QUESTION BANK

CS6301-Programming and Data structure –II

II Year –III Semester

Prepared By

S.Vanitha

AP, CSE Dept.
OBJECTIVES:
The student should be made to:
Be familiar with the C++ concepts of abstraction, encapsulation, constructor, polymorphism, overloading and Inheritance.
Learn advanced nonlinear data structures.
Be exposed to graph algorithms.
Learn to apply Tree and Graph structures.

UNIT I
OBJECT ORIENTED PROGRAMMING FUNDAMENTALS
static members – constant members – member functions – pointers – references – Role of this pointer –
Storage classes – function as arguments.

UNIT II
OBJECT ORIENTED PROGRAMMING CONCEPTS
String Handling – Copy Constructor – Polymorphism – compile time and run time polymorphisms –
function overloading – operators overloading – dynamic memory allocation – Nested classes –
Inheritance – virtual functions.

UNIT III
C++ PROGRAMMING ADVANCED FEATURES
Abstract class – Exception handling – Standard libraries – Generic Programming – templates –
Parameterizing the class – File handling concepts.

UNIT IV
ADVANCED NON-LINEAR DATA STRUCTURES

UNIT V
GRAPHS
Representation of Graphs – Breadth-first search – Depth-first search – Topological sort –

TOTAL: 45 PERIODS

TEXT BOOKS:
1. What is C++ and OOPS?

C++ is a general-purpose programming language with a bias towards systems programming that – is a better C, – supports data abstraction, – supports object-oriented programming, and – supports generic programming.

OOPS: Object Oriented Programming is an approach that provides a way of modularizing programs by creating partitioned memory area for both data and functions that can be used as templates for creating copies of such modules on demand

What are advantages of OOPS?

- OOP provides a clear modular structure for programs.
- It is good for defining abstract data types.
- Implementation details are hidden from other modules and other modules has a clearly defined interface.
- It is easy to maintain and modify existing code as new objects can be created with small differences to existing ones.
- objects, methods, instance, message passing, inheritance are some important properties provided by these particular languages
- encapsulation, polymorphism, abstraction are also counts in these fundamentals of programming language.
- It implements real life scenario.
- In OOP, programmer not only defines data types but also deals with operations applied for data structures.
2. List the features of OOPS.

- More reliable software development is possible.
- Enhanced form of C programming language.
- The most important Feature is that it’s procedural and object-oriented nature.
- Much suitable for large projects.
- Fairly efficient languages.
- It has the feature of memory management.

3. What are the features of C++?

The main features of the C++ are

- Classes
- Inheritance
- Data abstraction and encapsulation
- Polymorphism
- Dynamic Binding
- Message Passing

4. Define Class and Object?

The class is the collection of set of data and code. A class is a user-defined type provided to represent a concept in the code of a program.

The syntax for class is:

Class <class-name>

{

//Body of class;

};

Class is a user defined data type, which holds its own data members and member functions, which can be accessed and used by creating instance of that class.

The variables inside class definition are called as data members and the functions are called member functions.

For example: Class of birds, all birds can fly and they all have wings and beaks.
So here flying is a behavior and wings and beaks are part of their characteristics. And there are many different birds in this class with different names but they all posses this behavior and characteristics.

Syntax:

Class class_name
{
    Data Members;
    Methods;
}

class A
{
    public:
    double length; // Length of a box
    double breadth; // Breadth of a box
    double height; // Height of a box
}

Object: Objects are the basic run time entities in an object oriented system. They are instance of a class. They may represent a person, a place etc that a program has to handle. They may also represent user-defined data. They contain both data and code.

4. State dynamic initialization of objects.
   Class objects can be initialized dynamically. The initial values of an object may be provided during run time. The advantage of dynamic initialization is that various initialization formats can be used. It provides flexibility of using different data formats.

5. What is Inheritance?
   Inheritance allows one data type to acquire properties of other data types. Inheritance from a base class may be declared as public, protected, or private.

   If the access specifier is omitted, a “class” inherits privately, while a “struct” inherits publicly. This provides the idea of reusability that means it can add the new features to an existing class without modifying it.

6. Define Data Abstraction?
   Data Abstraction is defines as a clear separation between properties of data type and the associated implementation details.

   There are two types, they are "function abstraction" and "data abstraction".

Functions that can be used without knowing how its implemented is function abstraction.

Data abstraction is using data without knowing how the data is stored.
For example, your program can make a call to the `sort()` function without knowing what algorithm the function actually uses to sort the given values.

Data abstraction refers to, providing only essential features by hiding its background details.

**example:**
```cpp
class result{
    int marks;
    float percentage;
    char name[20];
    void input();
    void output();
}

main()
{
    bank b1;
    b1.input();
    b1.output();
}
```

in the above example, b1 is an object calling input and output member functions, but that code is invisible to the object b1.

**7. Write the benefits of Data Abstraction.**

Data abstraction provides two important advantages:

- Class internals are protected from inadvertent user-level errors, which might corrupt the state of the object.

- The class implementation may evolve over time in response to changing requirements or bug reports without requiring change in user-level code.

**8. What are the two fundamentals elements of C++ program?**

C++ programs are composed of the following two fundamental elements:

- **Program statements (code):** This is the part of a program that performs actions and they are called functions.

- **Program data:** The data is the information of the program which affected by the program functions.

**9. Define Data Encapsulation and data abstraction.**
**Data encapsulation** is a mechanism of bundling the data, and the functions that use them and **data abstraction** is a mechanism of exposing only the interfaces and hiding the implementation details from the user. The insulation of data from direct access by the program is called **data hiding** or **information hiding**.

10. **What is Encapsulation?**

   Encapsulation means hiding of data from the data structures.

   The wrapping up of data in single entity is known as Encapsulation. The data is not accessible to outside world and only the functions are allowed to access it.

11. **What is Polymorphism?**

   Polymorphism means that the one interface can be used for many implementations so that object can behave differently for each implementation. Polymorphism means one name, multiple forms. It is the ability of a function or operator to take more than one form at different instances.

   The different types of polymorphism are static (Compile time) and dynamic (Run time).

12. **What is dynamic binding?**

   It means that the linking of a procedure call to code to be executed in response to the call. A function call associated with a polymorphic reference depends on the dynamic type that reference. And at run-time the code matching the object under current reference will be called.

   **Message Passing**: Objects communicate between each other by sending and receiving information known as messages. A message to an object is a request for execution of a procedure. Message passing involves specifying the name of the object, the name of the function and the information to be sent.

13. **In what way is a private member function different from public member function?**

   A private member function can only be called by another function that is a member of its class. Even an object cannot invoke a private function using the dot operator.

14. **List the features of Class.**

   1. Class name must start with an uppercase letter. If class name is made of more than one word, then first letter of each word must be in uppercase. Example, class Study, class Student etc

   2. Classes contain, data members and member functions, and the access of these data members and variable depends on the access specifiers

   3. Class's member functions can be defined inside the class definition or outside the class definition.
4. Class in C++ are similar to structures in C, the only difference being, class defaults to private access control, whereas structure defaults to public.

5. Classes possess more characteristics, like we can create abstract classes, immutable classes.

15. What is an Object?

Objects are instances of class, which holds the data variables declared in class and the member functions work on these class objects.

Each object has different data variables. Objects are initialized using special class functions called Constructors.

**Example.**

```cpp
class Abc
{
    int x;
    void display(){} //empty function
};
int main()
{
    Abc obj; // Object of class Abc created
}
```

16. Write a program to declare Class and Object.

**Syntax:**

```cpp
class ClassName
{
    Access specifier:
    Data members;
    Member Functions()
};
```

*simple class named Student with appropriate members,*
A class definition ends with a semicolon, or with a list of object declarations.

Example:

```cpp
class Student
{
public:
    int rollno;
    string name;
};
```

Also declare objects separately, like declaring variable of primitive data types and that data type can be a class name, and variable is the object.

```cpp
int main()
{
    Student A;
    Student B;
}
```

Both A and B will have their own copies of data members.

17. Define data member and data function.

The data within the class is known as data member. The function defined within the class is known as member function.

Example:

```cpp
class temp
{
```

class temp, data1 and data2 are data members and func1() and func2() are member functions

18. What is Constructor?

Constructors are special class functions which performs initialization of every object. The Compiler calls the Constructor whenever an object is created. Constructors initialize values to object members after storage is allocated to the object.

```cpp
class A
{
    int x;
public:
    A();  //Constructor
};
```

- Constructor has same name as that of class and it does not have any return type
- Constructors can be defined either inside the class definition or outside class definition using class name and scope resolution :: operator.

Example:
class A
{
    int i;
    public:
    A(); //Constructor declared
};
A::A() // Constructor definition
{
    i=1;
}
19. Write the types of Constructor.

Constructors are of three types:

1. Default Constructor
2. Parametrized Constructor
3. Copy Constructor

20. What is Default constructor?

Default constructor is the constructor which doesn't take any argument. It has no parameter.

Syntax:

class_name ()
{
    Constructor Definition
}

Example:

class Cube
{
    int side;
    public:
    Cube()
    {
}
21. What is Parameterized Constructor?

These are the constructors with parameter. It can provide different values to data members of different objects, by passing the appropriate values as argument.

Example:

class Cube
{
    int side;

public:
    Cube(int x)
    {
        side=x;
    }
};

int main()
{
    Cube c1(10);
    Cube c2(20);
    Cube c3(30);
    cout << c1.side;
    cout << c2.side;
    cout << c3.side;
}
cout << c2.side;
cout << c3.side;
} OUTPUT : 10 20 30

22. What is Copy Constructor?

These are special type of Constructors which takes an object as argument, and is used to copy values of data members of one object into other object.

Example.

```cpp
int main() {
    Area A1,A2(2,1);
    Area A3(A2); // Copies the content of A2 to A3 */
    OR,
    Area A3=A2; // Copies the content of A2 to A3 */
}
```

23. What is destructor?

Destructor is a special class function which destroys the object as soon as the scope of object ends. The destructor is called automatically by the compiler when the object goes out of scope.

The syntax for destructor is same as that for the constructor, the class name is used for the name of destructor, with a tilde ~ sign as prefix to it.

```cpp
class A {
    public:
        ~A();
};
```

Destructors will never have any arguments.

24. Define static member.
A static member is shared by all objects of the class. All static data is initialized to zero when the first object is created, if no other initialization is present.

