Contents

1 DP

1.1 Convex Hull Trick

```
const ll is_query=-(1LL<<62);
const ll inf=1e18L;
struct Line {
 ll m, b;
 mutable function<const Line*()> succ;
 bool operator<(const Line& rhs) const
 {
  if (rhs.b != is_query) return m < rhs.m;
  const Line* s = succ();
  if (!s) return 0;
  int x = rhs.m;
  return b - s->b < (s->m - m) * x;
 }
};
/* will maintain upper hull for maximum
 * for minimum insert line as (-m,-b) !!
 * then the ans would be -ans
*/
struct HullDynamic : public multiset<Line>
{
 bool bad(iterator y)
 {
  auto z=next(y);
  if(y==begin())
  {
   if(z==end()) return 0;
   return y->m == z->m && y->b <=z->b;
  }

  auto x=prev(y);
  if(z==end()) return y->m == x->m && y->b <= x->b;
  __int128 left=__int128(x->b - y->b)*(z->m - y->m);
  __int128 right=__int128(y->b - z->b)*(y->m -
   x->m);
  return left>=right;
 }
 void insert_line(ll m,ll b)
 {
  auto y=insert({m,b});
  y->succ=[=] { return next(y)==end()?0: &*next(y);
   };
  if(bad(y)) { erase(y); return; }
  while(next(y)!=end() && bad(next(y)))
   erase(next(y));
  while(y!=begin() && bad(prev(y))) erase(prev(y));
 }
 ll query(ll x)
 {
  auto l=*lower_bound((Line){x,is_query});
  return l.m*x+l.b;
 }
};
```

1.2 DnC

```
const ll inf=1e15L;
const int mx=1e5+5;
ll ara[mx],sum[mx];
ll dp[105][mx];
ll C(int i, int j,int k)
{
 return 1LL*k*(sum[j]-sum[i-1]);
}
void compute(int groupNo,int l,int r,int optL,int
 optR)
{
 if(l>r) return;
 int mid=(l+r)/2;

 dp[groupNo][mid]=-inf;
 int optNow=optL;

 for(int endOfLast=optL;endOfLast<=optR &&
   endOfLast<mid;endOfLast++)
 {
  ll ret=dp[groupNo-1][endOfLast]+C(endOfLast+1,mid
   ,groupNo);
  if(ret>=dp[groupNo][mid])
  {
   dp[groupNo][mid]=ret;
   optNow=endOfLast;
  }
 }
 compute(groupNo,l,mid-1,optL,optNow);
 compute(groupNo,mid+1,r,optNow,optR);
}

void solve()
{
 //your code goes here
 for(int groupNo=2;groupNo<=k;groupNo++)
 {
  compute(groupNo,1,n,1,n);
  for(int i=groupNo+1;i<=n;i++) dp[groupNo][i]=max(
   dp[groupNo][i-1],dp[groupNo][i]);
 }
 cout<<dp[k][n]<<"\n";
}
```

1.3 Li Chao Tree

```
/*
 * Li Chao Tree for maximum query, represents line
   as mx+b
 * for minimum query initialize line with (0,inf)
 * change the sign of update and query function
   too( > to <, max to min)
 * for update call update(1,n,1,{m,b})
 * for query at point x call query(1,n,1,x)
 *
 * This tree works for almost any functions
 * just change the parameters of the Line
 *
*/
const ll inf=1LL<<62;
const int mx=1e6+5;
struct Line
{
 ll m,b;
 Line(ll _m=0,ll _b=-inf) { m=_m; b=_b; }
};

ll f(Line line,int x) { return line.m*x+line.b; }

Line Tree[4*mx];

void update(int l,int r,int at,Line line)
{
 int mid=(l+r)/2;
 bool left=f(line,l)>f(Tree[at],l);
 bool middle=f(line,mid)>f(Tree[at],mid);
 if(middle) swap(Tree[at],line);
 if(l==r) return;
 if(left!=middle) update(l,mid,at*2,line);
 else update(mid+1,r,at*2+1,line);
}

ll query(int l,int r,int at,int x)
{
 ll val=f(Tree[at],x);
 if(l==r) return val;
 int mid=(l+r)/2;
 if(x<=mid) return max(val,query(l,mid,at*2,x));
 return max(val,query(mid+1,r,at*2+1,x));
}
```

