the second method, records are sorted (either ascending or descending) each time they are inserted into the system. This method is called **sorted file method**. Sorting of records may bebased on the primary key or on any other columns. Whenever a new record is inserted, it will be inserted at the end of the file and then it will sort – ascending or descending based on key value and placed at the correct position. In the case of update, it will update the record and then sort thefile to place the updated record in the right place. Same is the case with delete.



Inserting a new record:



Advantages:

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

Disadvantages:

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

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

Program Code:

```
#include <iostream>
#include<fstream>
#include<cstring>
#include<iomanip>
using namespace std;
const int MAX=20;
class Student
{
```

```
 int rollno;
 char name[20],city[20];
 char div;
 int year;
public:
 Student()
{
  strcpy(name,"");
  strcpy(city,"");
  rollno=year=div=0;
}
 Student(int rollno,char name[MAX],int year,char div,char city[MAX])
 {
  strcpy(this->name,name);
  strcpy(this->city,city);
  this->rollno=rollno;
  this->year=year;
  this->div=div;
 }
 int getRollNo()
 {
  return rollno;
 }
 void displayRecord()
 {

  cout<<endl<<setw(5)<<rollno<<setw(20)<<name<<setw(5)<<year<<setw(5)<<div<<setw(10)<<city;
 }
};
//==========File Operations ===========
class FileOperations
{
 fstream file;
public:
 FileOperations(char* filename)
{
file.open(filename,ios::in|ios::out|ios::ate|ios::binary);
}
```

```
void insertRecord(int rollno, char name[MAX],int year, char div,char city[MAX])
{
 Student s1(rollno,name,year,div,city);
 file.seekp(0,ios::end);
 file.write((char *)&s1,sizeof(Student));
 file.clear();
}
void displayAll()
{
 Student s1;
 file.seekg(0,ios::beg);
 while(file.read((char *)&s1, sizeof(Student)))
 {
  s1.displayRecord();
 }
 file.clear();
}
void displayRecord(int rollNo)
{
 Student s1;
 file.seekg(0,ios::beg);
 bool flag=false;
 while(file.read((char*)&s1,sizeof(Student)))
 {
  if(s1.getRollNo()==rollNo)
  {
   s1.displayRecord();
   flag=true;
   break;
  }
 }
 if(flag==false)
 {
  cout<<"\nRecord of "<<rollNo<<"is not present.";
 }
 file.clear();
}
void deleteRecord(int rollno)
```

```
 {
  ofstream outFile("new.dat",ios::binary);
  file.seekg(0,ios::beg);
  bool flag=false;
  Student s1;

  while(file.read((char *)&s1, sizeof(Student)))
  {
   if(s1.getRollNo()==rollno)
   {
    flag=true;
    continue;
   }
   outFile.write((char *)&s1, sizeof(Student));
  }
  if(!flag)
  {
   cout<<"\nRecord of "<<rollno<<" is not present.";
  }
  file.close();
  outFile.close();
  remove("student.dat");
  rename("new.dat","student.dat");
  file.open("student.dat",ios::in|ios::out|ios::ate|ios::binary);
 }
 ~FileOperations()
 {
  file.close();
  cout<<"\nFile Closed.";
 }
};
int main() {
 ofstream newFile("student.dat",ios::app|ios::binary);
  newFile.close();
  FileOperations file((char*)"student.dat");
     int rollNo,year,choice=0;
     char div;
     char name[MAX],address[MAX];
```

```
    while(choice!=5)
    {
       //clrscr();
       cout<<"\n*****Student Database*****\n";
       cout<<"1) Add New Record\n";
       cout<<"2) Display All Records\n";
       cout<<"3) Display by RollNo\n";
       cout<<"4) Deleting a Record\n";
       cout<<"5) Exit\n";
       cout<<"Choose your choice : ";
       cin>>choice;
       switch(choice)
       {
          case 1 : //New Record
            cout<<endl<<"Enter RollNo and name : \n";
            cin>>rollNo>>name;
            cout<<"Enter Year and Division : \n";
            cin>>year>>div;
            cout<<"Enter address : \n";
            cin>>address;
            file.insertRecord(rollNo,name,year,div,address);
            cout<<"\nRecord Inserted.";
            break;
          case 2 :

cout<<endl<<setw(5)<<"ROLL"<<setw(20)<<"NAME"<<setw(5)<<"YEAR"<<setw(5)<<"DIV"<<setw(10)
<<"CITY";
            file.displayAll();
            break;
          case 3 :
            cout<<"Enter Roll Number";
            cin>>rollNo;
             file.displayRecord(rollNo);

            break;
          case 4:
            cout<<"Enter rollNo";
            cin>>rollNo;
```

```
            file.deleteRecord(rollNo);
            break;
          case 5 :break;
        }
}
return 0;
}
```

