Contents

```
#include <bits/stdc++.h>
#define in freopen("input.txt", "r", stdin);
#define out freopen("output.txt", "w", stdout);
#define clr(arr, key) memset(arr, key, sizeof arr)
#define pb push_back
#define mp(a, b) make_pair(a, b)    #define infinity (1
<< 28)
#define LL long long    #define pii pair <int, int>
#define PI acos(-1)     #define gcd(a, b) __gcd(a, b)
#define CF ios_base::sync_with_stdio(0);cin.tie(0);
#define lcm(a, b) ((a)*((b)/gcd(a,b)))
#define all(v) v.begin(), v.end()
#define no_of_ones __builtin_popcount //
__builtin_popcountll for LL
#define SZ(v) (int)(v.size())    #define eps 10e-7
//int col[8] = {0, 1, 1, 1, 0, -1, -1, -1};
//int row[8] = {1, 1, 0, -1, -1, -1, 0, 1};
//int col[4] = {1, 0, -1, 0};
//int row[4] = {0, 1, 0, -1};
//int months[13] = {0,
,31,28,31,30,31,30,31,31,30,31,30,31};
//int X[]={1,1,2,2,-1,-1,-2,-2};//knight move//
//int Y[]={2,-2,1,-1,2,-2,1,-1};//knight move//
using namespace std;
struct point{int x, y; point () {} point(int a, int b)
{x = a, y = b;}};
template <class T> T sqr(T a){return a * a;}
template <class T> T power(T n, T p) { T res = 1;
for(int i = 0; i < p; i++) res *= n; return res;}
template <class T> double getdist(T a, T b){return
sqrt((a.x - b.x) * (a.x - b.x) + (a.y - b.y) * (a.y -
b.y));}    // distance between a and b
template <class T> T extract(string s, T ret)
{stringstream ss(s); ss >> ret; return ret;}    //
extract words or numbers from a line
template <class T> string tostring(T n) {stringstream
ss; ss << n; return ss.str();}    // convert a number to
string
LL bigmod(LL B,LL P,LL M){LL R=1; while(P>0)
{if(P%2==1){R=(R*B)%M;}P/=2;B=(B*B)%M;} return R;}
```

1. 2D SEGMENT TREE

```
void build_y (int vx, int lx, int rx, int vy, int ly,
int ry) {
      if (ly == ry)
            if (lx == rx)
                  t[vx][vy] = a[lx][ly];
            else
                  t[vx][vy] = t[vx*2][vy] +
t[vx*2+1][vy];
      else {
            int my = (ly + ry) / 2;
            build_y (vx, lx, rx, vy*2, ly, my);
            build_y (vx, lx, rx, vy*2+1, my+1, ry);
            t[vx][vy] = t[vx][vy*2] + t[vx][vy*2+1];
      }
}

void build_x (int vx, int lx, int rx) {
      if (lx != rx) {
            int mx = (lx + rx) / 2;
            build_x (vx*2, lx, mx);
            build_x (vx*2+1, mx+1, rx);
      }
      build_y (vx, lx, rx, 1, 0, m-1);
}

void update_y (int vx, int lx, int rx, int vy, int ly,
int ry, int x, int y, int new_val) {
      if (ly == ry) {
            if (lx == rx)
                  t[vx][vy] = new_val;
            else
                  t[vx][vy] = t[vx*2][vy] +
t[vx*2+1][vy];
      }
      else {
            int my = (ly + ry) / 2;
            if (y <= my)
                  update_y (vx, lx, rx, vy*2, ly, my, x,
y, new_val);
            else
```

```
                update_y (vx, lx, rx, vy*2+1, my+1,
ry, x, y, new_val);
            t[vx][vy] = t[vx][vy*2] + t[vx][vy*2+1];
      }
}

void update_x (int vx, int lx, int rx, int x, int y, int
new_val) {
      if (lx != rx) {
            int mx = (lx + rx) / 2;
            if (x <= mx)
                  update_x (vx*2, lx, mx, x, y,
new_val);
            else
                  update_x (vx*2+1, mx+1, rx, x, y,
new_val);
      }
      update_y (vx, lx, rx, 1, 0, m-1, x, y, new_val);
}

int sum_y (int vx, int vy, int tly, int try_, int ly,
int ry) {
      if (ly > ry)
            return 0;
      if (ly == tly && try_ == ry)
            return t[vx][vy];
      int tmy = (tly + try_) / 2;
      return sum_y (vx, vy*2, tly, tmy, ly, min(ry,tmy))
            + sum_y (vx, vy*2+1, tmy+1, try_,
max(ly,tmy+1), ry);
}

int sum_x (int vx, int tlx, int trx, int lx, int rx, int
ly, int ry) {
      if (lx > rx)
            return 0;
      if (lx == tlx && trx == rx)
            return sum_y (vx, 1, 0, m-1, ly, ry);
      int tmx = (tlx + trx) / 2;
      return sum_x (vx*2, tlx, tmx, lx, min(rx,tmx), ly,
ry)
            + sum_x (vx*2+1, tmx+1, trx, max(lx,tmx+1),
rx, ly, ry);
}
```

2. 2-SAT

```
//0 based
VI adj[2*sz]; //2*sz for true and false argument(only
adj should be cleared)
int col[2*sz],low[2*sz],tim[2*sz],timer;
int group_id[2*sz],components;//components=number of
components, group_id = which node belongs to which node
bool ans[sz]; //boolean assignment ans
stack<int>S;
void scc(int u);
int TarjanSCC(int n); //n=nodes (some change may be
required here)
//double nodes needed normally
bool TwoSAT(int n) //n=nodes (some change may be
required here)
{
    TarjanSCC(n);
    int i;
    for(i=0;i<n;i+=2)
    {
        if(group_id[i]==group_id[i+1])
            return false;
        if(group_id[i]<group_id[i+1]) //Checking who is
lower in Topological sort
            ans[i/2]=true;
        else ans[i/2]=false;
    }
    return true;
}
void add(int ina,int inb)
{
    adj[ina].pb(inb);
}
int complement(int n)
{
    return n^1;
}
```

```
void initialize(int n)
{
    for(int i=0;i<n;i++)
        adj[i].clear();
}
int main()
{
    int n, m, i, u, v;
    while(~scanf("%d %d", &n, &m))
    {
        initialize(n<<1);
        fr(i,0,m-1)
        {
            scanf("%d %d", &u, &v);
            if(u>0) u = 2*u-2;
            else u = -2*u-1;
            if(v>0) v = 2*v-2;
            else v = -2*v-1;
            add(complement(u),v);
            add(complement(v),u);
        }
        if(TwoSAT(n<<1)) puts("YES");
        else puts("NO");
    }
    return 0;
}
```

3. AHO CORASICK

```
struct tt
{
    int par, child[26], dep;
    vector<int>str;
};

tt T[250010]; /// size = total number of pattern strings
* length per string
char words[502][502];
int sz1;
char str[1000010]; /// main string

void init(int lim)
{
    for(int i=0;i<=lim;i++)
    {
        T[i].par=0;
        T[i].dep=0;
        memset(T[i].child, 0, sizeof T[i].child);
        T[i].str.clear();
    }
    sz1=1;
    return;
}

void build(int n)
{
    int i, j, last, len;
    char ch;
    for(i=0;i<n;i++)
    {
        last=0;
        len = strlen(words[i]);
        for(j=0;j<len;j++)
        {
            ch = words[i][j] - 'a';
            if(T[last].child[ch]==0)
                T[last].child[ch]=sz1++;
            T[T[last].child[ch]].dep = T[last].dep + 1;
            last = T[last].child[ch];
        }
        T[last].str.pb(i);
    }

    queue<int>Q;
    for(i=0;i<26;i++)
    {
        if(T[0].child[i])
        {
            Q.push(T[0].child[i]);
            T[T[0].child[i]].par = 0;
        }
    }
    int u, v, k;
```

```
    while(!Q.empty())  /// implementing kmp in the trie
tree with kind of bfs
    {
        u = Q.front(); Q.pop();
        for(i=0;i<26;i++)
        {
            if(T[u].child[i])
            {
                v = T[u].child[i];
                k = T[u].par;
                while(k>0 && T[k].child[i]==0)
                    k = T[k].par;
                T[v].par = T[k].child[i];
                Q.push(v);
            }
        }
    }
    return;
}

int freq[250000], ans[505];

void search() /// this function will take a string as
main input and find the frequency of pattern strings in
this string
{
    int i, j, k, len, u, v;
    char ch;
    len = strlen(str);
    int cur=0;
    memset(freq, 0, sizeof freq);
    for(i=0;i<len;i++)
    {
        ch = str[i] - 'a';
        if(T[cur].child[ch]==0)
        {
            k = T[cur].par;
            while(k>0 && T[k].child[ch]==0)
                k = T[k].par;
            cur = T[k].child[ch];
        }
        else
            cur = T[cur].child[ch];
        freq[cur]++; /// ei node ei frequency pabe
    }
    vector<pii>store;
    for(i=0;i<sz1;i++)
        store.pb(MP(T[i].dep, i));
    sort(store.rbegin(), store.rend());
    for(i=0;i<sz1;i++)
    {
        v = store[i].second;
        freq[T[v].par]+=freq[v]; /// parent gulake
cummulatively frequency gula die dea
    }

    for(i=1;i<sz1;i++)
    {
        if(SZ(T[i].str))
        {
            for(j=0;j<SZ(T[i].str);j++)
            {
                ans[T[i].str[j]] = freq[i];
            }
        }
    }
}

int main()
{
    int t, cas=1;
    scanf("%d", &t);
    sz1=1;
    while(t--)
    {
        init(sz1);
        int n, i, j;
        scanf("%d", &n);
        scanf(" %s", &str);
        for(i=0;i<n;i++)
            scanf(" %s", &words[i]); /// input of
pattern strings
```

```
        build(n); ///building trie with kmp idea (this
function deals with only the patterns)
        search();
        csprnt;
        for(i=0;i<n;i++)
            printf("%d\n", ans[i]);
    }
    return 0;
}
```

