**Lower-Upper Bound**

**Lower-Upper Bound**

#include <iostream>

#include <algorithm>

#include <vector>

using namespace std;

int main ()

{

int myints[] = {10,20,30,30,20,10,10,20};

std::vector<int> v(myints,myints+8); // 10 20 30 30 20 10 10 20

std::sort (v.begin(), v.end()); // 10 10 10 20 20 20 30 30

std::vector<int>::iterator low,up;

low=std::lower\_bound (v.begin(), v.end(), 20); // 0 index system

up= std::upper\_bound (v.begin(), v.end(), 20); // 0 index system

cout << "lower\_bound at position " << (low- v.begin()) << '\n'; //ans: 3

cout << "upper\_bound at position " << (up - v.begin()) << '\n'; //ans: 6

low=lower\_bound (v.begin(), v.end(), 25); //

up= upper\_bound (v.begin(), v.end(), 25); //

cout << "lower\_bound at position " << (low- v.begin()) << '\n'; //ans: 6

cout << "upper\_bound at position " << (up - v.begin()) << '\n'; //ans: 6

low=lower\_bound (v.begin(), v.end(), 50); //

up= upper\_bound (v.begin(), v.end(), 50); //

cout << "lower\_bound at position " << (low- v.begin()) << '\n'; //ans: 8

cout << "upper\_bound at position " << (up - v.begin()) << '\n'; //ans: 8

return 0;

}

#include <iostream>

#

**LCM of n numbers [Prime and mod]**

using namespace std;

#define inf 1e9

#define NN 100010

#define mod 1000000007

vector<int>arr;

bool pr[350];

vector<int>prim;

int mx;

int fact[NN];

void sieve(int n) {

memset(pr,0,sizeof(pr));

long i,j,k,l;

pr[1]=1;

prim.push\_back(2);

for(i=4; i<=n; i+=2)

pr[i]=1;

for(i=3; i<=n; i+=2) {

if(pr[i]==0) {

prim.push\_back(i);

for(j=i\*i; j<=n; j+=2\*i)

pr[j]=1;

}

}

}

void factor(int n) {

int i,j,count;

for(j=0; j<prim.size() &&

prim[j]\*prim[j]<=n; j++) {

i=prim[j];

count=0;

if(n%i==0)

mx=max(i,mx);

while(n%i==0) {

n/=i;

count++;

}

fact[i]=max(fact[i],count);

if(n==1)

break;

}

if(n>1) {

mx=max(n,mx);

fact[n]=max(fact[n],1);

}

}

int bigmod(int m,int n) {

int sum;

if(n==0)

return 1;

if(n%2==0) {

sum=bigmod(m,n/2);

return ((sum%mod)\*(sum%mod))%mod;

}

else {

sum=bigmod(m,n-1);

return ((m%mod)\*(sum%mod))%mod;

}

}

int LCM(void) {

//LCM of elemets of arr with mod

long long sum=1;

int i,j,k;

mx=-inf;

mem(fact,0);

for(i=0; i<arr.size(); i++)

factor(arr[i]);

for(i=2; i<=mx; i++)

if(fact[i])

sum=(sum\*bigmod(i,fact[i]))%mod;

return sum;

}

main(){

int t,tc;

cin>>tc;

int cnt=0,sum=0;

int i,j,k,l,n,m;

sieve(345); //Sieve

while(tc--) {

cin>>n;

arr.clear();

for(i=1; i<=n; i++)

cin>>k,arr.pb(k);

sum=LCM();

printf("%d\n",sum);

}

return 0;

}

Input:

4

5

1 2 3 4 5

6

1 2 3 4 5 6

7

1 2 3 4 5 6 7

5

7 11 13 19 21

Output:

60

60

420

57057

**LCM of n numbers -String [without mod]**

using namespace std;

#define mem(a,b) memset(a,b,sizeof(a))

bool pr[106];

vector<int>prim;

int mx, fact[10001];

void sieve(int n) {

memset(pr,0,sizeof(pr));

long i,j,k,l;

pr[1]=1;

prim.push\_back(2);

for(i=4;i<=n;i+=2)

pr[i]=1;

for(i=3;i<=n;i+=2) {

if(pr[i]==0) {

prim.push\_back(i);

for(j=i\*i;j<=n;j+=2\*i)

pr[j]=1;

}

}

}

void factor(int n) {

int i,j,count;

for(j=0;j<prim.size() &&

prim[j]\*prim[j]<=n;j++) {

i=prim[j];

count=0;

if(n%i==0) mx=max(i,mx);

while(n%i==0) {

n/=i;

count++;

}

fact[i]=max(fact[i],count);

if(n==1)

break;

}

if(n>1) {

mx=max(n,mx);

fact[n]=max(fact[n],1);

}

}

string s;

void mult(int n,int r) {

while(r--) {

long long k,i,carry=0;

for(i=0;i<s.size();i++)

{

k=s[i]-'0';

k=(n\*k)+carry;

s[i]=k%10+'0';

carry=k/10;

}

while(carry>0)

{

s+=carry%10+'0';

carry/=10;

}

}

}

main()

{

sieve(101);

int n,k,i,m,c;

int tc,t=1;

cin>>tc;

while(tc--)

{

cin>>n;

mem(fact,0);

mx=-inf;

while(n--)

{

cin>>k;

factor(k);

}

s="1";

for(i=2;i<=mx;i++)

{

if(fact[i])

{

mult(i,fact[i]);

}

}

reverse(s.begin(),s.end());

printf("Case %d: %s\n"

,t++,s.c\_str());

}

return 0;

}

/\*

Input:

2

3

2 20 10

4

5 6 30 60

Output:

Case 1: 20

Case 2: 60

\*/

**Base Conversion**

void dec2other(char a[],char b[],int m) {

long long sum=atoi(a);

itoa(sum,b,m); //m-> required base.