Output:

```
*****Student Database*****
1) Add New Record
2) Display All Records
3) Display by RollNo
4) Deleting a Record
5) Exit
Choose your choice : 1

Enter RollNo and name :
1
Raj
Enter Year and Division :
2 B
Enter address :
Pune

Record Inserted.
*****Student Database*****
1) Add New Record
2) Display All Records
3) Display by RollNo
4) Deleting a Record
5) Exit
Choose your choice : 1

Enter RollNo and name :
2
Shaan
Enter Year and Division :
2 B
Enter address :
Nashik

Record Inserted.
*****Student Database*****
1) Add New Record
2) Display All Records
```

```
3) Display by RollNo
4) Deleting a Record
5) Exit
Choose your choice : 1

Enter RollNo and name :
3
Sita
Enter Year and Division :
1 A
Enter address :
Jalna

Record Inserted.
*****Student Database*****
1) Add New Record
2) Display All Records
3) Display by RollNo
4) Deleting a Record
5) Exit
Choose your choice : 2

 ROLL              NAME YEAR  DIV      CITY
    1               Raj    2    B      Pune
    2             Shaan    2    B    Nashik
    3              Sita    1    A     Jalna
*****Student Database*****
1) Add New Record
2) Display All Records
3) Display by RollNo
4) Deleting a Record
5) Exit
Choose your choice : 3
Enter Roll Number 2

    2             Shaan    2    B    Nashik
*****Student Database*****
1) Add New Record
2) Display All Records
3) Display by RollNo
4) Deleting a Record
5) Exit
Choose your choice : 4
Enter rollNo 2

*****Student Database*****
1) Add New Record
2) Display All Records
```

3) Display by RollNo
4) Deleting a Record
5) Exit
Choose your choice : 2

ROLL	NAME	YEAR	DIV	CITY
1	Raj	2	B	Pune
2	Shaan	2	B	Nashik
3	Sita	1	A	Jalna

*****Student Database*****
1) Add New Record
2) Display All Records
3) Display by RollNo
4) Deleting a Record
5) Exit
Choose your choice : 5

File Closed.

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

Assignment-12

Title:

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

Learning Objectives:

- ✓ To understand concept of file organization in data structure.
- ✓ To understand concept & features of Indexed Sequential file organization.

Learning Outcome:

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

Theory:

