```
ios_base::sync_with_stdio(0);

// ---------- FLOYD WARSHALL
for (int k = 0;k<n;k++)
 for (int i = 0;i<n;i++)   //Floyd
  for (int j = 0;j<n;j++)
   adj[i][j] = min(adj[i][j], adj[i][k] + adj[k][j]);

// ---------- Kruskal
int par(int x){
      if (parent[x]==x) return x;
            return parent[x]=par(parent[x]); // Path Compression
}
void merge(int x,int y){
            parent[par(x)]=par(y); //Path Compression version
}
void kruskal(){
  for(i=0;i<m;i++){
    if(par(e[i].x)!= par(e[i].y)) {merge(e[i].x,e[i].y);cost+=e[i].l;t++;} //Don't forget to sort edges
  }
}

// ---------- SIEVE NIKHIL
const int maxn = (int)1e7 + 10;
int isComposite[maxn >> 6];
int factor[maxn];
int primes[665000];
int P;
int A,B;
inline bool check(int i)
{
            return ( isComposite[i >> 5] & ( 1 << ( i & 31) ) );
}
inline void fix(int i)
{
            isComposite[ i >> 5] |= ( 1 << ( i & 31) );
}
void sieve()
{
            int i,j, k;
            for(i = 6; i <= 3160; i += 6) //sqrt maxn
            {
                        for(k = i -1; k <= i + 1; k += 2)
                                    if(!check(k >> 1) )
                                                for(j = k * k; j < maxn; j += k << 1)
                                                            fix( j >> 1);
            }
            P = 1;
            primes[P++] = 2;
            primes[P++] = 3;
            for(i = 6; i < maxn ;  i += 6)
            {
                        for(j = i -1; j <= i + 1; j +=2)
                                    if(!check(j >> 1 ) )
                                                primes[P++] = j;
            }
            primes[0] = primes[P++] = 1 << 30;
}

// ---------- SIEVE
vector<unsigned long> get_primes(unsigned long max){
    vector<unsigned long> primes;
    char *sieve;
    sieve = new char[max/8+1];
    memset(sieve, 0xFF, (max/8+1) * sizeof(char));   // fill with  1
    for(unsigned long x = 2; x <= max; x++)
        if(sieve[x/8] & (0x01 << (x % 8))){
            primes.push_back(x);
            // Is prime. Mark multiplicates.
```

```
            for(unsigned long j = 2*x; j <= max; j += x)
                sieve[j/8] &= ~(0x01 << (j % 8));
        }
    delete[] sieve;
    return primes;
}

// ---------- BIT -- initialize 0 -- 1 indexed
const int max_n = 100000;
int BIT[max_n];
void update(int incr, int idx)     // increments "array value" at position idx by "incr"
{
        int pos = idx;
        while(pos < max_n)
        {
                BIT[pos] += incr;
                pos += pos & (-pos);
        }
}
int query(int idx)                                              // finds "prefix sum" of elements from 1 to idx.
{
        int pos = idx;
        int sum = 0;
        while(pos > 0)
        {
                sum += BIT[pos];
                pos -= pos & (-pos);
        }
        return sum;
}

// ---------- SEG TREE
const int max_n = 100000;                //10^5
int val[max_n];                                  //point values
int beg[4*max_n], end[4*max_n], seg[4*max_n];
void make_seg(int pos, int start, int finish)
{
        beg[pos] = start;
        end[pos] = finish;
        seg[pos] = 0;
        if(start == finish)
                return;
        int mid = (start + end)/2;
        make_seg(2*pos, start, mid);
        make_seg(2*pos+1, mid+1, finish);
}
int query(int pos, int start, int finish)
{
        if(start <= beg[pos] && finish >= end[pos])
                return seg[pos];
        if(finish <= end[left])
                return query(left, start, finish);
        if(start >= beg[right])
                return query(right, start, finish);
        int leftans = query(left, start, finish);
        int rightans = query(right, start, finish);
        return leftans + rightans;
}
void update(int pos, int idx, int num)           //update the value at idx (DIFFERENT FROM POS).
Recursively: currently at node pos
{
        if(beg[pos] == end[pos])                          // Here it would be equal to idx
        {
                seg[pos] = num;                                         // modify value here
                return;
        }

        if(idx >= beg[2*pos + 1])
                update(2*pos + 1, idx, num);
```

```
	else
		update(2*pos, idx, num);
	seg[pos] = seg[2*pos] + seg[2*pos+1];
}
int main()
{
	make_seg(1, 0, max_n-1);
}

// ---------- SEG TREE W/ LAZY PROPOGATION
int n,q,bs=1<<17,seg[1<<18][3],flag[1<<18];//bs=base size
void init() //Initialise seg tree
{
	int i;
	for(i=bs;i<bs+n;i++)
	{
		seg[i][0]=1;//Initial Values
	}
	for(i=bs-1;i>0;i--)
	{
		seg[i][0]=seg[i<<1][0]+seg[i<<1|1][0];//Change it to required function like
max,gcd,etc
	}
}
void prop(int node,int L,int R) //Lazy Propagation
{
	int temp;
	if(flag[node]==1)
	{
		//Updating Value at node
		temp=seg[node][0];
		seg[node][0]=seg[node][2];
		seg[node][2]=seg[node][1];
		seg[node][1]=temp;
		//Flag children
		if(L!=R)
		{
			flag[node<<1]++;flag[node<<1]%=3;
