Jagannath University

TheOneYouDontWant

Ashraful Haque, Neaj Morshad, Muhammad Selim

Contents

Dynamic Programming (1)

DigitDPAllDigitSum.cpp

23 lines

```
ll dp[15][2][400][2];
const ll mpos=11; char ch[40];
void convert(ll n){
 for(ll i=0; i<mpos; i++){
  ch[i]=(n%10)+'0'; n/=10;
 }
 reverse(ch,ch+mpos); ch[mpos]=0;
}
ll func(ll pos,ll Smlornot, ll dcnt,ll Strt){
 if(pos==mpos) return dcnt;
 ll &val=dp[pos][Smlornot][dcnt][Strt];
 if(val!=-1) return val;
 ll be=0, en=9,re=0;
 if(!Smlornot) en=ch[pos]-'0';
 for(ll i=be; i<=en; i++){
  ll iSml= Smlornot | (i<en);
  ll idigitvalcnt=dcnt+ i;
  ll isStrt= Strt | (i!=0);
  re+=func(pos+1,iSml,idigitvalcnt,isStrt);
 }
 return val=re;
}
func(0,0,0,0);
```

SOSDP.cpp

18 lines

```
memset(dp,-1,sizeof(dp));
for(int i=1;i<=n;i++)dp[ar[i]]=ar[i];
for(int i=0;i<22;i++){
 for(int mask=0;mask<(1<<22);mask++){
  if(chk(mask,i))
   dp[mask]=max(dp[mask],dp[mask^(1<<i)]);
 }
}
int boro=(1<<22)-1;
//iterate all the submask of a mask
for(int mask=1;mask<(1<<sz);mask++) {
 int tmask=mask&(mask-1);
 while(tmask) {
  cout<<tmask<<endl;
//dp[mask]=min(dp[mask],dp[tmask]+dp[mask^tmask]);
  tmask=(tmask-1)&mask;
 }
}
```

1.1 Optimization

CHT(Dynamic).cpp

62 lines

```
#define ll long long int
#define ld long double
ll inf = 9e18 + 5;
// for min query Add(-m,-b) also get the result in -ans
struct HullDynamic{ // Max Query
  struct line {
        ll m, b; ld x;
        ll val; bool isQuery;
        line(ll _m = 0, ll _b = 0) :
            m(_m), b(_b), val(0), x(-inf), isQuery(
                false) {}
        ll eval(ll x) const { return m * x + b; }
        bool parallel(const line &l) const { return m
            == l.m; }
        ld intersect(const line &l) const {
            return parallel(l) ? inf : 1.0 * (l.b - b)
                / (m - l.m);
        }
        bool operator < (const line &l) const {
            if(l.isQuery) return x < l.val;
            else return m < l.m;
        }
    };
    set<line> hull;
    typedef set<line> :: iterator iter;
    bool cPrev(iter it) { return it != hull.begin(); }
    bool cNext(iter it) { return it != hull.end() &&
        next(it) != hull.end(); }
    bool bad(const line &l1, const line &l2, const line
         &l3) {
        return l1.intersect(l3) <= l1.intersect(l2);
    }
    bool bad(iter it) {
        return cPrev(it) && cNext(it) && bad(*prev(it),
            *it, *next(it));
    }
    iter update(iter it) {
        if(!cPrev(it)) return it;
        ld x = it -> intersect(*prev(it));
        line tmp(*it); tmp.x = x;
        it = hull.erase(it);
        return hull.insert(it, tmp);
    }
    void Add(ll m, ll b) {
        line l(m, b);
        iter it = hull.lower_bound(l);
        if(it != hull.end() && l.parallel(*it)) {
            if(it -> b < b) it = hull.erase(it);
        else return;
        }

        it = hull.insert(it, l);
        if(bad(it)) return (void) hull.erase(it);

        while(cPrev(it) && bad(prev(it))) hull.erase(
            prev(it));
        while(cNext(it) && bad(next(it))) hull.erase(
            next(it));

        it = update(it);
        if(cPrev(it)) update(prev(it));
        if(cNext(it)) update(next(it));
    }
    ll Query(ll x) const {
        if(hull.empty()) return -inf;
        line q; q.val = x, q.isQuery = 1;
        iter it = --hull.lower_bound(q);
        return it -> eval(x);
    }
};
```

CHT(offline).cpp

41 lines

```
#define ll long long
#define INF 1000000000000000000
struct Line{
    ll m,c;
    Line(ll x,ll y){m=x; c=y;}
    ll Get(ll x) {return m*x+c;}
    bool operator<(const Line &other) const {return m <
         other.m;}
};
bool Bad(Line &P,Line &C,Line &N){
    return (P.c-C.c) * 1.0L * (N.m-P.m) > (P.c-N.c) *
        1.0L * (C.m - P.m);
}
//This Convex Hull always maintains lower convex hull
//m1 >= m2 >= m3 .... >= mk
//For Min Query : Add(m,c)
//For Max Query : Add(-m,-c)
struct ConvexHull{
    vector<Line>hull;
    void Add(ll m,ll c){
        //always maintaining the minimum c in case
            multiple equal m
        if(hull.size()>0 && hull.back().m==m) {
```

```
            if(hull.back().c>c) hull.pop_back();
            else return;
        }
        hull.push_back(Line(m, c));
        int sz=hull.size();
        while(sz>2 && Bad(hull[sz-3],hull[sz-2],hull[sz
            -1])){
            swap(hull[sz-2],hull[sz-1]); hull.pop_back
                (); sz--;
        }
    }
    ll Query(ll x){
        int lo=-1;
        int hi=hull.size()-1;
        while(hi-lo>1) {
            int mid=(lo+hi)/2;
            if(hull[mid].Get(x)>=hull[mid+1].Get(x)) lo
                =mid;
            else hi=mid;
        }
        if(hi<0 || hi>=hull.size()) return INF;
        return hull[hi].Get(x);
    }
};
```

DandCOptimization.cpp

22 lines

```
/*Complexity : O(n log n)
dp[i][j]=min(dp[i-1][k-1]+Cost(k,j)[k<=j]
Condition for D&C:
Cost(L+1,j+1)-Cost(L+1,j)<=Cost(k+1,j+1)- Cost(k+1,j)
    for any(L<k<j)For Max Query
Cost(L+1,j+1)-Cost(L+1,j)>=Cost(k+1,j+1)- Cost(k+1,j)
    for any(L<k<j)For Min Query*/
ll dp[2][MAX];
void compute(int K,int L,int R,int OptL,
 int OptR){ if(L > R) return;
 int mid = (L + R)/2,optNow = -1;
 dp[K & 1][mid] = 0;
 for(int i=OptL;i<=min(OptR,mid);i++){
  ll tmp =dp[(K & 1)^1][i-1]+Cost(i,mid);
  if(tmp >= dp[K & 1][mid]){
    dp[K & 1][mid] = tmp;optNow = i;
  }
 }
 compute(K, L, mid - 1, OptL, optNow);
 compute(K, mid + 1, R, optNow, OptR);
}
for(int i=1;i<=n;i++)dp[1][i]=Cost(1,i);
for(int i=2;i<=K;i++)compute(i,1,n,1,n);
printf("%lld\n", dp[K & 1][n]);
```

KnuthOptimization.cpp

19 lines

```
//Complexity : O(n^2) for any k <= n
const ll INVALID = LLONG_MIN;
ll C[MAX][MAX],dp[MAX][MAX],Opt[MAX][MAX];
/*Recurrence : dp[i][j]=min/max i<=k<=j (dp[i-1][k-1]+C
    [k][j])*/
/*Condition: Opt[i-1][j]<=Opt[i][j] <=Opt[i][j+1]*/
for(int i=0;i<=K;i++) dp[i][0]=0;
for(int i=0;i<=K;i++)
 for(int j=1;j<=N;j++)dp[i][j]=INVALID;
for(int i=1;i<=N;i++){
 Opt[0][i]=1; Opt[i][N+1]=N;}
for(int i=1;i<=K;i++){
 for(int j=N;j>=1;j--){
  for(int k=Opt[i-1][j];k<=Opt[i][j+1];k++){
   if(dp[i-1][k-1]== INVALID) continue;
   if(dp[i][j]<dp[i-1][k-1]+C[k][j]){
     dp[i][j]=dp[i-1][k-1]+C[k][j];
     Opt[i][j]=k;
   }}}
} printf("%lld\n",dp[K][N]);
```

Data structures (2)

BIT.cpp

29 lines

```
ll BIT[2][MAXN];
void update(int cs, int indx, ll val){
 while(indx < MAXN){
  BIT[cs][indx]+=val;indx+=(indx&-indx);}
}
ll sum(int cs, int indx){
 ll ans = 0;
 while(indx != 0) {
  ans+=BIT[cs][indx];indx-=(indx&-indx);}
 return ans;
}
void updateRange(int l, int r, ll val){
  update(0,l,val); update(0,r+1,-val);
  update(1,l,val*(l-1)); update(1,r+1,-val*r);
}
ll sumRange(int indx)
 {return sum(0,indx)*indx - sum(1,indx);}
ll QueryRange(int l, int r)
 {return sumRange(r)-sumRange(l-1);}
const int LOGN = 20;
int LowerBound(int cs, ll v){
 ll sum = 0; int indx = 0;
 for(int i = LOGN; i >= 0; i--){
  int nPos = indx + (1<<i);
  if(nPos < MAXN && sum + BIT[cs][nPos] < v){
   sum += BIT[cs][nPos]; indx = nPos;}
 }//pos = maximal x such that Sum(x) < v
 return indx + 1; //+1 for LowerBound
}
```

HLD.cpp

70 lines

```
vector<pair<int,int>>g[mx];
int par[mx],sub_sz[mx],T,Rin[mx];;
int Head[mx],st[mx],sesh[mx];
/*In SegTree init Tree[bode]=ar[Rin[be]]*/
using namespace Segment_Tree;
void sz_dfs(int u,int p){
 sub_sz[u]=1; par[u]=p;
 for(auto &v: g[u]){
  if(v.first==p)continue;
  sz_dfs(v.first,u);
  sub_sz[u]+=sub_sz[v.first];
  if(sub_sz[v]>sub_sz[g[u][0].first]) swap(v,g[u][0]);
 }
}
void hld_dfs(int u,int p,int cost){
 st[u]=++T;Rin[st[u]]=u;
 ar[st[u]]=cost;/*not for node value*/
 for(auto v:g[u]){
  if(v.first==p)continue;
  Head[v.first] = (v.first==g[u][0].first ? Head[u]:v.
      first);
  hld_dfs(v,u,v.second);
 }
 sesh[u]=T;
}
void hld_build(int root){
 T=0;Head[root]=root;
 sz_dfs(root,root); hld_dfs(root,root,0);
}
bool Is_it_parent(int p,int u){
 return st[p]<=st[u] && sesh[u]<=sesh[p];
}
int path_query(int u,int v){
 int re=-inf;
 while(1){
  if(Is_it_parent(Head[u],v))break;
  re=max(re,query(1,1,n,st[Head[u]],st[u]));
  /*for sum just add in all query*/
  u=par[Head[u]];
 }
 swap(u,v);
 while(1){
  if(Is_it_parent(Head[u],v))break;
  re=max(re,query(1,1,n,st[Head[u]],st[u]));
  u=par[Head[u]];
 }
 if(st[u]>st[v])swap(u,v);
 re=max(re,query(1,1,n,st[u]+1,st[v]));
 /* node hole st[u] theke start*/
 return re;
}
void path_update(int u,int v,int val){
 while(1){
  if(Is_it_parent(Head[u],v))break;
  Rupdate(1,1,n,st[Head[u]],st[u],val);
  u=par[Head[u]];
 }
 swap(u,v);
 while(1){
  if(Is_it_parent(Head[u],v))break;
  Rupdate(1,1,n,st[Head[u]],st[u],val);
  u=par[Head[u]];
 }
```

```
 if(st[u]>st[v])swap(u,v);
 Rupdate(1,1,n,st[u]+1,st[v],val);
 /*node hole st[u] theke start*/
}
void update_subtree(int u,int val){
 Rupdate(1,1,n,st[u]+1,sesh[u],val);
 /*node hole st[u] theke start*/
}
```

2.1 Sparse Table

SparseTable.cpp

14 lines

```
int ST[mx][MAX_logN], Jump_LOG[mx];
void Build_Sparse(){
 for(int i=1;i<=n;i++)ST[i][0]=ar[i];
 for(int j=1;(1<<j)<=n;j++){
  for(int i=1;(i+(1<<j)-1)<=n;i++){
  ST[i][j]=min(ST[i][j-1], ST[i+(1<<(j-1))][j-1]);
  }}
}
int query(int i,int j){
 int boro_lav=Jump_LOG[j-i+1];
 return min(ST[i][boro_lav], ST[j-(1<<boro_lav)+1][
     boro_lav]);
}
for(int i=2;i<=n;i++)
 Jump_LOG[i]=Jump_LOG[i-1]+!(i&(i-1));
```

RectangleQuery2D.cpp

30 lines

```
int ST[mx][mx][MAX_logN][MAX_logN];
int Jump_LOG[505];
void Build_2D_Sparse(){
 for(int i=1;i<=n;i++){
  for(int j=1;j<=m;j++)
   ST[i][j][0][0]=ar[i][j];
  for(int l=1;(1<<l)<=m;l++){
   int pre=1<<(l-1);
   for(int j=1;j+pre<=m;j++){
    ST[i][j][0][l]=max(ST[i][j][0][l-1], ST[i][j+pre
        ][0][l-1]);
   }}
 }
 for(int l=1;(1<<l)<=n;l++){
  int pre=1<<(l-1);
  for(int i=1;i+pre<=n;i++){
   for(int k=0;(1<<k)<=m;k++){
    for(int j=1;j<=m;j++){
     ST[i][j][l][k]=max(ST[i][j][l-1][k], ST[i+pre][j][
         l-1][k]);
    }}}}
}
int query(int i,int j,int p,int q){
 int boro_jum1=Jump_LOG[p-i+1], re1, re2;
 int boro_jum2=Jump_LOG[q-j+1];
 int pre1=1<<boro_jum1,pre2=1<<boro_jum2;
 re1=max(ST[i][j][boro_jum1][boro_jum2],
     ST[i][q-pre2+1][boro_jum1][boro_jum2]);
 re2=max(ST[p-pre1+1][j][boro_jum1][boro_jum2],
  ST[p-pre1+1][q-pre2+1][boro_jum1][boro_jum2]);
 return max(re1,re2);
}
```

SquareQuery2D.cpp

18 lines

```
int ar[mx][mx],ST[mx][mx][LOG],Jump_LOG[mx];
void Build_sparse_square(int N){
 for(int l=0;(1<<l)<=N;l++){
  for(int i=1;i+(1<<l)<N;i++){
   for(int j=1;j+(1<<l)<N;j++){
    if(l==0)ST[i][j][l]=dp[i][j];
    else{
     int val1=max(ST[i][j][l-1], ST[i+(1<<(l-1))][j][l
         -1]);
     int val2=max(ST[i][j+(1<<(l-1))][l-1], ST[i+(1<<(l
         -1))][j+(1<<(l-1))][l-1]);
     ST[i][j][l]=max(val1,val2);
    }}}}
}
int query(int i,int j,int l){
 int lg=Jump_LOG[l],add=1<<lg,re1,re2;
 re1=max(ST[i][j][lg],ST[i+l-add][j][lg]);
 re2=max(ST[i][j+l-add][lg], ST[i+l-add][j+l-add][lg]);
 return max(re1,re2);
}
```

2.2 Segment Tree

SegTreePersistent.cpp

59 lines

```
namespace Persistent{
    struct node{
        node *left,*right; int val;
        node(){val=0;left=NULL;right=NULL;}
        node(int a,node *b,node *c) {val=a,left=b,right
            =c;}
        node* update(int lo,int hi,int i,int v){
            node* ret = new node(val,left,right);
            if(lo==hi) {ret->val += v;return ret;}
            int mid=(lo+hi)/2;
            if(i<=mid){
                if(!left) left = new node();
                ret->left=left->update(lo,mid,i,v);
            }
            else{
                if(!right) right = new node();
                ret->right = right->update(mid+1,hi,i,v
                    );
            }
            ret->val=0;
            if(ret->left)ret->val+=ret->left->val;
            if(ret->right)ret->val+=ret->right->val;
            return ret;
        }
        int query(int lo,int hi,int i,int j) {
            if(hi<i || lo>j) return 0;
            if(i<=lo && hi<=j) return val;
            int mid=(lo+hi)/2;
            int ret = 0;
            if(left)ret+=left->query(lo,mid,i,j);
            if(right)ret+=right->query(mid+1,hi,i,j);
            return ret;
        }
    };node* root[mx];
    inline int Val(node* x){return x?x->val:0;}
    inline node* Left(node* x){return x?x->left:NULL;}
    inline node* Right(node* x){return x?x->right:NULL;
        }

    //Searching kth minimum element in sorted order
    int Search(node *a,node *b,node* c, node* d,int l,
        int r,int k){
      if(l==r) return l;
      int Count=Val(Left(a))+Val(Left(b))- Val(Left(c))
          -Val(Left(d));
      int mid=(l+r)/2;
      if(Count>=k) return Search(Left(a),Left(b), Left(
          c),Left(d),l,mid,k);
      return Search(Right(a),Right(b),Right(c), Right(d
          ),mid+1,r,k-Count);
    }
}
using namespace Persistent;
void dfs(int u,int p,int d){
    ......
    root[u]=root[p]->update(1,Size,w[u],1);
    ......
}
int main(){
    ......
    dfs(1,0,0);
    Size=Map.size();
    //number of distinct elements after compression
    root[0]=new node();
    ......
}
```

VariousSegTree.cpp

58 lines

```
/*Bracket Sequence */
struct info{
 int open,close,ans;
};
info Merge(info a,info b){
 info re;
 int valid=min(a.open,b.close);
 re.open=a.open+b.open-valid;
 re.close=a.close+b.close-valid;
 re.ans=a.ans+b.ans+valid;
 /* works for maximum length of correct bracket
     sequence in l to r range*/
 return re;
}
/* Kth element merge sort tree */
int query(int node,int be,int en,int l,int r,int k){
```

```
 if(be==en)return seg[node][0];
 int pos = upper_bound(seg[node*2+1].begin(), seg[node
     *2+1].end(),r)
 -lower_bound(seg[node*2+1].begin(), seg[node*2+1].end
     (),l);
 int mid=(be+en)/2;
 if(pos>=k) {
  return query(node*2+1,be,mid,l,r,k);
 }
 else return query(node*2+2,mid+1,en,l,r,k-pos);
}
/* Delete Type Id Found */
int id_query(int node,int be,int en,int pos){
 if(be==en)return be;
 int mid=(be+en)/2;
 if(Present[node*2]>=pos){
  return id_query(node*2,be,mid,pos);
 }
 else return id_query(node*2+1,mid+1,en, posPresent[
     node*2]);
}
/* Range max subarray / suffix-prefix sum*/
struct info{
 ll max_pref,max_suf,ans,sum;
 void Merge(info p1,info p2){
  sum=p1.sum+p2.sum;
  max_pref=max(p1.max_pref,p1.sum+p2.max_pref);
  max_suf=max(p2.max_suf,p2.sum+p1.max_suf);
  ans=max(max(p1.ans,p2.ans), p1.max_suf+p2.max_pref);
 }
};
void Relax(int node,int be,int en){
 if(!cur[node])return;
 Tree[node].sum=Lazy[node]*(en-be+1);
 Tree[node].max_pref=max(0LL,Tree[node].sum);
 Tree[node].max_suf=max(0LL,Tree[node].sum);
 Tree[node].ans=max(0LL,Tree[node].sum);
 if(be!=en){
  Lazy[node*2]=Lazy[node];
  Lazy[node*2+1]=Lazy[node];
  cur[node*2]=true;
  cur[node*2+1]=true;
 }
 cur[node]=false;
 Lazy[node]=0;
}
```

