/\* Articulation Point \*/

const int maxn=10001;

using namespace std;

bool artpoint[maxn] ;

int predfn,n,m;

vector<int>g[maxn];

int dis[maxn];

int back[maxn];

void dfs(int u,int p){

int child = 0 ;

predfn++;

dis[u]=back[u]=predfn;

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

int v = g[u][i] ;

if(!dis[v]){

child++ ;

dfs(v,u) ;

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

if(back[v]>=dis[u]) artpoint[u] = 1 ;

}

else if(v!=p) back[u] =min(back[u],dis[v]) ;

}

if(p==-1&&child<=1) artpoint[u] = 0 ; //this line contains vagal

}

void reset(){

memset(artpoint,0,sizeof(artpoint)) ;

memset(dis,0,sizeof(dis)) ;

for(int i=1;i<=n;i++) g[i].clear() ;

predfn = 0;

}

int main() {

// freopen("//home//mazhar//Desktop//in.txt","r",stdin) ;

int tc,ct=1,u,v;

scanf("%d",&tc) ;

while(tc--){

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

reset() ;

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

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

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

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

}

dfs(1,-1);

int res=0 ;

for(int i=1;i<=n;i++) res+=artpoint[i] ;

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

}

return 0;

}

const int MAXN=210;

#define INF 0xffffff

inline int cube(int x){ return x\*x\*x ; }

using namespace std;

struct edges{

int to,cost;

};

vector<edges>g[MAXN] ;

int main() {

int kase,caseno,n,busyness[MAXN],e;

int dis[MAXN],rdis[MAXN];

scanf("%d",&caseno) ;

for(kase=1;kase<=caseno;kase++){

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

scanf("%d",&n) ;

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

scanf("%d",&busyness[i]) ;

dis[i]=INF ;

rdis[i] = INF ;

g[i].clear() ;

}

dis[1] = rdis[1] = 0 ;

scanf("%d",&e) ;

int u,v;

edges EDG ;

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

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

EDG.to = v ;

EDG.cost= cube( busyness[v]-busyness[u] ) ;

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

}

//Bellman-Ford

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

for(int u=1;u<=n;u++){

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

//relaxing

int v=g[u][j].to ;

int cost = g[u][j].cost ;

if(dis[u]!=INF && dis[u]+cost<dis[v]) {

rdis[v] = dis[v]=dis[u]+cost ;

}

}

}

}

//checking for -ve cycle

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

for(int u=1;u<=n;u++){

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

//relaxing

int v=g[u][j].to ;

int cost = g[u][j].cost ;

if(rdis[u]!=INF && rdis[v]>rdis[u]+cost) {

rdis[v] = rdis[u]+cost ;

}

}

}

}

int q ;

scanf("%d",&q) ;

while(q--){

scanf("%d",&u) ;

if(rdis[u]!=dis[u] || dis[u]==INF ||dis[u]<3){

printf("?\n") ;

}

else {

printf("%d\n",dis[u]) ;

}

}

}

return 0;

}

/\* From topcoder \*/

int read(int idx){

/\* this will return single value of the idx-th index \*/

int sum = 0 ;

while(idx>0){

sum+=tree[idx] ;

idx -= (idx &(-idx)) ;

}

return sum ;

}

void update(int idx,int val){

while(idx<=n){

tree[idx]+=val ;

idx += (idx &(-idx)) ; ;

}

}

int readSingle(int idx){

int sum = tree[idx];

if (idx > 0){

int z = idx - (idx & -idx);

idx--;

while (idx != z){

sum -= tree[idx];

idx -= (idx & -idx);

}

}

return sum;

}

/\* BIT -masking From http://www.shafaetsplanet.com

=======================================================================================================================\*/

int Set(int N,int pos){return N=N | (1<<pos);} /\* this will set the bit of the number N at pos to 1 \*/

int reset(int N,int pos){return N= N & ~(1<<pos);} /\* this will set the bit of the number N at pos to 0 \*/

bool check(int N,int pos){return (bool)(N & (1<<pos));} /\* this will return the bit of the number N at pos \*/

#include <bits/stdc++.h>

#define MAXN 1007

using namespace std ;

int n,m ;

vector<int>to[MAXN];

int dfsn[MAXN],low[MAXN],tm;

vector<pair<int,int> >ans;

void dfs(int u,int p){

dfsn[u]=low[u]=++tm;

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

int v=to[u][i];

if(!dfsn[v]){

dfs(v,u);

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

if(low[v]>=dfsn[v])

ans.push\_back(make\_pair(u,v));

}else if(v!=p) low[u]=min(low[u],dfsn[v]);

