SATHYABAMA UNIVERSITY

(Established under Section 3, UGC Act 1956)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

SCSX4008-Data structures and OOP‘s Lab

Syllabus

1. Write C++ program to calculate final velocity using the formula: $v = u + a * t$, with initial velocity, acceleration and time as input.
2. Write C++ program to find the area of square, rectangle, circle using function overloading concept.
3. Write C++ program to change the sign of an operands using unary operator overloading concept.
4. Write C++ program to add two complex numbers using binary operator overloading concept.
5. Write C++ program to find mean value of two integers using friend function concept.
6. Write C++ program to multiple and divide two different data type using inline function concept.
7. Implement parametrized constructor, default conctructor, copy constructor and destructor
8. Write C++ program to enter the sale value and print the agent's commission using single inheritance.
9. Write C++ program to enter salary and output income tax and net salary using multiple inheritance concept.
10. Write C++ program to enter the unit reading and output the customer's telephone bill using hierarchical inheritance.
11. Write C++ program to find the grade of the students based on academic marks and sports using multilevel inheritance.
12. Write a program having student as an abstract class and create many derived classes such as Engineering, Medical etc from student's class. Create their objects and process them.
13. Write C++ program to count the words and characters in given text using virtual function.
14. Write C++ program to calculate net pay of employee using virtual base class concept.
15. Write C++ program to calculate division of two number with a try block to detect and throw an exception if the condition" divide by-zero" occurs.
16. Write C++ program to merge two files into one single file.
17. Write C++ program to swap two values using funtion templates.
18. Write C++ program to sort the numbers using class templates.

EXP NO: 1 **CALCULATING VELOCITY**

AIM:

To write a program to calculate final velocity using the formula: v=u + a*t, with initial velocity, acceleration and time as input.

ALGORITHM:

Step 1: Declare the class, velocity with the variables, u, a and t as integer type.

Step 2: Define the function, getdata() to get the input from the user.

Step 3: Define the function, calculate() to find final velocity and display the result.

Step 4: Call the getdata() and calculate() functions from the main function by creating an instance of the class velocity.

EXP NO: 2 **FUNCTION OVERLOADING**

AIM:

To write a program to find the area of square, rectangle, circle using function overloading concept.

ALGORITHM:

Step 1: Declare the class with class name shapes.

Step 2: Define the function, area() to be overloaded to find the area of different Shapes such as square, rectangle and circle.

Step 3: In the main(), call all those functions using the instance of class shapes.

EXP NO: 3 **UNARY OPERATOR OVERLOADING**

AIM: To write a program to change the sign of an operands using unary operator overloading concept.

ALGORITHM:

Step 1: Declare the class with the class name, space and three data members x, y and z of int type.

Step 2: Define the operator '–' to be overloaded using the keyword operator in the public area of the class.

Step 3: Define getdata() and display functions to get the data and to display results to user.

Step 4: Create an instance and call the getdata() from main()

Step 5: Change the sign of the object by calling the operator overloaded function.

Step 6: Call the display() to display the manipulated data.

EXP NO: 4 BINARY OPERATOR OVERLOADING

AIM:

To write a program to add two complex numbers using binary operator overloading concept.

ALGORITHM:

Step 1: Declare the class with the class name, complex and two data members r, im of integer type.

Step 2: Define getdata() to get the values of r and im.

Step 3: Define the operator '+' to be overloaded using the keyword operator in the public area of the class.

Step 4: Define display() to display the result.

Step 5: Create two instances c1, c2 to invoke getdata().

Step 6: Add two complex instances c1 and c2 and assign it to c3.

EXP NO: 5 FRIEND FUNCTION

AIM:

To write a program to find mean value of two integers using friend function concept.

ALGORITHM:

Step 1: Declare a class with the class name, sample.

Step 2: Define the setvalue() to initialize the variables x and y.

Step 3: Declare the function mean() with the keyword Friend and pass the object as an argument.

Step 4: Define the friend function mean() outside the class.

Step 5: Create an instance to call setvalue().

Step 6: From the main(), call the friend function like a normal function.

EXP NO: 6 INLINE FUNCTIONS

AIM:

To write a program to multiple and divide two different data type using inline function concept.

ALGORITHM:

Step 1: Declare the class with the class name, inlinefunc with data members a, b as integer type and c, d as float type.