2.3 SQRT Decomposition

MOonTree.cpp

57 lines

```
/* Rest of the part include from MO's part*/
namespace MO {
 int l,r,id,lca; node(){}
 node(int l,int r,int lca,int id){
 this->l=l;this->r=r;this->lca=lca;
 this->id=id; }
 vector<int> g[N];
 int Euler[2*N],st[N],en[N],Time;
 int depth[mx],par[mx][25];
 void dfs(int u,int p,int lvl){
  st[u]=++Time; Euler[Time]=u;
  par[u][0]=p; depth[u]=lvl;
  for(int v:g[u]){
    if(v==p)continue;
    dfs(v,u,lvl+1);
  }
  en[u]=++Time; Euler[Time]=u;
 }
 /*Subtree niye kaj korle
  vector<int> g[N];
  int Euler[N],st[N],en[N],Time;
  void dfs(int u,int p){
  st[u]=++Time;Euler[Time]=u;
  for(int v:g[u]){
   if(v==p)continue;
    dfs(v,u);
   }
  en[u]=Time;
  }*/
}
using namespace MO;
/* init_LCA */
LOG=log2(n)+1; Time=0;
for(int i=1;i<n;i++){
 scanf("%d%d",&x,&y);
 g[x].push_back(y);
 g[y].push_back(x);
}
init(root);
for(int i=1;i<=q;i++){
 scanf("%d%d",&x,&y);
 if(st[x]>st[y])swap(x,y); int p=lca(x,y);
 if(x==p)query[i]=node(st[x],st[y],-1,i);
 else query[i]=node(en[x],st[y],p,i);
}
sort(query+1,query+1+q);
int left=query[1].l,right=left-1;
for(int i=1;i<=q;i++){
  node Now=query[i];
  while(left<Now.l)check(Euler[left++]);
  while(left>Now.l)check(Euler[--left]);
  while(right<Now.r)check(Euler[++right]);
  while(right>Now.r)check(Euler[right--]);
  if(Now.lca!=-1)check(Now.lca);
  ans[Now.id]=re;
  if(Now.lca!=-1)check(Now.lca);
}
```

MOs.cpp

39 lines

```
namespace MO{
 const int N=100005;const int Q=100005;
 int BlockId[N],ans[Q]; bool vis[N];
 struct node{
  int l,r,id; node(){}
  node(int l,int r,int id){
   this->l=l;this->r=r;this->id=id;
  }
  bool operator < (const node& u){
   int a=BlockId[l],b=BlockId[u.l];
   if(a==b)return (a&1?(r > u.r):(r < u.r));
   else return a<b;
  }
 }query[Q];
 void check(int pos){
  if(vis[pos]){}
  else{}
  vis[pos]^=1;
 }
}
using namespace MO;
int sz=sqrt(n);
for(int i=1;i<=n;i++){
 BlockId[i]=i/sz;vis[i]=false;
}
for(int i=1;i<=q;i++){
 int x,y;scanf("%d%d",&x,&y);
 query[i]=node(x,y,i);
}
sort(query+1,query+q+1);
int left=query[1].l,right=left-1;
for(int i=1;i<=q;i++){
 node Now=query[i];
 while(left<Now.l)check(left++);
 while(left>Now.l)check(--left);
 while(right<Now.r)check(++right);
 while(right>Now.r)check(right--);
 ans[Now.id]=boro;
}
```

NumberOfInversionInARange.cpp

28 lines

```
// MO's template
// For segment Tree update
if(be==en)Tree[node]+=val;
Tree[node]=Tree[node*2]+Tree[node*2+1];
using namespace Segment_Tree;
/* at first compress the value of arrat=y*/
int left=que[1].l;
int right=left-1;
for(int i=1;i<=q;i++){
 node Now=que[i];
 while(left<Now.l){
  re-=query(1,1,n,1,ar[left]-1);
  update(1,1,n,ar[left++],-1);
 }
 while(left>Now.l){
  re+=query(1,1,n,1,ar[--left]-1);
  update(1,1,n,ar[left],1);
 }
 while(right<Now.r){
  re+=query(1,1,n,ar[++right]+1,n);
  update(1,1,n,ar[right],1);
```

```
 }
 while(right>Now.r){
  re-=query(1,1,n,ar[right]+1,n);
  update(1,1,n,ar[right--],-1);
 }
 ans[Now.id]=re;
}
```

2.4 Trie

Persistent Trie.cpp

66 lines

```
/* find maximum value (x^a[j]) in
the range (l,r) where l<=j<=r*/
const int N = 2e5 + 05;
const int K = 30;
struct node_t {
  int time; node_t* to[2];
  node_t() : time(0) {
    to[0] = to[1] = 0;
  }
  bool go(int l) const {
    if (!this) return false;
    return time >= l;
  }
};
typedef node_t* pnode;
pnode clone(pnode p) {
  pnode cur = new node_t();
  if (p) {
    cur->time = p -> time;
    cur->to[0] = p -> to[0];
    cur->to[1] = p -> to[1];
  }
  return cur;
}
pnode last,version[N];
void insert(int a, int time){
  pnode v = clone(last);
  version[time] = last = v;
  for (int i = K-1;i>=0;--i){
    int bit = (a >> i) & 1;
    pnode &child = v->to[bit];
    child = clone(child);
    v = child;
    v->time = time;
  }
}
int query(pnode v,int x,int l){
  int ans = 0;
  for (int i = K-1;i>=0;--i){
    int bit = (x >> i) & 1;
    if (v->to[bit]->go(l)) {
      ans |= 1 << i;
      v = v->to[bit];
    } else {
      v = v->to[bit ^ 1];
    }
  }
  return ans;
}
void solve() {
  int n, q,a,x,l,r,ans;
  scanf("%d %d", &n, &q);
  last = 0;
  for (int i = 0; i < n; ++i) {
    scanf("%d", &a);
    insert(a, i);
  }
  while (q--) {
    scanf("%d%d%d",&x,&l,&r);
    --l,--r;
    ans=query(version[r],~x,l);
    printf("%d\n",ans);
/*Trie version[r] contains the
trie for [0...r] elements*/
  }
}
```

Trie.cpp

34 lines

```
/* Max xor and Min xor subarray */
int Trie[mx*30][2],End[mx*30],ar[mx],st=1;
void Insert(int val){
 int cur=1;
 for(int i=29;i>=0;i--){
  int bit=0;
  if(((1<<i) & val))bit=1;
  if(Trie[cur][bit]==0)Trie[cur][bit]=++st;
  cur=Trie[cur][bit];
 }
 End[cur]=val;
}
// for max query just go to opposite bit
int query_min(int val){
 int cur=1;
 for(int i=29;i>=0;i--){
  int bit=0;
  if(((1<<i) & val))bit=1;
  if(Trie[cur][bit])cur=Trie[cur][bit];
  else if(Trie[cur][bit^1])cur=Trie[cur][bit^1];
 }
 return End[cur]^val;
}
void solve(){
 st=1; memset(End,0,sizeof(End));
 int re; memset(Trie,0,sizeof(Trie));
 re_min=INT_MAX,re_max=0,suffix=0;Insert(0);
 for(int i=1;i<=n;i++){
  suffix^=ar[i];
  re_min=min(re_min,query_min(suffix));
  re_max=max(re_max,query_max(suffix));
  Insert(suffix);
 }
}
```

Numerical (3)

3.1 Matrices

GaussianEliminationOffline.cpp

17 lines

```
ll a[MAX], n; //0 base index
ll maxxor(){
 int r = 0; ll ret = 0;
 for(int c = 63; c >= 0; c--){
  int idx = -1;
  for(int i = r; i < n && idx < 0; i++)
   if(a[i] >> c & 1) idx = i;
  if(idx == -1) continue;
  swap(a[r], a[idx]);
  for(int i = 0; i < n; i++) if(i != r)
   if(a[i] >> c & 1) a[i] ^= a[r];
  r++;
 }
 for(int i = 0; i < n; i++)
  ret = max(ret, ret ^ a[i]);
 return ret;
}
```

GaussuanElimantionOnline.cpp

33 lines

```
// Gaussian Elimination Online
struct Max_xor{
    vector<ll> basis;void init() {basis.clear();}
    void add(ll x){
        // Keep the basis sorted in increasing order
        for(ll b : basis) x = min(x, x ^ b);
        for(ll &b : basis) b = min(b, x ^ b);
        if(x){
            basis.push_back(x);
            for(ll i = basis.size()-1;i>0;i--){
                if(basis[i] < basis[i - 1]) swap(basis[
                    i], basis[i - 1]);
                else break;
            }
        }
    }
    //returns max subset xor
    ll getMax(){
        ll ans=0;for(ll b:basis)ans ^= b;
        return ans;
    }
    //returns max xor over (k ^ some subset)
    ll getMax(ll k){
        ll ans = k;for(ll b:basis)ans=max(ans,ans^b);
        return ans;
    }
 //returns k-th (0-idx) smallest distinct subset xor
    ll getKth(ll k){
        ll ans = 0;
        for(ll i = 0; i < basis.size(); i++)
            if((k >> i) & 1) ans ^= basis[i];
        return ans;
```

```
    }
};
```

MatrixExponentiation.cpp

99 lines

```
#define MAX 105 #define ll long long int
ll MOD = 1e9 + 7;
ll MOD2 =MOD*MOD*2;//carefull about overflow
inline ll inv(ll n) {return bigMod(n,MOD-2);}
inline ll Mul(ll a,ll b){return (a*b)%MOD;}
inline ll Div(ll a,ll b){return Mul(a,inv(b));}
/* 1 base row columun index */
struct Matrix{
 int row, col;
 ll m[MAX][MAX];
 Matrix() {memset(m,0,sizeof(m));}
 void Set(int r,int c) {row = r; col = c;}
 Matrix(int r,int c)
 {memset(m,0,sizeof(m)); Set(r,c);}
 void normalize(){
  for(int i=1; i<=row; i++){
   for(int j=1; j<=col; j++){
    m[i][j] %= MOD;
    if(m[i][j] < 0) m[i][j] += MOD;
   }}}
};
Matrix Multiply(Matrix A,Matrix B){
 Matrix ans(A.row,B.col);
 for(int i=1;i<=A.row;i++){
  for(int j=1;j<=B.col;j++){
   ans.m[i][j]=0;
   ll sm = 0;
   for(int k=1;k<=A.col;k++){
    sm+=(A.m[i][k]*B.m[k][j]);
    if(sm >= MOD2) sm -= MOD2;
   }
   ans.m[i][j] = sm % MOD;
  } }
 return ans;
}
Matrix Power(Matrix mat,ll p){
 Matrix res(mat.row , mat.col);
 Matrix ans(mat.row , mat.col);
 int n = ans.row;
 for(int i=1;i<=n;i++){
  for(int j=1;j<=n;j++){
   ans.m[i][j]=0;
   res.m[i][j]=mat.m[i][j];
  }
  ans.m[i][i]=1;
 }
 while(p){
  if(p&1) ans=Multiply(ans,res);
  res=Multiply(res,res);
  p=p/2;
 }
 return ans;
}
```

```
ll Det(Matrix mat){
 assert(mat.row == mat.col);int n = mat.row;
 mat.normalize(); ll ret = 1;
 for(int i = 1; i <= n; i++){
  for(int j = i + 1; j <= n; j++){
   while(mat.m[j][i]){
    ll t = Div(mat.m[i][i], mat.m[j][i]);
    for(int k = i; k <= n; ++k){
     mat.m[i][k] -= Mul(mat.m[j][k] , t);
     if(mat.m[i][k] < 0) mat.m[i][k] += MOD;
     swap(mat.m[j][k], mat.m[i][k]);
    }
    ret = MOD - ret;
   }
  }
  if(mat.m[i][i] == 0) return 0;
  ret =Mul(ret, mat.m[i][i]);
 }
 if(ret < 0) ret += MOD;return ret;
}
ll Tmp[MAX<<1][MAX<<1];
Matrix Inverse(Matrix mat){
 assert(mat.row==mat.col);assert(Det(mat)!=0);
 int n = mat.row; mat.normalize();
 for(int i=1;i<=n;i++){
  for(int j=1;j<=n;j++) Tmp[i][j]=mat.m[i][j];
  for(int j=n+1; j<=2*n; j++) Tmp[i][j] = 0;
  Tmp[i][i+n] = 1;
 }
 for(int i=1; i<=n; i++){
  assert(Tmp[i][i] != 0);
  for(int j=1; j<=n; j++){
   if(i == j) continue;
   ll c = Div(Tmp[j][i], Tmp[i][i]);
   for(int k=i; k<=2*n; k++){
    Tmp[j][k] = Tmp[j][k]-Mul(Tmp[i][k], c);
    if(Tmp[j][k] < 0) Tmp[j][k] += MOD;
   }}
 }
 Matrix Inv(n,n);
 for(int i=1; i<=n; i++){
  for(int j = 1; j <= n; j++){
   Inv.m[i][j] = Div(Tmp[i][j+n],Tmp[i][i]);
  }
 }return Inv;
}
```

GaussianElimination.cpp

39 lines

```
//Gaussian Elimination
//format : (a[0]*x[0]+a[1]*x[1] ... a[m-1]*x[m-1]) % k
    = a[m], where 0 <= ai < k
//number of solution : k^(number of free variable) = k^
    (n-rank)
ll A[105][105],X[105];int Rank;
ll gcdExtended(ll a, ll b, ll& x, ll& y){
    if(a==0) {x=0;y=1; return b;}
    ll x1,y1;
    ll gcd = gcdExtended(b%a,a,x1,y1);
    x=y1-(b/a)*x1;
    y=x1;
    return gcd;
}
ll modinverse(ll x,ll y) {ll a,b; gcdExtended(x,y,a,b);
     return a;}
//n equations (n rows), m variables (m+1 columns)
void Gauss(int n,int m,int k){
    int r,c;
    for(r=0,c=0;r<n && c<m;c++){
        for(int i=r+1;i<n;i++) if(abs(A[i][c])>abs(A[r
            ][c])) swap(A[i],A[r]);
        if(!A[r][c]) continue;
        ll s = modinverse(A[r][c],k);
        for(int i=r+1;i<n;i++) if(A[i][c]){
            ll w = (s*A[i][c])%k;
            /* s bhag hobe r A[i][c] gun hobe*/
            for(int j=c;j<=m;j++){A[i][j]-=(A[r][j]*w)%
                k; A[i][j]%=k; if(A[i][j]<0) A[i][j]+=k
                ;}
        }
        r++;
    }
    //Rank = r
    for(int i=r;i<n;i++) if(A[i][m]) return;  //No
        solution

    //Unique Solution for r variables
    for(int i=r-1;i>=0;i--){
        X[i]=A[i][m];
        for(int j=i+1;j<m;j++) {X[i]-=(A[i][j]*X[j])%k;
            X[i]%=k; if(X[i]<0) X[i]+=k;}

        ll inv=modinverse(A[i][i],k);
        X[i]=(X[i]*inv)%k; if(X[i]<0) X[i]+=k;
    }
}
```

3.2 Fourier transforms

FFT.cpp

89 lines

```
namespace FFT{
 #define ll long long
 #define VI vector<ll>
 #define op operator
 #define ld long double
 #define CN complex<double>
 #define eps 1e-8
 const double PI = 2*acos( 0.0 );
 struct base {
  typedef double T; T re, im;
  base() :re(0), im(0) {}
  base(T re) :re(re), im(0) {}
  base(T re, T im) :re(re), im(im) {}
  base op + (const base& o) const { return base(re + o.
      re, im + o.im); }
```

```
 base op - (const base& o) const { return base(re - o.
     re, im - o.im); }
 base op * (const base& o) const { return base(re * o.
     re - im * o.im, re * o.im + im * o.re); }
 base op * (ld k) const { return base(re * k, im * k)
     ;}
 base conj() const { return base(re, -im); }
};
const int N = 21; /// check before coding
const int MAXN = (1<<N);
base w[MAXN],f1[MAXN]; ll rev[MAXN];
void build_rev(int k) {
 static int rk = -1;
 if( k == rk )return ; rk = k;
 for(int i=1;i<=(1<<k);i++) {
  int j = rev[i-1], t = k-1;
  while( t >= 0 &&((j>>t)&1)){j^=1<<t;--t;}
  if( t >= 0 ) { j ^= 1 << t; --t; }
  rev[i] = j;
 }
}
void fft(base *a, ll k) {
 build_rev(k); ll n = 1 << k;
 for(ll i=0; i<n; i++)
  if( rev[i] > i ) swap(a[i],a[rev[i]]);
 for(ll l=2,llo=1;l<=n;l+=l,llo+=llo) {
  if(w[llo].re == 0 && w[llo].im == 0 ) {
   ld angle = M_PI / llo;
   base ww( cosl(angle), sinl(angle) );
   if( llo > 1 )for(ll j = 0; j < llo;++j){
    if(j&1) w[llo + j] = w[(llo+j)/2]*ww;
    else w[llo + j] = w[(llo+j)/2];
   }
   else w[llo] = base(1, 0);
  }
  for(ll i = 0; i < n; i += l) {
   for(ll j=0; j<llo; j++) {
    base v=a[i+j],u=a[i+j+llo]*w[llo+j];
    a[i + j] = v + u;
    a[i + j + llo] = v - u;
   }}}
}

VI Multiply(VI& a,VI& b) {
 ll k = 1;
 while( (1<<k) <(a.size()+b.size()))++k;
 ll n = (1<<k);
 for(ll i=0;i<n;i++)f1[i]=base(0,0);
 for(ll i=0;i<a.size();i++)
  f1[i]=f1[i]+base(a[i],0);
 for(ll i=0; i<b.size(); i++)
  f1[i] = f1[i] + base(0, b[i]);
 fft(f1, k);
 for(ll i=0; i<1+n/2; i++) {
  base p = f1[i] + f1[(n-i)%n].conj();
  base _q = f1[(n-i)%n] - f1[i].conj();
  base q(_q.im, _q.re);
  f1[i] = (p * q) * 0.25;
  if( i > 0 ) f1[(n - i)] = f1[i].conj();
 }
 for(ll i=0; i<n; i++) f1[i] = f1[i].conj();
 fft(f1, k); VI res(a.size() + b.size());
 for(ll i=0; i<res.size(); i++) {
  if(fabs(f1[i].re) < eps) res[i]=0;
  else res[i] = f1[i].re / fabs(f1[i].re) * (ll) (abs(
      f1[i].re / n) + 0.5);
 }
 return res;
}
VI bigMod(VI& n,ll p){
 VI res=n; VI Ans; Ans.push_back(1);
 while(p){
  if(p%2==1) Ans=Multiply(Ans,res);
  res=Multiply(res,res);p=p/2;
 }
 return Ans;
}
}
using namespace FFT;
```