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

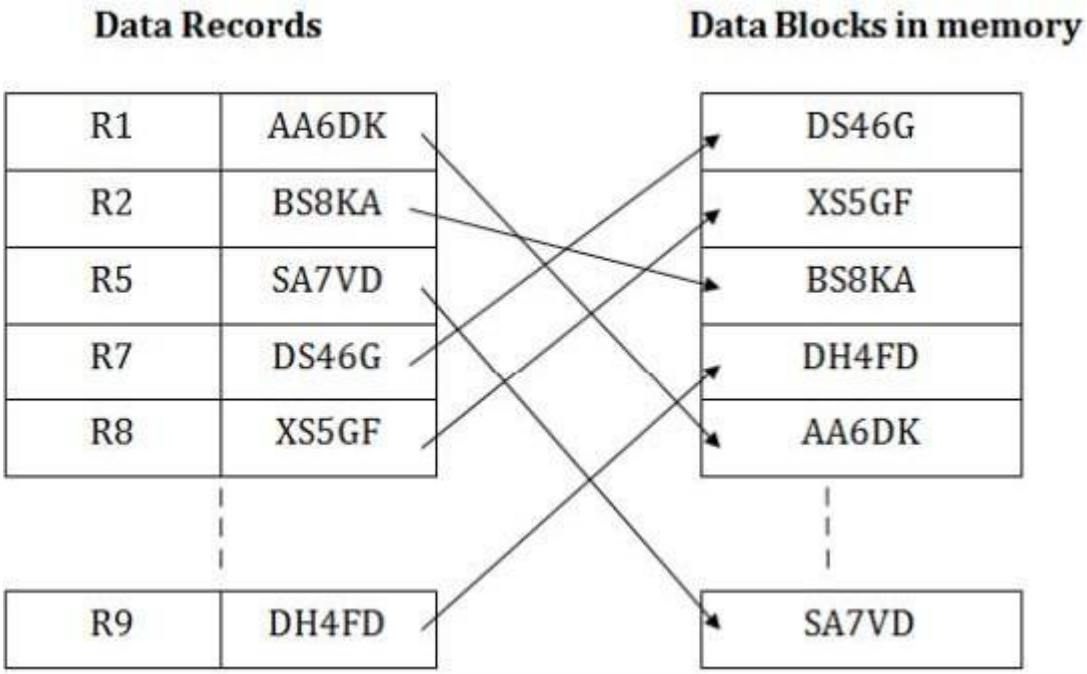
There are various methods of file organizations. These methods may be efficient for certain types of access/selection meanwhile it will turn inefficient for other selections. Hence it isup to the programmer to decide the best suited file organization method depending on his requirement.

Some of the file organizations are

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

Indexed sequential access method (ISAM)

ISAM method is an advanced sequential file organization. In this method, records are stored in the file using the primary key. An index value is generated for each primary key and mapped with the record. This index contains the address of the record in the file.



If any record has to be retrieved based on its index value, then the address of the data block is fetched and the record is retrieved from the memory.

Pros of ISAM:

- ✓ In this method, each record has the address of its data block, searching a record in a huge database is quick and easy.
- ✓ This method supports range retrieval and partial retrieval of records. Since the index is based on the primary key values, we can retrieve the data for the given range of value. In the same way, the partial value can also be easily searched, i.e., the student name starting with 'JA' can be easily searched.

Cons of ISAM

- ✓ This method requires extra space in the disk to store the index value. When the new records are inserted, then these files have to be reconstructed to maintain the sequence.
- ✓ When the record is deleted, then the space used by it needs to be released. Otherwise, the performance of the database will slow down.

Expected Output: If record of employee does not exist an appropriate message is displayed otherwise theemployee‘s details are displayed.