1.4 SOS DP

```
/// O( N * 2^N )
///memory optimized version
for(int i = 0; i<(1<<N); ++i) F[i] = A[i];
for(int i = 0;i < N; ++i) for(int mask = 0; mask <
   (1<<N); ++mask)
{
 if(mask & (1<<i)) F[mask] += F[mask^(1<<i)];
}

/// How many pairs in ara[] such that (ara[i] &
   ara[j]) = 0
/// N --> Max number of bits of any array element
const int N = 20;
int inv = (1<<N) - 1;
int F[(1<<N) + 10];
int ara[MAX];

/// ara is 0 based
long long howManyZeroPairs(int n,int ara[]) {
    CLR(F);
    for(int i=0;i<n;i++) F[ara[i]]++;
    for(int i = 0;i < N; ++i) for(int mask = 0;
       mask < (1<<N); ++mask)
    {
        if(mask & (1<<i)) F[mask] += F[mask^(1<<i)];
    }
    long long ans = 0;
    for(int i=0;i<n;i++) ans += F[ara[i] ^ inv];
    return ans;
}

/// F[mask] = sum of A[i] given that (i&mask)=mask
for(int i = 0; i<(1<<N); ++i) F[i] = A[i];
for(int i = 0;i < N; ++i) for(int mask = (1<<N)-1;
   mask >= 0; --mask)
{
 if (!(mask & (1<<i))) F[mask] += F[mask | (1<<i)];
}

/// Number of subsequences of ara[0:n-1] such that
/// sub[0] & sub[2] & ... & sub[k-1] = 0
const int N = 20;
int inv = (1<<N) - 1;
int F[(1<<N) + 10];
int ara[MAX];
int p2[MAX]; /// p2[i] = 2^i
///0 based array
```

```cpp
int howManyZeroSubSequences(int n,int ara[]) {
    CLR(F);
    for(int i=0;i<n;i++) F[ara[i]]++;
    for(int i = 0;i < N; ++i)
        for(int mask = (1<<N)-1; mask >= 0; --mask){
            if (!(mask & (1<<i)))
                F[mask] += F[mask | (1<<i)];
        }
    int ans = 0;
    for(int mask=0;mask<(1<<N);mask++) {
        if(__builtin_popcount(mask) & 1) ans =
         ↪  sub(ans, p2[F[mask]]);
        else ans = add(ans, p2[F[mask]]);
    }
    return ans;
}

/// Number of subsequences of ara[0:n-1] such that
/// sub[0] | sub[2] | ... | sub[k-1] = Q

int F[(1<<20) + 10], ara[MAX];
int p2[MAX]; /// p2[i] = 2^i

/// ara is 0 based
int howManySubsequences(int n, int ara[], int m,
 ↪  int Q) {
    CLR(F);
    for(int i=0;i<n;i++) F[ara[i]]++;
    if(Q == 0) return sub(p2[F[0]], 1);
    for(int i = 0;i < m; ++i)
        for(int mask = 0; mask < (1<<m); ++mask){
            if (mask & (1<<i))
                F[mask] += F[mask ^ (1<<i)];
        }
    int ans = 0;
    for(int mask=0;mask<(1<<m);mask++) {
        if(mask & Q != mask) continue;
        if(__builtin_popcount(mask ^ Q) & 1) ans =
         ↪  sub(ans, p2[F[mask]]);
        else ans = add(ans, p2[F[mask]]);
    }
    return ans;
}
```

2 Data Structures

2.1 2D fenwick tree

```cpp
struct FenwickTree2D {
    vector<vector<int>> bit;
    int n, m;

    // init(...) { ... }

    int sum(int x, int y) {
        int ret = 0;
        for (int i = x; i >= 0; i = (i & (i + 1)) -
         ↪  1)
            for (int j = y; j >= 0; j = (j & (j +
             ↪  1)) - 1)
                ret += bit[i][j];
        return ret;
    }

    void add(int x, int y, int delta) {
        for (int i = x; i < n; i = i | (i + 1))
            for (int j = y; j < m; j = j | (j + 1))
                bit[i][j] += delta;
    }
}
```

2.2 Fenwick_tree

```cpp
int bit[1000], arra[1000];
int n;
void update( int idx,int val )
{
    for( int i = idx; i <= n; i += i&(-i) ) bit[i]
     ↪  += val;
    return;
}
int query( int idx )
{
    int sum = 0;
    for( int i = idx; i > 0; i -= i&(-i) ) sum +=
     ↪  bit[i];
    return sum;
}
```

