HARD TO CRACK
(VJTI)

TEAM NOTEBOOK
(ICPC PUNE)

```
//ret.resize(v1.size()+v2.size()-1); //as if first ka size a,second ka b, so degree of ret=a-1+b-1=a+b-2,so size of ret=a+b-1
```

Mo's Algorithm:

```
const int N=2e5+5;
const int M=1e6+5;
struct data
{
	int l;int r;int idx;long long store_ans;};
int n, q, blocksz=1000;
int a[N];
data queries[N];
long long freq[M];
long long ans=0;
bool comp(data &d1, data &d2)
{
	int blocka=d1.l/blocksz;
	int blockb=d2.l/blocksz;
	if(blocka<blockb)
		return true;
	else if(blocka==blockb)
		return (d1.r<d2.r)^(blocka%2);
	else
		return false;
}
bool comp2(data &d1, data &d2)
{
	return d1.idx<d2.idx;
}
void update(long long k, int sign) //Sign 1 = Add, -1 = Remove
{
	if(sign==1)
	{
		ans-=freq[k]*freq[k]*k;
```

Game Theory:

1. If nim-sum is non-zero, player starting first wins.
2. Mex: smallest non-negative number not present in a set.
3. Grundy=0 means game lost.
4. Grundy=mex of all possible next states.
5. Sprague-Grundy theorem:

If a game consists of sub games (nim with multiple piles)
Calculate grundy number of each sub game (each pile)
Take xor of all grundy numbers:
If non-zero, player starting first wins.

Pattern Matching:

Suffix Arrays:(remaining to read)

```
struct suffix
{
  int index; // To store original index
  int rank[2]; // To store ranks and next rank pair
};
int cmp(struct suffix a, struct suffix b)
{
  return (a.rank[0] == b.rank[0])? (a.rank[1] < b.rank[1] ?1: 0):
		(a.rank[0] < b.rank[0] ?1: 0);
}
int *buildSuffixArray(char *txt, int n)
{
  struct suffix suffixes[n];
  for (int i = 0; i < n; i++)
  {
    suffixes[i].index = i;
    suffixes[i].rank[0] = txt[i] - 'a';
```

```
                freq[k]++;
                ans+=freq[k]*freq[k]*k;
        }
        else
        {
                ans-=freq[k]*freq[k]*k;
                freq[k]--;
                ans+=freq[k]*freq[k]*k;
        }
}
void calcmo()
{
        int moleft=1;
        int moright=0;
        for(int i=1;i<=q;i++)
        {
                int r=queries[i].r;
                int l=queries[i].l;
                while(moright<r)
                {
                        moright++;
                        update(a[moright], 1);
                }
                while(moright>r)
                {
                        update(a[moright], -1);
                        moright--;
                }
                while(moleft<l)
                {
                        update(a[moleft], -1);
                        moleft++;
                }
                while(moleft>l)
                {
                        moleft--;
                        update(a[moleft], 1);
                }
```

```
    suffixes[i].rank[1] = ((i+1) < n)? (txt[i + 1] -
'a'): -1;
  }
  sort(suffixes, suffixes+n, cmp);
  int ind[n];
  for (int k = 4; k < 2*n; k = k*2)
  {
    int rank = 0;
    int prev_rank = suffixes[0].rank[0];
    suffixes[0].rank[0] = rank;
    ind[suffixes[0].index] = 0;
    for (int i = 1; i < n; i++)
    {
      if (suffixes[i].rank[0] == prev_rank &&
          suffixes[i].rank[1] ==
suffixes[i-1].rank[1])
      {
        prev_rank = suffixes[i].rank[0];
        suffixes[i].rank[0] = rank;
      }
      else
      {
        prev_rank = suffixes[i].rank[0];
        suffixes[i].rank[0] = ++rank;
      }
      ind[suffixes[i].index] = i;
    }
    for (int i = 0; i < n; i++)
    {
      int nextindex = suffixes[i].index + k/2;
      suffixes[i].rank[1] = (nextindex < n)?

suffixes[ind[nextindex]].rank[0]: -1;
    }
    sort(suffixes, suffixes+n, cmp);
  }
```

```
            queries[i].store_ans=ans;
        }
}
/*
for(int i=1;i<=q;i++)
{
   cin>>queries[i].l>>queries[i].r;
   queries[i].idx=i;
}
sort(queries+1, queries+q+1, comp);
calcmo();
sort(queries+1, queries+q+1, comp2);
for(int i=1;i<=q;i++)
   cout<<queries[i].store_ans<<endl;
*/
```

HLD:

```
//just call init(),then updateUp or queryUp,always remember to clear adjacency adj[N],and g[N]
//,init() doesn't do it for u,subtree query for v [in[v],out[v]] in array rin
//1 based indexing for everything
ll n; //number of nodes
vector<pair<ll,ll> > adj[N]; //adjacency list of form (v,cost of edges)
ll treeSize[N]; //subtree size
ll cost[N]; //cost of a node->if nodes have a cost then cost[i]=givenCost[i],if edges have cost then cost[i]=givenCost(edge(parent[i],i))
//with some value to root(-inf for maximisation and vice versa could be kept)
ll depth[N]; //depth of a node
ll dp[N][20];

void dfs(ll curr,ll prev1=-1,ll depth1=0,ll cost1=-1)
//to initaialise dp,cost,depth
{
```

```
    // Store indexes of all sorted suffixes in the suffix array
   int *suffixArr = new int[n];
   for (int i = 0; i < n; i++)
      suffixArr[i] = suffixes[i].index;
   return  suffixArr;
}

void search(char *pat, char *txt, int *suffArr, int n)
{
   int m = strlen(pat);
   int l = 0, r = n-1;
   while (l <= r)
   {
      int mid = l + (r - l)/2;
      int res = strncmp(pat, txt+suffArr[mid], m);
      if (res == 0)
      {
         cout << "Pattern found at index " << suffArr[mid];
         return;
      }
      if (res < 0) r = mid - 1;
      else l = mid + 1;
   }
   cout << "Pattern not found";
}
```

KMP Algorithm:

```
ll arr[200005],arr2[200005];
ll n,w;
ll lps[200005];
void computeLps();
ll kmp()
{
   ll i=1,j=1; //i is pointing to main,j is pointing to pattern
```

Techniques:

1. For counting problems, try counting number of incorrect ways instead of correct ways.
2. Prune Infeasible/Inferior Search Space Early
3. Utilize Symmetries
4. Try solving the problem backwards
5.Binary Search the answer
6. Meet in the middle (Solve left half, Solve right half, combine)
7. Greedy
8. DP
9. Analyse complexity carefully
10. Reduce the problem to some standard problem
11. Add m when doing modular arithmetic.
12. Carefully analyse reasoning behind adding small details in the Q.
13. Use exponential search in case of unbounded search.

STL DS:

stack<type> name
empty(),size(),pop(),top(),push(x)

queue<type> name
empty(),size(),pop(),front(),back(),push(x)

```
    depth[curr]=depth1;
    dp[curr][0]=prev1;
    treeSize[curr]=1;
    cost[curr]=cost1;
    for(pair<ll,ll> x:adj[curr])
    {
        if(x.first!=prev1)
        {
            dfs(x.first,curr,depth1+1,x.second);
            treeSize[curr]+=treeSize[x.first];
        }
    }
}
ll t=0; //euler tour,and hld time
vector<ll> g[N];
ll sz[N],in[N],nxt[N],out[N],rin[N]; //nxt means name of head node of the chain this node belongs to
void dfs_sz(int v=1,int prev1=-1)
{
    sz[v] = 1;
    for(auto &u: g[v])
    {
        if(u!=prev1){
        dfs_sz(u,v);
        sz[v] += sz[u];
        if(sz[u] > sz[g[v][0]])
            swap(u, g[v][0]);
        }
    }
}

void dfs_hld(int v=1,int prev1=-1)
{
    in[v] = ++t;
    rin[in[v]] = v;
    for(auto u: g[v])
    {
        if(u!=prev1){
```

```
    computeLps();
    ll len1=0; //length matched uptil now
    ll res=0;
    while(i<=n)
    {
        if(arr[i]==arr2[j])
        {
            i++;
            j++;
            if(j==(w+1))
            {
                j=lps[j-1]+1;
                res++;
            } }
else
        {
            if(j!=1)
            {
                j=lps[j-1]+1;
            }
            else
            {
                i++;
                j=1;
            }
        }
    }
    return res;

}
void computeLps()
{
    lps[1]=0;
    ll i=2;
    ll len1=0; //best longest proper prefix which is also a suffix upto prev Val
    while(i<=w)
    {
```

priority_queue <type> name
empty(),size(),pop(),top(),push(x)

deque<type> name
pop_front(),pop_back(),push_front(),push_back(),size(),at(index),front()
,back()

set/multiset/map/multimap<type>name
begin(),end(),size(),empty(),insert(val),erase(itr or val),find(val),
lower_bound(val),upper_bound(val)
(lower bound includes val, upper bound does not)
pair<type,type> name (first and second)

STL Algorithms:

1.sort(first_iterator, last_iterator) – To sort the given vector.