Program code:

```
#include<iostream>
#include<fstream>
#include<stdio.h>

using namespace std;

//Employee class Declaration
class Employee{
   private:
      int code;
      char name[20];
      float salary;
   public:
      void read();
      void display();
      //will return employee code
      int getEmpCode()          { return code;}
      //will return employee salary
      int getSalary()           { return salary;}
      //will update employee salary
      void updateSalary(float s)  { salary=s;}
};

//Read employee record
void Employee::read(){
   cout<<"Enter employee code: ";
   cin>>code;
   cout<<"Enter name: ";
```

```
    cin.ignore(1);
    cin.getline(name,20);
    cout<<"Enter salary: ";
    cin>>salary;
}

//Display employee record
void Employee::display()
{
    cout<<code<<" "<<name<<"\t"<<salary<<endl;
}

//global declaration
fstream file;

//Will delete file when program is being executed
//because we are create file in append mode
void deleteExistingFile(){
    remove("EMPLOYEE.DAT");
}

//function to append record into file
void appendToFille(){
    Employee    x;

    //Read employee record from user
    x.read();

    file.open("EMPLOYEE.DAT",ios::binary|ios::app);
    if(!file){
        cout<<"ERROR IN CREATING FILE\n";
        return;
    }
```

```
    //write into file
    file.write((char*)&x,sizeof(x));
    file.close();
    cout<<"Record added sucessfully.\n";
}

void displayAll(){
    Employee    x;

    file.open("EMPLOYEE.DAT",ios::binary|ios::in);
    if(!file){
        cout<<"ERROR IN OPENING FILE \n";
        return;
    }
    while(file){
    if(file.read((char*)&x,sizeof(x)))
        if(x.getSalary()>=10000 && x.getSalary()<=20000)
            x.display();
    }
  file.close();
}

void searchForRecord(){
    //read employee id
    Employee    x;
    int c;
    int isFound=0;

    cout<<"Enter employee code: ";
    cin>>c;


    file.open("EMPLOYEE.DAT",ios::binary|ios::in);
```

```
    if(!file){
        cout<<"ERROR IN OPENING FILE \n";
        return;
    }
    while(file){
        if(file.read((char*)&x,sizeof(x))){
            if(x.getEmpCode()==c){
                cout<<"RECORD FOUND\n";
                x.display();
                isFound=1;
                break;
            }
        }
    }
    if(isFound==0){
        cout<<"Record not found!!!\n";
    }
    file.close();
}

//Function to increase salary
void increaseSalary(){
    //read employee id
    Employee    x;
    int c;
    int isFound=0;
    float sal;

    cout<<"enter employee code \n";
    cin>>c;


    file.open("EMPLOYEE.DAT",ios::binary|ios::in);
```

```
    if(!file){
        cout<<"ERROR IN OPENING FILE \n";
        return;
    }
    while(file){
        if(file.read((char*)&x,sizeof(x))){
            if(x.getEmpCode()==c){
                cout<<"Salary hike? ";
                cin>>sal;
                x.updateSalary(x.getSalary()+sal);
                isFound=1;
                break;
            }
        }
    }
    if(isFound==0){
        cout<<"Record not found!!!\n";
    }
    file.close();
    cout<<"Salary updated successfully."<<endl;
}

//Insert record by assuming that records are in
//ascending order
void insertRecord(){
    //read employee record
    Employee    x;
    Employee newEmp;

    //Read record to insert
    newEmp.read();

    fstream fin;
```

```
    //read file in input mode
    file.open("EMPLOYEE.DAT",ios::binary|ios::in);
    //open file in write mode
    fin.open("TEMP.DAT",ios::binary|ios::out);

    if(!file){
        cout<<"Error in opening EMPLOYEE.DAT file!!!\n";
        return;
    }
    if(!fin){
        cout<<"Error in opening TEMP.DAT file!!!\n";
        return;
    }
    while(file){
        if(file.read((char*)&x,sizeof(x))){
            if(x.getEmpCode()>newEmp.getEmpCode()){
                fin.write((char*)&newEmp, sizeof(newEmp));
            }
            //no need to use else
            fin.write((char*)&x, sizeof(x));
        }
    }

    fin.close();
    file.close();

    rename("TEMP.DAT","EMPLOYEE.DAT");
    remove("TEMP.DAT");
    cout<<"Record inserted successfully."<<endl;
}

int main()
{
```

```
    char ch;

   //if required then only remove the file
    deleteExistingFile();

    do{
    int n;

    cout<<"ENTER CHOICE\n"<<"1.ADD AN
EMPLOYEE\n"<<"2.DISPLAY\n"<<"3.SEARCH\n"<<"4.INCREASE
SALARY\n"<<"5.INSERT RECORD\n";
    cout<<"Make a choice: ";
    cin>>n;

    switch(n){
        case 1:
          appendToFille();
          break;
        case 2 :
          displayAll();
          break;
        case 3:
          searchForRecord();
          break;
      case 4:
          increaseSalary();
          break;
      case 5:
          insertRecord();
          break;

        default :
            cout<<"Invalid Choice\n";
    }
```

```
    cout<<"Do you want to continue ? : ";
    cin>>ch;

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

   return 0;
}
```

Output:

```
ENTER CHOICE
1.ADD AN EMPLOYEE
2.DISPLAY
3.SEARCH
4.INCREASE SALARY
5.INSERT RECORD
Make a choice: 1
Enter employee code: 34
Enter name: Joy
Enter salary: 25000
Record added sucessfully.
Do you want to continue ? : y
ENTER CHOICE
1.ADD AN EMPLOYEE
2.DISPLAY
3.SEARCH
4.INCREASE SALARY
5.INSERT RECORD
Make a choice: 1
Enter employee code: 50
Enter name: Sima
Enter salary: 25000
Record added sucessfully.
```