			flag[node<<1|1]++;flag[node<<1|1]%=3;
		}
	}
	else if(flag[node]==2)
	{
		//Updating Value at node
		temp=seg[node][0];
		seg[node][0]=seg[node][1];
		seg[node][1]=seg[node][2];
		seg[node][2]=temp;
		//Flag children
		if(L!=R){
			flag[node<<1]+=2;flag[node<<1]%=3;
			flag[node<<1|1]+=2;flag[node<<1|1]%=3;
		}
	}
	flag[node]=0;
}
void rupdate(int node)
{
	seg[node][0]=seg[node<<1][0]+seg[node<<1|1][0];
	seg[node][1]=seg[node<<1][1]+seg[node<<1|1][1];
	seg[node][2]=seg[node<<1][2]+seg[node<<1|1][2];
}
void update(int node,int l,int r,int L,int R)
{
	if (flag[node])
	{icpc
		prop(node,L,R); //Lazy Propagation
	}
	int M = (L+R)>>1;
```

```
        if(l>r) return;//Used for decoy updates
        if (l==L && r==R)
        {
                flag[node]++;
                prop(node,L,R);
        }
        else if(r<=M)
        {
                update(node<<1,l,r,L,M);
                update(node<<1|1,1,0,M+1,R);//Decoy update to remove flags
                rupdate(node);          //Updating Values while traversing up
        }
        else if(l>M)
        {
                update(node<<1,1,0,L,M);//Decoy update to remove flags
                update(node<<1|1,l,r,M+1,R);
                rupdate(node);          //Updating Values while traversing up
        }
        else
        {
                update(node<<1,l,M,L,M);
                update(node<<1|1,M+1,r,M+1,R);
                rupdate(node);          //Updating Values while traversing up
        }
}
int query(int node,int l,int r,int L,int R)
{
                if (flag[node]) prop(node,L,R); //Lazy Propagation
                int M = (L+R)>>1;
                if (l==L && r==R) return seg[node][0]; //Return Value
                else if(r<=M) return query(node<<1,l,r,L,M);
                else if(l>M) return query(node<<1|1,l,r,M+1,R);
                else return query(node<<1,l,M,L,M)+query(node<<1|1,M+1,r,M+1,R); //Split and
Merge
}



// ----------  KMP
const int max_n = (int)1e6;
int pi[max_n];
char P[max_n], T[max_n];
int n,m;
void table()
{
      pi[0] = -1;
      int k = -1;
      for(int i = 1; i <= m; i++) {
            while(k >= 0 && P[k+1] != P[i])
            k = pi[k];
            pi[i] = ++k;
      }
}
void kmp()
{
      int k = 0;
      for(int i = 1; i <= n; i++) {
            while(k >= 0 && P[k+1] != T[i])
                  k = pi[k];
            k++;
            if(k == m) {
                  cout << i-m+1 << endl;
                  k = pi[k];
            }
      }
}
int main(){
      int N;
      while(scanf("%d",&N) != EOF){
```

```
		scanf("%s",P+1);
		scanf("%s",T+1);
		n = strlen(T+1);
		m = strlen(P+1);
		table();
		kmp();
		cout << endl;
	}
}

//Another KMP
struct kmp{
	string t,p;
	int* b;
	int* m;
	kmp(string _t, string _p){
		t = _t;
		p = _p;
		b = new int[p.length()+1];
		m = new int[t.length()];
	}
	void init(){
		int i=0,j=-1;
		b[0]=-1;
		while(i<p.length()){
			while(j>=0 && p[j] != p[i]) j=b[j];
			i++; j++;
			b[i]=j;
		}
	}
	void match(){
		int i=0,j=0;
		while(i<t.length()){
			while(j>=0 && p[j] != t[i]) j=b[j];
			i++; j++;
			m[i]=j;
			if(j==p.length()){
				cout << "Match: " << i-j << endl;
				j = b[j];
			}
		}
	}
};

// ----- NUMBER THEORY -------
// return a % b (positive value)
int mod(int a, int b) {
  return ((a%b)+b)%b;
}

// computes gcd(a,b)
int gcd(int a, int b) {
  int tmp;
  while(b){a%=b; tmp=a; a=b; b=tmp;}
  return a;
}

// computes lcm(a,b)
int lcm(int a, int b) {
  return a/gcd(a,b)*b;
}

// returns d = gcd(a,b); finds x,y such that d = ax + by
int extended_euclid(int a, int b, int &x, int &y) {
  int xx = y = 0;
  int yy = x = 1;
  while (b) {
    int q = a/b;
    int t = b; b = a%b; a = t;
```

```
    t = xx; xx = x-q*xx; x = t;
    t = yy; yy = y-q*yy; y = t;
  }
  return a;
}

// finds all solutions to ax = b (mod n)
VI modular_linear_equation_solver(int a, int b, int n) {
  int x, y;
  VI solutions;
  int d = extended_euclid(a, n, x, y);
  if (!(b%d)) {
    x = mod (x*(b/d), n);
    for (int i = 0; i < d; i++)
      solutions.push_back(mod(x + i*(n/d), n));
  }
  return solutions;
}

// computes b such that ab = 1 (mod n), returns -1 on failure
int mod_inverse(int a, int n) {
  int x, y;
  int d = extended_euclid(a, n, x, y);
  if (d > 1) return -1;
  return mod(x,n);
}

// Chinese remainder theorem (special case): find z such that
// z % x = a, z % y = b.  Here, z is unique modulo M = lcm(x,y).
// Return (z,M).  On failure, M = -1.
PII chinese_remainder_theorem(int x, int a, int y, int b) {
  int s, t;
  int d = extended_euclid(x, y, s, t);
  if (a%d != b%d) return make_pair(0, -1);
  return make_pair(mod(s*b*x+t*a*y,x*y)/d, x*y/d);
}

// Chinese remainder theorem: find z such that
// z % x[i] = a[i] for all i.  Note that the solution is
// unique modulo M = lcm_i (x[i]).  Return (z,M).  On
// failure, M = -1.  Note that we do not require the a[i]'s
// to be relatively prime.
