bubble_sort.c

Details
Activity

```
#include <stdio.h>
#define SIZE 5
void BubbleSort(int [ ]);

int main() {
int a[SIZE]= {2, 1, 5, 3, 2};
int i;
printf("The elements of the array before sorting\n");
for (i=0; i < SIZE; i++)
                        printf("%4d", a[i]);
BubbleSort(a);
printf("\n\nThe elements of the array after sorting\n");
for (i=0; i< SIZE; i++)
                        printf("%4d", a[i]);
return 0;
}


void BubbleSort(int A[ ]) {
int i, pass, temp;
for (pass=1; pass < SIZE; pass++)
for (i=0; i < SIZE-1; i++)
if(A[i] > A[i+1]){
temp = A[i];
A[i] = A[i+1];
A[i+1] = temp;
}
}
```

Array average

```
#include<stdio.h>
#define SIZE 10
int main() {
int x[10] = {4, 3, 7, -1, 7, 2, 0, 4, 2, 13};
int i, sum=0;
float av;
for(i=0; i<SIZE; i++)
sum = sum + x[i];
av = (float)sum / SIZE;
printf("The average of the numbers = %.2f\n", av);
return 0;
}
```

Linear search

```
#include <stdio.h>
#define SIZE 10
int LinearSearch(int [], int);
```

```
int main() {
int a[SIZE]= {9, 4, 5, 1, 7, 78, 22, 15, 96, 45};
int key, pos;
printf("Enter the Search Key\n");
scanf("%d", &key);
pos = LinearSearch(a, key);
if(pos==-1)
printf("The search key is not in the array\n");
else
printf("The search key %d is at location %d\n", key, pos+1);
return 0;
}


int LinearSearch (int b[ ], int skey) {
int i;

for (i=0; i < SIZE; i++)
{
    if(b[i]==skey)
        return i;

}

    return -1;

}
```

Binary search

```
#include <stdio.h>
#define SIZE 10
int BinarySearch(int [ ], int);
int main(){
int a[SIZE]= {3, 5, 9, 11, 15, 17, 22, 25, 37, 68};
int key, pos;
printf("Enter the Search Key\n");
scanf("%d",&key);
pos = BinarySearch(a, key);
if(pos == -1)
        printf("The search key is not in the array\n");
else
        printf("The search key %d is at location %d\n", key, pos+1);
return 0;
}


int BinarySearch (int A[], int skey){
    int low=0, high=SIZE-1, middle;
    while(low <= high){
    middle = (low+high)/2;
    if (skey == A[middle])
        return middle;
```

```
    else if(skey <A[middle])
        high = middle - 1;
    else
        low = middle + 1;
    }

return -1;
}
```

Array average

```
#include<stdio.h>

int main()
{
    int n, sum=0;
    float avg;
    printf("Enter number of elements: ");
    scanf("%d", &n);

    int arr[n];

    printf("\nEnter %d values: ", n);
    for(int i=0; i<n;i++)
    {
        scanf("%d", &arr[i]);
        sum+=arr[i];
    }
    avg=(float)sum/n;

    printf("\nAvg is: %.2f", avg);
}
```

SLL.c

Details
Activity

```
#include<stdio.h>
#include<stdlib.h>
struct node{
        int value;
        struct node *next;
};
struct node *head;

void printList()
{
  if (head==NULL)   // no list at all
          return;
  struct node *temp =  head;
  while (temp != NULL)
  {
       printf("%d ->", temp->value);
        temp = temp->next;
  }
  printf("END (for now!)");
}

void insertHead(int num){
   //create a new node
   struct node *newItem;
   newItem=(struct node *)malloc(sizeof(struct node));
   newItem->value = num;
   newItem->next = NULL;
   //insert the new node at the head
   newItem->next = head;
   head = newItem;
}

void insertTail(int num){
   //create a new node to be inserted
   struct node *newItem;
   newItem=(struct node *)malloc(sizeof(struct node));
   newItem->value = num;
   newItem->next = NULL;
   // set prev to point to the last node of the list
   struct node *prev = head;
   while (prev->next != NULL)
       prev = prev->next;
   //newItem->next = NULL;
   prev->next = newItem;
}


void insertatPosition(int num, int pos){
   //create a new node to be inserted
   struct node *newItem;
   newItem=(struct node *)malloc(sizeof(struct node));
   newItem->value = num;
```

```
   newItem->next = NULL;
   // set prev to point to the desired node of the list
   struct node *prev = head;
   for (int i=0;i<pos-1;i++)
   {
       prev = prev->next;
   }

   newItem->next = prev->next;
   prev->next = newItem;
}


void insertAfterValue(int num, int val){
   //create a new node to be inserted
   struct node *newItem;
   newItem=(struct node *)malloc(sizeof(struct node));
   newItem->value = num;
   newItem->next = NULL;