Step 2: Define the functions, multiply() and divide() using the keyword inline.

Step 3: Create an instance and invoke the inline functions.

EXP NO: 7 CONSTRUCTORS

AIM:

To write a program to implement parameterized constructor, copy constructor, default constructor and destructor.

ALGORITHM:

Step 1: Declare the class with the class name, code with a data member id.

Step 2: Define the constructor functions, such as default, parameterized and copy constructors.

Step 3: Create instances and invoke all the constructors.

EXP NO: 8 SINGLE INHERITANCE

AIM:

To write a program to enter the sale value and print the agent's commission using single inheritance.

ALGORITHM:

Step 1: Declare the class with the class name, base and define the member function getdata() to get the input for a and b.

Step 2: Declare the class with class name, derived which derives publicly from class base and define the function addition() to find the total and print the result.

Step 3: In the main(), create an instance of class derived.

Step 4: Invoke the getdata() and addition() of class base and derived respectively.

EXP NO: 9 MULTIPLE INHERITANCE

AIM:

To write a program to enter salary and output income tax and net salary using multiple inheritance concept.

ALGORITHM:

Step 1: Declare the class with the class name, salary with variables BP, HRA and DA. Define getdetails() to get the input of three variables and find the total.

Step 2: Declare the class with class name, taxcal and find the tax amount to be deducted in the function deduct().

Step 3: Declare the class with the class name, netsalary which publicly derives from classes salary and tax. Define netpay() to deduce the tax from total.

Step 4: Create an instance for the derived class netsalary and invoke the functions.

EXP NO: 10 HIERARCHICAL INHERITANCE

AIM:

To write a program to generate the telephone bill using hierarchical inheritance.

ALGORITHM:

Step 1: Declare the class with the class name, bill with attributes like customer name, phone number and address. Define methods getdata() to get the input and display() to display.

Step 2: Declare two sub classes for the above base class, name them as mobile and landline. Declare relevant attributes and methods.

Step 3: Declare again sub classes prepaid and postpaid by inheriting mobile class. Declare relevant attributes and methods.

EXP NO: 11 MULTI- LEVEL INHERITANCE

AIM:

To write a program to find the grade of the students based on academic marks and sports using multilevel inheritance.

ALGORITHM:

Step 1: Declare the class with the class name, student with variables rollno and name. Define getdata() to get the rollno and name.

Step 2: Declare the class with class name, academic which derives publicly from student class. Declare the varaibles m1 and m2 and getmarks() as member function.

Step 3: Declare the class with class name, grade which publicly derives from academic class. Declare getscore() to get the value of variable score.

Step 4: Declare the variable total and find the total of m1, m2, score.

Step 5: Define the calculate() and find the grade of the student.

EXP NO: 12 ABSTRACT CLASS

AIM:

To write a program to present student's profile using Abstract class.

ALGORITHM:

Step 1: Declare the class with the class name, student with variables rollno and name. Only declare getdata(), display() methods without defining.

Step 2: Declare few more sub classes called medical dept, eng dept etc for the above student class with relevant attributes and methods.

Step 3: Define the inherited methods seperately in sub classes according to the department's need.

EXP NO: 13 VIRTUAL FUNCTION

AIM:

To write a program to implement the virtual function concept.

ALGORITHM:

Step 1: Declare the class with the class name, base and define the display() and show() methods.

Step 2: Declare the show() with the keyword virtual.

Step 3: Declare the class with class name, derived which publicly derives from base and define the display() and show() methods .

Step 4: In main(), create object for base and derived classes. Create a base class pointer and invoke all the functions using base pointer.

EXP NO: 14 VIRTUAL BASE CLASS

AIM:

To write a program to calculate net pay of employee using virtual base class concept.

ALGORITHM:

Step 1: Declare the class with the class name, employee and define the function getname() to input the employee name.

Step 2: Declare the class with class name, salary which is publicly derives from employee with the keyword virtual. Define getdetails() to get the input of the variables BP, HRA and DA and find the total.

Step3: Declare the class with class name, taxcal which publicly derives from employee with the keyword virtual. Find the tax amount to be deducted in the function deduct().

Step4: Declare the class with the class name, netsalary which publicly derives from classes salary and tax. Define netpay() to deduce the tax from total.