2.3 GP Hash Table

```cpp
// policy based hash table. use random to reduce
 ↪  collisions and pass anti-hash tests

#include <ext/pb_ds/assoc_container.hpp>
using namespace __gnu_pbds;

const int RANDOM = chrono::high_resolution_clock::n
 ↪  ow().time_since_epoch().count();
struct chash {
    int operator()(int x) const { return x ^
     ↪  RANDOM; }
};
gp_hash_table<int, int, chash>table;

struct chash {
    int operator()(pii x) const { return x.first*
     ↪  31 + x.second; }
};
gp_hash_table<pii, int, chash> table; //for pairs
```

2.4 LIS

```cpp
//define INF
int lis(vector<int>a){
 int n = a.size();
 vector<int>d(n+1, INF);
 d[0] = -INF;
 for(int i=0; i<n; i++){
  int j = upper_bound(d.begin(), d.end(), a[i]) -
   ↪  d.begin();
  if(d[j-1] < a[i] and a[i] < d[j]) d[j] = a[i];
 }
 int ret = 0;
 for(int i=1; i<=n; i++){
  if(d[i] < INF) ret = i;
 }
 return ret;
}
```

2.5 MO_s Algo

```cpp
const int mx=
const int sz=
struct query
{
    int l,r,id;
    bool operator<(const query &a) const
    {
  int x=l/sz; int y=a.l/sz;
  if(x!=y) return x<y;
  if(x%2) return r<a.r;
  return r>a.r;
    }
} ques[mx];
void add(int indx) { }
void baad(int indx) { }
void solve()
{
 //write code here
 int l=0;
 int r=-1;
 sort(ques+1,ques+q+1);
 for(int i=1;i<=q;i++)
 {
  while(l>ques[i].l) add(--l);
  while(r<ques[i].r) add(++r);
  while(l<ques[i].l) baad(l++);
  while(r>ques[i].r) baad(r--);
  ans[ques[i].id]=sum[now];
 }
 for(int i=1;i<=q;i++) cout<<ans[i]<<" ";
}
```

2.6 PBDS

```cpp
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
template<class T>
using ordered_set= tree<T, null_type,
  less<T>,
  rb_tree_tag, tree_order_statistics_node_update> ;

/*
PBDS
-------------------------------------------------------
   0 based indexing
-------------------------------------------------------
 ↪
 ↪
1) insert(value)
2) erase(value)
3) order_of_key(value) // Number of items strictly
 ↪  smaller than value
4) *find_by_order(k) : K-th element in a set
 ↪  (counting from zero)
*/
```

2.7 Persistent Segment Tree(pointer)

```
struct node
{
 ll sum;
 node *left,*right;
 node(ll _sum=0)
 {
  sum=_sum;
  left=right=NULL;
 }
 void build(int l,int r)
 {
  if(l==r) return;
  left=new node();
  right=new node();
  int mid=(l+r)/2;
  left->build(l,mid);
  right->build(mid+1,r);
 }
 node *update(int l,int r,int i,int x)
 {
  if(l>i || r<i) return this;
  if(l==r) return new node(x);

  int mid=(l+r)/2;

  node *ret=new node();
  ret->left=left->update(l,mid,i,x);
  ret->right=right->update(mid+1,r,i,x);

  ret->sum=ret->left->sum+ret->right->sum;

  return ret;
 }

 ll query(int tL,int tR,int rL,int rR)
 {
  if(tL>rR || tR<rL) return 0;
  if(tL>=rL && tR<=rR) return sum;

  int mid=(tL+tR)/2;

  ll a=left->query(tL,mid,rL,rR);
  ll b=right->query(mid+1,tR,rL,rR);

  return a+b;
 }

 int size() { return sizeof(*this)/sizeof(node*); }
};

const int mx=2e5+5;

node *root[mx];

void solve()
{
 int n,q;
 cin>>n>>q;

 root[0]=new node();
 root[0]->build(1,n);

 for(int i=1;i<=n;i++)
 {
  int x;
  cin>>x;

  root[0]=root[0]->update(1,n,i,x);
 }

 int sz=0;

 while(q--)
 {
  int op;
  cin>>op;

  if(op==1)
  {
   int version,i,x;
   cin>>version>>i>>x;

   version--;
   root[version]=root[version]->update(1,n,i,x);
  }
  else if(op==2)
  {
   int version,l,r;
   cin>>version>>l>>r;
   version--;
   cout<<root[version]->query(1,n,l,r)<<"\n";
  }
  else
  {
   int version;
   cin>>version;
   version--;
   root[++sz]=root[version];
  }
 }
}
```