   // set prev to point to the desired node of the list
   struct node *prev = head;
   while (prev->value != val){
       prev = prev->next;
   }
   newItem->next = prev->next;
   prev->next = newItem;
}


void deleteHead()
{
   struct node *cur;
   if (head==NULL)  //list empty
           return;
   cur = head;  // save head pointer
   head = head->next; //advance head
   free(cur);
}

void deleteTail(){
   if (head==NULL)       //list empty
           return;
   struct node *cur = head;
   struct node *prev = NULL;
   while (cur->next != NULL){
           prev = cur;
           cur=cur->next;
   }
   if (prev != NULL)
           prev->next = NULL;
   free(cur);
}


void deletefromPosition(int pos){
   if (head==NULL)       //list empty
           return;
```

```
   struct node *cur = head;
   struct node *prev = NULL;

   for(int i=0;i<pos;i++)
   {
       prev = cur;
           cur=cur->next;
   }

 if (prev != NULL)
           prev->next = cur->next;
 free(cur);
}

void deleteVal(int x){
   if (head==NULL)       //list empty
           return;
   struct node *cur = head;
   struct node *prev = NULL;
   while (cur->value != x){
           prev = cur;
           cur=cur->next;
   }
 if (prev != NULL)
           prev->next = cur->next;
 free(cur);
}



int main()
{

    head=NULL;

    printf("\t\tImplementation of a Single Linked List\n\t\tCheck the slide
for Explanation\n\t\tCSE 203, Lec Raiyan");
    while(1)
    {
        int ch, num, pos, val;
        printf("\n\n1.Insert First\n2.Insert Last\n3.Insert Middle (Any other
pos)\n4.Insert After a Target Val\n5.Delete Head\n6.Delete Tail\n7. Delete
from a Position\n8.Delete a Value\n9.Print\n10. Add a function for practice
\n11.Exit\n\nEnter Choice: ");
        scanf("%d", &ch);

        if(ch==1)
        {
            printf("\nEnter val to insert: ");
            scanf("%d", &num);
            insertHead(num);
        }
        else if (ch==2)
        {
            printf("\nEnter val to insert: ");
            scanf("%d", &num);
            insertTail(num);
```

```
    }
    else if (ch==3)
    {
        printf("\nEnter val to insert: ");
        scanf("%d", &num);
        printf("\nEnter Position to insert: ");
        scanf("%d", &pos);
        insertatPosition(num, pos);
    }

    else if(ch==4)
    {
        printf("\nEnter val to insert: ");
        scanf("%d", &num);
        printf("\nEnter Value to insert after: ");
        scanf("%d", &val);
        insertAfterValue(num, val);
    }

    else if(ch==5)
    {
        deleteHead();
    }
    else if(ch==6)
    {
        deleteTail();
    }
    else if(ch==7)
    {
        printf("\nEnter Position to Delete from: ");
        scanf("%d", &pos);
        deletefromPosition(pos);
    }
    else if (ch==8)
    {
        printf("\nEnter Value to delete: ");
        scanf("%d", &val);
        deleteVal(val);
    }

    else if(ch==9)
    {
        printList();
    }

    else if(ch==10)
    {
        //ADD YOUR OWN FUNCTION - PRACTICE
    }

    else
    {
        printf("\n\n\t\tProgram Terminated\n\n");
        break;
    }
}
return 0;
```

```
}
```

```
#include<stdio.h>
#include<stdlib.h>

struct node
{
  int data;
  struct node *next;
};

struct node *f = NULL;
struct node *r = NULL;

void enqueue (int d)                  //Insert elements in Queue
{
  struct node *n;
  n = (struct node *) malloc (sizeof (struct node));
  n->data = d;
  n->next = NULL;
  if ((r == NULL) && (f == NULL))
    {
      f = r = n;
      r->next = f;
    }
  else
    {
      r->next = n;
      r = n;
      n->next = f;
    }
}

void dequeue ()                       // Delete an element from Queue
{
  struct node *t;
  t = f;
  if ((f == NULL) && (r == NULL))
    printf ("\nQueue is Empty");
  else if (f == r)
    {
      f = r = NULL;
      free (t);
    }
  else
    {
      f = f->next;
      r->next = f;
      free (t);
    }
```

```
}

void display ()
{                               // Print the elements of Queue
  struct node *t;
  t = f;
  if ((f == NULL) && (r == NULL))
    printf ("\nQueue is Empty");
  else
    {
      do
       {
         printf (" %d", t->data);
