// Flow Adj 1-based

```
int res[201][201];
int find_path(int src, int dest, int n){
   int ret = -1, backlink[n+1];
   queue<int> BFS;BFS.push(src);
   memset(backlink, 0, sizeof(backlink));
   backlink[src] = -1;
   while(!BFS.empty()){
      int x = BFS.front();BFS.pop();
      if(x == dest){
         ret = 1000;
         while(backlink[x] != -1){
            int pre = backlink[x];
            ret = min(ret, res[pre][x]);
            x = pre;
         }
         x = dest;
         while(backlink[x] != -1){
            int pre = backlink[x];
            res[pre][x] -= ret;
            res[x][pre] += ret;
            x = pre;
         }
         return ret;
      }
      for(int i = 1;i<=n;i++)
         if(i != x && !backlink[i] && res[x][i])
            backlink[i] = x, BFS.push(i);
   }
   return ret;
}
int max_flow(int src, int sink, int n){
   int ret = 0, path_capacity;
   while((path_capacity = find_path(src, sink, n)) != -1)
      ret += path_capacity;
   return ret;
}
```

// Max Matching Bipartite

```
vector<vector<int> > neigh;
int bw[100000]; // all -1s
int vis[100000];
int find_path(int n){
   if(vis[n])
      return 0;
   vis[n] = 1;
   for(int i = 0;i<neigh[n].size();i++)
      if(bw[neigh[n][i]] == -1 || find_path(bw[neigh[n]
[i]])){
         bw[neigh[n][i]] = n;
         return 1;
```

// Flow General Graph

```
struct node{
   map<node*, int> edges;
}nodes[1000000];
int cnt = 0;
node* nxtNode(){
   nodes[cnt].edges.clear();
   return &nodes[cnt++];
}
int find_path(node *src, node *dest){
   int ret = -1;
   map<node*, node*> backlink;
   queue<node*> BFS;BFS.push(src);
   backlink[src] = 0;
   while(!BFS.empty()){
      node *x = BFS.front();BFS.pop();
      if(x == dest){
         ret = 1000;
         while(backlink[x] != 0){
            node *pre = backlink[x];
            ret = min(ret, pre->edges[x]);
            x = pre;
         }
         x = dest;
         while(backlink[x] != 0){
            node *pre = backlink[x];
            pre->edges[x] -= ret;
            x->edges[pre] += ret;
            x = pre;
         }
         return ret;
      }
      for(map<node*, int>::iterator it = x->edges.begin();
it != x->edges.end(); it++)
         if(backlink.find(it->first) == backlink.end() &&
it->second)
            backlink[it->first] = x, BFS.push(it->first);
   }
   return ret;
}
int max_flow(node *src, node *sink){
   int ret = 0, path_capacity;
   while((path_capacity = find_path(src, sink)) != -1)
      ret += path_capacity;
   return ret;
}
```