PII chinese_remainder_theorem(const VI &x, const VI &a) {
  PII ret = make_pair(a[0], x[0]);
  for (int i = 1; i < x.size(); i++) {
    ret = chinese_remainder_theorem(ret.second, ret.first, x[i], a[i]);
    if (ret.second == -1) break;
  }
  return ret;
}

// computes x and y such that ax + by = c; on failure, x = y =-1
void linear_diophantine(int a, int b, int c, int &x, int &y) {
  int d = gcd(a,b);
  if (c%d) {
    x = y = -1;
  } else {
    x = c/d * mod_inverse(a/d, b/d);
    y = (c-a*x)/b;
  }
}

// ---------- Strongly Connected Components
//Initialize deg[] and adjList[][] before calling SCC()
const int maxNode = 5000;
int deg[maxNode];
int adjList[maxNode][maxNode];

int component[maxNode];                    // what is component number of vertex v
```

```
int compSize[maxNode];                  // how many vertices in comp c
int totalSCC;                           // how many total components found
bool adjComp[maxNode][maxNode]; // adjacency matrix for components

int dfsNum[maxNode], minDfsNum[maxNode], dfsNext;
int currentComp[maxNode], currentSize;
bool inComp[maxNode];

void dfs(int u)
{
	if( dfsNum[u] >= 0) return;

	minDfsNum[u] = dfsNum[u] = dfsNext++;

	currentComp[currentSize++] = u;	// Insert u in current component
	inComp[u] = true;

	for(int i = 0; i < deg[u]; i++)
	{
		int v = adjList[u][i];
		dfs(v);

		if( inComp[v] )		// Check is only for cross edges
			minDfsNum[u] = min (minDfsNum[u], minDfsNum[v] );
	}

	if( minDfsNum[u] == dfsNum[u] )			// New component found
	{
		while(true)
		{
			int v = currentComp[--currentSize];
			component[v] = totalSCC;
			compSize[totalSCC] ++;
			inComp[v] = false;
			if( u == v) break;
		}
		totalSCC ++;
	}
}

void scc(int N)
{

	memset( dfsNum, -1, sizeof dfsNum);		// dfsNum also works as visited array
	memset( compSize, 0, sizeof compSize);
	memset( inComp, false, sizeof inComp);

	currentSize = dfsNext= totalSCC = 0;

	for( int i = 0; i < N; i++ )
		if( dfsNum[i] < 0)	dfs( i );

	for(int i=0;i<N;i++)
		for(int j=0;j<deg[i];j++)
			if(component[i] != component[ adjList[i][j] ])
				adjComp[ component[i] ][ component[adjList[i][j]] ] = true;
}


//----------------TRIE
struct node
{
   char val;int deg,depth;	vector<node> children;
	node(char i='\0',int d=0,int od=0)
	{
		val=i;deg=od;depth=d;
	}
};
```

```
struct trie
{
	node top;
	trie()
	{
		top = node('\0',0,2);
	}
	void insert(char *arr)
	{
		int i,j,flag;
		node* temp = &top;
		for(i=0;i<strlen(arr);i++)
		{
			flag=0;
			for(j=0;j<temp->children.size();j++)
			{
				if(arr[i]==temp->children[j].val)
				{
					flag=1;temp = &(temp->children[j]);temp->deg++;break;
				}
			}
			if(flag==0)
			{
				temp->children.pb(node(arr[i],temp->depth+1,1));temp=&(temp-
>children[temp->children.size()-1]);
			}
		}
	}

	int dfs(node temp)
	{
		if(temp.deg<2) return 0;int maxm=temp.depth;
		for(int i=0;i<temp.children.size();i++)
		{
			maxm = max(dfs(temp.children[i]),maxm);
		}
		return maxm;
	}
};

//---------------------------graham scan pseudocode
# Three points are a counter-clockwise turn if ccw > 0, clockwise if
# ccw < 0, and collinear if ccw = 0 because ccw is a determinant that
# gives the signed area of the triangle formed by p1, p2 and p3.
function ccw(p1, p2, p3):
    return (p2.x - p1.x)*(p3.y - p1.y) - (p2.y - p1.y)*(p3.x - p1.x)


let N           = number of points
let points[N+1] = the array of points
swap points[1] with the point with the lowest y-coordinate
sort points by polar angle with points[1]

# We want points[0] to be a sentinel point that will stop the loop.
let points[0] = points[N]

# M will denote the number of points on the convex hull.
let M = 1
for i = 2 to N:
    # Find next valid point on convex hull.
    while ccw(points[M-1], points[M], points[i]) <= 0:
          if M > 1:
                M -= 1
          # All points are collinear
          else if i == N:
                break
          else
                i += 1
```

```
    # Update M and swap points[i] to the correct place.
    M += 1
    swap points[M] with points[i]

// -------------------- emaxx code ---------
struct pt {
	double x, y;
};

bool cmp (pt a, pt b) {
	return ax <bx l l ax == bx && ay <by;
}

bool cw (pt a, pt b, pt c) {
	return ax * (by-cy) + bx * (cy-ay) + cx * (ay-by) <0;
}

bool ccw (pt a, pt b, pt c) {
	return ax * (by-cy) + bx * (cy-ay) + cx * (ay-by)> 0;
}

void convex_hull (vector <pt> & a) {
	if (a.size () == 1) return;
	sort (a.begin (), a.end (), & cmp);
	pt p1 = a [0], p2 = a.back ();
	vector <pt> up, down;
	up.push_back (p1);
	down.push_back (p1);
	for (size_t i = 1; i <a.size (); + + i) {
		if (i == a.size () -1 l l cw (p1, a [i], p2)) {
			while (up.size ()> = 2 &&! cw (up [up.size () -2], up [up.size () -1], a [i]))
				up.pop_back ();
			up.push_back (a [i]);
		}
		if (i == a.size () -1 l l ccw (p1, a [i], p2)) {
			while (down.size ()> = 2 &&! ccw (down [down.size () -2], down [down.size ()
-1], a [i]))
				down.pop_back ();
			down.push_back (a [i]);
		}
	}
	a.clear ();
	for (size_t i = 0; i <up.size (); + + i)
		a.push_back (up [i]);
