Codes Page

1. Euler Phi-------------------------------------1

2. Sieve------------------------------------------1

3. Divisor Count-------------------------------1

4. Bitwise sieve--------------------------------1

5. Histogram-----------------------------------1

6. BIT--------------------------------------------2

7. 2D BIT----------------------------------------2

8. TRIE-------------------------------------------2

9. KMP------------------------------------------3

10. Inclusion-Exclusion Principle---------4

11. MST-Kruskal------------------------------4

12. MST-Prim---------------------------------5

12. Coin Change------------------------------5

13. Segment Tree-----------------------------6

14. Lazy Propagation-----------------------6

15. LCA -----------------------------------------7

16. Suffix Array--------------------------------8

17. Hashing (Double Hash) ----------------9

18. Matrix Expo -----------------------------10

19. Lazy Propagation 2---------------------11

20. Topological Sort-----------------------11

21. Topological Sort using indegree----12

22. Sparse Table----------------------------12

23. BFS on 2D Grid------------------------12

24. DFS--------------------------------------13

25. Dijkstra---------------------------------14

26. Floyd-Warshall-------------------------15

**Euler Phi**

//The coprime number

//Order n

#define MAX 1000000

int E[MAX+10];

bool flag[MAX+10];

void phi(void){

int i,j;

for(i=1;i<=MAX;i++) E[i]=i;

for(i=2;i<=MAX;i++){

if(flag[i]==0){

for(j=i;j<=MAX;j+=i){

flag[j]=1;

E[j]/=i;

E[j]\*=(i-1);

}

}

}

//for(i=1;i<=20;i++)printf("%d %d\n",i,E[i]);

}

/////////////order (sqrt(n))

int phi (int n){

int result = n;

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

if (n % i == 0){

while (n % i == 0){

n /= i;

result -= result / i;

}

}

}

if(n>1)

result -= result / n;

return result;

}

**Sieve**

#define ll long long

//EFFICIENT CODE FOR SEARCHING PRIME

#define MAX 1000000+7

bool a[MAX+10];

ll pr[MAX],pi;

void siv()

{

ll i,j,k;

pr[pi++]=2;

for(i=4;i<=MAX;i+=2) a[i]=1;

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

if(a[i]==0)pr[pi++]=i;

if(i<=1000)//1000 is sq.root of MAX

if(a[i]==0){

for(j=i\*i;j<=MAX;j+=2\*i) a[j]=1;

}

}

//printf("%d",pi);//number of prime

//for(i=0;i<pi;i++)

//printf("%d ",pr[i]);//prm stored

}

**Divisor Count**

ll divcount(ll n){

ll divcnt=1;

for(ll i=0;pr[i]\*pr[i]<=n;i++){

ll cou=0;

if(n%pr[i]==0){

while(n%pr[i]==0){

n/=pr[i];

cou++;

}

}

divcnt\*=cou+1;

}

if(n>1) divcnt\*=2;

return divcnt;

}

**Bitwise Sieve**

//Prime up to 10^9

#define MAX 2000000000

#define LMT 44722

long long flag[MAX>>6];

#define ifc(n) (flag[n>>6]&(1<<((n>>1)&31)))

#define isc(n) (flag[n>>6]|=(1<<((n>>1)&31)))

//storing even prime only

void sieve() {

long long i, j, k;

for(i=3; i<LMT; i+=2)

if(!ifc(i))

for(j=i\*i, k=i<<1; j<MAX; j+=k)

isc(j);

}

///if(!ifc(i)) then i is a prime

**Histogram**

//Finding máximum area of rectangle

int main(){

int i,n,in,t,cs,j,mxar,nwar,cou,y;

scanf("%d",&t);

for(cs=1;cs<=t;cs++){

mxar=-1;

pair <int, int > a;

pair <int, int > b;

stack< pair <int, int> > st;

scanf("%d",&n);

scanf("%d",&in);

a= make\_pair(in,0);

st.push(a);

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

scanf("%d",&in);

a= st.top();

if(a.first<in){

a=make\_pair(in,i);

st.push(a);

}

else if(a.first>=in){

while(a.first>=in && !st.empty()){

b=make\_pair(in,a.second);

nwar=(i-a.second)\*a.first;

if(nwar>mxar) mxar=nwar;

st.pop();

if(!st.empty()) a=st.top();

}

st.push(b);

}

}

while(!st.empty()){

a=st.top();

nwar=(i-a.second)\*a.first;

if(nwar>mxar) mxar=nwar;

st.pop();

} printf("Case %d: %d\n",cs,mxar);

}

}

**BIT**

Time Comp: O(lgn(Maxval)

/\*Keeps cumulative sum as ascending order, Read=sum (1->n), update=adds value, memset tree with 0, 1 indexing, MaxVal – maximum value which will have non-zero frequency, bitMask=less than equal power of 2 in Maxval\*/

int tree[100009],MaxVal;

int read(int idx){

int sum = 0;

while (idx > 0){

sum += tree[idx];

idx -= (idx & -idx);

}

return sum;

}

void update(int idx ,int val){

while (idx <= MaxVal){

tree[idx] += val;

idx += (idx & -idx);