         t = t->next;
       }
      while (t != f);
    }
}

int main ()
{
  enqueue (34);
  enqueue (22);
  enqueue (75);
  enqueue (99);
  enqueue (27);
  printf ("Circular Queue: ");
  display ();
  printf ("\n");

  dequeue ();
  printf ("Circular Queue After dequeue: ");
  display ();
  return 0;
}
```

```
// Queue implementation in C

#include <stdio.h>
#define SIZE 5

void enQueue(int);
void deQueue();
void display();

int items[SIZE], front = -1, rear = -1;

int main() {
  //deQueue is not possible on empty queue
  deQueue();

  //enQueue 5 elements
  enQueue(1);
  enQueue(2);
  enQueue(3);
```

```
  enQueue(4);
  enQueue(5);

  // 6th element can't be added to because the queue is full
  enQueue(6);

  display();

  //deQueue removes element entered first i.e. 1
  deQueue();

  //Now we have just 4 elements
  display();

  return 0;
}

void enQueue(int value) {
  if (rear == SIZE - 1)
    printf("\nQueue is Full!!");
  else {
    if (front == -1)
      front = 0;
    rear++;
    items[rear] = value;
    printf("\nInserted -> %d", value);
  }
}

void deQueue() {
  if (front == -1)
    printf("\nQueue is Empty!!");
  else {
    printf("\nDeleted : %d", items[front]);
    front++;
    if (front > rear)
      front = rear = -1;
  }
}

// Function to print the queue
void display() {
  if (rear == -1)
    printf("\nQueue is Empty!!!");
  else {
    int i;
    printf("\nQueue elements are:\n");
    for (i = front; i <= rear; i++)
      printf("%d  ", items[i]);
  }
  printf("\n");
}
```

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

```
#include<stdlib.h>

struct node
{
  int data;
  struct node *next;
};

struct node *f = NULL;
struct node *r = NULL;

void enqueue (int d)                 //Insert elements in Queue
{
  struct node *n;
  n = (struct node *) malloc (sizeof (struct node));
  n->data = d;
  n->next = NULL;
  if ((r == NULL) && (f == NULL))
    {
      f = r = n;
      r->next = f;
    }
  else
    {
      r->next = n;
      r = n;
      n->next = f;
    }
}

void dequeue ()                      // Delete an element from Queue
{
  struct node *t;
  t = f;
  if ((f == NULL) && (r == NULL))
    printf ("\nQueue is Empty");
  else if (f == r)
    {
      f = r = NULL;
      free (t);
    }
  else
    {
      f = f->next;
      r->next = f;
      free (t);
    }


}

void display ()
{                              // Print the elements of Queue
  struct node *t;
```

```
  t = f;
  if ((f == NULL) && (r == NULL))
    printf ("\nQueue is Empty");
  else
    {
      do
       {
         printf (" %d", t->data);
         t = t->next;
       }
      while (t != f);
    }
}

int main ()
{
  enqueue (34);
  enqueue (22);
  enqueue (75);
  enqueue (99);
  enqueue (27);
  printf ("Circular Queue: ");
  display ();
  printf ("\n");

  dequeue ();
  printf ("Circular Queue After dequeue: ");
  display ();
  return 0;
}
```

```
#include<stdio.h>
#include<stdlib.h>

struct node
{
  int data;
  struct node *next;
};

struct node *f = NULL;
struct node *r = NULL;

void enqueue (int d)                //Insert elements in Queue
{
  struct node *n;
  n = (struct node *) malloc (sizeof (struct node));
  n->data = d;
  n->next = NULL;
  if ((r == NULL) && (f == NULL))
    {
      f = r = n;
      r->next = f;
    }
  else
```

```
    {
      r->next = n;
      r = n;
      n->next = f;
    }
}

void dequeue ()                       // Delete an element from Queue
{
  struct node *t;
  t = f;
  if ((f == NULL) && (r == NULL))
    printf ("\nQueue is Empty");
  else if (f == r)
    {
      f = r = NULL;
      free (t);
    }
  else
    {
      f = f->next;
      r->next = f;
      free (t);
    }


}

void display ()
{                               // Print the elements of Queue
  struct node *t;
  t = f;
  if ((f == NULL) && (r == NULL))
    printf ("\nQueue is Empty");
  else
    {
      do
       {
         printf (" %d", t->data);
         t = t->next;
       }
      while (t != f);
    }
}

int main ()
{
  enqueue (34);
  enqueue (22);
  enqueue (75);
  enqueue (99);
  enqueue (27);
  printf ("Circular Queue: ");
  display ();
```

```
  printf ("\n");

  dequeue ();