	for (size_t i = down.size () -2; i> 0; - i)
		a.push_back (down [i]);
}

//CONVEX HULL (kunal's code)
struct point
{
      int x,y;
      point(int a, int b){x=a;y=b;}
      point(){x=y=0;}
};
struct line
{
      point p1,p2;
      bool segment;
      line(point a,point b,bool s){p1=a;p2=b;segment=s;}
      line(point a,point b){p1=a;p2=b;segment=1;}
      line(){p1=point();p2=point();segment=1;}
      void print(){cout<<"("<<p1.x<<","<<p1.y<<")-("<<p2.x<<","<<p2.y<<")\n";}
};
int CrossPdt(line a, point b)
{
 return ((a.p1.x-a.p2.x)*(a.p2.y-b.y)-(a.p1.y-a.p2.y)*(a.p2.x-b.x));
}
int compare(point a, point b)
```

```
{
    return a.x<b.x||(a.x==b.x&&a.y<b.y);
}
int n;
vector<point> arr;
stack<point> L,P;
stack<line> hull;
int main()
{
    cin>>n;
    int x,y;
    for(int i=0;i<n;i++)
    {
        cin>>x>>y;
        arr.push_back(point(x,y));
    }
    sort(arr.begin(),arr.end(),compare);
    //convex hull construction
    L.push(arr[0]);L.push(arr[1]);
    hull.push(line(arr[0],arr[1]));
    for(int i=2;i<n;i++)
    {
        while(L.size()>=2)
        {
                if(CrossPdt(hull.top(),arr[i])>=0)
                {
                    hull.push(line(L.top(),arr[i]));
                    L.push(arr[i]);
                    break;
                }
                hull.pop();L.pop();
        }
        if(L.size()==1)
        {
            hull.push(line(L.top(),arr[i]));
            L.push(arr[i]);
        }
    }
    P.push(arr[n-1]);P.push(arr[n-2]);
    hull.push(line(arr[n-1],arr[n-2]));
    for(int i=n-3;i>=0;i--)
    {
        while(P.size()>=2)
        {
                if(CrossPdt(hull.top(),arr[i])>=0)
                {
                    hull.push(line(P.top(),arr[i]));
                    P.push(arr[i]);
                    break;
                }
                hull.pop();P.pop();
        }
        if(P.size()==1)
        {
            hull.push(line(P.top(),arr[i]));
            P.push(arr[i]);
        }
    }
    int j=1;
    while(!hull.empty())
    {
                cout<<j++<<": ";
        hull.top().print();hull.pop();
    }
    system("pause");
}
```

//Merge Sort (Bad Space complexity ie., nlogn can be easily made O(n) space)

```
int* merge(int a[],int x,int b[],int y) //merging with n efficiency
{
     int* arr;
     int i=0,u=0,v=0;
     arr=new int[x+y];
     for(i=0;u<x&&v<y;i++)
     {
            if(a[u]<b[v])arr[i]=a[u++];
            else arr[i]=b[v++];
     }
     while(u<x){arr[i]=a[u];i++;u++;}
     while(v<y){arr[i]=b[v];i++;v++;}
     return arr;
}

int* mergesort(int *arr,int n) //recursion with log n iterations
{
     if(n==1)return arr;
     return merge(mergesort(arr,n/2),n/2,mergesort(arr+n/2,n-n/2),n-n/2);
}
//ternary search
long long tenary(VI A,function f)
{
    long long s=0,e=A.size()-1,x=0,y=0;
    while(s<e)
    {
             x=(2*s+e)/3;
             y=(s+2*e)/3;
             if(f(x)<=f(y))e=y;
             else s=x+1;
    }
    return s;
}

//Topological Sort
VI sorted,visited;
long long current;
vector<VI> arr;
void dfs(long long i){
  if(visited[i]==1)return;
  visited[i]=1;
  for(long long k=0;k<arr[i].size();k++){
    dfs(arr[i][k]);
  }
  sorted[current--]=i;
}
void Tsort(){
  current =arr.size()-1;
  for(long long i=0;i<arr.size();i++){
    dfs(i);
  }
}
//Another Topological Sort
VI sorted,degree;
vector<VI> arr;
stack<long long> S;
void Tsort()
{
  int i;
  while(!S.empty()){
    i=S.top();
    sorted.pb(i);
    S.pop();
    for(int k=0;k<arr[i].size();k++){
      degree[arr[i][k]]--;
      if(degree[arr[i][k]]==0)S.push(arr[i][k]);
    }
  }
}
```

```
int main()
{
  long long n;fcin(n);degree.resize(n);
  arr.resize(n);
  long long m; fcin(m);
  long long x,y;
  for(long long i=0;i<m;i++){fcin(x);fcin(y);
    arr[x].pb(y);
    degree[y]++;
  }
  for(int i=0;i<n;i++)if(degree[i]==0)S.push(i);
  Tsort();
  for(int i=0;i<n;i++)cout<<sorted[i]<<" ";cout<<endl;
  return 0;
}


//NICE BIT
#define ll long long
typedef struct struct_fenwick{
  int size, memory;
  ll *data;
}FenwickTree;
FenwickTree FenwickTreeNew(int memory){
  FenwickTree res;
  res.memory=memory; res.data=(ll*)malloc(memory*sizeof(ll));
  return res;
}
void FenwickTreeDelete(FenwickTree *t){free(t->data);}
void FenwickTreeInit(FenwickTree *t, int size){int i; t->size=size; for(i=0;i<size;i++) t->data[i]=0;}
void FenwickTreeAdd(FenwickTree *t,int k,ll add){while(k<t->size)t->data[k]+=add, kl=k+1;}

        ll FenwickTreeGet(FenwickTree *t,int k){ll res=0; while(k>=0)res+=t->data[k],k=(k&(k+1))-1;
return res;}
        FenwickTree bit = FenwickTreeNew(50000); //Creation with memory = 500000
    FenwickTreeInit(&bit,n); //Initialisation with size = n
    FenwickTreeGet(&bit,idx); //Get idx val
    FenwickTreeAdd(&bit,x,y); //Add y at position x

// TOTIENT FUNTION + SIEVE
int num[10000001];
int i,j,temp;
void sieve(){
        for(i=2;i<=3162;i++){
                if(!num[i])for(j = i*i;j<10000001;j+=i){
                        num[j] = i;
                }
        }
}
int phi[10000001]= {0,1,1,2,2,4};
long long arr[10000001] = {0,0,3};
int main(){
        sieve();