FWHT.cpp

31 lines

```
#define bitwiseXOR
//#define bitwiseAND
//#define bitwiseOR
void FWHT(vector <ll> &p, bool inverse){
 int n = p.size();
 while(n&(n-1)) {p.push_back(0);n++;}
 for(int len = 1; 2*len <= n; len <<= 1){
  for(int i = 0; i < n; i += len+len) {
   for(int j = 0; j < len; j++) {
    ll u = p[i+j],v = p[i+len+j];
     #ifdef bitwiseXOR
     p[i+j] = u+v; p[i+len+j] = u-v;
     #endif // bitwiseXOR
     #ifdef bitwiseAND
     if(!inverse){p[i+j]=v;p[i+len+j]=u+v;}
     else{p[i+j]=v-u;p[i+len+j] = u;}
     #endif // bitwiseAND
     #ifdef bitwiseOR
     if(!inverse){p[i+j]=u+v;p[i+len+j]=u;}
     else{p[i+j]=v;p[i+len+j]=u-v;}
     #endif // bitwiseOR
   }
  }
 }
 #ifdef bitwiseXOR
 if(inverse) {
  for(int i = 0; i < n; i++)p[i] /= n;
 }
 #endif // bitwiseXOR
}
//FWHT(A,0);for i A[i]*=A[i];FWHT(A,1)
```

NTT.cpp

139 lines

```
#define VI vector<LL>
const int M = 786433;
/*
7340033, 5, 4404020, 1<<20
13631489, 11799463,6244495, 1<<20
23068673, 177147,17187657, 1<<21
463470593, 428228038, 182429, 1<<21
415236097, 73362476, 247718523, 1<<22
918552577, 86995699, 324602258, 1<<22
998244353, 15311432, 469870224, 1<<23
167772161, 243, 114609789, 1<<25
469762049, 2187, 410692747, 1<<26
*/
LL power(LL a, LL p, LL mod) {
    if (p==0)   return 1;
    LL ans = power(a, p/2, mod);
    ans = (ans * ans)%mod;
    if (p%2)    ans = (ans * a)%mod;
    return ans;
}

struct NTT {
    int N;
    vector<int> perm;
    int mod, root, inv, pw;
    NTT(int mod, int root, int inv, int pw) :
        mod(mod), root(root), inv(inv), pw(pw) {}
    void precalculate() {
        perm.resize(N);
        perm[0] = 0;
        for (int k=1; k<N; k<<=1) {
            for (int i=0; i<k; i++) {
                perm[i] <<= 1;
                perm[i+k] = 1 + perm[i];
            }
        }
    }
    void fft(VI &v, bool invert = false) {
        if (v.size() != perm.size()) {
            N = v.size();
            assert(N && (N&(N-1)) == 0);
            precalculate();
        }

        for (int i=0; i<N; i++)
            if (i < perm[i])
                swap(v[i], v[perm[i]]);

        for (int len = 2; len <= N; len <<= 1) {
            LL factor = invert ? inv : root;
            for (int i = len; i < pw; i <<= 1)
                factor = (factor * factor) % mod;

            for (int i=0; i<N; i+=len) {
                LL w = 1;
                for (int j=0; j<len/2; j++) {
```

```
                    LL x = v[i+j], y = (w * v[i+j+len
                        /2])%mod;
                    v[i+j] = (x+y)%mod;
                    v[i+j+len/2] = (x-y+mod)%mod;
                    w = (w * factor)%mod;
                }
            }
        }
        if (invert) {
            LL n1 = power(N, mod-2, mod);
            for (LL &x : v) x=(x*n1)%mod;
        }
    }

    VI multiply(VI a, VI b) {
        while (a.back() == 0 && a.size())   a.pop_back
            ();
        while (b.back() == 0 && b.size())   b.pop_back
            ();
        int n = 1;
        while (n < a.size()+ b.size())  n<<=1;
        a.resize(n);
        b.resize(n);
        fft(a);
        fft(b);
        for (int i=0; i<n; i++) a[i] = (a[i] * b[i])%
            mod;
        fft(a, true);
        return a;
    }
    VI bigMod(VI &base, int p) {
        if(p==0)return {1};
        VI ans = bigMod(base, p/2);
        ans = multiply(ans, ans);
        if (p%2)ans = multiply(ans, base);
        return ans;
    }
};

/** Find primitive root of p assuming p is prime.
if not, we must add calculation of phi(p).
Complexity : O(Ans * log (phi(n)) * log n + sqrt(p)) (
    if exists)
      O(p * log (phi(n)) * log n + sqrt(p))(if does
          not exist)
Returns -1 if not found.
*/
ll primitive_root(ll p){
    if (p == 2) return 1;
    vector<ll> factor;
    ll phi = p-1,  n = phi;

    for (ll i=2; i*i<=n; ++i)
        if (n%i == 0) {
            factor.push_back (i);
            while (n%i==0)  n/=i;
        }
    if (n>1)  factor.push_back(n);

    for (ll res=2; res<=p; ++res) {
        bool ok = true;
        for (ll i=0; i<factor.size() && ok; ++i)
            ok &= power(res, phi/factor[i], p) != 1;
        if (ok)  return res;
    }
    return -1;
}
/**
  Generates necessary info for NTT (for offline usage
      :3).
  Returns maximum k such that 2^k % mod = 1,
  NTT can only be applied for arrays not larger than
      this size.
  mod MUST BE PRIME!!!!!
  We use the fact that if primes have the form p=c*2^k
      +1,
  there always exists the 2^k-th root of unity.
  It can be shown that g^c is such a 2^k-th root
  of unity, where g is a primitive root of p.
*/
ll nttdata(ll mod, ll &root, ll &inv, ll &pw) {
    ll c = 0, n = mod-1;
    while (n%2 == 0) c++, n/=2;
    pw = (mod-1)/n;
    ll g = primitive_root(mod);
    if(g == -1) return -1; // No primitive root exists
    root = power(g, n, mod);
    inv = power(root, mod-2, mod);
    return c;
}
NTT ntt(998244353, 15311432, 469870224, 1<<23);
```

StirlingNumberViaNTT.cpp

28 lines

```
NTT ntt(998244353,15311432,469870224,1<<23);
VI v[MAX];/*strlng1(n,k)=co-effof x^k in
x*(x+1)*(x+2)*....(x+n-1)*/
int strlng1(int n, int r) {
 int nn = 1;while(nn < n) nn <<= 1;
 for(int i = 0; i < n; ++i)
  {v[i].push_back(i);v[i].push_back(1);}
 for(int i=n;i<nn;++i)v[i].push_back(1);
 for(int j = nn; j > 1; j >>= 1) {
  int hn = j >> 1;
  for(int i=0;i<hn;++i)
   v[i]=ntt.multiply(v[i],v[i+hn]);
 }
 return v[0][r];
}
#define mod 100000007
/*strlng2 (n,k) = co-eff of x^k in product of
    polynomials A & B where A(i) = (-1)^i / i!  and B(i
    ) = i^n / i! */
int strlng2(int n, int r) {
 vector<ll>a,b,res;
 a.resize(n+1); b.resize(n+1);
 for(int i = 0; i <= n; i++){
  a[i]=invfct[i]; if(i&2)a[i]=mod-a[i];
 }
 for(int i = 0; i <= n; i++){
  b[i]=(bigmod(i,n,mod)*invfct[i])%mod;
 }
 res=ntt.multiply(a,b);return res[r];
}
```

Number theory (4)

BIGInteger.java

24 lines

```
import static java.lang.System.in;
import java.util.Scanner;
import java.math.BigInteger;
public class Main {
 public static void main(String[] args){
  Scanner in = new Scanner(System.in);
  int n; n = in.nextInt();
  BigInteger A =  in.nextBigInteger();
  A  = BigInteger.valueOf(54);

  String z = A.toString();  intValue();
  BigInteger C = A.add(B);multiply(B);divide(B);
  subtract(B);
  gcd(B); max(B); mod(B); modInverse(mod);
  or(B); pow(B); sqrt(); xor(B);
  BigInteger fact = BigInteger.valueOf(1);
  if (a < b); if(A.compareTo(B) < 0);
  for(int i = 2; i <= 100; i++){
     BigInteger val = BigInteger.valueOf(i);
     fact = fact.multiply(val);
     System.out.println(fact);
  }
 }
}
```

BigModFactInv.cpp

15 lines

```
ll bigmod(ll b,ll e){
 ll ans=1;
 while(e){
 if(e&1)ans=(ans*b)%mod;e>>=1;b=b*b%mod;
 }
 return ans;
}
void fact_cal(){
 fact[0]=1,inv[0]=1;
 for(int i=1;i<=mx-3;i++){
  fact[i]=(fact[i-1]*i)%mod;
 }
 inv[mx-3]=bigmod(fact[mx-3],mod-2);
 for(int i=mx-4;i>=1;i--) inv[i]=(inv[i+1]*(i+1))%mod;
}
```

BabyStepGaintStep.cpp

50 lines

```
int egcd(int a,int b,int& x,int& y){
    if(!b){y=0, x=1; return a;}
    int g = egcd(b,a%b,y,x);
    y-=((a/b)*x);return g;
}

int disc_log(int g,int h,int p){
/*returns smallest x such that
(g^x)%p=h,-1 if none exists*/
    if (h >= p) return -1;
    if ((g % p) == 0){
        if (h == 1) return 0;
        else return -1;
    }
    int i,c,x,y,z,r,m,counter=0;
    ll v=1,d=1,mul=1,temp=1%p;
    for (int i=0;i<100;i++){
        if(temp==h)return i;
        temp =(temp*g)%p;
    }
    while((v=__gcd(g,p))>1){
        if(h % v) return -1;
        h /= v, p /= v;
        d =(d*(g/v))%p;
        counter++;
    }
    m=ceil(sqrt(p));//sqrtl()
    unordered_map<int,int>mp;
    for (i = 0; i < m; i++){
        if(!mp[mul])mp[mul]=i+1;
        mul = (mul * g) % p;
    }
    for (i = 0; i < m; i++){
        z = egcd(d, p, x, y);
        c = p / z;
        r=((((ll)x*h)/z)%p+p)%p;
        if(mp[r])
return((i*m)+mp[r]+counter-1);
        d = (d * mul) % p;
    }
    return -1;
}
int main(){
    int g, h, p, res;
    scanf("%d%d%d",&g,&p,&h);
    res = disc_log(g,h %p,p);
    if (res == -1)
        puts("No Solution");
    else printf("%d\n", res);
}
```

CRT.cpp

24 lines

```
ll ar[mx],br[mx];
struct GCD_type { ll x, y, d; };
GCD_type ex_GCD(ll a, ll b){
 if (b == 0) return {1, 0, a};
 GCD_type pom = ex_GCD(b, a % b);
 return {pom.y, pom.x - a / b * pom.y, pom.d};
}
ll normalize(ll val,ll mod)
 {val%=mod;if(val<0)val+=mod;return val;}
void solve(){
  ll ans=br[1]; /// here br remainder
  ll lcm=ar[1]; bool f=true;
  for(int i=2;i<=n;i++) {
   auto pom=ex_GCD(lcm,ar[i]);
   ll x1=pom.x; ll d=pom.d;
   if((br[i]-ans)%d!=0){
    f=false;break;
   }
   ans=ans+x1*(br[i]-ans)/d%(ar[i]/d)*lcm;
   ans=normalize(ans,lcm*ar[i]/d);
   lcm=(lcm*ar[i])/__gcd(lcm,ar[i]);
  }
  if(f)printf("%lld %lld\n",ans,lcm);
  }///smallest answer .next xth answer will be ans+x*
      lcm where x=[1,2,....]
```

ExtendedEuclidean.cpp

10 lines

```
int Ext_Eucli(int a, int b, int &x, int &y){
 if (b == 0) { x = 1; y = 0;return a;}
 int d = Ext_Eucli(b, a % b, y, x);
 y = y - (a / b) * x; return d;
}
int Inverse_Modulo(int a, int m) {
 int x, y, d; d = Ext_Eucli(a,m,x,y);
 if (d == 1) return (x + m) % m;
 return -1; //No Solution
}
```

InclusionExclusion.cpp

25 lines

```
void func(int idx,int cnt,ll lcm){
 if(lcm>n)break;
 if(idx==m) {
  if(cnt==0)return;
  if(cnt & 1)re1+=n/lcm;else re1-=n/lcm;
  return;
 }
 func(idx+1,cnt+1,(lcm*ar[idx])/ __gcd(lcm,(ll)ar[idx])
     );
 func(idx+1,cnt,lcm);
}
void solve(){
 scanf("%lld%d",&n,&m);
 for(int i=0;i<m;i++)scanf("%d",&ar[i]);
  for(int i=1; i<(1<<m);i++) {
   ll lcm=1;int cnt=0;
   for(int j=0;j<m;j++) {
    if(i & (1<<j)) {
    cnt++;lcm=(lcm*ar[j])/__gcd(lcm,(ll)ar[j]);
     if(lcm>n)break;
    }
   }
   if(cnt&1)re+=n/lcm;
   else re-=n/lcm;
  }
 }
```

LinearSieve.cpp

28 lines

```
bitset<mx>is_composite;vector<int>prime;
int phi[mx],mobius[mx];
void seive(int n){
phi[1]=mobius[1]=1;
for(int i=2; i<=n; i++){
 mobius[i]=1;
 if(!is_composite[i]){
  prime.push_back(i); phi[i]=i-1;
 }
 for(int j=0;j<prime.size()&& i*prime[j]<=n;j++){
  is_composite[i*prime[j]]=true;
  if(i%prime[j]==0){
   phi[i*prime[j]]=phi[i]*prime[j];
   break;
  }
  else{
   phi[i*prime[j]]=phi[i]*phi[prime[j]];
  }
 }
}
 for(int val:prime){
  int temp=val*val;if(temp>n)break;
  for(int j=temp; j<=n; j+=temp)mobius[j]=0;
 }
 for(int val:prime){
  for(int j=val; j<=n; j+=val)mobius[j]*=-1;
 }
}
```

NeajMorshad'sExtraFormula.cpp

7 lines

```
//inner circle radius, r = area * s;
//outer circle area, A = (abc) / 4R;
//N point Polygons Regions, R = (E ... V + 2)
//V = (n + nC4) E = (n * (n ... 1) + nC4 * 4) / 2
/* 0*nC0+1*nC1+2*nC2+3*nC3+..+n*nCn=n*2^(n..1)
0Cr+1Cr+2Cr+3Cr+4Cr+5Cr+6Cr+..+nCr=(n+1)C(r+1)
(nC0)^2+(nC1)^2+(nC2)^2+..+(nCn)^2=(2*n)Cn */
```

PollardRho.cpp

91 lines

```
#define pii pair<ll,int>
ll Mul(ll a, ll b, ll Mod) {
 ll Ans = 0;
 while (b) {
  if (b & 1) {Ans+=a;if(Ans>=Mod)Ans-=Mod;}
  a+=a; if(a>=Mod)a-=Mod;b >>= 1;
 }
 return Ans;
}
ll bigMod(ll n, ll r, ll Mod) {
 if (r == 0) return 1LL;
```

```
 ll ret = bigMod(n, r / 2, Mod);
 ret = Mul(ret, ret, Mod);
 if (r % 2 == 1) ret = Mul(ret, n, Mod);
 return ret;
}
//Miller-Rabin
bool witness(ll wit, ll n) {
 if (wit >= n) return false;
 int s = 0; ll t = n - 1;
 while (t % 2 == 0) s++, t /= 2;
 wit = bigMod(wit, t, n);
 if (wit == 1 || wit == n - 1) return false;
 for (int i = 1; i < s; i++) {
  wit = Mul(wit, wit, n);
  if (wit == 1) return true;
  if (wit == n - 1) return false;
 }
 return true;
}
//Is n prime?
bool miller(ll n) {
 if (n == 2) return true;
 if (n % 2 == 0 || n < 2) return false;
 if (witness(2, n)||witness(7, n)||witness(61, n))
     return false;
 return true;
}
// Pollard's Rho
// a must not equal 0 or -2.
// returns a divisor, a proper one when succeeded,
    equal to n if failed
// in case of failure, change a
ll rho(ll n, ll a) {
 auto f = [&](ll x) {return (Mul(x, x, n) + a) % n; };
 ll x = 2, y = 2;
 for (int i = 1;; i++) {
  x = f(x); y = f(f(y));
  ll d = __gcd(n, abs(x - y));
  if (d != 1) return d;
 }
 return n;
}
ll get_factor(ll n) {
 if(n%2==0) return 2;if(n%3==0)return 3;
 if (n % 5 == 0) return 5;
 while (true) {
  ll a=2+rand()%100; ll d=rho(n,a);
  if (d != n) return d;
 }
 return n;
}
void factorize(ll n, vector<ll> &x) {
 if (n == 1) return;
 else if (miller(n)) x.push_back(n);
 else {
  ll d = get_factor(n);
  factorize(d, x); factorize(n / d, x);
 }
}
vector<ll>factorize(ll n) {vector<ll>x; factorize(n, x)
    ; return x;}
vector<pii>Factors; // store factor
vector<ll>Divisors;//strore divisors
void findDiv(int pos, ll val) {
 if (pos < 0){Divisors.push_back(val);return;}
 ll Now = 1;
 for (int i=0;i<=Factors[pos].second;i++){
  findDiv(pos - 1, val * Now);
  Now = Now * Factors[pos].first;
 }
}
void findAllDiv(ll n) {
 vector<ll>now = factorize(n);
 sort(now.begin(), now.end());
 Factors.clear(); Divisors.clear();
 int Count = 1;
 for (int i = 1; i < now.size(); i++) {
  if (now[i] == now[i - 1]) Count++;
  else {Factors.push_back({now[i - 1], Count}); Count =
      1;}
 }
 Factors.push_back({now.back(), Count});
 findDiv(Factors.size() - 1, 1);
}
```