It must be initialized explicitly, always outside the class. If not initialized, Linker will give error.

Example:
```cpp
class X
{
    static int i;
    public:
    X(){};
};
int X::i=1;
int main()
{
    X obj;
    cout << obj.i;  // prints value of i
}
```

24. Define constant member.

These are data variables in class which are made const. They are not initialized during declaration. Their initialization occurs in the constructor.

Example:
```cpp
class Test
{
    const int i;
    public:
    Test (int x)
    {
        i=x;
    }
```
int main()
{
    Test t(10);
    Test s(20);
}

A **const member function** is a member function that guarantees it will not change any class variables or call any non-const member functions.

25. **What are Pointers?**

A **pointer** is a variable whose value is the address of another variable. Like any variable or constant, you must declare a pointer before you can work with it. The general form of a pointer variable declaration is:

```cpp
    type *var-name;
    int *p;
    OR,
    int* p;
```

Here, `type` is the pointer's base type; it must be a valid C++ type and `var-name` is the name of the pointer variable.

26. **How will you to declare references in C++?**

References are like constant pointers that are automatically dereferenced. It is a new name given to an existing storage.

Example:

```cpp
int main()
{
    int y=10;
    int &r = y;  // r is a reference to int y
    cout << r;
}
```

Output : 10 There is no need to use the `*` to dereference a reference variable.
27. Difference between Reference and Pointer

<table>
<thead>
<tr>
<th>References</th>
<th>Pointers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reference must be initialized when it is created.</td>
<td>Pointers can be initialized any time</td>
</tr>
<tr>
<td>Once initialized, we cannot reinitialize a reference.</td>
<td>Pointers can be reinitialized any number of time.</td>
</tr>
<tr>
<td>can never have a NULL reference.</td>
<td>Pointers can be NULL.</td>
</tr>
<tr>
<td>Reference is automatically dereferenced.</td>
<td>* is used to dereference a pointer.</td>
</tr>
</tbody>
</table>

28. Write about the Role of this pointer.

The ‘this’ pointer is passed as a hidden argument to all nonstatic member function calls and is available as a local variable within the body of all nonstatic functions. ‘this’ pointer is a constant pointer that holds the memory address of the current object.

Following are the situations where ‘this’ pointer is used:

1) When local variable’s name is same as member’s name

2) To return reference to the calling object

Or

The this keyword is used to represent an object that invokes the member function. It points to the object for which this function was called. It is automatically passed to a member function when it is called.

e.g. when you call A.func(), this will be set to the address of A.

29. What are storage classes?

A storage class defines the scope (visibility) and life-time of variables and/or functions within a C++ Program. These specifiers precede the type that they modify. There are following storage classes, which can be used in a C++ Program

- auto
- register
- static
- extern
mutable

Storage classes are used to specify the lifetime and scope of variables and storage is allocated for variables and variable is treated by compiler depends on these storage classes.

These are basically divided into 5 different types:

1. Global variables
2. Local variables
3. Register variables
4. Static variables
5. Extern variables

30. What is a friend function?

Friend function is a special type of function which is used to access all the private and protected members of a class. The functions that are declared with the keyword friend are called friend functions. A function can be a friend to multiple classes.

31. What are the characteristics of a static data member?

Static data member is initialized to zero when the first object of its class is created. No other initialization is permitted. Only one copy of that member is created for the entire class and is shared by all the objects of that class, no matter how many objects are created. It is visible only within the class, but its lifetime is the entire program.

32. What are the properties of a static member function?

A static function can have access to only other static members, declared in the same class. A static member function can be called using the class name as follows

Classname :: function-name;

33. List out the applications of OOP.

- Real time systems
- Simulation and modeling
- Object oriented databases
- Hypertext, Hypermedia and expertext
- AI and expert systems
- Neural networks and parallel programming
- Decision support and office automation systems
- CIM/CAM/CAD systems
34. List out some of the benefits of OOP.
   - Eliminate redundant code
   - Saves development time and leads to higher productivity
   - Helps to build secure programs
   - Easy to partition work
   - Small programs can be easily upgraded to large programs
   - Software complexity can easily be managed

35. What are the characteristics of member functions?
The various characteristics of member functions are,
   - Different classes can use the same function name and their scope can be resolved using the membership label.
   - Member functions can access the private data of a class while a nonmember function cannot.
   - A member function can call another member function directly without using a dot operator.

36. List out the special characteristics of a friend function.
   - It is not in the scope of a class in which it is declared as friend.
   - It cannot be called using the object of that class.
   - It can be invoked without an object.
   - It cannot access the member names directly and uses the dot operator.
   - It can be declared as either public or private.
   - It has the objects as arguments.

37. How can objects be used as function arguments?
An object can be used as a function argument in two ways,
   - A copy of the entire object is passed to the function. (Pass by value)
   - Only the address of the object is transferred to the function. (Pass by reference)

38. Why do we use default arguments?
The function assigns a default value to the parameter which does not have a matching argument in the function call. They are useful in situations where some arguments always have the same value.
   e.g., float amt (float P, float n, float r = 0.15);

39. State the advantages of default arguments.
The advantages of default arguments are,
   - We can use default arguments to add new parameters to the existing function.
   - Default arguments can be used to combine similar functions into one.

PART-B
1. Explain the basic concepts of Object oriented programming
2. Write about features of C++
3. Explain the idea of Classes, Data abstraction and encapsulation.
4. What are the copy constructors and explain with example?
5. (i) What are the rules to be followed in function overloading? (ii) Write a C++ program that can take either two integers or two floating point numbers and outputs the smallest number using class, friend functions and function overloading
6. (i) Explain the need for pure virtual functions. (ii) Write a C++ program for calculating the area of rectangle and circle using run-time polymorphism.
7. Explain briefly about the constructors and destructors with example.
8. What do you understand by static member and static function? How to declare them? Illustrate with an example program.
9. Define friend class and specify its importance. Explain with suitable example.
10. Explain the role of this pointer with a suitable program.
11. What are the difference between pointer to constants and constant to pointers? Give an example program and explain it.
12. Write a program to get the student details and print the same using pointers to objects and pointers to members of a class. Create a class student. And use appropriate functions and data members.
13. Write a program to demonstrate how a static data is accessed by a static member function.
14. I. Discuss in detail about default arguments with an example.
   ii Write a program to demonstrate how a static data is accessed by a static member function.
15. What is function overloading? Explain briefly with program.
16. Discuss the different types of inheritance supported in C++ with suitable illustration
17. Write short notes on i. Static Member function ii. This Pointer and references
18. Write a C++ program to generate Fibonacci using Copy Constructor. Ii. Explain Friend Function with example.
19. Write a C++ program to construct student mark list for three subjects. Write the program to display name, rollno, marks, avg and total. Use class and objects.
20. Write a program to demonstrate how a static data is accessed by a static member function.
21. What are the advantages of using default arguments? Explain with an example program.
1. What are the types of strings used in C++?

There are two types of strings commonly used in C++ programming language:

- Strings that are objects of string class (The Standard C++ Library string class)
- C-strings (C-style Strings)

2. Write a C++ program to display a string entered by user.

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

int main() {
    char str[100];
    cout << "Enter a string: ";
    cin >> str;
    cout << "You entered: " << str << endl;
    cout << "Enter another string: ";
    cin >> str;
    cout << "You entered: " << str << endl;
    return 0;
}
```

Output

Enter a string: C++
You entered: C++
Enter another string: Programming is fun.
You entered: Programming

3. Define String

A collection of characters written in double quotation is called string. It may consist of alphabetic characters, digits and special symbols.

Eg. Int a[25]; Char ch;

A String variable can be initialized with a string value as follows:

Char str[50] = "OOPS in C++";

It can also be initialized without defining the length of string:

Char a[] = "OOPS";

4. List out the string manipulation functions.

- strcmp() - used to compare two string character by character
Syntax: Strcmp(str1,str2);
• strcpy() - used to copies the contents of one string to another.
  Syntax: strcpy(str1,str2);
• strcat() - used to append a copy of one string to the end of another
  Syntax: strcat(str1,str2);
• strlen() - used to find the length of string (includes character & space)
  Syntax: strlen(str);

5. How will you declare and define a constructor

A constructor is a special member function whose task is to initialize the objects of its class. It has the same name as the class. It gets invoked whenever an object is created to that class. It is called so since it constructs the values of data members of the class.

Constructor declaration :

Class integer
{
  int m,n;
  public :
}

integer ( void )

constructor definition

integer :: integer ( void )
{
  m=0; n=0;
}

6. What are the characteristics of a constructor?

- They should be declared in the public section.
- They are invoked automatically when the objects are created.
- They do not have return types, not even void and therefore, they cannot return values.
- They cannot be inherited, though a derived class can call the base class constructor.

7. What are the ways in which a constructor can be called?

- The constructor can be called by two ways. They are,
  - By calling the constructor explicitly
7. What is a parameterized constructor?

The constructors that can take arguments are called parameterized constructors.

The constructor that takes arguments are called parameterized constructor. When a constructor has been parameterized the object declaration statement such as integer i will not work.

8. Give the various types of constructors.

There are four types of constructors. They are

- **Default constructors** – A constructor that accepts no parameters
- **Parameterized constructors** – The constructors that can take arguments
- **Copy constructor** – It takes a reference to an object of the same class as itself as an argument
- **Dynamic constructors** – Used to allocate memory while creating objects

9. Define copy constructor

A **copy constructor** is a special constructor that initializes a new object from an existing object.

Copy constructor are invoked whenever

1. A newly-created object is initialized to the value of an existing object.
2. An object is passed to a function as a non-reference parameter.
3. An object is returned from a function.

Eg. MyClass one;
    MyClass two = one;
    MyClass three = two;

10. Difference between the copy constructor and assignment

- If a new object has to be created before the copying can occur, the copy constructor is used.
- If a new object does not have to be created before the copying can occur, the assignment operator is used.

11. What is meant by Polymorphism?

The word **polymorphism** means having many forms. Polymorphism means that a call to a member function will cause a different function to be executed depending on the type of object that invokes the function.

A single function usage or an operator functioning in many ways can be called polymorphism. Polymorphism refers to codes, operations or objects that behave differently in different contexts.

Eg:
Class 2DObject
{
 public:
 int x_pos;
 int y_pos;
};
Class Circle: public 2DObject
{
 public:
 int radius;
};

12. What is virtual and pure virtual function?

A virtual function is a function in a base class that is declared using the keyword virtual.

A virtual function is a member function that is declared within a base class and redefined by a
derived class.

A function that has no body is called pure virtual function. Classes that contain at least one pure
virtual function are known as abstract base classes.

Ex. class Shape {
 protected:
 int width, height;
 public:
 Shape( int a=0, int b=0)
 {
     width = a;
     height = b;
 }
 // pure virtual function
 virtual int area() = 0;
};

13. What are the rules for virtual functions?

Virtual functions must be members of some class.
• They cannot be static members.
• They are accessed by using object pointers.
• A virtual function can be a friend of another class.
• We can have virtual destructors, but we cannot have virtual constructors

14. Types of Polymorphism:

C++ provides two different types of polymorphism.
• run-time
• compile-time
15. Difference between the compile time and run time polymorphisms.

<table>
<thead>
<tr>
<th>Compile time polymorphism</th>
<th>Runtime polymorphisms</th>
</tr>
</thead>
<tbody>
<tr>
<td>Appropriate member function could be selected while the programming is running</td>
<td>The compiler is able to select the appropriate function for a particular call at compile-time itself</td>
</tr>
<tr>
<td>It is implemented with</td>
<td>Compile-time polymorphism is implemented with templates.</td>
</tr>
<tr>
<td>- inheritance</td>
<td>- Function name overloading</td>
</tr>
<tr>
<td>- Virtual functions.</td>
<td>- Operator overloading</td>
</tr>
<tr>
<td>- Dynamic Binding</td>
<td></td>
</tr>
</tbody>
</table>

16. What is meant by function overloading?

A single function name can be used to perform different types of tasks. The same function name can be used to handle different number and different types of arguments. This is known as function overloading or function polymorphism.

Eg: An overloaded `add()` function handles different data types as shown below.