}

void other2dec(char a[],char b[],int n) {

long long sum=0;

int i,j=0,k,l;

l=strlen(a);

j=0;

for(i=l-1;i>=0;i--) {

if(a[i]>='A') k=a[i]-'A'+10;

else k=a[i]-48;

sum+=k\*pow(n,j);

j++;

}

sprintf(b,"%lld",sum);

}

main() {

char a[100],b[100];

int i,j,k,l,m,n;

printf("CURRENT base: ");

scanf("%d",&n);

printf("\nNumber: ");

scanf("%s",&a);

printf("\nREQUIRED base: ");

scanf("%d",&m);

for(i=0;a[i]!=0;i++) {

a[i]=toupper(a[i]);

if(a[i]>='A') k=a[i]-'A'+10;

else k=a[i]-'0';

if(k>=n) {

printf("\n\*\*%s is not of %d base.\n\n",a,n);

return 0;

}

}

if(n==10) {

dec2other(a,b,m);

printf("\nNumber in %d base: %s\n\n",m,b);

}

else if(m==0) {

other2dec(a,b,n);

printf("\nNumber in %d base: %s\n\n",m,b);

}

else {

other2dec(a,b,n);

dec2other(b,a,m);

printf("\nNumber in %d base: %s\n\n",m,a);

}

return 0;

}

**BFS**

#define NIL -1

#define white 0

#define gray 1

#define black 2

using namespace std;

int dis[MAX], parent[MAX], color[MAX];

vector<int> g[MAX];

void BFS(int s,int v){

int len,x,k;

queue<int> Q;

parent[s]=NIL;

dis[s]=0;

color[s]=gray;

Q.push(s);

while(!Q.empty()){

x=Q.front(),Q.pop();

len=g[x].size();

for(int i=0; i<len; i++)

if(g[x][i] && color[g[x][i]]==white) {

k=g[x][i];

color[k]=gray;

dis[k]=dis[x]+1;

parent[k]=x;

Q.push(k);

}

color[x]=black;

}

printf("\n\*\*\*distances\*\*\*\n");

for(int i=1; i<=v; i++)

printf("distance[%d]= %d\n",i,dis[i]);

return;

}

int main() {

int v,e,s,d;

printf("Enter no of vertices: ");

scanf("%d",&v);

printf("Enter no of edges: ");

scanf("%d",&e);

for(int i=1; i<=e; i++) {

printf("Enter source and destination: ");

scanf("%d %d",&s,&d);

g[s].push\_back(d);

g[d].push\_back(s);

}

printf("Enter source of graph: ");

scanf("%d",&s);

BFS(s,v);

return 0;

}

**Dijkstra**

#include <bits/stdc++.h>

using namespace std;

typedef long long ll;

typedef unsigned long long llu;

#define ft first

#define sd second

#define mp make\_pair

#define pb(x) push\_back(x)

#define all(x) x.begin(),x.end()

#define allr(x) x.rbegin(),x.rend()

#define mem(a,b) memset(a,b,sizeof(a))

#define meminf(a) memset(a,126,sizeof(a))

#define inf 1e11

#define eps 1e-9

#define mod 1000000007

#define NN 30100

#define mx 100002

vector<int>g[mx],cost[mx];

struct node

{

int u,w;

node(int a,int b)

{

u=a;

w=b;

}

bool operator < ( const node& p ) const

{

return w > p.w;

}

};

int d[mx],par[mx];

int dijkstra(int n)

{

memset(d,63,sizeof(d)); //huge value=63

memset(par,-1,sizeof(par));

priority\_queue<node>q;

q.push(node(1,0));

d[1]=0;

while(!q.empty())

{

node top=q.top();

q.pop();

int u=top.u;

if(u==n)

return d[n];

for(int i=0; i<(int)g[u].size(); i++)

{

int v=g[u][i];

if(d[u]+cost[u][i]<d[v])

{

d[v]=d[u]+cost[u][i];

par[v]=u;

q.push(node(v,d[v]));

}

}

}

return -1;

}

int main()

{

int n,e;

cin>>n>>e;

for(int i=0; i<e; i++)

{

int u,v;

int w;

cin>>u>>v>>w;

g[u].push\_back(v);

g[v].push\_back(u);

cost[u].push\_back(w);

cost[v].push\_back(w);

}

int ret=dijkstra(n);

if(ret==-1) puts("No path!");

else

{

int u=n;

vector<int>out;

while(u!=-1)

{

out.push\_back(u);

u=par[u];

}

reverse(out.begin(),out.end());

for(int i=0; i<(int)out.size(); i++)

cout<<out[i]<<" ";

puts("");

}

}

**Floyd–Warshall**

#define inf 100000000

#define NN 300

int a[NN+7][NN+7];

int next[NN+7][NN+7];

int main()

{

int i,j,k,l,n,r,c,u,v,w,tc,t=1,m;

scanf("%d%d",&n,&r); //r = edges, n=nodes

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

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

a[i][j]=d[i][j]=inf;

next[i][j]=j;

}

a[i][i]=d[i][i]=0;

}

while(r--) {

scanf("%d%d%d",&u,&v,&k);

a[u][v]=a[v][u]=k;

}

for(k=1; k<=n; k++)

for(i=1; i<=n; i++)

for(j=1; j<=n; j++)

if(a[i][j]>a[i][k]+a[k][j]) {

a[i][j]=a[i][k]+a[k][j];

next[i][j]=k;

}

int first,last;

while(scanf("%d %d",&first,&last)==2) {

printf("From %d to %d :\n",first,last);

printf("Path: ");

i=first;

j=last;

printf("%d-->",i);

while(i!=j) {

i=next[i][j];

if(i==j) {

printf("%d",j);

break;

}

printf("%d-->",i);

}

printf("\nTotal cost : %d\n\n",mat[first][last]);

}

return 0;

}

**Articulation Bridge**

#define mp make\_pair

#define pb(x) push\_back(x)

#define all(x) x.begin(),x.end()