Step5: Create an instance for the derived class netsalary and invoke the functions.

EXP NO: 15 EXCEPTION HANDLING

AIM:

To write a program to calculate division of two number with a try block to detect and throw an exception if the condition" divide –by-zero" occurs.

ALGORITHM:

Step 1: In the main(), get the inputs for dividend and divisor from the user.

Step 2: In the try block find the division of two values. If the divisor is zero, it has to throw the "divide-by-zero" exception.

Step 3: The exception thrown should be caught in the catch block.

EXP NO: 16 **FILE MERGE**

AIM:

To write a program to merge two files into one single file.

ALGORITHM:

Step 1: Declare the main()

Step 2: Declare the two source file name and the destination file names.

Step 3: Open the destination file in append mode.

Step 4: Open the two source files in read mode.

Step 5: Copy the contents of the two source files to the destination files.

Step 6: Close the source files and the two destination files.

EXP NO: 17 **FUNCTION TEMPLATES**

AIM:

To write a program to swap two values using function templates.

ALGORITHM:

Step1: Create the template function swap() to exchange any two values.

Step 2: Define the function fun() and pass the arguments/parameters to be swapped. Invoke the swap() within fun().

Step 3: In the main(), invoke the fun().

EXP NO: 18 **CLASS TEMPLATES**

AIM:

To write a program to sort the numbers using class templates.

ALGORITHM:

Step1: Create a class template function sort() to arrange the numbers.

Step 2: Create a class and declare the instance of the template class as a variable.

Step 3: Create the main(), invoke the required functions.

Step 4: Display the result.

1. **SEARCHING**
    a. Linear Search
    b. Binary Search
2. **SORTING**
    a. Quick Sort
    b. Heap Sort
3. Divide & Conquer – Merge Sort
4. Implementation of singly linked lists.
5. Stack implementation using Array.
6. Queue implementation using Linked List.
7. Binary Search Tree Traversal.
8. Implementation of Dijkstra's Algorithm
9. Rabin Karp String Pattern Matching Algorithm.
10. Implement Kruskal's algorithm using Greedy approach.
11. N-queens problem using backtracking.

1. SEARCHING ALGORITHMS

A.LINEAR SEARCH

AIM: To write and execute a program to search for the given element in a array of n elements.

ALGORITHM:
Step 1: Declare the class with array a and the six N as data member.
Step 2: Declare getdata(),search(element) as member functions of the class.
Step 3: Create objects for the given class and exercise the routines present in the given class.

Algorithm for getdata()
Step 1: Repeat the following for I varying from 1 to N by 1
Step 1.1 Read a[I]
Algorithm for search (element)
Step 1: Repeat the following for I varying from 1 to N by 1
Step 1.1 check if a[I] =element then
Step 1.1.1: Store i in the position array
Step 1.1.2: increment the no. of occurrences by 1
Step 2: Check if the no. of occurrences<0 then
Step 2.1 Print the element not found
Else
Step 2.2 Print the element is found in the following positions
Step 2.2.1: For i=0;i<no.of occurances;i=i+1
Step 2.2.1.1: Print pos[i]
Step 3: Print the no. of occurrences.

B.BINARY SEARCH

AIM: To write and execute a program to search for the given elements in an array of n elements using binary search.

ALGORITHM:
Step 1: Declare the class with array a and the size N as data member.
Step 2: Declare getdata(),search(element) as member functions of the class
Step 3: Create objects for the given class and exercise the routines present in the given class.
Algorithm for getdata()
Step 1: Repeat the following for I varying from 1 to N by 1
Step 1.1: read a[I]
Algorithm for search (element)
Step 1: Let lower limit=1
Step 2: Let higher limit = N
Step 3: Repeat the following until lower <=upper
Step 3.1: Let middle position = (lower limit + Upper limit) / 2
Step 3.2: Check if a [I] =a [middle] then print the position 'mid' and return
Step 3.3: Check if a [I] < a [middle] then upper limit=middle -1
Step 3.3.1: Else lower limit = middle +1

2. SORTING TECHNIQUES:

A. QUICK SORT

AIM: To write and execute a program to sort the given n numbers in ascending order using quick sort method.