2. reverse(first_iterator, last_iterator) – To reverse a vector.

3. *max_element (first_iterator, last_iterator) – To find the maximum element of a vector.

4. *min_element (first_iterator, last_iterator) – To find the minimum element of a vector.

5.lower_bound(first_iterator, last_iterator, x) – returns an iterator pointing to the first element in the range [first,last) which has a value not less than ‘x’.

6.upper_bound(first_iterator, last_iterator, x) – returns an iterator pointing to the first element in the range [first,last) which has a value greater than ‘x’.

```
        nxt[u] = (u == g[v][0] ? nxt[v] : u);
        dfs_hld(u,v);
        }
    }
    out[v] = t;
}

void init()
{
    nxt[1]=1;
    t=0;
    memset(dp,-1,sizeof(dp));
    //change this according to cost
    dfs(1);
    /*dont change this*/
    dfs_sz();
    dfs_hld();
    /*dont change this*/

    for(ll i=1;i<20;i++)
    {
        for(ll j=1;j<=n;j++)
        {
            if(dp[j][i-1]!=-1)
                dp[j][i]=dp[dp[j][i-1]][i-1];
        }
    }
    /*update part
    memset(tr,-1,sizeof(tr));
    for(ll i=1;i<=n;i++)
    {
        update(i,cost[rin[i]],1,n);
    }
    */
}

ll queryUp(ll a,ll b) //a is always down wala node
{
```

```
        if(arr2[i]==arr2[len1+1])
        {
            len1++;
            lps[i]=len1;
            i++;
        }
        else
        {
            if(len1!=0)
            {
                len1=lps[len1];
            }
            else
            {
                len1=0;
                lps[i]=len1;
                i++;
            }}}}
```

Advanced Techniques:

Catalan numbers:

1, 1, 2, 5, 14, 42, 132, 429, 1430,.....
C(n) =(1/(n+1)) * choose(2n, n);
C(n+1) = Summation(i = 0 to n) [C(i) * C(n-i)]

Centroid Decomposition:

```
int subtree[N], parentcentroid[N];
set<int> g[N];
ll nodes=0;
void dfs(int k, int par)
{
        nodes++;
        subtree[k]=1;
```

7.next_permutation(first_iterator, last_iterator) – This modified the vector to its next permutation.

8.prev_permutation(first_iterator, last_iterator) – This modified the vector to its previous permutation

9. random_shuffle(arr.begin(), arr.end());

Number Theory:

1. To calculate sum of factors of a number, we can find the number of prime factors and their exponents. N = ae1 * be2 * ce3 …
Then sum = (1 + a + a^2….)(1 + b + b^2 ..)...
Number of factors=(a+1)*(b+1)...

2.Every even integer greater than 2 can be expressed as the sum of 2 primes.

3. For rootn prime method, check for 2, 3 then:
for (i=5; i*i<=n; i=i+6) n%i and n%(i+2)

4. Number of divisors will be prime only if N=p^x where p is prime.

5. fib(n+m)=fib(n)fib(m+1)+fib(n-1)fib(m)

6. A number is Fibonacci if and only if one or both of (5*n2 + 4) or (5*n2 – 4) is a perfect square

7. every positive Every positive integer can be written uniquely as a sum of distinct non-neighbouring Fibonacci numbers.

8. Matrix multiplication

```
    if(b==-1)
        return -1;
    ll curr=a;
    ll res=-1;
    while(nxt[curr]!=nxt[b])
    {
        res=max(res,query(in[nxt[curr]],in[curr],1,n));
        curr=dp[nxt[curr]][0];
    }
    res=max(res,query(in[b],in[curr],1,n));
    //if wanted to do a upto b,but excluding b,comment above,and uncomment below,and make sure that segment tree handles l>r case,with inf for min,..
   // res=max(res,query(in[b]+1,in[curr],1,n));
    return res;
}


void updateUp(ll a,ll b,ll val) //a is always down wala node
{
    ll curr=a;
    while(nxt[curr]!=nxt[b])
    {
        updateRange(in[nxt[curr]],in[curr],val);
        curr=dp[nxt[curr]][0];
    }
    updateRange(in[b],in[curr],val);
    //if wanted to do a upto b,but excluding b,comment above,and uncomment below
    //updateRange(in[b]+1,in[curr],val);
}
```

Articulation Point (cut-vertices):

```
void APUtil(LL u, bool visited[], LL disc[],
        for(auto it:g[k])
        {
                if(it==par)
                        continue;
                dfs(it, k);
                subtree[k]+=subtree[it];
        }
}
int centroid(int k, int par)
{
        for(auto it:g[k])
        {
                if(it==par)
                        continue;
                if(subtree[it]>(nodes>>1))
                        return centroid(it, k);
        }
        return k;
}
void decompose(int k, int par)
{
        nodes=0;
        dfs(k, k);
        int node=centroid(k, k);
        parentcentroid[node]=par;
        for(auto it:g[node])
        {
                g[it].erase(node);
                decompose(it, node);
        }
}
//call decompose (1,-1)
```

Persistent Segment Tree(kth number):

```
ll n,arr[100005];
struct node
{
```

mul[i][j] += a[i][k]*b[k][j];

9. Root n under mod p exists only if
n^((p-1)/2) % p = 1

10..divisibility by 4: last 2 digits divisible by 4

11.divisibility by 8: last 3 digits divisible by 8

12. Divisibility by 3,9: sum of digs divisible by 3,9

13. Divisibility by 11: alternate (+ve,-ve) digit sum is divisible by 11

14. Divisibility by 12: divisible by 3 and 4

15. Divisibility by 13: alternating sum in blocks of 3 (L to R) div 13

16. Integral solution of ax+by=c exists if gcd(a,b) divides c

Extended Euclid's Algorithm:

```
1.  LL gcde(LL a,LL b,LL *x,LL *y)
2.  {
3.    if (a == 0)
4.    {
5.       *x = 0, *y = 1;
6.       return b;
7.    }
8.    LL x1, y1;
9.    LL gcd = gcde(b%a, a, &x1, &y1);
10.   *x = y1 - (b/a) * x1;
11.   *y = x1;
```

```
                                LL low[], LL parent[], bool
ap[])
{
   static LL time = 0;
   LL children = 0;
   visited[u] = true;
   disc[u] = low[u] = ++time;
   list<LL>::iterator i;
   for (i = adj[u].begin(); i != adj[u].end(); ++i)
   {
      LL v = *i;
      if (!visited[v])
      {
         children++;
         parent[v] = u;
         APUtil(v, visited, disc, low, parent, ap);
         if (parent[u] == NIL && children > 1)
           ap[u] = true;
         if (parent[u] != NIL && low[v] >= disc[u])
           ap[u] = true;
      }.
      else if (v != parent[u])
         low[u]  = min(low[u], disc[v]);
   }
}
void AP()
{
   bool *visited = new bool[V];
   LL *disc = new LL[V];
   LL *low = new LL[V];
   LL *parent = new LL[V];
   bool *ap = new bool[V];
   for (LL i = 0; i < V; i++)
   {
      parent[i] = NIL;
      visited[i] = false;
      ap[i] = false;
   }
```

```
        node *left,*right;
        ll cnt=0;
        node(node *left,node *right,ll cnt)
        {
                this->left=left;
                this->right=right;
                this->cnt=cnt;
        }
        //since insert return a ptr to child
        node* insert(ll l,ll r,ll pos);

};
//base pointer pointing to null tree
node *null=new node(NULL,NULL,0);
//in this all leaf nodes ka left and right points to
base pointer
node* node:: insert(ll l,ll r,ll pos)
{
        //pos in range
        if(pos>=l && pos<=r)
        {
                if(l==r)
                {
                        return new
node(null,null,this->cnt+1);
                }
                ll m=(l+r)>>1;
                return new
node(this->left->insert(l,m,pos),this->right->inse
rt(m+1,r,pos),this->cnt+1);
        }
        //if out of range,we can use previous
version tree node
        return this;
}
ll query(node *a,node *b,ll l,ll r,ll w)
{
        if(l==r)
```

```
    return gcd;
}
```

To find inverse of a wrt m:
gcde(a,m,&x,&y);
x is the inverse of a.