#define mem(a,b) memset(a,b,sizeof(a))

#define inf 1e9

#define eps 1e-9

#define NN 10010

vector<int>e[NN];

vector< pair<int,int> >bridge;

int depth[NN];

int par[NN];

int low[NN];

bool color[NN];

int Time;

int dfs(int u)

{

low[u]=depth[u]=++Time;

color[u]=true;

int i;

for(i=0; i<e[u].size(); i++)

{

int v=e[u][i];

if(!color[v])

{

par[v]=u;

dfs(v);

low[u]=min(low[u],low[v]);

if(depth[u]<low[v])

bridge.pb(mp(u,v));

}

else if(v!=par[u])

low[u]=min(low[u],depth[v]);

}

return 0;

}

int articulation\_Point(int n)

{

mem(depth,0);

mem(par,-1);

mem(low,0);

mem(color,0);

Time=0;

bridge.clear();

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

if(!color[i])

dfs(i);

int ans=bridge.size();

printf("%d critical links\n",ans);

for(int i=0; i<bridge.size(); i++)

printf("%d - %d\n",bridge[i].first,bridge[i].second);

return 0;

}

main()

{

ios\_base::sync\_with\_stdio(false);

int t=1,tc;

cin>>tc; //Test Case

int i,j,k,l,m,n;

int node,edge;

while(tc--)

{

cin>>node>>edge;

for(i=0; i<edge; i++)

{

cin>>k>>l;

e[k].pb(l);

e[l].pb(k);

}

printf("Case %d:\n",t++);

articulation\_Point(node);

for(i=0; i<=node; i++)

e[i].clear();

}

return 0;

}

Input

3

8 6

0 1

1 2

1 3

2 3

3 4

6 7

4 4

0 1

1 2

2 3

3 1

2 1

0 1

Output

Case 1:

3 critical links

3 - 4

0 - 1

6 - 7

Case 2:

1 critical links

0 - 1

Case 3:

1 critical links

0 – 1

**Articulation Point**

#define mp make\_pair

#define pb(x) push\_back(x)

#define all(x) x.begin(),x.end()

#define mem(a,b) memset(a,b,sizeof(a))

#define inf 1e9

#define eps 1e-9

#define NN 10010

vector<int>e[NN];

int depth[NN];

int par[NN];

int low[NN];

bool color[NN],Flag[NN];

int Time;

int dfs(int u)

{

low[u]=depth[u]=++Time;

color[u]=true;

int i,call=0;

for(i=0; i<e[u].size(); i++)

{

int v=e[u][i];

if(!color[v])

{

call++;

par[v]=u;

dfs(v);

low[u]=min(low[u],low[v]);

if(depth[u]<=low[v])

Flag[u]=true;

}

else if(v!=par[u])

low[u]=min(low[u],depth[v]);

}

if(par[u]==-1)

Flag[u]=(call>1);

}

int articulation\_Point(int n)

{

mem(depth,0);

mem(par,-1);

mem(low,0);

mem(color,0);

mem(Flag,0);

Time=0;

for(int i=1; i<=n; i++)

if(!color[i])

dfs(i);

int ans=0;

for(int i=1; i<=n; i++)

if(Flag[i])

ans++;

return ans;

}

main()

{

ios\_base::sync\_with\_stdio(false);

int t=1,tc;

cin>>tc; //Test Case

int i,j,k,l,m,n;

int node,edge;

while(tc--)

{

cin>>node>>edge;

for(i=0; i<edge; i++)

{

cin>>k>>l;

e[k].pb(l);

e[l].pb(k);

}

int ans=articulation\_Point(node);

printf("Case %d: %d\n",t++,ans);

for(i=0; i<=node; i++)

e[i].clear();

}

return 0;

}

Output

Case 1: 2

Case 2: 0

Case 3: 2

Input

3 //Test Case

5 4 //node edge

2 1

1 3

5 4

4 1

3 3

1 2

2 3

1 3

5 5

1 2

2 3

3 4

2 5

5 3

**SCC - DFS (Strongly Connected Component)**

int color[NN];

vector<int>arr; //topological sorted node

vector<int>Graph[NN], transGraph[NN],newGraph[NN

vector<pair<int,int> >v; //Edges Before SCC

int id[NN], amount[NN]; //Amount of original node in a SCC node

int dfs\_1st(int u) {

color[u]=true;

for(int i=0; i<Graph[u].size(); i++) {

if(!color[Graph[u][i]])

dfs\_1st(Graph[u][i]);

}

arr.pb(u);

}

int dfs\_2nd(int u,int k) {

color[u]=true;

id[u]=k;

for(int i=0; i<transGraph[u].size(); i++) {

if(!color[transGraph[u][i]])

dfs\_2nd(transGraph[u][i],k);

}

}

int scc(int n) {

arr.clear();

mem(color,0);

int i,j,k,l;

for(i=1; i<=n; i++) //Topological Sort

if(color[i]==0)

dfs\_1st(i);

reverse(all(arr));

mem(id,-1);

mem(color,0);

k=0;

for(i=0; i<arr.size(); i++) //Identify SCC {

if(!color[arr[i]]) {

dfs\_2nd(arr[i],k+1);

amount[id[arr[i]]]=1; //Amount of actual node

//in SCC node

k++;

}

else

amount[id[arr[i]]]++;

}

int node=k; //Number of SCC node

for(i=0; i<v.size(); i++) //Build SCC graph

{

k=v[i].first;

l=v[i].second;

if(id[k]!=id[l])

newGraph[id[k]].pb(id[l]);

}

return node; //Number of SCC node.