//GCD

```
int GCD(int a,int b){
   if(b != 0) while(b^=a^=b^=a%=b);
   return a;}
```

```
    }
  return 0;
}
int max_flow(){
  int res = 0;
  for(int i = 0;i<neigh.size();i++){
    if(neigh[i].size() == 0)
      continue;
    memset(vis, 0, sizeof(int)*neigh.size());
    res += find_path(i);
  }
  return res;
}

//MATPOW
void matpow(long long M[2][2],int n, long long x)
{
  if(n>1)
  {
    matpow(M,n/2,x);
    long long a = M[0][0], b = M[0][1], c = M[1][0], d
= M[1][1];
    M[0][0] = ((a*a)%mod + (b*c)%mod)%mod;
    M[0][1] = ((a*b)%mod + (b*d)%mod)%mod;
    M[1][0] = ((a*c)%mod + (c*d)%mod)%mod;
    M[1][1] = ((b*c)%mod + (d*d)%mod)%mod;
  }
  if(n&1 && n>1)
  {
    long long a = M[0][0], b = M[0][1], c = M[1][0], d
= M[1][1];
    M[0][0] = (x*(a+b))%mod;
    M[0][1] = (a*x)%mod;
    M[1][0] = (x*(c+d))%mod;
    M[1][1] = (c*x)%mod;
  }
}
long long fib(int n, long long start)
{
  long long M[2][2] = {{1,1},{1,0}};
  matpow(M,n,1);
  return start*M[0][0];
}

// TARJAN SCC
int n;
struct vertex{
  vector<int> neigh;
  int index, component, lowlink;
}v_tmp, vertices[10000];
bool visited[10000];
```

```
// Dinic Adjancency
int max_flow_dinic(int src, int sink, int n){
  int ret = 0;
  int prev[n+1];
  while(1){
    queue<int> BFS;
    memset(prev, 0, sizeof(prev));
    prev[src] = -1;
    BFS.push(src);
    while(!BFS.empty() && !prev[sink]){
      int curr = BFS.front();
      BFS.pop();
      for(int i = 1;i<=n;i++)
        if(!prev[i] && res[curr][i])
          prev[i] = curr, BFS.push(i);
    }
    if(!prev[sink])
      break;
    for(int i = 1;i<=n; i++){
      if(res[i][sink] && prev[i]){
        int inc = res[i][sink];
        for(int u = prev[i], v = i; u>=0; inc = min(inc,
res[u][v]), v = u, u = prev[u]);
        for(int u = i, v = sink; u>=0; res[u][v]-=inc,
res[v][u]+=inc, v = u, u = prev[u]);
        ret += inc;
      }
    }
  }
  return ret;
}
//BIT
int BIT[1000001];
int query(int idx, int n)
{
  int ret = 0;
  while(idx>0)
  {
    ret += BIT[idx];
    idx -= idx & -idx;
  }
  return ret;
}
void update(int idx, int n, int val)
{
  while(idx<=n)
  {
    BIT[idx] += val;
    idx += idx & -idx;
  }
}
```

```
stack<int> DFS;
int index;
int connected;
void strongconnect(int id){
   vertices[id].index = vertices[id].lowlink = index;
   index++;
   DFS.push(id);
   visited[id] = true;
   int w;
   for(int j=0;j<vertices[id].neigh.size(); j++){
      w = vertices[id].neigh[j];
      if(vertices[w].index == -1){
         strongconnect(w);
         vertices[id].lowlink = min(vertices[id].lowlink,
vertices[w].lowlink);
      }else if(visited[w])
         vertices[id].lowlink = min(vertices[id].lowlink,
vertices[w].index);
   }
   if(vertices[id].lowlink == vertices[id].index){
      do{
         w = DFS.top();
         DFS.pop();
         visited[w] = false;
         vertices[w].component = connected;
      }while(id != w);
      connected++;
   }
}
int scc(){
   index=0;
   connected = 0;
   for(int i=0;i<n;i++)
      if(vertices[i].index == -1)
         strongconnect(i);
}

//Stable Matching
function stableMatching {
   Initialize all m ∈ M and w ∈ W to free
   while ∃ free man m who still has a woman w to
propose to {
      w = m's highest ranked such woman to whom he has
not yet proposed
      if w is free
        (m, w) become engaged
      else some pair (m', w) already exists
        if w prefers m to m'
          (m, w) become engaged
          m' becomes free
        else
```

```
//POWER
int mod = 1000000007;
int POW(long long r, long long n){
   int ans = 1;
   while(n>0){
      if(n&1)
         ans = (ans*r)%mod;
      n >>= 1;
      r = (r*r)%mod;
   }
   return ans;
}
//SEGMENT TREE
#define ARR_SIZE 510000
struct node{
   //Node variables
};
node TREE[ARR_SIZE<<2];
int n,arr[ARR_SIZE];
using namespace std;
void build_segment_tree(int NODE,int a,int b)
{
   if(a==b){
      // Single node condition
      return;
   }
   int mid=(a+b)/2;
   int left=2*NODE,right=2*NODE+1;
   build_segment_tree(left,a,mid);
   build_segment_tree(right,mid+1,b);

   // Merge Logic for left and right
}
node query_segment_tree(int NODE,int a,int b,int x,int
y)
{
   if(x<=a && y>=b)return TREE[NODE];
   // lazy_prop(a, b);
   int mid=(a+b)/2;
   int left=2*NODE,right=2*NODE+1;
   if(y<=mid)return query_segment_tree(left,a,mid,x,y);
   if(x>mid)return
query_segment_tree(right,mid+1,b,x,y);

   node left_query=query_segment_tree(left,a,mid,x,y);
   node
right_query=query_segment_tree(right,mid+1,b,x,y);

   node ans;
   // Merge left_query and right_query into ans
   return ans;
```

```
        (m', w) remain engaged
    }
}

// KMP
COMPUTE_PI(P):
        m = P.length
        pi[1] = 0
        k = 0
        for q = 2 to m:
                while k>0 and P[k+1] != P[q]:
                        k = pi[k]
                if P[k+1] == P[q]:
                        k++
                pi[q] = k
        return pi
KMP_MATCH(S, P):
        n = S.length
        m = P.length
        pi = COMPUTE_PI(P)
        q = 0
        for i = 1 to n
                while q>0 and P[q+1] != T[i]:
                        q = pi[q]
                if P[q+1] == T[i]:
                        q++
                if q == m:
                    print "Pattern occur with shift " i-m
                        q = pi[q]

//Suffix Array
#define MAX_STR_SIZE 50001
#define MAX_AUX 18
int ranks[200];
int LOG2[MAX_STR_SIZE];
class suffix_array{
public:
    char str[MAX_STR_SIZE];
    struct node{
        int pos, tmp1, tmp2;
        friend bool operator<(const node &a, const node
&b){
            if(a.tmp1 == b.tmp1)
                return a.tmp2<b.tmp2;
            return a.tmp1<b.tmp1;
        }
    }SA[MAX_STR_SIZE];
    int lcp_arr[MAX_STR_SIZE];
    int aux_lcp[MAX_STR_SIZE][MAX_AUX];
    int aux[MAX_AUX][MAX_STR_SIZE];
    int steps, cnt, len;
}
void update_segment_tree(int NODE,int a,int b,int x)
{
    if(a==b){   // l<=a & r>=b for lazy
        //Single node logic
        // Set lazy bit
        return;
    }
    // lazy_prop(a, b)
    int mid=(a+b)/2;
    int left=2*NODE,right=2*NODE+1;
    if(x<=mid)
    update_segment_tree(left,a,mid,x);
    else
    update_segment_tree(right,mid+1,b,x);

    //Merge left and right nodes
}

//RMQ
int n;
int aux[100002][22];
int LOG2[100002];
int RMQ(int x, int y){
    int k = LOG2[y-x+1];
    return min(aux[x][k], aux[y-(1<<k)+1][k]);
}
void init(){
    int curr = 2;
    LOG2[1] = 0;
    for(int i = 2; i<=100001;i++){
        if(i == curr){
            curr <<= 1;
            LOG2[i] = LOG2[i-1] + 1;
        }else
            LOG2[i] = LOG2[i-1];
    }

    for(int i=1;i<=n;i++)
        in_I(aux[i][0]);
    for(int j=1; (1<<j)<=n+1; j++)
        for(int i=1;i+(1<<j)<=n+1;i++)
          aux[i][j] = min(aux[i][j-1], aux[i+(1<<(j-1))][j-1]);
}

//Miller Rabin
long long n,s,d,a,lt,k,x,j;
long long mul(long long a,long long b){
    long long x=0,y=a%n;
    while(b>0){
        if(b%2)x=(x+y)%n;
```

```
    void create();
    void gen_lcp_arr();
    int lcp(int x, int y);
    int lcp_RMQ(int x, int y);
}S;
void suffix_array::create(){
    len = 0;
    while(str[len]){
        aux[0][len] = ranks[str[len]];
        len++;
    }
    int pre = 0;
    cnt = 1;
    for(steps = 1;pre<len;steps++, cnt<<=1){
        for(int i=0;i<len;i++){
            SA[i].pos = i;
            SA[i].tmp1 = aux[steps-1][i];
            SA[i].tmp2 = (i+cnt<len)?aux[steps-1][i+cnt]:-1;
        }
        sort(SA, SA+len);
        aux[steps][SA[0].pos] = 0;
        for(int i=1;i<len;i++){
            if(SA[i].tmp1 == SA[i-1].tmp1 && SA[i].tmp2
== SA[i-1].tmp2)
                aux[steps][SA[i].pos] = aux[steps][SA[i-
1].pos];
            else
                aux[steps][SA[i].pos] = i;
        }
        pre = cnt;
    }
}
int suffix_array::lcp(int x, int y){
    int ret = 0;
    if(x == y)return len-x;
    for(int k=steps-1;k>=0 && x<len && y<len; k--)
        if(aux[k][x] == aux[k][y]) x += 1<<k, y += 1<<k,
ret += 1<<k;
    return ret;
}
void suffix_array::gen_lcp_arr(){
    lcp_arr[0] = 0;
    for(int i=1;i<len;i++)
        lcp_arr[i] = lcp(SA[i-1].pos, SA[i].pos);

    for(int i=0;i<len;i++)
        aux_lcp[i][0] = lcp_arr[i];

    for(int j=1; 1<<j <= len; j++)
        for(int i=0; i+(1<<j)<=len; i++)
            aux_lcp[i][j] = min(aux_lcp[i][j-1],
        y=(y+y)%n;
        b>>=1;
    }
    return x%n;
}
long long pow_1(long long r2,long long n1){
    long long ans=1;
    while(n1>0){
        if((n1&1)>0)
        ans=mul(ans,r2);
        n1>>=1;
        r2=mul(r2,r2);
    }
    return ans%n;
}
bool miller_rabin(){
    if(n==1)return false;
    if(n==2)return true;
    if(n%2==0)return false;
    s=0;d=n-1;
    while(d%2==0){
        s++;d/=2;
    }
    for(int i=0;i<20;i++){
        a=rand()%(n-1)+1;
        x=pow_1(a,d);
        if(x==1 || x==n-1)continue;
        for(j=1;j<=s;j++){
            x=mul(x,x);
            if(x==1)return false;
            if(x==(n-1))break;
        }
        if(j==s+1)return false;
    }
    return true;
}

// Trie
struct node{
    // 0 - new 1 - end 2 - not end
    int status;
    node* kids[10];
}nodes[1000000];
int cnt = 0;
int n;
char str[12];
bool insert(node *curr, int idx){
    if(!str[idx]){
        if(curr->status == 0){
            curr->status = 1;
            return true;
```

```
aux_lcp[i+(1<<(j-1))][j-1]);
}
int suffix_array::lcp_RMQ(int x, int y){
    x = aux[steps-1][x];
    y = aux[steps-1][y];
    if(x>y)
        swap(x, y);
    int k = LOG2[y-x];
    return min(aux_lcp[x+1][k], aux_lcp[y-(1<<k)+1][k]);
}
int main(){
    int cnt = 0;
    for(char ch = 'a';ch<='b';ch++)
        ranks[ch] = cnt++;
    int x = 0;
    for(int i=0;i<MAX_STR_SIZE;i++){
        if(1<<x <= i)
            x++;
        LOG2[i] = x-1;
    }
}
```

```
        }else
            return false;
    }
    if(curr->kids[str[idx]-'0']){
        return insert(curr->kids[str[idx]-'0'], idx+1);
    }else{
        if(curr->status == 1)
            return false;
        else{
            if(curr->status == 0)
                curr->status = 2;
            for(int i=0;i<10;i++)
                nodes[cnt].kids[i] = 0, nodes[cnt].status = 0;
            curr->kids[str[idx]-'0'] = &nodes[cnt];
            cnt++;
            return insert(curr->kids[str[idx]-'0'], idx+1);
        }
    }
}
```