PrimeCountingFunction.cpp

76 lines

```
const int N = 3e5 + 9;
namespace pcf {
  // initialize once by calling init()
  #define MAXN 20000010 // initial sieve limit
  #define MAX_PRIMES 2000010 // max size of the prime
      array for sieve
  #define PHI_N 100000
  #define PHI_K 100
  int len = 0; // total number of primes generated by
      sieve
  int primes[MAX_PRIMES];
  int pref[MAXN]; // pref[i] --> number of primes <= i
  int dp[PHI_N][PHI_K]; // precal of yo(n,k)
  bitset <MAXN> f;
  void sieve(int n) {
    f[1] = true;
    for (int i = 4; i <= n; i += 2) f[i] = true;
    for (int i = 3; i * i <= n; i += 2) {
      if (!f[i]) {
        for (int j = i * i; j <= n; j += i << 1) f[j] =
            1;
      }
    }
    for (int i = 1; i <= n; i++) {
      if (!f[i]) primes[len++] = i;
      pref[i] = len;
    }
  }
  void init() {
    sieve(MAXN - 1);
    // precalculation of phi upto size (PHI_N,PHI_K)
    for (int n = 0; n < PHI_N; n++) dp[n][0] = n;
    for (int k = 1; k < PHI_K; k++) {
      for (int n = 0; n < PHI_N; n++) {
        dp[n][k] = dp[n][k - 1] - dp[n / primes[k -
            1]][k - 1];
      }
    }
  }
  // returns the number of integers less or equal n
      which are
  // not divisible by any of the first k primes
  // recurrence --> yo(n, k) = yo(n, k-1) - yo(n / p_k
      , k-1)
  // for sum of primes yo(n, k) = yo(n, k-1) - p_k * yo
      (n / p_k , k-1)
  long long yo(long long n, int k) {
    if (n < PHI_N && k < PHI_K) return dp[n][k];
    if (k == 1) return ((++n) >> 1);
    if (primes[k - 1] >= n) return 1;
    return yo(n, k - 1) - yo(n / primes[k - 1], k - 1);
  }
  // complexity: n^(2/3).(log n^(1/3))
  long long Legendre(long long n) {
    if (n < MAXN) return pref[n];
    int lim = sqrt(n) + 1;
    int k = upper_bound(primes, primes + len, lim) -
        primes;
    return yo(n, k) + (k - 1);
  }
  // runs under 0.2s for n = 1e12
  long long Lehmer(long long n) {
    if (n < MAXN) return pref[n];
    long long w, res = 0;
    int b = sqrt(n), c = Lehmer(cbrt(n)), a = Lehmer(
        sqrt(b)); b = Lehmer(b);
    res = yo(n, a) + ((1LL * (b + a - 2) * (b - a + 1))
        >> 1);
    for (int i = a; i < b; i++) {
      w = n / primes[i];
      int lim = Lehmer(sqrt(w)); res -= Lehmer(w);
      if (i <= c) {
        for (int j = i; j < lim; j++) {
          res += j;
          res -= Lehmer(w / primes[j]);
        }
      }
    }
    return res;
  }
}
int32_t main() {
  pcf::init();
  long long n; cin >> n;
  cout << pcf::Lehmer(n) << '\n';
}
```

Combinatorial (5)

5.0.1 Derangements

Permutations of a set such that none of the elements appear in their original position.

$$D(n) = (n-1)(D(n-1)+D(n-2)) = nD(n-1)+(-1)^n = \left\lfloor \frac{n!}{e} \right\rceil$$

5.0.2 Lucas' Theorem

Let n, m be non-negative integers and p a prime. Write $n = n_k p^k + ... + n_1 p + n_0$ and $m = m_k p^k + ... + m_1 p + m_0$. Then $\binom{n}{m} \equiv \prod_{i=0}^{k} \binom{n_i}{m_i} \pmod{p}$.

5.0.3 Stirling numbers of the first kind

Number of permutations on n items with k cycles.

$$c(n,k) = c(n-1,k-1) + (n-1)c(n-1,k),\ c(0,0) = 1$$
$$\sum_{k=0}^{n} c(n,k)x^k = x(x+1)\dots(x+n-1)$$

$c(8,k) = 8, 0, 5040, 13068, 13132, 6769, 1960, 322, 28, 1$
$c(n,2) = 0, 0, 1, 3, 11, 50, 274, 1764, 13068, 109584, \dots$

5.0.4 Stirling numbers of the second kind

Partitions of n distinct elements into exactly k groups.

$$S(n,k) = S(n-1,k-1) + kS(n-1,k)$$
$$S(n,1) = S(n,n) = 1$$
$$S(n,k) = \frac{1}{k!}\sum_{j=0}^{k}(-1)^{k-j}\binom{k}{j}j^n$$

5.0.5 Catalan numbers

$$C_n = \frac{1}{n+1}\binom{2n}{n} = \binom{2n}{n} - \binom{2n}{n+1} = \frac{(2n)!}{(n+1)!n!}$$
$$C_0 = 1,\ C_{n+1} = \frac{2(2n+1)}{n+2}C_n,\ C_{n+1} = \sum C_i C_{n-i}$$

$C_n = 1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, \dots$

- sub-diagonal monotone paths in an $n \times n$ grid.
- strings with n pairs of parenthesis, correctly nested.
- binary trees with with $n+1$ leaves (0 or 2 children).
- ordered trees with $n+1$ vertices.
- ways a convex polygon with $n+2$ sides can be cut into triangles by connecting vertices with straight lines.
- permutations of $[n]$ with no 3-term increasing subseq.

Graph (6)

6.1 Fundamentals

BellmanFord.cpp — 19 lines

```
typedef double ll;
const int maxn = 105;
const int maxm = 10005;
const ll inf = 1e9;
ll d[maxn],w[maxm];
int u[maxm],v[maxm],n,m;
bool BellmanFord(){//1-indexed
 for(int i=1;i<=n;i++)d[i]=inf;d[1]=0;
 for(int i=1; i<=n; i++)
  for(int j=0; j<m; j++)
   if(d[u[j]]+w[j] < d[v[j]])
    d[v[j]]=d[u[j]]+w[j];
 bool negCycle = false;
 for(int j=0; j<m; j++)
  if(d[u[j]]+w[j] < d[v[j]]) {
   negCycle=true; break;
  }
 return negCycle;
}
```

FloydWarshal.cpp — 13 lines

```
for(int i=1;i<=n;i++){
    for(int j=1;j<=n;j++){
        if(i==j || dis[i][j]>0)continue;
        dis[i][j]=1e18;
    }
}
for(int l=1;l<=n;l++){
    for(int i=1;i<=n;i++){
        for(int j=1;j<=n;j++){
            dis[i][j]=min(dis[i][j],dis[i][l]+dis[l][j
                ]);
        }
    }
}
```

DSU.cpp — 18 lines

```
struct DSU{
 vector<int>sz,rnk,par; int c;
 DSU(int n):par(n+1),sz(n+1,1),rnk(n+1,0){
  for(int i=1;i<=n;i++)par[i]=i;c=n;
 }
 int pfind(int u){
 return (par[u]==u?u:(par[u]=pfind(par[u])));
 }
 int get_sz(int u){ return sz[pfind(u)];}
 int Components(){ return c;}
 int Union(int u,int v){
 if((u=pfind(u))==(v=pfind(v)))return -1;
 else --c;
 if(rnk[u]>rnk[v])swap(u,v);par[u]=v;
  sz[v]+=sz[u];if(rnk[u]==rnk[v])rnk[v]++;
  return v;
 }
};
```

DSURollBacks.cpp — 39 lines

```
const int N = 2e5 + 9;
struct DSU {
  vector<int> par,sz;
  vector<int> op;
  DSU() {}
  DSU(int n) {
    par.resize(n + 1);
    sz.resize(n + 1);
    for (int i=1;i<=n;i++){
      par[i] = i;sz[i] = 1;
    }
  }
  int find(int u) {
    int ans = 0;
    while(par[u]!=u){
      u = par[u];
    }
    return u;
  }
  bool merge(int u, int v){
    u = find(u),v = find(v);
    if (u == v) {
      op.push_back(-1);
      return false;
    }
    if(sz[u]>sz[v])swap(u,v);
    op.push_back(u);
    par[u]=v;sz[v]+=sz[u];
    return true;
  }
  void undo() {
    int u =op.back();
    if (u != -1) {
      sz[par[u]] -=sz[u];
      par[u] = u;
    }
    op.pop_back();
  }
};
```

6.2 Network flow

Dinic.cpp — 60 lines

```
// Complexity O(V^2E)
const ll eps = 0; #define INF 1e12
```

```
struct edge {
 int a, b,yo, x, y; ll cap,flow;
};
struct Dinic {
 int s,t,d[mx], ptr[mx] ; //int Id[mx][mx];
 vector<edge>e;
 vector<int>g[mx];
 void init() {
  e.clear(); memset(d,0,sizeof(d));
  for(int i = 0; i < mx ; i++)g[i].clear();
  // for(int i=0;i<mx;i++)
  //  for(int j=0;j<mx;j++)Id[i][j]=0;
 }
 void addEdge(int a,int b,ll cap, int x = -1, int y=
     -1) {
  edge e1={a, b, cap, 0, 1, x, y } ;
  edge e2={b,a,0,0,0,x,y};//Id[a][b]=e.size();
  g[a].push_back((int)e.size());
  e.push_back(e1); //Id[b][a]=e.size();
  g[b].push_back((int)e.size());
  e.push_back(e2);
 }
 bool bfs() {
  queue < int > Q ; Q.push(s);
  memset(d,-1,sizeof(d)); d[s]=0 ;
  while (!Q.empty()) {
   int u=Q.front() ; Q.pop() ;
   for(int i=0; i<g[u].size(); i++) {
    int id=g[u][i], v=e[id].b;
    if(d[v]==-1&&e[id].flow<e[id].cap) {
      Q.push(v); d[v]=d[u]+1 ;
    }}
  }
  return d[t]!=-1 ;
 }
 ll dfs(int u,ll flow) {
  if (flow<=eps)  return 0 ;
  if ( u==t )  return flow ;
  for(int& i = ptr[u] ; i<g[u].size(); i++) {
   int id = g[u][i], v =  e[id].b ;
   if ( d[v] != d[u]+1 )  continue ;
   ll pushed = dfs (v,min (flow,e[id].cap-e[id].flow))
       ;
   if (pushed>eps){e[id].flow+=pushed;
    e[id^1].flow-=pushed;return pushed;
   }
  } return 0 ;
 }
 ll dinic(){ ll flow = 0 ;
  while(true) {
   if(!bfs())  break ;
   memset(ptr, 0, sizeof(ptr)) ;
   while (true){
    ll pushed = dfs(s,INF );
    if(pushed<=eps)break; flow+=pushed;
  } }
  return flow ;
 }
};
Dinic dc;
```

MinCostMaxFlow.cpp

82 lines

```
//Bellmanford O(E^2*V^2), SPFA O(VE)
typedef long long T1;//for cost
typedef long long T2;//for flow
const int maxn = 20100;
const T1 INF = 1e12;
const T2 inf = 1e12;
const T1 eps = 0;
struct Edge {
 int from, to; T2 cap, flow,cost;
};
struct MCMF {   //0-indexed
 int n, m, s, t; vector<Edge> edges;
 vector<int> G[maxn]; int p[maxn],inq[maxn];
 T1 d[maxn]; T2 a[maxn];
 void init() {
  for(int i = 0; i < n; i++) G[i].clear();
  edges.clear();
 }
 void AddEdge(int from,int to,T2 cap,T1 cost){
  edges.push_back((Edge){from,to,cap,0,cost});
  edges.push_back((Edge){to,from,0,0,-cost});
  m = edges.size();
  G[from].push_back(m-2);
  G[to].push_back(m-1);
 }
 pair<T1,T2> Mincost() {//bellmanFord
  T1 tot_cost = 0; T2 tot_flow = 0;
  while(true) {
   for(int i = 0; i < n; i++) d[i] = INF;
   d[s] = 0; p[s] = 0; a[s] = inf;
   bool up=true;
   while(up) {
    up=false;
    for(int u = 0; u < n; u++) {
     if(d[u]-INF>=-eps)continue;
     for(int j:G[u]) {
      Edge &e=edges[j];
      if(e.cap > e.flow &&d[e.to] > d[u]+e.cost+eps){
       d[e.to] = d[u] + e.cost; p[e.to] = j;
       a[e.to] = min(a[u], e.cap - e.flow);
       up=true;
      }}}
   }
   if(abs(d[t]-INF)<=eps)break;
   tot_cost += (T1)d[t] * a[t];
   tot_flow += (T2)a[t]; int u = t;
   while(u != s) {
    edges[p[u]].flow += a[t];
    edges[p[u]^1].flow -= a[t];
    u = edges[p[u]].from;
   }
  }
  return {tot_cost,tot_flow};
 }
 pair<T1,T2> Mincost2() {//SPFA
  T1 tot_cost = 0; T2 tot_flow = 0;
  while(true) {
   for(int i = 0; i < n; i++) d[i] = INF;
   memset(inq, 0, sizeof(inq));
   d[s] = 0; inq[s]=1; p[s]=0; a[s]=inf;
   queue<int>Q; srand(time(NULL)); Q.push(s);
   while(!Q.empty()) {
    int u = Q.front(); Q.pop(); inq[u] = 0;
    for(int i = 0; i < G[u].size(); i++) {
     Edge& e = edges[G[u][i]];
     if(e.cap>e.flow &&d[e.to]>d[u]+e.cost+eps){
      d[e.to] = d[u]+e.cost; p[e.to]=G[u][i];
      a[e.to] = min(a[u], e.cap - e.flow);
      if(!inq[e.to]){Q.push(e.to);inq[e.to]=1;}
     }}
   }
   if(abs(d[t]-INF)<=eps)break;
   tot_cost+=(T1)d[t] *a[t];tot_flow+=a[t];
   int u = t;
   while(u != s) {
    edges[p[u]].flow += a[t];
   edges[p[u]^1].flow-=a[t];u=edges[p[u]].from;
   }
  }
  return {tot_cost,tot_flow};
 }
};
```

UpperLowerBoundFlow.cpp

47 lines

```
Dinic dc; int x,y; // Source and Sink
struct tem{
    int u,v,a,b;
};
vector<tem>ed;
ll func(ll val){
 dc.init(); dc.s=n+1; dc.t=n+2;
 /*for upperbound(0,val),SSS=SuperSuperSource
 dc.addEdge(y,n+3,val);sink to SSS
 dc.addEdge(n+1,x,0);sink to source
 dc.addEdge(n+3,n+2,0);SSS to super sink
 dc.addEdge(n+3,x,val);SSS to source */
 // for lowerbound(val,inf)
 dc.addEdge(y,n+3,INF); //sink to SSS
 dc.addEdge(n+1,x,val); //sink to source
 dc.addEdge(n+3,n+2,val); //SSS to super sink
 dc.addEdge(n+3,x,INF);  //SSS to source
 for(auto it:ed){
  dc.addEdge(n+1,it.v,it.a);
  dc.addEdge(it.u,n+2,it.a);
  dc.addEdge(it.u,it.v,it.b-it.a);
 }
 return dc.dinic();
}
void solve(){
 scanf("%d%d",&n,&m); scanf("%d%d",&x,&y);
```

```
 dc.addEdge(y,x,INF); dc.s=n+1; dc.t=n+2;
 ll val=0; ll en=0;
 for(int i=1;i<=m;i++){
  int u,v,a,b;
  scanf("%d%d%d%d",&u,&v,&a,&b);
  ed.push_back({u,v,a,b});
  val+=a; en+=b; dc.addEdge(n+1,v,a);
  dc.addEdge(u,n+2,a);dc.addEdge(u,v,b-a);
 }
 if(dc.dinic()<val){
  printf("0\n");
  return;
 }
 ll be=re=val;
 while(be<=en){
  ll mid=(be+en)/2; ll have=func(mid);
  if(have>=mid+val){re=mid;be=mid+1;}
  else en=mid-1;
 }
 printf("%lld\n",re);
}
```

6.3 Matching

HopcroftKarp.cpp

54 lines

```
// Maximum Matching takes O(E*sqrt(V))
#define mx 40005 #define INF (1<<28)
struct Hopcroft_Karp {
 vector< int > g[mx];
 int n, m, Matching[mx], Distance[mx];
 /*n: number of nodes on left side, nodes are numbered
      1 to n
 m: number of nodes on right side, nodes are numbered n
      +1 to n+m
*/
 void init(int num){
  for(int i=0;i<=num;i++)
   Matching[i]=0,Distance[i]=0,g[i].clear();
 }
 void addEdge(int u,int v){
  g[u].push_back(v); // Directed graph
 }
 bool bfs() {
  int i, u, v, len; queue< int >q;
  for(i=1; i<=n; i++) {
  if(Matching[i]==0){Distance[i]=0;q.push(i);}
  else Distance[i] = INF;
  }
  Distance[0] = INF;
  while(!q.empty()) {
   u = q.front(); q.pop();
   if(u!=0) {
    for(int v:g[u]) {
     if(Distance[Matching[v]]==INF) {
      Distance[Matching[v]] = Distance[u]+1;
      q.push(Matching[v]);
    }}}
  }
  return (Distance[0]!=INF);
 }
 bool dfs(int u) {
  int i, v, len;
  if(u!=0) {
   for(int v:g[u]) {
    if(Distance[Matching[v]]==Distance[u]+1){
     if(dfs(Matching[v])) {
      Matching[v] = u; Matching[u] = v;
      return true;
     } }
   }
   Distance[u] = INF; return false;
  }
  return true;
 }
 int hopcroft_karp(){ int Matchinging=0,i;
  while(bfs())
   for(i=1; i<=n; i++)
    if(Matching[i]==0 && dfs(i))
     Matchinging++; return Matchinging;
 }
};
```

Hungarian.cpp

31 lines

```
/*Given a n*n square matrix, you need to select n
    elements in it so that exactly one element is
    selected in each row and column, and the sum of the
     values of these elements is the smallest.
    Complexity O(n^3)*/
#define INF 1e18
pair<ll,vector<int>> hungarian (vector<vector<ll>>mat,
    int f,int sz){
 vector<int>par(sz+1,0),way(sz+1,0),match(sz+1,0);
 vector<bool>vis(sz+1,0);
 vector<ll>U(sz+1,0),V(sz+1,0),MinV(sz+1,0);
 for(int i=1;i<=sz;i++){
  for(int j=1;j<=sz;j++){mat[i][j]*=f;}
 }
 int a,b,d; ll r,w;
 for(int i=1;i<=sz;i++){ par[0]=i; b=0;
  for(int j=1;j<=sz;j++)MinV[j]=INF,vis[j]=0;
  do{ vis[b]=1; a=par[b],d=0,w=INF;
    for(int j=1;j<=sz;j++){
     if(!vis[j]){
      r=mat[a][j]-U[a]-V[j];
      if(r<MinV[j])MinV[j]=r,way[j]=b;
      if(MinV[j]<w)w=MinV[j],d=j;
     }
    }
    for(int j=0;j<=sz;j++){
     if(vis[j])U[par[j]]+=w,V[j]-=w;
     else MinV[j]-=w;
    } b=d;
  }
  while(par[b]!=0);
  do{d=way[b];par[b]=par[d],b=d;} while(b!=0);
 }
 for(int j=1;j<=sz;j++)match[par[j]]=j;
 return {-f*V[0],match};
} // called hungarain(mat,1,n)
```