ALGORITHM:
Step1: Declare the class with array a and the size N as data member.
Step2: Declare getdata(),quicksort() and display() as member functions of the class.
Step3: Create objects for the given class and exercise the routines present in the given class.

Algorithm for getdata ()
Step1: Read the size of the array (n)
Step2: Repeat the following for I varying from 1 to n by 1
Step 2.1: Input a (I)
Step3: Call quicksort(0,n-1)

Algorithm for display ()
Step 1: Repeat the following for I varying from 1 to n by 1
Step 1.1) Print a (I)

Algorithm for quicksort (l, r)
Step 1: if (l>=r) return
Step 2: i=1, j=r+1
Step 3: pivot=a [l]
Step 4: Repeat the following
Step 4.1: Repeat the following as long as a [l] < pivot
Step 4.1.1: l=l+1
Step 4.2: Repeat the following as long as a[j]>pivot
Step 4.2.1: j=j-1
Step 4.3: if (l>=j) break
Step 4.4: swap (a [l], a[j])
Step 5: a [l] =a[j]
Step 6: a[j] =pivot
Step 7: Call quicksort(l,j-1)
Step 8: Call quicksort(j+1,r)
Step 9: return

B. HEAP SORT

AIM: To write and execute a program in java to sort the given N numbers using heap sort.

ALGORITHM:
Step 1: Declare the class with array a and the size N as data member
Step 2: Declare heapsort(),buildheap(),heapify() and display() as member functions of the class.
Step 3: Create objects for the given class and exercise the routines present in the given class.

Algorithm For Heapsort()
Step 1: Call Buildheap(A)
Step 2: Repeat For I=Heapsize Down To 2
Step 2.1 Swap Between A[1] And A[I]
Step 2.2 heapsize=heapsize-1
Step 2.3 call heapify(a,1)

Algorithm For buildheap(a)
Step 1: n=heapsize
Step 2: Repeat for i= n/2 down to 1 step -1
Step 2.1 call heapify(a,i)

Algorithm For heapify(a,i)
Step 1: l=2 * i
Step 2: r=l + 1
Step 3: if l<=heapsize and a[l]>a[i] then large=l
Step 3.1: else large=i
Step 4: if r<=heapsize and a[r]>a[large] then large=r
Step 5: if i != large
Step 5.1: swap a[i] and a[large]
Step 5.2: call heapify(a,large)

3. DIVIDE & CONQUER - MERGE SORT

AIM: To write and execute a program to merge the given two sorted lists using merge sort using divide and conquer technique.

ALGORITHM:
Step1: Declare the class with array a and the size N as data member
Step2: Declare getdata(),mergesort(),merge() and display() as member functions of the class
Step 3: Create objects for the given class and exercise the routines present in the given class.

Algorithm for mergesort (low, high)
Step 1: Check if low<high then
Step 1.1: Call mergesort(low,mid)
Step 1.2: Call mergesort(mid+1,high)
Step 1.3: Call merge (low, mid, high)
Step 2: Return

Algorithm for merge(low,mid,high)
Step 1: L=0
Step 2: i=low
Step 3: j=mid+1
Step 4: Repeat until (i<=mid) and (j<=high)
Step 4.1: Check if a[i]<a[j]
Step 4.1.1: temp[L]=a[i]
Step 4.1.2: i=i+1
Step 4.1.3: L=L+1
Step 4.2: else
Step 4.2.1: temp[L]=a[j]
Step 4.2.2: j=j+1
Step 4.2.3: L=L+1

Step 5: Check if i > mid
Step 5.1: Repeat until j<=high
Step 5.1.1: temp[L]=a[j]
Step 5.1.2: j=j+1
Step 5.1.3: L=L+1
Step 6: Else
Step 6.1: Repeat until i<=mid
Step 6.1.1: temp[L]=a[i]
Step 6.1.2: i=i+1
Step 6.1.3: L=L+1
Step 7: Repeat for m=0 , m<=L-1, m++
Step 7.1: a[low+m]=temp[m]

4.SINGLY LINKED LIST

AIM: To Write and execute a program in c++ to create a singly linked list.

ALGORITHM:
Step 1: Define a class for the list
Step 2: Let the node structure of the classs list contains data and link (address of the next node)
Step 3: Let us create() ,insertion_begin(),insert_middle(), append(), deletion (), search(), display() be the member functions of the class.
Step 4: Create objects for the class set and do insertion, deletion, search & display operations.