Do you want to continue ? : y
ENTER CHOICE
1.ADD AN EMPLOYEE
2.DISPLAY
3.SEARCH
4.INCREASE SALARY
5.INSERT RECORD
Make a choice: 1
Enter employee code: 23
Enter name: Nisha
Enter salary: 27000
Record added sucessfully.
Do you want to continue ? : n

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

Assignment-13

Title:

Design a mini project to implement Snake and Ladders Game using Python.

Learning Objectives:

- ✓ To understand concept & features of Python Programming.

Learning Outcome:

- Analyze working of various python file scripts (snakes_ladders.py), resources files and sound files.

Theory:

Python:

- Python is a high-level, general-purpose and a very popular programming language.
- Python programming language (latest Python 3) is being used in web development, Machine Learning applications, along with all cutting-edge technology in Software Industry.
- Python Programming Language is very well suited for Beginners, also for experienced programmers with other programming languages like C++ and Java.

❖ **Python Programming Language:**

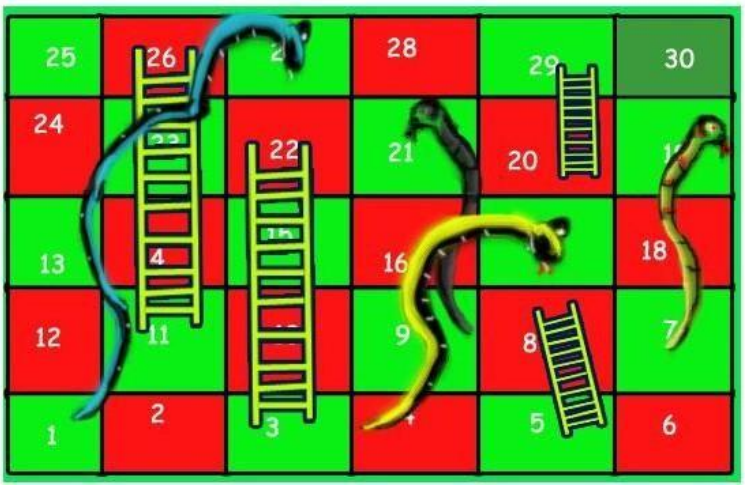
1. Python is currently the most widely used multi-purpose, high-level programming language.
2. Python allows programming in Object-Oriented and Procedural paradigms.
3. Python programs generally are smaller than other programming languages like Java. Programmers have to type relatively less and indentation requirement of the language, makes them readable all the time.
4. Python language is being used by almost all tech-giant companies like – Google, Amazon, Facebook, Instagram, Dropbox, Uber… etc.
5. The biggest strength of Python is huge collection of standard library which can be used for the following:
    - Machine Learning
    - GUI Applications (like Kivy, Tkinter, PyQt etc.)
    - Web frameworks like Django (used by YouTube, Instagram, Dropbox)
    - Image processing (like OpenCV, Pillow)
    - Web scraping (like Scrapy, BeautifulSoup, Selenium)

- Test frameworks
- Multimedia
- Scientific computing

This Snakes and Ladders Game contains python file scripts (snakes_ladders.py), resources files and sound files. The gameplay of the system, which is the user can choose an option either to play multiple participant or with the computer.

Beginning of the game, the player needs to roll the dice and in the wake of moving it the game moves the token consequently as indicated by the dice number. The interactivity is like the genuine one. Here, the player likewise gets one more opportunity to roll the dice at whatever point he/she gets 6 number.

There are quantities of stepping stools and snakes in the game which causes the player to update or minimization the square number. The player who arrives at the last square of the track is the champ.



Snakes and Ladders Game

- **Step 1: Create a project name.**

  First when you finished installed the **Pycharm IDE** in your computer, open it and then create a **–project name‖** after creating a project name click the **–create‖** button.