//BIGNUM

```
class BIGNUM{
    long long *N;
    int BASE, BASE_pow, max_size;
    bool sign;

    int int_len(long long x);
    void do_carry();
    void compress();
public:
    int size;
    BIGNUM();
    ~BIGNUM(){delete []N;}
    BIGNUM(const BIGNUM &x);

    void print();
    string get_str();
    void load_from_LL(long long x);
    void load_from_str(const char *tmp);
    BIGNUM add_mag(const BIGNUM &x);
    BIGNUM sub_mag(const BIGNUM &x);
    int cmp(const BIGNUM &x); // 0-> x is equal, -1 -> x is smaller, 1 -> x is greater
    int cmp_mag(const BIGNUM &x);

    BIGNUM operator-(){
        if(this->size == 1 && this->N[0] == 0)
            return *this;
        BIGNUM ret = *this;
        ret.sign = !this->sign;
        return ret;
```

*** Binomial Coefficient**

C(n, r) = n!/(r! * (n-r)!)

C(n, r) = C(n-1, r-1) + C(n-1, r)

*** Number of derangements**

Given an arrangements of n objects, derangements are the number of arrangements such that none object are in their original places

D(n) = n!*(1 - 1/1! + 1/2! - 1/3! +...+ (-1)^n 1/n!)

Recurrence -> n*D(n-1) = -(-1)^n + D(n)

-> D(n) = n*D(n-1) + (-1)^n

D(n) = (n-1) * (D(n-1) + D(n-2))

*** Bayes Theorem**

P(Ai/B) = P(Ai)P(B/Ai)/sum(P(Ai)P(B/Ai))

- rC(n, r) = n*C(n-1, r-1)

-C(n, r)/(r+1) = C(n+1, r+1)/(n+1)

// Series

*** AGP**

a1, a2, ... in AP(a, d). b1, b2, ... in GP(b, r)

S(n) = ab/(1-r) + dbr(1-r^(n-1))/(1-r)^2 - (a+(n-1)d)br^n/(1-r)

* (1-x)^-1 = 1+x+x^2+.... (-1 < x < 1)

* (1-x)^-2 = 1+2x+3x^2+... (-1 < x < 1)

```
    }

    BIGNUM operator+(){return *this;}

    BIGNUM operator+(const BIGNUM &x);
    BIGNUM operator-(const BIGNUM &x);
    BIGNUM operator*(const BIGNUM &x);

    bool operator<(const BIGNUM &x){return cmp(x) == 1;}
    bool operator>(const BIGNUM &x){return cmp(x) == -1;}
    bool operator==(const BIGNUM &x){return cmp(x) == 0;}
    bool operator<=(const BIGNUM &x){return cmp(x)+1 > 0;}
    bool operator>=(const BIGNUM &x){return cmp(x)-1<0;}

    friend void test();
};
BIGNUM::BIGNUM():BASE(1000000), BASE_pow(6), max_size(3500), sign(false){
    // Default setting is enough for operations on 10000 digits
    N = new long long[max_size];
    N[0] = 0;
    size = 1;
}
BIGNUM::BIGNUM(const BIGNUM &x){
    size = x.size; BASE = x.BASE; BASE_pow = x.BASE_pow; max_size = x.max_size; sign = x.sign;
    N = new long long[max_size];
    for(int i=0;i<size;i++) N[i] = x.N[i];
}
int BIGNUM::int_len(long long x){
    int count=0;
    do{ x/=10; count++;} while(x!=0);
    return count;
}
void BIGNUM::print(){
    if(sign)    printf("-");
    for(int i=size-1;i>=0;i--){
        int len=int_len(N[i]);
        for(int j=0;j<BASE_pow-len && i!=size-1;j++)    printf("0");
        printf("%d",N[i]);
    }
}
string BIGNUM::get_str(){
    string ret;
    int i, len;
    for(i=size-1;N[i]<=0 && i>0;i--);
    int start=i;
    if(sign)    ret.push_back('-');
    for(;i>=0;i--){
        int len=int_len(N[i]);
        for(int j=0;j<BASE_pow-len && i!=start;j++) ret.push_back('0');
        string foo;
        int x = N[i];
        do { foo.push_back(x%10 + '0'); x/=10;} while(x);
```