Segmented Sieve for primes

```
void segsieve(LL l,LL r)
{
   LL limit = floor(sqrt(r))+1;
   vector<LL> prime;
   sieve(limit, prime);
      limit=r-l+1;
      bool mark[limit+1];
      memset(mark, true, sizeof(mark));
     //True= is prime
      for (int i = 0; i < prime.size(); i++)
      {
         int loLim = floor(l/prime[i]) *
    prime[i];
         if (loLim < l)
            loLim += prime[i];

         for (int j=loLim; j<=r; j+=prime[i])
            mark[j-l] = false;
      }
}
```

Matrix Exponentiation

```
LL power(LL F[3][3], LL n)
{
  LL M[3][3] = {{1,1,1}, {1,0,0}, {0,1,0}};
  if (n==1)
     return F[0][0] + F[0][1];
  power(F, n/2);
  multiply(F, F);
  if (n%2 != 0)
     multiply(F, M);
  for (LL i = 0; i < V; i++)
     if (visited[i] == false)
        APUtil(i, visited, disc, low, parent, ap);
  for (LL i = 0; i < V; i++)
     if (ap[i] == true)
        cout << i << " ";
}
```

Bridges:

Replaceboth condition for articulation point with
if (low[v] > disc[u])

Euler path/circuit: path->every edge exactly once

Euler path in undirected graph:
All vertices have even degree except or 2 have odd degrees, and vertex with non zero degree in same component.

Euler Circuit in undirected graph:
All vertices have even degree and vertex with non zero degree in same component.

Euler circuit in directed graph:
All no zero degree vertices are a part of a single strongly connected component and indegree and outdegree of all vertices is same.

Euler path in directed graph:
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.

Hierholzer's algorithm for directed graph:

```
        return l;
        ll m=(l+r)>>1;
        ll cnt1=(b->left->cnt-a->left->cnt);
        if(cnt1>=w)
        return query(a->left,b->left,l,m,w);
        return
query(a->right,b->right,m+1,r,w-cnt1);

}
/*
null->left=null->right=null;
root[0]=null;
for(ll i=1;i<=n;i++)
{
   root[i]= (root[i-1])->insert(1,cnt/N(of segment
tree),arr[i]);
}
ll res=query(root[l-1],root[r],1,cnt/N(of segment
tree),k);
*/
```

2-sat:

```
struct TwoSAT
{
        static const int MAXV=2e5+5;(2*number
of variables)
        int n, cnt; //n is number of
vertices(2*number of variables)
        vector<int> g[MAXV], rg[MAXV];
//g=forward, rg=backward
        bool vis[MAXV];
        int order[MAXV], comp[MAXV];
        void init(int curn)
        {
                n=curn;
                for(int i=0;i<n;i++)
                {
```

```
    return F[0][0] + F[0][1] ;
}

LL findNthTerm(LL n)
{
    LL F[3][3] = {{1,1,1}, {1,0,0}, {0,1,0}} ;
    return power(F, n-2);
}
```

Euler's totient:

Number of integers coprime to n less than n

```
LL phi(LL n)
{
    LL result = n;
    for (LL p=2; p*p<=n; ++p)
    {
        if (n % p == 0)
        {
            while (n % p == 0)
                n /= p;
            result -= result / p;
        }
    }
    if (n > 1)
        result -= result / n;
    return result;
}
```

Largest power of p that divides n!:

```
// Returns largest power of p that divides n!
int largestPower(int n, int p)
{
    // Initialize result
    int x = 0;

    // Calculate x = n/p + n/(p^2) + n/(p^3) + ....
```

```
void printCircuit(vector< vector<int> > adj)
{
    unordered_map<int,int> edge_count;
    for (int i=0; i<adj.size(); i++)
    {
        edge_count[i] = adj[i].size();
    }
    if (!adj.size())
        return;
    stack<int> curr_path;
    vector<int> circuit;
    curr_path.push(0);
    int curr_v = 0;
    while (!curr_path.empty())
    {
        if (edge_count[curr_v])
        {
            curr_path.push(curr_v);
            int next_v = adj[curr_v].back();
            edge_count[curr_v]--;
            adj[curr_v].pop_back();
            curr_v = next_v;
        }
        else
        {
            circuit.push_back(curr_v);
            curr_v = curr_path.top();
            curr_path.pop();
        }
    }
    for (int i=circuit.size()-1; i>=0; i--)
    {
        cout << circuit[i];
        if (i)
            cout<<" -> ";
    }
}
```

```
                g[i].clear();
                rg[i].clear();
        }
    }
    void add(int u, int v)
    {
        g[u].push_back(v);
        rg[v].push_back(u);
    }
    void dfs1(int u)
    {
        vis[u] = true;
        for(auto it:g[u])
            if(!vis[it])
                dfs1(it);
        order[cnt++] = u;
    }
    void dfs2(int u, int c)
    {
        comp[u] = c;
        for(auto it:rg[u])
            if(comp[it]==-1)
                dfs2(it, c);
    }
    int solve(vector<int> &ans)
    {
        cnt=0;
        memset(vis, 0, sizeof(vis));
        for(int i=0;i<n;i++)
            if(!vis[i])
                dfs1(i);
        memset(comp, -1, sizeof(comp));
        int grp=0;
        for(int i=n-1;i>=0;i--)
        {
            int u=order[i];
            if(comp[u] == -1)
                dfs2(u, grp++);
```

```
    while (n)
    {
        n /= p;
        x += n;
    }
    return x;
}
```

nCr (with lucas Theorem):(nCr%p)

P ->prime,complexity->O(p^2logn)

```
1.  LL ncrp(LL n, LL r, LL p)
2.  {
3.    LL C[r+1];
4.    memset(C, 0, sizeof(C));
5.    C[0] = 1;
6.    for (LL i = 1; i <= n; i++)
7.    {
8.        for ( LL j = min(i, r); j > 0; j--)
9.            C[j] = (C[j] + C[j-1])%p;
10.   }
11.   return C[r];
12. }
13. LL ncrpl(LL n,LL r, LL p)
14. {
15.  if (r==0)
16.     return 1;
17.  int ni = n%p, ri = r%p;
18.  return (ncrpl(n/p, r/p, p) *
19.          ncrp(ni, ri, p)) % p;
20. }
```

Bipartite graph:

Coloring possible with 2 colors.