2.8 RMQ

```
template<typename T>
struct sparse_table
{
 vector<T>ara;
 vector<int>logs;
 vector<vector<T>>table;
 sparse_table(int n)
 {
  ara.resize(n+1);
  logs.resize(n+1);
 }
 T func(T a,T b) { }
 void build(int n)
 {
  logs[1]=0;
  for(int i=2;i<=n;i++) logs[i]=logs[i/2]+1;
  table.resize(n+1,vector<T>(logs[n]+1));
  for(int i=1;i<=n;i++) table[i][0]=ara[i];
  for(int j=1;j<=logs[n];j++)
  {
   int sz=1<<j;
   for(int i=1;i+sz-1<=n;i++)
   {
    table[i][j]=func(table[i][j-1],table[i+sz/2][j-
     1]);
   }
  }
 }
 T query(int l,int r)
 {
  int d=logs[r-l+1];
  return func(table[l][d],table[r-(1<<d)+1][d]);
 }
};
```

2.9 Segment Tree (Inz)

```
class SegmentTree {
    #define Lc(idx)      idx * 2
    #define Rc(idx)      idx * 2 + 1
    public:
        struct node {
            int value;
            int lazy;
            node() {
                this->value = ??;
                this->lazy = ??;
            }
        };
        vector <node> segT;
        vector <int> A;

        SegmentTree(int sz) {
            // need to clear!
            segT.resize(4 * sz + 10);
            A.resize(sz + 1);
        }

        node Merge(node L, node R) {
            node F;
            F = ??
            return F;
        }

        void Relax(int L, int R, int idx) {
            //Do something

            segT[idx].lazy = ??; //after Relaxing
        }

        void MakeSegmentTree(int L, int R, int idx)
         {
            if (L == R) {
                segT[idx].value = ??;
                return;
            }

            int M = (L + R) / 2;

            MakeSegmentTree(L, M, Lc(idx));
            MakeSegmentTree(M + 1, R, Rc(idx));
            segT[idx] = Merge(segT[Lc(idx)],
              segT[Rc(idx)]);
        }

        node RangeQuery(int L, int R, int idx, int
         l, int r) {
            Relax(L, R, idx);

            node F;

            if (L > r || R < l)     return F;
            if (L >= l && R <= r)  return segT[idx];

            int M = (L + R) / 2;

            F = Merge(RangeQuery(L, M, Lc(idx), l,
              r), RangeQuery(M + 1, R, Rc(idx),
              l, r));
```

```
            segT[idx] = Merge(segT[Lc(idx)],
             ↪  segT[Rc(idx)]); //is it useful?
            return F;
        }
        void RangeUpdate(int L, int R, int idx, int
         ↪  l, int r, int lz) {
            Relax(L, R, idx);

            if (L > r || R < l)     return;
            if (L >= l && R <= r) {
                // Do something
                segT[idx].lazy = ??;
                Relax(L, R, idx);
                return;
            }
            int M = (L + R) / 2;

            RangeUpdate(L, M, Lc(idx), l, r, lz);
            RangeUpdate(M + 1, R, Rc(idx ), l, r,
             ↪  lz);

            segT[idx] = Merge( segT[Lc(idx)],
             ↪  segT[Rc(idx)]);
        }
};
```

2.10 Segment Tree (Point Update, Range Query)

```
ll tree[4 * N];

inline ll merge(ll a, ll b){ return a + b; }

void update(int rt, int l, int r, int p, ll v){
 if(l == r) return void(tree[rt] = v);
 int m = l + r >> 1, lc = rt << 1, rc = lc | 1;
 if(p <= m) update(lc, l, m, p, v);
 else update(rc, m + 1, r, p, v);
 tree[rt] = merge(tree[lc], tree[rc]);
}

ll query(int rt, int l, int r, int b, int e){
 if(l > e or r < b or b > e) return 0;
 if(l >= b and r <= e) return tree[rt];
 int m = l + r >> 1, lc = rt << 1, rc = lc | 1;
 return merge(query(lc, l, m, b, e), query(rc, m +
  ↪  1, r, b, e));
}
```

