**DAA MINOR-2 PART-1**

**2. Starting with (4), show that: as n grows large, rate of growth of In is in O(n log n).**

Since we know that In= 2-k(k-1)/2

In /2k(k-1)/2

In = 1 + nC1 + nC2/2 + nC3/23 + ………………. + 1/2n(n-1)/2

In < n + n2 + n3 + …………………..+ nk

Here we know that ‘n’ no. of vertices and ‘k’ no. of edges

So considering all possible combinations ..

Let n=2k

So we have the equation like

In < n + n2 + n3 + ………… + nlog n

Hence **T(In) = O(nlog n)** .

**1: Write a C or C++ code implementing the backtracking algorithm.**

C++ code for backtracking algorithm:

#include<iostream>

#include<stack>

using namespace std;

#define n 5;

int clearArray(int A[]){

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

A[i]=A[i+10];

}

int del1(int A[] ,int i)

{

A[i]=A[i+10];

return A[i+1];

}

int size(A[]){

int i=1;

int a=0;

while(A[i++]>0){

a++;}

return a+1;

}

int isConnected(int G[][], int A[], int v){

if(!A)

return 0;

for(int i=0; i<size(A); i++){

if(G[A[i]][v]){

return 0;

}}

else

return 1;

}

int main(){

int G[][]={{0,1,0,0,1},{1,0,1,1,1},{0,1,0,1,0},{0,1,1,0,1},{1,1,0,1,0}};

int V[n];

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

V[i]=i;}

int A[100];

memset(A,-1,100);

stack<int>S;

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

int T[i]=A[i];

for(int i=0 ; i<5; i++){

clearArray(A);

A[0]=V[i];

S.push(V[i]);

for(int j=0; j<5; j++){

T[j]=V[j+i];

}

int d=0;

while(size(A)!=0){

if(size(T)==0){

int x=S.top();

S.pop();

A[Size(A)]=0;

for(int i=x+1;i<5;j++){

T[i]=V[i];

}

int p=del1(A,d)

else if(!isConnected(G,A,p)){

A[i+1]=p;

S.push(p);

if(size(A)>size(MIS)){

for(int i=0; i<size(A);i++)

MIS[i]=A[i];

} }

del1(A,d);

d++;

}}}

for(int i=0;i<size(MIS);i++){

cout<<MIS;

cout<<size(MIS);}