}

}

int readSingle(int idx){

int sum = tree[idx]; // sum will be decreased

if (idx > 0){ // special case

int z = idx - (idx & -idx); // make z first

idx--; /\* idx is no important any more, so instead y, you can use idx\*/

while (idx != z){ /\* at some iteration idx (y) will become z\*/

sum -= tree[idx];

/\*substruct tree frequency which is between y and the same path\*/

idx -= (idx & -idx);

}

}

return sum;

}

/ \*if in tree exists more than one index with a same cumulative frequency, this procedure will return the greatest one\*/

int findG(int cumFre){

int idx = 0;

int bitMask=1<<17;

while ((bitMask != 0) && (idx < MaxVal)){

int tIdx = idx + bitMask;

if (cumFre >= tree[tIdx]&&tIdx<=MaxVal){

idx = tIdx;

cumFre -= tree[tIdx];

}

bitMask >>= 1;

}

if (cumFre != 0)

return -1;

else

return idx;

}

**2D BIT**

void update(int x , int y , int val){

int y1;

while (x <= max\_x){

y1 = y;

while (y1 <= max\_y){

tree[x][y1] += val;

y1 += (y1 & -y1);

}

x += (x & -x);

}

}

int read(int x1,int y1){

int sum = 0;

int x=x1,y=y1;

while (x > 0){

y=y1;

while(y>0)

{

sum+=tree[x][y];

y-=(y & -y);

}

x-=(x & -x);

}

return sum;

}

**TRIE**

//Complexity: O(length)

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);

return 0;

}

**KMP**

/\* the prefix under index i in the table above is the string from pattern[0] to pattern[i - 1] inclusive, so the last character of the string under index i is pattern[i - 1] \*/

function build\_failure\_function(pattern[])

{

// let m be the length of the pattern

F[0] = F[1] = 0; // always true

for(i = 2; i <= m; i++) {

// j is the index of the largest next partial match (the largest suffix/prefix) of the string under index i – 1\*/

j = F[i - 1];

for( ; ; ) {

if(pattern[j] == pattern[i - 1]) {

F[i] = j + 1; break;

}

// if we cannot "expand" even the empty string

if(j == 0) { F[i] = 0; break; }

// else go to the next best "candidate" partial match

j = F[j];

}

}

}

function Knuth\_Morris\_Pratt(text[], pattern[])

{

// let n be the size of the text, m the size of the pattern, and F[] - the "failure function"\*/

build\_failure\_function(pattern[]);

i = 0; // the initial state of the automaton is

// the empty string

j = 0; // the first character of the text

for( ; ; ) {

if(j == n) break; // we reached the end of the text

// if the current character of the text "expands" the

// current match

if(text[j] == pattern[i]) {

i++; // change the state of the automaton

j++; // get the next character from the text

if(i == m) // match found

}

// if the current state is not zero (we have not

// reached the empty string yet) we try to

// "expand" the next best (largest) match

else if(i > 0) i = F[i];

// if we reached the empty string and failed to

// "expand" even it; we go to the next

// character from the text, the state of the

// automaton remains zero

else j++;

}

}

**INCLUSION-EXCLUSION**

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

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

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

}

ll sum=0;

for(bit=1;bit<(1<<m);bit++){

ll cnt=0,lcm=1;

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

mask=(1<<j);

if(bit&mask){

cnt++;

lcm=(lcm\*ar[j])/\_\_gcd(lcm,ar[j]);

}

}

if(cnt%2==0) sum-=(n/lcm);

else sum+=(n/lcm);

}

**Minimum Spanning Tree-Kruskal**

/\*Let P[x] = the parent of node x.

CREATE-SET(x)

P[x] = x

rank[x] = 0

Sort node. Connect if doesn't have connection,

Time Complexity: O(nlogn)\*/

//Problem: Lightoj-1029

struct triplet{

int u,v,w;

};

int P[150],rnk[150];

int P2[150],rnk2[150];

bool cmp(triplet a,triplet b){

if(a.w<b.w) return true;

return false;

}

int fnd(int u){

if(u!=P[u]) P[u]=fnd(P[u]);

return P[u];

}

int fnd2(int u){

if(u!=P2[u]) P2[u]=fnd2(P2[u]);

return P2[u];

}

bool MERGE\_SET(int u,int v){

int PX=fnd(u);

int PY=fnd(v);

if(PX==PY) return 0;

if(rnk[PX]>rnk[PY]) P[PY]=PX;

else P[PX]=PY;

if (rnk[PX] == rnk[PY]) rnk[PY] = rnk[PY] + 1;

return 1;

}

bool MERGE\_SET2(int u,int v){

int PX=fnd2(u);

int PY=fnd2(v);

if(PX==PY) return 0;

if(rnk2[PX]>rnk2[PY]) P2[PY]=PX;

else P2[PX]=PY;

if (rnk2[PX] == rnk2[PY]) rnk2[PY] = rnk2[PY] + 1;

return 1;

}

int main(){

int n,t,cs,u,v,w,i;

scanf("%d",&t);

for(cs=1;cs<=t;cs++){

vector<triplet> vc;

scanf("%d",&n);

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

P[i]=i;

P2[i]=i;

rnk[i]=0;

rnk2[i]=0;

}

triplet x;

while(scanf("%d%d%d",&u,&v,&w)==3)