Kuhn.cpp

34 lines

```
// for weighted lightoj 1150 solution,O(VE)
struct BPM{
 bool Done[mx];vector<int>g[mx];int macth[mx];
 void addEdge(int u,int v) g[u].push_back(v);
 void init(){for(int i=0;i<mx;i++)g[i].clear();}
 bool Tem_Matching(int u){
  for(int i=0;i<(int)g[u].size();i++){
   int v=g[u][i];if(Done[v])continue;Done[v]=1;
    if(macth[v]==-1 ||Tem_Matching(macth[v]))
      {macth[v] = u; return true;}
  }return false;
 }
 int Max_Matching(int num){
//Be Carefull when passing the num.
  memset(macth,-1,sizeof(macth));int re=0;
   for(int i=1;i<=num;i++){
    memset(Done,false,sizeof(Done));
    if(Tem_Matching(i)) re++;
   }return re;
 }
};
/*Maximum Independent Set in Bipartite Graph
-> Largest set of nodes who do not have any edge
    between themselves
-> Solution: V- Max Matching
Minimum Vertex Cover in Bipartite Graph
-> Smallest set of nodes where at least one end-point
    of each edge is present
-> Solution: Max Matching
Minimum Edge Cover in General Graph
-> Smallest set of edges where each vertex is end-point
     of at least one edge
-> V- Matching(if edge cover exists)
Minimum Path Cover(Vertex Disjoint) in DAG
-> Minimum number of vertex disjoint paths that visit
    all nodes
Minimum Path Cover(Vertex Not Disjoint) in General
    Graph
-> Minimum number paths that visit all nodes*/
```

6.4 DFS algorithms

SCC.cpp

52 lines

```
vector<int>g[mx],g_rev[mx],st(mx),en(mx), component[mx
    ],option,visit;
vector<pair<int,int>>dekhi;
int node,edge,cnt,tem,mp[mx];
void dfs1(int u){
 visit[u]=true; st[u]=++cnt;
 for(auto it:g[u]) {
```

```
  if(visit[it])continue; dfs1(it);
 }
 en[u]=++cnt;
}
void dfs2(int u){
 visit[u]=true;component[cnt].push_back(u);
 for(auto it:g_rev[u]) {
  if(visit[it])continue; dfs2(it);
 }
}
void clean(){
 for(int i=1;i<=node+2;i++) {
  g[i].clear(); g_rev[i].clear();
  component[i].clear();
 }
 option.clear(); cnt=0; st.clear();
 en.clear(); dekhi.clear();
 memset(mp,0,sizeof(mp));
}
void solve(){
 scanf("%d%d",&node,&edge);
 for(int i=1;i<=edge;i++) {
  int u,v;  scanf("%d%d",&u,&v);
  g[u].push_back(v); g_rev[v].push_back(u);
  mp[u]++;mp[v]++;
 }
 visit.assign(node+2,false);
 for(int i=1;i<=node;i++) {
  if(visit[i]==true)continue;
  dfs1(i);
 }
 for(int i=1;i<=node;i++) {
  if(visit[i]==true) dekhi.push_back({en[i],i});
 }
 sort(dekhi.begin(),dekhi.end());
 reverse(dekhi.begin(),dekhi.end());
 visit.assign(node+2,false); cnt=1;
 for(int i=0;i<dekhi.size();i++) {
  int pos=dekhi[i].second;
  if(visit[pos])continue;
  dfs2(pos); cnt++;
 }
 for(int i=1;i<cnt;i++) {
  for(auto it:component[i]) cout<<it<<" "; cout<<endl;
 }
}
```

ArticulationBridge.cpp

14 lines

```
vector<int>g[mx];int Time=1;int st[mx];
vector<pair<int,int>>Bridge;int low[mx];
void dfs(int u,int p){
 st[u]=low[u]=Time++;int child=0;
 for(auto it:g[u]) {
  if(it==p)continue;
  if(st[it]==0){
   dfs(it,u);
  if(st[u]<low[it])Bridge.push_back({u,it});
  low[u]=min(low[u],low[it]);
  }
  else low[u]=min(low[u],st[it]);
 }
}
```

ArticulationPoint.cpp

19 lines

```
vector<int>g[mx]; int Time=1;
int articular_point[mx],st[mx],low[mx];
int dfs(int u,int p){
 st[u]=low[u]=Time++; int child=0;
 for(auto it:g[u]) {
  if(it==p)continue;
  if(st[it]==0) {
   child++; dfs(it,u);
   if(st[u]<=low[it])articular_point[u]=1;
   low[u]=min(low[u],low[it]);
  }
  else low[u]=min(low[u],st[it]);
 }
 return child;
}
for(int i=1;i<=n;i++) {
 if(st[i])continue;
 articular_point[i]=(dfs(i,-1)>1);
}
```

6.5 Trees

LCA.cpp

48 lines

```
int par[mx][20]; ll ans[mx][20];
int depth[mx],LOG; vector<pair<int,ll>>g[mx];
void dfs(int u,int p,int lvl){
 par[u][0]=p; depth[u]=lvl;
 for(auto it:g[u]) {
 int v=it.first;ll w=it.second;if(v==p)continue;
 ans[v][0]=w;dfs(v,u,lvl+1);
 }
} // for node value ans[u][0]=ar[u]
void init(int root){
 dfs(root,-1,1);
 for(int j=1;j<LOG;j++){
  for(int i=1;i<=n;i++){
   if(par[i][j-1]!=-1){
    par[i][j]=par[par[i][j-1]][j-1];
    ans[i][j]=max(ans[i][j-1], ans[par[i][j-1]][j-1]);}
   else par[i][j]=-1;
  }}
}
ll query(int u,int v){
if(u==v)return 0;if(depth[u]<depth[v])swap(u,v);
 int diff=depth[u]-depth[v]; ll re=0;
 for(int i=LOG-1;i>=0;i--) {
  if(diff>=(1<<i)){
   diff-=(1<<i);re=max(re,ans[u][i]);
   u=par[u][i];
 }}
 if(u==v)return re;
 for(int i=LOG-1;i>=0;i--){
  if(par[u][i]!=par[v][i]){
   re=max({re,ans[u][i],ans[v][i]});
   u=par[u][i];v=par[v][i];
 }}// for node also re=max(re,ans[par[u][0]][0])
 re=max({re,ans[u][0],ans[v][0]});
 return re;
}
int dist(int u,int v){
 return depth[u]+depth[v]-2*depth[lca(u,v)];
}
int kth_parent(int u,int k){
 for(int i=LOG-1;i>=0;i--){
  if(k>=(1<<i)) {
   k-=(1<<i);u=par[u][i];
  }
  if(u==-1)return u;
 }
 return u;
}
```

CentroidDecomposition.cpp

78 lines

```
int dis[18][mx],re[mx],vis[mx];
int p[mx],sub[mx],lvl[mx];
vector<int>g[mx],ng[mx];
/* p[u] = parent of u in centroid tree
dis[x][u] = distance from u to a parent of u at level x
      of centroid tree
if u is in subtree of centroid c, then dis[lvl[c]][u] =
      dist(c, l)
If (x, y) edge exist, then x must be in g[y] and y must
      be in g[x]*/
/* we can do more pre work in dfs function*/
void dfs(int l,int u,int par){
 if(par!=-1)dis[l][u]=dis[l][par]+1;
 for(int v:g[u])
  if(v!=par && !vis[v])dfs(l,v,u);
}
int centroid(int u,int par,int r){
 for(int v:g[u])
  if(v!=par && !vis[v] && sub[v]>r)
   return centroid(v,u,r);
 return u;
}
void pre_cal(int u,int par){
 sub[u]=1;
 for(int v:g[u])
  if(v!=par && !vis[v]) pre_cal(v,u),sub[u]+=sub[v];
}
void decompose(int u,int par){
 pre_cal(u,-1);
 int tem=centroid(u,-1,sub[u]>>1);
 vis[tem]=1,p[tem]=par,lvl[tem]=0;
 if(par!=-1)lvl[tem]=lvl[par]+1, ng[par].push_back(tem)
      ;
 dfs(lvl[tem],tem,-1);
```

```
 for(int v:g[tem])
  if(v!=par && !vis[v])decompose(v,tem);
}
void update(int u){
 for(int v=u;v!=-1;v=p[v])re[v] = min(re[v],dis[lvl[v
     ]][u]);
}
int query(int u){
 int ans=1e9;
 for(int v=u;v!=-1;v=p[v])
  ans=min(ans,re[v]+dis[lvl[v]][u]);
 return ans;
}
int lca(int u,int v){
 if(lvl[u]<lvl[v])swap(u,v);
 while(lvl[u]>lvl[v])u=p[u];
 while(u!=v && p[u]!=-1)u=p[u],v=p[v];
 return u;
}
int dist(int u,int v){
 int lc=lca(u,v);
 return dis[lvl[lc]][u]+dis[lvl[lc]][v];
}
int GetRoot(int u){
 while(p[u]!=-1)u=p[u]; return u;
}
//for all pair
void update(int u,int p){
 int val=dis[lvl[p]][u];
 for(int i=0;i<20;i++){
  cnt[i][chk(val,i)]++;
 }
 for(int v:ng[u])update(v,p);
}
void query(int u,int p){
 int val=dis[lvl[p]][u]^ar[p];
 for(int i=0;i<20;i++){
  ans+=cnt[i][!chk(val,i)]*(1LL<<i);
 }
 for(int v:ng[u])query(v,p);
}
void Go_Ahead(int u){
 memset(cnt,0,sizeof(cnt));
 for(int i=0;i<20;i++)cnt[i][chk(ar[u],i)]++;
 for(int v:ng[u]){query(v,u); update(v,u);}
 ans+=ar[u];
 for(int v:ng[u])Go_Ahead(v);
}
// at first call decompose(1,-1)
```

MDST.cpp

57 lines

```
const int inf = 1e9 ;
struct edge {
    int u, v, w;
    edge() {}
    edge(int a,int b,int c) : u(a), v(b), w(c) {}
    bool operator < (const edge& o) const {
        if (u == o.u)
            if (v == o.v)return w < o.w;
            else  return v < o.v;
        return u < o.u;
    }
};
int dmst(vector<edge> &edges, int root) { // 0 base
    node 0 to n-1
    int ans = 0;
    int cur_nodes = n;
    while (true) {
        vector<int> lo(cur_nodes, inf), pi(cur_nodes,
            inf);
        for (int i = 0; i < edges.size(); ++i) {
            int u = edges[i].u, v = edges[i].v, w =
                edges[i].w;
            if (w < lo[v] and u != v) {
                lo[v] = w;
                pi[v] = u;
            }
        }
        lo[root] = 0;
        for (int i = 0; i < lo.size(); ++i) {
            if (i == root) continue;
            if (lo[i] == inf) return -1;
        }
        int cur_id = 0;
        vector<int> id(cur_nodes, -1), mark(cur_nodes,
            -1);
        for (int i = 0; i < cur_nodes; ++i) {
            ans += lo[i];
            int u = i;
            while (u != root and id[u] < 0 and mark[u]
                != i) {
                mark[u] = i;
                u = pi[u];
            }
            if (u != root and id[u] < 0) { // Cycle
                for (int v = pi[u]; v != u; v = pi[v])
                    id[v] = cur_id;
                id[u] = cur_id++;
            }
        }
        if (cur_id == 0) break;
        for (int i = 0; i < cur_nodes; ++i)
            if (id[i] < 0) id[i] = cur_id++;
        for (int i = 0; i < edges.size(); ++i) {
            int u = edges[i].u, v = edges[i].v, w =
                edges[i].w;
            edges[i].u = id[u];
            edges[i].v = id[v];
            if (id[u] != id[v]) edges[i].w -= lo[v];
        }
        cur_nodes = cur_id;
        root = id[root];
    }
    return ans;
}
```

Geometry (7)

ClosestPair.cpp

30 lines

```
ll closest_pair(vector<pair<int,int>>point)
{
    sort(point.begin(),point.end());
    set<pair<int,int>>event;
    ll re=4e18;
    int id=0;
    int n=point.size();
    for(int i=0;i<n;i++)
    {
        int sq_re=ceil(sqrt(re));
        while(point[i].first-point[id].first>=re)
        {
            event.erase(event.find({point[id].second,
                point[id].first}));
            id++;
        }
        pair<int,int>a={point[i].second-sq_re,point[i].
            first};
        pair<int,int>b={point[i].second+sq_re,point[i].
            first};
        auto it1=event.lower_bound(a);
        auto it2=event.upper_bound(b);
        while(it1!=it2)
        {
            int dx=point[i].first-it1->second;
            int dy=point[i].second-it1->first;
            re=min(re,1LL*dx*dx+1LL*dy*dy);
            it1++;
        }
        event.insert({point[i].second,point[i].first});
    }
    return re;
}
```

7.1 Geometric primitives

Point.cpp

50 lines

```
const int N = 3e5 + 9;
typedef long double Tf;
const double inf = 1e100;
const double eps = 1e-9;
const double PI = acos((double) - 1.0);

int sign(double x) { return (x > eps) - (x < -eps); }
struct PT {
    Tf x, y;
    void read () { scanf("%LF%LF", &x, &y); }
    void write () { printf("%LF %LF\n", x, y); }
    PT() { x = 0, y = 0; }
    PT(double x, double y) : x(x), y(y) {}
    PT(const PT &p) : x(p.x), y(p.y)    {}
    PT operator + (const PT &a) const { return PT(x + a
        .x, y + a.y); }
```

```
    PT operator-(const PT &a) const { return PT(x - a.x
        , y - a.y); }
    PT operator * (const double a) const { return PT(x
        * a, y * a); }
    friend PT operator * (const double &a, const PT &b)
         { return PT(a * b.x, a * b.y); }
    PT operator / (const double a) const { return PT(x
        / a, y / a); }
    bool operator == (PT a) const { return sign(a.x - x
        ) == 0 && sign(a.y - y) == 0; }
    bool operator != (PT a) const { return !(*this == a
        ); }
    bool operator < (PT a) const { return sign(a.x - x)
         == 0 ? y < a.y : x < a.x; }
    bool operator > (PT a) const { return sign(a.x - x)
         == 0 ? y > a.y : x > a.x; }
    double norm() { return sqrt(x * x + y * y); }
    double norm2() { return x * x + y * y; }
    PT perp() { return PT(-y, x); }
    double arg() { double x = atan2(y, x); return x; }
        /// if (sign(x) < 0) x += 2 * PI;
    PT truncate(double r) { // returns a vector with
        norm r and having same direction
        double k = norm();
        if (!sign(k)) return *this;
        r /= k;
        return PT(x * r, y * r);
    }
    friend istream &operator >> (istream &is, PT &p) {
        return is >> p.x >> p.y; }
    friend ostream &operator << (ostream &os, const PT
        &p) { return os << p.x << " " << p.y; }
};
inline double dot(PT a, PT b) { return a.x * b.x + a.y
    * b.y; }
inline double dist2(PT a, PT b) { return dot(a - b, a -
     b); }
inline double dist(PT a, PT b) { return sqrt(dot(a - b,
     a - b)); }
inline double cross(PT a, PT b) { return a.x * b.y - a.
    y * b.x; }
inline double cross2(PT a, PT b, PT c) { return cross(b
     - a, c - a); }
inline int orientation(PT a, PT b, PT c) { return sign(
    cross(b - a, c - a)); }
PT perp(PT a) { return PT(-a.y, a.x); }
PT rotateccw90(PT a) { return PT(-a.y, a.x); }
PT rotatecw90(PT a) { return PT(a.y, -a.x); }
PT rotateccw(PT a, double t) { return PT(a.x * cos(t) -
     a.y * sin(t), a.x * sin(t) + a.y * cos(t)); }
PT rotatecw(PT a, double t) { return PT(a.x * cos(t) +
    a.y * sin(t), -a.x * sin(t) + a.y * cos(t)); }
double SQ(double x) { return x * x; }
double rad_to_deg(double r) { return (r * 180.0 / PI);
    }
double deg_to_rad(double d) { return (d * PI / 180.0);
    }
```

AngleInTwoVector.cpp

4 lines

```
double get_angle(PT a, PT b) {
    double costheta = dot(a, b) / a.norm() / b.norm();
    return acos(max((double) - 1.0, min((double)1.0,
        costheta)));
}
```

PointInAngle.cpp

5 lines

```
bool is_point_in_angle(PT b, PT a, PT c, PT p) { //
    does point p lie in angle <bac
    assert(orientation(a, b, c) != 0);
    if (orientation(a, c, b) < 0) swap(b, c);
    return orientation(a, c, p) >= 0 && orientation(a,
        b, p) <= 0;
}
```

Half.cpp

3 lines

```
bool half(PT p) {
    return p.y > 0.0 || (p.y == 0.0 && p.x < 0.0);
}
```

PolarSort.cpp

5 lines

```
void polar_sort(vector<PT> &v) { // sort points in
    counterclockwise
    sort(v.begin(), v.end(), [](PT a, PT b) {
        return make_tuple(half(a), 0.0, a.norm2()) <
            make_tuple(half(b), cross(a, b), b.norm2())
            ;
    });
}
```

7.2 Line

LineStructure.cpp

48 lines

```
struct line {
    PT a, b; // goes through points a and b
    PT v; double c;  //line form: direction vec [cross]
         (x, y) = c
    line() {}
    line(PT v, double c) : v(v), c(c) {
        auto p = get_points(); //direction vector v and
             offset c
        a = p.first; b = p.second;
    }
    line(double _a, double _b, double _c) : v( {_b, -_a
        }), c(-_c) {
        auto p = get_points(); // equation ax + by + c
            = 0
        a = p.first; b = p.second;
    }
    // goes through points p and q
    line(PT p, PT q) : v(q - p), c(cross(v, p)), a(p),
        b(q) {}
    pair<PT, PT> get_points() { //extract any two
        points from this line
        PT p, q; double a = -v.y, b = v.x; // ax + by =
            c
        if (sign(a) == 0) {
            p = PT(0, c / b);
            q = PT(1, c / b);
        }
        else if (sign(b) == 0) {
            p = PT(c / a, 0);
            q = PT(c / a, 1);
        }
        else {
            p = PT(0, c / b);
            q = PT(1, (c - a) / b);
        }
        return {p, q};
    }
    array<double, 3> get_abc() { //ax + by + c = 0
        double a = -v.y, b = v.x;
        return {a, b, c};
    }
    // 1 if on the left, -1 if on the right, 0 if on
        the line
    int side(PT p) { return sign(cross(v, p) - c); }
    // line that is perpendicular to this and goes
        through point p
    line perpendicular_through(PT p) { return {p, p +
        perp(v)}; }
    // translate the line by vector t i.e. shifting it
        by vector t
    line translate(PT t) { return {v, c + cross(v, t)};
         }
    // compare two points by their orthogonal
        projection on this line
    // a projection point comes before another if it
        comes first according to vector v
    bool cmp_by_projection(PT p, PT q) { return dot(v,
        p) < dot(v, q); }
    line shift_left(double d) {
        PT z = v.perp().truncate(d);
        return line(a + z, b + z);
    }
};
```

PointAlongLineDistanceD.cpp

4 lines

```
// find a point from a through b with distance d
PT point_along_line(PT a, PT b, double d) {
    return a + (((b - a) / (b - a).norm()) * d);
}
```

ProjectFromPointToLine.cpp

4 lines

```
// projection point c onto line through a and b
    assuming a != b
PT project_from_point_to_line(PT a, PT b, PT c) {
    return a + (b - a) * dot(c - a, b - a) / (b - a).
        norm2();
}
```