  printf ("Circular Queue After dequeue: ");
  display ();
  return 0;
}
```

```
#include<stdio.h>
#include<stdlib.h>

// Structure to create a node with data and the next pointer
struct node {
    int data;
    struct node * next;
};

struct node * front = NULL;
struct node * rear = NULL;

// Enqueue() operation on a queue
void enqueue(int value) {
    struct node * ptr;
    ptr = (struct node * ) malloc(sizeof(struct node));
    ptr -> data = value;
    ptr -> next = NULL;
    if ((front == NULL) && (rear == NULL)) {
        front = rear = ptr;
    } else {
        rear -> next = ptr;
        rear = ptr;
    }
    printf("Node is Inserted\n\n");
}

// Dequeue() operation on a queue
int dequeue() {
    if (front == NULL) {
        printf("\nUnderflow\n");
        return -1;
    } else {
        struct node * temp = front;
        int temp_data = front -> data;
        front = front -> next;
        free(temp);
        return temp_data;
    }
}

// Display all elements of the queue
void display() {
    struct node * temp;
    if ((front == NULL) && (rear == NULL)) {
```

```
        printf("\nQueue is Empty\n");
    } else {
        printf("The queue is \n");
        temp = front;
        while (temp) {
            printf("%d--->", temp -> data);
            temp = temp -> next;
        }
        printf("NULL\n\n");
    }
}

int main() {
    int choice, value;
    printf("\nImplementation of Queue using Linked List\n");
    while (choice != 4) {
        printf("1.Enqueue\n2.Dequeue\n3.Display\n4.Exit\n");
        printf("\nEnter your choice : ");
        scanf("%d", & choice);
        switch (choice) {
            case 1:
                printf("\nEnter the value to insert: ");
                scanf("%d", & value);
                enqueue(value);
                break;
            case 2:
                printf("Popped element is :%d\n", dequeue());
                break;
            case 3:
                display();
                break;
            case 4:
                exit(0);
                break;
            default:
                printf("\nWrong Choice\n");
        }
    }
    return 0;
}
```

```
#include<bits/stdc++.h>

using namespace std;

class Stack{
private:
    int top;
    int capacity;
```

```
    int *arr;

public:
    Stack(int n){
        top=-1;
        capacity=n;
        arr=new int[capacity];
    }

    int isStackFull(){
        if(top==capacity-1){
            return 1;
        }
        else{
            return 0;
        }
    }

    int isStackEmpty(){
        if(top==-1){
            return 1;
        }
        else{
            return 0;
        }
    }


    int push(int val){
        if(isStackFull()){      ///top=-1; capacity=5;
            return 0;           ///push(5);
        }
        else{
            top++;                  ///top=0; arr[0]=5;
            arr[top]=val;           /// top=4;
            return 1;
        }                       ///push(9);
    }

    int pop(){
        if(isStackEmpty()){
            return -1;
        }
        else{
            int tmp;
```

```
            tmp=arr[top];
            top--;
            return tmp;
        }
    }



    void displaystack(){
        if(isStackEmpty()){
            cout<<"Stack is empty"<<endl;
        }
        else{
            cout<<"\nElements of Stack:"<<endl;
            for(int i=top;i>=0;i--){
                cout<<arr[i]<<" "<<endl;
            }
        }

    }

};


int main(){

    int choice=1, n, value,xx,yy;
    Stack stk(5);
    while(choice!=0)
    {
        cout<<endl;
        cout<<"0 - Exit."<<endl;
        cout<<"1 - Push Item."<<endl;
        cout<<"2 - Pop Item."<<endl;
        cout<<"3 - Display Items (Print STACK)."<<endl;

        cout<<"Enter your choice: ";
        cin>>choice;

        switch(choice){
            case 0:
                break;

            case 1:
                cout<<"Enter Value:\n";
```

```
                cin>>value;
                xx=stk.push(value);
                if(xx==0){
                    cout<<"\nStack is full (overflow)"<<endl;
                }
                else{