{

if(u==0&&v==0&&w==0) break;

x.u=u;

x.v=v;

x.w=w;

vc.push\_back(x);

}

sort(vc.begin(),vc.end(),cmp);

int l=vc.size();

int cou=0;

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

x=vc[i];

bool nw=MERGE\_SET(x.u,x.v);

if(nw==1) cou+=x.w;

}

int cou2=0;

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

x=vc[i];

bool nw=MERGE\_SET2(x.u,x.v);

if(nw==1) cou2+=x.w;

}

int res=cou+cou2;

if(res%2==0){

printf("Case %d: %d\n",cs,res/2);

}

else printf("Case %d: %d/2\n",cs,res);

}

return 0;

}

**MST-Prims**

/\*Input: A non-empty connected weighted graph with vertices V and edges E (the weights can be negative).

\* Initialize: Vnew = {x}, where x is an arbitrary node (starting point) from V, Enew = {}

\* Repeat until Vnew = V:

o Choose an edge (u, v) with minimal weight such that u is in Vnew and v is not

(if there are multiple edges with the same weight, any of them may be picked)

o Add v to Vnew, and (u, v) to Enew

\* Output: Vnew and Enew describe a minimal spanning tree

\*/

struct triplet{

int u,v,w;

bool operator < (const triplet& t) const{

if(w != t.w) return w < t.w;

else return false;

}

};

int main(){

vector<int> edge[210];

vector<int> cost[210];

int vis[210],u,sz,dst,mx;

triplet inp;

int n,r,w,i,v,cnt,cs=1;

string a,b;

while(scanf("%d%d",&n,&r)==2){

if(n==0&&r==0) break;

map<string,int> mp;

memset(vis,0,sizeof(vis));

cnt=1;

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

cin>>a>>b>>w;

if(mp[a]==0) mp[a]=cnt++;

if(mp[b]==0) mp[b]=cnt++;

u=mp[a],v=mp[b];

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

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

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

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

}

cin>>a>>b;

priority\_queue<triplet> qu;

u=mp[a],dst=mp[b];

sz=edge[u].size();

mx=10009;

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

{

inp.u=u;

inp.v=edge[u][i];

inp.w=cost[u][i];

qu.push(inp);

//mx=max(mx,inp.w);

}

vis[u]=1;

while(!qu.empty())

{

inp=qu.top();

qu.pop();

u=inp.u,v=inp.v,w=inp.w;

if(vis[u]==0&&vis[v]==1){

vis[u]=1;

mx=min(mx,w);

if(dst==u||dst==v) break;

sz=edge[u].size();

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

{

inp.u=u;

inp.v=edge[u][i];

inp.w=cost[u][i];

qu.push(inp);

}

}

else if(vis[u]==1&&vis[v]==0){

mx=min(mx,w);

if(dst==u||dst==v) break;

vis[v]=1;

sz=edge[v].size();

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

{

inp.u=edge[v][i];

inp.v=v;

inp.w=cost[v][i];

qu.push(inp);

}

}

}

printf("Scenario #%d\n",cs++);

printf("%d tons\n",mx);

printf("\n",mx);

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

edge[i].clear();

cost[i].clear();

}

}

return 0;

}

**Coin Change**

LOJ-1232

//Iterative way of taking n coins

long long n,k;

int memo[10700];

int coin[150];

int main(){

int res,m,t,i,j,cs;

scanf("%d",&t);

for(cs=1;cs<=t;cs++){

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

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

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

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

memo[i]=0;

memo[0]=1;

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

for(j=coin[i];j<=k;j++){

memo[j]+=memo[j-coin[i]];

memo[j]%=mod;

}

}

res=memo[k]%mod;

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

}

return 0;

}

**Segment Tree**

/\*Can be used to find min, max, update a range, (Single update, Range query-log n),size of M= N\*4\*/

LOJ-1082

#define inf 100009

int A[100090],M[524300];

void build(int lo,int hi,int idx){

if(lo==hi) {M[idx]=A[lo];return;}

int mid=(lo+hi)/2;

build(lo,mid,2\*idx);

build(mid+1,hi,(2\*idx)+1);

M[idx]=min(M[2\*idx],M[(2\*idx)+1]);

}

//Update value of index i with J

void update(int lo, int hi,int i,int v,int idx){

if(i<lo || i>hi) return;

if(lo==hi) {M[idx]=v; return;}

int mid=(lo+hi)/2;

update(lo,mid,i,v,(2\*idx));

update(mid+1,hi,i,v,(2\*idx)+1);

M[idx]=min(M[2\*idx],M[2\*idx+1]);

}

//Search from I to J

int query(int lo, int hi,int i,int j,int idx){

if(j<lo || i>hi) return inf;

if(lo>=i && hi<=j) return M[idx];

int mid=(lo+hi)/2;

int a=query(lo,mid,i,j,2\*idx);

int b=query(mid+1,hi,i,j,(2\*idx)+1);

return min(a,b);

}

int main(){

int t,n,q,cs,i,j,x,y;

scanf("%d",&t);

for(cs=1;cs<=t;cs++) {

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

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

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

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

}

build(1,n,1);

//Start to end and 1

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

scanf("%d%d",&x,&y);

int res=query(1,n,x,y,1);