4. ARTICULATION POINT, BRIDGE & BCC

```
vector <int> adj[SZ];
int discover[SZ], bedge[SZ], discovery_time;
bool arti[SZ];
pair <int, int> pr, e, cur;
vector <pair <int, int> > bridges;
stack <pair <int, int> > s;
void dfs(int node, int from)
{
    arti[node] = false;
    discover[node] = bedge[node] = discovery_time++;
    int i, connected = adj[node].size(), to, child = 0;
    for(i = 0; i < connected; i++)
    {
        to = adj[node][i];
        if(to == from) continue;
        ///for bcc
        if(!discover[to])
        {
            s.push(make_pair(node, to));
            dfs(to, node);
            bedge[node] = min(bedge[node], bedge[to]);
            if(bedge[to] >= discover[node])
            {
                bcc++;    cur = make_pair(node, to);
                do {
                    e = s.top(); s.pop();
                } while(e != cur);
            }
        }
        else if(discover[node] > discover[to])
        {
            s.push(make_pair(node, to));
            bedge[node] = min(discover[to],
bedge[node]);
        }///for bcc
        if(!discover[to])
        {
            dfs(to, node);
            child++;
            bedge[node] = min(bedge[node], bedge[to]);
            ///for point
            if(bedge[to] >= discover[node] && from != -
1)
                arti[node] = true; ///for point
            ///for bridges
            if(bedge[to] > discover[node])
            {
                if(node < to)
                    pr = make_pair(node, to);
                else
                    pr = make_pair(to, node);
                bridges.pub(pr);
            }///for bridges
        }
        else if(discover[node] > discover[to])
            bedge[node] = min(discover[to],
bedge[node]);
    }
    if(from == -1 && child >= 2)
        arti[node] = true;///for point only
}
```

5. BELLMAN FORD

```
struct edge
{
    int u, v, w;
    edge();
    edge(int a, int b, int c)
    {
        u = a;
```

```
        v = b;
        w = c;
    }
};

vector <edge> graph;
vector <int> adj[MAX];

int dist[MAX], n, m;

///graph is a vector of edges
bool belford(void)
{
    for(i = 1; i <= n; i++) dist[i] = infinity;
        dist[0] = 0;
    int i, j, u, v, w;
    for(i = 1; i < n; i++)
    {
        for(j = 0; j < m; j++)
        {
            u = graph[j].u;
            v = graph[j].v;
            w = graph[j].w;
            if(dist[u] + w < dist[v])
                dist[v] = dist[u] + w;
        }
    }
    for(j = 0; j < m; j++)
    {
        u = graph[j].u;
        v = graph[j].v;
        w = graph[j].w;
        if(dist[u] + w < dist[v])
            return true;
    }
    return false;
}
```

6. BPM

NORMAL

```
int par[MAX];
bool col[MAX];
int MAX_BMP(int n) // finds maximum possible bipartite
matching
{
    int ret = 0, i;
    clr(par, -1);
    for(i = 0; i < n; i++)
    {
        clr(col, 0);
        if(dfs(i)) ret++;
    }
    return ret;
}
int dfs(int u)
{
    if(col[u])
        return false;
    col[u] = true;
    for(int i = 0; i < SZ(adj[u]); i++)
    {
        int v = adj[u][i];
        if(par[v] == -1 || dfs(par[v]))
        {
            par[v] = u;
            return true;
        }
    }
    return false;
}
```

HOPCROFT CARP

```
/*
 * Complexity : O (|E|√|V|)
 * 1 based indexing
 */

namespace hopcroftKarp{
    #define MAXN 100001 /// Maximum possible Number of
nodes
    #define MAXE 150001 /// Maximum possible Number of
edges
    #define INF (1<<29)
```

```
    int ptr[MAXN],next[MAXE],zu[MAXE];
    int n,m,match[MAXN],D[MAXN],q[MAXN];
    void init(int _n){ /// initialization _n=number of
nodes
        n=_n;
        m=0;
        memset(ptr,~0,sizeof(int)*(n+1));
    }
    void add_edge(int u,int v){ /// Adding edge between
u and v
        next[m]=ptr[u];ptr[u]=m;zu[m]=v;++m;
    }
    bool bfs(){
        int u,v;
        register int i;
        int qh=0, qt=0;
        for(i=1; i<=n; i++){
            if(!match[i]){
                D[i]=0;
                q[qt++]=i;
            }
            else D[i]=INF;
        }
        D[0]=INF;
        while(qh<qt){
            u=q[qh++];
            if(u!=0){
                for(i=ptr[u]; ~i; i=next[i]){
                    v=zu[i];
                    if(D[match[v]]==INF){
                        D[match[v]]=D[u]+1;
                        q[qt++]=match[v];
                    }
                }
            }
        }
        return D[0]!=INF;
    }
    bool dfs(int u){
        int v;
        register int i;
        if(u){
            for(i=ptr[u]; ~i; i=next[i]){
                v=zu[i];
                if(D[match[v]]==D[u]+1){
                    if(dfs(match[v])){
                        match[v]=u;
                        match[u]=v;
                        return true;
                    }
                }
            }
            D[u]=INF;
            return false;
        }
        return true;
    }
    int run(){
        int matching=0;
        register int i;
        while(bfs())
            for(i=1; i<=n; i++)
                if(!match[i] && dfs(i))
                    matching++;
        return matching;
    }
    #undef MAXN
    #undef INF
};
```

MIN COST

```
// 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.

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;
}
```

7. BIT

```
LL tree[MAX];

LL read(int idx)
{
    LL sum = 0;
    while(idx > 0){
        sum += tree[idx];
        idx -= (idx & -idx);
    }
    return sum;
}

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;
}

void update(int idx, LL val, int n)
{
    while(idx <= n)
    {
        tree[idx] += val;
        idx += (idx & -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);
    }
}

read(r1,c1,r2,c2) = read(r2,c2) - read(r2,c1-1) -
read(r1-1,c2) + read(r1-1,c1-1);
int read(int x,int y)  // return sum from 1,1 to x,y.
{
    int sum= 0;
    while( x)
    {
        int y1 = y;
        while(y1)
        {
            sum += tree[x][y1];
            y1 -= y1 & -y1;
        }
        x -= x & -x;
    }
    return sum;
}*/
```

8. CENTROID DECOMPOSITION

```
void dfs2(int u, int prev)
{
    child[u] = 1;
    for(auto v: adj[u])
        if(v != prev && !deleted[v])
            dfs2(v, u), child[u] += child[v];
}

int dfs2(int u, int prev, int nodesLeft)
{
    for(auto v: adj[u])
        if(v != prev && !deleted[v] && child[v] >
nodesLeft/2)
            return dfs2(v, u, nodesLeft);
    return u;
}

void decompose(int u, int prev)
{
    dfs2(u, -1);
    int centroid = dfs2(u, -1, child[u]);
    par[centroid] = prev;
    deleted[centroid] = 1;
    for(auto v: adj[centroid])
        if(!deleted[v])
            decompose(v, centroid);
}
```

9. CLOSEST PAIR

```
point arr[MAX], sortedY[MAX];
bool flag[MAX];

bool compareX(const point &a, const point &b){
    return a.x < b.x;
}
bool compareY(const point &a, const point &b){
    return a.y < b.y;
}

double closest_pair(point X[], point Y[], int n)
{
    double left_call, right_call, mindist;
    if(n == 1) return infinity;
    if(n == 2)
        return getdist(X[0], X[1]);
    int n1, n2, ns, j, m = n / 2, i;
```

```
    point xL[m + 1], xR[m + 1], yL[m + 1], yR[m + 1], Xm
= X[m - 1], yS[n];
    for(i = 0; i < m; i++)
    {
        xL[i] = X[i];
        flag[X[i].i] = 0;
    }
    for(; i < n; i++)
    {
        xR[i - m] = X[i];
        flag[X[i].i] = 1;
    }
    for(i = n2 = n1 = 0; i < n; i++)
    {
        if(!flag[Y[i].i]) yL[n1++] = Y[i];
        else yR[n2++] = Y[i];
    }
    left_call = closest_pair(xL, yL, n1);
    right_call = closest_pair(xR, yR, n2);
    mindist = min(left_call, right_call);
    for(i = ns = 0; i < n; i++)
        if(sqr(Y[i].x - Xm.x) < mindist)
            yS[ns++] = Y[i];
    for(i = 0; i < ns; i++)
        for(j = i + 1; j < ns && sqr(yS[j].y - yS[i].y)
< mindist; j++)
            mindist = min(mindist, getdist(yS[i],
yS[j]));
    return mindist;
}

int main()
{
    int n, i;
    double ans;
    while(scanf("%d", &n) == 1 && n)
    {
        ans = infinity;
        for(i = 0; i < n; i++)
        {
            scanf("%lf %lf", &arr[i].x, &arr[i].y);
            arr[i].i = i;
            sortedY[i] = arr[i];
        }
        sort(arr, arr + n, compareX);
        sort(sortedY, sortedY + n, compareY);
        ans = closest_pair(arr, sortedY, n);
        ans = sqrt(ans);
        if(ans - 10000.0 > 1e-7)
            printf("INFINITY\n");
        else
            printf("%.4lf\n", ans);
    }
    return 0;
}
```

10. CONVEX HULL

```
#define REMOVE_REDUNDANT

typedef double T;
const T EPS = 1e-7;
struct PT
{
    T x, y;
    PT() {}
    PT(T x, T y) : x(x), y(y) {}
    bool operator<(const PT &rhs) const
    {
        return make_pair(y,x) < make_pair(rhs.y,rhs.x);
    }
    bool operator==(const PT &rhs) const
    {
        return make_pair(y,x) == make_pair(rhs.y,rhs.x);
    }
};

T cross(PT p, PT q)
{
    return p.x*q.y-p.y*q.x;
}
T area2(PT a, PT b, PT c)
```

```
{
    return cross(a,b) + cross(b,c) + cross(c,a);
}

#ifdef REMOVE_REDUNDANT
bool between(const PT &a, const PT &b, const PT &c)
{
    return (fabs(area2(a,b,c)) < EPS && (a.x-b.x)*(c.x-
b.x) <= 0 && (a.y-b.y)*(c.y-b.y) <= 0);
}
#endif

void ConvexHull(vector<PT> &pts)
{
    sort(pts.begin(), pts.end());
    pts.erase(unique(pts.begin(), pts.end()),
pts.end());
    vector<PT> up, dn;
    for (int i = 0; i < pts.size(); i++)
    {
        while (up.size() > 1 && area2(up[up.size()-2],
up.back(), pts[i]) >= 0) up.pop_back();
        while (dn.size() > 1 && area2(dn[dn.size()-2],
dn.back(), pts[i]) <= 0) dn.pop_back();
        up.push_back(pts[i]);
        dn.push_back(pts[i]);
    }
    pts = dn;
    for (int i = (int) up.size() - 2; i >= 1; i--)
pts.push_back(up[i]);

#ifdef REMOVE_REDUNDANT
    if (pts.size() <= 2) return;
    dn.clear();
    dn.push_back(pts[0]);
    dn.push_back(pts[1]);
    for (int i = 2; i < pts.size(); i++)
    {
        if (between(dn[dn.size()-2], dn[dn.size()-1],
pts[i])) dn.pop_back();
        dn.push_back(pts[i]);
    }
    if (dn.size() >= 3 && between(dn.back(), dn[0],
dn[1]))
    {
        dn[0] = dn.back();
        dn.pop_back();
    }
    pts = dn;
#endif
}
```