                    cout<<value<<" is pushed into stack"<<endl;
                }
                break;

            case 2:
                yy=stk.pop();
                if(yy==-1){
                    cout<<"\nStack is empty! (Underflow condition)"<<endl;
                }
                else{
                    cout<<"Popped value is "<<yy<<endl;
                }
                break;

            case 3:

                stk.displaystack();
                break;

            default:
                cout<<"Invalid choice. Enter Again."<<endl;
        }
    }return 0;
}
BST by faria maam


#include<bits/stdc++.h>
using namespace std;

struct Node{
    int key,level;
    Node *left, *right, *parent;
};

Node *root;

void init(){
    root = NULL;
```

```
}

void insertRoot(int val){

    root=new Node;
    root->key=val;
    root->left=NULL  ;
    root->right=NULL;
    root->parent=NULL;

}

void insertOther(int val){

    Node *temp=root;
    Node *prev=NULL;///prev pointer is used to keep trac of previos poiter of
temp
    while(temp!=NULL)
    {
        prev=temp;
        if(val<temp->key)
        {
            temp=temp->left;
        }
        else
        {
            temp=temp->right;
        }

    } ///while loop finish

    temp=new Node;
    temp->key=val;
    temp->left=NULL;
    temp->right=NULL;
    temp->parent=prev;

    if(val<prev->key)
    {
        prev->left=temp;
    }
    else{
        prev->right=temp;
    }
```

```
}

void insert(int val){
    if(root==NULL)
    {
        insertRoot(val);
    }
    else{
        insertOther(val);
    }
}

Node* findNode(int val){
    Node *temp=root;
    while(temp!=NULL)
    {
        if(temp->key==val)
        {
            return temp;
        }
        else if(val<temp->key)
        {
            temp=temp->left;
        }
        else
        {
            temp=temp->right;
        }
    }
    return temp;

}

Node* findMaximum(Node *node){
    Node *temp=node;
    while(temp->right!=NULL)
    {
        temp=temp->right;
    }
    return temp;

}
```

```
Node* findMinimum(Node *node){
    Node *temp=node;
    while(temp->left!=NULL)
    {
        temp=temp->left;
    }
    return temp;
}


Node* findSuccessor(Node *node){
    Node *temp=node->right;
    while(temp->left!=NULL)
    {
       temp=temp->left;
    }
    return temp;
}

void delete0Child(Node *node){
    Node *par=node->parent;
    if((node->key)<(par->key))
    {
        par->left=NULL;
    }

    else
    {
        par->right=NULL;
    }
    free(node);
}

void delete1Child(Node *node){
    Node *par,*child;
    par=node->parent;
    ///we will grab the child of deleting node
    if(node->left==NULL)
    {
        child=node->right;
    }
    else{
        child=node->left;
    }
```

```
    ///deleting node will be on left
    if((node->key)<(par->key))
    {
        par->left=child;
        child->parent=par;
    }
    ///deleting node is on right
    else
    {
        par->right=child;
        child->parent=par;
    }
}

void delete2Child(Node *node){
    Node *f=findSuccessor(node);
    node->key=f->key;///so successor will set
    if(f->left==NULL && f->right==NULL)
    {
        delete0Child(f);
    }
    else
    {
        delete1Child(f);
    }
}

bool deleteNode(int val)
{
    Node *t=findNode(val);
    if(t==NULL)
    {
        return false;
    }
    else
    {
        if(t->left==NULL && t->right==NULL)
        {
            delete0Child(t);
        }

        else if(t->left==NULL || t->right==NULL)
        {
            delete1Child(t);
        }
```

```
        else
        {
            delete2Child(t);
        }
        return true;
    }
}



int main(){
    init();


    while(1){
        cout<<"1. Insert\n2. Search\n3. Delete\n\n";
        int x;
        cin>>x;

        if(x==1){
            cout<<"Insert Value: ";
            int y;
            cin>>y;
            insert(y);
        }

        else if(x==2){
            cout<<"Enter the value you want to search ";
            int s;
            cin>>s;
            Node* t=findNode(s);
            if(t==NULL)
                cout<<"Not found"<<endl;
            else