}

main()

{

int t=1,tc,i,j,k,l,m,n,man;

cin>>tc; //Test Case

while(tc--)

{

cin>>n>>m; //n=node, m=edge

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

Graph[i].clear(), transGraph[i].clear(), newGraph[i].clear();

v.clear();

for(i=0; i<m; i++)

{

cin>>k>>l;

Graph[k].pb(l);

transGraph[l].pb(k);

v.pb(make\_pair(k,l));

}

int sum=scc(n);

printf("Case %d: %d\n",t++,sum);

}

return 0;

}

=======[ input ]=======

2

4 4

1 2

2 1

3 4

4 3

3 3

1 2

2 3

3 1

=======[ output ]=======

Case 1: 2

Case 2: 1

**SCC - Tarjan (Strongly Connected Component)**

#define mp make\_pair

#define pb(x) push\_back(x)

#define all(x) x.begin(),x.end()

#define mem(a,b) memset(a,b,sizeof(a))

#define NN 1050

#define MAX 1000000

bool Flag[MAX]; //If a node already belongs to a scc or not.

int depth[MAX]; //The time when a node is visited

int Lowlink[MAX]; //A node connected with lowest timed node [if scc exist]

bool color[MAX];

int belong[MAX]; //A node blongs to which SCC

vector<int> G[MAX]; //Graph Store

stack<int>mystack; //order of nodes r visited

int time,top,scc;

void tarjan(int u)

{

int v,i;

depth[u]=Lowlink[u]=++time;

color[u]=true;

mystack.push(u);

Flag[u]=true;

for(i=0; i<G[u].size(); i++)

{

v=G[u][i];

if(!color[v])

{

tarjan(v);

Lowlink[u]=min(Lowlink[u],Lowlink[v]);

}

else if(Flag[v])

Lowlink[u]=min(Lowlink[u],depth[v]);

}

if(Lowlink[u]==depth[u])

{

scc++;

do

{

v=mystack.top(),mystack.pop();

Flag[v]=false;

belong[v]=scc;

}

while(u!=v);

}

}

void findSCC(int n)

{

mystack=stack<int>();

scc=top=time=0;

mem(depth,-1);

mem(Flag,0);

mem(color,0);

mem(Lowlink,126);

for(int i=1; i<=n; i++)

if(!color[i])

tarjan(i);

}

int main()

{

int node,edge;

cin>>node>>edge;

for(int i=0; i<edge; i++)

{

int k,l;

cin>>k>>l;

G[k].pb(l);

}

findSCC(node);

cout<<scc;

return 0;

}

/\*

Input:

5 5

1 2

2 3

3 4

5 2

3 5

Output:

3

\*/

**BCC [Biconnected Component]**

/\*Undirected Graph.

One Biconnected component means a region where nodes

will Be connected after deleting exactly one edge.

\*/

vector<int>Graph[NN]; //Graph Before BCC

vector<int>newGraph[NN]; //Graph after BCC

vector< pair<int,int> >edge;//Input edges

stack<int>mystack; //order of nodes r visited

int depth[NN]; //The depth(time) when a node is visited

int par[NN]; //Parent of node

int low[NN]; //A node connected with lowest timed node [if bcc exist]

bool color[NN]; //Color if a node is visited or not

int belong[NN]; //A node blongs to which BCC

int Time,bcc;

int dfs(int u) {

low[u]=depth[u]=++Time;

color[u]=true;

mystack.push(u);

int i,v;

for(i=0; i<Graph[u].size(); i++) {

v=Graph[u][i];

if(!color[v]) {

par[v]=u;

dfs(v);

low[u]=min(low[u],low[v]);

}

else if(v!=par[u])

low[u]=min(low[u],depth[v]);

}

if(low[u]==depth[u]) {

bcc++;

do {

v=mystack.top();

mystack.pop();

belong[v]=bcc;

}

while(u!=v);

}

return 0;

}

int findbcc(int n) {

mem(depth,0);

mem(par,-1);

mem(low,0);

mem(color,0);

mystack=stack<int>();

Time=bcc=0;

for(int i=0; i<n; i++) //lowest node=0

if(!color[i])

dfs(i);

int Highest\_Node=bcc;

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

int u=belong[edge[i].first];

int v=belong[edge[i].second];

if(u!=v) {

newGraph[u].pb(v);

newGraph[v].pb(u);

}

}

return Highest\_Node;

}

/\*

Input:

2

4 4

0 1

1 2

2 3

3 1

6 6

0 1

1 2

1 3

3 4

4 5

1 4

Output:

Case 1:

1 : 2

2 : 1

Case 2:

1 : 3

2 : 3

3 : 4 1 2

4 : 3

\*/

int Print\_NewGraph(int n) {

int i,j;

for(i=1; i<=n; i++) { //lowest node=1

if(newGraph[i].size()) {

printf("%d :",i);

for(j=0; j<newGraph[i].size(); j++)

printf(" %d",newGraph[i][j]);

puts("");

}

}

return 0;

}

main() {

int t=1,tc,i,j,k,l,m,n,e;

cin>>tc; //Test Cas

while(tc--) {

cin>>n>>e;

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

cin>>k>>l;

Graph[k].pb(l);

Graph[l].pb(k);

edge.pb(mp(k,l));

}

printf("Case %d:\n",t++);

k=findbcc(n);

Print\_NewGraph(k);

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

Graph[i].clear(),newGraph[i].clear();

edge.clear();

}

return 0;

}

**Matrix Expo - Fibonacchi**

/\*

==========[ Theme ]==========

|1 1|^k \* |f(1)| = |f(k+1)|

|1 0| |f(0)| | f(k) |

here,

f(0)=aa;

f(1)=bb;

============[ END ]===========

\*/

ll M;

ll m[3][3];

void mult(ll a[3][3],ll b[3][3])

{

ll temp[3][3];

int i,j,k;

mem(temp,0);

for(i=0;i<2;i++)

for(j=0;j<2;j++)

for(k=0;k<2;k++)

temp[i][j]+=

a[i][k]\*b[k][j];

for(i=0;i<2;i++)

for(j=0;j<2;j++)

a[i][j]=temp[i][j]%M;

return;

}

void BigMat(ll a[3][3],int pos)

{

int i,j,k;

if(pos==1)

return;

if(pos%2==1)