11. CONVEX HULL TRICK

Recurrence: dp[i] = min(j<i) {dp[j]+b[j]*a[i]}
Condition: b[j] ≥ b[j + 1], a[i] ≤ a[i + 1]
Complexity: O(n^2) -> O(n)

Problem: The manager of logging factory wants them to go to the jungle and cut n trees with heights $a_1, a_2, \ldots, a_n$. They bought a chain saw from a shop. Each time they use the chain saw on the tree number i, they can decrease the height of this tree by one unit. Each time that Kalila and Dimna use the chain saw, they need to recharge it. Cost of charging depends on the id of the trees which have been cut completely (a tree is cut completely if its height equal to 0). If the maximum id of a tree which has been cut completely is i (the tree that have height a_i in the beginning), then the cost of charging the chain saw would be b_i. If no tree is cut completely, Kalila and Dimna cannot charge the chain saw. The chainsaw is charged in the beginning. We know that for each $i < j$, $a_i < a_j$ and $b_i > b_j$ and also $b_n = 0$and $a_1 = 1$. Kalila and Dimna want to cut all the trees completely, with minimum cost.

```
Output: The only line of output must contain the minimum
cost of cutting all the trees completely.
int pointer; //Keeps track of the best line from
previous query
vector<long long> M; //Holds the slopes of the lines in
the envelope, aka M[i]
vector<long long> B; //Holds the y-intercepts of the
lines in the envelope, aka C[i]
//Returns true if either line l1 or line l3 is always
better than line l2
bool bad(int l1,int l2,int l3)
{
      /*
      intersection(l1,l2) has x-coordinate (b1-b2)/(m2-
m1)
      intersection(l1,l3) has x-coordinate (b1-b3)/(m3-
m1)
      set the former greater than the latter, and cross-
multiply to
      eliminate division
      */
      return 1.0 * (B[l3]-B[l1])*(M[l1]-M[l2])< 1.0 *
(B[l2]-B[l1])*(M[l1]-M[l3]); // must check overflow
}
//Adds a new line (with lowest slope) to the structure
void add(long long m,long long b)
{
      //First, let's add it to the end
      M.push_back(m);
      B.push_back(b);
      //If the penultimate is now made irrelevant
between the antepenultimate
      //and the ultimate, remove it. Repeat as many
times as necessary
      while (M.size()>=3&&bad(M.size()-3,M.size()-
2,M.size()-1))
      {
            M.erase(M.end()-2);
            B.erase(B.end()-2);
      }
}
//Returns the minimum y-coordinate of any intersection
between a given vertical
//line and the lower envelope
long long query(long long x)
{
      //If we removed what was the best line for the
previous query, then the
      //newly inserted line is now the best for that
query
      if (pointer>=M.size())
            pointer=M.size()-1;
      //Any better line must be to the right, since
query values are
      //non-decreasing
      while (pointer<M.size()-1&&

M[pointer+1]*x+B[pointer+1]<M[pointer]*x+B[pointer])
            pointer++;
      return M[pointer]*x+B[pointer];
}
LL a[MAX], b[MAX], dp[MAX];
int main()
{
    int n, i;
    cin >> n;
    for(i = 1; i <= n; i++)
        cin >> a[i];
    for(i = 1; i <= n; i++)
        cin >> b[i];
    add(b[1], 0);
    for(i = 2; i <= n; i++)
    {
        dp[i] = query(a[i]);
        add(b[i], dp[i]);
    }
    cout << dp[n] << endl;
    return 0;
}
```

12. DISJOINT SET

```
int root(int v)
```

```
{
    return par[v] < 0 ? v : (par[v] = root(par[v]));
}
void merge(int x,int y)       //    x and y are some
tools (vertices)
{
    x = root(x), y = root(y);
    if(par[y] < par[x]) // balancing the height of the
tree
        swap(x, y);
    par[x] += par[y];
    par[y] = x;
}
```

13. DIVIDE & CONQUER

```
Recurrence: dp[i][j] = min(k<j) {dp[i-1][k]+C[k][j]}
Condition: A[i][j] <=A[i][j+1]
Complexity: O(kn^2) -> O(knlgn)

/**
Define P(g,l) as the lowest position k that minimizes
dp(g,l) ,
i.e. P(g,l) is the lowest k such that
dp(g,l)=dp(g−1,k)+Cost(k+1,l)
P(g,0)≤P(g,1)≤P(g,2)≤…≤P(g,L−1)≤P(g,L)
**/

LL dp[801][MAX], C[MAX], L, cum[MAX], P[801][MAX];

LL getCost(int l, int r)
{
    if(l > r) return 0;
    return (cum[r] - cum[l-1])*(r-l+1);
}

void call(int g, int l1, int l2, int p1, int p2)
{
    if(l1 > l2) return;
    int m = l1+l2 >> 1, i;
    dp[g][m] = 1LL<<60;
    for(i = p1; i <= p2; i++)
    {
        LL cur = dp[g-1][i] + getCost(i+1, m);
        if(cur < dp[g][m])
        {
            dp[g][m] = cur;
            P[g][m] = i;
        }
    }
    call(g, l1, m-1, p1, P[g][m]);
    call(g, m+1, l2, P[g][m], p2);
}

int main()
{
    int G, i;
    cin >> L >> G;
    for(i = 1; i <= L; i++)
        cin >> C[i], cum[i] = C[i]+cum[i-1];
    for(i = 1; i <= L; i++)
    {
        dp[1][i] = getCost(1, i);
        P[1][i] = 1;
    }
    for(i = 2; i <= G; i++)
        call(i, 1, L, 1, L);
    cout << dp[G][L] << "\n";
    return 0;
}
```

14. FFT

```
#include <cassert>
#include <cstdio>
#include <cmath>

struct cpx
{
  cpx(){}
  cpx(double aa):a(aa),b(0){}
  cpx(double aa, double bb):a(aa),b(bb){}
  double a;
  double b;
```

```
  double modsq(void) const
  {
    return a * a + b * b;
  }
  cpx bar(void) const
  {
    return cpx(a, -b);
  }
};

cpx operator +(cpx a, cpx b)
{
  return cpx(a.a + b.a, a.b + b.b);
}

cpx operator *(cpx a, cpx b)
{
  return cpx(a.a * b.a - a.b * b.b, a.a * b.b + a.b *
b.a);
}

cpx operator /(cpx a, cpx b)
{
  cpx r = a * b.bar();
  return cpx(r.a / b.modsq(), r.b / b.modsq());
}

cpx EXP(double theta)
{
  return cpx(cos(theta),sin(theta));
}

const double two_pi = 4 * acos(0);

// in:     input array
// out:    output array
// step:   {SET TO 1} (used internally)
// size:   length of the input/output {MUST BE A POWER
OF 2}
// dir:    either plus or minus one (direction of the
FFT)
// RESULT: out[k] = \sum_{j=0}^{size - 1} in[j] *
exp(dir * 2pi * i * j * k / size)
void FFT(cpx *in, cpx *out, int step, int size, int dir)
{
  if(size < 1) return;
  if(size == 1)
  {
    out[0] = in[0];
    return;
  }
  FFT(in, out, step * 2, size / 2, dir);
  FFT(in + step, out + size / 2, step * 2, size / 2,
dir);
  for(int i = 0 ; i < size / 2 ; i++)
  {
    cpx even = out[i];
    cpx odd = out[i + size / 2];
    out[i] = even + EXP(dir * two_pi * i / size) * odd;
    out[i + size / 2] = even + EXP(dir * two_pi * (i +
size / 2) / size) * odd;
  }
}

// Usage:
// f[0...N-1] and g[0..N-1] are numbers
// Want to compute the convolution h, defined by
// h[n] = sum of f[k]g[n-k] (k = 0, ..., N-1).
// Here, the index is cyclic; f[-1] = f[N-1], f[-2] =
f[N-2], etc.
// Let F[0...N-1] be FFT(f), and similarly, define G and
H.
// The convolution theorem says H[n] = F[n]G[n]
(element-wise product).
// To compute h[] in O(N log N) time, do the following:
//   1. Compute F and G (pass dir = 1 as the argument).
//   2. Get H by element-wise multiplying F and G.
//   3. Get h by taking the inverse FFT (use dir = -1 as
the argument)
//      and *dividing by N*. DO NOT FORGET THIS SCALING
FACTOR.
```

```
int main(void)
{
  printf("If rows come in identical pairs, then
everything works.\n");

  cpx a[8] = {0, 1, cpx(1,3), cpx(0,5), 1, 0, 2, 0};
  cpx b[8] = {1, cpx(0,-2), cpx(0,1), 3, -1, -3, 1, -2};
  cpx A[8];
  cpx B[8];
  FFT(a, A, 1, 8, 1);
  FFT(b, B, 1, 8, 1);

  for(int i = 0 ; i < 8 ; i++)
  {
    printf("%7.2lf%7.2lf", A[i].a, A[i].b);
  }
  printf("\n");
  for(int i = 0 ; i < 8 ; i++)
  {
    cpx Ai(0,0);
    for(int j = 0 ; j < 8 ; j++)
    {
      Ai = Ai + a[j] * EXP(j * i * two_pi / 8);
    }
    printf("%7.2lf%7.2lf", Ai.a, Ai.b);
  }
  printf("\n");

  cpx AB[8];
  for(int i = 0 ; i < 8 ; i++)
    AB[i] = A[i] * B[i];
  cpx aconvb[8];
  FFT(AB, aconvb, 1, 8, -1);
  for(int i = 0 ; i < 8 ; i++)
    aconvb[i] = aconvb[i] / 8;
  for(int i = 0 ; i < 8 ; i++)
  {
    printf("%7.2lf%7.2lf", aconvb[i].a, aconvb[i].b);
  }
  printf("\n");
  for(int i = 0 ; i < 8 ; i++)
  {
    cpx aconvbi(0,0);
    for(int j = 0 ; j < 8 ; j++)
    {
      aconvbi = aconvbi + a[j] * b[(8 + i - j) % 8];
    }
    printf("%7.2lf%7.2lf", aconvbi.a, aconvbi.b);
  }
  printf("\n");

  return 0;
}
```

15. FLOW

DINIC

```
// Running time: O(|V|^2 |E|)  OUTPUT: - maximum flow
value
// To obtain the actual flow values, look at all edges
with
// capacity > 0 (zero capacity edges are residual
edges).
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) // source, sink
    {
        long long totflow = 0;
        while (long long flow = BlockingFlow(s, t))
            totflow += flow;
        return totflow;
    }
};
```