//Idx always 1

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

}

}

return 0;

}

**Lazy Propagation**

/\*Range update, Range Query, Array size N\*4, Memset st & flag with invalid vlaue\*/

int st[1048580],flag[1048580];

void build(int cur, int s, int e,int val) {

if(s == e) {

st[cur] = val;

flag[cur] = -1;

return;

}

int m = (s+e)>>1, c1 = cur << 1, c2 = c1 | 1;

build(c1, s, m, val);

build(c2, m+1, e, val);

st[cur] = val;

flag[cur] = -1;

}

void resolve(int cur, bool leaf) {

if(flag[cur] == -1) return;

if(!leaf) {

flag[cur<<1] = flag[cur];

flag[(cur<<1)|1] = flag[cur];

}

st[cur] = flag[cur];

flag[cur] = -1;

}

void update(int cur, int s, int e, int S, int E, int value) {

resolve(cur, s == e);

if(s > E || e < S) return;

if(s >= S && e <= E) {

flag[cur] = value;

resolve(cur, s == e);

return;

}

int m = (s+e)>>1, c1 = cur << 1, c2 = c1 | 1;

update(c1, s, m, S, E, value);

update(c2, m+1, e, S, E, value);

st[cur] = max(st[c1], st[c2]);

}

// Gives the minimum in the range [S, E]

int query(int cur, int s, int e, int S, int E) {

resolve(cur, s == e);

if(s > E || e < S) return 0;/\*it shd return an minimum value for maximaization and vice versa or an invalid value return 0;\*/

if(s >= S && e <= E) return st[cur];

int m = (s+e)>>1, c1 = cur << 1, c2 = c1 | 1;

return max( query(c1, s, m, S, E), query(c2, m+1, e, S, E) );

}

**LCA**

LOJ-1162

#define filer() freopen("in1.txt","r",stdin)

#define filew() freopen("out.txt","w",stdout)

#define SET(a, x) memset((a), (x), sizeof(a))

#define ll long long

#define INF 1<<29

#define MAX(a,b) ((a)>(b)?(a):(b))

#define MIN(a,b) ((a)<(b)?(a):(b))

#include<bits/stdc++.h>

using namespace std;

struct Node{

int end;

int weight;

};

vector<Node>adj[100009];

int parent[100009];

int min\_P[100009][18];

int max\_P[100009][18];

int P[100009][18];

int L[100009];

bool vst[100009];

void DFS(int S)

{

vst[S]=1;

int sz=adj[S].size();

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

if(!vst[adj[S][i].end]){

parent[adj[S][i].end]=S;

L[adj[S][i].end]=L[S]+1;

P[adj[S][i].end][0]=S;

min\_P[adj[S][i].end][0]=adj[S][i].weight;

max\_P[adj[S][i].end][0]=adj[S][i].weight;

DFS(adj[S][i].end);

}

}

}

///\*

void initialize(int N){

//for(int i=1;i<=N;i++)P[i][0]=parent[i];

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

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

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

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

min\_P[i][j]=MIN(min\_P[i][j-1],min\_P[P[i][j-1]][j-1]);

max\_P[i][j]=MAX(max\_P[i][j-1],max\_P[P[i][j-1]][j-1]);

}

}

}

}

struct result{

int mn,mx;

};

result query(int x,int y){

result ans;

int tmp,log;

if (L[x] < L[y]){

tmp = x;

x = y;

y = tmp;

}

ans.mx=0;

ans.mn=INF;

for (log = 1; 1 << log <= L[x]; log++);

log--;

int i;

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

if (L[x] - (1 << i) >= L[y]){

ans.mx=MAX(ans.mx,max\_P[x][i]);

ans.mn=MIN(ans.mn,min\_P[x][i]);

x = P[x][i];

}

}

if(x==y)return ans;

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

if (P[x][i] != -1 && P[x][i] != P[y][i]){

ans.mx=MAX(ans.mx,max\_P[x][i]);

ans.mn=MIN(ans.mn,min\_P[x][i]);

x = P[x][i];

ans.mx=MAX(ans.mx,max\_P[y][i]);

ans.mn=MIN(ans.mn,min\_P[y][i]);

y = P[y][i];

}

}

ans.mx=MAX(ans.mx,max\_P[x][0]);

ans.mx=MAX(ans.mx,max\_P[y][0]);

ans.mn=MIN(ans.mn,min\_P[x][0]);

ans.mn=MIN(ans.mn,min\_P[y][0]);

return ans;

}

//\*/

int main(){

//filer();

//filew();

int T,cas=0;

int i,j,N,x,y,w,q;

for(i=0;i<100009;i++){

parent[i]=-1;L[i]=-1;

for(j=0;j<18;j++){P[i][j]=-1;min\_P[i][j]=-1;max\_P[i][j]=-1;}

}

Node S;

scanf("%d",&T);

while(T--){

cas++;

scanf("%d",&N);

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

scanf("%d%d%d",&x,&y,&w);

S.end=y;

S.weight=w;

adj[x].push\_back(S);

//parent[y]=x;

S.end=x;

//S.weight=w;

adj[y].push\_back(S);

}

P[1][0]=-1;

L[1]=0;

DFS(1);

initialize(N);