```c++
// Declarations
i. int add( int a, int b); //add function with 2 arguments of same type
ii. int add( int a, int b, int c); //add function with 3 arguments of same type
iii. double add( int p, double q); //add function with 2 arguments of different type
```

17. What is operator overloading?

C++ has the ability to provide the operators with a special meaning for a data type.

The process of making an operator to exhibit different behaviors in different instances is known as operator overloading. The general form of an operator function is:

```c++
return type classname: : operator(op-arglist)
{
    Function body
}
```

The process of overloading involves the following steps:

1. Create a class that defines the data type that is to be used in the overloading operation.
2. Declare the operator function `operator op ()` in the public part of the class. It may be either a member function or a friend function.
3. Define the operator function to implement the required operations.

18. List out the operators that cannot be overloaded.
• Class member access operator (. , .*)
• Scope resolution operator (::)
• Size operator ( sizeof )
• Conditional operator (?:)

• Only existing operators can be overloaded.
• We cannot change the basic meaning of an operator.
• The overloaded operator must have at least one operand.
• Overloaded operators follow the syntax rules of the original operators.

19. What are the ways to allocate the memory for data storage?
There are two ways that memory gets allocated for data storage:

1. Compile Time (or static) Allocation
   • Memory for named variables is allocated by the compiler
   • Exact size and type of storage must be known at compile time
   • For standard array declarations, this is why the size has to be constant

2. Dynamic Memory Allocation
   • Memory allocated "on the fly" during run time
   • dynamically allocated space usually placed in a program segment known as the heap or
     the free store
   • Exact amount of space or number of items does not have to be known by the compiler in
     advance.
   • For dynamic memory allocation, pointers are crucial

20. What is Dynamic memory allocation?
Dynamically allocate storage space while the program is running and it can be allocated using the
new operator. Dynamically allocated memory must be referred to by pointers. Dynamic allocation
requires two steps:

1. Creating the dynamic space.
2. Storing its address in a pointer (so that the space can be accessed)

Eg. use the unary operator new

    new int;       // dynamically allocates an int
    new double;   // dynamically allocates a double
    new int[40]   // dynamically allocates an array of 40 ints

21. What is De- allocation?
Deallocation is the "clean-up" of space being used for variables or other data storage.
• Compile time variables are automatically deallocated based on their known.
• To de-allocate dynamic memory, use the delete operator
  Eg.
    int * ptr = new int;       // dynamically created int
22. Define Nested Classes.

Classes can be defined inside other classes. Classes that are defined inside other classes are called nested classes that can be used within the scope of the class in which it is defined.

```cpp
#include <iostream.h>
class Nest
{
public:
  class Display
  {
    private:
      int s;
    public:
      void sum(int a, int b)
      { s =a+b; }
      void show()
      { cout << "nSum of a and b is:: " << s; }
    }
  }