*** Trigo**

- sin(A+B) = sinAcosB + cosAsinB
- cos(A+B) = cosAcosB - sinAsinB
- tan(A+B) = (tanA + tanB)/(1-tanAtanB)
- sin(A+B)sin(A-B) = sin^2(A) - sin^2(B)
- cos(A+B)cos(A-B) = cos^2(A) - sin^2(B)
- sin(3A) = 3sinA - 4sin^3(A) = 4sin(60-A)sinAsin(60+A)
- cos(3A) = 4cos^3(A) - 3cosA = 4cos(60-A)cosAcos(60+A)
- tan3A = (3tanA - tan^3(A))/(1-3tan^3(A)) = tan(60-A)tanAtan(60+A)
- sinA + sinB = 2sin((A+B)/2)cos((A-B)/2)
- sinA - sinB = 2sin((A-B)/2)cos((A+B)/2)
- cosA + cosB = 2cos((A-B)/2)cos((A+B)/2)
- tanA + tanB = sin(A+B)/(cosAcosB)

```
        reverse(foo.begin(), foo.end());
        ret += foo;
    }
    return ret;
}
void BIGNUM::load_from_LL(long long x){
    if(x<0){sign = true;x *= -1;}
    else    sign = false;
    size = 0;
    do { N[size++] = x%BASE; x /= BASE;} while(x);
}
void BIGNUM::load_from_str(const char *tmp){
    if(tmp[0] == '-'){sign = true;tmp++;}
    else    sign = false;
    int len=strlen(tmp);
    size=(len-1)/BASE_pow + 1;
    for(int i=0;i<size;i++) N[i] = 0;
    for(int i=0;i<len;i++)  N[((len-1-i)/BASE_pow)]=N[(len-1-i)/BASE_pow]*10+tmp[i]-'0';
    if(size == 1 && N[0] == 0)  sign = false;
}
int BIGNUM::cmp(const BIGNUM &x){
    if(sign && !x.sign) return 1;
    if(!sign && x.sign) return -1;
    int LT = -1, GT = 1;
    if(sign && x.sign)  LT = 1, GT = -1;
    if(size<x.size) return GT;
    if(size>x.size) return LT;
    int idx = size-1;
    while(idx>=0){
        if(N[idx] < x.N[idx])   return GT;
        if(N[idx] > x.N[idx])   return LT;
        idx--;
    }
    return 0;
}
int BIGNUM::cmp_mag(const BIGNUM &x){
    int LT = -1, GT = 1;
    if(size<x.size) return GT;
    if(size>x.size) return LT;
    int idx = size-1;
    while(idx>=0){
        if(N[idx] < x.N[idx])   return GT;
        if(N[idx] > x.N[idx])   return LT;
        idx--;
    }
    return 0;
}
BIGNUM BIGNUM::add_mag(const BIGNUM &x){
    BIGNUM ret;
    int size1 = this->size, size2 = x.size, lt = min(size1, size2), i;
    ret.size = max(size1, size2);
    for(i=0;i<lt;i++)   ret.N[i] = this->N[i]+x.N[i];
```

*** Euler Totient Function**

- phi(n) = number of k (1<=k<=n) such that GCD(k,n) = 1
- Multiplicative, i.e. if GCD(m,n) = 1, then phi(mn) = phi(m)*phi(n)
- phi(n) = n*(1-1/p1)*(1-1/p2).... p1,p2, ... -> distinct primes dividing n
- phi(p^k) = p^k - p^(k-1)
- phi(d1)+phi(d2)+.... = n d1,d2,... -> positive divisors of n
- if a and n are coprime, then a^phi(n) = 1 (mod n)
- b%a = 0 implies phi(b)%phi(a) = 0
- n divides phi(a^n - 1) (a, n > 1)
- phi(mn) = phi(m)*phi(n)*d/phi(d) d = gcd(m,n)
    - phi(2m) = (m is even? 2: 1)*phi(m)
    - phi(n^m) = n^(m-1)*phi(n)
- phi(LCM(a,b))*phi(GCD(a,b)) = phi(a)*phi(b)
- for n>=3, phi(n) is even
- if n has r distinct odd prime factors, then 2^r divides phi(n)

```
    if(ret.size == this->size)
        for(;i<ret.size;i++)    ret.N[i] = this->N[i];
    else
        for(;i<ret.size;i++)    ret.N[i] = x.N[i];
    ret.do_carry();
    return ret;
}
BIGNUM BIGNUM::sub_mag(const BIGNUM &x){
    BIGNUM ret;ret.size = 0;
    int c = 0, lt = min(x.size, size), i;
    for(i=0;i<lt;i++){
        ret.N[ret.size] = this->N[i]+(BASE-(i!=0)-x.N[i])+c;
        c = ret.N[ret.size]/BASE;
        ret.N[ret.size] %= BASE;
        ret.size++;
    }
    lt = max(x.size, size);
    if(lt == x.size){
        for(;i<lt;i++){
            ret.N[ret.size] = (BASE-1-x.N[i])+c;
            c = ret.N[ret.size]/BASE;
            ret.N[ret.size] %= BASE;    ret.size++;
        }
    }else
        for(;i<lt;i++){
            ret.N[ret.size] = this->N[i]+(BASE-1)+c;
            c = ret.N[ret.size]/BASE;
            ret.N[ret.size] %= BASE;    ret.size++;
        }
    ret.compress();
    if(!c){
        ret.sign = true;
        for(int i=0;i<ret.size;i++) ret.N[i] = BASE-(i!=0)-ret.N[i];
    }
    ret.compress();
    return ret;
}
void BIGNUM::compress(){
    while(N[size-1] == 0 && size>1)size--;
}
void BIGNUM::do_carry(){
    int c = 0;
    for(int i=0;i<size;i++){
        N[i] += c;
        c = N[i]/BASE;
        N[i] %= BASE;
    }
    if(c)
        N[size++] = c;
}
inline BIGNUM BIGNUM::operator+(const BIGNUM &x){
    if(!sign && !x.sign){    // (+) + (+)
```

*** Fermat's Little Theorem**

if m = n (mod p-1), p->prime
  m = b(p-1)+n
  a^m = a^(b(p-1)) . a^n = 1^b . a^n = a^n (mod n)
a^p = a (mod p) , p->prime
a^(p-1) = 1 (mod p), p and a -> coprime

*** Euler Theorem**

a^ETF(n) = 1 (mod n)

*** Primitive root modulo n**

g^k = a (mod n)
a is coprime to n (all coprimes should satisfy this if g is primitive root modulo n)
g is the primitive root modulo n
k is the index/discrete logarithm of a to the base g mod n
-The product of all primitive roots of prime p != 3 is 1 (mod p), for p=3, it is 2
-If multiplicative order of a number m mod n is phi(n), m is primitive root.
  - Multiplicative order of a mod n is the smallest positive int k with a^k = 1 (mod n), gcd(a, n)=1
-m^(phi(n)/p_i) (mod n)
  where p_i is a prime factor of phi(n). If for a m, all results for p_i are different from 1, then it is a primitive root
-number of primitive root mod n = phi(phi(n))
-g is a primitive root modulo p, then g is a primitive root modulo all powers p^k unless g^(p-1) ≡ 1 (mod p^2); in that case, g + p is

```
        BIGNUM ret = this->add_mag(x);
        return ret;
    }
    if(sign && x.sign){     // (-) + (-)
        BIGNUM ret = this->add_mag(x);
        ret.sign = true;
        return ret;
    }
    if(sign && !x.sign){    // (-) + (+)
        BIGNUM ret = sub_mag(x);
        if(!(ret.size == 1 && ret.N[0] == 0))ret.sign = !ret.sign;
        return ret;
    }
    // (+) + (-)
    BIGNUM ret = sub_mag(x);
    return ret;
}
inline BIGNUM BIGNUM::operator-(const BIGNUM &x){
    if(!sign && !x.sign){    // (+) - (+)
        BIGNUM ret = this->sub_mag(x);
        return ret;
    }
    if(sign && x.sign){     // (-) - (-)
        BIGNUM ret = this->sub_mag(x);
        if(!(ret.size == 1 && ret.N[0] == 0))ret.sign = !ret.sign;
        return ret;
    }
    if(sign && !x.sign){    // (-) - (+)
        BIGNUM ret = add_mag(x);
        ret.sign = true;
        return ret;
    }
    // (+) - (-)
    BIGNUM ret = add_mag(x);
    return ret;
}
BIGNUM BIGNUM::operator*(const BIGNUM &x){
    BIGNUM ret;
    ret.size = size+x.size-1;
    for(int i=0;i<ret.size;i++) ret.N[i] = 0;
    for(int i=0;i<x.size;i++)
        for(int j=0;j<size;j++) ret.N[i+j] += x.N[i]*N[j];
    ret.do_carry(); ret.compress();
    ret.sign = (sign ^ x.sign);
    if(ret.sign && (ret.size == 1 && ret.N[0] == 0))    ret.sign = false;
    return ret;
}

//POINT
template <class T>
class POINT{
    public:
```