Algorithm for creating empty list
Step 1: Assign head =Null
Step 2:Assign no. of nodes in the list as 0

Algorithm for create
Step 1: assign temp=new node
Step 2: Read temp->data
Step 3: assign temp->link=NULL

Algorithm for insert_begin
Step 1: Call create()
Step 2: Assign temp->link=head
Step 3: Assign head = temp
Step 4: Increment the no. of nodes in the list

Algorithm for append
Step 1: if head= NULL
Step 1.1: Call insert_begin()
Step 2: else
Step 2.1: Call create()
Step 2.2: temp->link=NULL
Step 2.3: Assign prev=head
Step 2.4: Repeat the following until prev->link!=NULL
prev=prev->link;

Step 2.5: prev->link=temp
Step 2.6: Increment the count of nodes

Algorithm for insertion in the middle
Step 1: Read the position 'pos' to be inserted
Step 2: if (head->link==NULL) and pos=1
Step 2.1: insert_begin()
Step 3: if (head->link==NULL) and pos=2
Step 3.1: Call append()
Step 4: else if (pos>=2) and (pos<=ct))
Step 4.1: prev=head
Step 4.2: Repeat for i=2 , i<=pos-1 , i++
Step 4.2.1: prev=prev->link
Step 4.3: next=prev->link
Step 4.4: temp=new node
Step 4.5: Read temp->data
Step 4.6: temp->link=next
Step 4.7: prev->link=temp
Step 4.8: Increment the count

Algorithm for deleting a node
Step 1: if head =NULL
Step 1.1: Print "Empty list"
Stept 2: Else do the following
Step 2.1: Read the postion whose element is to be deleted
Step 2.2: if pos=1 then
Step 2.2.1: next=head->link
Step 2.2.2: head=next
Step 2.2.3: Decrement the no. of nodes present in the list
Step 2.3: else do th following
Step 2.3.1: if pos>=2 and pos<=ct
Step 2.3.1.1: Assign prev=head
Step 2.3.1.2: Repeat for i=2 , i<=pos-1, i=i+1
prev=prev->link;
Step 2.3.1.3: temp=prev->link
Step 2.3.1.4: next=temp->link
Step 2.3.1.5: prev->link=next
Step 2.3.1.5: Decrement the count

Algorithm for search
Step 1: Assign flag=0
Step 2: if head=NULL then
Step 2.1: Print "Empty list"
Step 3: Else do the following
Step 3.1: Read the element 'e' which is to be searched
Setp 3.2: cur=head;

Step 3.3: Repeat for i=1 ; i<=ct Increment i to 1
Step 3.3.1: if cur->data =e then
Step 3.3.1.1: pos=i
Step 3.3.1.2: Increment the flag and break the loop
Step 3.3.2 else cur=cur->link
Step 3.4: if flag==1 then
Step 3.4.1: Print the position
Step 3.5: Else
Step 3.4.2: print element not found

Algorithm for display
Step 1: Assign current node cur=head
Step 2: Display the no. of nodes in the list
Step 3: Repeat until cur!=NULL
Step 3.1: cur =cur->link

5. STACK IMPLEMENTATION USING ARRAY

AIM: To write and execute a program in c++ to perform push and pop operations in stack using array.

ALGORITHM:
Step 1: Declare the class for the stack data structures.
Step 2: Read the maximum capacity of the stack
Step 3: Initialize top=0
Algorithm for push
Step1: Read the element 'x' to be pushed
Step2: stack[top]=x
Step3: top++
Algorithm for pop
Step1: If top = NULL print empty stack
Else
Step 1.1: return stack[top]
Step 1.2: top--
Algorithm for display ()
Step 1: For I=0 to top
Step 1.1: print stack[top]

6. QUEUE IMPLEMENTATION

AIM: To write and execute a program in c++ to perform insertion and deletion operations in queue using linked list.

ALGORITHM:
Step 1: Declare the class for the queue data structure.
Step 2: Declare the front and rear pointers and queue structure as data members.

Step 3: Declare insert () and delete () as member functions of the class.
Step 4: Create objects for the given class and do insertion and deletion operations.