        for(i=6;i<=10000000;i++){
                if(!num[i])phi[i] = i-1;
                else {
                        temp = i/num[i];
                        if(temp % num[i] == 0){
                                phi[i] = phi[temp] * num[i];
                        } else {
                                phi[i] = phi[temp] * (num[i]-1);
                        }
                }
        }
        for(i=3;i<=10000000;i++){
                arr[i] = arr[i-1] + (phi[i-1]<<1);
        }
        int tests=fcin();
        while(tests--){
```

```
		printf("%lld\n",arr[fcin()]);
	}
}

//matrix module starts
typedef vector<vector<int> > VVI;
#define mod 1000000007
VVI iden(int n){
	VVI arr;
	arr.resize(n);
	for(int i=0;i<n;i++){
		arr[i].resize(n,0);
		arr[i][i] = 1;
	}
	return arr;
}
VVI mul(VVI A,VVI B){
	int a = A.size();
	int c = B.size();
	int b = B[0].size();
	VVI res;
	res.resize(a);
	for(int i=0;i<a;i++)res[i].resize(b);
	for(int i=0;i<a;i++){
		for(int j=0;j<b;j++){
			res[i][j] = 0;
			for(int k=0;k<c;k++){
				res[i][j] = (res[i][j] + (A[i][k]*1ll*B[k][j])%mod)%mod;
			}
		}
	}
	return res;
}
VVI pow(VVI A,int n){
	int sz = A.size();
	VVI res = iden(sz);
	while (n) {
		if(n&1){
			res = mul(res,A);
		}
		n >>= 1;
		A = mul(A,A);
	}
	return res;
}
void print(VVI a){
	for(int i=0;i<a.size();i++){
		for(int j=0;j<a[i].size();j++)cout << a[i][j] <<" " ; cout << endl;
	}
}
// matrix module ends


//LCA
int n,m;
int T[maxN],P[maxN][maxL],L[maxN];
void doDFS(int p,int v,int level){
	T[v] = p;
	L[v] = level;
	for(int i=0;i<arr[v].size();i++){
		if(arr[v][i] != p)doDFS(v,arr[v][i],level+1);
	}
}
void makeLCAtable(){
	doDFS(-1,1,0); //-1 is parent of root(1)
	memset(P,-1,sizeof P);
	for(int i=1;i <= n;i++)
		P[i][0] = T[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];
}
int LCA(int x, int y){
	if(L[x] < L[y]) x^=y^=x^=y; // swap x and y
	int log;
	for(log = 1; 1<< log <= L[x];log++); // log is log(L[x]) floor
	log--;
	for(int i = log ; i >= 0; i--)
		if(L[x] - (1<<i) >= L[y])
			x = P[x][i];
	if(x==y)return x;
	for(int i=log;i >= 0; i--)
		if(P[x][i] != -1 && P[x][i] != P[y][i])
			x = P[x][i] , y = P[y][i];
	return T[x];
}
int ancestor(int v,int k){
	if(k==0)return v;
	int log = 1;
	while(1 << log <= k)log++;log--;
	return ancestor(P[v][log],k-(1<<log));
}
int main(){
	int tests=fcin();
	int x,y;
	while(tests--){
		n = fcin();
		m = fcin();
		arr.resize(n+1);
		for(int i=0;i<n-1;i++){
			x = fcin();
			y = fcin();
			arr[x].pb(y);
			arr[y].pb(x);
		}
		makeLCAtable();
		while (m--) {
			x = fcin();
			y = fcin();
			cout << ancestor(x,y) << endl;
		}
	}
}

//LCA Online
int n,m;
int T[maxN],P[maxN][maxL],L[maxN];
int LCA(int x, int y){
	if(L[x] < L[y]) x^=y^=x^=y; // swap x and y
	int log;
	for(log = 1; 1<< log <= L[x];log++); // log is log(L[x]) floor
	log--;
	for(int i = log ; i >= 0; i--)
		if(L[x] - (1<<i) >= L[y])
			x = P[x][i];
	if(x==y)return x;
	for(int i=log;i >= 0; i--)
		if(P[x][i] != -1 && P[x][i] != P[y][i])
			x = P[x][i] , y = P[y][i];
	return T[x];
}
int ancestor(int v,int k){
	if(k==0)return v;
	int log = 1;
	while(1 << log <= k)log++;log--;
	return ancestor(P[v][log],k-(1<<log));
}
int main(){
```

```
	int tests=fcin();
	int x,y;
	while(tests--){
		n = fcin();
		memset(P,-1,sizeof P);
		L[1] = 0;
		T[1] = -1;
		int maxLevel = 0, farPoint = 1, diameter = 0;
		for(int i=2;i<=n;i++){
			x = fcin();
			T[i] = x;
			L[i] = L[x] + 1;
			P[i][0] = T[i];
			//make LCA Part
			for(int j=1;1<<j < n; j++)
				if(P[i][j-1] != -1) P[i][j] = P[P[i][j-1]][j-1];

			if(L[i] > maxLevel){
				maxLevel ++;
				diameter ++;
				cout << diameter << endl;
				farPoint = i;
			}
			else {
				int parent = LCA(i,farPoint);
				int temp = L[i] + maxLevel - 2*L[parent];
				if(temp > diameter){
					diameter = temp;
				}
				cout << diameter << endl;
			}
		}
	}
}

//LCP + SUFFIX
long long P[MAXL][MAXN];

long long N, stp, cnt;
//stp is number of steps done so far
//cnt is gonna be the length of the substring
struct str{long long nr[2];long long p;};
bool operator<(str a,str b){return a.nr[0]<b.nr[0] || (a.nr[0]==b.nr[0] && a.nr[1]<b.nr[1]);}
str L[MAXN];
long long A[MAXN];

void suffix(){
	//N = strlen(A);
	for(long long i=0;i<N;i++)P[0][i] = 1000+ A[i];		// initial ranks
	for(stp = 1, cnt = 1; (cnt >> 1) < N ; stp++ , cnt <<= 1){
		for(long long 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; //-1 is implimentaion of fake null
characters
			L[i].p = i;
		}
		sort(L,L+N);
		for(long long i = 0;i<N;i++) // generating new ranks
			P[stp][L[i].p] = i>0 && L[i-1].nr[0] == L[i].nr[0] && L[i-1].nr[1] == L[i].nr[1] ? P[stp]
[L[i-1].p] : i;
	}
}
// lcp code
long long lcp(long long x,long long y){
	x = L[x].p, y = L[y].p;
	long long ans = 0;
	for ( long long k = stp-1 ; x < N && y < N && k >=0; k--) {
		if(P[k][x] == P[k][y]) ans += (1<<k) , x += (1<<k) , y += (1<<k);
	}
```

```
	return ans;
}
//make LCP array
long long LCP[MAXN];
void makeLCP(){
	for(long long k = 1;k<N;k++)LCP[k] = lcp(k,k-1);
}
long long unique(){
	long long ans = N-L[0].p;