2.11 Segment Tree (Range Update, Range Query)

```
ll tree[4 * N], lazy[4 * N];

inline ll merge(ll a, ll b) { return a + b; }

void push(int rt, int l, int r){
 if(l ^ r){
  lazy[rt << 1] += lazy[rt];
  lazy[rt << 1 | 1] += lazy[rt];
 }
 tree[rt] += (r - l + 1) * lazy[rt];
 lazy[rt] = 0;
}

void update(int rt, int l, int r, int b, int e, ll
 ↪  v){
 if(lazy[rt]) push(rt, l, r);
 if(l > e or r < b or b > e) return;
 if(l >= b and r <= e) {
  lazy[rt] += v;
  return push(rt, l, r);
 }
 int m = l + r >> 1, lc = rt << 1, rc = lc | 1;
 update(lc, l, m, b, e, v); update(rc, m + 1, r, b,
  ↪  e, v);
 tree[rt] = merge(tree[lc], tree[rc]);
}

ll query(int rt, int l, int r, int b, int e){
 if(lazy[rt]) push(rt, l, r);
 if(l > e or r < b or b > e) return 0;
 if(l >= b and r <= e) return tree[rt];
 int m = l + r >> 1, lc = rt << 1, rc = lc | 1;
 return merge(query(lc, l, m, b, e), query(rc, m +
  ↪  1, r, b, e));
}
```

2.12 Sqrt Decomposition

```
int block_size = ??;
int Block[block_size + 5];

int getBlock(int idx)
{
    return (idx + block_size - 1) / block_size;
     ↪  //for 1-base index
    return idx / block_size;
     ↪  //for 0-base index
}

int getQueryAns(int L, int R)  //0-base index
{
    int ANS = 0;
    int CL = L / block_size;
    int CR = R / block_size;

    if (CL == CR) {
        for (int i = L; i <= R; ++i)
            ANS += ArrName[i];
    }
    else {
        for (int i = L, LM = (CL + 1) * block_size
         ↪  - 1; i <= LM; ++i)
            ANS += ArrName[i];
        for (int i = CL + 1; i <= CR - 1; ++i)
            ANS += Block[i];
        for (int i = CR * block_size; i <= R; ++i)
            ANS += ArrName[i];
    }

    return ANS;
}
//Update :   Block[ idx / block_size ] += ??
```

2.13 segment tree(lazy propagation)

```
int n, q, arra[100005];
struct idk{
    int sum, prop;
} tree[300005];
void init( int node,int b,int e )
{
    if( b == e ){
        tree[node].sum = arra[b];
        return;
    }
    int left = node*2;
    int right = node*2 + 1;
    int mid = (b+e)/2;
    init(left,b,mid);
    init(right,mid+1,e);
    tree[node].sum = tree[left].sum +
     ↪  tree[right].sum;
    return;
}
void update( int node,int b,int e,int i,int j,int
 ↪  val )
{
    if( b > j || e < i ) return;
    if( b >= i && e <= j ){
        tree[node].sum += (e-b+1)*val;
        tree[node].prop += val;
        return;
    }
    int left = node*2;
    int right = node*2 + 1;
    int mid = (b+e)/2;
    update( left,b,mid,i,j,val );
    update( right,mid+1,e,i,j,val );
    tree[node].sum = tree[left].sum +
     ↪  tree[right].sum + (e-b+1)*tree[node].prop;
    return;
}
int query( int node,int b,int e,int i,int j,int
 ↪  carry )
{
    if( b > j || e < i ) return 0;
    if( b >= i && e <= j ) return tree[node].sum +
     ↪  (e-b+1)*carry;
    int left = node*2;
    int right = node*2 + 1;
    int mid = (b+e)/2;
    int p1 = query(
     ↪  left,b,mid,i,j,carry+tree[node].prop );
    int p2 = query(
     ↪  right,mid+1,e,i,j,carry+tree[node].prop );
    return p1+p2;
}
```