}

}

int main(){

int u,v ;

while(cin>>n>>m){

if(!n && !m) break ;

for(int i=0 ;i<n;i++) to[i].clear();

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

cin>>u>>v ;

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

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

}

tm=0 ;

ans.clear() ;

memset(dfsn,0,sizeof(dfsn)) ;

memset(low,0,sizeof(low)) ;

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

if(!dfsn[i]) dfs(i,-1);

}

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

if(ans[i].first>ans[i].second)

swap(ans[i].first,ans[i].second);

sort(ans.begin(),ans.end());

printf("%d",int(ans.size()));

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

printf(" %d %d",ans[i].first,ans[i].second);

cout<<"\n" ;

}

return 0;

}

#define MAX 100009

#define LL long long

#define sq(x) ((x)\*(x))

struct point {

LL x, y;

} P[MAX], C[MAX], P0;

inline LL TriArea2(point a, point b, point c) {

return (a.x\*(b.y-c.y) + b.x\*(c.y-a.y) + c.x\*(a.y-b.y));

}

inline LL sqDist(point a, point b) {

return (sq(a.x-b.x) + sq(a.y-b.y));

}

bool comp(point a, point b) {

LL d = TriArea2(P0, a, b);

if(d<0) return false;

if(!d && sqDist(P0, b) > sqDist(P0, a)) return false;

return true;

}

void ConvexHull(int np, int &nc) {

int i, j, pos = 0;

for(i=1; i<np; i++){

if(P[i].y<P[pos].y || (P[i].y==P[pos].y && P[i].x>P[pos].x)) pos = i;

}

swap(P[0], P[pos]);

P0 = P[0];

sort(&P[1], P+np, comp);

C[0] = P[0], C[1] = P[1], C[2] = P[2];

for(i=j=3; i<np; i++) {

while(TriArea2(C[j-2], C[j-1], P[i]) <= 0) j--;

C[j++] = P[i];

}

nc = j;

}

/\* Dijkstra-from https://sites.google.com/site/smilitude

=======================================================================================================================\*/

vector<int> edge[100], cost[100];

const int infinity = 1000000000;

edge[i][j] = jth node connected with i

cost[i][j] = cost of that edge

struct data

{

int city, dist;

bool operator < ( const data& p ) const

{

return dist > p.dist ; // if is return true true than the swap is occured

}

};

int dijkstra(int source, int destination) {

int d[100];

for(int i=0; i<100; i++) d[i] = infinity;

priority\_queue<data> q;

data u, v;

u.city = source, u.dist = 0;

q.push( u );

d[ source ] = 0;

while( !q.empty() ) {

u = q.top(); q.pop();

int ucost = d[ u.city ];

int edge\_count = edge[u.city].size() ;

for(int i=0; i<edge\_count; i++) {

v.city = edge[u.city][i] ; v.dist = cost[u.city][i] + ucost;

// relaxing :)

if( d[v.city] > v.dist ) {

d[v.city] = v.dist;

q.push( v );

}

}

}

return d[ destination ];

}

typedef pair<int,int>pii ;

#define ff first

#define ss second

pii extendedEuclid(int a,int b){

if(b==0) return pii(1,0) ;

else{

pii d = extendedEuclid(b,a%b) ;

return pii(d.ss , d.ff-d.ss\*(a/b)) ;

}

}

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

FOR(k,0,n-1){

FOR(i,0,n-1){

if(dis[i][k]!=-1)

{

FOR(j,0,n-1){

if(dis[j][k]!=-1) dis[i][k] = max(dis[i][j],dis[i][k],dis[k][j]) ;

}

}

}

}

int GCD(int a,int b){

while(b>0){

a = a%b ;

a ^=b ; b^=a ; a^=b ;

}

return a ;

}

int GCD(int a,int b){

return b==0?a:GCD(b,a%b) ;

}

void computeLPSArray(char \*pat,int M,int \*lps){

int len = 0 ;

int i = 1 ;

lps[0] = 0 ;

while(i<M){

if(pat[i] == pat[len]){

len++ ;

lps[i] = len ;

i++ ;

}

else{

if(len!=0) len = lps[len-1] ;

else{

lps[i] = 0 ;

i++ ;

}

}

}

}

void KMPSearch(char \*pat,char \*txt){

int M = strlen(pat) ;

int N = strlen(txt) ;

int \*lps = (int \*)malloc(sizeof9(int)\*M) ;

int j = 0 ,i=0 ; //the two pointers

computeLPSArray(pat,M,lps) ;

while(i<N){

if(pat[j] == text[i]) { i++ ; j++ ; }

if(j==M) { printf("MATCH FOUND\n") ; j=lps[j-1] ; }

else if(pat[j]!=txt[i]){

if(j==0) i++ ;

else j = lps[j-1] ;