	for(long long i = 1;i < N;i++)ans =(ans + N-L[i].p -lcp(i,i-1))%mod;
	return ans;
}
long long U(){
	if(N == 1)return 1;
	suffix();
	return unique() % mod;
}

int main(){
	long long t = fcin();
	long long x,prev;
	while (t--) {
		long long n = fcin();
		cin>>prev;
		for(long long i=1;i<n;i++){
			cin>>x;
			A[i-1] = x-prev;
			prev = x;
		}
		N = n-1;
		if(N == 0)cout << 0 << endl;
		else cout << U() << endl;
	}
}
//------------ MILLER RABIN
/* Deterministic for 2**64, --> all 12 primes till 37
if n < 1,373,653, it is enough to test a = 2 and 3;
if n < 9,080,191, it is enough to test a = 31 and 73;
if n < 4,759,123,141, it is enough to test a = 2, 7, and 61;
if n < 1,122,004,669,633, it is enough to test a = 2, 13, 23, and 1662803;
if n < 2,152,302,898,747, it is enough to test a = 2, 3, 5, 7, and 11;
if n < 3,474,749,660,383, it is enough to test a = 2, 3, 5, 7, 11, and 13;
if n < 341,550,071,728,321, it is enough to test a = 2, 3, 5, 7, 11, 13, and 17.*/
unsigned long long modmult(unsigned long long a,unsigned long long b,unsigned long long N)
{
	if(a>=N)a%=N;
	unsigned long long res=0;
	while(b)
	{
		if(b&1)
		{
			res+=a;
			if(res>=N)res-=N;
		}
		b>>=1;
		a<<=1;
		if(a>=N)a-=N;
	}
	return res;
}

unsigned long long modpow(unsigned long long a,unsigned long long b,unsigned long long N)
{
	if(a>=N)a%=N;
	unsigned long long res=1ll;
	while(b)
	{
		if(b&1)res=modmult(a,res,N);
		b>>=1;
```

```
			a=modmult(a,a,N);
		}
		return res;
}

bool Miller(unsigned long long N)
{
  if(N<2)return false;
  else if(N<4) return true;
  else if((N&1)==0)return false;

		unsigned long long D=N-1,S=0;
		while((D&1)==0)
		{
			D>>=1;
			S++;
		}

		for(int a=0;a<5;a++)
		{
			unsigned long long ad=modpow(rand()%(N-4)+2,D,N);
			if(ad==1||ad==N-1)continue;
			for(unsigned long long r=0;r<S-1;r++)
				if((ad=modmult(ad,ad,N))==N-1)
					goto BPP;
			return false;
			BPP:;
		}
		return true;
}

// FLOYD FULKERSON FLOW MODULE STARTS
int source, sink;
int N;
vector<vector<int> > arr;
#define maxn 1000
#define INF 2000000001
int capacity[maxn][maxn];
int from[maxn], v[maxn];

int bfs(){
		queue<int> q;
		memset(v,0,sizeof v);
		memset(from,-1,sizeof from);
		v[source] = 1;
		q.push(source);
		int where, next = -1, prev;
		while (!q.empty()) {
			where = q.front();
			q.pop();
			for(int i=0;i<arr[where].size();i++){
				next = arr[where][i];
				if(v[next] == 0 && capacity[where][next] > 0){
					from[next] = where;
					v[next] = 1;
					if(next == sink) goto outofwhile;
					q.push(next);
				}
			}
		}
		outofwhile:
		int minCap = INF;
		where = sink;
		while(from[where] != -1){
			prev = from[where];
			minCap = min(minCap,capacity[prev][where]);
			where = prev;
		}
		where = sink;
```

```
	while(from[where] != -1){
		prev = from[where];
		capacity[prev][where] -= minCap;
		capacity[where][prev] += minCap;
		where = prev;
	}
	if(minCap == INF) return 0;
	return minCap;
}
int maxFlow(){
	int ret = 0;
	int temp;
	do{
		temp = bfs();
		ret += temp;
	} while(temp > 0);
	return ret;
}
//FLOYD FLUKERSON FLOW MODULE ENDS

// 2014 -- edmond karp -- max flow ---


#define maxn 500

int arr[maxn][maxn];
int gr[maxn][maxn];

int source, sink;

int vis[maxn];
int par[maxn];
int N;
bool bfs(){
	queue<int> q;
	memset(vis, 0, sizeof vis);
	vis[source] = 1;
	q.push(source);
	while(!q.empty()){
		int u = q.front();
		q.pop();
		if(u == sink)
			return true;
		for(int v=0;v<N;v++){
			if(!vis[v] && gr[u][v] > 0){
				vis[v] = 1;
				par[v] = u;
				q.push(v);
			}
		}
	}
	return false;
}
int edmond(){
	int flow = 0;
	for(int i=0;i<N;i++)
		for(int j=0;j<N;j++)
			gr[i][j] = arr[i][j];

	while (bfs()){
		int del = INT_MAX;
		for(int v=sink; v != source; v=par[v]){
			int u = par[v];
			del = min(del, gr[u][v]);
		}
		for(int v=sink; v != source; v=par[v]){
			int u = par[v];
			gr[u][v] -= del;
			gr[v][u] += del;
```

```
			}
			flow += del;
		}
		return flow;
}

// new flow ends


//max matching
//   INPUT: w[i][j] = edge between row node i and column node j
//   OUTPUT: mr[i] = assignment for row node i, -1 if unassigned
//           mc[j] = assignment for column node j, -1 if unassigned
//           function returns number of matches made
bool FindMatch(int i, const VVI &w, VI &mr, VI &mc, VI &seen) {
  for (int j = 0; j < w[i].size(); j++) {
    if (w[i][j] && !seen[j]) {
      seen[j] = true;
      if (mc[j] < 0 || FindMatch(mc[j], w, mr, mc, seen)) {
        mr[i] = j;
        mc[j] = i;
        return true;
      }
    }
  }
  return false;
}

int BipartiteMatching(const VVI &w, VI &mr, VI &mc) {
  mr = VI(w.size(), -1);
  mc = VI(w[0].size(), -1);

  int ct = 0;
  for (int i = 0; i < w.size(); i++) {
    VI seen(w[0].size());
    if (FindMatch(i, w, mr, mc, seen)) ct++;
  }
  return ct;
}

//////////////////////////////////////////////////////////////////////
// Min cost bipartite matching via shortest augmenting paths
//
// This is an O(n^3) implementation of a shortest augmenting path
// algorithm for finding min cost perfect matchings in dense
// graphs.  In practice, it solves 1000x1000 problems in around 1
// second.