ReflectionFromPointToLine.cpp

5 lines

```
// reflection point c onto line through a and b
    assuming a != b
PT reflection_from_point_to_line(PT a, PT b, PT c) {
    PT p = project_from_point_to_line(a, b, c);
    return point_along_line(c, p, 2.0 * dist(c, p));
}
```

DistFromPointToLine.cpp

4 lines

```
// minimum distance from point c to line through a and
    b
double dist_from_point_to_line(PT a, PT b, PT c) {
    return fabs(cross(b - a, c - a) / (b - a).norm());
}
```

IsPointOnSegment.cpp

8 lines

```
bool is_point_on_seg(PT a, PT b, PT p) {
    if (fabs(cross(p - b, a - b)) < eps) {
        if (p.x < min(a.x, b.x) || p.x > max(a.x, b.x))
            return false;
        if (p.y < min(a.y, b.y) || p.y > max(a.y, b.y))
            return false;
        return true;
    }
    return false; // returns true if  point p is on
        line segment ab
}
```

ProjectFromPointToSeg.cpp

9 lines

```
// minimum distance point from point c to segment ab
    that lies on segment ab
PT project_from_point_to_seg(PT a, PT b, PT c) {
    double r = dist2(a, b);
    if (fabs(r) < eps) return a;
    r = dot(c - a, b - a) / r;
    if (r < 0) return a;
    if (r > 1) return b;
    return a + (b - a) * r;
}
```

DistFromPointToSeg.cpp

4 lines

```
// minimum distance from point c to segment ab
double dist_from_point_to_seg(PT a, PT b, PT c) {
    return dist(c, project_from_point_to_seg(a, b, c));
}
```

IsParallel.cpp

8 lines

```
bool is_parallel(PT a, PT b, PT c, PT d) {
    double k = fabs(cross(b - a, d - c));
    if (k < eps) {
        if (fabs(cross(a - b, a - c)) < eps && fabs(
            cross(c - d, c - a)) < eps) return 2;
        else return 1;
    } // 0 if not parallel, 1 if parallel, 2 if
        collinear
    else return 0;
}
```

AreLinesSame.cpp

4 lines

```
bool are_lines_same(PT a, PT b, PT c, PT d) {
    if (fabs(cross(a - c, c - d)) < eps && fabs(cross(b
        - c, c - d)) < eps) return true;
    return false; // check if two lines are same
}
```

AngleBisector.cpp

4 lines

```
PT angle_bisector(PT &a, PT &b, PT &c) {
    PT p = a - b, q = c - b; // bisector vector of <abc
    return p + q * sqrt(dot(p, p) / dot(q, q));
}
```

PointLineRelation.cpp

6 lines

```
int point_line_relation(PT a, PT b, PT p) {
    int c = sign(cross(p - a, b - a));
    if (c < 0) return 1; // if point is ccw
    if (c > 0) return 2; // if point is cw to the line
    return 3; // if point is on the line
}
```

LineLineIntersection.cpp

9 lines

```
// intersection point between ab and cd assuming unique
    intersection exists
bool line_line_intersection(PT a, PT b, PT c, PT d, PT
    &ans) {
    double a1 = a.y - b.y, b1 = b.x - a.x, c1 = cross(a
        , b);
    double a2 = c.y - d.y, b2 = d.x - c.x, c2 = cross(c
        , d);
    double det = a1 * b2 - a2 * b1;
    if (det == 0) return 0;
    ans = PT((b1 * c2 - b2 * c1) / det, (c1 * a2 - a1 *
        c2) / det);
    return 1;
}
```

SegSegIntersection.cpp

10 lines

```
// intersection point between segment ab and segment cd
    assuming unique intersection exists
bool seg_seg_intersection(PT a, PT b, PT c, PT d, PT &
    ans) {
    double oa = cross2(c, d, a), ob = cross2(c, d, b);
    double oc = cross2(a, b, c), od = cross2(a, b, d);
    if (oa * ob < 0 && oc * od < 0) {
        ans = (a * ob - b * oa) / (ob - oa);
        return 1;
    }
    else return 0;
}
```

SegSegIntersectionInside.cpp

12 lines

```
// intersection point between segment ab and segment cd
    assuming unique intersection may not exists
// se.size()==0 for no intersection,1 for one
    intersection,2 for range intersection
set<PT> seg_seg_intersection_inside(PT a,  PT b,  PT c,
    PT d) {
    PT ans;
    if (seg_seg_intersection(a, b, c, d, ans)) return {
        ans};
    set<PT> se;
    if (is_point_on_seg(c, d, a)) se.insert(a);
    if (is_point_on_seg(c, d, b)) se.insert(b);
    if (is_point_on_seg(a, b, c)) se.insert(c);
    if (is_point_on_seg(a, b, d)) se.insert(d);
    return se;
}
```

SegLineRelation.cpp

9 lines

```
// intersection  between segment ab and line cd
// 0 if do not intersect, 1 if proper intersect, 2 if
    segment intersect
int seg_line_relation(PT a, PT b, PT c, PT d) {
    double p = cross2(c, d, a);
    double q = cross2(c, d, b);
    if (sign(p) == 0 && sign(q) == 0) return 2;
    else if (p * q <= 0) return 1;
    else return 0;
}
```

SegLineIntersection.cpp

7 lines

```
// intersection between segament ab and line cd
    assuming unique intersection exists
bool seg_line_intersection(PT a, PT b, PT c, PT d, PT &
    ans) {
    bool k = seg_line_relation(a, b, c, d);
    assert(k != 2);
    if (k) line_line_intersection(a, b, c, d, ans);
    return k;
}
```

DistFromSegToSeg.cpp

8 lines

```
// minimum distance from segment ab to segment cd
double dist_from_seg_to_seg(PT a, PT b, PT c, PT d) {
    PT dummy;
    if (seg_seg_intersection(a, b, c, d, dummy)) return
        0.0;
    else return min({dist_from_point_to_seg(a, b, c),
        dist_from_point_to_seg(a, b, d),
                     dist_from_point_to_seg(c, d, a
                         ), dist_from_point_to_seg(
                         c, d, b)
                    });
}
```

```
}
```

DistFromPointToSeg.cpp

4 lines

```
// minimum distance from point c to segment ab
double dist_from_point_to_seg(PT a, PT b, PT c) {
    return dist(c, project_from_point_to_seg(a, b, c));
}
```

RayRayIntersection.cpp

9 lines

```
bool ray_ray_intersection(PT as, PT ad, PT bs, PT bd) {
    double dx = bs.x - as.x, dy = bs.y - as.y;
    double det = bd.x * ad.y - bd.y * ad.x;
    if (fabs(det) < eps) return 0;
    double u = (dy * bd.x - dx * bd.y) / det;
    double v = (dy * ad.x - dx * ad.y) / det;
    if (sign(u) >= 0 && sign(v) >= 0) return 1;
    else return 0; // starting point as and direction
        vector ad
}
```

RayRayDistance.cpp

6 lines

```
double ray_ray_distance(PT as, PT ad, PT bs, PT bd) {
    if (ray_ray_intersection(as, ad, bs, bd)) return
        0.0;
    double ans = dist_from_point_to_ray(as, ad, bs);
    ans = min(ans, dist_from_point_to_ray(bs, bd, as));
    return ans;
}
```

7.3 Circles

CircleStructure.cpp

31 lines

```
struct circle {
    PT p; double r;
    circle() {}
    circle(PT _p, double _r): p(_p), r(_r) {};
    // center (x, y) and radius r
    circle(double x, double y, double _r): p(PT(x, y)),
        r(_r) {};
    // circumcircle of a triangle
    // the three points must be unique
    circle(PT a, PT b, PT c) {
        b = (a + b) * 0.5;
        c = (a + c) * 0.5;
        line_line_intersection(b, b + rotatecw90(a - b)
            , c, c + rotatecw90(a - c), p);
        r = dist(a, p);
    }
    // inscribed circle of a triangle
    Tf sector(Tf alpha) { return r * r * 0.5 * (alpha -
        sin(alpha)); }
    circle(PT a, PT b, PT c, bool t) {
        line u, v;
        double m = atan2(b.y - a.y, b.x - a.x), n =
            atan2(c.y - a.y, c.x - a.x);
        u.a = a;
        u.b = u.a + (PT(cos((n + m) / 2.0), sin((n + m)
            / 2.0)));
        v.a = b;
        m = atan2(a.y - b.y, a.x - b.x), n = atan2(c.y
            - b.y, c.x - b.x);
        v.b = v.a + (PT(cos((n + m) / 2.0), sin((n + m)
            / 2.0)));
        line_line_intersection(u.a, u.b, v.a, v.b, p);
        r = dist_from_point_to_seg(a, b, p);
    }
    bool operator == (circle v) { return p == v.p &&
        sign(r - v.r) == 0; }
    double area() { return PI * r * r; }
    double circumference() { return 2.0 * PI * r; }
};
```

CirclePointRelation.cpp

6 lines

```
int circle_point_relation(PT p, double r, PT b) {
    double d = dist(p, b);
    if (sign(d - r) < 0) return 2; // if inside circle
    if (sign(d - r) == 0) return 1; // if on
        circumference
    return 0; // if outside
}
```

CircleLineRelation.cpp

6 lines

```
int circle_line_relation(PT p, double r, PT a, PT b) {
    double d = dist_from_point_to_line(a, b, p);
    if (sign(d - r) < 0) return 2; // if inside circle
    if (sign(d - r) == 0) return 1; // if on
        circumference
    return 0; // if outside
}
```

CircleLineIntersection.cpp

12 lines

```
//compute intersection of line through points a and b
    with
//circle centered at c with radius r > 0
vector<PT> circle_line_intersection(PT c, double r, PT
    a, PT b) {
    vector<PT> ret;
    b = b - a; a = a - c;
    double A = dot(b, b), B = dot(a, b);
    double C = dot(a, a) - r * r, D = B * B - A * C;
    if (D < -eps) return ret;
    ret.push_back(c + a + b * (-B + sqrt(D + eps)) / A)
        ;
    if (D > eps) ret.push_back(c + a + b * (-B - sqrt(D
        )) / A);
    return ret;
}
```

CircleCircleRelation.cpp

10 lines

```
int circle_circle_relation(PT a, double r, PT b, double
     R) {
    double d = dist(a, b);
    if (sign(d - r - R) > 0)  return 5;//outside and do
        not intersect
    if (sign(d - r - R) == 0) return 4;//intersect
        outside in one point
    double l = fabs(r - R);
    if (sign(d - r - R) < 0 && sign(d - l) > 0) return
        3;//intersect in 2 points
    if (sign(d - l) == 0) return 2;//intersect inside
        in one point
    if (sign(d - l) < 0) return 1;//inside and do not
        intersect
    assert(0); return -1;
}
```

CircleCircleIntersection.cpp

12 lines

```
vector<PT> circle_circle_intersection(PT a, double r,
    PT b, double R) {
    if (a == b && sign(r - R) == 0) return {PT(1e18, 1
        e18)};
    vector<PT> ret;
    double d = sqrt(dist2(a,  b));
    if (d > r + R || d + min(r, R) < max(r, R)) return
        ret;
    double x = (d * d - R * R + r * r) / (2 * d);
    double y = sqrt(r * r - x * x);
    PT v = (b - a) / d;
    ret.push_back(a + v * x + rotateccw90(v) * y);
    if (y > 0) ret.push_back(a + v * x - rotateccw90(v)
        * y);
    return ret;
}
```

GetCircle.cpp

9 lines

```
// returns two circle c1, c2 through points a, b and of
    radius r
int get_circle(PT a, PT b, double r, circle &c1, circle
    &c2) {
    vector<PT> v = circle_circle_intersection(a, r, b,
        r);
    int t = v.size();
    if (!t) return 0; // 0 if there is no such circle
    c1.p = v[0], c1.r = r;
    if (t == 2) c2.p = v[1], c2.r = r;
    return t; // 1 if one circle, 2 if two circle
}
```

GetCircle2.cpp

28 lines

```
// returns two circle c1, c2 which is tangent to line u
    , goes through
// point q and has radius r1; 0 for no circle, 1 if c1
    = c2 , 2 if c1 != c2
int get_circle(line u, PT q, double r1, circle &c1,
    circle &c2) {
```

```
    double d = dist_from_point_to_line(u.a, u.b, q);
    if (sign(d - r1 * 2.0) > 0) return 0;
    if (sign(d) == 0) {
        cout << u.v.x << ' ' << u.v.y << '\n';
        c1.p = q + rotateccw90(u.v).truncate(r1);
        c2.p = q + rotatecw90(u.v).truncate(r1);
        c1.r = c2.r = r1;
        return 2;
    }
    line u1 = line(u.a + rotateccw90(u.v).truncate(r1),
         u.b + rotateccw90(u.v).truncate(r1));
    line u2 = line(u.a + rotatecw90(u.v).truncate(r1),
        u.b + rotatecw90(u.v).truncate(r1));
    circle cc = circle(q, r1);
    PT p1, p2; vector<PT> v;
    v = circle_line_intersection(q, r1, u1.a, u1.b);
    if (!v.size()) v = circle_line_intersection(q, r1,
        u2.a, u2.b);
    v.push_back(v[0]);
    p1 = v[0], p2 = v[1];
    c1 = circle(p1, r1);
    if (p1 == p2) {
        c2 = c1;
        return 1;
    }
    c2 = circle(p2, r1);
    return 2;
}
```

CircleCircleArea.cpp

10 lines

```
/// returns area of intersection between two circles
double circle_circle_area(PT a, double r1, PT b, double
     r2) {
    double d = (a - b).norm();
    if (r1 + r2 < d + eps) return 0;
    if (r1 + d < r2 + eps) return PI * r1 * r1;
    if (r2 + d < r1 + eps) return PI * r2 * r2;
    double theta_1 = acos((r1 * r1 + d * d - r2 * r2) /
         (2 * r1 * d)),
           theta_2 = acos((r2 * r2 + d * d - r1 * r1) /
                (2 * r2 * d));
    return r1 * r1 * (theta_1 - sin(2 * theta_1) / 2.)
        + r2 * r2 * (theta_2 - sin(2 * theta_2) / 2.);
}
```

TangentLinesFromPoint.cpp

16 lines

```
// tangent lines from point q to the circle
int tangent_lines_from_point(PT p, double r, PT q, line
     &u, line &v) {
    int x = sign(dist2(p, q) - r * r);
    if (x < 0) return 0; // point in cricle
    if (x == 0) { // point on circle
        u = line(q, q + rotateccw90(q - p));
        v = u;
        return 1;
    }
    double d = dist(p, q);
    double l = r * r / d;
    double h = sqrt(r * r - l * l);
    u = line(q, p + ((q - p).truncate(l) + (rotateccw90
        (q - p).truncate(h))));
    v = line(q, p + ((q - p).truncate(l) + (rotatecw90(
        q - p).truncate(h))));
    return 2;
}
```

TangentsLinesFromCircle.cpp

17 lines

```
int tangents_lines_from_circle(PT c1, double r1, PT c2,
     double r2, bool inner, line &u, line &v) {
    if (inner) r2 = -r2; // returns outer tangents line
         of two circles
    PT d = c2 - c1; // if inner == 1 it returns inner
        tangent lines
    double dr = r1 - r2, d2 = d.norm(), h2 = d2 - dr *
        dr;
    if (d2 == 0 || h2 < 0) {
        assert(h2 != 0);
        return 0;
    }
    vector<pair<PT, PT>>out;
    for (int tmp : { -1, 1}) {
        PT v = (d * dr + rotateccw90(d) * sqrt(h2) *
            tmp) / d2;
        out.push_back({c1 + v * r1, c2 + v * r2});
    }
    u = line(out[0].first, out[0].second);
    if (out.size() == 2) v = line(out[1].first, out[1].
        second);
    return 1 + (h2 > 0);
}
```

CircleUnion.cpp

98 lines

```
struct CircleUnion { /// OK //O(n^2 log n)
    int n;
    double x[2020], y[2020], r[2020];
    int covered[2020];
    vector<pair<double, double> > seg, cover;
    double arc, pol;
    inline int sign(double x) {return x < -eps ? -1 : x
         > eps;}
    inline int sign(double x, double y) {return sign(x
        - y);}
    inline double SQ(const double x) {return x * x;}
    inline double dist(double x1, double y1, double x2,
         double y2) {return sqrt(SQ(x1 - x2) + SQ(y1 -
        y2));}
    inline double angle(double A, double B, double C) {
        double val = (SQ(A) + SQ(B) - SQ(C)) / (2 * A *
             B);
        if (val < -1) val = -1;
        if (val > +1) val = +1;
        return acos(val);
    }
    CircleUnion() {
        n = 0;
        seg.clear(), cover.clear();
        arc = pol = 0;
    }
    void init() {
        n = 0;
        seg.clear(), cover.clear();
        arc = pol = 0;
    }
    void add(double xx, double yy, double rr) {
        x[n] = xx, y[n] = yy, r[n] = rr, covered[n] =
            0, n++;
    }
    void getarea(int i, double lef, double rig) {
        arc += 0.5 * r[i] * r[i] * (rig - lef - sin(rig
            - lef));
        double x1 = x[i] + r[i] * cos(lef), y1 = y[i] +
             r[i] * sin(lef);
        double x2 = x[i] + r[i] * cos(rig), y2 = y[i] +
             r[i] * sin(rig);
        pol += x1 * y2 - x2 * y1;
    }
    double circle_solve() {
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < i; j++) {
                if (!sign(x[i] - x[j]) && !sign(y[i] -
                    y[j]) && !sign(r[i] - r[j])) {
                    r[i] = 0.0;
                    break;
                }
            }
        }
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                if (i != j && sign(r[j] - r[i]) >= 0 &&
                     sign(dist(x[i], y[i], x[j], y[j])
                    - (r[j] - r[i])) <= 0) {
                    covered[i] = 1;
                    break;
                }
            }
        }
        for (int i = 0; i < n; i++) {
            if (sign(r[i]) && !covered[i]) {
                seg.clear();
                for (int j = 0; j < n; j++) {
                    if (i != j) {
                        double d = dist(x[i], y[i], x[j
                            ], y[j]);
                        if (sign(d - (r[j] + r[i])) >=
                            0 || sign(d - abs(r[j] - r[
                            i])) <= 0) {
                            continue;
                        }
                        double alpha = atan2(y[j] - y[i
                            ], x[j] - x[i]);
                        double beta = angle(r[i], d, r[
                            j]);
```

```
                    pair<double, double> tmp(alpha
                        - beta, alpha + beta);
                    if (sign(tmp.first) <= 0 &&
                        sign(tmp.second) <= 0) {
                        seg.push_back(pair<double,
                            double>(2 * PI + tmp.
                            first, 2 * PI + tmp.
                            second));
                    }
                    else if (sign(tmp.first) < 0) {
                        seg.push_back(pair<double,
                            double>(2 * PI + tmp.
                            first, 2 * PI));
                        seg.push_back(pair<double,
                            double>(0, tmp.second))
                            ;
                    }
                    else {
                        seg.push_back(tmp);
                    }
                }
            }
            sort(seg.begin(), seg.end());
            double rig = 0;
            for (vector<pair<double, double> >::
                iterator iter = seg.begin(); iter
                != seg.end(); iter++) {
                if (sign(rig - iter->first) >= 0) {
                    rig = max(rig, iter->second);
                }
                else {
                    getarea(i, rig, iter->first);
                    rig = iter->second;
                }
            }
            if (!sign(rig)) {
                arc += r[i] * r[i] * PI;
            }
            else {
                getarea(i, rig, 2 * PI);
            }
        }
    }
    return pol / 2.0 + arc;
  }
} CU;
```