- **Step 2: Create a python file.**

  Second after creating a project name, **–right click‖** your project name and then click **–new‖** after that click the **–python file–**.

- **Step 3: Name your python file.**

Third after creating a python file, Name your python file after that click **–enter–**.

- **Step 4: The Python code.**
  The actual coding of how to create **Snakes and Ladders Game in Python**

Expected Output:

Implementation of Snake and Ladders Game using Python.

Program code:

```
import time
import random
import sys

# just of effects. add a delay of 1 second before performing any action
SLEEP_BETWEEN_ACTIONS = 1
MAX_VAL = 100
DICE_FACE = 6

# snake takes you down from 'key' to 'value'
snakes = {
   8: 4,
   18: 1,
   26: 10,
   39: 5,
   51: 6,
   54: 36,
   56: 1,
   60: 23,
   75: 28,
   83: 45,
   85: 59,
   90: 48,
   92: 25,
   97: 87,
   99: 63
}

# ladder takes you up from 'key' to 'value'
ladders = {
   3: 20,
   6: 14,
   11: 28,
   15: 34,
```

```
    17: 74,
    22: 37,
    38: 59,
    49: 67,
    57: 76,
    61: 78,
    73: 86,
    81: 98,
    88: 91
}

player_turn_text = [
    "Your turn.",
    "Go.",
    "Please proceed.",
    "Lets win this.",
    "Are you ready?",
    "",
]

snake_bite = [
    "boohoo",
    "bummer",
    "snake bite",
    "oh no",
    "dang"
]

ladder_jump = [
    "woohoo",
    "woww",
    "nailed it",
    "oh my God...",
    "yaayyy"
]


def welcome_msg():
    msg = """
    Welcome to Snake and Ladder Game.
    Version: 1.0.0

    Rules:
      1. Initally both the players are at starting position i.e. 0.
         Take it in turns to roll the dice.
         Move forward the number of spaces shown on the dice.
      2. If you lands at the bottom of a ladder, you can move up to the top of the ladder.
```

```
    3. If you lands on the head of a snake, you must slide down to the bottom of the snake.
    4. The first player to get to the FINAL position is the winner.
    5. Hit enter to roll the dice.

    """
    print(msg)


def get_player_names():
    player1_name = None
    while not player1_name:
        player1_name = input("Please enter a valid name for first player: ").strip()

    player2_name = None
    while not player2_name:
        player2_name = input("Please enter a valid name for second player: ").strip()

    print("\nMatch will be played between '" + player1_name + "' and '" + player2_name + "'\n")
    return player1_name, player2_name


def get_dice_value():
    time.sleep(SLEEP_BETWEEN_ACTIONS)
    dice_value = random.randint(1, DICE_FACE)
    print("Its a " + str(dice_value))
    return dice_value


def got_snake_bite(old_value, current_value, player_name):
    print("\n" + random.choice(snake_bite).upper() + " ~~~~~~~~>")
    print("\n" + player_name + " got a snake bite. Down from " + str(old_value) + " to " + str(current_value))


def got_ladder_jump(old_value, current_value, player_name):
    print("\n" + random.choice(ladder_jump).upper() + " ########")
    print("\n" + player_name + " climbed the ladder from " + str(old_value) + " to " + str(current_value))


def snake_ladder(player_name, current_value, dice_value):
    time.sleep(SLEEP_BETWEEN_ACTIONS)
    old_value = current_value
    current_value = current_value + dice_value

    if current_value > MAX_VAL:
        print("You need " + str(MAX_VAL - old_value) + " to win this game. Keep trying.")
        return old_value
```

```
    print("\n" + player_name + " moved from " + str(old_value) + " to " + str(current_value))
    if current_value in snakes:
        final_value = snakes.get(current_value)
        got_snake_bite(current_value, final_value, player_name)

    elif current_value in ladders:
        final_value = ladders.get(current_value)
        got_ladder_jump(current_value, final_value, player_name)

    else:
        final_value = current_value

    return final_value


def check_win(player_name, position):
    time.sleep(SLEEP_BETWEEN_ACTIONS)
    if MAX_VAL == position:
        print("\n\n\nThats it.\n\n" + player_name + " won the game.")
        print("Congratulations " + player_name)
        print("\nThank you for playing the game!!\n\n")
        sys.exit(1)


def start():
    welcome_msg()
    time.sleep(SLEEP_BETWEEN_ACTIONS)
    player1_name, player2_name = get_player_names()
    time.sleep(SLEEP_BETWEEN_ACTIONS)

    player1_current_position = 0
    player2_current_position = 0

    while True:
        time.sleep(SLEEP_BETWEEN_ACTIONS)
        input_1 = input("\n" + player1_name + ": " + random.choice(player_turn_text) + " Hit the enter to roll
dice: ")
        print("\nRolling dice...")
        dice_value = get_dice_value()
        time.sleep(SLEEP_BETWEEN_ACTIONS)
        print(player1_name + " moving....")
        player1_current_position = snake_ladder(player1_name, player1_current_position, dice_value)

        check_win(player1_name, player1_current_position)

        input_2 = input("\n" + player2_name + ": " + random.choice(player_turn_text) + " Hit the enter to roll
dice: ")
```

```
        print("\nRolling dice...")
        dice_value = get_dice_value()
        time.sleep(SLEEP_BETWEEN_ACTIONS)
        print(player2_name + " moving....")
        player2_current_position = snake_ladder(player2_name, player2_current_position, dice_value)

        check_win(player2_name, player2_current_position)


if __name__ == "__main__":
    start()
```