TRIANGLE

R -> circumradius
r = inradius
r1, r2, r3 = exradius
* sine rule
sinA/a = sinB/b = sinC/c = 1/2R = 2(area)/abc
* cosine rule
cosA = (b^2+c^2-a^2)/2bc
..
* sin(A/2) = sqrt((s-b)(s-c)/bc)
  cos(A/2) = sqrt(s(s-a)/bc)
* a = bcosC+ccosB
* r = area/s
* 2r<=R
* r1+r2+r3 = 4R+r
  r1 = area/(s-a) = stan(A/2) = 4Rsin(A/2)cos(B/2)cos(C/2)
  r1r2+r2r3+r3r1 = s^2 = r1r2r3/r
* r+2R = s in right traingle

* **PnC**
* Circular Permutation of n things taken r at a time = P(n, r)/r (clock+anticlock)
* selection of n distinct objects taken r at a time with repetition = C(n+r-1, r)
* no of ways to divide n distinct objects into r unequal groups of size a1, a2,...
  = n!/(a1!a2!..)
* divide m.n distinct objects equally into n groups = (mn)!/(m!^n)n!
* Multinomial
x1+x2+...+xm = n, a1<=x1<=b1, a2<=x2<=b2,...
number of solutions = coefficient of x^n in (x^a1+...+x^b1)(x^a2+...+x^b2)...
coefficient of x^r in (1-x)^-n = C(n+r-1, r)

```
    T P[2];
    POINT(){}
    POINT(T x, T y){P[0] = x, P[1] = y;}
    //dot
    T operator*(const POINT &a){
        return a.P[0]*this->P[0] + a.P[1]*this->P[1];
    }
    POINT operator-(const POINT &b){
        return POINT(this->P[0] - b.P[0], this->P[1] - b.P[1]);
    }
    //cross
    T operator^(const POINT &b){
        return this->P[0]*b.P[1] - this->P[1]*b.P[0];
    }
};
template <class T>
vector<int> convex_hull(POINT<T> points[],int n)
{
    points[0].P[0]=20000;
    points[0].P[1]=20000;
    bool used[n+1];
    for(int i=1;i<=n;i++)   used[i]=false;
    int bot_left=0;
    vector<int> pos;
    for(int i=1;i<=n;i++){
        if(points[bot_left].P[1]>points[i].P[1])
            bot_left = i;
        else if(points[bot_left].P[1]==points[i].P[1] && points[bot_left].P[0]>points[i].P[0])
            bot_left = i;
    }
    pos.push_back(bot_left);
    int start = bot_left;
    do{
        int n2=-1;
        int dis = 0;
        for(int i=1;i<=n;i++){
            if(i==bot_left)continue;
            if(used[i])continue;
            if(n2==-1)n2=i;
            T cross = (points[i]-points[bot_left])^(points[n2]-points[bot_left]);
            T d = (points[i]-points[bot_left])*(points[i]-points[bot_left]);
            if(cross>0)n2=i,dis=d;
            else if(cross==0)
                if(d>dis)dis=d,n2=i;
        }
        bot_left = n2;
        used[n2]=true;
        pos.push_back(n2);
    }while(start!=bot_left);
    return pos;
}
// Distance of line b-c from point a
```

Extended Euclid

```
pair<int, pair<int, int> > GCD(int a,int b)
{
    if(a == 0)
        return make_pair(b, make_pair(0, 1));
    else{
        pair<int, pair<int, int> > foo = GCD(b%a,
a);
        return make_pair(foo.first,
make_pair(foo.second.second-
b/a*foo.second.first, foo.second.first));
    }
}
long long modInv(long long a, long long m){
    pair<long long, pair<long long, long long> >
foo = GCD(a, m);
    return ((foo.second.first % m) + m)%m;
}
```

```
int linedist(POINT<int> &a, POINT<int> &b, POINT<int> &c)
{
   int dot1 = (b-a)*(c-b);
   if(dot1>0)return 1;
   int dot2 = (a-b)*(c-a);
   if(dot2>0)return 1;
   return (b-a)^(c-a);
}
// sign of ba x ca
int ccw(POINT<int> a, POINT<int> b, POINT<int> c)
{
   long long foo = ((b-a)^(c-a));
   return foo > 0?1:(foo<0?-1:0);
}
// If a-b intersects with c-d
bool intersect(POINT<int> a, POINT<int> b, POINT<int> c, POINT<int> d)
{
   if(ccw(a,b,c) == 0 && ccw(a,b,d) == 0){
      if((((b-a)*(c-b))>0 && ((b-a)*(d-b))>0) || (((a-b)*(c-a))>0 && ((a-b)*(d-a))>0))
         return false;
      else
         return true;
   }
   return ccw(a,c,d) != ccw(b,c,d) && ccw(a,b,c) != ccw(a,b,d);
}
// Number of points from points1[] at which a-points2[b] intersects.
int getans(POINT<int> &a, int b, POINT<int> points1[], POINT<int> points2[], vector<int> &points)
{
   int cnt = 0;
   for(int i=0;i<-1+points.size();i++)
      if(intersect(a, points2[b], points1[points[i]], points1[points[i+1]]))
         cnt++;
   return cnt;
}
// if pt (in points2) is inside the polygon denoted by pts (in points1)
bool is_inside(vector<int> &pts, int pt, POINT<int> points1[], POINT<int> points2[], int xmin, int xmax, int ymin,
int ymax)
{
   if(points2[pt].P[0]<xmin || points2[pt].P[0]>xmax || points2[pt].P[1]<ymin || points2[pt].P[1]>ymax)
      return false;
   POINT<int> coord;
   for(int i=0;i<pts.size()-1;i++)
      if(linedist(points1[pts[i]], points1[pts[i+1]], points2[pt]) == 0)
         return true;
   while(1){
      POINT<int> candi;
      candi.P[0] = rand()%200000 + 200000;
      candi.P[1] = rand()%200000 + 200000;
      bool flag = true;
      for(int i=0;i<pts.size()-1;i++){
         if(linedist(candi, points2[pt], points1[pts[i]])==0){
            flag = false;
```

```
            break;
          }
        }
        if(!flag)continue;
        return getans(candi, pt, points1, points2, pts)%2;
    }
}
```