  void main()
  {
    Nest::Display x;
    x.sum(12, 10);
    x.show();
  }
}
```

OUTPUT: Sum of a and b is::22

23. What is dynamic binding?

Binding refers to the linking of a procedure call to the code to be executed in response to the call. Dynamic binding also known as late binding means that the code associated with a given procedure call is not known until the time of call at run time.

24. What are the properties of pure virtual function?

- It has no definition in the base class.
- We cannot create object for the class containing pure virtual function.
- Pure virtual function can be called through derived class objects.
- It acts as an empty bucket which has to be filled by its derived classes.

25. What are the drawbacks of static memory allocation?

- Excess wastage of memory or insufficient memory.
- Inefficient, because memory allocated is not freed

26. How an overloaded operator can be invoked using Friend functions?
In case of unary operators, overloaded operator can be invoked as
Operator op (x);
In case of binary operators, overloaded operator can be invoked as
Operator op (x , y)

27. What is a Friend Function?
A friend function is used for accessing the non-public members of a class. A class can
allow non-member functions and other classes to access its own private data, by making them
friends. Thus, a friend function is an ordinary function or a member of another class.

28. List out the operators that cannot be overloaded using Friend function.
_ Assignment operator =
_ Function call operator ()
_ Subscripting operator [ ]
_ Class member access operator

29. Define inheritance?
Inheritance is the process of creating new classes from the existing classes. The new classes are
called derived classes. The existing classes are called base classes. The derived classes inherit all the
properties of the base class plus its own properties. The properties of inheritance do not affect the base
classes.

30. What are the types of inheritance?
- Single inheritance
- Multilevel inheritance
- Multiple inheritances.
- Hierarchical inheritance
- Hybrid inheritance
- Multipath inheritance

31. Define abstract class
A class is said to be an abstract class if it satisfies the following conditions:
- It should act as a base class
- It should not be used to create any objects

32. Define Base & Derived Classes:

Base Class:
It is the class whose properties are inherited by another class. It is also called Super class.

Derived Class:
It is the class that inherits properties from base class. It is also called sub class.

The existing class is called the base class and the new class is referred to as the derived class.

A class can be derived from more than one class, which means it can inherit data and functions
from multiple base classes.

Syntax: class derived-class: access-specifier base-class

Where access-specifier is one of public, protected, or private.
33. Define the types of Inheritance.

**Single Inheritance:**

In this type of inheritance one derived class inherits from only one base class. It is the simplest form of Inheritance.

![Diagram of Single Inheritance]

**Multiple Inheritance**

In this type of inheritance a single derived class may inherit from two or more than two base classes.

![Diagram of Multiple Inheritance]

**Hierarchical Inheritance**

In this type of inheritance, multiple derived classes inherits from a single base class.

![Diagram of Hierarchical Inheritance]

**Multilevel Inheritance**

In this type of inheritance the derived class inherits from a class, which in turn inherits from some other class. The Super class for one is sub class for the other.
Hybrid (Virtual) Inheritance

Hybrid Inheritance is combination of Hierarchical and Multilevel Inheritance.

34. **What are the restriction and limitations of overloading operators?**
   Operator function must be member functions or friend functions. The overloading operator must have at least one operand that is user defined datatype.

35. **Define unary and binary operator overloading?**
   Overloading without explicit argument to an operator function is known as **Unary operator overloading**.
   Overloading with single explicit argument is known **Binary operator overloading**.

36. **What is a virtual base class?**

   Here, class D inherits both classes B and C which are derived from the same base class A. Hence D has two copies of the properties of class A. This can be avoided by declaring classes B and C as virtual base classes.

37. **How can a private member be made inheritable?**
A private member can be made inheritable by declaring the data members as protected. When declared as protected the data members can be inherited by the friend classes.

38. What are the rules for virtual function?
   - They cannot be static members
   - They can access by using object pointers
   - A virtual function can be a friend of another class.

39. What is RTTI?
   Runtime type identification (RTTI) lets you find the dynamic type of an object when you have only a pointer or a reference to the base type. RTTI is the official way in standard C++ to discover the type of an object and to convert the type of a pointer or reference (that is, dynamic typing). The need came from practical experience with C++. RTTI replaces many homegrown versions with a solid, consistent approach.

PART-B

1. Explain copy constructor with suitable example

2. What is a parameterized constructor? Explain with example.

3. Describe the syntax of multiple inheritances. When do we use such an inheritance?

4. What is a virtual function? When do we make a virtual function “pure”?

5. What is operator overloading? Overload the numerical operators ‘+’ and ‘/’ for complex numbers “addition” and “division” respectively.

6. Define friend class and specify its importance. Explain with suitable example.

7. Explain the concept of inheritance by considering an example of “vehicle”.

8. Define polymorphism. Explain the different types of polymorphism.

9. Explain the operators used for dynamic memory allocation with examples.

10. Write a C++ program to define overloaded constructor and to perform string initialization and string copy.

11. Illustrate the use of copy constructor and function overloading with C++ program.

12. What are the different forms of inheritance supported by C++? Explain with relevant example code.

13. Explain protected data with private and public inheritance.

14. Write a C++ program for to solve eight queens problem with friend functions.

15. Explain about static member and this pointer with suitable code.

16. For a supermarket, define a bill class. All the bill objects will contain bill number, name of clerk preparing the bill, each item with quantity and price and total amount to be paid. Total items in the bill are
varying. Define dynamic memory allocation constructor for bill class such that any number of items from 1 to 50 can be accommodated in a single bill. There is an array describing each item with a price. The price is to be picked up from that array. Now overload = operator and provide reason for the need of such operator.

16. Write a program to define a class person with more than three constructors. Define data and function members in the class in such a way that all three constructors are meaningful.

17. What are abstract classes? Give an example (with the program) to illustrate the use of abstract classes.
UNIT III - C++ PROGRAMMING ADVANCED FEATURES

PART-A

1. What is Abstract Class?

   A class that contains at least one pure virtual function is called abstract class.

   If there's only one pure virtual function, the class becomes abstract. An abstract class is a class which does not fully represent an object.

   Example:  
   ```
   Virtual void print () =0;
   ```

2. What is Pure Abstract Class?

   An abstract class is one in which there is a declaration but no definition for a member function. A pure Abstract class has only abstract member functions and no data or concrete member functions.

   Example:
   ```
   class PureAbstractClass
   {
   public:
   virtual void AbstractMemberFunction() = 0;
   }
   ```

3. What is an exception?

   Exception refers to unexpected condition in a program. The unusual conditions could be faults, causing an error which in turn causes the program to fail. The error handling mechanism of C++ is generally referred to as exception handling.

4. Give the syntax of Syntax of Exception handling

   ```
   try //try block
   {
   ........
   throw exception; // Block of statements which
   ........ // detects and throws an exception
   }
   catch(type arg) // Catches exception
   {
   ........
   ........ // Block of statements that
   ........ // handles the exception
   }
   ```

5. What are the Exception handling mechanisms?

   To detect and report error
   The error handling code performs the following task
   1. Find the problem (Hit the exception)
   2. Inform that an error has occurred.(Throw the exception)
   3. Receive the error information.(Catch the exception)
4. Take corrective actions (Handle the exception)

6. Write the keywords in Exception handling.

1. try
2. throw
3. catch

TRY BLOCK - The keyword try is used to preface a block of statements (surrounded by braces) which may generate exceptions.

THROW - When an exception is detected, it is thrown using a throw statement in the try block.

• CATCH BLOCK - defined by the keyword catch ‘catches’ the exception ‘thrown’ by the throw statement in the try block, and handles it appropriately.

7. List the types of Exception handling.

Synchronous Exception:
– Out of rage
– Over flow

The exception which occur during program execution due to some fault in the input data or technique that is not suitable to handle the current class of data, within the program are known as synchronous exception.

For instance errors such as out-of-range, overflow, underflow and so on.

Asynchronous Exception
- Error that are caused beyond the control of the program
– Keyboard interrupts

The exceptions caused by events or faults unrelated to the program and beyond the control of the program are called asynchronous exception.

For instance, errors such as keyboard interrupts, hardware malfunctions, disk failure, and so on.

In C++ only synchronous exception can be handled.

8. How the exception can be handled?

i) Find the problem (hit the exception)

ii) Inform that an error has occurred (throw the exception)

iii) Receive the error information (catch the exception)

iv) Take corrective actions (handle the exception)

9. What are the factors and need for Exception Handling?

– Division by zero
– Access to an array outside of its bounds
– Running out of memory
– Running out of disk space

Need for Exception Handling

• Dividing the error handling
• Unconditional termination & programmer preferred termination
10. List some examples for exceptions

i) Divide by zero error.

ii) Accessing an array outside its memory bounds.

iii) Running out-of memory or disk space.

iv) Running out of disk space

v) Abnormal program termination

11. List out the steps of exception handling?

- Find the problem (hit the exception)
- Inform the error that has occurred (throw the exception)
- Receive the error details (catch the exception)

12. Define Unexpected () and Terminate () Functions?

Two special library functions are implemented in C++ to process exceptions not properly handled by catch blocks or exceptions thrown outside of a valid try block. These functions are unexpected () and terminate ()

13. What is mean by Catching mechanism?

The catch statement catches an exception whose type matches with the type of catch argument. When it is caught, the code in the catch block is executed.

14. What are the blocks related to exception handling constructs?

The blocks related to exception handling constructs are

- try
- throw
- catch

The keyword try is used to preface a block of statements. This block of statements is known as try block.

When an exception is detected, it is thrown using throw statement in the try block.

A catch block catches the exception thrown by the throw statement in the try block and handles it appropriately.

Syntax:

try

{........

........

throw exception;

}
catch (type arg)
{
}

15. **List the different throwing mechanism.**
   1. throw (exception);
   2. throw exception;
   3. throw;

16. **Define Multiple Catch Statements**
   - A try can have multiple catches
   - if there are multiple catches for a try, only one of the matching catch is selected and that corresponding catch block is executed.

   **syntax:**
   ```
   try
   {
   any statements
   if (some condition) throw value1;
   else if (some other condition) throw value2;
   else if (some last condition) throw valueN;
   }
   catch (type1 name)
   {
   any statements
   }
   catch (type2 name)
   {
   any statements
   }
   catch (typeN name)
   {
   any statements
   }
   ```

17. **What is the use of Terminate function?**
   - Terminate () is the function which calls abort() to exit the program in the event of run time error related to exceptions
   - the user can provide his or her own terminate function instead of built-in terminate
   **Use:**
   Used to close all open files & deallocate resources before quitting the program.
   **Syntax:** set_terminate (myterminate);

18. **What is meant by Uncaught function?**
   - This function returns true if an exception has been thrown but not yet caught.
   - Once caught, the function returns false.
   **Syntax:** bool uncaught_exceptions.
   If (uncaught_exception())
   {
   //Do not call the function which might throw an exception
   }
Otherwise
{
Follow the natural sequence of the destructor Algorithm
}

19. Describe the rules for function template?
   Rules:
   - The keyword template should be placed in front of function name.
   - The function template call is same as ordinary function call.
   - The user can use any number for template class data type normally we can use T.
   - The argument list should contain at least one argument from each template class data type.

20. What is re-throwing an exception?

   An exception can be re-thrown using a statement called throw. The re-thrown exception will be caught only by the next try-catch block sequence and not by the same try-catch sequence

21. What are the functions supported by C++ to handle uncaught exceptions?

   The functions supported by C++ to handle uncaught exceptions are
   
   terminate ( )
   set_terminate ( )
   unexpected ( )
   set_unexpected ( )

22. Write the syntax to test throwing restrictions.

   Some functions may be restricted in throwing some types of exceptions. That can be done simply by adding a throw list clause to the function definition.

   Syntax:

   Type function_name(arg-list) throw(type-list) { … }

23. What is generic programming?

   Generic programming is an approach where generic types are used as parameters in algorithms so that they work for a variety of suitable data types and data structures.

24. What is template class or instantiation?

   A specific class created from class template is known as template class. The process of creating template class is known as instantiation
25. List the application of templates.

It is a new concept which enables to define generic classes and functions and thus provides support for generic programming.

26. What is a template?

Templates support generic programming, which allows developing reusable software components such as functions, classes etc., and supporting different data types in a single framework.

27. What is function template?

The templates declared for functions are called function templates. They perform appropriate operations depending on the data type of the parameters passed to them.

Function templates are generic function, which work for any data that is passed to them.

The data type is not specified while declaring the function . It performs appropriate operation depending on the data type we passed to them.

Template is a method for writing a single function or class for a family of similar function or classes in generic manner. When a single function is writing for a family of similar function it called as “function template”. In this function at least one formal argument is generic.

Syntax:
```
Template<typename T ,….>
{ 
Function body
}
```

28. What are the rules for declaring template?

- The keyword template should be placed in front of function name
- The function template call is same as ordinary function call
- The user can use any number for template class data type normally we can use T

The argument list should contain at least one argument from each template class data type.

29. How will you overload Function template?

A template function may be overloaded either by defining template function or ordinary functions of its name. That is to overload template function we need at least one template function.

30. List the types of error.

i) Logical error ii) Syntax error

31. Define class template with multiple parameters.

A class template use more than one generic data type is class template with multiple parameters. template<class T1,classT2,…>

Class classname {

// class member specification with anonymous type T
}

32. Define function template with multiple parameters.
A function template use more than one generic data type is function template with multiple parameters.

```cpp
template <class T1, class T2, ...>
return 

type function
-name(arg of type T)

{ // body of function with anonymous type T }
```

33. What is the syntax used to declare Function Template with multiple parameters?

Syntax:
```cpp
Template<class T1, class T2...>
Return type fun _ name(arguments T1,T2,T3,....)
{
 Body of the function;
}
```

34. What are the template Compilation models?

Standard C++ defines two models for compiling template code. In each of these models, class declarations and function declarations are placed in header files and functions and member from the source files are made available to the compiler. The models the definitions from the source files are made available to the compiler.

35. List the types of Compilation models.

The models are
- Inclusion compilation model
- Separate compilation model

Inclusion compilation model
It is possible to compile the template spread across multiple files by having the copy of the template text in all the files. This model is known as the inclusion compilation model.

Separate compilation model
It is possible to compile template separately and then use them by providing their prototype in other files. This model is known as the separate compilation model.

36. What is a stream?

Stream is a series of bytes, which act either as a source from which input data can be extracted or as a destination to which the output can be sent. The source stream provides data to the program called the input stream and the destination stream that receives data from the program is called the output stream.

37. How do you classify ios class?

Istream – input stream does formatted input.
Ostream – output stream does formatted output.
Iostream – input / output stream does formatted input and output.

38. What are console stream classes in C++?

- Cin     - standard input (keyword)
- Cout    - standard output (screen)
39. **What are the file stream classes in C++?**
   - filebuf
   - fstreambase
   - ifstream
   - ofstream
   - fstream

40. **Define Ios stream.**
   - It is the base class for istream and ostream.
   - It is declared as the virtual base class.
   - It provides the basic support for formatted and unformatted I/O operations.

41. **Define Istream**
   - Provides facilities for formatted and unformatted input operations.
   - Functions such as getc(), getline(), read() are declared.
   - Overloaded extraction operator (>>)

42. **Define Ostream:**
   - Provides the facilities for formatted and unformatted output operations.
   - Putc() and write() functions are declared.
   - Overloaded insertion operator (<<)

43. **Define Iostream:**
   - Provides facilities for handling both input and output streams.
   - Inherits the properties of ios, istream and ostream through multiple inheritance

44. **What are the file manipulation functions in C++?**
   - seekg( )
   - seekp( )
   - tellg( )
   - tellp( )

45. **What are unformatted output operations in C++?**
   - The input operations are used to carry out the following member functions. They are
     - overload operator(<<)
     - put()
     - write()

46. **What are the different types of formatted console I/O operations?**
   - Manipulators
   - ios class function and flags
   - User defined output functions
   - The ios class contains a large number of member functions that would help us to format the output in a number of ways.

46. **What are manipulators?**
   - Manipulators are special functions that are specifically designed to modify the working of a stream. They can be embedded in the I/O statements to modify the form parameters of a stream.
Example:
```cpp
cout<<manip1<<manip2<<item;
cout<<manip1<<item1<<manip2<<item2
```

47. Define non-type template arguments
In addition to the type argument T in template, we can also use other arguments such as strings, function names, constant expressions and built-in-types. This is known as non-type template arguments.
```cpp
template<class T, int variable>
class classname
{ .... };
```

48. What is a Parameterized type?
This concept is most obviously useful for working with collections in a strongly typed language. This way, you can define behavior for sets in general by defining a template class Set.
```cpp
class Set <T> {
    void insert (T newElement);
    void remove (T anElement);
}
```

49. List out the parameterized manipulators?
- setw()
- setprecision()
- setfill()
- setbase()

50. What are non-parameterized manipulators?
- flush
- end1

51. What is class template?
Class templates are used to create generic class which support different data types.
```cpp
Syntax:
template <class T>
class <class_name>
{
    Member;
    Member function;
};
```

52. What are the methods that can be defined as friend to the class template?
A friend of a class template can be one of the following:
- Class template
- Function template
- Non-template function
- Non-template class
- A specialization of class template
- A specialization of function template
53. List the manipulators.

The manipulators are:

- `setw()`
- `setprecision()`
- `setfill()`
- `setiosflags()`
- `resetiosflags()`

54. Mention the equivalent ios function for manipulators.

<table>
<thead>
<tr>
<th>Manipulator</th>
<th>Equivalent ios function</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>setw(int w)</code></td>
<td><code>width()</code></td>
</tr>
<tr>
<td><code>setprecision(int d)</code></td>
<td><code>precision()</code></td>
</tr>
<tr>
<td><code>setfill(int c)</code></td>
<td><code>fill()</code></td>
</tr>
<tr>
<td><code>setiosflags(long f)</code></td>
<td><code>setf()</code></td>
</tr>
<tr>
<td><code>resetiosflags(long f)</code></td>
<td><code>unsetf()</code></td>
</tr>
</tbody>
</table>

55. Define fill functions.

The `fill()` function can be used to fill the unused positions of the field by any desired character rather than by white spaces (by default). It is used in the following form:

```cpp
cout.fill(ch);
```

where `ch` represents the character which is used for filling the unused positions.

For example, the statements

```cpp
cout.fill("*");
cout.width(10);
cout<<5250 <<"\n";
```

56. List the ios format function.

The ios format functions are as follows:

- `width()`
• precision()
• fill()
• setf()
• unsetf()

57. Define Standard Template Library (STL)?
A library of container templates approved by the ANSI committee for inclusion in the standard C++ specification. A programmer who then launches into a discussion of the generic programming model, iterators, allocators, algorithms, and such, has a higher than average understanding of the new technology that STL brings to C++ programming.

58. What is container?
A container is an object that actually stores data. It is a way data is organized in memory. The STL containers are implemented by template classes and therefore can be easily customized to hold different types of data.

59. What is a container class? What are the types of container classes?
A container class is a class that is used to hold objects in memory or external storage. A container class acts as a generic holder. A container class has a predefined behavior and a well-known interface. A container class is a supporting class whose purpose is to hide the topology used for maintaining the list of objects in memory. When a container class contains a group of mixed objects, the container is called a heterogeneous container; when the container is holding a group of objects that are all the same, the container is called a homogeneous container.

60. STL containers-What are the types of STL containers?
There are three types of STL containers;
• Adaptive containers like queue, stack
• Associative containers like set, map
• Sequence containers like vector, deque

61. What is an iterator?
An iterator is an object that points to an element in a container. We can use iterators to move through the contents of containers. Iterators are handled just like pointers.

62. What are the three types of containers?
The STL contains three types of containers
1) Sequence containers
2) Associative containers
3) Derived containers

63. Define file?
A file is a collection to inter related data stored in a particular area on the disk.

64. What is the syntax used for closing a file give example?
file object.close();
where
   file object  - opened file name
   close        - member function
Example:
   Employee. Close (); // this closes the file std.dat with the file object employee

65. What are the file stream classes in C++?
   - filebuf
   - fstreambase
   - ifstream
   - ofstream
   - fstream
   - )

67. What are the file open modes?
   - ios::app, ios::binary, ios::oput, ios::ate, ios::nocreate, ios::nreplace, ios::trunk

PART-B

1. Define a DivideBy Zero definition and use it to throw exceptions on attempts to divide by zero.
2. What is the need of Templates? Explain.
3. Explain about function templates?
4. Implement selection sort as a generic function.
5. Implement quick sort as a generic function.
6. Write a program for generic queue class with two member functions, insert and delete. Use the array to implement the queue.
7. Define a stack. The class should throw an exception when the stack underflow and overflow takes place.
8. Using the Time class, throw an exception when invalid time is input, write set_terminate to provide your own terminate function, which care of this problem.
9. What is uncaught exception function? Give an example.
10. What are the use of terminate () and Unexpected functions? Explain with a program.
11. How to use multiple catch functions inside a program? Explain with a program.
12. Write all blocks of exception handling? Explain with a program
13. Discuss about Streams and stream classes
14. Write notes on Formatted and Unformatted Console I/O Operations.
15. Explain about File Pointers and Manipulations with example.
16. Discuss about manipulators and file streams with Program.
17. Write on Details about File modes and File I/O
18. Explain about File Pointers and their manipulations with example.
19. Give the differences between Manipulators and ios Functions.
20. Write a template function to sort the elements of an array.
21. What is an Exception? Explain how the control is transferred and handled in c++ programs.
22. Write a program to write the text in a file. Read the text from the file, from end of the file.
    Display the contents of file in reverse order. Append the contents to the existing file.
23. Discuss about the different components in STL.
24. Write a class template to insert an element into a linked list.
25. Write a class template to implement a stack
26. What is a user defined exception? Explain with an example
27. Write a C++ program to demonstrate function template to print array of different types
28. Explain with an example. How exception handling is carried out in C++.
29. Explain multiple catch statement with help of suitable C++ coding.
30. Describe the various file modes and its syntax.
31. Describe class template with suitable example
32. Explain the overloading of template function with suitable example
33. Explain exception handling in C++ with suitable example
34. Highlight the features of STL.
35. List the different stream classes supported in C++
36. Explain the use of any six manipulators with example.
37. Discuss in detail the unformatted I/O operations
UNIT IV - ADVANCED NON-LINEAR DATA STRUCTURES

PART – A

1. Define Tree.

A Tree is a collection of one or more nodes with a distinct node called the root, while remaining nodes are partitioned as \(T_1, T_2, \ldots, T_k\), \(K \geq 0\) each of which are sub trees, the edges of \(T_1, T_2, \ldots, T_k\) are connected the root.

![Tree Diagram]

Trees are non-linear data structure, which is used to store data items in a shuffled sequence. It represents any hierarchical relationship between any data item.

It is a collection of nodes, which has a distinguish node called the root and zero or more non-empty sub trees \(T_1, T_2, \ldots, T_k\) each of which are connected by a directed edge from the root.

2. Give some applications of Trees

- Implementing the file system of several operating systems.
- Evaluation of arithmetic expression.
- Set representation.
- Gaming/Decision making problems.

3. List the applications of trees.

   a. Binary search trees
   b. Expression trees
   c. Threaded binary trees
   d. Syntax analysis
   e. Symbol table construction

4. Define node, degree, siblings, depth/height, level.

   **Node:** A node is an item of information with branches to other items.
   **Degree:** The number of subtrees of a node is called is degree.
   **Siblings:** The children of the same parent is said to be siblings.
   **Level:** The level of a node is defined recursively by assuming the level of the root to be one and if a node is at level \(l\), then its children at level \(l+1\).
   **Depth/Height:** The depth/height of a tree is defined to be the level of a node which is maximum.

5. Define Terminal node or leaf?

   Nodes with no children are known as leaves. A leaf will always have degree zero and is also called as terminal node.
6. Define Non-terminal node?
Any node except the root node whose degree is a non-zero value is called as a non-terminal node. Non-terminal nodes are the intermediate nodes in traversing the given tree from its root node to the terminal node.

7. Define balanced search tree.
Balanced search tree have the structure of binary tree and obey binary search tree properties with that it always maintains the height as O(log n) by means of a special kind of rotations. Eg. AVL, Splay, B-tree.

8. Define AVL tree.
An empty tree is height balanced. If T is a non-empty binary tree with T_L and T_R as Its left and right subtrees, then T is height balanced if

1. T_L and T_R are height balanced.
2. \(|h_L - h_R| \leq 1.\)
Where h_L and h_R are the height of T_L and T_R respectively

Or

AVL tree also called as height balanced tree .It is a height balanced tree in which every node will have a balancing factor of -1,0,1.

Balancing factor of a node is given by the difference between the height of the left sub tree and the height of the right sub tree.

The balance factor of a node in binary tree is defined to be h_R - h_L where h_L and h_R are heights of left and right subtrees of T. For any node in AVL tree the balance factor should be 1,0 or -1.

10. Mention the four cases to rebalance the AVL tree.
- An insertion of new node into Left subtree of Left child(LL).
- An insertion of new node into Right subtree of Left child(LR).
- An insertion of new node into Left subtree of Right child(RL).
- An insertion of new node into Right subtree of Right child(RR)

11. What are the various transformation performed in AVL tree?
Single rotation
- Single L rotation
- Single R rotation
Double rotation
12. **What are the drawbacks of AVL trees?**

The drawbacks of AVL trees are:
- Frequent rotations
- The need to maintain balances for the tree’s nodes
- Overall complexity, especially of the deletion operation.

13. **Define Binary Tree?**

A Binary tree is a finite set of data items which is either empty or consists of a single item called root and two disjoint binary trees called left sub tree max degree of any node is two.

14. **Define Splay Tree.**

A splay tree is a self-balancing binary search tree with the additional property that recently accessed elements are quick to access again. It performs basic operations such as insertion, look-up and removal in $O(\log(n))$ amortized time. For many non-uniform sequences of operations, splay trees perform better than other search trees, even when the specific pattern of the sequence is unknown.

15. **List the types of rotations available in Splay Tree.**

Let us assume that the splay is performed at vertex $v$, whose parent and grandparent are $p$ and $g$ respectively. Then, the three rotations are named as:

**Zig:** If $p$ is the root and $v$ is the left child of $p$, then left-left rotation at $p$ would suffice. This case always terminates the splay as $v$ reaches the root after this rotation.

**Zig-Zig:** If $p$ is not the root, $p$ is the left child and $v$ is also a left child, then a left-left rotation at $g$ followed by a left-left rotation at $p$, brings $v$ as an ancestor of $g$ as well as $p$.

**Zig-Zag:** If $p$ is not the root, $p$ is the left child and $v$ is a right child, perform a left-right rotation at $g$ and bring $v$ as an ancestor of $p$ as well as $g$.

16. **List the characteristics of Splay Tree.**

- Does not require any accounting information (color, level, height, etc.)
- $O(N)$ worst case for any single operation, but $O(\log N)$ average case
- Frequently-accessed keys are moved up so that they are near the root

17. **What are the various operations performed in the binary search tree?**

i. insertion
18. Define B-tree?