Max flow Algorithm(V^3):

```
//To obtain the actual flow values, look at all edges with capacity > 0
//Zero capacity edges are residual edges
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) {}
};
//to check flow just dont consider the edges with 0 capcities,as when dding (a->b with capaacity c,we add b->a with capacity 0)
//and it could have negative flow which we ignore
/*
for(ll i=1;i<=2*n;i++)
{
   for(auto&edge:obj.g[i])
   {
      if((edge.to!=0) && (edge.to!=(2*n+1)) && (edge.cap!=0))
      {

        ans[i][edge.to-n]+=edge.flow;
        ans[i][i]-=edge.flow;
      }}}*/
struct PushRelabel
{
        int n;
        vector<vector<edge> > g;
        vector<long long> excess;
        vector<int> height, active, count;
```

```
                }
                for(int i=0;i<n;i+=2)
                        if(comp[i]==comp[i^1])
                                return 0;
                ans.clear();
                for(int i=0;i<n;i+=2)
                {
                        int choose = (comp[i] > comp[i^1]) ? i : (i^1);
                        ans.push_back(choose);
                }
                return 1;
        }
};sat;

/*use 0 based indexing of variables
sat.init(number of vertices) where number of vertices=2*number of variables
variable no.i at 2*i,and 2*i+1,always keep p at 2*i,~p at 2*i+1,as ans will be value of p,and not ~p
*/
```

Sos Dp:

```
const ll K=22;
ll dp[(1<<K)+5][K]; //dp[mask][i] represents the f(x) for all x -subset of mask(ie some operations over all subset of mask ,eg:sum(arr[x]))
      //changes in first i elements only

//initialise dp
//memset(dp,0,sizeof(dp));

//dp[mask][0]=f(mask)->no changes
for(ll i=1;i<=n;i++)
{
   dp[arr[i]][0]=arr[i];
```

Chinese Remainder Theorem:

The **Chinese remainder theorem** is a theorem of number theory, which states that if one knows the remainders of the Euclidean division of an integer *n* by several integers, then one can determine uniquely the remainder of the division of *n* by the product of these integers, under the condition that the divisors are pairwise coprime.

```
1.  LL crt(LL num[], LL rem[], LL k)
2.  {
3.     LL prod = 1;
4.     for (int i = 0; i < k; i++)
5.        prod *= num[i];
6.     LL result = 0;
7.     for (int i = 0; i < k; i++)
8.     {
9.        LL pp = prod / num[i];
10.       LL inv,y;
11.       gcde(pp,num[i],&inv,&y);
12.       result += rem[i] * inv * pp;
13.    }
14.    return result % prod;
15. }
```

For combining wrt a large number, use it 2 numbers at a time.

Wilson's theorem(doubt):
((p-1)!)%p=-1

Number of solutions to a linear eqn:(doubt):

```
LL countSol(LL coeff[], LL start, LL end, LL rhs)
{
   // Base case
   if (rhs == 0)
      return 1;
```

```
        queue<int> Q;
        PushRelabel(int n): //number of nodes
                n(n), g(n), excess(n), height(n),
active(n), count(2*n) {}
        void addEdge(int from, int to, int cap)
//direction dependent
        {
                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));
        }
        void enqueue(int v)
        {
                if(!active[v] && excess[v] > 0)
                {
                        active[v]=true;
                        Q.push(v);
                }
        }
        void push(edge &e)
        {
                int amt=(int)min(excess[e.from],
(long long)e.cap - e.flow);
                if(height[e.from]<=height[e.to] ||
amt==0)
                        return;
                e.flow += amt;
                g[e.to][e.index].flow -= amt;
                excess[e.to] += amt;
                excess[e.from] -= amt;
                enqueue(e.to);
        }
        void relabel(int v)
        {
                count[height[v]]--;
```

```
}
//for optimised approach use i&1,(i+1)&1
for(ll i=1;i<=K;i++)
{
   for(ll mask=0;mask<(1<<(K));mask++)
   {
      if(mask&(1<<(K-i)))
      {
         //instead of +,anything like &,or
sth..could be there

dp[mask][i]=dp[mask][i-1]+dp[mask^(1<<(K-i))
][i-1];
      }
      else
      {
         dp[mask][i]=dp[mask][i-1];
      }
   }
}
```

Parallel binary Search(Algorithm):

Q. There are *N* member states and *M* sectors. Each sector is owned by a member state. There are *Q* queries, each of which denote the amount of meteor shower in a [*L*, *R*] range of sectors on that day. The *ith* member state wants to collect *reqd*[*i*] meteors over all its sectors. For every member state, what is the minimum number of days it would have to wait to collect atleast the required amount of meteors?

```
    LL result = 0;  // Initialize count of solutions

    // One by subtract all smaller or equal
coefficiants and recur
    for (LL i=start; i<=end; i++)
      if (coeff[i] <= rhs)
        result += countSol(coeff, i, end, rhs-coeff[i]);

    return result;
}
```

Sum of GP:

```
long long gp(LL r, LL p,LL m){
if(p==0)
return 1;
if(p==1)
return 1;
LL ans=0;
if(p%2==1){
ans=Mpow(r,p-1,m);
ans=(ans+((1+r)*gp(Mpow(r,2,m),(p-1)/2,m))%m)
%m;
}
else{
    ans=((1+r)*gp(Mpow(r,2,m),p/2,m))%m;
}
return ans;
}
```

Ternary Search (max of unimodal function):

```
double ts(double start, double end)
{
    double l = start, r = end;

    for(int i=0; i<200; i++) {
      double l1 = (l*2+r)/3;
                int d=2*n;
                for(auto &it:g[v])
                {
                        if(it.cap-it.flow>0)
                                d=min(d,
height[it.to]+1);
                }
                height[v]=d;
                count[height[v]]++;
                enqueue(v);
        }
        void gap(int k)
        {
                for(int v=0;v<n;v++)
                {
                        if(height[v]<k)
                                continue;
                        count[height[v]]--;
                        height[v]=max(height[v],
n+1);
                        count[height[v]]++;
                        enqueue(v);
                }
        }
        void discharge(int v)
        {
                for(int i=0; excess[v]>0 &&
i<g[v].size(); i++)
                        push(g[v][i]);
                if(excess[v]>0)
                {
                        if(count[height[v]]==1)
                                gap(height[v]);
                        else
                                relabel(v);
                }
        }
        long long max_flow(int source, int dest)
```

```
for all logQ/or any like 30 steps:
    clear range tree and linked list check
    for all member states i:
        if L[i] != R[i]:
            mid = (L[i] + R[i]) / 2
            insert i in check[mid]
    for all queries q:
        apply(q)
        for all member states m in check[q]:
            if m has requirements fulfilled:
                R[m] = q

            else:
                L[m] = q + 1
```

Closest Pair of Points(Nlog^2N):

```
struct Point
{
        int x, y;

        Point operator -(Point p)
        {
                return {x-p.x, y-p.y};
        }

        int dist()
        {
```

```
    double l2 = (l+2*r)/3;
    //cout<<l1<<" "<<l2<<endl;
    if(func(l1) > func(l2)) r = l2; else l = l1;
    }
  return func(r);
}
```

Data Structures:

Segment tree(point update to val,and range Sum):

```
//just do memset(tr,0,sizeof(tr))
ll tr[4*N];
void update(ll indx,ll val,ll ogL,ll ogR,ll si=1)
{
  if((indx<ogL) or (indx>ogR))
    return;
  if(ogL==ogR)
  {
    tr[si]=val;
    return;
  }
  ll mid=(ogL+ogR)>>1;
  update(indx,val,ogL,mid,si<<1);
  update(indx,val,mid+1,ogR,(si<<1)|1);
  tr[si]=tr[si<<1]+tr[(si<<1)|1];
}
ll query(ll l,ll r,ll ogL,ll ogR,ll si=1)
{
  if(l>r)
    return 0;
  if((r<ogL) or (l>ogR))
    return 0;
  if((ogL>=l) && (ogR<=r))
  {
    return tr[si];
```

```
    {
        count[0] = n-1;
        count[n] = 1;
        height[source] = n;
        active[source] = active[dest] = 1;
        for(auto &it:g[source])
        {
            excess[source]+=it.cap;
            push(it);
        }
        while(!Q.empty())
        {
            int v=Q.front();
            Q.pop();
            active[v]=false;
            discharge(v);
        }
        long long max_flow=0;
        for(auto &e:g[source])
            max_flow+=e.flow;
        return max_flow;
    }
};
PushRelabel obj(2*n+2); //obj(number of nodes)
obj.addEdge(src,dest,capacity)
ll flow=obj.max_flow(0,2*n+1); (src,dest)
```