MIN-COST

```
typedef vector<int> VI;
typedef vector<VI> VVI;
typedef long long L;
typedef vector<L> VL;
typedef vector<VL> VVL;
typedef pair<int, int> PII;
typedef vector<PII> VPII;

const L INF = (1LL << 60);

struct MinCostMaxFlow
{
    int N;
    VVL cap, flow, cost;
    VI found;
    VL dist, pi, width;
    VPII dad;

    MinCostMaxFlow(int N) :
        N(N), cap(N, VL(N)), flow(N, VL(N)), cost(N,
VL(N)),
        found(N), dist(N), pi(N), width(N), dad(N) {}

    void AddEdge(int from, int to, L cap, L cost)
    {
        this->cap[from][to] = cap;
        this->cost[from][to] = cost;
```

```
    }

    void Relax(int s, int k, L cap, L cost, int dir)
    {
        L val = dist[s] + pi[s] - pi[k] + cost;
        if (cap && val < dist[k])
        {
            dist[k] = val;
            dad[k] = make_pair(s, dir);
            width[k] = min(cap, width[s]);
        }
    }

    L Dijkstra(int s, int t)
    {
        fill(found.begin(), found.end(), false);
        fill(dist.begin(), dist.end(), INF);
        fill(width.begin(), width.end(), 0);
        dist[s] = 0;
        width[s] = INF;

        while (s != -1)
        {
            int best = -1;
            found[s] = true;
            for (int k = 0; k < N; k++)
            {
                if (found[k]) continue;
                Relax(s, k, cap[s][k] - flow[s][k],
cost[s][k], 1);
                Relax(s, k, flow[k][s], -cost[k][s], -
1);
                if (best == -1 || dist[k] < dist[best])
best = k;
            }
            s = best;
        }

        for (int k = 0; k < N; k++)
            pi[k] = min(pi[k] + dist[k], INF);
        return width[t];
    }

    pair<L, L> GetMaxFlow(int s, int t)
    {
        L totflow = 0, totcost = 0;
        while (L amt = Dijkstra(s, t))
        {
            totflow += amt;
            for (int x = t; x != s; x = dad[x].first)
            {
                if (dad[x].second == 1)
                {
                    flow[dad[x].first][x] += amt;
                    totcost += amt *
cost[dad[x].first][x];
                }
                else
                {
                    flow[x][dad[x].first] -= amt;
                    totcost -= amt *
cost[x][dad[x].first];
                }
            }
        }
        return make_pair(totflow, totcost);
    }
};
```

16. FORMULA

Formula for Arithmetic Series:

→n-th term, $x_n = a+(n-1)d$

→Summation of first n term, $S_n = n\{2a + (n-1)d\}/2$

→Summation of first n odd terms = n^2

→$1+2+3+\ldots+n = n(n+1)/2$

→$1^2+2^2+3^2+\ldots+n^2 = n(n+1)(2n+1)/6$

→$1^3+2^3+3^3+\ldots+n^3 = \{n(n+1)/2\}^2$

Formula for Geometric Series:

→n-th term, $x_n = ar^{(n-1)}$

→Summation of first n term, $S_n = a/(1-r)$, (while n tens to inf)/ $S_n = a(1-r^n)/(1-r)$, (while(r<1)/ $S_n = a(r^n-1)/(r-1)$, (while(r>1)

Formula for Permutation and Combination:

→$^nC_r = n! / r!(n-r)!$

→$^nP_r = n! / (n-r)!$

→$^nP_r = n!\ ^nC_r$

→ $^nC_r + {}^nC_{r-1} = {}^{n+1}C_r$

Formula for Cubic Geometry:

→Area of regular polygon = $(1/2)\ n \sin(360°/n)\ S^2$ when n = # of sides and S = length from center to a corner

→angle = $(n-2)*180^o$

Formula for Triangle:

→Area of an equilateral triangle = $(c/2)*\sqrt{(a^2 - (c/2)^2)}$.

→Area of an isosceles triangle = $(a^2 * \sqrt{3})/4$,

→Law of cosine: $c^2=a^2+b^2 - 2ab \cos C$.

→Sin (A/2) = $\sqrt{((s-b)*(s-c)/bc)}$, Cos (A/2) = $\sqrt{(s*(s-a)/bc)}$

length of median to side c = sqrt(2*(a*a+b*b)-c*c)/2

length of bisector of angle C = sqrt(ab[(a+b)*(a+b)-c*c])/(a+b)

Formula of Straight Line:

→Slope from point (x_1, y_1) to point $(x_2, y_2) = (y_2 - y_1)/(x_2 - x_1)$

→Equation for a straight line going through point (x_1, y_1) to point (x_2, y_2):

$(y - y_1) = m(x - x_1)$

→Absolute distance from point (x_1, y_1) to straight line $ax + by + c = 0$:

$|((ax_1 + by_1 + c)/\sqrt{(a^2+b^2)})|$

→Angles between two straight lines $y = m_1x + c_1$ and $y = m_2x + c_2$ is $\tan^{-1}((m_1-m_2)/(1+m_1m_2))$

Formula of Circle:

→ Equation of a circle centering (h,k) with radius r: $(x-h)^2+(y-k)^2=r^2$

→ Equation of a circle having diameter from point (x_1, y_1) to point (x_2, y_2): $(x-x_1)(x-x_2) + (y-y_1)(y-y_2) = 0$

→Equation of a circle going through the point where the straight line lx+my+n=0 and the circle $x^2+y^2+2gx+2fy+c=0$ cross each-other: $x^2+y^2+2gx+2fy+c +k(lx+my+n)=0$

→Equation of a circle going through the point where the circle $x^2+y^2+2g_1x+2f_1y+c=0$ cross each-other: and the circle $x^2+y^2+2g_2x+2f_2y+c=0$ cross each-other: $x^2+y^2+2g_1x+2f_1y+c + k(x^2+y^2+2g_2x+2f_2y+c)=0$

→ Equation of a circle centering (-g,-f) with radius $\sqrt{(g^2+f^2-c)}$: $x^2+y^2+2gx+2fy+c=0$

Formula of Probability:

→P(an event) = # of probable event in favor / total # of probable event

→P(A U B) = P(A) + P(B) – P(A ∩ B)

→ P(A ∩ B) = P(A) * P(B) for independent event, → P(A ∩ B) = P(A) * P(B/A) for dependent event

To use bits as flag:

```
int arr[MAX/32 + 5];
#define get(x) ((arr[(x)>>5]>>((x)&31))&1)
#define set(x) arr[(x)>>5]|=(1<<((x)&31));
```

have to print the digit(s) of factorial n in the given base:

```
for(n = 1; n < MAX; n++)
    arr[n] = log10(n) + arr[n - 1];
scanf("%d %d", &n, &b);
cout << (int) (arr[n] / log10(b) + 1) << endl;
```

Dijkstra:

```
struct pq{ int cost,node;
    bool operator<(const pq &b)const
    {
        return cost>b.cost; // Min Priority Queue(b is curret)
    }};
```

17. FRACTION

```
int _lcm(int a,int b)
{
    return a/__gcd(a,b)*b;
}

struct Fraction
{
    int num,dnm,gcd;

    void normalize()
    {
```

```
        // numerator and denominator must be co-prime.
        gcd = __gcd(num,dnm);
        num/=gcd;dnm/=gcd;
        // negative sing is always with numerator.
        num = (num*dnm<0?-num:num);
        dnm = (dnm<0?-dnm:dnm);
    }

    Fraction (int a,int b)
    {
        num = a,dnm = b;
        normalize();
    }

    void operator=(const Fraction  &other)
    {
        num = other.num;
        dnm = other.dnm;
        normalize();
    }

    bool operator<(Fraction  other) const
    {
        int lcm = _lcm(dnm,other.dnm);
        return (lcm/dnm)*num <
(lcm/other.dnm)*other.num;
//        return num*other.dnm < dnm*other.num;// skip
to reduce overflow.
    }

    bool operator>(Fraction  other) const
    {
        int lcm = _lcm(dnm,other.dnm);
        return (lcm/dnm)*num >
(lcm/other.dnm)*other.num;
    }

    bool operator==(Fraction  other) const
    {
        int lcm = _lcm(dnm,other.dnm);
        return (lcm/dnm)*num ==
(lcm/other.dnm)*other.num;
    }

    bool operator!=(Fraction  other) const
    {
        int lcm = _lcm(dnm,other.dnm);
        return (lcm/dnm)*num !=
(lcm/other.dnm)*other.num;
    }

    bool operator>=(Fraction  other) const
    {
        int lcm = _lcm(dnm,other.dnm);
        return (lcm/dnm)*num >=
(lcm/other.dnm)*other.num;
    }

    bool operator<=(Fraction  other) const
    {
        int lcm = _lcm(dnm,other.dnm);
        return (lcm/dnm)*num <=
(lcm/other.dnm)*other.num;
    }

    Fraction operator+(Fraction  other) const
    {
        int lcm = _lcm(dnm,other.dnm);
        return Fraction((lcm/dnm)*num +
(lcm/other.dnm)*other.num,lcm);
    }

    Fraction operator-(Fraction  other) const
    {
        int lcm = _lcm(dnm,other.dnm);
        return Fraction((lcm/dnm)*num -
(lcm/other.dnm)*other.num,lcm);
    }

    Fraction operator*(Fraction  other) const
    {
```

```
        return Fraction(num*other.num,dnm*other.dnm);
    }

    Fraction operator/(Fraction  other) const
    {
        return Fraction(num*other.dnm,dnm*other.num);
    }

    Fraction operator- ()
    {
        return Fraction(-num,dnm);
    }

    string to_string()
    {
        ostringstream oss;
        oss<<num<<"/"<<dnm;
        oss.flush();
        return oss.str();
    }

    double to_double()
    {
        return double(num)/double(dnm);
    }

};

int main()
{
    cout<<(Fraction(1,2)==Fraction(5,10))<<endl;
    return 0;
}
```