3 Graph

3.1 Articulation Point Detection

```
vector<int> adj[N];
bool vis[N], articulation[N];
int low[N], tin[N], taim;
void dfs(int node, int par = -1)
{
    vis[node] = 1;
    tin[node] = low[node] = taim++;
    int children = 0;
    for(int x : adj[node])
    {
```

```
        if(x==par) continue;
        if(vis[x]) low[node] =
         ↪  min(low[node],tin[x]);
        else
        {
            dfs(x,node);
            low[node] = min(low[node],low[x]);
            if(low[x]>=tin[node] && par!=-1)
            {
                articulation[node] = 1;
            }
            children++;
        }
    }
    if(children>1 and par==-1) articulation[node] =
     ↪  1;
}
int main()
{
    int n,m,i;
    while(1)
    {
        scanf("%d %d",&n,&m);
        if(!n and !m) break;
        for(i=0;i<=n;i++) adj[i].clear();
        memset(vis,0,sizeof vis);
        memset(articulation,0,sizeof articulation);
        memset(tin,-1,sizeof tin);
        memset(low,-1,sizeof low);
        taim = 0;
        while(m--)
        {
            int a,b;
            scanf("%d %d",&a,&b);
            adj[a].push_back(b);
            adj[b].push_back(a);
        }
        for(i=1;i<=n;i++)
        {
            if(!vis[i]) dfs(i);
        }
        printf("%d\n",count(articulation+1,articula⌋
         ↪  tion+1+n,1));
    }
}
```

3.2 Bridge Detection

```
vector<int> adj[N];
bool visited[N];
int low[N], tin[N], timer;
vector<pair<int,int>> bridges;
void IS_BRIDGE(int a, int b)
{
    bridges.push_back({min(a,b),max(a,b)});
}

void dfs(int v, int p = -1) {
    visited[v] = true;
    tin[v] = low[v] = timer++;
    for (int to : adj[v]) {
        if (to == p) continue;
        if (visited[to]) low[v] = min(low[v],
         ↪  tin[to]);
        else {
            dfs(to, v);
            low[v] = min(low[v], low[to]);
            if (low[to] > tin[v]) IS_BRIDGE(v, to);
        }
    }
}
int main()
{
    int n,m,i,t,tc;
    scanf("%d",&t);
    for(tc=1;tc<=t;tc++)
    {
        scanf("%d %d",&n,&m);
        for(i=0;i<=n;i++) adj[i].clear();
        timer = 0;
        memset(tin,-1,sizeof tin);
        memset(low,-1,sizeof low);
        memset(visited,0,sizeof visited);
        bridges.clear();
        while(m--)
        {
            int a,b;
            scanf("%d %d",&a,&b);
            adj[a].push_back(b);
            adj[b].push_back(a);
        }
        for(i=1;i<=n;i++)
        {
            if(!visited[i]) dfs(i);
        }
        sort(bridges.begin(),bridges.end());
    }
}
```

3.3 Centroid Decomposition

```
// Builds a centroid tree of height O(logn) in
 ↪  O(nlogn).

const int M = 2e5 + 3;
int sz[M], done[M], cpar[M], root;
vector<int>ctree[M];

void go(int u, int p=-1){
 sz[u] = 1;
 for(int v : g[u]){
  if(v == p or done[v]) continue;
  go(v, u);
  sz[u] += sz[v];
 }
}

int find_centroid(int v, int p, int n){
 for(int x : g[v]){
  if(x != p and !done[x] and sz[x] > n/2) return
   ↪  find_centroid(x, v, n);
 }
 return v;
}

void decompose(int v=0, int p=-1){
 go(v);
 int c = find_centroid(v, -1, sz[v]);
 if(p == -1) root = c;
 done[c] = 1;
 cpar[c] = p;
 if(p != -1) ctree[p].push_back(c);
 for(int x : g[c]){
  if(!done[x]) decompose(x, c);
 }
}
```

3.4 DSU On Tree

```
vector <int> G[mx]; /// adjacency list of the tree
int sub[mx]; /// subtree size of a node
int color[mx]; /// color of a node
int freq[mx];
int n;

void calcSubSize(int s,int p) {
    sub[s] = 1;
    for(int x : G[s]) {
        if(x==p) continue;
        calcSubSize(x,s);
        sub[s] += sub[x];
    }
}

void add(int s,int p,int v,int bigchild = -1) {
    freq[color[s]] += v;
    for(int x : G[s]) {
        if(x==p || x==bigchild) continue;
        add(x,s,v);
    }
}

void dfs(int s,int p,bool keep) {
    int bigChild = -1;
    for(int x : G[s]) {
        if(x==p) continue;
        if(bigChild==-1 || sub[bigChild] < sub[x] )
         ↪  bigChild = x;
    }

    for(int x : G[s]) {
        if(x==p || x==bigChild) continue;
        dfs(x,s,0);
    }

    if(bigChild!=-1) dfs(bigChild,s,1);

    add(s,p,1,bigChild);

    /// freq[c] now contains the number of nodes in
    /// the subtree of 'node' that have color  c
    /// Save the answer for the queries here

    if(keep==0) add(s,p,-1);
}

int main() {
    input color
    construct G

    calcSubSize(root,-1);
    dfs(root,-1,0);
    return 0;
}
```