{

BigMat(a,pos-1);

mult(a,m);

}

else

{

BigMat(a,pos/2);

mult(a,a);

}

return;

}

main()

{

int t=1,tc;

cin>>tc;

ll i,j,k,l,n;

ll aa,bb;

while(tc--)

{

cin>>aa>>bb>>n>>M;

if(n==0)

{

printf("Case %d: %lld\n"

,t++,aa);

continue;

}

if(n==1)

{

printf("Case %d: %lld\n",

t++,bb);

continue;

}

ll a[3][3];

a[0][0] = m[0][0]=1;

a[0][1] = m[0][1]=1;

a[1][0] = m[1][0]=1;

a[1][1] = m[1][1]=0;

if(M==1)

M=10;

else if(M==2)

M=100;

else if(M==3)

M=1000;

else if(M==4)

M=10000;

BigMat(a,n);

m[0][0]=bb;

m[1][0]=aa;

ll temp[3][3];

mem(temp,0);

for(i=0;i<2;i++)

for(j=0;j<1;j++)

for(k=0;k<2;k++)

temp[i][j]+=

a[i][k]\*m[k][j];

printf("Case %d: %lld\n"

,t++,temp[1][0]%M);

}

return 0;

}

**Matrix Expo - nth term of Function**

/\*

f(n) =a\*f(n-1)+b\*f(n-3)+c, if(n > 2)

=0 if(n <= 2)

f(n+1)= a\*f(n)+0\*f(n-1)+b\*f(n-2)+c

|a 0 b 1|^k \* |f(2)| |f(k+2)|

|1 0 0 0| |f(1)| = |f(k+1)|

|0 1 0 0| |f(0)| | f(k) |

|0 0 0 1| | c | | c |

here,

a = aa;

b = bb;

c = cc;

\*/

ll m[5][5];

void mult(ll a[5][5],ll b[5][5]) {

ll temp[5][5];

int i,j,k;

mem(temp,0);

for(i=0;i<4;i++)

for(j=0;j<4;j++)

for(k=0;k<4;k++)

temp[i][j]+=

a[i][k]\*b[k][j];

for(i=0;i<4;i++)

for(j=0;j<4;j++)

a[i][j]=temp[i][j]%mod;

return;

}

void BigMat(ll a[5][5],int pos)

{

int i,j,k;

if(pos==1)

return;

if(pos%2==1)

{

BigMat(a,pos-1);

mult(a,m);

}

else

{

BigMat(a,pos/2);

mult(a,a);

}

return;

}

main()

{

int t=1,tc;

cin>>tc;

ll i,j,k,l,n;

ll aa,bb,cc;

while(tc--)

{

cin>>n>>aa>>bb>>cc;

if(n<=2)

{

printf("Case %d: 0\n",t++);

continue;

}

ll a[5][5];

a[0][0] = m[0][0] = aa;

a[0][1] = m[0][1] = 0;

a[0][2] = m[0][2] = bb;

a[0][3] = m[0][3] = 1;

a[1][0] = m[1][0] = 1;

a[1][1] = m[1][1] = 0;

a[1][2] = m[1][2] = 0;

a[1][3] = m[1][3] = 0;

a[2][0] = m[2][0] = 0;

a[2][1] = m[2][1] = 1;

a[2][2] = m[2][2] = 0;

a[2][3] = m[2][3] = 0;

a[3][0] = m[3][0] = 0;

a[3][1] = m[3][1] = 0;

a[3][2] = m[3][2] = 0;

a[3][3] = m[3][3] = 1;

BigMat(a,n);

ll b[5][2];

b[0][0] = 0, b[1][0] = 0;

b[2][0] = 0, b[3][0] = cc;

ll temp[5][5];

mem(temp,0);

for(i=0;i<4;i++)

for(j=0;j<1;j++)

for(k=0;k<4;k++)

temp[i][j]+=

a[i][k]\*b[k][j];

printf("Case %d: %lld\n"

,t++,temp[2][0]%mod);

}

return 0;

}

void init(ll a[8][8])

{

ll i,j,k;

mem(a,0);

mem(m,0);

m[0][0]=a1, m[0][1=b1, m[0][5]=c1;

m[1][0]=1, m[2][1]=1;

m[3][2]=c2, m[3][3]=a2, m[3][4]=b2;

m[4][3]=1, m[5][4]=1;

for(i=0;i<6;i++)

for(j=0;j<6;j++)

a[i][j]=m[i][j];

}

main()

{

int t=1,tc;

cin>>tc;

ll i,j,k,l,n,r;

while(tc--)

{

cin>>a1>>b1>>c1;

cin>>a2>>b2>>c2;

cin>>f[0]>>f[1]>>f[2];

cin>>g[0]>>g[1]>>g[2];

cin>>M;

cin>>r;

printf("Case %d:\n",t++);

ll b[8][2],a[8][8],temp[8][2];

b[0][0]=f[2], b[1][0]=f[1],

b[2][0]=f[0], b[3][0]=g[2],

b[4][0]=g[1], b[5][0]=g[0];

while(r--)

{

cin>>n;

if(n<=2)

{

printf("%lld %lld\n"

,f[n]%M,g[n]%M);

continue;

}

init(a);

BigMat(a,n);

mem(temp,0);

for(i=0; i<6; i++)

for(j=0; j<1; j++)

for(k=0; k<6; k++)

temp[i][j]+=

a[i][k]\*b[k][j];

printf("%lld %lld\n"

,temp[2][0]%M,temp[5][0]%M);

}

}

return 0;