/\*for(i=1;i<=N;i++)

{

for(j=0;1<<j<N;j++)printf("%d ",min\_P[i][j]);printf("\n");

}\*/

scanf("%d",&q);

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

while(q--){

scanf("%d%d",&x,&y);

if(x==y){printf("%d %d\n",0,0);continue;}

result ans=query(x,y);

printf("%d %d\n",ans.mn,ans.mx);

}

if(T){

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

parent[i]=-1;L[i]=-1;vst[i]=0;

for(j=0;(1<<j)<N;j++){P[i][j]=-1;min\_P[i][j]=INF;max\_P[i][j]=-1;}

adj[i].clear();

}

}

}

return 0;

}

**Suffix Array**

/\*bobocel=0516234(val)

N=length of String, memset val(done in for loop)

Maximum Length substring match-Lightoj\_1347\*/

#define MAXN 65536

#define MAXLG 17

#define ll long long

char A[MAXN],B[MAXN];

struct entry{

int nr[2],p;

}L[MAXN];

ll val[15515];

ll P[MAXLG][MAXN],N,i,stp,cnt;

ll cmp(struct entry a, struct entry b){

return a.nr[0]==b.nr[0] ? (a.nr[1]< b.nr[1] ? 1 : 0) : (a.nr[0] < b.nr[0] ? 1: 0);

}

ll lcp(ll x,ll y){

ll k,ret=0;

if(x==y) return N-x;

for(k=stp-1; k>=0 && x<N &&y<N; k--){

if(P[k][x]==P[k][y]){

x+= 1 << k, y+= 1 << k, ret+= 1 << k;

}

}

return ret;

}

int main(){

ll t,cs,j,l1,l2,l3;

scanf("%lld",&t);

getchar();

for(cs=1;cs<=t;cs++)

{

gets(A);

gets(A);

N=strlen(A);

l1=N;

gets(B);

l2=strlen(B);

A[N]='@';

for(i=N+1,j=0;j<l2;j++,i++)

{

A[i]=B[j];

}

l2=i;

A[i++]='#';

gets(B);

l3=strlen(B);

for(j=0;j<l3;j++,i++)

{

A[i]=B[j];

}

l3=i;

A[i++]='$';

N=i;

/\*Added three string above, if single string, continue from here\*/

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

if(A[i]=='@') P[0][i]=1;

else if(A[i]=='#') P[0][i]=2;

else if(A[i]=='$') P[0][i]=3;

else P[0][i]=A[i]-93;

val[i]=0;

}

for(stp=1,cnt=1; cnt >> 1<N; stp++, cnt<<=1)

{

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

{

L[i].nr[0]=P[stp-1][i];

L[i].nr[1]= i+cnt < N ? P[stp-1][i+cnt] : -1;

L[i].p=i;

}

sort(L,L+N, cmp);

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

P[stp][L[i].p]= i > 0 && L[i].nr[0]== L[i-1].nr[0]&&L[i].nr[1]==L[i-1].nr[1]? P[stp][L[i-1].p] : i;

}

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

{

val[P[stp-1][i]]=i;

}

/\*for(i=0;i<N;i++)

{

//printf("%lld %lld\n",i,val[i]);

for(j=val[i];j<N;j++)

{

printf("%c",A[j]);

}

printf("\n");

}\*/

ll p,q,lcpmatch,length;

long long mx=0;

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

{

lcpmatch=lcp(val[i],val[i-2]);

{

if(val[i]<l1)

{

if (val[i-1]<l2&&val[i-1]>l1&&val[i-2]>l2) mx=max(mx,lcpmatch);

else if (val[i-1]>l2&&val[i-2]<l2&&val[i-2]>l1) mx=max(mx,lcpmatch);

}

else if(val[i]<l2&&val[i]>l1)

{

if((val[i-1]<l1&&val[i-2]>l2)||(val[i-1]>l2&&val[i-2]<l1)) mx=max(mx,lcpmatch);

}

else if(val[i]>l2)

{

if((val[i-1]<l1&&val[i-2]<l2&&val[i-2]>l1)||(val[i-1]<l2&&val[i-1]>l1&&val[i-2]<l1)) mx=max(mx,lcpmatch);

}

}

}

printf("Case %lld: %lld\n",cs,mx);

}

return 0;

}

**Hashing (Double Hash)**

/\*pattern=P[len]\*b^0+P[len-1]\*b^1+...+P[0]\*b^len

Text=T[plen]\*b^0+T[plen-1]\*b^1+...+T[0]\*b^len

slide text=text-T[i-1]\*b^len+text\*b+T[plen+i]\*b^0

Substring frequency-LOJ-1255\*/

char txt[1000009];

char pattern[1000009];

#define ll long long

#define B 29

#define mod 1000000007

#define C 37

#define mod2 1000000009

int main()

{

ll t,cs,i,n,a,respat1,respat2,j,pw,pw2,lpat,restxt1,restxt2,cou,ltxt,val,val2;

scanf("%lld",&t);

getchar();

for(cs=1;cs<=t;cs++)

{

gets(txt);

gets(pattern);

lpat=strlen(pattern);

ltxt=strlen(txt);

respat1=0,respat2=0,pw=1,pw2=1;

for(i=lpat-1;i>=0;i--)