Algorithm for insert (element)

Step 1: Create a node pointed to by a pointer ptr
Step 2: Let ptr -> data = element
Step 3: Check if front =NULL then
Step 3.1: Let front = rear = ptr
Step 3.2: rear -> link = NULL
Else
Step 3.1: ptr -> link =NULL
Step 3.2: rear -> link = ptr
Step 3.3': rear = ptr

Algorithm for delete (element)

Step 1: If front = NULL print empty queue
Else
Step 1.1: front = front -> link

Algorithm for display ()

Step 1: Let current = front
Step 2: while current -> link ! = NULL do the following
Step 2.1: Display current node contents
Step 2.2: current = current -> link
Step 3: return.

7. BINARY SEARCH TREE TRAVERSAL

AIM : To write and execute a program in c++ to perform traversals in a binary search tree.

ALGORITHM:

Step 1: Declare the class for the BST data structure.
Step 2: Declare data, left child pointers,right child pointers of the BST structure as data members.
Step 3: Declare insert () , create(), preorder(), postorder() and inorder() as member functions of the class.
Step 4: Create objects for the given class and do insertions and binary search tree traversals

Algorithm for insert()

Step 1: tmp=new node
Step 2: Read tmp->data
Step 3: tmp->lc=NULL
Step 4: tmp->rc=NULL
Step 5: root=create(tmp,root)

Algorithm for *create(node *tmp,node *p)

Step 1: Check if p==NULL
Step 1.1: p=tmp

Step 1.2: return p
Step 2: else if tmp->data<p->data
Step 2.1: p->lc=create(tmp,p->lc)
Step 3: else if tmp->data > p->data
Step 3.1: p->rc=create(tmp,p->rc)
Step 4: else if tmp->data = p->data
Step 4.1: Print "already exist..try again!"
Step 4.2: return p
Step 5: return p

Algorithm for inorder(node *p)
Step 1: Chek if p!=NULL
Step1.1: Call inorder(p->lc)
Step1.2: Print p->data
Step 1.3: Call inorder(p->rc)

Algorithm for preorder(node *p)
Step 1: Chek if p!=NULL
Step1.1 Print p->data
Step1.2: Call preorder(p->lc)
Step 1.3: Call preorder(p->rc);

Algorithm for postorder(node *p)
Step 1: Chek if p!=NULL
Step1.1: Call postorder(p->lc)
Step 1.2: Call postorder(p->rc)
Step 1.3: Print p->data

8. DIJKSTRA's ALGORITHM

AIM : To write and execute a program in c++ to solve the single source shortest-paths problem on a weighted,directed graph.

ALGORITHM:
Step 1: Declare the class
Step 3: Declare read(), dijkstra(), output() as member functions of the class.
Step 4: Create objects for the given class and find the shortest path from a single source

Algorithm for read()
Step 1: Read the no. Of vertices 'n'
Step 2: Repeat for i=1, i<=n , i++
Step 2.1: Repeat for j=1, j<=n, j++
Step 2.1.1: if i = j then assign cost[i][j]=INFINITY
Step 2.1.2: else Read cost[i][j]
Step 3: Read the source 'src' where value of src should be between 0 and n

Algorithm for initialize()
Step 1: Repeat for int i=1, i<=n, i++
Step 1.1: visited[i] = 0
Step 1.2: predecessor[i] = -1
Step 1.3: distance[i] = INFINITY
Step 2: Assign distance[src]=0

Algorithm for getmin()
Step 1: Assign min = INFINITY
Step 2: Repeat for i=1, i<=n, i++
Step 2.1: check if (!visited[i]) and (min >= distance[i])
Step 2.1.1: min = distance[i]
Step 2.1.2: u = i
Step 3: return u

Algorithm for dijkstra()
Step 1: Call initialize()
Step 2: Initialize count = 0
Step 3: Repeat until count < n
Step 3.1: u = getmin()
Step 3.2: visited[u] = 1;
Step 3.3: Repeat for i=1, i<=n, i++
Step 3.3.1: Check if(!visited[i]) and (cost[u][i]>0)
Step 3.3.1.1: Check if(distance[i] > distance[u]+cost[u][i])
distance[i] = distance[u]+cost[u][i]
predecessor[i] = u
Step 3.4: Increment the count

Algorithm for printPath(int node)
Step 1: Check if(node= src) then print node
Step 2: else if(predecessor[node] == -1) then print no path from src to node
Step 3: else Call the function printPath(predecessor[node])
Step 3.1: Print node

Algorithm for output()
Step 1: Repeat for j=1, j<=n, j++
Step 1.1: Print distance[j], predecessor[j]
Step 2: Repeat for i=1, i<=n ,i++
Step 2.1: check if i = src then print src else call the function printpath(i)
Step 2.2: Print distance[i]

9. RABIN-KARP String pattern matching algorithm

AIM : To write and execute a program in C++ to perform Rabin Karp String Pattern Matching.