}

**Matrix Expo - Two Functions**

/\*

f(n+1)= a1\*f(n) + b1\*f(n-1)+ c1\*g(n-2)

g(n+1)= a2\*g(n) + b2\*g(n-1)+ c2\*f(n-2)

|a1 b1 0 0 0 c1 |^k |f(2)| |f(k+2)|

|1 0 0 0 0 0 | |f(1)| |f(k+1)|

|0 1 0 0 0 0 | |f(0)| | f(k) |

|0 0 c2 a2 b2 0 | \* |g(2)| = |g(k+2)|

|0 0 0 1 0 0 | |g(1)| |g(k+1)|

|0 0 0 0 1 0 | |g(0)| | g(k) |

\*/

ll M;

ll a1,b1,c1;

ll a2,b2,c2;

ll m[8][8];

ll g[4],f[4];

void mult(ll a[8][8],ll b[8][8])

{

ll temp[8][8];

int i,j,k;

mem(temp,0);

for(i=0; i<6; i++)

for(j=0; j<6; j++)

for(k=0; k<6; k++)

temp[i][j]+=

a[i][k]\*b[k][j];

for(i=0; i<6; i++)

for(j=0; j<6; j++)

a[i][j]=temp[i][j]%M;

return;

}

void BigMat(ll a[8][8],int pos)

{

int i,j,k;

if(pos==1)

return;

if(pos%2==1)

{

BigMat(a,pos-1);

mult(a,m);

}

else

{

BigMat(a,pos/2);

mult(a,a);

}

return;

}

**Segment Tree [Sum of a segment, update & query]**

/\*

=================[ Input and Operation ]=============================

1. 0 x y v - add v to all numbers in the range of x to y (inclusive).

2. 1 x y - Total sum in x,y

\*/

struct data

{

long long sum;

long long xtra;

}tree[300010];

void update(int node, int low, int high, int rlow, int rhigh, int value)

{

if(low>=rlow && high<=rhigh)

{

tree[node].sum += (high-low+1)\*value;

tree[node].xtra += value;

return;

}

int left = node\*2;

int right = left+1;

int mid = (low+high)/2;

if(rhigh <= mid)

update(left, low, mid, rlow, rhigh, value);

else if(rlow > mid)

update(right, mid+1, high, rlow, rhigh, value);

else

{

update(left, low, mid, rlow, mid, value);

update(right, mid+1, high, mid+1, rhigh, value);

}

tree[node].sum=tree[left].sum+tree[right].sum+tree[node].xtra\*(highlow+1);

}

long long query(int node,int low,int high,int rlow,int rhigh,long long carry)

{

if(low>=rlow && high<=rhigh)

{

return tree[node].sum + carry\*(high-low+1);

}

int left = node\*2;

int right = left + 1;

int mid = (low + high)/2;

long long p1=0, p2=0;

if(rhigh<=mid)

p1=query(left, low, mid, rlow, rhigh, carry+tree[node].xtra);

else if(rlow>mid)

p2=query(right, mid+1, high, rlow, rhigh, carry+tree[node].xtra);

else

{

p1=query(left, low, mid, rlow, mid, carry+tree[node].xtra);

p2=query(right, mid+1, high, mid+1, rhigh, carry+tree[node].xtra);

}

return p1+p2;

}

main()

{

ios\_base::sync\_with\_stdio(false);

int tc, t=1;

cin>>tc;

while(tc--)

{

int n, q;

cin>>n>>q;

printf("Case %d:\n", t++);

mem(tree, 0);

while(q--)

{

int i, j, k, l;

cin>>i;

if(i==0)

{

cin>>j>>k>>l;

update(1, 1, n, j+1, k+1, l);

}

else if(i==1)

{

cin>>j>>k;

long long ans=query(1, 1, n, j+1, k+1, 0);

printf("%lld\n", ans);

}

}

}

return 0;

Input:

2

10 5

0 0 9 10

1 1 6

0 3 7 2

0 4 5 1

1 5 5

20 3

0 10 12 1

1 11 12

1 19 19

Output:

Case 1:

60

13

Case 2:

2

0

}

**Segment Tree[Maximum sum of a segment,init & query]**

struct data

{

int totalsum, maxsum, leftmax, rightmax;

data(int k)

{

totalsum = maxsum = leftmax = rightmax = k;

}

data()

{

}

}arr[NN];

int a[65010];

data merge(data a, data b)

{

data ret;

ret.totalsum = (a.totalsum + b.totalsum);

ret.maxsum = max(max(a.maxsum, b.maxsum), a.rightmax + b.leftmax);

ret.leftmax = max(a.leftmax, a.totalsum + b.leftmax);

ret.rightmax = max(b.rightmax, b.totalsum + a.rightmax);

return ret;

}

void init(int node, int low, int high)

{

if(low==high)

{

arr[node] = data(a[low]);

return;

}

int left = node\*2;

int right = left + 1;

int mid = (low + high)/2;

init(left, low, mid);

init(right, mid + 1, high);

arr[node] = merge(arr[left], arr[right]);

return;

}

data query(int node, int low, int high, int rlow, int rhigh)

{

if(low>=rlow && high<=rhigh)

return arr[node];

int left = node\*2;

int right = left + 1;

int mid = (low + high)/2;

if(rhigh<=mid)

return query(left, low, mid, rlow, rhigh);

else if(rlow>mid)

return query(right, mid + 1, high, rlow, rhigh);

else

{

data L = query(left, low, mid, rlow, mid);

data R = query(right, mid + 1, high, mid + 1, rhigh);

return merge(L, R);

}

}

main()

{

ios\_base::sync\_with\_stdio(false);

int t, tc;

int i, j, k;

int res, u,w,p,n,x,y,z,m,q,r,v,zero;

//cin>>tc;

while(cin>>n)

{

for(i=1;i<=n;i++)

cin>>a[i];

init(1, 1, n);

cin>>k;

while(k--)

{

cin>>x>>y;

data l = query(1, 1, n, x, y);

printf("%d\n", l.maxsum);

}

}

return 0;

}

/\*

Input:

3

-1 2 3

1

1 2

Output:

2

\*/

**BIT (Binary Indexed Tree)**

\*\*\* 1 base indexing

(I) Point Update, Range Query:

Add v at point x : update(x,v)