{

respat1=(respat1+(pw\*pattern[i])%mod)%mod;

respat2=(respat2+(pw2\*pattern[i])%mod2)%mod2;

while(respat1<0) respat1=(respat1+mod);

respat1=respat1%mod;

while(respat2<0) respat2=(respat2+mod2);

respat2=respat2%mod2;

pw=(pw\*B)%mod;

pw2=(pw2\*C)%mod2;

}//Created hash of pattern

restxt1=0,restxt2=0,pw=1,pw2=1;

for(i=lpat-1;i>=0;i--)

{

restxt1=(restxt1+(pw\*txt[i])%mod)%mod;

restxt2=(restxt2+(pw2\*txt[i])%mod2)%mod2;

while(restxt1<0) restxt1=(restxt1+mod);

restxt1=restxt1%mod;

while(restxt2<0) restxt2=(restxt2+mod2);

restxt2=restxt2%mod2;

if(i!=0) pw=(pw\*B)%mod;

if(i!=0) pw2=(pw2\*C)%mod2;

}//Created one hash of text

cou=0;

if(respat1==restxt1&&respat2==restxt2) cou++;

for(i=1;i<=(ltxt-lpat);i++)

{

val=(pw\*txt[i-1])%mod;

val2=(pw2\*txt[i-1])%mod2;

restxt1=(restxt1-val)%mod;

restxt2=(restxt2-val2)%mod2;

while(restxt1<0) restxt1=(restxt1+mod);

restxt1=restxt1%mod;

while(restxt2<0) restxt2=(restxt2+mod2);

restxt2=restxt2%mod2;

restxt1=((restxt1\*B)%mod+txt[i+lpat-1])%mod;

restxt2=((restxt2\*C)%mod2+txt[i+lpat-1])%mod2;

if(respat1==restxt1&&restxt2==respat2) cou++;

}//sliding through the text

printf("Case %lld: %lld\n",cs,cou);

}

return 0;

}

**Matrix Expo (LOJ-1096)**

#define filer() freopen("in.txt","r",stdin)

#define filew() freopen("out.txt","w",stdout)

#define SET(a, x) memset((a), (x), sizeof(a))

#define pb push\_back

#define i64 long long

#define INF 1<<29

#define PI pair<int,int>

#define SZ 4

using namespace std;

typedef vector<int> VI;

int MOD=10007;

//i64 INF=(i64)((i64)1<<(i64)59);

struct M{

int Mat[SZ][SZ];

M(){

SET(Mat,0);

}

M(int a,int b){

SET(Mat,0);

Mat[0][0]=a;

Mat[0][2]=b;

Mat[0][3]=1;

Mat[1][0]=1;

Mat[2][1]=1;

Mat[3][3]=1;

}

};

M MULTIPLY(M a,M b){

M c;

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

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

for(int k=0;k<SZ;k++){

c.Mat[i][j]+=(((a.Mat[i][k]%MOD)\*(b.Mat[k][j]%MOD))%MOD);

c.Mat[i][j]%=MOD;

}

}

}

return c;

}

M Mod(M a){

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

for(int j=0;j<SZ;j++)a.Mat[i][j]%=MOD;

}

return a;

}

M BigMod(M a,int n){

if(n==1)return a;

if(n%2){

return Mod(MULTIPLY(a,BigMod(a,n-1)));

}

M t=BigMod(a,n/2);

return Mod(MULTIPLY(t,t));

}

int main(){

//filer();

int T,cas=0,i,j,x,y,z,n,a,b,c;

scanf("%d",&T);

while(T--){

scanf("%d%d%d%d",&n,&a,&b,&c);

if(n<=2)

{

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

}

M X(a,b);

X=BigMod(X,n-2);

int ans=((X.Mat[0][3]%MOD)\*(c%MOD))%MOD;

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

}

return 0;

}

**Lazy Propagation-2**

**Horrible Queries -1164**

#define ll long long

ll M[700000],FLAG[700000];

void update(ll lo, ll hi,ll i,ll j,ll v,ll idx){

if(j<lo || i>hi) return;

if(lo>=i&&hi<=j) {

M[idx] += ((hi-lo+1)\*v);

FLAG[idx] +=v;

return;

}

ll mid=(lo+hi)/2;

update(lo,mid,i,j,v,(2\*idx));

update(mid+1,hi,i,j,v,(2\*idx)+1);

M[idx]=M[2\*idx]+M[2\*idx+1]+(hi-lo+1)\*FLAG[idx];

}

//Search from I to J

ll query(ll lo, ll hi,ll i,ll j,ll idx,ll carry){

if(j<lo || i>hi) return 0;

if(lo>=i && hi<=j) return (M[idx]+carry\*(hi-lo+1));

ll mid=(lo+hi)/2;

ll a=query(lo,mid,i,j,2\*idx,carry+FLAG[idx]);

ll b=query(mid+1,hi,i,j,(2\*idx)+1,carry+FLAG[idx]);

return a+b;

}

int main()

{

ll t,cs,n,i,j,q,in,x,y,v;

scanf("%lld",&t);

for(cs=1;cs<=t;cs++)

{

printf("Case %lld:\n",cs);