Output:

```
Welcome to Snake and Ladder Game.
    Version: 1.0.0

    Rules:
      1. Initally both the players are at starting position i.e. 0.
         Take it in turns to roll the dice.
         Move forward the number of spaces shown on the dice.
      2. If you lands at the bottom of a ladder, you can move up to the top of the ladder.
      3. If you lands on the head of a snake, you must slide down to the bottom of the snake.
      4. The first player to get to the FINAL position is the winner.
      5. Hit enter to roll the dice.


Please enter a valid name for first player: chiki
Please enter a valid name for second player: mickey

Match will be played between 'chiki' and 'mickey'


chiki: Are you ready? Hit the enter to roll dice:

Rolling dice...
Its a 3
chiki moving....

chiki moved from 0 to 3

WOWW ########

chiki climbed the ladder from 3 to 20

mickey: Lets win this. Hit the enter to roll dice:

Rolling dice...
```

Its a 2
mickey moving....

mickey moved from 0 to 2

chiki: Your turn. Hit the enter to roll dice:

Rolling dice...
Its a 4
chiki moving....

chiki moved from 20 to 24

mickey: Your turn. Hit the enter to roll dice:

Rolling dice...
Its a 6
mickey moving....

mickey moved from 2 to 8

DANG ~~~~~~~~>

mickey got a snake bite. Down from 8 to 4

chiki: Hit the enter to roll dice:

Rolling dice...
Its a 3
chiki moving....

chiki moved from 24 to 27

mickey: Go. Hit the enter to roll dice:

Rolling dice...
Its a 4
mickey moving....

mickey moved from 4 to 8

DANG ~~~~~~~~>

mickey got a snake bite. Down from 8 to 4

chiki: Are you ready? Hit the enter to roll dice:

Rolling dice...
Its a 4
chiki moving....

chiki moved from 27 to 31

mickey: Your turn. Hit the enter to roll dice:

Rolling dice...
Its a 3
mickey moving....

mickey moved from 4 to 7

chiki: Hit the enter to roll dice:

Rolling dice...
Its a 2
chiki moving....

chiki moved from 31 to 33

mickey: Lets win this. Hit the enter to roll dice:

Rolling dice...
Its a 1
mickey moving....

mickey moved from 7 to 8

BUMMER ~~~~~~~~>

mickey got a snake bite. Down from 8 to 4

chiki: Please proceed. Hit the enter to roll dice:

Rolling dice...
Its a 2
chiki moving....

chiki moved from 33 to 35

mickey: Are you ready? Hit the enter to roll dice:

Rolling dice...
Its a 4
mickey moving....