Sum[a,b] = query(b) - query(a-1)

(II) Range Update, Point Query:

Add v at range [a,b] : update(a,v), update(b+1,-v)

Value at point x = query(X)

(III) Range Update, Range Query:

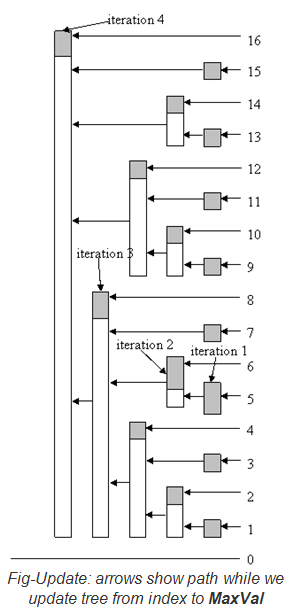
We have to use 2 BIT

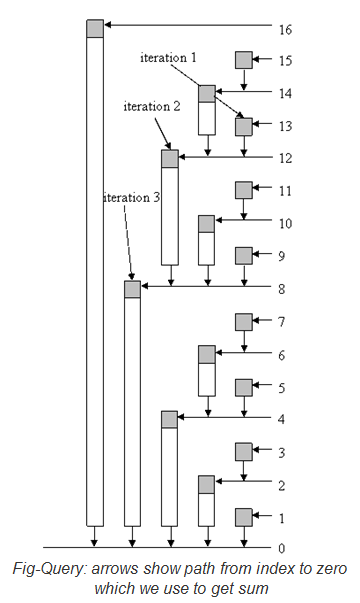
Add v at range [a,b] : update(a,v), update(b+1,-v),

update2(a,v\*(a-1)), update2(b+1,-v\*b)

Sum[0,x] = x\*query(x) - query2(x)

Sum[a,b] = Sum[0,b] - Sum[0,a-1]





**BIT-1 (Point Update, Range Query:)**

============[ Theme ]=============

j=1 => input k =>a[k]=0;

j=2 => input k,value=>a[k]+=value

j=3 => input k,l =>

output=a[k]+a[k+1]+....+a[l]

=============[ END ]==============

ll MaxVal;

ll tree[NN];

ll arr[NN];

ll update(ll idx,ll val)

{

while(idx<=MaxVal)

{

tree[idx]+=val;

idx+=idx & (-idx);

}

return 0;

}

ll query(ll idx)

{

ll sum=0;

while(idx>0)

{

sum+=tree[idx];

idx-=idx & (-idx);

}

return sum;

}

main() {

int t=1,tc;

scanf("%d",&tc);

ll i,j,k,l,m,n;

while(tc--)

{

scanf("%lld %lld",&n,&m);

mem(tree,0), MaxVal=n;

for(i=1;i<=n;i++)

{

scanf("%lld",&arr[i]);

update(i,arr[i]);

}

printf("Case %d:\n",t++);

while(m--) {

scanf("%lld",&j);

if(j==1) {

scanf("%lld",&k);

k++;

update(k,-arr[k]);

printf("%lld\n",arr[k]);

arr[k]=0;

}

else if(j==2) {

scanf("%lld %lld",&k,&l);

k++;

update(k,l);

arr[k]+=l;

}

else {

scanf("%lld %lld",&k,&l);

k++,l++;

ll temp=query(l);

temp-=query(k-1);

printf("%lld\n",temp);

}

}

}

return 0;

}

main()

{

int t=1,tc;

cin>>tc;

int i,j,k,l,n;

int x1,x2,y1,y2;

int r;

while(tc--)

{

printf("Case %d:\n",t++);

cin>>n;

mem(flag,0);

mem(tree,0);

max\_x=max\_y=1001;

while(n--)

{

cin>>j;

if(j==0)

{

cin>>x1>>y1;

x1++,y1++;

if(flag[x1][y1]==0)

{

flag[x1][y1]=1;

update(x1,y1,1);

}

}

else

{

cin>>x1>>y1>>x2>>y2;

x1++,y1++,x2++,y2++;

int temp=query(x2,y2);

temp-=query(x2,y1-1);

temp-=query(x1-1,y2);

temp+=query(x1-1,y1-1);

printf("%d\n",temp);

}

}

}

return 0;

}

**BIT - 2D (Points in rectangle)**

/\*

1)=> 0 x y, Add a point in x,y

if a point (x, y) is already

listed, then skip this query.

2)=> 1 x1 y1 x2 y2, Total point

in (x1,y1) to (x2,y2) rectangle

\*/

int tree[NN][NN];

bool flag[NN][NN];

int max\_x,max\_y;

void update(int idx,int idy,int val)

{

int y;

while(idx<=max\_x)

{

y=idy;

while(y<=max\_y)

{

tree[idx][y]+=val;

y+=y & -y;

}

idx+=idx & -idx;

}

return;

}

int query(int idx,int idy)

{

int sum=0,y;

while(idx>0)

{

y=idy;

while(y>0)

{

sum+=tree[idx][y];

y-=y & -y;

}

idx-=idx & -idx;

}

return sum;

}

**LCA (Lowest Common Ancestor)**

ll par[NN]; //Parent

ll level[NN]; //level in tree

bool color[NN]; //DFS color

ll P[NN][20]; //Sparse table

vector<ll>g[NN]; //Gragh store

//1 based index.

void dfs(ll u)

{

ll i,v;

color[u]=1;

for(i=0;i<g[u].size();i++)

{

v=g[u][i];

if(color[v]==0)

{

par[v]=u;

level[v]=level[u]+1;

dfs(v);

}

}

return;

}

ll lca\_query(ll p,ll q)

{

if(level[p]<level[q])

swap(p,q);

ll i,j,k,log;

log=1;

while(1)

{

ll next=log+1;

if(1<<next >level[p])

break;

log++;