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

memset(M,0,sizeof(M));

memset(FLAG,0,sizeof(FLAG));

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

{

scanf("%lld",&in);

if(in==0){

scanf("%lld%lld%lld",&x,&y,&v);

update(1,n,x+1,y+1,v,1);

}

else{

scanf("%lld%lld",&x,&y);

ll res=query(1,n,x+1,y+1,1,0);

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

}

}

}

return 0;

}

**Topological Sort, LOJ-1003**

#define M 100009

int visited[M], cycle = 0;

vector<int> edge[M];

void visit(int p) {

if(visited[p] == 2 || cycle ) return;

// we won't process anything if we already have found a cycle

if( visited[p] == 1 ) {

cycle = 1; // found cycle

return;

}

int i,sz=edge[p].size();

visited[p] = 1;

for(i=0;i<sz;i++) visit(edge[p][i]);

visited[p] = 2;

}

int main()

{

int n,i,j,ln,t,cs,a,b;

string s1,s2;

scanf("%d",&t);

for(cs=1;cs<=t;cs++)

{

cycle=0;

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

edge[i].clear();

}

memset(visited,0,sizeof(visited));

map<string,int> mp;

scanf("%d",&ln);

int n=1;

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

{

cin>>s1>>s2;

if(mp[s1]==0) mp[s1]=n++;

if(mp[s2]==0) mp[s2]=n++;

b= mp[s2], a = mp[s1];

edge[b].push\_back(a);

}

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

if(cycle==0) printf("Case %d: Yes\n",cs);

else printf("Case %d: No\n",cs);

}

return 0;

}

**Topological Sort –Using Indegree Uva10305**

/\* Find indegree of all node. if one's indegree is zero, reduce indegree by 1 who has edge with this,

find again,whose indegree is zeor, don't take one node twice.\*/

int main(){

int n,m,indeg[110],u,v,i,j,sz,k,ispush[110];

vector<int> edge[110];

vector<int> res;

while(scanf("%d%d",&n,&m)==2){

if(!n && !m) break;

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

edge[i].clear();

indeg[i]=0;

ispush[i]=0;

}

res.clear();

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

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

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

indeg[v]++;

}

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

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

if(indeg[j]==0&&ispush[j]==0){

ispush[j]=1;

res.push\_back(j);

sz=edge[j].size();

for(k=0;k<sz;k++){

indeg[edge[j][k]]--;

}

}

}

}

sz=res.size();

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

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

if(i==sz-1) printf("\n");

else printf(" ");

}

}

return 0;

}

**Sparse Table**

/\*Range Minimum Query, PreCalculation (nlogn) Query O(1), table memory (nlogn)\*/

#define Max 10000005

int ST[24][Max];

int A[Max];

void Compute\_ST(int N){

for (int i=0;i<N;i++)ST[0][i] = i;

for (int k = 1;(1 << k)<N; k++){

for (int i=0;i+(1<<k)<=N;i++){

int x = ST[k-1][i];

int y = ST[k-1][i+(1<<k-1)];

ST[k][i]=A[x]<=A[y]?x:y;

}

}

}

int RMQ(int i, int j){//Query

int k = log2(j-i);

int x = ST[k][i];

int y = ST[k][j-(1<<k)+1];

return A[x] <= A[y] ? x : y;

}

int main(){

int N;

cin>>N;

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

{

cin>>A[i];

}

Compute\_ST(N);

int Q;

cin>>Q;

while(Q--){

int x,y;

cin>>x>>y;

cout<<A[RMQ(x,y)]<<endl;

}

return 0;

}

**BFS on 2D Grid**

**/**/UVA-10653

#define pii pair<int,int>

int ar[1001][1001],vis[1001][1001],R,C,srci,srcj,dsti,dstj,d[1001][1001];

#define inrange(x,y) (x>=0&&x<R&&y>=0&&y<C) ? true : false

int dx[]={0,-1,0,+1};

int dy[]={-1,0,+1,0};

int bfs(){

int vx,vy,i;

pii u,v;

queue<pii> qu;

vis[srci][srcj]=1;

qu.push(make\_pair(srci,srcj));

d[srci][srcj]=0;

while(!qu.empty()){

u=qu.front();

qu.pop();

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

vx=u.first+dx[i];

vy=u.second+dy[i];

int rng=inrange(vx,vy);

if(rng==1&&vis[vx][vy]==0&&ar[vx][vy]!=-1){

vis[vx][vy]=1;

d[vx][vy]=d[u.first][u.second]+1;

qu.push(make\_pair(vx,vy));

}

}

if(u.first==dsti&&u.second==dstj) return d[dsti][dstj];

}

return d[dsti][dstj];

}

int main(){s

int i,j,rn,r,n,in;

while(scanf("%d%d",&R,&C)==2){

if(R==0&&C==0) break;

memset(ar,0,sizeof(ar));

memset(vis,0,sizeof(vis));

scanf("%d",&rn);

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

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

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

scanf("%d",&in);

ar[r][in]=-1;

}

}

scanf("%d%d",&srci,&srcj);

scanf("%d%d",&dsti,&dstj);

int ans=bfs();

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

}

return 0;

}

**DFS**

//UVA 11504

//O(V+E)

/\*DFS (V, E)