//Splay Tree

```
/*
Insert x
   - foo.insert(x)
Remove x
   - foo.remove(x)
Kth smallest number
   - if(foo.root == NULL || k>foo.root->size+1)
        printf("invalid\n");
     else
        find_kth(foo.root, k);
Number of occurance of x
   - find_c(foo.root, x)
*/
class node{
public:
   int data;
   node *left,*right,*parent;
   node(int val, node *l, node *r, node *par):data(val),left(l),right(r),parent(par),size(0){}
   int size;
};
class BST_splay{
public:
   node *root;
   node *insert(int value);
   void remove(int value);
   node *find(int value, bool getClosest);
   node *successor(node *n);
   void inorder(node *x);
   BST_splay():root(NULL){}
   void splay(node *x);
   void rotate_left(node *x);
   void rotate_right(node *x);
};
void BST_splay::rotate_left(node *x){
   x->parent->right = x->left;
   if(x->left != NULL)
      x->left->parent = x->parent;
   x->left = x->parent;
   if(x->parent->parent!=NULL){
      if(x->parent == x->parent->parent->left)
         x->parent->parent->left = x;
      else
         x->parent->parent->right = x;
```

```
    }
    node *tmp = x->parent;
    x->parent->size = (x->parent->left == NULL?0:(1+x->parent->left->size)) + (x->parent->right == NULL?0:(1+x->parent->right->size));
    x->parent = x->parent->parent;
    tmp->parent = x;
    x->size = (x->left == NULL?0:(1+x->left->size)) + (x->right == NULL?0:(1+x->right->size));
    if(x->parent == NULL)
        root = x;
}
void BST_splay::rotate_right(node *x){
    x->parent->left = x->right;
    if(x->right != NULL)
        x->right->parent = x->parent;
    x->right = x->parent;
    if(x->parent->parent!=NULL){
        if(x->parent == x->parent->parent->left)
            x->parent->parent->left = x;
        else
            x->parent->parent->right = x;
    }
    node *tmp = x->parent;
    x->parent->size = (x->parent->left == NULL?0:(1+x->parent->left->size)) + (x->parent->right == NULL?0:(1+x->parent->right->size));
    x->parent = x->parent->parent;
    tmp->parent = x;
    x->size = (x->left == NULL?0:(1+x->left->size)) + (x->right == NULL?0:(1+x->right->size));
    if(x->parent == NULL)
        root = x;
}
void BST_splay::splay(node *x){
    if(x == root)
        return;
    if(x == NULL)
        return;
    if(x->parent == root){   //zig
        if(x == x->parent->left)
            rotate_right(x);
        else
            rotate_left(x);
    }else if(x == x->parent->left && x->parent == x->parent->parent->right){  //zig-zag
        rotate_right(x);
        rotate_left(x);
    }else if(x == x->parent->right && x->parent == x->parent->parent->left){  //zig-zag
        rotate_left(x);
        rotate_right(x);
    }else if(x == x->parent->right && x->parent == x->parent->parent->right){ //zig-zig
        rotate_left(x->parent);
        rotate_left(x);
    }else if(x == x->parent->left && x->parent == x->parent->parent->left){  //zig-zig
        rotate_right(x->parent);
        rotate_right(x);
```

```
    }
    splay(x);
}
node *BST_splay::insert(int value){
    node *ret;
    if(root == NULL){
        root = new node(value,NULL,NULL,NULL);
        ret = root;
    }else{
        node *parent, *child = root;
        while(child!=NULL){
            parent = child;
            if(child->data > value)
                child = child->left;
            else if(child->data < value)
                child = child->right;
            else{
                splay(child);
                return child;
            }
        }
        if(parent->data > value){
            parent->left = new node(value, NULL, NULL, parent);
            ret = parent->left;
        }else{
            parent->right = new node(value, NULL, NULL, parent);
            ret = parent->right;
        }
        parent = ret->parent;
        while(parent!=NULL){
            parent->size++;
            parent = parent->parent;
        }
    }
    splay(ret);
    return ret;
}
node *BST_splay::find(int value, bool getClosest = false){
    if(root == NULL)
        return NULL;
    else{
        node *parent, *child = root;
        while(child!=NULL){
            parent = child;
            if(child->data > value)
                child = child->left;
            else if(child->data < value)
                child = child->right;
            else
                return child;
        }
        return getClosest?parent:NULL;
```

```
    }
}
node *BST_splay::successor(node *n){
    node *parent, *child = n->right;
    while(child != NULL){
        parent = child;
        child = child->left;
    }
    return parent;
}
void BST_splay::remove(int value){
    if(root == NULL)
        return;
    node *x = find(value, true);
    if(x->data != value){
        splay(x);
        return;
    }
    node *p = x->parent;
    if(x->left == NULL && x->right == NULL){
        if(root == x)
            root = NULL;
        else if(x == x->parent->left)
            x->parent->left = NULL;
        else
            x->parent->right = NULL;
        node *tmp = x->parent;
        while(tmp != NULL){
            tmp->size--;
            tmp = tmp->parent;
        }
        delete x;
    }else if(x->left != NULL && x->right == NULL){
        if(root == x){
            root = x->left;
            x->left->parent = NULL;
        }
        else if(x == x->parent->left){
            x->parent->left = x->left;
            x->left->parent = x->parent;
        }
        else{
            x->parent->right = x->left;
            x->left->parent = x->parent;
        }
        node *tmp = x->parent;
        while(tmp != NULL){
            tmp->size--;
            tmp = tmp->parent;
        }
        delete x;
    }else if(x->left == NULL && x->right != NULL){
```

//3D

* Point dividing line joining P(x1, y1,..) and Q(x2, y2,..)
  - (mx2+nx1/(m+n), ...)
* direction cosine -> unit vector along the line
  l^2+m^2+n^2 = 1
* cos(angle bet 2 lines) = (a1a2+b1b2+c1c2)/(sqrt(a1^2+b1^2+c1^2)sqrt(a2^2+b2^2+c2^2))
* sin is 0 when l1/l2 = m1/m2 = n1/n2
* Projection of line seg joining P(x1,y1,z1) and Q(x2,..)
  on a line(l,m,n) is l(x2-x1)+m(y2-y1)+n(z2-z1)
* perpendicular dist of a point P(x,y,z) from a line passing through A(a,b,c) and
  (l,m,n). AN = projection, PN = sqrt(AP^2-AN^2)
* ar(x) = 1/2((y1,z1,1),(y2,z2,1)..). area^2 = ar(x)^2 + ar(y)^2 + ar(z)^2
* eq of plane -> ax+by+cz+d = 0
  normal form -> lx+my+nz = p, p is length of normal from origin to plane, l,m,n is DC of normal
  plane passing through P(x1,y1,z1) and perpendicular to line (l,m,n) is l(x-x1)+m(y-y1)+...=0
  plane passng though three nin-collinear pts |(x-x1,y-y1,z-z1),(x2-x1,y2-y1,...| = 0
  intercept form -> x/a+y/b+z/c = 1
* perpendicular dist of point to plane = mod((ax1+by1+cz1+d)/sqrt(a^2+b^2+c^2))
* bisector planes
  plane1/sqrt(a1^2+..) = +- plane2/sqrt(a2^2..)