ALGORITHM:

Step 1: Initialize t=0,p=0
Step 2: Read the text 'txt' and pattern 'pat'
Step 3: Calculate n as the txt length and m as pattern length
Step 4: Initialize d=1,q=97,h=1
Step 5: repeat for i=0, i<m, i++
Step 5.1: calculate h=(h*d)%q
Step 6: Repeat for i=0, i<m, i++
Step 6.1: p=(d*p+pat[i]*h)%q
Step 6.2: t=(d*t+txt[i]*h)%q
Step 7: Repeat for i=0,i<=n-m, i++
Step 7.1: check if p = t
Step 7.1.1: Initialize found=1
Step 7.1.2: Repeat for j=0, j<m, j++
Step 7.1.2.1: check if (txt[i+j]!=pat[j]) then found=0 and break
Step 7.1.3: if(found =1) then print the position i and return
Step 7.2: t=0
Step 7.3: Check if (i<n-m)
Step 7.3.1: Repeat for r=i+1, r<=i+m, r++
Step 7.3.1.1: t=(d*t+txt[r]*h)%q
Step 8: Print "Match not found"

10. KRUSKAL's ALGORITHM (E, cost, n, t)

AIM : To write and execute a program in c++ to solve the minimum spanning –tree problem by implementing Kruskal's algorithm using greedy approach.

ALGORITHM :

Step 1: Construct a queue with edge costs such that they are in ascending order.
Step 2: Assign i = 0, mincost = 0
Step 3 : Repeat the following while i < n – 1 and queue is not empty
Step 3.1: Delete minimum cost edge (u, v) from queue
Step 3.2: Assign j = Find(u), k = Find(v)
Step 3.3: Check If j ≠ k then
Step 3.3.1: Assign i = i + 1
Step 3.3.2: Assign t[i,1] = u, t[i,2] = v
Step 3.3.3: Assign mincost = mincost + cost[u, v]
Step 3.3.4: Invoke Union(j, k)
Step 4: Check if i ≠ n – 1 then
Step 4.1 : Write "No spanning tree"
Else
Step 4.2 : Return mincost
Step 5 : Return

11. N-QUEENS PROBLEM USING BACKTRACKING

AIM: To write and execute a program in C++ to solve the n Queen problems using backtracking technique.

ALGORITHM:
Step 1: Define a class for n-Queen board
Step 2: The member functions of the class are nqueen () , trial(int row),good(int row), drawboard()
Step 3: Create objects for the given class and exercise the given

Algorithm for NQueen()
Step 1: Get the number of queens 'n'
Step 2: Check if feasible solution is found then print the solution
Step2.1: Check if(trial(0) then
Step 2.1.1: call the function drawboard();
else
Step 2.1.2: Write " there is no such feasible solution"

Algorithm for Trial(int row)
Step 1: Repeat the following for row varying from 0 to n-1 by 1
Step 1.1: Assign x[row]=i;
Step 1.2: check if(row = n-1 AND good(row)=TRUE) then
Step 1.2.1: return "TRUE"
Step 1.3 : Check if(row<n-1 AND good(row)=TRUE AND trial(row+1)=="TRUE" then
Step1.3.1: return "TRUE"
Step1.4: Assign x[row]= -1;
Step 2: return "FALSE"

Algorithm for Good(int row):

Step1.1 : Check if X[i]=row and abs(i-row)==abs(x[i]-x[row]) then
Step 1.1.1: return "false"
Else
Step1.1.2 : return "true"

Algorithm Drawboard()
Step 1: Repeat the following for j varying from 0 to n-1 by 1
Step 1.1: Check if(j==x[i]) then
Step 1.1.1: Write " 1 "
else
Step 1.1.3: Write " 0 "