Min cost max flow:

```
//uncomment depending on cost type
//typedef long long int ct
//typedef double ct
//typedef int ct
//typedef long double ct

//also DO : ll MAXDIST=1e9;(or more if u
want,depending on max possible cost)
```

```
        return x*x + y*y;
    }
};
bool by_x(Point &a, Point &b)
{
    return a.x < b.x;
}
bool by_y(Point &a, Point &b)
{
    return a.y < b.y;
}
int n, ans=1e18;
int a[N], pref[N];
Point pt[N];

int solve(int L, int R)
{
    if(L==R)
        return 1e18;
    int M=(L+R)/2;
    sort(pt+L, pt+R+1, by_x);
    int d=min(solve(L, M), solve(M+1, R));
    int midx=pt[L+(R-L+1)/2].x;
    vector<Point> v;
    for(int i=L;i<=R;i++)
    {
        if(Point{pt[i].x-midx, 0}.dist()<d)
        {
            v.push_back(pt[i]);
        }
    }
    sort(v.begin(), v.end(), by_y);
    for(int i=0;i<v.size();i++)
    {
        for(int j=i+1;j<v.size();j++)
        {
            if(Point{0,
v[i].y-v[j].y}.dist()>d)
```

```
    }
    ll mid=(ogL+ogR)>>1;
    return
query(l,r,ogL,mid,(si<<1))+query(l,r,mid+1,ogR,(si<<1)|1);
}
```

Lazy Segment tree(range addition,range sum):

```
ll tr[4*N],lazy[4*N];
//initilialise both with 0
void push(ll ogL,ll ogR,ll si)
{
    if(ogL!=ogR)
    {
        lazy[si<<1]+=lazy[si];
        lazy[(si<<1)|1]+=lazy[si];
    }
    tr[si]+=lazy[si]*(ogL-ogR+1);  //for minimum
or max ,do tr[si]+=lazy[si]
    lazy[si]=0;
}
void update(ll l,ll r,ll val,ll ogL,ll ogR,ll si=1)
{
    if(lazy[si])
    {
        push(ogL,ogR,si);
    }
    if((r<ogL) or (l>ogR))
        return;
    if((ogL>=l) && (ogR<=r))
    {
        lazy[si]+=val;
        push(ogL,ogR,si);
        return;
    }
    ll mid=(ogL+ogR)>>1;
    update(l,r,val,ogL,mid,si<<1);
```

```
//Works for negative costs, but does not work for
negative cycles
//Complexity: O(min(E^2 *V log V, E logV * flow))
struct edge
{
        int to, flow, cap;
        ct cost;
        int rev;
};
struct MinCostMaxFlow
{
        int nodes;
        vector<int> curflow, prevedge, prevnode, q;
        vector<ct> pot,prio;
        vector<bool> inqueue;
        vector<vector<edge> > graph;
        MinCostMaxFlow() {}
        MinCostMaxFlow(int n): nodes(n), prio(n,
0), curflow(n, 0),  //number of nodes
        prevedge(n, 0), prevnode(n, 0), q(n, 0), pot(n,
0), inqueue(n, 0), graph(n) {}
        void addEdge(int source, int to, int capacity,
ct cost) //direction dependent
        {
                edge a = {to, 0, capacity, cost,
(int)graph[to].size()};
                edge b = {source, 0, 0, -cost,
(int)graph[source].size()};
                graph[source].push_back(a);
                graph[to].push_back(b);
        }
        void bellman_ford(int source, vector<ct>
&dist)
        {
                fill(dist.begin(), dist.end(),
MAXDIST);
                dist[source] = 0;
```

```
                        break;
                d=min(d, (v[i]-v[j]).dist());
            }
        }
        return d;
}
//pt[i]={x,y} //initialise like this
//int res=solve(1,n);
```

Python Syntax:

```
1 ) from fractions import gcd  # gcd(a,b)
2) raw_input()  # returns string , int()
3) map(int,raw_input(),split())  # for full int array
4) sorted(A)
5 ) Fast I/O
from sys import stdin, stdout
        stdin.readline() # in place of raw_input()
        stdout.write(str())
6) List functions , let A be list
        A.count('apple') # returns count
        A.reverse() , len(A)
        A.sort()
        A.pop(),A.append()
        del A[l:r] # deletes from (l,r) inclusive
7) Sets
        A={1,2,3,2} # will remove duplicates
        A=set('vinit') # A will contain
{'v','i','n','t'}
        a-b,a|b,a&b
8) dictionary
        A={'a':23,'b':34}
        list(A) # will contain list of keys
A=['a','b']
        A.get(key_name) # returns None if not
found
        # don't use A[key_name]
```

```
    update(l,r,val,mid+1,ogR,(si<<1)|1);
    tr[si]=tr[si<<1]+tr[(si<<1)|1];
}
ll query(ll l,ll r,ll ogL,ll ogR,ll si=1)
{
    if(lazy[si])
    {
        push(ogL,ogR,si);
    }
    if((r<ogL) or (l>ogR))
        return 0;
    if((ogL>=l) && (ogR<=r))
    {
        return tr[si];
    }
    ll mid=(ogL+ogR)>>1;
    return
query(l,r,ogL,mid,(si<<1))+query(l,r,mid+1,ogR,(si<<1)|1);
}
```

Lazy Segment tree(range update to val,range sum):

```
ll tr[4*N],lazy[4*N];
//initialialise tr with 0,and lazy with -1(or anything like -inf,which will never come in update)
void push(ll ogL,ll ogR,ll si)
{
    if(ogL!=ogR)
    {
        lazy[si<<1]=lazy[si];
        lazy[(si<<1)|1]=lazy[si];
    }
    tr[si]=lazy[si]*(ogL-ogR+1);
    lazy[si]=-1;
}
void update(ll l,ll r,ll val,ll ogL,ll ogR,ll si=1)
```

```
                int qt=0;
                q[qt++] = source;
                for(int qh=0;(qh-qt)%nodes!=0;qh++)
                {
                        int u = q[qh%nodes];
                        inqueue[u] = false;
                        for(auto &e : graph[u])
                        {
                                if(e.flow >= e.cap)
                                        continue;
                                int v = e.to;
                                ct newDist = dist[u] +
e.cost;
                                if(dist[v] > newDist)
                                {
                                        dist[v] =
newDist;
                                        if(!inqueue[v])
                                        {

inqueue[v] = true;
                                                q[qt++
% nodes] = v;
                                        }
                                }
                        }
                }
        }
        pair<int, ct> minCostFlow(int source, int
dest, int maxflow) //max flow,min cost   (pass
maxflow as 1e9,or depending on largest possible
max flow)
        {
                bellman_ford(source, pot);
                int flow = 0;
                ct flow_cost = 0;
                while(flow < maxflow)
                {
```

```
9) from collections import deque
        A=deque(list_name)
        append(),popleft()
10) import bisect
        A =[1,3,4,5,5,5,6] # must be sorted
        bisect.bisect(A,5) # returns index after all
5's , i.e 6
        bisect.bisect_left(A,5) #lower_bound
index , i.e 3
11)
from operator import itemgetter, attrgetter
class node:
    def __init__(self, L, R):
        self.L = L
        self.R = R
    def __repr__(self):
        return repr((self.L, self.R))

A=sorted(A, key=attrgetter('L'))
12)bin(ans).count("1") #counts number of set bits
in binary representation of ans
13)2d array input
arr=[[] for i in xrange(100005)]
for i in xrange(n):
        arr[i]= map(int, raw_input().split())
14)initialising and creating 2d array
dp=[[0]*K for i in range(N)]
1d array: a=[0]*N
15)array of map
arr3=list( {} for i in xrange(1005) )
if x in arr3[i]:
        arr3[i][x]+=1
    else:
        arr3[i][x]=1
16)
Dict functions:clear(),get(key),pop(key),
```

```
{
   if(lazy[si]!=-1)
   {
      push(ogL,ogR,si);
   }
   if((r<ogL) or (l>ogR))
      return;
   if((ogL>=l) && (ogR<=r))
   {
      lazy[si]=val;
      if(lazy[si]!=-1)
      push(ogL,ogR,si);
      return;
   }
   ll mid=(ogL+ogR)>>1;
   update(l,r,val,ogL,mid,si<<1);
   update(l,r,val,mid+1,ogR,(si<<1)|1);
   tr[si]=tr[si<<1]+tr[(si<<1)|1];
}
ll query(ll l,ll r,ll ogL,ll ogR,ll si=1)
{
   if(lazy[si]!=-1)
   {
      push(ogL,ogR,si);
   }
   if((r<ogL) or (l>ogR))
      return 0;
   if((ogL>=l) && (ogR<=r))
   {
      return tr[si];
   }
   ll mid=(ogL+ogR)>>1;
   return
query(l,r,ogL,mid,(si<<1))+query(l,r,mid+1,ogR,(s
i<<1)|1);
}
```

```
                priority_queue<pair<ct, int>,
vector<pair<ct, int> >, greater<pair<ct, int> > > q;
                q.push({0, source});
                fill(prio.begin(), prio.end(),
MAXDIST);
                prio[source] = 0;
                curflow[source] = INT_MAX;
                while(!q.empty())
                {
                        ct d = q.top().first;
                        int u = q.top().second;
                        q.pop();
                        if(d != prio[u])
                                continue;
                        for(int
i=0;i<graph[u].size();i++)
                        {
                                edge
&e=graph[u][i];
                                int v = e.to;
                                if(e.flow >=
e.cap)

continue;
                                ct newPrio =
prio[u] + e.cost + pot[u] - pot[v];
                                if(prio[v] >
newPrio)
                                {
                                        prio[v]
= newPrio;

q.push({newPrio, v});

prevnode[v] = u;

prevedge[v] = i;
```