// Do not match lps[0..lps[j-1]] characters,

// they will match anyway

}

}

}

//LCA using sparse table

//Complexity: O(NlgN,lgN)

#define mx 100002

int L[mx]; //লেভেল

int P[mx][22]; //স্পার্স টেবিল

int T[mx]; //প্যারেন্ট

vector<int>g[mx];

void dfs(int from,int u,int dep)

{

T[u]=from;

L[u]=dep;

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

{

int v=g[u][i];

if(v==from) continue;

dfs(u,v,dep+1);

}

}

int lca\_query(int N, int p, int q) //N=নোড সংখ্যা

{

int tmp, log, i;

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

tmp = p, p = q, q = tmp;

log=1;

while(1) {

int next=log+1;

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

log++;

}

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

if (L[p] - (1 << i) >= L[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 T[p];

}

void lca\_init(int N)

{

memset (P,-1,sizeof(P)); //শুরুতে সবগুলো ঘরে -১ থাকবে

int i, j;

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

P[i][0] = T[i];

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

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

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

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

}

int main(void) {

g[0].pb(1);

g[0].pb(2);

g[2].pb(3);

g[2].pb(4);

dfs(0, 0, 0);

lca\_init(5);

printf( "%d\n", lca\_query(5,3,4) );

return 0;

}

int LisSequence[32]; // for storing the sequence

int n; // the number of items in the sequence

int Sequence[32]; // the sequence of integers

int L[32]; // L[] as described in the algorithm

void takeInput() {

scanf("%d", &n); // how many numbers in the sequence ?

// take the sequence

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

scanf("%d", &Sequence[i]);

}

int LIS() { // which runs the LIS algorithm and returns the result

int i, j; // auxilary variables for iteration

// initialize L[] with 1

for( i = 0; i < n; i++ ) L[i] = 1;

// start from the left most item and itetare right

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

// for the ith item item find all items that are in right

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

if( Sequence[j] > Sequence[i] ) {

// the item is greater than the ith item

// so, L[j] = L[i] + 1, since jth item can be added after ith

// item. if L[j] is already greater than or equal to L[i] + 1

// then ignore it

if( L[j] < L[i] + 1 )

L[j] = L[i] + 1;

}

}

}

// now find the item whose L[] value is maximum

int maxLength = 0;

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

if( maxLength < L[i] )

maxLength = L[i];

}

// return the result

return maxLength;

}

int main() {

takeInput();

int result = LIS();

printf("The LIS length is %d\n", result);

return 0;

}

void findSequence( int maxLength ) { // finds a valid sequence

int i, j; // variable used for iteration

// at first find the position of the item whose L[] is maximum

i = 0;

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

if( L[j] > L[i] )

i = j;

}

// initialize the position in LisSequence where the items can be added.

// observe that the data are saving from right to left!

int top = L[i] - 1;

// insert the item in i-th position to LisSequence

LisSequence[top] = Sequence[i];

top--; // is decreasing such that a new item can be added in a new place

// now find the other valid numbers to form the sequence

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

if( Sequence[j] < Sequence[i] && L[j] == L[i] - 1 ) {

// we have found a valid item so, we will save it

i = j; // as in our algorithm

LisSequence[top] = Sequence[i]; // stored

top--; // decreased for new items

}

}

// so, we have got the sequence, now we want to print it

printf("LIS is");

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

printf(" %d", LisSequence[i]);

}

puts("");

}

int main() {

takeInput();

int result = LIS();

printf("The LIS length is %d\n", result);

findSequence( result );

return 0;

}

//========================================================Maximum Bipartite Matching (From igor) =================================//

//still n oac :(

// define M and N to be the maximum sizes of the left and right set respectively

bool graph[M][N];

bool seen[N];

int matchL[M], matchR[N];

int m, n;

bool bpm( int u ) {

for( int v = 0; v < n; v++ ) if( graph[u][v] ) {

if( seen[v] ) continue;

seen[v] = true;

if( matchR[v] < 0 || bpm( matchR[v] ) ) {

matchL[u] = v;

matchR[v] = u;

return true;

}

}

return false;

}

int main() {

// Read input and populate graph[][]

// Set m to be the size of L, n to be the size of RCPSC 490

Graph Theory: Network Flows and Matching

memset( matchL, ­1, sizeof( matchL ) );

memset( matchR, ­1, sizeof( matchR ) );

int cnt = 0;