7.4 Polygons

AreaOfTriangle.cpp

3 lines

```
double area_of_triangle(PT a, PT b, PT c) {
    return fabs(cross(b - a, c - a) * 0.5);
}
```

IsPointInTriangle.cpp

10 lines

```
// -1 if strictly inside, 0 if on the polygon, 1 if
    strictly outside
int is_point_in_triangle(PT a, PT b, PT c, PT p) {
    if (sign(cross(b - a, c - a)) < 0) swap(b, c);
    int c1 = sign(cross(b - a, p - a));
    int c2 = sign(cross(c - b, p - b));
    int c3 = sign(cross(a - c, p - c));
    if (c1 < 0 || c2 < 0 || c3 < 0) return 1; // if
        strictly outside
    if (c1 + c2 + c3 != 3) return 0; // if on the
        polygon
    return -1; // if strictly inside
}
```

Perimeter.cpp

5 lines

```
double perimeter(vector<PT> &p) {
    double ans = 0; int n = p.size();
    for (int i = 0; i < n; i++) ans += dist(p[i], p[(i
        + 1) % n]);
    return ans;
}
```

Area.cpp

5 lines

```
double area(vector<PT> &p) {
    double ans = 0; int n = p.size();
    for (int i = 0; i < n; i++) ans += cross(p[i], p[(i
        + 1) % n]);
    return fabs(ans) * 0.5;
}
```

Centroid.cpp

15 lines

```
// centroid of a (possibly non-convex) polygon,
// assuming that the coordinates are listed in a
    clockwise or
// counterclockwise fashion. Note that the centroid is
    often known as
// the "center of gravity" or "center of mass".
PT centroid(vector<PT> &p) {
    int n = p.size(); PT c(0, 0);
    double sum = 0;
    for (int i = 0; i < n; i++) sum += cross(p[i], p[(i
        + 1) % n]);
    double scale = 3.0 * sum;
    for (int i = 0; i < n; i++) {
        int j = (i + 1) % n;
        c = c + (p[i] + p[j]) * cross(p[i], p[j]);
    }
    return c / scale;
}
```

GetDirection.cpp

6 lines

```
bool get_direction(vector<PT> &p) {
    double ans = 0; int n = p.size();
    for (int i = 0; i < n; i++) ans += cross(p[i], p[(i
        + 1) % n]);
    if (sign(ans) > 0) return 1; // 1 if ccw
    return 0; // 0 if cw
}
```

GeoMetricMedian.cpp

32 lines

```
// it returns a point such that the sum of distances
// from that point to all points in p  is minimum // O(
    n log^2 MX)
PT geometric_median(vector<PT> p) {
    auto tot_dist = [&](PT z) {
        double res = 0;
        for (int i = 0; i < p.size(); i++) res += dist(
            p[i], z);
        return res;
    };
    auto findY = [&](double x) {
        double yl = -1e5, yr = 1e5;
        for (int i = 0; i < 60; i++) {
            double ym1 = yl + (yr - yl) / 3;
            double ym2 = yr - (yr - yl) / 3;
            double d1 = tot_dist(PT(x, ym1));
            double d2 = tot_dist(PT(x, ym2));
            if (d1 < d2) yr = ym2;
            else yl = ym1;
        }
        return pair<double, double> (yl, tot_dist(PT(x,
             yl)));
    };
    double xl = -1e5, xr = 1e5;
    for (int i = 0; i < 60; i++) {
        double xm1 = xl + (xr - xl) / 3;
        double xm2 = xr - (xr - xl) / 3;
        double y1, d1, y2, d2;
        auto z = findY(xm1); y1 = z.first; d1 = z.
            second;
        z = findY(xm2); y2 = z.first; d2 = z.second;
        if (d1 < d2) xr = xm2;
        else xl = xm1;
    }
    return {xl, findY(xl).first };
}
```

ConvexHull.cpp

25 lines

```
vector<PT> convex_hull(vector<PT> p) {
    if (p.size() <= 1) return p;
    vector<PT> v = p;
    sort(v.begin(), v.end());
    vector<PT> up, dn;
    for (auto& p : v) {
        while (up.size() > 1 && orientation(up[up.size
            () - 2], up.back(), p) >= 0) {
            up.pop_back();
        }
        while (dn.size() > 1 && orientation(dn[dn.size
            () - 2], dn.back(), p) <= 0) {
            dn.pop_back();
        }
        up.push_back(p);
        dn.push_back(p);
```

```
    }
    v = dn;
    if (v.size() > 1) v.pop_back();
    reverse(up.begin(), up.end());
    up.pop_back();
    for (auto& p : up) {
        v.push_back(p);
    }
    if (v.size() == 2 && v[0] == v[1]) v.pop_back();
    return v;
}
```

IsConvex.cpp

11 lines

```
bool is_convex(vector<PT> &p) { //checks if convex or
    not
    bool s[3]; s[0] = s[1] = s[2] = 0;
    int n = p.size();
    for (int i = 0; i < n; i++) {
        int j = (i + 1) % n;
        int k = (j + 1) % n;
        s[sign(cross(p[j] - p[i], p[k] - p[i])) + 1] =
            1;
        if (s[0] && s[2]) return 0;
    }
    return 1;
}
```

IsPointInConvex.cpp

17 lines

```
// it must be strictly convex, otherwise make it
    strictly convex first
int is_point_in_convex(vector<PT> &p, const PT& x) { //
     O(log n)
    int n = p.size(); assert(n >= 3);
    int a = orientation(p[0], p[1], x), b = orientation
        (p[0], p[n - 1], x);
    if (a < 0 || b > 0) return 1; // 1 if strictly
        outside
    int l = 1, r = n - 1;
    while (l + 1 < r) {
        int mid = l + r >> 1;
        if (orientation(p[0], p[mid], x) >= 0) l = mid;
        else r = mid;
    }
    int k = orientation(p[l], p[r], x);
    if (k <= 0) return -k;
    if (l == 1 && a == 0) return 0; // 0 if on the
        polygon
    if (r == n - 1 && b == 0) return 0; // if on the
        polygon
    return -1;// -1 if strictly inside
}
```

IsPointOnPolygon.cpp

7 lines

```
bool is_point_on_polygon(vector<PT> &p, const PT& z) {
    int n = p.size();
    for (int i = 0; i < n; i++) {
        if (is_point_on_seg(p[i], p[(i + 1) % n], z))
            return 1;
    }
    return 0;
}
```

IsPointInPolygon.cpp

20 lines

```
// returns 1e9 if the point is on the polygon
int winding_number(vector<PT> &p, const PT& z) { // O(n
    )
    if (is_point_on_polygon(p, z)) return 1e9;
    int n = p.size(), ans = 0;
    for (int i = 0; i < n; ++i) {
        int j = (i + 1) % n;
        bool below = p[i].y < z.y;
        if (below != (p[j].y < z.y)) {
            auto orient = orientation(z, p[j], p[i]);
            if (orient == 0) return 0;
            if (below == (orient > 0)) ans += below ? 1
                : -1;
        }
    }
    return ans;
}
// -1 if strictly inside, 0 if on the polygon, 1 if
    strictly outside
int is_point_in_polygon(vector<PT> &p, const PT& z) {
    // O(n)
    int k = winding_number(p, z);
    return k == 1e9 ? 0 : k == 0 ? 1 : -1;
}
```

ExtremeVertex.cpp

26 lines

```
// id of the vertex having maximum dot product with z
// polygon must need to be convex
// top-upper right vertex
// for minimum dot prouct negate z and return -dot(z, p
    [id])
int extreme_vertex(vector<PT> &p, const PT &z, const
    int top) { // O(log n) /// not tested
    int n = p.size();
    if (n == 1) return 0;
    double ans = dot(p[0], z); int id = 0;
    if (dot(p[top], z) > ans) ans = dot(p[top], z), id
        = top;
    int l = 1, r = top - 1;
    while (l < r) {
        int mid = l + r >> 1;
        if (dot(p[mid + 1], z) >= dot(p[mid], z)) l =
            mid + 1;
        else r = mid;
    }
    if (dot(p[l], z) > ans) ans = dot(p[l], z), id = l;
    l = top + 1, r = n - 1;
    while (l < r) {
        int mid = l + r >> 1;
        if (dot(p[(mid + 1) % n], z) >= dot(p[mid], z))
            l = mid + 1;
        else r = mid;
    }
    l %= n;
    if (dot(p[l], z) > ans) ans = dot(p[l], z), id = l;
    return id;
}
```

Diameter.cpp

16 lines

```
double diameter(vector<PT> &p) {
    int n = (int)p.size();
    if (n == 1) return 0;
    if (n == 2) return dist(p[0], p[1]);
    double ans = 0;
    int i = 0, j = 1;
    while (i < n) {
        while (cross(p[(i + 1) % n] - p[i], p[(j + 1) %
            n] - p[j]) >= 0) {
            ans = max(ans, dist2(p[i], p[j]));
            j = (j + 1) % n;
        }
        ans = max(ans, dist2(p[i], p[j]));
        i++;
    }
    return sqrt(ans);
}
```

Width.cpp

12 lines

```
double width(vector<PT> &p) {
    int n = (int)p.size();
    if (n <= 2) return 0;
    double ans = inf;
    int i = 0, j = 1;
    while (i < n) {
        while (cross(p[(i + 1) % n] - p[i], p[(j + 1) %
            n] - p[j]) >= 0) j = (j + 1) % n;
        ans = min(ans, dist_from_point_to_line(p[i], p
            [(i + 1) % n], p[j]));
        i++;
    }
    return ans;
}
```

MinimumEnclosingRectangle.cpp

23 lines

```
// minimum perimeter
double minimum_enclosing_rectangle(vector<PT> &p) {
    int n = p.size();
    if (n <= 2) return perimeter(p);
    int mndot = 0; double tmp = dot(p[1] - p[0], p[0]);
    for (int i = 1; i < n; i++) {
        if (dot(p[1] - p[0], p[i]) <= tmp) {
            tmp = dot(p[1] - p[0], p[i]);
            mndot = i;
        }
    }
```

```
    double ans = inf;
    int i = 0, j = 1, mxdot = 1;
    while (i < n) {
        PT cur = p[(i + 1) % n] - p[i];
        while (cross(cur, p[(j + 1) % n] - p[j]) >= 0)
            j = (j + 1) % n;
        while (dot(p[(mxdot + 1) % n], cur) >= dot(p[
            mxdot], cur)) mxdot = (mxdot + 1) % n;
        while (dot(p[(mndot + 1) % n], cur) <= dot(p[
            mndot], cur)) mndot = (mndot + 1) % n;
        ans = min(ans, 2.0 * ((dot(p[mxdot], cur) / cur
            .norm() - dot(p[mndot], cur) / cur.norm())
            + dist_from_point_to_line(p[i], p[(i + 1) %
             n], p[j])));
        i++;
    }
    return ans;
}
```

MinimumEnclosingCircle.cpp

24 lines

```
// given n points, find the minimum enclosing circle of
     the points
// call convex_hull() before this for faster solution
// expected O(n)
circle minimum_enclosing_circle(vector<PT> p) { ///
    vector<PT> &p
    random_shuffle(p.begin(), p.end());
    int n = p.size();
    circle c(p[0], 0);
    for (int i = 1; i < n; i++) {
        if (sign(dist(c.p, p[i]) - c.r) > 0) {
            c = circle(p[i], 0);
            for (int j = 0; j < i; j++) {
                if (sign(dist(c.p, p[j]) - c.r) > 0) {
                    c = circle((p[i] + p[j]) / 2, dist(
                        p[i], p[j]) / 2);
                    for (int k = 0; k < j; k++) {
                        if (sign(dist(c.p, p[k]) - c.r)
                             > 0) {
                            c = circle(p[i], p[j], p[k
                                ]);
                        }
                    }
                }
            }
        }
    }
    return c;
}
```

CutPolygon.cpp

18 lines

```
// returns a vector with the vertices of a polygon with
     everything
// to the left of the line going from a to b cut away.
vector<PT> cut(vector<PT> &p, PT a, PT b) {
    vector<PT> ans;
    int n = (int)p.size();
    for (int i = 0; i < n; i++) {
        double c1 = cross(b - a, p[i] - a);
        double c2 = cross(b - a, p[(i + 1) % n] - a);
        if (sign(c1) >= 0) ans.push_back(p[i]);
        if (sign(c1 * c2) < 0) {
            if (!is_parallel(p[i], p[(i + 1) % n], a, b
                )) {
                PT tmp; line_line_intersection(p[i], p
                    [(i + 1) % n], a, b, tmp);
                ans.push_back(tmp);
            }
        }
    }
    return ans;
}
```

PolygonLineIntersection.cpp

29 lines

```
// not necessarily convex, boundary is included in the
    intersection
// returns total intersected length
double polygon_line_intersection(vector<PT> p, PT a, PT
    b) {
    int n = p.size();
    p.push_back(p[0]);
    line l = line(a, b);
    double ans = 0.0;
    vector< pair<double, int> > vec;
    for (int i = 0; i < n; i++) {
        int s1 = sign(cross(b - a, p[i] - a));
        int s2 = sign(cross(b - a, p[i + 1] - a));
        if (s1 == s2) continue;
        line t = line(p[i], p[i + 1]);
        PT inter = (t.v * l.c - l.v * t.c) / cross(l.v,
            t.v);
        double tmp = dot(inter, l.v);
        int f;
        if (s1 > s2) f = s1 && s2 ? 2 : 1;
        else f = s1 && s2 ? -2 : -1;
        vec.push_back(make_pair(tmp, f));
    }
    sort(vec.begin(), vec.end());
    for (int i = 0, j = 0; i + 1 < (int)vec.size(); i
        ++) {
        j += vec[i].second;
        if (j) ans += vec[i + 1].first - vec[i].first;
    }
    ans = ans / sqrt(dot(l.v, l.v));
    p.pop_back();
    return ans;
}
```

DistFromPointToPolygon.cpp

21 lines

```
// it assumes point does not lie strictly inside the
    polygon
double dist_from_point_to_polygon(vector<PT> &v, PT p)
    { // O(log n)
    int n = (int)v.size(); // minimum distance from a
        point to a convex polygon
    if (n <= 3) {
        double ans = inf;
        for (int i = 0; i < n; i++) ans = min(ans,
            dist_from_point_to_seg(v[i], v[(i + 1) % n
            ], p));
        return ans;
    }
    PT bscur, bs = angle_bisector(v[n - 1], v[0], v[1])
        ;
    int ok,  i,  pw = 1,  ans = 0,  sgncur,  sgn = sign
        (cross(bs, p - v[0]));
    while (pw <= n) pw <<= 1;
    while ((pw >>= 1)) {
        if ((i = ans + pw) < n) {
            bscur = angle_bisector(v[i - 1], v[i], v[(i
                + 1) % n]);
            sgncur = sign(cross(bscur, p - v[i]));
            ok = sign(cross(bs, bscur)) >= 0 ? (sgn >=
                0 || sgncur <= 0) : (sgn >= 0 && sgncur
                <= 0);
            if (ok) ans = i, bs = bscur, sgn = sgncur;
        }
    }
    return dist_from_point_to_seg(v[ans], v[(ans + 1) %
        n], p);
}
```

DistFromPolygonToLine.cpp

10 lines

```
// minimum distance from convex polygon p to line ab
// returns 0 is it intersects with the polygon
// top=upper right vertex
double dist_from_polygon_to_line(vector<PT> &p, PT a,
    PT b, int top) { //O(log n)
    PT orth = (b - a).perp();
    if (orientation(a, b, p[0]) > 0) orth = (a - b).
        perp();
    int id = extreme_vertex(p, orth, top);
    if (dot(p[id] - a, orth) > 0) return 0.0; //if orth
        and a are in the same half of the line, then
        poly and line intersects
    return dist_from_point_to_line(a, b, p[id]); //does
        not intersect
}
```

DistFromPolygonToPolygon.cpp

11 lines

```
// minimum distance from a convex polygon to another
    convex polygon
double dist_from_polygon_to_polygon(vector<PT> &p1,
    vector<PT> &p2) { // O(n log n)
    double ans = inf;
    for (int i = 0; i < p1.size(); i++) {
        ans = min(ans, dist_from_point_to_polygon(p2,
            p1[i]));
    }
    for (int i = 0; i < p2.size(); i++) {
```

```
        ans = min(ans, dist_from_point_to_polygon(p1,
            p2[i]));
    }
    return ans;
}
```

MaximumDistFromPolygonToPolygon.cpp

20 lines

```
// maximum distance from a convex polygon to another
    convex polygon
double maximum_dist_from_polygon_to_polygon(vector<PT>
    &u, vector<PT> &v) { //O(n)
    int n = (int)u.size(), m = (int)v.size();
    double ans = 0;
    if (n < 3 || m < 3) {
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < m; j++) ans = max(ans,
                dist2(u[i], v[j]));
        }
        return sqrt(ans);
    }
    if (u[0].x > v[0].x) swap(n, m), swap(u, v);
    int i = 0, j = 0, step = n + m + 10;
    while (j + 1 < m && v[j].x < v[j + 1].x) j++;
    while (step--) {
        if (cross(u[(i + 1) % n] - u[i], v[(j + 1) % m]
            - v[j]) >= 0) j = (j + 1) % m;
        else i = (i + 1) % n;
        ans = max(ans, dist2(u[i], v[j]));
    }
    return sqrt(ans);
}
```

TangentsFromPointToPolygon.cpp

33 lines

```
pair<PT, int> point_poly_tangent(vector<PT> &p, PT Q,
    int dir, int l, int r) {
    while (r - l > 1) {
        int mid = (l + r) >> 1;
        bool pvs = orientation(Q, p[mid], p[mid - 1])
            != -dir;
        bool nxt = orientation(Q, p[mid], p[mid + 1])
            != -dir;
        if (pvs && nxt) return {p[mid], mid};
        if (!(pvs || nxt)) {
            auto p1 = point_poly_tangent(p, Q, dir, mid
                + 1, r);
            auto p2 = point_poly_tangent(p, Q, dir, l,
                mid - 1);
            return orientation(Q, p1.first, p2.first)
                == dir ? p1 : p2;
        }
        if (!pvs) {
            if (orientation(Q, p[mid], p[l]) == dir)  r
                = mid - 1;
            else if (orientation(Q, p[l], p[r]) == dir)
                r = mid - 1;
            else l = mid + 1;
        }
        if (!nxt) {
            if (orientation(Q, p[mid], p[l]) == dir)  l
                = mid + 1;
            else if (orientation(Q, p[l], p[r]) == dir)
                r = mid - 1;
            else l = mid + 1;
        }
    }
    pair<PT, int> ret = {p[l], l};
    for (int i = l + 1; i <= r; i++) ret = orientation(
        Q, ret.first, p[i]) != dir ? make_pair(p[i], i)
         : ret;
    return ret;
}
// (cw, ccw) tangents from a point that is outside this
     convex polygon
// returns indexes of the points
pair<int, int> tangents_from_point_to_polygon(vector<PT
    > &p, PT Q) {
    int cw = point_poly_tangent(p, Q, 1, 0, (int)p.size
        () - 1).second;
    int ccw = point_poly_tangent(p, Q, -1, 0, (int)p.
        size() - 1).second;
    return make_pair(cw, ccw);
}
```