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

time ← time + 1 ▷ stopwatch

f[u] ← time ▷ we are done with u\*/

#define MX 100009

vector<int> edge[MX];

int n,color[MX],d[MX],f[MX],p[MX],tim,ar[MX],cou;

void DFS\_visit(int u);

void DFS(){

int i,u,v;

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

color[i]=0;

d[i]==f[i]=p[i]=0;

}

cou=0;

tim=0;

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

if(color[ar[i]]==0){

DFS\_visit(ar[i]);

cou++;

}

}

}

void DFS\_visit(int u){

int i,v,sz;

color[u]=1;

d[u]=++tim;

sz=edge[u].size();

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

v=edge[u][i];

if(color[v]==0){

p[v]=u;

DFS\_visit(v);

}

}

f[u]=++tim;

color[u]=2;

}

int main(){

vector< pair<int,int> > vc;

int t,cs,i,j,E,u,v;

scanf("%d",&t);

for(cs=1;cs<=t;cs++){

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

vc.clear();

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

ar[i]=i;

edge[i].clear();

}

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

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

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

}

DFS();

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

vc.push\_back(make\_pair(f[i],i));

}

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

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

ar[j]=vc[i].second;

}

DFS();

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

}

return 0;

}

**Diajkstra**

//O(Vlog(v+e))

/\*Source to node minimum distance with edge cost, Edge relaxation+pop the element of minimum distance

procedure dijkstra(G, source):

Q = priority\_queue(), distance[] = infinity

Q.enqueue(source)

distance[source]=0

while Q is not empty

u = nodes in Q with minimum distance[]

remove u from the Q

for all edges from u to v in G.adjacentEdges(v) do

if distance[u] + cost[u][v] < distance[v]

distance[v] = distance[u] + cost[u][v]

Q.enqueue(v)

end if

end for

end while

Return distance\*/

ll n,m;

vector<ll> edge[100009],cost[100009];

ll dist[100009],par[100009];

struct doublet{

ll v,w;

bool operator < (const doublet& t) const {

if(w != t.w) return w < t.w;

else return false;

}

};

void dijkstra(ll source,ll dst){

ll sz,i;

priority\_queue< doublet> qu;

doublet d1,d2;

d1.v=source;

d1.w=0;

dist[source]=0;

par[source]=1;

qu.push(d1);

while(!qu.empty()){

d1=qu.top();

qu.pop();

sz=edge[d1.v].size();

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

d2.v=edge[d1.v][i];

d2.w=cost[d1.v][i];

if(dist[d1.v]+d2.w<dist[d2.v]){

dist[d2.v]=dist[d1.v]+d2.w;

qu.push(d2);

par[d2.v]=d1.v;

}

}

}

}

int main(){

ll i,u,v,w;

while(scanf("%lld%lld",&n,&m)==2){

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

edge[i].clear();

cost[i].clear();

dist[i]=inf;

par[i]=0;

}

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

scanf("%lld%lld%lld",&u,&v,&w);

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

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

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

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

}

dijkstra(1,n);

if(par[n]==0){

printf("-1\n");

continue;

}

vector<ll> vc;

vc.push\_back(n);

i=par[n];

while(i!=1){

vc.push\_back(i);

i=par[i];

}

vc.push\_back(1);

ll sz=vc.size();

for(i=sz-1;i>=0;i--)

{

printf("%lld",vc[i]);

if(i==0) printf("\n");

else printf(" ");

}

}

return 0;

}

**Floyd-Warshall**

//All pair shortest path

Time-O(n3) spaceO(n2)

/\*initialize path[i][j]=j for all(i,j)

ar[i][j]=inf if i doesn’t have path with j\*/

#define inf 100000000

int ar[1000][1000],tx[1009],path[1000][1000];

void floyd(int n){

int i,j,k;

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

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

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

if(ar[i][j]>(ar[i][k]+ar[k][j]+tx[k])){

ar[i][j]=ar[i][k]+ar[k][j]+tx[k];

path[i][j]=path[i][k];

}

}

}

}

}

void findpath(int i,int j){

printf("Path: %d",i);

while(i!=j)

{

i=path[i][j];

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

}

printf("\n");

}

int main(){

int t,m,cs,i,j,n,flag=0,in1,in2,res;

char line[1000];

scanf("%d",&t);

getchar();

gets(line);

for(cs=1;cs<=t;cs++){

gets(line);

stringstream ss(line);

int num; vector<int> v;

while(ss>>num) v.push\_back(num);

n=v.size();

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

{

ar[1][j]=v[j-1];

if(ar[1][j]==-1) ar[1][j]=inf;

path[1][j]=j;

}

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

{

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

{

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

if(ar[i][j]==-1) ar[i][j]=inf;

path[i][j]=j;

}

}

for(i=1;i<=n;i++) scanf("%d",&tx[i]);

getchar();

floyd(n);

while(gets(line))

{

if(strlen(line)==0) break;

if(flag==1) printf("\n");

flag=1;

stringstream ss(line);

v.clear();

while(ss>>num) v.push\_back(num);

in1=v[0],in2=v[1];

printf("From %d to %d :\n",in1,in2);

findpath(in1,in2);

res=ar[in1][in2];

printf("Total cost : %d\n",res);

}

}

return 0;

}