}

for(i=log;i>=0;i--)

if(level[p]-(1<<i) >= level[q])

p=P[p][i];

if(p==q)

return p;

for(i=log;i>=0;i--)

if(P[p][i]!=-1 && P[p][i]!=P[q][i])

p=P[p][i],q=P[q][i];

return par[p];

}

void lca\_init(ll n)

{

mem(color,0);

mem(P,-1);

level[1]=0;

dfs(1);

ll i,j;

for(i=1;i<=n;i++)

P[i][0]=par[i];

for(j=1;1<<j <= n;j++)

for(i=1;i<=n;i++)

if(P[i][j-1]!=-1)

P[i][j]=P[P[i][j-1]][j-1];

return;

}

main()

{

int t=1,tc;

scanf("%d",&tc);

ll i,j,k,l,n,r;

while(tc--)

{

scanf("%lld",&n);

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

g[i].clear();

for(i=0;i<n-1;i++)

{

scanf("%lld %lld",&k,&l);

g[k].pb(l);

g[l].pb(k);

}

lca\_init(n);

cin>>r;

while(r--)

{

cin>>k>>l;

cout<<lca\_query(k,l)<<"\n";

}

puts("");

}

return 0;

}

void lca\_init(ll n)

{

mem(color,0), mem(P,-1);

level[1]=0, dist[1]=0;

dfs(1);

ll i,j;

for(i=1; i<=n; i++)

P[i][0]=par[i];

for(j=1; 1<<j <= n; j++)

for(i=1; i<=n; i++)

if(P[i][j-1]!=-1)

P[i][j]=P[P[i][j-1]][j-1];

return;

}

ll DIST(ll p,ll q)

{

ll temp=dist[p]+dist[q];

ll node=lca\_query(p,q);

temp-=2\*dist[node];

return temp;

}

ll KTH(ll p,ll q,ll k)

{

k--;

int i,j;

ll node=lca\_query(p,q);

ll temp=level[p]-level[node];

if(temp>=k)

{

ll LVL=level[p]-k;

for(j=20; j>=0; j--)

if(level[p]-(1<<j)>=LVL)

p=P[p][j];

return p;

}

k-=temp;

temp=level[q]-level[node];

temp-=k;

ll LVL=level[q]-temp;

for(j=15; j>=0; j--)

if(level[q]-(1<<j)>=LVL)

q=P[q][j];

return q;

}

**LCA (Lowest Common Ancestor)**

/\*

DIST a b : ask for the distance between node a and node b

KTH a b k : ask for the k-th node on the path from node a to node b

\*/

ll par[NN],level[NN],dist[NN];

bool color[NN]; //DFS color

ll P[NN][20]; //Sparse table

vector<ll>g[NN],cost[NN];

void dfs(ll u)

{

ll i,v;

color[u]=1;

for(i=0; i<g[u].size(); i++)

{

v=g[u][i];

if(color[v]==0)

{

par[v]=u;

level[v]=level[u]+1;

dist[v]=dist[u]+cost[u][i];

dfs(v);

}

}

return;

}

ll lca\_query(ll p,ll q)

{

if(level[p]<level[q]) swap(p,q);

ll i,j,k,log;

log=1;

while(1)

{

ll next=log+1;

if(1<<next >level[p]) break;

log++;

}

for(i=log; i>=0; i--)

if(level[p]-(1<<i) >= level[q])

p=P[p][i];

if(p==q)

return p;

for(i=log; i>=0; i--)

if(P[p][i]!=-1&&P[p][i]!=P[q][i])

p=P[p][i],q=P[q][i];

return par[p];

}

char s[10];

main() {

int t=1,tc;

scanf("%d",&tc);

ll i,j,k,l,n,r;

while(tc--) {

scanf("%lld",&n);

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

g[i].clear();

cost[i].clear();

}

for(i=0;i<n-1;i++) {

scanf("%lld %lld %lld",&k,&l,&r);

g[k].pb(l);

g[l].pb(k);

cost[k].pb(r);

cost[l].pb(r);

}

lca\_init(n);

while(1) {

scanf("%s",s);

if(strcmp(s,"DONE")==0)

break;

if(strcmp(s,"DIST")==0) {

scanf("%lld %lld",&k,&l);

ll ret=DIST(k,l);

printf("%lld\n",ret);

}

else {

scanf("%lld %lld %lld",&k,&l,&r);

ll ret=KTH(k,l,r);

printf("%lld\n",ret);

}

}

}

return 0;

}

input:

1

6

1 2 1

2 4 1

2 5 2

1 3 1

3 6 2

DIST 4 6

KTH 4 6 4

DONE

Output:

5

3

**Trie**

struct node

{

bool endmark;

node \*next[26+1];

node()

{

endmark=false;

for(int i=0; i<26; i++)

next[i]=NULL;

}

}\*root;

void insert(char \*str,int len)

{

node \*curr=root;

for(int i=0; i<len; i++)

{

int id=str[i]-'a';

if(curr->next[id]==NULL)

curr->next[id]=new node();

curr=curr->next[id];

}

curr->endmark=true;

}

bool search(char \*str,int len)

{

node \*curr=root;

for(int i=0; i<len; i++)

{

int id=str[i]-'a';

if(curr->next[id]==NULL)

return false;

curr=curr->next[id];

}

return curr->endmark;

}

void del(node \*cur)

{

for(int i=0; i<26; i++)

if(cur->next[i])

del(cur->next[i]) ;

delete(cur) ;

}

int main()

{

puts("ENTER NUMBER OF WORDS");

root=new node();

int num\_word;

cin>>num\_word;

for(int i=1; i<=num\_word; i++)

{

char str[50];

scanf("%s",str);

insert(str,strlen(str));

}

puts("ENTER NUMBER OF QUERY";);

int query;

cin>>query;

for(int i=1; i<=query; i++)

{

char str[50];

scanf("%s",str);

if(search(str,strlen(str)))

puts("FOUND");

else

puts("NOT FOUND");

}

del(root); //destroy trie;

return 0;

}