Strings (8)

AhoCorasick.cpp

66 lines

```
struct Aho_Corasick{
 int Trie[mx][27],Suffix_Link[mx];
 vector<int> Mark[mx];
 int Node;
 void Init() {
  fill(Trie[0],Trie[0]+26,-1);
  Mark[0].clear();
  Node=0;
 }
 void Insert(char ch[],int idx) {
  int len=strlen(ch);
  int cur=0;
  for(int i=0;i<len;i++){
   int val=ch[i]-'a';
   if(Trie[cur][val]==-1) {
    Trie[cur][val]=++Node;
    fill(Trie[Node],Trie[Node]+26,-1);
    Mark[Node].clear();
   }
   cur=Trie[cur][val];
  }
  Mark[cur].push_back(idx);
 }
 void Cal_Suffix_Link() {
  queue<int>q;
  Suffix_Link[0]=0;
  for(int i=0;i<26;i++){
   if(Trie[0][i]!=-1){
    q.push(Trie[0][i]);
    Suffix_Link[Trie[0][i]]=0;
   }
   else Trie[0][i]=0;
  }
  while(!q.empty()){
   int u=q.front();
   q.pop();
   for(int v: Mark[Suffix_Link[u]]){
    Mark[u].push_back(v);
   }
   for(int i=0;i<26;i++) {
    if(Trie[u][i] != -1) {
     Suffix_Link[Trie[u][i]] = Trie[Suffix_Link[u]][i];
     q.push(Trie[u][i]);
    }
    else
     Trie[u][i] = Trie[Suffix_Link[u]][i];
   }
  }
 }
}Automata;
/// Pattern Occurence Count
int cnt[mx];
void Count_Pattern(char ch[]){
 int cur=0;
 int len=strlen(ch);
 for(int i=0;i<len;i++) {
  int val=ch[i]-'a';
  cur= Automata.Trie[cur][val];
  for(int id: Automata.Mark[cur])cnt[id]++;
 }
}
// all pattern string
Automata.Insert(ch,i);
Automata.Cal_Suffix_Link();
// Text string
Count_Pattern(ch1);
```

Hashing.cpp

40 lines

```
/*backup prime 307,367,1040160883,1066517951,
1072857881,1000004249*/
struct Hash_dui{
 ll base,mod;int sz; vector<int>Rev,Forw,P;
 Hash_dui(){}
 Hash_dui(const char* s,ll b,ll m){
 sz=strlen(s),base=b,mod=m;
 Rev.resize(sz+2,0), Forw.resize(sz+2,0), P.resize(sz
     +2,1);
 for(int i=1;i<=sz;i++) P[i]=(base*P[i-1])%mod;
 for(int i=1;i<=sz;i++) Forw[i]=(Forw[i-1]*base+ (s[i
     -1]-'a'+1))%mod;
 for(int i=sz;i>=1;i--) Rev[i]=(Rev[i+1]*base+ (s[i-1]-
     'a'+1))%mod;
```

```
 }
 void Single_char_ad(char cc){
 P.push_back((P.back()*base)% mod);
 Forw.push_back((Forw.back()*base+ (cc-'a'+1))% mod);
 }
 inline int Range_Hash(int l,int r){
 int re_hash=Forw[r+1]-((ll)P[r-l+1]*Forw[l]%mod);
 if(re_hash<0)re_hash+=mod;return re_hash;
 }
 inline int Reverse_Hash(int l,int r){
 int re_hash = Rev[l+1]-((ll)P[r-l+1]*Rev[r+2]%mod);
 if(re_hash<0)re_hash+=mod; return re_hash;
 }
};
struct Hash_Main{
 Hash_dui h1,h2; Hash_Main(){}
 Hash_Main(const char* s){
 h1=Hash_dui(s,1949313259, 2091573227);
 h2=Hash_dui(s,1997293877, 2117566807);
 }
 void Char_Add(char cc){
 h1.Single_char_ad(cc); h2.Single_char_ad(cc);}
 inline ll Range_Hash(int l,int r){//0 base
  return ((ll)h1.Range_Hash(l,r)<<32) ^ h2.Range_Hash(l
      ,r);
 }
 inline ll Reverse_Hash(int l,int r){
  return ((ll)h1.Reverse_Hash(l,r)<<32) ^ h2.
      Reverse_Hash(l,r);
 }
};Hash_Main h_ek(ch);
```

Kmp.cpp

31 lines

```
vector<int> build_lps(string s){
 vector<int>tem(s.size());
 int idx=0,len=s.size();
 for(int i=1;i<len;){
 if(s[i]==s[idx]){tem[i]=idx+1;idx++;i++;}
 else{
  if(idx!=0)idx=tem[idx-1];
  else tem[i]=idx,i++;
  }
 }
 return tem;
}
void kmp(string text,string pattern){
bool f=false;int cnt=0;
vector<int>lps=build_lps(pattern);
int j=0,i=0,len1=text.size(), len2=pattern.size();
while(i<len1){
 if(text[i]==pattern[j])i++,j++;
 else{
   if(j!=0)j=lps[j-1];
   else i++;
 }
 if(j==len2){
  f=true;
  cout<<"found at: "<<(i-len2)<<endl;
  j=lps[j-1];
  cnt++;///koy bar ace sei tar jonno
  }
 }
 if(!f)cout<<"not found\n";
}
```

Manachar.cpp

19 lines

```
int oddPlen[mx],evenPlen[mx];
void Manachers(){
 int l=0,r=-1;
 for(int i=0;i<n;i++){
 int k=(i>r)?1:min(oddPlen[l+r-i],r-i+1);
 while(k<=i && i+k<n && ch[i-k]==ch[i+k])k++;
 oddPlen[i]=k--;
 if(i+k>r){l=i-k;r=i+k;}
 }
 l=0,r=-1;
 for(int i=0;i<n;i++){
 int k=(i>r)?0:min(evenPlen[l+r-i+1],r-i+1);
 while(k+1<=i && i+k<n && ch[i-k-1]==ch[i+k])k++;
 evenPlen[i]=k--;
 if(i+k>r){l=i-k-1;r=i+k;}
 }
}
for index i
oddPlen[i]*2-1,evenPlen[i]*2
```

SuffixArray.cpp

72 lines

```
int wa[mx],wb[mx],wv[mx],Ws[mx];
int sa[mx],Rank[mx],LCP[mx];
int cmp(int *r,int a,int b,int l){return r[a]==r[b] &&
    r[a+l]==r[b+l];}
void buildSA(string s,int* sa,int n,int m){
int i,j,p,*x=wa,*y=wb,*t;
for(i=0; i<m; i++)Ws[i]=0;
for(i=0; i<n; i++)Ws[x[i]=s[i]]++;
for(i=1; i<m; i++) Ws[i]+=Ws[i-1];
for(i=n-1; i>=0; i--) sa[--Ws[x[i]]]=i;
for(j=1,p=1; p<n; j<<=1,m=p){
 for(p=0,i=n-j; i<n; i++)y[p++]=i;
 for(i=0; i<n; i++) if(sa[i]>=j) y[p++]=sa[i]-j;
 for(i=0; i<n; i++) wv[i]=x[y[i]];
 for(i=0; i<m; i++) Ws[i]=0;
 for(i=0; i<n; i++) Ws[wv[i]]++;
 for(i=1; i<m; i++) Ws[i]+=Ws[i-1];
 for(i=n-1; i>=0; i--) sa[--Ws[wv[i]]]=y[i];
 for(t=x,x=y,y=t,p=1,x[sa[0]]=0,i=1; i<n; i++)
  x[sa[i]]=cmp(y,sa[i-1],sa[i],j) ? p-1 : p++;
 }
}
//Kasai's LCP algorithm (O(n))
void buildLCP(string s,int *sa,int n){
int i,j,k=0;
for(i=1; i<=n; i++) Rank[sa[i]]=i;
for(i=0; i<n; LCP[Rank[i++]]=k)
for(k?k--:0, j=sa[Rank[i]-1]; s[i+k]==s[j+k]; k++);
}
pair<int,int> Patterntern_occurence(string Text,string
    Pattern){
 int n=Text.size();
 int m=Pattern.size();
 int be=1,en=n;
 while(be<en){
 int mid = (en+be)/2;
 int ok=0;
 for(int i=0;i<m;i++){
 if(Text[i+sa[mid]]>Pattern[i]){ok=1;break;}
 if(Text[i+sa[mid]]<Pattern[i]){ok=-1;break;}
 }
 if(ok+1) en=mid;
 else be=mid+1;
 }
 bool ok = 1;
 for(int i=0;i<m;i++) if(Text[i+sa[be]]!=Pattern[i]){ok
     =0;break;}
 if(!ok) return {-1,-1};
 pair<int,int> re;
 re.first=be;
 be=1,en=n;
 while(be<en){
 int mid = (en+be)/2;
 int ok=0;
 for(int i=0;i<m;i++){
 if(Text[i+sa[mid]]>Pattern[i]){ok=1;break;}
 if(Text[i+sa[mid]]<Pattern[i]){ok=-1;break;}
 }
 if(ok>0) en=mid;
 else be=mid+1;
 }
 ok = 1;
 for(int i=0;i<m;i++) if(Text[i+sa[en]]!=Pattern[i]){ok
     =0;break;}
 if(!ok) en--;
 re.second=en;
 return re;
}
/*for LCP from index i to index j. Set ST[i][0]=LCP[i]
    in sparse table
just run a query from min(Rank[i-1],Rank[j-1])+1 to max
    (Rank[i-1],Rank[j-1])*/
int n=s.size();
buildSA(s,sa,n+1,130);
buildLCP(s,sa,n);
sa[i] 1 base index;
Rank[i] 0 base index;
LCP[i] 1 base index;
```

TreeHashvalue.cpp

61 lines

```
int N, treeID, sz[2][maxN], name[2][maxN];
vector<int> centroids[2], G[2][maxN];
map<vector<int>,int> mp;
void reset(){
```

```
    mp.clear();
    treeID = 0;
    for(int t = 0; t < 2; t++){
        centroids[t].clear();
        for(int i = 1; i <= N; i++){
            sz[t][i] = name[t][i] = 0;
            G[t][i].clear();
        }
    }
}
void dfs1(int t, int u, int p){
    sz[t][u] = 1;
    bool is_centroid = true;
    for(int v : G[t][u]){
        if(v != p){
            dfs1(t, v, u);
            sz[t][u] += sz[t][v];
            if(sz[t][v] > N/2)  is_centroid = false;
        }
    }
    if(N-sz[t][u] > N/2) is_centroid = false;
    if(is_centroid) centroids[t].push_back(u);
}
void dfs2(int t, int u, int p){
    vector<int> childNames;
    for(int v : G[t][u]){
        if(v != p){
            dfs2(t, v, u);
            childNames.push_back(name[t][v]);
        }
    }
    sort(childNames.begin(), childNames.end());
    if(!mp[childNames]) mp[childNames] = ++treeID;
    name[t][u] = mp[childNames];
}
void solve_case(){
    scanf("%d", &N);reset();
    for(int t = 0; t < 2; t++){
        for(int i = 0, a, b; i < N-1; i++){
            scanf("%d %d", &a, &b);
            G[t][a].push_back(b);
            G[t][b].push_back(a);
        }
        dfs1(t, 1, -1);
    }
    for(int root1 : centroids[0]){
        for(int root2 : centroids[1]){
            dfs2(0, root1, -1);
            dfs2(1, root2, -1);
            if(name[0][root1] == name[1][root2]){
                printf("YES\n");
                return;
            }
        }
    }
    printf("NO\n");
}
```

Contest (9)

ExtraTools.cpp

34 lines

```
ll Set(ll N, ll pos)  return N = N | (1LL << pos);
ll Reset(ll N, ll pos)  return N = N & ~(1LL << pos);
bool chk(ll N, ll pos) return (bool)(N & (1LL << pos));
__builtin_ctz(); __builtin_popcount();
/*bitset<mx>bt;
bt.set(); bt.reset();
bt.count() ; bt._Find_first() // first 1 idx
bt._Find_next() // next one bit
for (int i = bt._Find_first(); i < mx; i = bt.
    _Find_next())*/
freopen("input.txt", "r", stdin);
freopen("output.txt", "w", stdout);
ios_base::sync_with_stdio(0); cin.tie(0);
#define watch2(x,y) cout<< _LINE_ << "says:"
<<#x<<" = "<<x<<" "<<#y<<" = "<<y <<endl;
/*Linux: s.sh + gen.cpp:
for((i=1;i<=1000;++i));do

  ./generator $i >int
  ./ans < int > out1
  ./brute < int >out2
  diff out1 out2 || break
done
mt19937_64 rng(chrono::steady_clock::now().
    time_since_epoch().count());
ll my_rand(ll l, ll r) {
    return uniform_int_distribution<ll>(l, r)(rng);
}
/*#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> ordered_set;
how many numbers are smaller than a given num
 order_of_key(num)
kth value *os.find_by_order(kth) 0 base*/
```

172. $\sum_{i=1}^{n}\sum_{j=1}^{n} i\cdot j[\gcd(i,j)=1] = \sum_{i=1}^{n}\phi(i)i^2$

173. $F(n) = \sum_{i=1}^{n}\sum_{j=1}^{n}\operatorname{lcm}(i,j) = \sum_{i=1}^{n}\left(\frac{(1+\lfloor\frac{n}{i}\rfloor)(\lfloor\frac{n}{i}\rfloor)}{2}\right)^2\sum_{d|i}\mu(d)id$

174. $\gcd(\operatorname{lcm}(a,b),\operatorname{lcm}(b,c),\operatorname{lcm}(a,c)) = \operatorname{lcm}(\gcd(a,b),\gcd(b,c),\gcd(a,c))$

175. $\gcd(A_L, A_{L+1}, \ldots, A_R) = \gcd(A_L, A_{L+1}-A_L, \ldots, A_R-A_{R-1})$.

176. Given n, If $SUM = LCM(1,n)+LCM(2,n)+\ldots+LCM(n,n)$
then SUM = $\frac{n}{2}(\sum_{d|n}(\phi(d)\times d)+1$

1. $\sum_{0\le k\le n}\binom{n-k}{k} = Fib_{n+1}$
2. $\binom{n}{k} = \binom{n}{n-k}$
3. $\binom{n}{k}+\binom{n}{k+1} = \binom{n+1}{k+1}$
4. $k\binom{n}{k} = n\binom{n-1}{k-1}$
5. $\binom{n}{k} = \frac{n}{k}\binom{n-1}{k-1}$
6. $\sum_{i=0}^{n}\binom{n}{i} = 2^n$
7. $\sum_{i\ge 0}\binom{n}{2i} = 2^{n-1}$
8. $\sum_{i\ge 0}\binom{n}{2i+1} = 2^{n-1}$
9. $\sum_{i=0}^{k}(-1)^i\binom{n}{i} = (-1)^k\binom{n-1}{k}$
10. $\sum_{i=0}^{k}\binom{n+i}{i} = \sum_{i=0}^{k}\binom{n+i}{n} = \binom{n+k+1}{k}$
11. $1\binom{n}{1}+2\binom{n}{2}+3\binom{n}{3}+\ldots+n\binom{n}{n} = n2^{n-1}$
12. $1^2\binom{n}{1}+2^2\binom{n}{2}+3^2\binom{n}{3}+\ldots+n^2\binom{n}{n} = (n+n^2)2^{n-2}$
13. **Vandermonde's Identify:** $\sum_{k=0}^{r}\binom{m}{k}\binom{n}{r-k} = \binom{m+n}{r}$
14. **Hockey-Stick Identify:** $n,r\in N, n>r, \sum_{i=r}^{n}\binom{i}{r} = \binom{n+1}{r+1}$
15. $\sum_{i=0}^{k}\binom{k}{i}^2 = \binom{2k}{k}$
16. $\sum_{k=0}^{n}\binom{n}{k}\binom{n}{n-k} = \binom{2n}{n}$
17. $\sum_{k=q}^{n}\binom{n}{k}\binom{k}{q} = 2^{n-q}\binom{n}{q}$
18. $\sum_{i=0}^{n}k^i\binom{n}{i} = (k+1)^n$
19. $\sum_{i=0}^{n}\binom{2n}{i} = 2^{2n-1}+\frac{1}{2}\binom{2n}{n}$
20. $\sum_{i=1}^{n}\binom{n}{i}\binom{n-1}{i-1} = \binom{2n-1}{n-1}$
21. $\sum_{i=0}^{n}\binom{2n}{i}^2 = \frac{1}{2}\left(\binom{4n}{2n}+\binom{2n}{n}^2\right)$

163. $\gcd(a,b) = \sum_{k|a \text{ and } k|b}\phi(k)$

164. $\sum_{i=1}^{n}[\gcd(i,n)=k] = \phi\left(\frac{n}{k}\right)$

165. $\sum_{i=1}^{n}\gcd(k,n) = \sum_{d|n}d\cdot\phi\left(\frac{n}{d}\right)$

166. $\sum_{k=1}^{n}x^{\gcd(k,n)} = \sum_{d|n}x^d\cdot\phi\left(\frac{n}{d}\right)$

167. $\sum_{k=1}^{n}\frac{1}{\gcd(k,n)} = \sum_{d|n}\frac{1}{d}\cdot\phi\left(\frac{n}{d}\right) = \frac{1}{n}\sum_{d|n}d\cdot\phi(d)$

168. $\sum_{k=1}^{n}\frac{k}{\gcd(k,n)} = \frac{n}{2}\cdot\sum_{d|n}\frac{1}{d}\cdot\phi\left(\frac{n}{d}\right) = \frac{n}{2}\cdot\frac{1}{n}\cdot\sum_{d|n}d\cdot\phi(d)$

169. $\sum_{k=1}^{n}\frac{n}{\gcd(k,n)} = 2*\sum_{k=1}^{n}\frac{k}{\gcd(k,n)}-1$, for $n>1$

170. $\sum_{i=1}^{n}\sum_{j=1}^{n}[\gcd(i,j)=1] = \sum_{d=1}^{n}\mu(d)\lfloor\frac{n}{d}\rfloor^2$

171. $\sum_{i=1}^{n}\sum_{j=1}^{n}\gcd(i,j) = \sum_{d=1}^{n}\phi(d)\lfloor\frac{n}{d}\rfloor^2$