18. GEOMETRY

```
double INF = 1e100;
double EPS = 1e-12;
struct PT {
  double x, y; PT() {}
  PT(double x, double y) : x(x), y(y) {}
  PT(const PT &p) : x(p.x), y(p.y)    {}
  PT operator + (const PT &p)  const { return PT(x+p.x,
y+p.y); }
  PT operator - (const PT &p)  const { return PT(x-p.x,
y-p.y); }
  PT operator * (double c)     const { return PT(x*c,
y*c  ); }
  PT operator / (double c)     const { return PT(x/c,
y/c  ); }
};
double dot(PT p, PT q)     { return p.x*q.x+p.y*q.y; }
double dist2(PT p, PT q)   { return dot(p-q,p-q); }
double cross(PT p, PT q)   { return p.x*q.y-p.y*q.x; }
ostream &operator<<(ostream &os, const PT &p) {
  os << "(" << p.x << "," << p.y << ")";
}
// rotate a point CCW or CW around the origin
PT RotateCCW90(PT p)   { return PT(-p.y,p.x); }
PT RotateCW90(PT p)    { return PT(p.y,-p.x); }
PT RotateCCW(PT p, double t) {
  return PT(p.x*cos(t)-p.y*sin(t),
p.x*sin(t)+p.y*cos(t));
}
// project point c onto line through a and b
// assuming a != b
PT ProjectPointLine(PT a, PT b, PT c) {
  return a + (b-a)*dot(c-a, b-a)/dot(b-a, b-a);
}
// project point c onto line segment through a and b
PT ProjectPointSegment(PT a, PT b, PT c) {
  double r = dot(b-a,b-a);
  if (fabs(r) < EPS) return a;
  r = dot(c-a, b-a)/r;
  if (r < 0) return a;
  if (r > 1) return b;
  return a + (b-a)*r;
}
// compute distance from c to segment between a and b
double DistancePointSegment(PT a, PT b, PT c) {
  return sqrt(dist2(c, ProjectPointSegment(a, b, c)));
}
```

```
// compute distance between point (x,y,z) and plane
ax+by+cz=d
double DistancePointPlane(double x, double y, double z,
    double a, double b, double c, double d){
  return fabs(a*x+b*y+c*z-d)/sqrt(a*a+b*b+c*c);
}
// determine if lines from a to b and c to d are
parallel or collinear
bool LinesParallel(PT a, PT b, PT c, PT d) {
  return fabs(cross(b-a, c-d)) < EPS;
}
bool LinesCollinear(PT a, PT b, PT c, PT d) {
  return LinesParallel(a, b, c, d)
      && fabs(cross(a-b, a-c)) < EPS
      && fabs(cross(c-d, c-a)) < EPS;
}
// determine if line segment from a to b intersects with
// line segment from c to d
bool SegmentsIntersect(PT a, PT b, PT c, PT d) {
  if (LinesCollinear(a, b, c, d)) {
    if (dist2(a, c) < EPS || dist2(a, d) < EPS ||
      dist2(b, c) < EPS || dist2(b, d) < EPS) return
true;
    if (dot(c-a, c-b) > 0 && dot(d-a, d-b) > 0 && dot(c-
b, d-b) > 0)
      return false;
    return true;
  }
  if (cross(d-a, b-a) * cross(c-a, b-a) > 0) return
false;
  if (cross(a-c, d-c) * cross(b-c, d-c) > 0) return
false;
  return true;
}
// compute intersection of line passing through a and b
// with line passing through c and d, assuming that
unique
// intersection exists; for segment intersection, check
if
// segments intersect first
PT ComputeLineIntersection(PT a, PT b, PT c, PT d) {
  b=b-a; d=c-d; c=c-a;
  assert(dot(b, b) > EPS && dot(d, d) > EPS);
  return a + b*cross(c, d)/cross(b, d);
}
// compute center of circle given three points
PT ComputeCircleCenter(PT a, PT b, PT c) {
  b=(a+b)/2;
  c=(a+c)/2;
  return ComputeLineIntersection(b, b+RotateCW90(a-b),
c, c+RotateCW90(a-c));
}
// determine if point is in a possibly non-convex
polygon (by William
// Randolph Franklin); returns 1 for strictly interior
points, 0 for
// strictly exterior points, and 0 or 1 for the
remaining points.
// Note that it is possible to convert this into an
*exact* test using
// integer arithmetic by taking care of the division
appropriately
// (making sure to deal with signs properly) and then by
writing exact
// tests for checking point on polygon boundary
bool PointInPolygon(const vector<PT> &p, PT q) {
  bool c = 0;
  for (int i = 0; i < p.size(); i++){
    int j = (i+1)%p.size();
    if ((p[i].y <= q.y && q.y < p[j].y ||
      p[j].y <= q.y && q.y < p[i].y) &&
      q.x < p[i].x + (p[j].x - p[i].x) * (q.y - p[i].y)
/ (p[j].y - p[i].y))
      c = !c;
  }
  return c;
}
// determine if point is on the boundary of a polygon
bool PointOnPolygon(const vector<PT> &p, PT q) {
  for (int i = 0; i < p.size(); i++)
    if (dist2(ProjectPointSegment(p[i],
p[(i+1)%p.size()], q), q) < EPS)
```

```
      return true;
    return false;
}
// compute intersection of line through points a and b
with
// circle centered at c with radius r > 0
vector<PT> CircleLineIntersection(PT a, PT b, PT c,
double r) {
  vector<PT> ret;
  b = b-a;
  a = a-c;
  double A = dot(b, b);
  double B = dot(a, b);
  double C = dot(a, a) - r*r;
  double D = B*B - A*C;
  if (D < -EPS) return ret;
  ret.push_back(c+a+b*(-B+sqrt(D+EPS))/A);
  if (D > EPS)
    ret.push_back(c+a+b*(-B-sqrt(D))/A);
  return ret;
}
// compute intersection of circle centered at a with
radius r
// with circle centered at b with radius R
vector<PT> CircleCircleIntersection(PT a, PT b, double
r, double R) {
  vector<PT> ret;
  double d = sqrt(dist2(a, b));
  if (d > r+R || d+min(r, R) < max(r, R)) return ret;
  double x = (d*d-R*R+r*r)/(2*d);
  double y = sqrt(r*r-x*x);
  PT v = (b-a)/d;
  ret.push_back(a+v*x + RotateCCW90(v)*y);
  if (y > 0)
    ret.push_back(a+v*x - RotateCCW90(v)*y);
  return ret;
}
// This code computes the area or centroid of a
(possibly nonconvex)
// polygon, assuming that the coordinates are listed in
a clockwise or
// counterclockwise fashion.  Note that the centroid is
often known as
// the "center of gravity" or "center of mass".
double ComputeSignedArea(const vector<PT> &p) {
  double area = 0;
  for(int i = 0; i < p.size(); i++) {
    int j = (i+1) % p.size();
    area += p[i].x*p[j].y - p[j].x*p[i].y;
  }
  return area / 2.0;
}
double ComputeArea(const vector<PT> &p) {
  return fabs(ComputeSignedArea(p));
}
PT ComputeCentroid(const vector<PT> &p) {
  PT c(0,0);
  double scale = 6.0 * ComputeSignedArea(p);
  for (int i = 0; i < p.size(); i++){
    int j = (i+1) % p.size();
    c = c + (p[i]+p[j])*(p[i].x*p[j].y - p[j].x*p[i].y);
  }
  return c / scale;
}
// tests whether or not a given polygon (in CW or CCW
order) is simple
bool IsSimple(const vector<PT> &p) {
  for (int i = 0; i < p.size(); i++) {
    for (int k = i+1; k < p.size(); k++) {
      int j = (i+1) % p.size();
      int l = (k+1) % p.size();
      if (i == l || j == k) continue;
      if (SegmentsIntersect(p[i], p[j], p[k], p[l]))
        return false;
    }
  }
  return true;
}
```

19. HLD

```
///spoj QTREE
/*
```

```
given weight of edges. query on max edge and update on
singe edge
*/

#define root 0
#define LN 16 ///log2(MAX)

vector <int> adj[MAX], costs[MAX];
int arr[MAX], now;
int chain_no, chain_indx[MAX], chain_head[MAX],
pos_in_arr[MAX];
int depth[MAX], par[MAX][LN], child[MAX];
int segtree[MAX*4];
struct edge{
    int u, v, c;
    edge(){}
    edge(int uu, int vv, int cc)
    {
        u = uu;
        v = vv;
        c = cc;
    }
};
vector <edge> edges;

void build(int s, int e, int cur);

void update(int s, int e, int cur, int indx, int val);

int query_tree(int s, int e, int cur, int L, int R)
{
    if(s > R || e < L)
        return 0;
    if(s >= L && e <= R)
        return segtree[cur];
    int lchild = (cur << 1), rchild = lchild | 1, m = (s
+ e) >> 1;
    return max(query_tree(s, m, lchild, L, R),
                query_tree(m+1, e, rchild, L, R));
}
/*
 * query_up:
 * It takes two nodes u and v, condition is that v is an
ancestor of u
 * We query the chain in which u is present till chain
head, then move to next chain up
 * We do that way till u and v are in the same chain, we
query for that part of chain and break
 */
int query_up(int u, int v)
{
    if(u == v) return 0;
    int uchain, vchain = chain_indx[v], ans = -1;
    while(true)
    {
        uchain = chain_indx[u];
        if(uchain == vchain)
        {
            if(u == v) break;
            ans = max(ans, query_tree(0, now, 1,
pos_in_arr[v]+1, pos_in_arr[u]));
            break;
        }
        u = chain_head[uchain];
        ans = max(ans, query_tree(0, now, 1,
pos_in_arr[u], pos_in_arr[u]));
        u = par[u][0];
    }
    return ans;
}

void process(int n);

int LCA(int high, int low);

int query(int u, int v)
{
    int lca = LCA(u, v);
    return max(query_up(u, lca), query_up(v, lca));
}

void change(int i, int val)
```

```
{
    edge tem = edges[i];
    int pos = pos_in_arr[tem.u];
    if(depth[tem.u] < depth[tem.v])
        pos = pos_in_arr[tem.v];
    update(0, now, 1, pos, val);
}

void HLD(int cur_node, int cost, int prev)
{
    if(chain_head[chain_no] == -1)
        chain_head[chain_no] = cur_node;
    chain_indx[cur_node] = chain_no;
    pos_in_arr[cur_node] = now;
    arr[now++] = cost;
    int schild = -1, ncost, i, v;
    for(i = 0; i < SZ(adj[cur_node]); i++)
    {
        v = adj[cur_node][i];
        if(v == prev) continue;
        if(schild == -1 || child[schild] < child[v])
        {
            schild = v;
            ncost = costs[cur_node][i];
        }
    }
    if(schild != -1)
        HLD(schild, ncost, cur_node);
    for(i = 0; i < SZ(adj[cur_node]); i++)
    {
        v = adj[cur_node][i];
        if(v == prev) continue;
        if(schild != v)
        {
            chain_no++;
            HLD(v, costs[cur_node][i], cur_node);
        }
    }
}

void dfs(int cur, int prev, int d)
{
    par[cur][0] = prev;
    depth[cur] = d;
    child[cur] = 1;
    for(int i = 0; i < SZ(adj[cur]); i++)
    {
        int v = adj[cur][i];
        if(v == prev) continue;
        dfs(v, cur, d + 1);
        child[cur] += child[v];
    }
}

void init(int n)
{
    now = chain_no = 0;
    edges.clear();
    for(int i = 0; i < n; i++)
    {
        adj[i].clear();
        costs[i].clear();
        chain_head[i] = -1;
        for(int j = 0; j < LN; j++)
            par[i][j] = -1;
    }
}

int main()
{
    int test, n, i, u, v, c;
    char str[10];
    scanf("%d", &test);
    while(test--)
    {
        scanf("%d", &n);
        init(n);
        for(i = 1; i < n; i++)
        {
            scanf("%d %d %d", &u, &v, &c);
            u--;
            v--;
```

```
            adj[u].pb(v);
            adj[v].pb(u);
            costs[u].pb(c);
            costs[v].pb(c);
            edges.pb(edge(u, v, c));
        }
        dfs(root, -1, 0);
        HLD(root, -1, -1);
        build(0, now, 1);
        process(n);
        while(scanf(" %s", str) == 1 && str[0] != 'D')
        {
            scanf("%d %d", &u, &v);
            if(str[0] == 'Q')
                printf("%d\n", query(u-1, v-1));
            else
                change(u-1, v);
        }
    }
    return 0;
}
```