Big integer c++:

```
#include <boost/multiprecision/cpp_int.hpp>
using namespace boost::multiprecision;

//arbitrary precision
cpp_int u = 1;
//fixed precision
int128_t v = 1;
```

2D bit (point update and query):

```
//initialise bit to 0
//memset(bit,0,sizeof(bit));
//1 based indexing
//query - prefix sum
//2d array ->1 based indexing
ll bit[N][N];
void update(ll x,ll y,ll val,ll n,ll m)
{
   for(ll i=x;i<=n;i+=i&-i)// (instead of n,m u can
do <N where N>max(n,m))
   {
      for(ll j=y;j<=m;j+=j&-j)
      {
         bit[i][j]+=val;
      }
   }
}

ll query(ll x,ll y)
{
   ll res=0;
   for(ll i=x;i>=1;i-=i&-i)
   {
      for(ll j=y;j>=1;j-=j&-j)
```

Policy based DS:(to read)

```
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
typedef tree<int, null_type, less<int>, rb_tree_tag,
tree_order_statistics_node_update> pbds;
insert(val),erase(),order_of_key(),find_by_order()
```

Graph Theory

Bellman-Ford(for negative edges):

```
void BellmanFord(struct Graph* graph, LL src)
{
   LL V = graph->V;
   LL E = graph->E;
   LL dist[V];
   for (LL i = 0; i < V; i++)
      dist[i]   = INT_MAX;
   dist[src] = 0;
   for (LL i = 1; i <= V-1; i++)
   {
      for (LL j = 0; j < E; j++)
      {
         LL u = graph->edge[j].src;
         LL v = graph->edge[j].dest;
         LL weight = graph->edge[j].weight;
         if (dist[u] != INT_MAX && dist[u] +
weight < dist[v])
            dist[v] = dist[u] + weight;
      }
   }//to check for negative weight cycle, repeat
above
}   // if shorter path is found, cycle exists
```

```
curflow[v] = min(curflow[u], e.cap - e.flow);
                                    }
                                }
                        }
                        if(prio[dest] == MAXDIST)
                                break;
                        for(int i=0;i<nodes;i++)
                                pot[i]+=prio[i];
                        int df = min(curflow[dest],
maxflow - flow);
                        flow += df;
                        for(int
v=dest;v!=source;v=prevnode[v])
                        {
                                edge &e =
graph[prevnode[v]][prevedge[v]];
                                e.flow += df;
                                graph[v][e.rev].flow -=
df;
                                flow_cost += df *
e.cost;
                        }
                }
                return {flow, flow_cost};
        }
};
//MinCostMaxFlow obj(2*n+2);
//obj.addEdge(i,j+n,1,sqrt((y[j]-y[i])*(y[j]-y[i])+(x[j]
-x[i])*(x[j]-x[i])));
//pair<int,ct> flow=obj.minCostFlow(0,2*n+1,1e9);
,rest same as prev
```

Maximum Bipartite Matching:

```
bool bpm(bool bpGraph[M][N], int u, bool seen[],
int matchR[])
{
```

```
    {
        res+=bit[i][j];
    }
  }
  return res;
}
//(upper left,bottom right)  x is row,y is column
ll subMatrixSum(ll x1,ll y1,ll x2,ll y2)
{
  return
query(x2,y2)-query(x1-1,y2)-query(x2,y1-1)+qu
ery(x1-1,y1-1);
}
```

2D bit Submatrix Update:

```
#define N 505
//N>max(n,m)
ll bit[4][N][N];
//initialise bit to 0
//lets say array given,two ways->first for each
element call updateSubMatrix(i,j,i,j) ,
//or calculate sum[i][j],and treat matrix as
0's(baadmei bas add kar dena)
void update(ll node,ll x,ll y,ll v)
{
  for(ll i=x;i<N;i+=i&-i)
  {
    for(ll j=y;j<N;j+=j&-j)
      bit[node][i][j]+=v;
  }
}

ll query(ll node,ll x,ll y)
{
  ll ans=0;
  for(ll i=x;i>=1;i-=i&-i)
```

Prim's Algorithm for MST

```
void primMST()
{

priority_queue<pair<LL,LL>,greater<pair<LL,LL>>> pq;
   LL src = 0;
   vector<LL> key(V, INF);
   vector<LL> parent(V, -1);
   vector<bool> inMST(V, false);
   pq.push(make_pair(0, src));
   key[src] = 0;
   while (!pq.empty())
   {
      LL u = pq.top().second;
      pq.pop();
      inMST[u] = true;  // Include vertex in MST
      list< pair<LL, LL> >::iterator i;
      for (i = adj[u].begin(); i != adj[u].end(); ++i)
      {
         LL v = (*i).first;
         LL weight = (*i).second;
         if (inMST[v] == false && key[v] > weight)
         {
            key[v] = weight;
            pq.push(make_pair(key[v], v));
            parent[v] = u;
         }
      }}}
```

Strongly Connected Components (Kasuraja's Algo):

```
//topological sort where for each edge a->b,a comes before b
stack<ll> st;
ll sccCnt=0; //for scc
//uncomment if want topological sort too
```

```
   // Try every job one by one
   for (int v = 0; v < N; v++)
   {
      // If applicant u is interested in job v and v is
      // not visited
      if (bpGraph[u][v] && !seen[v])
      {
         seen[v] = true; // Mark v as visited
         // If job 'v' is not assigned to an applicant OR
         // previously assigned applicant for job v (which is matchR[v])
         // has an alternate job available.
         // Since v is marked as visited in the above line, matchR[v]
         // in the following recursive call will not get job 'v' again
         if (matchR[v] < 0 || bpm(bpGraph, matchR[v], seen, matchR))
         {
            matchR[v] = u;
            return true;
         }
      }
   }
   return false;
}

int maxBPM(bool bpGraph[M][N])
{
// The value of matchR[i] is the applicant number
// assigned to job i
   int matchR[N];
   memset(matchR, -1, sizeof(matchR));

   int result = 0; // Count of jobs assigned to applicants
   for (int u = 0; u < M; u++)
   {
```

```
   {
      for(ll j=y;j>=1;j-=j&-j)
         ans+=bit[node][i][j];
   }
   return ans;
}
//top left,bottom right
void updateSubMatrix(ll x1,ll y1,ll x2,ll y2,ll val)
{
   update(0,x1,y1,val);
   update(0,x1,y2+1,-val);
   update(0,x2+1,y1,-val);
   update(0,x2+1,y2+1,val);

   update(1,x1,y1,val*(1-y1));
   update(1,x1,y2+1,val*y2);
   update(1,x2+1,y1,val*(y1-1));
   update(1,x2+1,y2+1,-val*y2);

   update(2,x1,y1,val*(1-x1));
   update(2,x1,y2+1,(x1-1)*val);
   update(2,x2+1,y1,val*x2);
   update(2,x2+1,y2+1,-x2*val);

   update(3,x1,y1,(x1-1)*(y1-1)*val);
   update(3,x1,y2+1,-y2*(x1-1)*val);
   update(3,x2+1,y1,-x2*(y1-1)*val);
   update(3,x2+1,y2+1,x2*y2*val);

}

ll queryPoint(ll x,ll y)
{
   return query(0,x,y)*x*y + query(1,x,y)*x
   + query(2,x,y)*y+query(3,x,y);
}
```

```
//vector<ll> topo;
void dfs(ll curr)
{
   visited[curr]=true;
   for(ll x:adj[curr])
   {
      if(!visited[x])
         dfs(x);
   }
   st.push(curr);
}
void dfs2(ll curr)
{
   //uncomment if want topological sort too
   //topo.push_back(curr);
   visited[curr]=true;
   scc[curr]=sccCnt;
   for(ll x:adjr[curr])
   {
      if(!visited[x])
         dfs2(x);
   }
}
void SCC(ll n)
{
   //uncomment if want topological sort too
   //topo.clear()
   memset(visited,false,sizeof(visited));
   for(ll i=1;i<=n;i++)
   {
      if(!visited[i])
         dfs(i);
   }
   sccCnt=0;
   memset(visited,false,sizeof(visited));
   while(!st.empty())
   {
      ll curr=st.top();
```

```
      // Mark all jobs as not seen for next applicant.
      bool seen[N];
      memset(seen, 0, sizeof(seen));

      // Find if the applicant 'u' can get a job
      if (bpm(bpGraph, u, seen, matchR))
         result++;
   }
   return result;
}
```