                cout<<"Found"<<endl;
        }

        else if(x==3){
            cout<<"Delete Value: ";
            int y;
            cin>>y;
            bool b = deleteNode(y);
            if(b)   cout<<"Deleted"<<endl;
            else    cout<<y<<" not found"<<endl;
        }
```

```
    }
}

/*
1
44

1
17

1
88

1
32

1
65

1
97

1
28

1
54

1
82

1
29

1
76

1
80
*/
```

```
BFS by sazia maam
#include<iostream>
#include<queue>
using namespace std;

int n,e;
int adj[100][100];
int mark[100];
int dis[100];
int parent[100];

void initAdj(){
    for(int i=0; i<n; i++){
        for(int j=0; j<n; j++){
            adj[i][j] = 0;
        }
    }
}

void initMark(){
    for(int i=0; i<n; i++)  mark[i] = 0;
}

void initDis(){
    for(int i=0; i<n; i++)  dis[i] = 0;
}

void initParent(){
    for(int i=0; i<n; i++)  parent[i] = -1;
}

void printDisForAll(){
    for(int i=0; i<n; i++){
        cout<<"Distance of "<<i<<" from source is "<<dis[i]<<endl;
    }
    cout<<endl;
}

void printParentTree(int element){
   /* if(element==-1) return;
    printParentTree(parent[element]);*/

    cout<<element<<"->";

    while(parent[element]!=-1)
```

```
    {

        cout<<parent[element]<<"--";
        int x=parent[element];
        element = x;
    }
    cout<<"><";
}

void printParentTreeForAll(){
    for(int i=0; i<n; i++){
        cout<<"Parent tree for "<<i<<" is: ";
        printParentTree(i);
        cout<<endl;
    }
}

void bfs(int start){
    initMark();
    initDis();
    initParent();

    queue <int> q;
    q.push(start);
    mark[start] = 1;

    cout<<endl<<"Exploration order: ";

    while(q.size()!=0){       ///while(!q.empty()) this condition can be applied
also
        int king = q.front();
        q.pop();
        cout<<king<< " ";

        ///Finds the children of king
        for(int i=0; i<n; i++){
            if(adj[king][i]==1){
                int child = i;

                ///Finds the unmarked children of king
                if(mark[child]==0){
                    q.push(child);
                    mark[child] = 1;
                    dis[child] = dis[king] + 1;
                    parent[child] = king;
```

```
                }
            }
        }///For loop ends

    }///While Loop ends
    cout<<endl<<endl;
}


int main(){
    cin>>n>>e;
    initAdj();

    for(int i=1; i<=e; i++){
           // cout<<i<<endl;
        int x,y;
        cin>>x>>y;
        adj[x][y] = 1;
    }

    int start;
    cin>>start;

    bfs(start);
    printDisForAll();
    printParentTreeForAll();

    return 0;
}
/*
6 5

0 1
0 2
1 3
1 4
2 5
0
*/
```

```
Dfs by sazia maam

//#include<bits/stdc++.h>
#include<iostream>
#include <vector>
#include<cstdio>
#include<cstring>
using namespace std;

int t, visited[100], discover[100], finish[100], parent[100];
vector<int> adj[100];

void dfs(int node)
{
    visited[node] = 1;
    t = t+1;
    discover[node] = t;

    for(int i=0; i<adj[node].size(); i++)
    {
        int v = adj[node][i];

        if(visited[v]==0)
        {
            cout<<v<<" ";
            parent [v]=node;
            dfs(v);
        }
    }

    t = t+1;
    finish[node] = t;
    visited[node] = -1;
}

int main(){


        int edges, nodes, a, b;
        memset(visited,0, sizeof(visited));
        memset(discover,0, sizeof(discover));
        memset(finish,0,sizeof(finish));
        t=0;
        for(int i=0; i<=100;i++)adj[i].clear();
```

```
        cin>>edges>>nodes;
        for(int i=0; i<edges;i++)
        {
            cin>>a>>b;
            adj[a].push_back(b);
            adj[b].push_back(a);
        }

        for(int i=0;i<nodes;i++)
        {
            cout<<i<<"->";
            for(int j=0;j<adj[i].size();j++)
            {
                cout<<adj[i][j]<<" ";
            }
            cout<<endl;
        }
        cout<<endl;
        for(int i=1; i<=nodes ;i++)
        {
            if(visited[i] == 0)
            {
                cout<<i<<" ";
                dfs(i);
            }
        }
       cout<<endl;

       cout<<"Parent:  ";
         for(int i=1;i<=nodes;i++)
        {
            cout<<parent[i]<<" ";
        }



}
```