A B-tree of order $m$ in an $m$-way search tree that is either empty or is of height $\geq 1$ and

1. The root node has at least 2 children
2. All nodes other than the root node and failure nodes have at least $m/2$ children.

All failure nodes are at same level.

Or

A B-tree is a specialized multiway tree designed especially for use on disk. In a B-tree each node may contain a large number of keys. The number of subtrees of each node, then, may also be large.

A B-tree is designed to branch out in this large number of directions and to contain a lot of keys in each node so that the height of the tree is relatively small.

19. What are the applications of B-tree?

- Database implementation
- Indexing on non primary key fields

20. Explain the properties of Red-black trees.

1. Every node is either red or black.
2. The root and leaves (NIL ’s) are black.
3. If a node is red, then its parent is black.
4. All simple paths from any node $x$ to a descendant leaf have the same number of black nodes $= \text{black-height}(x)$.

21. What are red black trees?

A **red–black tree** is a binary search tree in which every node is colored with either red or black. It is a type of self-balancing binary search tree.

22. What are the steps to delete a node in Red Black Tree?

- Find node to delete
- Delete node as in a regular BST
- Node to be deleted will have at most one child
- If we delete a Red node tree still is a Red-Black tree
- Assume we delete a black node
- Let $x$ be the child of deleted node
- If $x$ is red, color it black and stop
- If $x$ is black mark it double black and apply the following:

![Red and Black nodes]

23. What is Fibonacci heap?

It is defined as a

- Set of heap-ordered trees.
- Maintain pointer to minimum element.
- Set of marked nodes.
24. What is a heap?

A heap is a partially ordered data structure, and can be defined as a binary tree assigned to its nodes, one key per node, provided the following two conditions are met.

The tree’s shape requirement-The binary tree is essentially complete, that is all the leaves are full except possibly the last level, where only some rightmost leaves will be missing.

The parental dominance requirement-The key at each node is greater that or equal to the keys of its children.

25. What are binomial heaps?

A binomial heap is implemented as a collection of binomial trees (compare with a binary heap, which has a shape of a single binary tree). A binomial tree is defined recursively:

- A binomial tree of order 0 is a single node (or)

The main application of Binary Heap is as implement priority queue. Binomial Heap is an extension of Binary Heap that provides faster union or merge operation together with other operations provided by Binary Heap. A Binomial Heap is a collection of Binomial Trees.

A binomial heap \( H \) is a set of binomial trees that satisfies the following binomial-heap properties.

- Each binomial tree in \( H \) is heap-ordered: the key of a node is greater than or equal to the key of its parent.
- There is at most one binomial tree in \( H \) whose root has a given degree.

(a) \( \text{head} [H] \)
26. Give the properties of Binomial Tree.

Binomial trees properties are as follows. For the binomial tree Bk,

1. There are $2^k$ nodes,
2. The height of the tree is $k$,
3. There are exactly nodes at depth $i$ for $i = 0, 1, \ldots, k$, and
4. The root has degree $k$, which is greater than that of any other node; moreover if the children of the root are numbered from left to right by $k - 1, k - 2, \ldots, 0$, childi is the root of a subtree $B_i$.

27. What are the advantages of Fibonacci heaps?

1. The use of Fibonacci heap improves the asymptotic running time of Dijkstra’s algorithm of computing the shortest path.
2. In prime’s algorithm Fibonacci heap is helpful in finding the minimum spanning tree.

28. How will you extract a minimum node in a Fibonacci heap?

- Amortized cost is $O(D(n))$
- Make a root out of each of the minimum node’s children.
- Remove the minimum node from the root list, min ptr to right(x)
- Consolidate the root list by linking roots of equal degree and key[x] <= key[y], until every root in the root list has a distinct degree value. (uses auxiliary array)

29. What are the different types of Rotation in AVL Tree? Explain rotation.

- Two types of rotation are
  1. Single rotation
  2. Double rotation.

30. Define Rotation.

Manipulation of tree pointers is centered at the pivot node to bring the tree back into height balance. The visual effect of this pointer manipulation so to rotate the sub tree whose root is the pivot node. This operation is referred as AVL rotation.

31. What are the main properties of a binary heap?

1. Structure property
2. Heap order property

32. Define splay tree.

A splay tree is a binary search tree in which restructuring is done using a scheme called splay. A splay is heuristic method which moves a given vertex v to the roof of the splay tree using a sequence of rotations.
33. Define Red-Black Tree.

A red–black tree is a data structure which is a type of self-balancing binary search tree.

Balance is preserved by painting each node of the tree with one of two colors (typically called 'red' and 'black') in a way that satisfies certain properties, which collectively constrain how unbalanced the tree can become in the worst case.

When the tree is modified, the new tree is subsequently rearranged and repainted to restore the coloring properties. The properties are designed in such a way that this rearranging and recoloring can be performed efficiently.

OR

Red-Black Tree is a self-balancing Binary Search Tree (BST) where every node follows following rules.

1) Every node has a color either red or black.

2) Root of tree is always black.

3) There are no two adjacent red nodes (A red node cannot have a red parent or red child).

4) Every path from root to a NULL node has same number of black nodes.

34. What are Amortized analysis and its techniques?

In an amortized analysis, the time of a sequence of operations is averaged Amortized Analysis of operations is averaged.

Amortized analysis: Determine worst-case running time of a sequence of data structure operations as a function of the input size.

– Average does not mean averaging over a distribution of inputs.

– No probability is involved.

– We do mean average cost in the worst case

Three methods are

1. Aggregate analysis

2. Accounting method
3. Potential method

35. What is Disjoint Set?
A disjoint sets data structure represents a collection of sets that are disjoint: that is, no item is found in more than one set. A disjoint-set data structure, also called a union–find data structure or merge–find set, that keeps track of a set of elements partitioned into a number of disjoint (non overlapping) subsets.

A disjoint-set data structure is a data structure that keeps track of a set of elements partitioned into a number of disjoint (non-overlapping) subsets. It supports two operations:

1. Find: Determine which subset a particular element is in. This can be used for determining if two elements are in the same subset.
2. Union: Join two subsets into a single subset.

36. Applications of Amortized analysis

- Splay trees.
- Dynamic table.
- Fibonacci heaps.
- Garbage collection.
- Move-to-front list updating.
- Push-relabel algorithm for max flow.
- Path compression for disjoint-set union.
- Structural modifications to red-black trees.
- Security, databases, distributed computing.

37. Define Accounting Method.

The accounting method determines the amortized running time with a system of credits and debits.

Assign differing charges to different operations, with some operations charged more or less than they actually cost. The amount we charge an operation is called its amortized cost.

When an operation's amortized cost exceeds its actual cost, the difference is assigned to specific objects in the data structure as credit.

Credit can be used later on to help pay for operations whose amortized cost is less than their actual cost.

Assign different charges to each operation.

- \( D_i = \) data structure after \( i \)th operation.
• $c_i$ = actual cost of ith operation.
• $\hat{c}_i$ = amortized cost of ith operation = amount we charge operation i.
• When $\hat{c}_i > c_i$, we store credits in data structure $D_i$ to pay for future ops; when $\hat{c}_i < c_i$, we consume credits in data structure $D_i$.
• Initial data structure $D_0$ starts with zero credits.

38. **What is Potential method?**

The potential method of amortized analysis represents the prepaid work as "potential energy," or just "potential," that can be released to pay for future operations. The potential is associated with the data structure as a whole rather than with specific objects within the data structure.

A potential function $\Phi$ maps each data structure $D_i$ to a real number $\Phi(D_i)$, which is the potential associated with data structure $D_i$.

The amortized cost $\hat{c}_i$ of the ith operation with respect to potential function $\Phi$ is defined by

$$\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1}).$$ \hspace{1cm} (18.1)

The amortized cost of each operation is therefore its actual cost plus the increase in potential due to the operation. By equation (18.1), the total amortized cost of the n operations is

$$\sum_{i=1}^{n} \hat{c}_i = \sum_{i=1}^{n} (c_i + \Phi(D_i) - \Phi(D_{i-1}))$$

$$= \sum_{i=1}^{n} c_i + \Phi(D_n) - \Phi(D_0).$$ \hspace{1cm} (18.2)

39. **Define Aggregate method.**

The aggregate method of amortized analysis, is that for all $n$, a sequence of $n$ operations takes worst-case time $T(n)$ in total. In the worst case, the average cost, or amortized cost, per operation is therefore $T(n)/n$.

40. **Main Application of B-tree.**

1. **File systems.**
   • Reading a page into memory from disk is expensive.
   • Accessing info on a page in memory is free.
   • Goal: minimize # page accesses.
   • Node size $M = $ page size.
2. Databases:

ORACLE, DB2, INGRES, SQL, PostgreSQL

41. Main Properties of B-Tree.

1) All leaves are at same level.
2) A B-Tree is defined by the term minimum degree ‘t’. The value of t depends upon disk block size.
3) Every node except root must contain at least t-1 keys. Root may contain minimum 1 key.
4) All nodes (including root) may contain at most 2t – 1 keys.
5) Number of children of a node is equal to the number of keys in it plus 1.
6) All keys of a node are sorted in increasing order. The child between two keys k1 and k2 contains all keys in range from k1 and k2.
7) B-Tree grows and shrinks from root which is unlike Binary Search Tree. Binary Search Trees grow downward and also shrink from downward.
8) Like other balanced Binary Search Trees, time complexity to search, insert and delete is O(Logn).

42. Define the condition for order of m for B-Tree

A B-tree of order m is a multiway search tree of order m such that:

- All leaves are on the bottom level.
- All internal nodes (except perhaps the root node) have at least ceil(m / 2) (nonempty) children.
- The root node can have as few as 2 children if it is an internal node, and can obviously have no children if the root node is a leaf (that is, the whole tree consists only of the root node).
- Each leaf node (other than the root node if it is a leaf) must contain at least ceil(m / 2) - 1 keys.

43. Why Red-Black Trees?

Most of the BST operations (e.g., search, max, min, insert, delete.. etc) take O(h) time where h is the height of the BST. The cost of these operations may become O(n) for a skewed Binary tree.

If the height of the tree remains O(Logn) after every insertion and deletion, then can guarantee an upper bound of O(Logn) for all these operations. The height of a Red Black tree is always O(Logn) where n is the number of nodes in the tree.

44. How does a Red-Black Tree ensure balance?

A chain of 3 nodes is not possible in red black tree; hence combination of colors will not violate Red-Black tree property.

A chain of 3 nodes is not possible in Red-Black Trees.

Following are NOT Red-Black Trees

| 30 | 30 | 30 |
|-----------------|
| /   /   /   |
| 20 NIL 20 NIL 20 NIL |
| /   /   /   |
| 10 NIL 10 NIL 10 NIL |

Violates Property 4. Violates Property 4 Violates Property 3

Following are different possible Red-Black Trees with above 3 keys

| 20 | 20 |
45. Comparison with AVL Tree
The AVL trees are more balanced compared to Red Black Trees, but they may cause more rotations during insertion and deletion. So if the application involves many frequent insertions and deletions, then Red Black trees should be preferred. And if the insertions and deletions are less frequent and search is more frequent operation, then AVL tree should be preferred over Red Black Tree.

46. What is a Binomial Tree?
A Binomial Tree of order 0 has 1 node. A Binomial Tree of order k can be constructed by taking two binomial trees of order k-1, and making one as leftmost child of other.
A Binomial Tree of order k has following properties.
a) It has exactly \(2^k\) nodes.
b) It has depth as k.
c) There are exactly kCi nodes at depth i for i = 0, 1, . . . , k.
d) The root has degree k and children of root are themselves Binomial Trees with order k-1, k-2,.. 0 from left to right.

47. How to represent Binomial Heap?
A Binomial Heap is a set of Binomial Trees. A Binomial Tree must be represented in a way that allows sequential access to all siblings, starting from the leftmost sibling (need this in and extractMin() and delete()). The idea is to represent Binomial Trees as leftmost child and right-sibling representation, i.e., every node stores two pointers, one to the leftmost child and other to the right sibling.

Space Complexity: \(O(N)\) for N elements
Time Complexity: \(O(N + M \ lg^*N)\) for N elements and M queries, where \(lg^*\) is the inverse of Ackermann’s function. For all practical applications, \(lg^*N \leq 5\)

49. Difference between asymptotic analysis and amortized analysis.
The critical difference between asymptotic and amortized analysis is that the former is dependent on the input itself, while the latter is dependent on the sequence of operations the algorithm will execute.

Therefore:
- **asymptotic analysis** allows us to assert that the complexity of the algorithm when it is given a best/worst/average case input of size approaching N is bounded by some function \(F(N)\) -- where N is a variable
- **amortized analysis** allows us to assert that the complexity of the algorithm when it is given an input of unknown characteristics but known size N is no worse than the value of a function \(F(N)\) -- where N is a known value
PART-B

1. Explain AVL tree with its rotation?
2. Explain Red-Black Trees with example
3. Explain in detail about Splay Tree with example
4. Explain in detail about Amortized Analysis and its methods
5. Explain disjoint set in detail
6. Explain in detail about the Fibonacci heaps
7. Give an example for binomial heap and explain the same
8. Discuss, compare and contrast Binomial heaps and Fibonacci heaps in terms of insertion, deletion operations and applications
9. Describe any one scheme for implementing Red-Black trees. Explain insertion and deletion algorithm with details. How do these algorithms balance the height of the tree?
10. Write algorithm to construct Fibonacci heap with suitable example
11. Write algorithm to construct Binomial heap with suitable example
12. What are the properties of Red black trees?
13. What is the need for splay trees? Give an example
14. Illustrate with an algorithm to show the insertion of data into an AVL tree
15. Write an algorithm to merge two AVL trees T1 and T2 to obtain new AVL tree pointed out by NT
16. Write C++ member function to implement the following Fibonacci Heap operations
   (i) Create an empty F-heap. (8)
   (ii) Insert element x into F-heap
17. Explain Fibonacci Heap Deletion Key operation using Cascading-Cut procedure with example
18. Explain insertion procedure in Red-Black tree and insert the following sequence: 
   \{20,10,5,30,40,57,3,2,4,35,25,18,22,21\}
19. Show the result of inserting 10,17,2,4,9,6,8 into an empty AVL tree
20. Write the procedure to implement single and double rotation while inserting nodes in AVL tree
UNIT V - GRAPHS

PART –A

1. What is a graph?
A graph consists of a set of vertices V and set of edges E which is mathematically
represented as G=(V,E). Each edge in a pair (V, W) where V, W, belongs to E, edges are sometimes
referred to as arcs.

In the above graph V₁, V₂, V₃, V₄ are the vertices and (V₁, V₂), (V₂, V₃), (V₃, V₄), (V₄, V₁), (V₁, V₃),
(V₂, V₄) are the edges

2. What is the running time for topological sort?
The running time of the algorithm for topological sort is O(V²). For the algorithm using
Queue, the running time is O(E + V) if adjacency list are used.

3. What are Directed graphs?
If a pair of vertices for any edge is ordered, then that graph is called as Digraph or directed
digraph.

Directed graph is a graph, which consists of directed edges, where each edge in E is
unidirectional. In directed graph, the edges are directed or one way. It is also called as digraphs. If
(v, w) is a directed edge, then (v, w) ≠ (w, v).

(V₁, V₂) ≠ (V₂, V₁)

4. Define undirected graph
**Undirected Graph** An undirected graph is a graph, which consists of undirected edges. In
undirected graph, the edges are undirected or two way. If (v, w) is a undirected edge, then (v, w) =
(w, v).

(V₁, V₂) = (V₂, V₁)

5. What is minimum spanning tree?
A minimum spanning tree of a weighted connected graph G is its spanning tree of the smallest
weight, where the weight of a tree is defined as the sum of the weights on all its edges.

Depth first works by selecting one vertex \( V \) of \( G \) as a start vertex; \( V \) is marked visited. Then each unvisited vertex adjacent to \( V \) is searched in turn using depth first search recursively.

This process continues until a dead end (i.e.) a vertex with no adjacent unvisited vertices is encountered. At a dead end, the algorithm backs up one edge to the vertex it came from and tries to continue visiting unvisited vertices from there. The algorithm eventually halts after backing up to the starting vertex, with the latter being a dead end.

7. State the greedy algorithm.

A greedy algorithm is an algorithm that follows the problem solving heuristic of making the locally optimal choice at each stage with the hope of finding a global optimum.

In many problems, a greedy strategy does not in general produce an optimal solution, but nonetheless a greedy heuristic may yield locally optimal solutions that approximate a global optimal solution in a reasonable time.

8. List any two applications that use greedy algorithm.

Applications of the Greedy Strategy:

- **Optimal solutions:**
  - Change making
  - Minimum Spanning Tree (MST)
  - Single-source shortest paths
  - Huffman codes

- **Approximations:**
  - Traveling Salesman Problem (TSP)
  - Fractional Knapsack problem


A path in a graph is a sequence of vertices \( w_1, w_2, w_3, w_N \) such that \( w_i, w_{i+1} \) belongs to \( E \) for a value \( 1 \leq i \leq N \). The length of such a path is the number of edges on the path, which is equal to \( n-1 \).

10. Define Cycle.

A cycle is a path in which the first and last vertices are the same.

11. Define Acyclic graph.

A graph with no cycles is called Acyclic graph. A directed graph with no edges is called as a directed. Acyclic graph (or) DAG. DAGS are used for Compiler Optimization process.

12. Define connected graph.

A graph is said to be a weighted graph is connected if there is a path from every vertex to every other vertex. A directed graph with this property is called as strongly connected graph. If a directed graph is not strongly connected but the underscore graph. Without direction is connected it is called as a weakly connected graph.

13. What are the conditions for a graph to become a tree?

A graph is a tree if it has two properties. i. If it is a connected graph.

ii. There should not be any cycles in the graph.
14. Define a Weighted Graph.
   If every edge in the graph is assigned some weight or value. The weight of the edge is a positive value that represents the cost of moving the edge or the distance between two vertices. It can be directed or undirected.

![Directed Graph](image1)

![Undirected Graph](image2)

15. Define a Weighted Graph.
   If every edge in the graph is assigned some weight or value. The weight of the edge is a positive value that represents the cost of moving the edge or the distance between two vertices.

   A symmetric digraph is a directed graph such that for every edge uv there is also a reverse edge vu.

17. What is Symmetric undirected graph?
   Every undirected graph is a symmetric digraph where each undirected edge is considered as a pair of directed edges in opposite direction.

18. What is strongly connected Graph?
   If there is a path from every vertex to every other vertex in a directed graph then it is said to be strongly connected graph. Otherwise, it is said to be weakly connected graph.

19. Define Path
   A path in a graph is defined as a sequence of vertices \( w_1, w_2, w_3, \ldots, w_n \) such that \( (w_1, w_2, w_3, \ldots) \in E \). Where E is the number of edges in a graph.
   Path from A to D is \{A, B, C, D\} or \{A, C, D\}
   Path from A to C is \{A, B, C\} or \{A, C\}

20. Define Directed Acyclic Graph (DAG)
   A directed graph is acyclic if it has no cycles, and such types of graph is called as Directed Acyclic Graph.

21. Define degree

   **Degree**
   The number of edges incident on a vertex determines its degree. The degree of the vertex V is written as degree \( (V) \).