//
//   cost[i][j] = cost for pairing left node i with right node j
//   Lmate[i] = index of right node that left node i pairs with
//   Rmate[j] = index of left node that right node j pairs with
//
// The values in cost[i][j] may be positive or negative.  To perform
// maximization, simply negate the cost[][] matrix.
//////////////////////////////////////////////////////////////////////

#include <algorithm>
#include <cstdio>
#include <cmath>
#include <vector>

using namespace std;

typedef vector<double> VD;
typedef vector<VD> VVD;
typedef vector<int> VI;

double MinCostMatching(const VVD &cost, VI &Lmate, VI &Rmate) {
  int n = int(cost.size());
```

```
// construct dual feasible solution
VD u(n);
VD v(n);
for (int i = 0; i < n; i++) {
  u[i] = cost[i][0];
  for (int j = 1; j < n; j++) u[i] = min(u[i], cost[i][j]);
}
for (int j = 0; j < n; j++) {
  v[j] = cost[0][j] - u[0];
  for (int i = 1; i < n; i++) v[j] = min(v[j], cost[i][j] - u[i]);
}

// construct primal solution satisfying complementary slackness
Lmate = VI(n, -1);
Rmate = VI(n, -1);
int mated = 0;
for (int i = 0; i < n; i++) {
  for (int j = 0; j < n; j++) {
    if (Rmate[j] != -1) continue;
    if (fabs(cost[i][j] - u[i] - v[j]) < 1e-10) {
        Lmate[i] = j;
        Rmate[j] = i;
        mated++;
        break;
    }
  }
}

VD dist(n);
VI dad(n);
VI seen(n);

// repeat until primal solution is feasible
while (mated < n) {

  // find an unmatched left node
  int s = 0;
  while (Lmate[s] != -1) s++;

  // initialize Dijkstra
  fill(dad.begin(), dad.end(), -1);
  fill(seen.begin(), seen.end(), 0);
  for (int k = 0; k < n; k++)
    dist[k] = cost[s][k] - u[s] - v[k];

  int j = 0;
  while (true) {

    // find closest
    j = -1;
    for (int k = 0; k < n; k++) {
        if (seen[k]) continue;
        if (j == -1 || dist[k] < dist[j]) j = k;
    }
    seen[j] = 1;

    // termination condition
    if (Rmate[j] == -1) break;

    // relax neighbors
    const int i = Rmate[j];
    for (int k = 0; k < n; k++) {
        if (seen[k]) continue;
        const double new_dist = dist[j] + cost[i][k] - u[i] - v[k];
        if (dist[k] > new_dist) {
          dist[k] = new_dist;
          dad[k] = j;
        }
```

```
      }
    }

    // update dual variables
    for (int k = 0; k < n; k++) {
      if (k == j || !seen[k]) continue;
      const int i = Rmate[k];
      v[k] += dist[k] - dist[j];
      u[i] -= dist[k] - dist[j];
    }
    u[s] += dist[j];

    // augment along path
    while (dad[j] >= 0) {
      const int d = dad[j];
      Rmate[j] = Rmate[d];
      Lmate[Rmate[j]] = j;
      j = d;
    }
    Rmate[j] = s;
    Lmate[s] = j;

    mated++;
  }

  double value = 0;
  for (int i = 0; i < n; i++)
    value += cost[i][Lmate[i]];

  return value;
}

//----------- RMQ-SPARSE TABLE

  void process2(int M[MAXN][LOGMAXN], int A[MAXN], int N)
  {
      int i, j;

  //initialize M for the intervals with length 1
      for (i = 0; i < N; i++)
          M[i][0] = i;
  //compute values from smaller to bigger intervals
      for (j = 1; 1 << j <= N; j++)
          for (i = 0; i + (1 << j) - 1 < N; i++)
              if (A[M[i][j - 1]] < A[M[i + (1 << (j - 1))][j - 1]])
                  M[i][j] = M[i][j - 1];
              else
                  M[i][j] = M[i + (1 << (j - 1))][j - 1];
  }
 /*
   Constants:
   ----------
   [1.2]  BigInteger.ONE  - The BigInteger constant one.
   [1.2]  BigInteger.ZERO - The BigInteger constant zero.