Geometry:

1.Area of a regular polygon(equal sides):

$$area = \frac{s^2 n}{4 \tan\left(\frac{180}{n}\right)}$$

2. Angle between (m1, b1) and (m2, b2):

arctan ((m2 − m1) / (m1 · m2 + 1))

3. Triangle: Area = a · b · sin γ / 2

• Area = | x1 · y2 + x2 · y3 + x3 · y1 − y1 · x2 − y2 · x3 − y3 · x1 | / 2

• Heron's formula:

```
ll querySubMatrix(ll x1,ll y1,ll x2,ll y2)
{
   return queryPoint(x2,y2)-queryPoint(x1-1,y2)
   -queryPoint(x2,y1-1)+queryPoint(x1-1,y1-1);
}
```

Convex Hull trick Offline:

```
// convex hull, minimum
struct LineContainer{
vector<ll> M, B;
int ptr;
bool bad(int a,int b,int c) {
  // use deterministic computation with long long
if sufficient
  return (long
double)(B[c]-B[a])*(M[a]-M[b])<(long
double)(B[b]-B[a])*(M[a]-M[c]);
}
// insert with non-increasing m
//m is slope,b is c in eqn y=mx+c
void insert(ll m, ll b) {
  M.push_back(m);
  B.push_back(b);
  while (M.size() >= 3 && bad(M.size()-3,
M.size()-2, M.size()-1)) {
    M.erase(M.end()-2);
    B.erase(B.end()-2);
  }
}
ll get(int i, ll x) {
  return M[i]*x + B[i];
```

```
    st.pop();
    if(!visited[curr])
    {
      sccCnt++;
      dfs2(curr);
    }}}
```

Others:

String Hashing:

```
struct Hashs
{
        vector<int> hashs;
        vector<int> pows;
        int P;
        int MOD;
        Hashs() {}
        Hashs(string &s, int P, int MOD) : P(P), MOD(MOD)
        {
                int n = s.size();
                pows.resize(n+1, 0);
                hashs.resize(n+1, 0);
                pows[0] = 1;
                for(int i=n-1;i>=0;i--)
                {
                        hashs[i]=(1LL * hashs[i+1] * P + s[i] - 'a' + 1) % MOD;
                        pows[n-i]=(1LL * pows[n-i-1] * P) % MOD;
                }
                pows[n] = (1LL * pows[n-1] * P)%MOD;
        }
        int get_hash(int l, int r)
        {
```

Let s = (a + b + c) / 2; then Area = s · (s − a) · (s − b) · (s − c)

4. Circle: (x − xc)^2+ (y − yc)^2= r^2

5.Polygon area (vertex cordinates):

| x1 · y2 + x2 · y3 + ... + xn · y1 − y1 · x2 − y2 · x3 − ... − yn · x1 | / 2

6. Sine Rule: $\frac{a}{\sin A} = \frac{b}{\sin B} = \frac{c}{\sin C} = 2R$
7. Cosine Rule: $a^2 = b^2 + c^2 - 2bc\cos A$
8. Half-Angle Rule: $\sin\frac{A}{2} = \sqrt{\frac{(s-b)(s-c)}{bc}}$;
$\cos\frac{A}{2} = \sqrt{\frac{s(s-a)}{bc}}\ \tan\frac{A}{2} = \sqrt{\frac{(s-b)(s-c)}{s(s-a)}}$
9. Circumradius: $R = \frac{abc}{4(area)}$
10. In-radius:
$r = 4R\sin\frac{A}{2}\sin\frac{B}{2}\sin\frac{C}{2} = (s-a)\tan\frac{A}{2}$
11.Area:
$\Delta = rs = \frac{1}{2}bc\sin A = 2R^2\sin A\sin B\sin C = \frac{abc}{4R} = \sqrt{s(s-a)(s-b)(s-c)}$

12.Medians: $m_a = \frac{1}{2}\sqrt{2b^2 + 2c^2 - a^2}$ and similar expressions for other medians.

13.Area of a quadrilateral ABCD with $AB = a$; $BC = b$; $CD = c$ $DA = d$ $A + C = 2\alpha$ is given by
$\Delta = \sqrt{(s-a)(s-b)(s-c)(s-d) - abcd\cos^2\alpha}$

Its diagonals are given by $AC = \sqrt{\frac{(ac+bd)(ad+bc)}{(ab+cd)}}$;
$BD = \sqrt{\frac{(ac+bd)(ab+cd)}{(ad+bc)}}$

```
}
// query with non-decreasing x
ll query(ll x) {
  ptr=min((int)M.size()-1,ptr);
  while (ptr<M.size()-1 &&
get(ptr+1,x)<get(ptr,x))
    ptr++;
  return get(ptr,x);
}
};
LineContainer obj;

//usage if want maximum,then if y=mx+c
,convert it to equation y'=-mx-c,and then do the
below,only at last res=-obj.query(xi)
//lets say lines (m1,x1),(m2,x2)...,sort these in
decreasing order of m,and then do
obj.insert(mi,xi)
//if query x1,x2,x3..xn,sort increasing order,and
then query->obj.query(xi)
```

Convex Hull Trick Online Set:

```
//Convex hull,maximum
bool Q;
struct Line {
    mutable ll k, m, p;
    bool operator<(const Line& o) const {
        return Q ? p < o.p : k < o.k;
    }
};
struct LineContainer : multiset<Line> {
    ll div(ll a, ll b){
        return a / b - ((a ^ b) < 0 && a % b);
    }
    bool isect(iterator x, iterator y) {
        if (y == end()) { x->p = inf; return false; }
        if (x->k == y->k) x->p = x->m > y->m ? inf
```

```
                int ans=hashs[l] + MOD -
(1LL*hashs[r+1]*pows[r-l+1])%MOD;
                ans%=MOD;
                return ans;
        }
};
//Hashs obj;
//obj=Hashs(s1,P1,mod1));
//obj.get_hash(l,r)
```

FFT:

```
//put this #define compulsarily,change int main()
to int32_t main()
#define int long long
const int N = 8e5+5;
typedef complex<double> base;
const double PI = acos(-1.0l);
const int Maxb = 19;
const int Maxp = 450;
const int MOD=13313;
vector<int> rev;
vector<base> omega;
void calc_rev(int n, int log_n) //Call this before
FFT
{
        omega.assign(n, 0);
         rev.assign(n, 0);

         for(int i=0;i<n;i++)
         {
                rev[i]=0;
                for(int j=0;j<log_n;j++)
                {
                        if((i>>j)&1)
                                rev[i] |=
1<<(log_n-j-1);
                }
```

14. If two chords AB, CD of a circle intersect at a point O (which may lie inside or outside the circle), then $AO.OB = CO.OD$

Orientation:

```
LL orientation(PoLL p1, PoLL p2, PoLL p3)
{
   LL val = (p2.y - p1.y) * (p3.x - p2.x) -
           (p2.x - p1.x) * (p3.y - p2.y);

   if (val == 0) return 0;  // colinear

   return (val > 0)? 1: 2; // clock or counterclock
wise
}
```