   **Indegree**: The indegree of the vertex V, is the number of edges entering into the vertex V.
   **Outdegree**: The outdegree of the vertex V, is the number of edges exiting from the vertex V.
22. Give the types of representation of graphs.
   1. Adjacency matrix
   2. Adjacency linked list

23. **Adjacency Matrix Representation**
   i. Adjacency matrix for directed graph
   ii. Adjacency matrix for undirected graph
   iii. Adjacency matrix for weighted graph

24. **What is a minimum spanning tree?**
A minimum spanning tree of an undirected graph G is a tree formed from graph edges that connect all the vertices of G at lowest total cost.

**Spanning tree** of an undirected graph, G is a tree formed from graph edges that connects all vertices of G.

An **Minimum Spanning tree** of an undirected graph, G is a tree formed from graph edges that connects all vertices of G at lowest cost.

A minimum spanning tree exists if and only if G is connected. The number of edges in the minimum spanning tree is |V| - 1.

25. **Explain about Adjacency Matrix**
    Adjacency matrix consists of a n*n matrix where n is the no. of vertices present. In the graph, this consists of values either 0 or 1.

26. **What is a back edge?**
The possibility of reaching an already marked vertex is indicated by a dashed line, in a graph is called as back edge.

27. **What is a single source shortest path problem?**
Given as an input, a weighted graph, G=<V,E> and a distinguished vertex „S as the source vertex. Single source shortest path problem finds the shortest weighted path from s to every other vertex in G.

28. **Explain about Unweighted shortest path.**
   Single source shortest path finds the shortest path from the source to each and every vertex present in an unweighted graph. Here no cost is associated with the edges connecting the vertices. Always unit cost is associated with each edge.

29. **Explain about Weighted shortest path**
   Single source shortest path finds the shortest path from the source to each and every vertex present in a weighted graph. In a weighted graph some cost is always associated with the edges connecting the vertices.
vertices.

30. What are the methods to solve minimum spanning tree?
   a) Prim’s algorithm
   b) Kruskal’s algorithm

31. List the two important key points of depth first search.
   i) If path exists from one node to another node, walk across the edge – exploring the edge.
   ii) If path does not exist from one specific node to any other node, return to the previous node where we have been before – backtracking.

32. What do you mean by breadth first search (BFS)?
   BFS performs simultaneous explorations starting from a common point and spreading out independently.

33. Differentiate BFS and DFS.

<table>
<thead>
<tr>
<th>No.</th>
<th>DFS</th>
<th>BFS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Backtracking is possible from a dead end</td>
<td>Backtracking is not possible</td>
</tr>
<tr>
<td>2</td>
<td>Vertices from which exploration is incomplete are processed in a LIFO order</td>
<td>The vertices to be explored are organized as a FIFO queue</td>
</tr>
<tr>
<td>3</td>
<td>Search is done in one particular direction</td>
<td>The vertices in the same level are maintained parallel</td>
</tr>
</tbody>
</table>

34. What is Topological Sort?
   A topological sort is an ordering of vertices in a directed acyclic graph, such that if there is a path from \( v_i \) to \( v_j \), then \( v_j \) appears after \( v_i \) in the linear ordering. Topological ordering is not possible if the graph has a cycle, since for two vertices \( v \) and \( w \) on the cycle, \( v \) precedes \( w \) and \( w \) precedes \( v \).

35. Explain briefly about Prim’s algorithm
   Prim’s algorithm creates the spanning tree in various stages. At each stage, a node is picked as the root and an edge is added and thus the associated vertex along with it.

36. Define a depth first spanning tree.
   The tree that is formulated by depth first search on a graph is called as depth first spanning tree. The depth first spanning tree consists of tree edges and back edges.
37. **What is a tree edge?**
   Traversal from one vertex to the next vertex in a graph is called as a tree edge.

38. **What is Kruskal’s algorithm?**
   Kruskal’s algorithm used for solving minimum spanning tree problem.

   **Procedure**
   1. Initially there are $|V|$ single node trees. Each vertex is initially in its own set.
   2. Select the edges $(u,v)$ in the order of smallest weight and accepted if it does not cause the cycle.
   3. Adding an edge merges 2 trees into one.
   4. Repeat step 2 until the tree contains all the vertices.

39. **PRIM’S ALGORITHM**
   In this method, minimum spanning tree is constructed in successive stages. One node is picked as a root and an edge is added (i.e) an associated vertex is added to the tree, until all the vertices are present in the tree with $|V| - 1$ edges.

   **Prims Algorithm**
   1. Consider any vertex in the graph.
   2. Process the vertex and add the vertex to the tree.
   3. Find the smallest edge from the graph connecting the edge of the vertex in the tree, such that it does not form a cycle.
   4. Add the vertex to the tree.
   5. Repeat step 3 until the tree contains all the vertices in the graph.

40. **SHORTEST PATH ALGORITHMS:**
   An algorithm to find the shortest distance path between the source and destination vertices is called the shortest path algorithm. **Types of shortest path problem**
   **i. Single source shortest path**
   Given an input graph $G = (V,E)$ and a distinguished vertex $S$, find the shortest path from $S$ to every other vertex in $G$. Example: Dijkstra’s algorithm (weighted graph and unweighted graph).
   **ii. All pairs shortest path problem**
   Given an input graph $G = (V,E)$. Find the shortest path from each vertex to all vertices in a graph.

41. **DIJKSTRA’S ALGORITHM:**
   **Unweighted shortest Path** In unweighted shortest path, all the edges are assigned to 1.

   **Required Information**
   **i. known** – Specifies whether the vertex is processed or not. It is set to 1 after it is processed, otherwise 0. Initially all vertices are unknown, so all entries marked as 0.

   **ii. dv** – it specifies distance form source to vertex. Initially all vertices are unreachable except for $S$ whose path length is zero.

   **iii. pv** – It is book keeping variable, which allow us to print te actual path. i.e., the vertex which makes the changes in $dv$. 
42. **BELLMAN-FORD ALGORITHM:**

*Single Source Shortest Path*

**Problem** Given a directed graph \(G(V,E)\) with weighted edges \(w(u,v)\), define the path weight of a path \(p\) as

\[
w(p) = \sum_{i=1}^{k} w(v_{i-1}, v_i)
\]

For a given source vertex \(s\), find the minimum weight paths to every vertex reachable from \(s\) denoted

\[
\delta(s,v) = \begin{cases} \min \{w(p)\} & s \rightarrow v \\ \infty & \text{otherwise} \end{cases}
\]

43. **Bellman-Ford Algorithm**

The Bellman-Ford algorithm uses relaxation to find single source shortest paths on directed graphs that may contain negative weight edges. The algorithm will also detect if there are any negative weight cycles (such that there is no solution).

**BELLMAN-FORD(G,w,s)**

1. **INITIALIZE-SINGLE-SOURCE(G,s)**
2. for \(i = 1\) to \(|G.V| - 1\)
3. for each edge \((u,v) \in G.E\)
4. RELAX\((u,v,w)\)
5. for each edge \((u,v) \in G.E\)
6. if \(v.d > u.d + w(u,v)\)
7. return FALSE
8. return TRUE

44. **FLOYD - WARSHALL ALGORITHM:**

The Floyd-Warshall algorithm works based on a property of intermediate vertices of a shortest path. An intermediate vertex for a path \(p = <v_1, v_2, ..., v_j>\) is any vertex other than \(v_1\) or \(v_j\). If the vertices of a graph \(G\) are indexed by \(\{1, 2, ..., n\}\), then consider a subset of vertices \(\{1, 2, ..., k\}\). Assume \(p\) is a minimum weight path from vertex \(i\) to vertex \(j\) whose intermediate vertices are drawn from the subset \(\{1, 2, ..., k\}\). If we consider vertex \(k\) on the path then either:

\(k\) is not an intermediate vertex of \(p\) (i.e. is not used in the minimum weight path)

\(\Rightarrow\) All intermediate vertices are in \(\{1, 2, ..., k-1\}\)

\(k\) is an intermediate vertex of \(p\) (i.e. is used in the minimum weight path)

\(\Rightarrow\) We can divide \(p\) at \(k\) giving two subpaths \(p_1\) and \(p_2\) giving \(v_i \sim k \sim v_j\) \(\Rightarrow\) Subpaths \(p_1\) and \(p_2\) are shortest paths with intermediate vertices in \(\{1, 2, ..., k-1\}\)

45. **Transitive Closure**

Floyd-Warshall can be used to determine whether or not a graph has transitive closure, i.e. whether or not there are paths between all vertices.

- Assign all edges in the graph to have weight = 1
- Run Floyd-Warshall
- Check if all \(d_{ij} < n\)
PART-B

1. Explain Graph traversals with examples
2. Describe the topological sorting method with suitable examples
3. Explain Prims algorithm with examples
4. Explain the kruskals algorithm for minimum spanning tree
5. Discuss any two shortest path algorithms
6. Compare depth first search and depth first search.
7. Explain the method of constructing minimum cost spanning tree using Kruskal’s algorithm.
8. Explain briefly articulation points and biconnected components.
9. Explain the traversals of directed graphs also give its analysis.
10. Explain Prim’s algorithm to construct minimum spanning tree from an undirected graph.
11. Explain Prim’s algorithm to construct minimum spanning tree from an undirected graph.
12. Explain Prim’s algorithm to construct minimum spanning tree from an undirected graph.
13. What is topological sort? Write an algorithm to perform topological sort.
14. Write the pseudo code to find a minimum spanning tree using Kruskal’s algorithm.
15. Find the shortest weighted path from A to all other vertices for the graph given below

![Figure 1](image)

16. Find the shortest unweighted path from B to all other vertices for the graph given in Figure 1.
17. Explain Dijkstra's algorithm and solve the single source shortest path problem with an example

18. Illustrate with an example, the linked list representation of graph
19. Find the shortest path from node 1 to 7 using shortest path algorithm.

![Figure 2](image)

20. Explain BFS and DFS in detail. Write the algorithm.
21. Write ADT routines for DFS algorithm.
22. Formulate an algorithm to find the shortest path using Dijkstra’s algorithm
23. Write an algorithm to find the minimum cost spanning tree of a weighted undirected graph
24. What is single source shortest path problem? Discuss Dijkstra’s single source shortest path algorithm with an example.
25. Explain Dijkstra's algorithm using the following graph. Find the shortest path between v1 to v2, v3, v4, v6, v7

26. Construct minimum spanning tree for the graph shown below

27. Discuss in detail the applications of graphs.
28. Find MST for the following graph
29. (i) What is a strongly connected graph? Give an example. (4)
(ii) Write the algorithm to compute lengths of shortest path