20. JOSEPHUS

```
int find_survivour(int n, int k)
{
    if(n == 1)
        return 1;
    if(dp[n][k])
        return dp[n][k];
    return dp[n][k] = ((find_survivour(n - 1, k) + k -
1) % n) + 1;
}
```

21. KMP

```
void compute_match_array(string &pat)
{
    int m = SZ(pat);
    int len = 0;
    int i;
    match[0] = 0, i = 1;
    // calculate match[i] for i = 1 to m - 1
    while(i < m)
    {
        if(pat[i] == pat[len])
        {
            len++;
            match[i] = len;
            i++;
        }
        else
        {
            if(len != 0)
                len = match[len - 1];
            else
            {
                match[i] = 0;
                i++;
            }
        }
    }
}
```

22. KNUTH OPTIMIZATION

Recurrence: dp[i][j] = min(i<k<j) {dp[i][k] + dp[k][j]} + C[i][j]
Condition: A[i, j - 1] ≤ A[i, j] ≤ A[i + 1, j]
Complexity: O(n^3) -> O(n^2)
This problem is solved with DP over substrings. Let's enumerate all required breaks and two ends of string with numbers 0, 1, 2, ..., k. Then res[L,R] will be result for the substring which starts in L-th point and ends in R-th point. To get this result we should look through all possible middle points M and consider res[L][M] + res[M][R] + (x[R]-x[L]) as a result. By doing this we get a clear O(k^3) solution (which is TL). What makes this problem exceptional is the application of Knuth's optimization. This trick works only for optimization DP over substrings for which optimal middle point depends monotonously on the end points. Let mid[L,R] be the first middle point for (L,R) substring which gives optimal result. It can be proven that mid[L,R-1] <= mid[L,R] <= mid[L+1,R] - this means

```
monotonicity of mid by L and R. If you are interested in
a proof, read about optimal binary search trees in
Knuth's "The Art of Computer Programming" volume 3
binary search tree section.
Applying this optimization reduces time complexity from
O(k^3) to O(k^2) because with fixed s (substring length)
we have m_right(L) = mid[L+1][R] = m_left(L+1). That's
why nested L and M loops require not more than 2k
iterations overall.
  for (int s = 0; s<=k; s++)                    //s -
length(size) of substring
    for (int L = 0; L+s<=k; L++) {              //L -
left point
      int R = L + s;                            //R -
right point
      if (s < 2) {
        res[L][R] = 0;                          //DP
base - nothing to break
        mid[L][R] = l;                          //mid is
equal to left border
        continue;
      }
      int mleft = mid[L][R-1];
//Knuth's trick: getting bounds on M
      int mright = mid[L+1][R];
      res[L][R] = 1000000000000000000LL;
      for (int M = mleft; M<=mright; M++) {
//iterating for M in the bounds only
        int64 tres = res[L][M] + res[M][R] + (x[R]-
x[L]);
        if (res[L][R] > tres) {                 //relax
current solution
          res[L][R] = tres;
          mid[L][R] = M;
        }
      }
    }
  int64 answer = res[0][k];
```

23. LCA

```
void dfs(int node, int prev)
{
    dp[node][0] = prev;
    level[node] = prev == -1? 0 : level[prev]+1;
    for(int i = 0; i < SZ(adj[node]); i++)
        if(adj[node][i] != prev)
            dfs(adj[node][i], node);
}

void process(int n)
{
    dfs(1, -1);
    int i, lev;
    for(lev = 1; lev < LN; lev++)
    {
        for(i = 1; i <= n; i++)
        {
            if(dp[i][lev - 1] != -1)
                dp[i][lev] = dp[dp[i][lev - 1]][lev -
1];
        }
    }
}

int lca(int high, int low)
{
    if(level[low] < level[high]) swap(low, high);
    int i, diff;
    diff = level[low] - level[high];
    for(i = 0; i < LN; i++)
        if(diff & (1 << i))
            low = dp[low][i];
    if(low == high) return low;
    for(i = LN-1; i >= 0; i--)
    {
        if(dp[low][i] != -1 && dp[low][i] !=
dp[high][i])
        {
            low = dp[low][i];
            high = dp[high][i];
        }
    }
```

```
    return dp[low][0];
}
```

24. LIS

```
int sequence[100], I[101], L[100], lislength;

int input(void)
{
    int n, i;
    scanf("%d", &n);
    for(i = 0; i < n; i++)
        scanf("%d", &sequence[i]);
    return n;
}

int lis(int n)
{
    int i, low, high, mid;
    I[0] = -infinity;
    for(i = 1; i <= n; i++)
        I[i] = infinity;
    lislength = 0;
    for(i = 0; i < n; i++)
    {
        low = 0, high = lislength;
        while(low <= high)
        {
            mid = low + high >> 1;
            if(I[mid] < sequence[i])
                low = mid + 1;
            else
                high = mid - 1;
        }
        I[low] = sequence[i];
        L[i] = low;
        if(lislength < low)
            lislength = low;
    }
    return lislength;
}
```

```
void printseq(void)
{
    int pos, i, n, j, arr[lislength], val = lislength;
    for(i = 0; i < 10; i++)
        if(L[i] == lislength)
        {
            pos = i;
            arr[val - 1] = sequence[pos];
            val--;
            break;
        }
    for(i = pos; i >= 0; i--)
    {
        if(L[i] == val && sequence[pos] > sequence[i])
        {
            arr[val - 1] = sequence[i];
            val--;
            pos = i;
        }
    }
    for(i = 0; i < lislength; i++)
        cout << arr[i] << ' ';
}
```

25. LUCAS

```
///calculates nCr for biiiiiiiig N with modulus

LL mod = 1000003, F[MAX]; /// F[MAX] is factorial, it
needs to be pre-calculated here MAX = mod value

LL small(LL n, LL r)
{
    LL ret = bigmod(F[r], mod-2, mod) * bigmod(F[n-r],
mod-2, mod);
    ret %= mod;
    ret = (F[n] * ret) % mod;
    return ret;
}

LL Lucas(LL n, LL m)
{
```

```
    if (n==0 && m==0) return 1;
    LL ni = n % mod;
    LL mi = m % mod;
    if (mi>ni) return 0;
    return (Lucas(n/mod, m/mod) * small(ni, mi)) % mod;
}

void pre()
{
    LL i;
    F[0]=F[1]=1;
    for(i=2;i<MAX;i++)
        F[i]=(i * F[i-1])%mod;
}
```

26. MANACHER

```
string s, t;
char str[1000005];

void prepare_string()
{
    t = "^#";
    for(int i = 0; i < SZ(s); i++)
        t += s[i], t += "#";
    t += "$";
}

int manacher()
{
    prepare_string();
    int P[SZ(t)], c = 0, r = 0, i, i_mirror, n = SZ(t) -
1;
    for(i = 1; i < n; i++)
    {
        i_mirror = (2 * c) - i;
        P[i] = r > i? min(r - i, P[i_mirror]) : 0;
        while(i + 1 + P[i] < n && t[i + 1 + P[i]] == t[i
- 1 - P[i]]) P[i]++;
        if(i + P[i] > r)
        {
            c = i;
            r = i + P[i];
        }
    }
    return *max_element(P + 1, P + n);
}
```

27. MAT EXPO

```
struct matrix
{
    LL x[5][5];
};
matrix base, zero;
matrix matmult(matrix &a, matrix &b, int n)//m*n and n*r
matrix  //1 based
{
    matrix ret;
    int i,j,k;
    for(i = 1; i <= n; i++)
        for(j = 1; j <= n; j++)
        {
            ret.x[i][j]=0;
            for(k = 1; k <= n; k++)
                ret.x[i][j] = (ret.x[i][j] + (a.x[i][k]
* b.x[k][j]) % mod) % mod;
            ret.x[i][j]%=mod;
        }
    return ret;
}

matrix mat_expo(matrix b, long long p, int n) //have to
pass dimension - n
{
    if(!p) return b;
    matrix xx = zero;
    int i;
    for(i = 1; i <= n; i++)  xx.x[i][i] = 1;
    matrix power = b;
    while(p)
    {
        if((p & 1) == 1) xx = matmult(xx, power, n);
        power = matmult(power, power, n);
```

```
        p /= 2;
    }
    return xx;
}
```

28. MAX RECT IN HISTOGRAM

```
int hist[MAX];
stack <int> st;
int get_max_rec(int n)
{
    int i = 0, res = 0, tem, top;
    while(i < n)
    {
        if(st.empty() || hist[st.top()] <= hist[i])
            st.push(i++);
        else
        {
            top = st.top();
            st.pop();
            tem = hist[top] * (st.empty() ? i : i -
st.top() - 1);
            res = max(tem, res);
        }
    }
    while(!st.empty())
    {
        top = st.top();
        st.pop();
        tem = hist[top] * (st.empty()? i : i - st.top()
- 1);
        res = max(tem, res);
    }
    return res;
}
```

29. MAX SUM

2D

```
int max_sum(int n, int r)
{
    int i, j, m = 0, sum = 0;
    for(i = 1; i <= n; i++)
    {
        for(j = 1; j <= r; j++)
                arr[i][j] += arr[i][j - 1];
    }
    for(int c1 = 1; c1 <= r; c1++)
    {
        for(int c2 = c1; c2 <= r; c2++)
        {
            sum = 0;
            for(int r = 1; r <= n; r++)
            {
                sum += (arr[r][c2] -
arr[r][c1 - 1]);
                if(sum < 0)
                    sum = 0;
                else if(sum > m)
                    m = sum;
            }
        }
    }
    return m;
}
```