Line intersection:

```
bool onSegment(PoLL p, PoLL q, PoLL r)
{
   if (q.x <= max(p.x, r.x) && q.x >= min(p.x, r.x)
&&
       q.y <= max(p.y, r.y) && q.y >= min(p.y, r.y))
      return true;
   return false;
}
bool doIntersect(PoLL p1, PoLL q1, PoLL p2, PoLL
q2)
{
   LL o1 = orientation(p1, q1, p2);
   LL o2 = orientation(p1, q1, q2);
   LL o3 = orientation(p2, q2, p1);
   LL o4 = orientation(p2, q2, q1);
   if (o1 != o2 && o3 != o4)
      return true;
   if (o1 == 0 && onSegment(p1, p2, q1)) return
true;
```

```
: -inf;
      else x->p = div(y->m - x->m, x->k - y->k);
      return x->p >= y->p;
   }
   void add(ll k, ll m) {
      auto z = insert({k, m, 0}), y = z++, x = y;
      while (isect(y, z)) z = erase(z);
      if (x != begin() && isect(--x, y)) isect(x, y =
erase(y));
      while ((y = x) != begin() && (--x)->p >=
y->p)
         isect(x, erase(y));
   }
   ll query(ll x) {
    //   cout<<endl<<jk<<endl;
      assert(!empty());
      Q = 1; auto l = *lower_bound({0,0,x}); Q =
0;
  //    cout<<l.k<<" h "<<l.m<<endl;
      return l.k * x + l.m;
   }
};

LineContainer obj;
//usage if want minimum,then if y=mx+c
,convert it to equation y'=-mx-c,and then do the
below,only at last res=-obj.query(xi)
//adding line y=mx+c->obj.add(m,c)
//querying-> obj.query(x)
```

```
        }
}
void fft(vector<base> &A, int n, bool invert)
{
        for(int i=0;i<n;i++)
        {
                if(i<rev[i])
                        swap(A[i], A[rev[i]]);
        }
        for(int len=2;len<=n;len<<=1)
        {
                double ang=2*PI/len *
(invert?-1:+1);
                int half=(len>>1);
                base curomega(cos(ang), sin(ang));
                omega[0]=base(1, 0);
                for(int i=1;i<half;i++)

omega[i]=omega[i-1]*curomega;
                for(int i=0;i<n;i+=len)
                {
                        base t;
                        int pu = i,
                                pv = i+half,
                                pu_end = i+half,
                                pw = 0;
                        for(; pu!=pu_end; pu++,
pv++, pw++)
                        {
                                t=A[pv] *
omega[pw];
                                A[pv] = A[pu] - t;
                                A[pu] += t;
                        }
                }
        }
        if(invert)
                for(int i=0;i<n;i++)
```

```
   if (o2 == 0 && onSegment(p1, q2, q1)) return
true;
   if (o3 == 0 && onSegment(p2, p1, q2)) return
true;
   if (o4 == 0 && onSegment(p2, q1, q2)) return
true;

   return false;}
```

Circle intersection area:

```
int areaOfIntersection(x0, y0, r0, x1, y1, r1){
var rr0 = r0*r0;
var rr1 = r1*r1;
var c = Math.sqrt((x1-x0)*(x1- x0) +(y1-y0)*(y1-
y0));
var phi =(Math.acos((rr0+(c*c)-rr1) /(2*r0*c)))*2;
var theta =(Math.acos((rr1+(c*c)-rr0) /(2*r1*c)))*2;
var area1 = 0.5*theta*rr1 - 0.5*rr1*Math.sin(theta);
var area2 = 0.5*phi*rr0 - 0.5*rr0*Math.sin(phi);
return area1 + area2;
}
```

Convex Hull:(nlogn)

```
Point nextToTop(stack<Point> &S)
{
   Point p = S.top();
   S.pop();
   Point res = S.top();
   S.push(p);
   return res;
}
int distSq(Point p1, Point p2)
{
   return (p1.x - p2.x)*(p1.x - p2.x) +
         (p1.y - p2.y)*(p1.y - p2.y);
}
int compare(const void *vp1, const void *vp2)
```

```
                A[i]/=n;
}
void multiply(int n, vector<base> &A,
vector<base> &B, vector<int> &C)
{
        fft(A, n, false);
        fft(B, n, false);
        for(int i=0;i<n;i++)
                A[i] *= B[i];
        fft(A, n, true);
        for(int i=0;i<n;i++)
        {
                C[i] = (int)(A[i].real() + 0.5);
                C[i] %= MOD;
        }
}
void Solve(int n, vector<int> &coeffA,
vector<int> &coeffB, vector<int> &result, bool
big1, bool big2) //Call 4 times: 00, 01, 10, 11
{
        vector<base> A(n), B(n);
        for(int i=0;i<n;i++)
        {
                A[i]=big1?coeffA[i]/Maxp :
coeffA[i]%Maxp;
                B[i]=0;
        }
        for(int i=0;i<n;i++)
        {
                B[i]=big2?coeffB[i]/Maxp :
coeffB[i]%Maxp;
        }
        vector<int> C(n);
        multiply(n, A, B, C);
        for(int i=0;i<n;i++)
        {
                int add=C[i];
                if(big1)
```

```
{
  Point *p1 = (Point *)vp1;
  Point *p2 = (Point *)vp2;
  int o = orientation(p0, *p1, *p2);
  if (o == 0)
    return (distSq(p0, *p2) >= distSq(p0, *p1))? -1 :
1;
  return (o == 2)? -1: 1;
}
void convexHull(Point points[], int n)
{
  int ymin = points[0].y, min = 0;
  for (int i = 1; i < n; i++)
  {
    int y = points[i].y;
    if ((y < ymin) || (ymin == y &&
        points[i].x < points[min].x))
      ymin = points[i].y, min = i;
  }
  swap(points[0], points[min]);
  p0 = points[0];
  qsort(&points[1], n-1, sizeof(Point), compare);
  int m = 1;
  for (int i=1; i<n; i++)
  {
     // Keep removing i while angle of i and i+1 is
same
     while (i < n-1 && orientation(p0, points[i],
                                   points[i+1]) == 0)
       i++;
     points[m] = points[i];
     m++;
  }
  if (m < 3) return;
  stack<Point> S;
  S.push(points[0]);
  S.push(points[1]);
  S.push(points[2]);
```

```
                        add*=Maxp;
                if(big2)
                        add*=Maxp;
                add%=MOD;
                result[i]+=add;
                result[i]%=MOD;
        }
}
void do_FFT(vector<int> &A, vector<int> &B,
vector<int> &result)
{
        int n=1, bits=0;
        while(n<2*A.size() || n<2*B.size())
                n<<=1, bits++;
        result.assign(n, 0);
        calc_rev(n, bits);
        vector<int> tempA(A.begin(), A.end());
        vector<int> tempB(B.begin(), B.end());
        tempA.resize(n);
        tempB.resize(n);
        for(int i=0;i<2;i++)
        {
                for(int j=0;j<2;j++)
                {
                        Solve(n, tempA, tempB,
result, i, j);
                }
        }
}
//vector<int> v1;
//v1.resize(n+1);
// fill(v1.begin(),v1.end(),0);
//if x^2+3*x then v1[2]=1,v1[1]=3
//vector<int> res;
//do_FFT(v1,v2,res);
//res.resize(2*n+1) //so agar x size ka tha ,then x to
2*n tak sab 0
```

```
  for (int i = 3; i < m; i++)
  {
    while (orientation(nextToTop(S), S.top(),
points[i]) != 2)
      S.pop();
    S.push(points[i]);
  }
  while (!S.empty())
  {
    Point p = S.top();
    cout << "(" << p.x << ", " << p.y <<")" << endl;
    S.pop();
  }
}
```

Point in a polygon:

```
bool isInside(Point polygon[], int n, Point p)
{
  if (n < 3)  return false;
  Point extreme = {INF, p.y};
  int count = 0, i = 0;
  do
  {
    int next = (i+1)%n;
    if (doIntersect(polygon[i], polygon[next], p,
extreme))
    {
      if (orientation(polygon[i], p, polygon[next])
== 0)
        return onSegment(polygon[i], p,
polygon[next]);

      count++;
    }
    i = next;
  } while (i != 0);
```

	return count&1; // Same as (count%2 == 1)
	}	