mickey moved from 4 to 8

SNAKE BITE ~~~~~~~~>

mickey got a snake bite. Down from 8 to 4

chiki: Go. Hit the enter to roll dice:

Rolling dice...
Its a 1
chiki moving....

chiki moved from 35 to 36

mickey: Are you ready? Hit the enter to roll dice:

Rolling dice...
Its a 1
mickey moving....

mickey moved from 4 to 5

chiki: Are you ready? Hit the enter to roll dice:

Rolling dice...
Its a 3
chiki moving....

chiki moved from 36 to 39

DANG ~~~~~~~~>

chiki got a snake bite. Down from 39 to 5

mickey: Are you ready? Hit the enter to roll dice:

Rolling dice...
Its a 2
mickey moving....

mickey moved from 5 to 7

chiki: Are you ready? Hit the enter to roll dice:

Rolling dice...
Its a 6

chiki moving....

chiki moved from 5 to 11

YAAYYY ########

chiki climbed the ladder from 11 to 28

mickey: Hit the enter to roll dice:

Rolling dice...
Its a 5
mickey moving....

mickey moved from 7 to 12

chiki: Hit the enter to roll dice:

Rolling dice...
Its a 6
chiki moving....

chiki moved from 28 to 34

mickey: Your turn. Hit the enter to roll dice:

Rolling dice...
Its a 1
mickey moving....

mickey moved from 12 to 13

chiki: Go. Hit the enter to roll dice:

Rolling dice...
Its a 3
chiki moving....

chiki moved from 34 to 37

mickey: Hit the enter to roll dice:

Rolling dice...
Its a 1
mickey moving....

mickey moved from 13 to 14

chiki: Please proceed. Hit the enter to roll dice:

Rolling dice...
Its a 4
chiki moving....

chiki moved from 37 to 41

mickey: Please proceed. Hit the enter to roll dice:

Rolling dice...
Its a 3
mickey moving....

mickey moved from 14 to 17

YAAYYY ########

mickey climbed the ladder from 17 to 74

chiki: Go. Hit the enter to roll dice:

Rolling dice...
Its a 6
chiki moving....

chiki moved from 41 to 47

mickey: Your turn. Hit the enter to roll dice:

Rolling dice...
Its a 5
mickey moving....

mickey moved from 74 to 79

chiki: Your turn. Hit the enter to roll dice:

Rolling dice...
Its a 1
chiki moving....

chiki moved from 47 to 48

mickey: Hit the enter to roll dice:

Rolling dice...
Its a 2
mickey moving....

mickey moved from 79 to 81

NAILED IT ########

mickey climbed the ladder from 81 to 98

chiki: Your turn. Hit the enter to roll dice:

Rolling dice...
Its a 4
chiki moving....

chiki moved from 48 to 52

mickey: Your turn. Hit the enter to roll dice:

Rolling dice...
Its a 6
mickey moving....
You need 2 to win this game. Keep trying.

chiki: Go. Hit the enter to roll dice:

Rolling dice...
Its a 6
chiki moving....

chiki moved from 52 to 58

mickey: Go. Hit the enter to roll dice:

Rolling dice...
Its a 3
mickey moving....
You need 2 to win this game. Keep trying.

chiki: Please proceed. Hit the enter to roll dice:

Rolling dice...
Its a 6
chiki moving....

chiki moved from 58 to 64

mickey: Are you ready? Hit the enter to roll dice:

Rolling dice...
Its a 5
mickey moving....
You need 2 to win this game. Keep trying.

chiki: Hit the enter to roll dice:

Rolling dice...
Its a 1
chiki moving....

chiki moved from 64 to 65

mickey: Your turn. Hit the enter to roll dice:

Rolling dice...
Its a 6
mickey moving....
You need 2 to win this game. Keep trying.

chiki: Go. Hit the enter to roll dice:

Rolling dice...
Its a 2
chiki moving....

chiki moved from 65 to 67

mickey: Lets win this. Hit the enter to roll dice:

Rolling dice...
Its a 2
mickey moving....
mickey moved from 98 to 100

Thats it.
mickey won the game.
Congratulations mickey

Thank you for playing the game.

Conclusion:
Implementation of a mini project using python.