* Line
(x-x1)/l = (y-y1)/m = (z-z1)/n
2 point-> replace l with (x2-x1) and so on
shortest dist bet 2 lines
|(x2-x1,y2-y1,z2-z1),(l1,m1,n1),(l2,m2,n2)|/sqrt((l1m2-m1l2)^2+(m1n2-n1m2)^2+(l1n2-l2n1)^2)

```
    if(root == x){
        root = x->right;
        x->right->parent = NULL;
    }
    else if(x == x->parent->left){
        x->parent->left = x->right;
        x->right->parent = x->parent;
    }
    else{
        x->parent->right = x->right;
        x->right->parent = x->parent;
    }
    node *tmp = x->parent;
    while(tmp != NULL){
        tmp->size--;
        tmp = tmp->parent;
    }
    delete x;
}else if(x->left != NULL && x->right != NULL){
    node *s = successor(x);
    node *tmp = s->parent;
    p = tmp;
    while(tmp!=NULL){
        tmp->size--;
        tmp = tmp->parent;
    }
    s->size = x->size;
    if(x == root){
        if(s == x->right){
            p = s;
            s->parent = NULL;
            x->left->parent = s;
            s->left = x->left;
            root = s;
        }else{
            if(s->right == NULL)
                s->parent->left = NULL;
            else{
                s->parent->left = s->right;
                s->right->parent = s->parent;
            }
            s->parent = NULL;
            s->left = x->left;
            s->right = x->right;
            x->right->parent = s;
            x->left->parent = s;
            root = x;
        }
    }
    else{
        if(s == x->right){
            p = s;
```