1D

```
info max_sum(int *data, int n)
{
    int start = 0, en = 0, tem = 0, i, sum = 0;
    info ret;
    ret.start = ret.en = ret.sum = 0;
    for(i = 0; i < n; i++)
    {
        sum += data[i];
        if(sum < 0)
        {
            tem = i + 1;
            sum = 0;
        }
        else if(sum > ret.sum)
        {
            ret.sum = sum;
```

```
            ret.start = tem;
            ret.en = i;
        }
    }
    return ret;
}
```

30. MOBIUS FUNCTION

```
#define s 1000010
bool col[s];
int mob[s]; //mobius function
/*
μ(n) = 1 if n is a square-free positive integer with an
even number of prime factors.
μ(n) = −1 if n is a square-free positive integer with an
odd number of prime factors.
μ(n) = 0 if n has a squared prime factor.
The Möbius function is multiplicative (i.e. μ(ab) =
μ(a)μ(b) whenever a and b are coprime).
The sum over all positive divisors of n of the Möbius
function is zero except when n = 1:
*/
int seive()//1 indexed
{
    long long i,j;
    col[0]=true;
    col[1]=true;
    fr(i,1,s-1) mob[i]=1;
    for(i=2;i<s;i++)
         if(!col[i])
         {
                mob[i]=-1;
                for(j=2*i;j<s;j+=i)
                {
                    col[j]=true;
                    if(j%(i*i)==0) mob[j]=0; //divisible
by square of prime
                    mob[j]*=-1;
                }
         }
    return 0;
}
```

31. MO'S ALGO

```
int res[MAX], freq[1000001], arr[30001], ans;

int block;

struct query{
    int no, l, r;
}Q[MAX];

bool comp(const query &a, const query &b)
{
    if(a.l/block == b.l/block)
        return a.r < b.r;
    return a.l/block < b.l/block;
}

void add(int val)
{
    freq[val]++;
    if(freq[val] == 1)
        ans++;
}

void remove(int val)
{
    freq[val]--;
    if(freq[val] == 0)
        ans--;
}

int main()
{
    int n, q, i;
    cin >> n;
    for(i = 0; i < n; i++)
        cin >> arr[i];
    cin >> q;
    for(i = 0; i < q; i++)
```

```
    {
        cin >> Q[i].l >> Q[i].r;
        Q[i].l--, Q[i].r--;
        Q[i].no = i;
    }
    block = sqrt(n);
    sort(Q, Q+q, comp);
    int Lp = 0, Rp = -1;
    for(i = 0; i < q; i++)
    {
        while(Lp < Q[i].l)
            remove(arr[Lp++]);
        while(Lp > Q[i].l)
            add(arr[--Lp]);
        while(Rp < Q[i].r)
            add(arr[++Rp]);
        while(Rp > Q[i].r)
            remove(arr[Rp--]);
        res[Q[i].no] = ans;
    }
    for(i = 0; i < q; i++)
        cout << res[i] << "\n";
    return 0;
}
```

32. MORE BITMASK

```
int more_bit[10];
int get_bit(int mask , int pos)
{
    return (mask / more_bit[pos]) % 3;
}
int set_bit(int mask, int pos , int bit)
{
    int tmp = (mask / more_bit[pos]) % 3;
    mask -= tmp * more_bit[pos];
    mask += bit * more_bit[pos];
    return mask;
}
void init(void)
{
    more_bit[0] = 3;
    for(int i = 1; i < 10; i++) more_bit[i] = 3 *
more_bit[i - 1];
}
```

33. NUMBER THEORY

E-GCD

```
// returns g = 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 ret;
      int g = extended_euclid(a, n, x, y);
      if (!(b%g)) {
            x = mod(x*(b / g), n);
            for (int i = 0; i < g; i++)
                  ret.push_back(mod(x + i*(n / g), n));
      }
      return ret;
}
```

MOD INVERSE

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

```
}
```

EULERS's FORMULA

If G is a connected plane graph with v vertices, e edges, and f faces, then
v – e + f = 1 + number of connected components.

GAUSS ELEMINATION

```
void gauss( int N, long double mat[NN][NN] )
{
    int i, j, k;
    for (i = 0; i < N; i++)
    {
        k = i;
        for (j = i+1; j < N; j++) if (fabs(mat[j][i]) >
fabs(mat[k][i])) k = j;
        if (k != i) for (j = 0; j <= N; j++)
swap(mat[k][j], mat[i][j]);
        for (j = i+1; j <= N; j++) mat[i][j] /=
mat[i][i];
        mat[i][i] = 1;
        for (k = 0; k < N; k++) if( k != i )
            {
                long double t = mat[k][i];
                if (t == 0.0L) continue;
                for (j = i; j <= N; j++) mat[k][j] -= t
* mat[i][j];
                mat[k][i] = 0.0L;
            }
    }
}
```

DIOPHANTINE

```
// computes x and y such that ax + by = c
// returns whether the solution exists
bool linear_diophantine(int a, int b, int c, int &x, int
&y) {
      if (!a && !b)
      {
            if (c) return false;
            x = 0; y = 0;
            return true;
      }
      if (!a)
      {
            if (c % b) return false;
            x = 0; y = c / b;
            return true;
      }
      if (!b)
      {
            if (c % a) return false;
            x = c / a; y = 0;
            return true;
      }
      int g = gcd(a, b);
      if (c % g) return false;
      x = c / g * mod_inverse(a / g, b / g);
      y = (c - a*x) / b;
      return true;
}
```

NUMBER OF DIVISORS

```
int nod[100000+5];
void Generate()
{
    nod[1]=1;
    for(int i=2; i<=100000; i++)
    {
        if(!nod[i])   //here checking i is prime or not
???
        {
            nod[i]=2;
            for(int j=i+i; j<=100000; j+=i)
            {
                if(!nod[j])nod[j]=1;
                int n=j,cnt=0;
                while(!(n%i))
                {
                    cnt++;
                    n/=i;
                }
                nod[j]*=(cnt+1);
            }
        }
```

```
    }
}
```

PHI

PHI

```
int phi[10000];
const int M=1000;
void Generate_phi()
{
    int i,j;
    phi[1]=1;
    for(i=2; i<M; i++)
    {
        if(!phi[i])
        {
            phi[i]=i-1;
            for(j=i+i; j<M; j+=I)
            {
                if(!phi[j])phi[j]=j;
                phi[j]=phi[j]/i*(i-1);
            }
        }
    }
}

int phi (int n)
{
    int ret = n;
    for (int i = 2; i * i <= n; i++)
    {
        if (n % i == 0)
        {
            while (n % i == 0)
            {
                n /= i;
            }
            ret -= ret / i;
        }
    }
    if (n > 1) ret -= ret / n;
    return ret;
}
```

PICK'S THEOREM

```
// Only for integer points
I = area + 1 – B/2
Where I = number of points inside
B = number of points on the border
```

SUM OF DIVISORS

```
int Sum_Of_Divisor(int N)
{
    int i,val,count,sum,p,s;
    val=sqrt(N)+1;
    sum=1;
    for(i=0; primes[i]<val; i++)
    {
        if(N%primes[i]==0)
        {
            p=1;
            while(N%primes[i]==0)
            {
                N/=primes[i];
                p=p*primes[i];
            }
            p=p*primes[i];
            s=(p-1)/(primes[i]-1);
            sum=sum*s;
        }
    }
    if(N>1)
    {
        p=N*N;
        s=(p-1)/(N-1);
        sum=sum*s;
    }
    return sum;
}
```

GENERATE ALL DIVISORS

```
void Generate(int cur,int num)
{
    int i,val;
    if(cur==Total_Prime)
    {
        store_divisor[ans++]=num;
```

```
    }
    else
    {
        val=1;
        for(i=0; i<=freq_primes[cur]; i++)
        {
            Generate(cur+1,num*val);
            val=val*store_primes[cur];
        }
    }
}
```

SEGMENTED SIEVE

```
void segmented_sieve( int l , int h )
{
    int i , j , k , m , end ;
    double L = (double) l ;
    memset ( composite , 0 , sizeof ( composite ) ) ;
    end = ceil ( sqrt ( h ) ) ;
    for ( i=3 ; i<end ; i+=2 )
    {
        if ( !COMPS[i] )
        {
            j = ceil ( L / i ) ;
            k = h / i ;
            m = i*j-l ;
            for ( j , m ; j<=k ; j++ , m+=i )
composite[m] = 1 ;
        }
    }
}
```

CONSTRUCT N FROM SUM OF DIVISORS

```
// powLL(a, b) computes a^b, rememver that prime upto i-
1 are used
LL table[NN+1][NN+1]; // if there is an overflow,
table[i][j] = inf;
void preprocessTable()
{
    for( int i = 0; i <= NN; i++ ) table[0][i] = 1;
    for( int i = 1; i <= NN; i++ )
    {
        table[i][0] = 1;
        for( int j = 1; j < NN; j++ ) table[i][j] =
table[i][j-1] + powLL(pr[i-1], j);
    }
}
vector <LL> calculateXFromSumOfDivisors( int sum )
{
    vector <LL> res;
    LL val = 1, prevD = 1;
    for( int i = NN; ; i-- )
    {
        if( sum == 1 )
        {
            res.push_back( val ); // Here the value is
saved
            sum *= prevD, val = 1;
        }
        if( i <= 0 || sum == 1 ) break;
        for( int j = NN - 1; j >= 0; j-- )
        {
            if( table[i][j] > 1 && ( sum % table[i][j]
== 0 ) )
            {
                val *= powLL( pr[i-1], j );
                sum /= table[i][j], prevD = table[i][j];
                break;
            }
        }
    }
    return res;
}
```

34. ORDERED STATISTICS TREE

```
#include <ext/pb_ds/assoc_container.hpp> // Common file
#include <ext/pb_ds/tree_policy.hpp> // Including
tree_order_statistics_node_update
using namespace __gnu_pbds;
typedef
tree<int,null_type,less<int>,rb_tree_tag,tree_order_stat
istics_node_update>ordered_set;
```

```
int main()
{
    ordered_set X;
    X.insert(1);

    cout<<*X.find_by_order(1)<<endl; // 2
    cout<<(end(X)==X.find_by_order(6))<<endl; // true

    cout<<X.order_of_key(-5)<<endl;  // 0
}
```

35. ROPE

```
#include <ext/rope> //header with rope
using namespace __gnu_cxx; //namespace with rope and
some additional stuff
int main()
{
    ios_base::sync_with_stdio(false);
    rope <int> v; //use as usual STL container
    int n, m;
    cin >> n >> m;
    for(int i = 1; i <= n; ++i)
        v.push_back(i); //initialization
    int l, r;
    for(int i = 0; i < m; ++i)
    {
        cin >> l >> r;
        --l, --r;
        rope <int> cur = v.substr(l, r - l + 1);
        v.erase(l, r - l + 1);
        v.insert(v.mutable_begin(), cur);
    }
    for(rope <int>::iterator it = v.mutable_begin(); it
!= v.mutable_end(); ++it)
        cout << *it << " ";
    return 0;
}
```