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

{

memset( seen, 0, sizeof( seen ) );

if( bpm( i ) ) cnt++;

}

// cnt contains the size of the matching

// matchL[i] is what left vertex i is matched to (or ­1 if unmatched)

// matchR[j] is what right vertex j is matched to (or ­1 if unmatched)

return 0;

}

int fordFulkerson(int n,int s,int t){

memset(fnet,0,sizeof(fnet)) ; //initialization of the flow

int flow = 0 ;

while(true){

//find an argumenting path

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

prev[s] = -2 ;

queue<int>q ;

q.push(s) ;

while(!q.empty() && prev[t]==-1){

int u = q.front() ;

//cout<<u<<"\n" ;

q.pop() ;

FOR(v,1,n){

//int v = adj[u][i] ;

if(prev[v]==-1){ //never seen before

if(fnet[v][u] || fnet[u][v]<cap[u][v]){ //taking backedge over forword edge

prev[v]= u ;

q.push(v) ;

}

}

}

}

if(prev[t]==-1) break ; //no argumenting path found

int bot = INT\_MAX ;

for(int v=t,u=prev[t];u>=0;v=u,u=prev[v]){ //tracing the argumenting path

if(fnet[v][u]){

bot = min(fnet[v][u],bot) ;

}

else{

bot = min(cap[u][v]-fnet[u][v],bot) ;

}

}

for(int v=t,u=prev[t];u>=0;v=u,u=prev[v]){ //tracing the argumenting path and updating the flow

if(fnet[v][u]) fnet[v][u] -= bot ;

else fnet[u][v]+=bot ;

}

flow+=bot ;

}

return flow ;

}

/\* Millar-Rabin primality test iteration signifies the accuracy of the test \*/

bool Millar(LL p,LL iteration){

if(p<2 || p%2==0) return 0 ;

LL s = p-1 ;

while(s%2) s /=2 ;

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

LL a = rand()%(p-1)+1 ,temp = s ;

LL mod = BigMod(a,s,p) ;

while(mod!=1 && mod!=p-1 && temp!=p-1){

mod = mulmod(mod,mod,p) ;

temp \*=2 ;

}

if(mod!=p-1 && temp%2==0) return false ;

}

return true ;

}

/\*

ax = 1 mod(n)

ax-1 = y\*n ;

ax-y\*n = 1 ;

so calling Extended Euclid with (a,n) will return (x,-y) from that we only need x :)

\*/

//Using Extended Euclid

int modularInverse(int a,int b){

pii ret = extendedEuclid(a,n) ;

return (ret.ff%n + n )%n ;

}

//using fermet works only when n is prime ( most of the time it is )

// a^(p-1) = 1 mod p

//so a^(p-2) \* a = 1 modp ;

// we only need to calculate a^(p-2) using bigMod

int modularInverse(int a,int b){

return bigMod(a,p-2) ;

}

/\* MST\_Kruskal - From http://www.shafaetsplanet.com

========================================================================================================================\*/

struct edge{

int u,v,w;

bool operator < ( const edge& p ) const

{

return w < p.w;

}

};

int par[MAXN];

vector<edge>e;

int find(int r){

return (par[r]==r) ? r: find(par[r]);

}

int mst(int n){

sort(e.begin(),e.end());

for(int i=1;i<=n;i++)par[i]=i;

int count=0,s=0;

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

int u=find(e[i].u);

int v=find(e[i].v);

if(u!=v)

{

par[u]=v;

count++;

s+=e[i].w;

if(count==n-1) break;

}

}

return s;

}

//normal method

vector<int>primes ; // we will preload the primes

int PHI(int n){

int ret = n ;

int primeSize = primes.size() ;

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

if(n%primes[i]==0) ret -= ret/primes[i] ;

}

return ret ;

}

int PHI(int n){

int ret = n ;

for(int i=;i\*i<=n;i++){

if(n%i==0){

while(n%i==0){

n/=i ;

}

ret -=ret/i ;

}

}

return ret ;

}

//This is called when phi is called again and again

#define MAXN 1000005

int phi[MAXN] ;

void CalculatePhi(){

for(int i=1;i<MAXN;i++) phi[i] = i ;

for(int p=2;p<MAXN;p++){

if(Phi[p]==p){

for(int k=p;k<MAXN;k+=p) phi[k] -=phi[k]/p ;

}

}

}

#include <cstring>

#define MAX 46656 // sqrt(2^31-1)

#define RNG 100000+100 //(b-a<=10^5)

#define LMT 216 //????