Divides a given range 1..n into set of disjoint ranges which union to 1..n

eg 1..1234 breaks down to
1230..1234
1200..1229
1000..1199
100..999
10..99
1..9

```
# len = number of digits of n
calc(a,b,len):
        # a>b
        # len = length of a and b
        # calc performs the query
divide(n, len):
        ans = 0
        lt = 10**(len-1)
        for i=0..len-1 && n!=lt-1 :
                foo = n
                n = pow_10[i+1]*(n/pow_10[i+1])
                ans += calc(foo, n, len)
                n--
        if n!=lt-1:
                foo = n
                n = pow_10[len-1]
                ans += calc(foo, n, len)
                n--
        if n>1:
                ans += divide(n, len-1)
        return ans
```

```
            s->parent = x->parent;
            x->left->parent = s;
            s->left = x->left;
            if(x == x->parent->left)
               x->parent->left = s;
            else
               x->parent->right = s;
         }else{
            if(s->right == NULL)
               s->parent->left = NULL;
            else{
               s->parent->left = s->right;
               s->right->parent = s->parent;
            }
            s->parent = x->parent;
            if(x == x->parent->left)
               x->parent->left = s;
            else
               x->parent->right = s;
            s->left = x->left;
            s->right = x->right;
            x->right->parent = s;
            x->left->parent = s;
         }
      }
      delete x;
   }
   if(p!=NULL)
      splay(p);
}
void BST_splay::inorder(node *x){
   if(x!=NULL){
      inorder(x->left);
      cout<<x->data<<endl;
      inorder(x->right);
   }
}
BST_splay foo;
void find_kth(node *x, int k){
   while(1){
      int q = 1+(x->left==NULL?0:(1+x->left->size));
      if(k==q)
         break;
      if(q<k){
         if(x->right == NULL)
            return;
         k-=q;x = x->right;
      }else{
         if(x->left == NULL)
            return;
         x = x->left;
      }
```

```
int pow_10[10];
int len_int(int n){
   int cnt = 0;
   while(n){
      cnt++;
      n/=10;
   }
   return cnt;
}
int postfix_divide(int n, int len){
//    cout<<n<<endl;
//    if(n==10)return 5;
   if(n==0)return 0;
   int ans = 0;int foo;
   int lt = pow_10[len-1];
   for(int i=0;i<len-1 && n != lt-1; i++){
//          cout<<n<<" ";
         foo = n;
         n = pow_10[i+1]*(n/pow_10[i+1]);
//          cout<<n<<endl;
         ans+=calc2(foo,n,len);
//          cout<<calc2(foo,n,len)<<" -- "<<ans<<endl;
         n--;
   }
   if(n!=lt-1){
//       cout<<n<<" ";
      foo = n;
      n = pow_10[len-1];
      ans+=calc2(foo,n,len);
//       cout<<n<<endl;
//       cout<<calc2(foo,n,len)<<" -- "<<ans<<endl;
      n--;
   }
   if(n>1)
   ans += postfix_divide(n, len-1);
//    cout<<ans<<endl;
   return ans;
}
```

```
    }
    foo.splay(x);printf("%d\n",x->data);
}
int find_c(node *x, int val){
    if(x==NULL)return 0;
    int ans = 0;
    while(1){
        if(x->data<val){
            ans += 1+(x->left == NULL?0:(1+x->left->size));
            if(x->right != NULL)
                x = x->right;
            else break;
        }else if(x->data == val){
            ans += x->left == NULL? 0:(1+x->left->size);
            break;
        }else{
            if(x->left != NULL){
                x = x->left;
            }else break;
        }
    }
    foo.splay(x);
    return ans;
}

//Heavy Light
struct vertex;struct edge;struct chain;
struct vertex{
    vector<edge*> adj;
    edge* parent_edge;
    int heavy_chain_id;
    int size,depth;
    void init(){
        adj.clear();
        parent_edge = NULL;
        heavy_chain_id = -1;
        size = depth =0;
    }
};
struct edge{
    vertex *u,*v;
    chain* host_chain;
    int weight;
    void init(){
        u = v = NULL;
        host_chain = NULL;
    }

};
struct chain{
    vector<int> weights;
    vertex *head;int size;
```

```
int rank[10004],sum[10004],p,n,m,a,b,c;
struct foo
{
    int a,b,wt;
    friend bool operator<(const foo &a,const foo &b)
    {
        return a.wt<b.wt;
    }
}bar;
vector<struct foo> path;
void kruskal(){
    // Answer in sum[1]
    for(int i=1;i<=n;i++)
        {rank[i]=i;sum[i]=0;}
    sort(path.begin(),path.end());
    for(int i=0;i<path.size();i++){
        if(rank[path[i].a]!=rank[path[i].b]){
            if(rank[path[i].a]<rank[path[i].b]){
                int tmp=rank[path[i].b];
                for(int j=1;j<=n;j++)
                    if(rank[j]==tmp)
                        rank[j]=rank[path[i].a];
                sum[rank[path[i].a]]+=path[i].wt+sum[tmp];
            }else{
                int tmp=rank[path[i].a];
                for(int j=1;j<=n;j++){
                    if(rank[j]==tmp)
                    rank[j]=rank[path[i].b];
                }
                sum[rank[path[i].b]]+=path[i].wt+sum[tmp];
            }
        }
    }
}
```

```
    int *TREE;

    // Chain Methods
    void clear(){
        weights.clear();
    }
    void build(){
        size = weights.size();
        TREE = new int[size*4];
        build_segment_tree(1, 0, size-1);
    }
    int query(vertex *u, vertex *v){
        int x,y;
        x = u->depth-head->depth,
        y = v->depth-head->depth-1;
        return query_segment_tree(1, 0, size-1, x, y);
    }

    //Segment Tree Methods
    void build_segment_tree(int NODE,int a,int b){
        if(a==b){
            TREE[NODE] = weights[a];
            return;
        }
        int mid=(a+b)/2;
        int left=2*NODE,right=2*NODE+1;
        build_segment_tree(left,a,mid);
        build_segment_tree(right,mid+1,b);

        TREE[NODE] = TREE[left]+TREE[right];
    }
    int query_segment_tree(int NODE,int a,int b,int x,int y){
        if(x<=a && y>=b)return TREE[NODE];
        int mid=(a+b)/2;
        int left=2*NODE,right=2*NODE+1;
        if(y<=mid)return query_segment_tree(left,a,mid,x,y);
        if(x>mid)return query_segment_tree(right,mid+1,b,x,y);

        int left_query=query_segment_tree(left,a,mid,x,y);
        int right_query=query_segment_tree(right,mid+1,b,x,y);

        return left_query + right_query;
    }
};
vertex *root = NULL;
int chain_cnt = 0;
vertex V[10002];
edge E[10002];
chain heavy_chains[5002];
void build(vertex *V, vertex* par=NULL){
    if(par == NULL){
        root = V;
```

```
        V->depth = 0;
    }else
        V->depth = par->depth+1;

    V->size = 1;
    for(int i=0;i<V->adj.size();i++){
        vertex *nxt = (V->adj[i]->u == V)?V->adj[i]->v:V->adj[i]->u;
        if(nxt != par){
            nxt->parent_edge = V->adj[i];
            build(nxt, V);
            V->adj[i]->u = V;
            V->adj[i]->v = nxt;
            V->size += nxt->size;
        }
    }
}
void heavy_light_decomposition(vertex *V, chain *curr_chain = NULL){
    for(int i=0;i<V->adj.size();i++){
        vertex *nxt = V->adj[i]->v;
        if(nxt!=V){
            if(2*nxt->size >= V->size){
                if(curr_chain == NULL){
                    curr_chain = &heavy_chains[chain_cnt++];
                    curr_chain->head = V;
                }
                V->adj[i]->host_chain = curr_chain;
                curr_chain->weights.push_back(V->adj[i]->weight);
                heavy_light_decomposition(nxt, curr_chain);
                nxt->heavy_chain_id = V->heavy_chain_id = curr_chain-heavy_chains;
            }else
                heavy_light_decomposition(nxt);
        }
    }
}
vertex* LCA(vertex *u, vertex *v){
    #ifdef VVVV
    cout<<"FINDING LCA..."<<endl;
    #endif // VVVV
    int du,dv;
    vertex *hu, *hv;
    while(u!=v && (u->heavy_chain_id==-1 || u->heavy_chain_id!=v->heavy_chain_id)){
        if(u->heavy_chain_id == -1){
            du = u->depth;
            hu = u;
        }else
            du = heavy_chains[u->heavy_chain_id].head->depth, hu = heavy_chains[u->heavy_chain_id].head;

        if(v->heavy_chain_id == -1){
            dv = v->depth;
            hv = v;
        }else
            dv = heavy_chains[v->heavy_chain_id].head->depth, hv = heavy_chains[v->heavy_chain_id].head;
```

```
        if(du<dv){
            v = hv->parent_edge->u;
        }else{
            u = hu->parent_edge->u;
        }
    }
    if(u->depth < v->depth)
        return u;
    return v;
}
int query(vertex *u, vertex *v){
    //u is ancestor of v
    if(u == v)
        return 0;
    if(v->parent_edge->host_chain == NULL){  //Light
        return v->parent_edge->weight + query(u, v->parent_edge->u);
    }else{   //Heavy
        chain* host_chain = v->parent_edge->host_chain;
        if(host_chain->head->depth <= u->depth)
            return host_chain->query(u,v);
        else
            return host_chain->query(host_chain->head, v) + query(u, host_chain->head);
    }
}
int query_k(vertex *u, vertex *v, int k)
{
    if(k == 1)
        return v-V;
    if(v->parent_edge->host_chain == NULL)  //Light
        return query_k(u, v->parent_edge->u, k-1);
    else{   //Heavy
        chain* host_chain = v->parent_edge->host_chain;
        if(host_chain->weights.size()>= k)
            return query_k(u, v->parent_edge->u, k-1);
        else
            return query_k(u, host_chain->head, k-(v->depth-host_chain->head->depth));
    }
}
int main()
{
    in_T{
        int n,a,b,c;
        char str[10];
        in_I(n);
        root = NULL;
        chain_cnt = 0;
        for(int i=1;i<=n;i++)
            V[i].init();
        for(int i=0;i<n-1;i++){
            E[i].init();
            scanf("%d%d%d", &a, &b, &c);
            E[i].u = &V[a];
```

```
    E[i].v = &V[b];
    E[i].weight = c;
    V[a].adj.push_back(&E[i]);
    V[b].adj.push_back(&E[i]);
}
build(&V[1]);
heavy_light_decomposition(&V[1]);
for(int i=0;i<chain_cnt;i++)
    heavy_chains[i].build();
#ifdef VVVV
cout<<"Edges:"<<endl;
for(int i=0;i<n-1;i++){
    cout<<i<<": "<<(E[i].u-V)<<" "<<(E[i].v-V)<<" "<<(E[i].host_chain == NULL?"light":"heavy")<<endl;
}
cout<<endl<<"Vertices:"<<endl;
for(int i=1;i<=n;i++){
    cout<<i<<": "<<V[i].size<<" "<<V[i].depth<<" "<<(root == (V+i)?"root":"")<<endl;
}
#endif
while(1){
    in_S(str);
    if(str[1] == 'I'){
            // Distance between two nodes
        scanf("%d%d", &a, &b);
        #ifdef VVVV
        cout<<"QUERY "<<a<<" "<<b<<endl;
        #endif
        vertex *lca = LCA(&V[a], &V[b]);
        long long ans = 0;
        #ifdef VVVV
        cout<<"LCA: "<<lca-V<<endl;
        #endif
        ans = query(root, &V[a]);
        ans += query(root, &V[b]) - 2*query(root, lca);
        printf("%lld\n", ans);
    }
    else if(str[0] == 'K'){
        // cth node on path from a to b
        scanf("%d%d%d", &a, &b, &c);
        #ifdef VVVV
        cout<<"QUERY "<<a<<" "<<b<<" "<<c<<endl;
        #endif
        vertex *lca = LCA(&V[a], &V[b]);
        int total_nodes = V[a].depth-lca->depth + V[b].depth-lca->depth + 1;
        if(V[a].depth-lca->depth + 1 >= c)
            P_I(query_k(lca, &V[a], c));
        else
            P_I(query_k(lca, &V[b], total_nodes-c+1));
    }else
        break;
}
for(int i=0;i<chain_cnt;i++)
```

```
            heavy_chains[i].clear();
        cout<<endl;
    }
}
```

Euler Path

An undirected graph has an Eulerian cycle if and only if every vertex has even degree, and all of its vertices with nonzero degree belong to a single connected component.
An undirected graph has an Eulerian trail if and only if at most two vertices have odd degree, and if all of its vertices with nonzero degree belong to a single connected component.

A directed graph has an Eulerian cycle if and only if every vertex has equal in degree and out degree, and all of its vertices with nonzero degree belong to a single strongly connected component.

directed graph has an Eulerian trail if and only if at most one vertex has (out-degree) − (in-degree) = 1, at most one vertex has (in-degree) − (out-degree) = 1, every other vertex has equal in-degree and out-degree, and all of its vertices with nonzero degree belong to a single connected component of the underlying undirected graph.