   Creating BigIntegers
   -------------
   1. From Strings
     a) BigInteger(String val);
     b) BigInteger(String val, int radix);

   2. From byte arrays
     a) BigInteger(byte[] val);
     b) BigInteger(int signum, byte[] magnitude)

   3. From a long integer
     a) static BigInteger BigInteger.valueOf(long val)

   Math operations:
   ----------------
```

```
   A + B = C               -->  C = A.add(B);
   A - B = C               -->  C = A.subtract(B);
   A * B = C               -->  C = A.multiply(B);
   A / B = C               -->  C = A.divide(B);
   A % B = C               -->  C = A.remainder(B);
   A % B = C where C > 0  -->  C = A.mod(B);
   A / B = Q & A % B = R  -->  C = A.divideAndRemainder(B);
                               (Q = C[0], R = C[1])
   A ^ b = C               -->  C = A.pow(B);
   abs(A) = C              -->  C = A.abs();
   -(A) = C                -->  C = A.negate();

   gcd(A,B) = C            -->  C = A.gcd(B);
   (A ^ B) % M             -->  C = A.modPow(B,M);
   C = inverse of A mod M -->  C = A.modInverse(M);

   max(A,B) = C            -->  C = A.max(B);
   min(A,B) = C            -->  C = A.min(B);

   Bit Operations
   ------------------
   ~A = C       (NOT)     -->  C = A.not();
   A & B = C    (AND)     -->  C = A.and(B);
   A | B = C    (OR)      -->  C = A.or(B);
   A ^ B = C    (XOR)     -->  C = A.xor(B);
   A & ~B = C   (ANDNOT)  -->  C = A.andNot(B);
   A << n = C   (LSHIFT)  -->  C = A.shiftLeft(n);
   A >> n = C   (RSHIFT)  -->  C = A.shiftRight(n);

   Clear n'th bit of A    -->  C = A.clearBit(n);
   Set   n'th bit of A    -->  C = A.setBit(n);
   Flip  n'th bit of A    -->  C = A.flipBit(n);
   Test  n'th bit of A    -->  C = A.testBit(n);

   Bitcount of A = n      -->  n = A.bitCount();
   Bitlength of A = n     -->  n = A.bitLength();
   Lowest set bit of A    -->  n = A.getLowestSetBit();

   Comparison Operations
   ---------------------
   A < B                  -->  A.compareTo(B) == -1;
   A == B                 -->  A.compareTo(B) ==  0
                          or A.equals(B);
   A > B                  -->  A.compareTo(B) ==  1;

   A < 0                  -->  A.signum() == -1;
   A == 0                 -->  A.signum() ==  0;
   A > 0                  -->  A.signum() ==  1;

   Conversion:
   -----------
   double                 -->  A.doubleValue();
   float                  -->  A.floatValue();
   int                    -->  A.intValue();
   long                   -->  A.longValue();
   byte[]                 -->  A.toByteArray();
   String                 -->  A.toString();
   String (base b)        -->  A.toString(b);

   ----------------------------------------------------------------*/

/* Reads in lines of input until EOF is encountered.  For each line
   of input it will extract two integers and then print out their
   GCD. */

import java.math.*;
import java.io.*;
import java.util.*;
```

```
class BigIntegers {
    public static void main(String[] args) {
        BufferedReader in = new BufferedReader(
                                        new InputStreamReader(System.in));
        String line;
            StringTokenizer st;
            BigInteger a;
            BigInteger b;

            try {
                while(true) {
                line = in.readLine();
                if(line == null) break;

                        st = new StringTokenizer(line);
                        a = new BigInteger(st.nextToken());
                        b = new BigInteger(st.nextToken());

                        System.out.println( a.gcd(b) );
                }
            } catch(Exception e) {
            System.err.println(e.toString());
            }
    }
}
```

```
#include <iostream>
#include <vector>

// Take d to be the size of the characters.

long long modulo(int a,int b,int c){
	long long x=1, y=a;
	while(b > 0){
		if (b % 2 == 1) {
			x=(x*y)%c;
		}
		y = (y*y)%c;
		b /= 2;
	}
	return x%c;
}

std::vector<int> rabin_karp(std::string t, std::string p, long long d, long long q) {
	std::vector<int> matches;
	int n;
	n = t.size();
	int m;
	m = p.size();
	long long h;
	h = modulo(d, m-1, q);
	long long pp = 0;
	long long tp[n-m+1];
	tp[0] = 0;
	for (int i = 0; i < m; i++) {
		pp = (d*pp + p[i]) % q;
		tp[0] = (d*tp[0] + t[i]) % q;
	}
	for (int s = 0; s < n-m+1; s++) {
		//std::cout << s << tp[s] << std::endl;
		if (pp == tp[s]) {
			bool val = true;
			for (int i = 0; i < m; i++) {
				if (p[i] != t[s+i]) {
					val = false;
					break;
				}
			}
			if (val == true) {
```

```
                    matches.push_back(s);
                }
            }
            if (s < n - m) {
                tp[s+1] = (d*(tp[s] - t[s]*h) + t[s+m]) % q;
                if (tp[s+1] < 0) {
                    tp[s+1] += q;
                }
            }
        }
        return matches;
}

using namespace std;
int main(int argc, char *argv[]) {
        std::vector<int> v;
        v = rabin_karp("this is the randomness of the lanthewitchthethethe", "the", 26, 101);
        for (int i = 0; i < v.size(); i++) {
            cout << v[i] << endl;
        }
}



// Adjacency list implementation of Dinic's blocking flow algorithm.
// This is very fast in practice, and only loses to push-relabel flow.
//
// Running time:
//     O(|V|^2 |E|)
//
// INPUT:
//     - graph, constructed using AddEdge()
//     - source
//     - sink
//
// OUTPUT:
//     - maximum flow value
//     - To obtain the actual flow values, look at all edges with
//       capacity > 0 (zero capacity edges are residual edges).

#include <cmath>
#include <vector>
#include <iostream>
#include <queue>

using namespace std;

const int INF = 2000000000;

struct Edge {
  int from, to, cap, flow, index;
  Edge(int from, int to, int cap, int flow, int index) :
    from(from), to(to), cap(cap), flow(flow), index(index) {}
};

struct Dinic {
  int N;
  vector<vector<Edge> > G;
  vector<Edge *> dad;
  vector<int> Q;

  Dinic(int N) : N(N), G(N), dad(N), Q(N) {}

  void AddEdge(int from, int to, int cap) {
    G[from].push_back(Edge(from, to, cap, 0, G[to].size()));
    if (from == to) G[from].back().index++;
    G[to].push_back(Edge(to, from, 0, 0, G[from].size() - 1));
  }
```

```
  long long BlockingFlow(int s, int t) {
    fill(dad.begin(), dad.end(), (Edge *) NULL);
    dad[s] = &G[0][0] - 1;

    int head = 0, tail = 0;
    Q[tail++] = s;
    while (head < tail) {
      int x = Q[head++];
      for (int i = 0; i < G[x].size(); i++) {
            Edge &e = G[x][i];
            if (!dad[e.to] && e.cap - e.flow > 0) {
              dad[e.to] = &G[x][i];
              Q[tail++] = e.to;
            }
      }
    }
    if (!dad[t]) return 0;

    long long totflow = 0;
    for (int i = 0; i < G[t].size(); i++) {
      Edge *start = &G[G[t][i].to][G[t][i].index];
      int amt = INF;
      for (Edge *e = start; amt && e != dad[s]; e = dad[e->from]) {
            if (!e) { amt = 0; break; }
            amt = min(amt, e->cap - e->flow);
      }
      if (amt == 0) continue;
      for (Edge *e = start; amt && e != dad[s]; e = dad[e->from]) {
            e->flow += amt;
            G[e->to][e->index].flow -= amt;
      }
      totflow += amt;
    }
    return totflow;
  }

  long long GetMaxFlow(int s, int t) {
    long long totflow = 0;
    while (long long flow = BlockingFlow(s, t))
      totflow += flow;
    return totflow;
  }
};
```