#define sq ((x)\*(x))

#define Set(x,n) ( x[n>>6] |= (1<<((n>>1)&31)))

#define check (x,n) (x[n>>6] & (1<<((n>>1)&31))) ;

LL base[MAX/64] ,segment[RNG/64] ;

void sieve(){

LL i,j,k ;

FOR(i=3;i<LMT;i+=2){

if(!check(base,i){

}

}

}

struct info

{

i64 prop,sum;

}tree[mx\*3]; //sum ছাড়াও নিচে অতিরিক্ত কত যোগ হচ্ছে সেটা রাখবো prop এ

#define mx 100001

void init(int node,int b,int e)

{

if(b==e)

{

tree[node]=arr[b];

return;

}

int Left=node\*2;

int Right=node\*2+1;

int mid=(b+e)/2;

init(Left,b,mid);

init(Right,mid+1,e);

tree[node]=tree[Left]+tree[Right];

}

int query(int node,int b,int e,int i,int j,int carry=0)

{

if (i > e || j < b) return 0;

if(b>=i and e<=j) return tree[node].sum+carry\*(e-b+1); //সাম এর সাথে যোগ হবে সেই রেঞ্জের সাথে অতিরিক্ত যত যোগ করতে বলেছে সেটা

int Left=node<<1;

int Right=(node<<1)+1;

int mid=(b+e)>>1;

int p1 = query(Left, b,mid, i, j, carry+tree[node].prop); //প্রপাগেট ভ্যালু বয়ে নিয়ে যাচ্ছে carry ভ্যারিয়েবল

int p2 = query(Right, mid+1, e, i, j,carry+tree[node].prop);

return p1+p2;

}

void update(int node,int b,int e,int i,int j,i64 x)

{

if (i > e || j < b) return;

if (b >= i && e <= j) //নোডের রেঞ্জ আপডেটের রেঞ্জের ভিতরে

{

tree[node].sum+=((e-b+1)\*x); //নিচে নোড আছে e-b+1 টি, তাই e-b+1 বার x যোগ হবে এই রেঞ্জে

tree[node].prop+=x; //নিচের নোডগুলোর সাথে x যোগ হবে

return;

}

int Left=node\*2;

int Right=(node\*2)+1;

int mid=(b+e)/2;

update(Left,b,mid,i,j,x);

update(Right,mid+1,e,i,j,x);

tree[node].sum=tree[Left].sum+tree[Right].sum+(e-b+1)\*tree[node].prop;

//উপরে উঠার সময় পথের নোডগুলো আপডেট হবে

//বাম আর ডান পাশের সাম ছাড়াও যোগ হবে নিচে অতিরিক্ত যোগ হওয়া মান

}

/\* Strongly Connected Components(Tarjan) frome Topcoder:

=========================================================================================================================================\*/

int idx[maxn],tim,lowlink[maxn],scc[maxn],components;

stack<int>s ;

vector<int>adj[maxn] ;

void reset(){

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

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

for(int i=0;i<=maxn;i++) adj[i].clear() ;

while(!s.empty()) s.pop() ;

components = 0 ;

}

void dfs(int here,int par) {

idx[here] = lowlink[here] = tim++;

s.push(here);

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

int there = adj[here][i];

if(idx[there] == -1) {

dfs(there,here) ;

lowlink[here] = min(lowlink[here],lowlink[there] );

}

else if(scc[there]==-1){

/\* This line is really trciky .) is means The lowlink[here] is defined as the lowest-numbered vertex reachable from the subtree rooted at here via a single edge in the same component as here. See the difference with line 23 :) \*/

lowlink[here] = min(lowlink[here], idx[there]);

}

}

if(lowlink[here] == idx[here]) {

int elem ;

do{

elem = s.top() ;

s.pop() ;

scc[elem]=components ;

} while(elem!=here) ;

components++ ;

}

}

for each vertex u in V[G]

do color[u] ← WHITE

π[u] ← NIL ▷ π[] is parent array

time ← 0

for each vertex u in V[G]

do if color[u] ← WHITE

then DFS-Visit(u) ▷ build a new DFS-tree from u

DFS-Visit(u)

color[u] ← GRAY ▷ discover u

time ← time + 1 ▷ stopwatch

d[u] ← time

for each vertex v adjacent to u ▷ explore (u, v)

do if color[v] ← WHITE

then π[v] ← u ▷ saving parent to print path

DFS-Visit(v)

color[u] ← BLACK

f[u] ← time ▷ we are done with u

struct node{

bool endmark ;

node \*next[26+1] ;

node(){

endmrk = 0 ;

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 = 1 ;

}

}

bool search(char \*str,int len){

node \*cur = root ;

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

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

if(cur->next[id]==NULL) return 0 ;

cur = cur->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) ;

}