36. SCC

```
int low[MAX], tim[MAX], col[MAX], no_of_component, n,
timer, group_id[MAX];
vector <int> adj[MAX], dag[MAX];
stack <int> st;
void scc(int u)
{
    low[u] = tim[u] = timer++;
    col[u] = 1;
    st.push(u);
    int i, elements = adj[u].size(), v, tem;
    for(i = 0; i < elements; i++)
    {
        v = adj[u][i];
        if(col[v] == 1)
            low[u] = min(low[u], tim[v]);
        else if(col[v] == 0)
        {
            scc(v);
            low[u] = min(low[u], low[v]);
        }
    }
    if(low[u] == tim[u])
    {
        do
        {
            tem = st.top();
            st.pop();
            group_id[tem]=no_of_component;
            col[tem] = 2;
        }
        while(tem != u);
        no_of_component++;
    }
}
void call_for_scc_check()
{
    no_of_component = timer = 0;
    clr(col, 0);
    int i;
    while(!st.empty()) st.pop();
    for(i = 0; i < n; i++)
    {
        if(col[i] == 0)
```

```
            scc(i);
    }
}

void make_new_DAG()
{
    int i,j,u,v;

    for(i = 0; i < no_of_component; i++) dag[i].clear();

    for(i = 0; i < n; i++)
    {
        for(j = 0; j < SZ(adj[i]); j++)
        {
            u=group_id[i];
            v=group_id[adj[i][j]];
            if(u!=v)
                dag[u].pb(v);
        }
    }
}
```

37. PALINDROMIC TREE

```
struct node{
    int next[26], len, suffLink, num; ///num->count of
palindromes associated with current letter as last
letter
    void init()
    {
        clr(next, -1);
        suffLink = 2;
        num = 1;
        len = 0;
    }
};
int last, totNodes; ///last->current suffix link
node tree[MAX];

void addLetter(string &s, int pos)
{
    int cur = last, curLen = 0;
    int id = s[pos] - 'a';
    while(true) ///find suffix link including id, xAx
    {
        curLen = tree[cur].len;
        if(pos-1-curLen >= 0 && s[pos-1-curLen] ==
s[pos])
            break;
        cur = tree[cur].suffLink;
    }
    if(tree[cur].next[id] != -1) /// node already exists
    {
        last = tree[cur].next[id];
        return;
    }
    totNodes++;
    tree[totNodes].init(); ///create new node
    last = totNodes;
    tree[totNodes].len = tree[cur].len + 2;
    tree[cur].next[id] = totNodes;
    if(tree[totNodes].len == 1)
        return;
    while(true) ///find suffix link xBx for current new
node, where B is suffix link of Ax
    {
        cur = tree[cur].suffLink;
        curLen = tree[cur].len;
        if(pos-1-curLen >= 0 && s[pos-1-curLen] ==
s[pos])
        {
            tree[totNodes].suffLink =
tree[cur].next[id];
            break;
        }
    }
    tree[totNodes].num =
1+tree[tree[totNodes].suffLink].num;
}

void init()
{
    tree[1].init(), tree[2].init();
```

```
    totNodes = last = 2;
    tree[1].len = -1, tree[2].len = 0;
    tree[1].suffLink = tree[2].suffLink = 1;
}

int main()
{
    string s;
    while(cin >> s)
    {
        init();
        int ans = 0;
        for(int i = 0; i < SZ(s); i++)
        {
            addLetter(s, i);
            ans += tree[last].num;
        }
        cout << ans << "\n";
    }
    return 0;
}
```

38. RMQ

```
int stable[MAX][LOGMAX], arr[MAX];

void preprocess(int n)
{
    int i, j;
    for(i = 0; i < n; i++) stable[i][0] = arr[i];
    for(j = 1; (1<<j) <= n; j++)
    {
        for(i = 0; i + (1<<j) - 1 < n; i++)
            stable[i][j] = min(stable[i][j-1],
stable[i+(1<<(j-1))][j-1]);
    }
}

int main()
{
    int n, i;
    cin >> n;
    for(i = 0; i < n; i++)
        cin >> arr[i];
    int q;
    cin >> q;
    preprocess(n);
    while(q--)
    {
        cin >> i >> n;
        int lg = log2(n-i+1);
        cout << min(stable[i][lg], stable[n-
(1<<lg)+1][lg]) << endl;
    }
    return 0;
}
```

39. SIEVE

NORMAL

```
const int N = 10000000;
int lp[N+1];
vector<int> pr;

for (int i=2; i<=N; ++i) {
      if (lp[i] == 0) {
            lp[i] = i;
            pr.push_back (i);
      }
      for (int j=0; j<(int)pr.size() && pr[j]<=lp[i] &&
i*pr[j]<=N; ++j)
            lp[i * pr[j]] = pr[j];
}
```

BITWISE

```
LL col[s/64+10], ma;
int seive()//1 indexed
{
    long long i,j,k;
    k=0;
    LL prev=0;
    for(i=3;i<s;i+=2)
         if(!(col[i/64]&(1LL<<(i%64))))
         {
```

```
            if((i%4)==1)
            {
                 k++;
                 ma=max(ma,i-prev);
                 prev=i;
             }
             for(j=i*i;j<s;j+=2*i)
                 col[j/64]|=(1LL<<(j%64));
        }
    return k;
}
```

40. SLIDING WINDOW (MIN)

```
void sliding_window_minimum(std::vector<int> & ARR, int
K) {
  // pair<int, int> represents the pair (ARR[i], i)
  std::deque< std::pair<int, int> > window;
  for (int i = 0; i < ARR.size(); i++) {
     while (!window.empty() && window.back().first >=
ARR[i])
       window.pop_back();
     window.push_back(std::make_pair(ARR[i], i));

     while(window.front().second <= i - K)
       window.pop_front();

     std::cout << (window.front().first) << ' ';
  }
}
```

41. SUFFIX ARRAY

```
string text;
int revSA[MAX],SA[MAX];
int cnt[MAX] , nxt[MAX];
bool bh[MAX],b2h[MAX];
int lcp[MAX];


bool cmp(int i,int j)
{
    return text[i]<text[j];
}

void sortFirstChar(int n)
{
    /// sort for the first char  ...
    for(int i =0 ; i<n ; i++)
        SA[i] = i;
    sort(SA,SA+n ,cmp);

    ///indentify the bucket ........
    for(int i=0 ; i<n ; i++)
    {
        bh[i] = (i==0  || text[SA[i]]!=text[SA[i-1]]);
        b2h[i] = false;
    }
    return;
}

int CountBucket(int n)
{
    int bucket = 0;
    for(int i =0 ,j; i<n ; i=j)
    {
        j = i+1;
        while(j<n && bh[j]==false) j++;
        nxt[i] = j;
        bucket++;
    }
    return bucket;
}

void SetRank(int n)
{
    for(int i = 0 ; i<n ; i=nxt[i])
    {
        cnt[i] = 0;
        for(int j =i ; j<nxt[i] ; j++)
        {
            revSA[SA[j]] = i;
        }
    }
```

```
    return;
}

void findNewRank(int l,int r,int step)
{
    for(int j = l  ; j<r ; j++)
    {
        int pre = SA[j] - step;
        if(pre>=0)
        {
            int head = revSA[pre];
            revSA[pre] = head+cnt[head]++;
            b2h[revSA[pre]] = true;
        }
    }
    return;
}

void findNewBucket(int l,int r,int step)
{
    for(int j = l  ; j<r ; j++)
    {
        int pre = SA[j] - step;
        if(pre>=0 && b2h[revSA[pre]])
        {
            for(int k = revSA[pre]+1 ; b2h[k] && !bh[k]
; k++) b2h[k] = false;
        }
    }
    return;
}

void buildSA(int n)
{
    ///start sorting in logn step ...
    sortFirstChar(n);
    for(int h =1 ; h<n ; h<<=1)
    {
        if(CountBucket(n)==n) break;
        SetRank(n);
        /// cause n-h suffix must be sorted
        b2h[revSA[n-h]] = true;
        cnt[revSA[n-h]]++;

        for(int i = 0 ; i<n ; i=nxt[i])
        {
            findNewRank(i,nxt[i] , h);
            findNewBucket(i , nxt[i] , h);
        }
        ///set the new sorted suffix array ...
        for(int i =0 ; i<n ; i++)
        {
            SA[revSA[i]] = i;
            bh[i] |= b2h[i]; ///new bucket ....
        }
    }
    return;
}

void buildLCP(int n)
{
    int len = 0;
    for(int i = 0 ;i<n ; i++)
        revSA[SA[i]] = i;
    for(int i =0 ; i< n ; i++)
    {
        int k = revSA[i];
        if(k==0)
        {
            lcp[k] = 0;
            continue;
        }
        int j = SA[k-1];
        while(i+len < n && j+len < n &&
text[i+len]==text[j+len]) len++;
        lcp[k] = len;
        if(len) len--;
    }
    return;
}

void printSA()
```

```
{
    for(int i=0;i<SZ(text);i++) printf("%d %d %d %s
%d\n", i, SA[i], revSA[SA[i]],
text.substr(SA[i]).c_str(), lcp[i]);
    puts("");
//    for(int i=1;i<SZ(text);i++) printf("%d ",lcp[i]);
    puts("");
    return ;
}

int main()
{
    while(cin >> text)
    {
        buildSA(SZ(text));
        buildLCP(SZ(text));
        printSA();
    }
    return 0;
}
```

42. TRIE

```
int trie[MAX][52], cnt[MAX], last;
char str[10001];

void add(char *str)
{
    int i, id, cur = 0;
    for(i = 0; str[i]; i++)
    {
        if(islower(str[i]))
            id = str[i] - 'a' + 26;
        else
            id = str[i] - 'A';
        if(trie[cur][id] == -1)
        {
            trie[cur][id] = ++last;
            clr(trie[last], -1);
            cnt[last] = 0;
        }
        cur = trie[cur][id];
    }
    cnt[cur]++;
}
/// do clr(trie[0], -1) and last = 0 for every case

int get(char *str)
{
    int id, i, cur = 0;
    for(i = 0; str[i]; i++)
    {
        if(islower(str[i]))
            id = str[i] - 'a' + 26;
        else
            id = str[i] - 'A';
        if(trie[cur][id] == -1)
            return 0;
        cur = trie[cur][id];
    }
    return cnt[cur];
}
```