3.5 Dijkstra

```
#define pii pair<long long,int>
vector<int>Edges[100005];
vector<long long>Cost[100005];
long long dis[100005];
int vis[100005];
void dijkstra( int source )
{
    priority_queue< pii,vector<pii>,greater<pii> >Q;
    Q.push( pii(0,source) );
    dis[source] = 0;
    pii q;
    while( !Q.empty() ){
        q = Q.top();
        Q.pop();
        int u = q.second;
        if( vis[u] != -1 ) continue; // *idk why*
        vis[u] = 1; // *idk why*
        for( int i = 0; i < Edges[u].size(); i++ ){
            int v = Edges[u][i];
            if( vis[v] != -1 ) continue; // *idk
            ↪ why*
            if( dis[u] + Cost[u][i] < dis[ v ] ){
                dis[ v ] = dis[u] + Cost[u][i];
                Q.push( pii( dis[v],v ) );
            }
        }
    }
    return;
}
```

3.6 Dinic

```
// O(V^2 E), solves SPOJ FASTFLOW

#include <bits/stdc++.h>

using namespace std;

typedef long long ll;

struct edge {
  int u, v;
  ll cap, flow;
  edge () {}
  edge (int u, int v, ll cap) : u(u), v(v),
  ↪ cap(cap), flow(0) {}
};

struct Dinic {
  int N;
  vector <edge> E;
  vector <vector <int>> g;
  vector <int> d, pt;

  Dinic (int N) : N(N), E(0), g(N), d(N), pt(N) {}

  void AddEdge (int u, int v, ll cap) {
    if (u ^ v) {
      E.emplace_back(u, v, cap);
      g[u].emplace_back(E.size() - 1);
      E.emplace_back(v, u, 0);
      g[v].emplace_back(E.size() - 1);
    }
  }
  bool BFS (int S, int T) {
    queue <int> q({S});
    fill(d.begin(), d.end(), N + 1);
    d[S] = 0;
    while (!q.empty()) {
      int u = q.front(); q.pop();
      if (u == T) break;
      for (int k : g[u]) {
        edge &e = E[k];
        if (e.flow < e.cap and d[e.v] > d[e.u] + 1)
        ↪ {
          d[e.v] = d[e.u] + 1;
          q.emplace(e.v);
        }
      }
    } return d[T] != N + 1;
  }

  ll DFS (int u, int T, ll flow = -1) {
    if (u == T or flow == 0) return flow;
    for (int &i = pt[u]; i < g[u].size(); ++i) {
      edge &e = E[g[u][i]];
      edge &oe = E[g[u][i] ^ 1];
      if (d[e.v] == d[e.u] + 1) {
        ll amt = e.cap - e.flow;
        if (flow != -1 and amt > flow) amt = flow;
        if (ll pushed = DFS(e.v, T, amt)) {
          e.flow += pushed;
          oe.flow -= pushed;
          return pushed;
        }
      }
    } return 0;
  }

  ll MaxFlow (int S, int T) {
    ll total = 0;
    while (BFS(S, T)) {
      fill(pt.begin(), pt.end(), 0);
      while (ll flow = DFS(S, T)) total += flow;
    }
    return total;
  }
};

int main() {
  int N, E;
  scanf("%d %d", &N, &E);
  Dinic dinic(N);
  for (int i = 0, u, v; i < E; ++i) {
    ll cap;
    scanf("%d %d %lld", &u, &v, &cap);
    dinic.AddEdge(u - 1, v - 1, cap);
    dinic.AddEdge(v - 1, u - 1, cap);
  }
  printf("%lld\n", dinic.MaxFlow(0, N - 1));
  return 0;
}
```

3.7 Heavy Light Decomposition

```
vector < int > List[ ?? ];   // Tree's Adj List ->
↪ Need to Clear??

class HeavyLightDecomposition
{
#define L_R ? ?
public:
    vector<int> ValueOfNode;
    vector<int> Position;
    vector<int> Parent;
    vector<int> Depth;
    vector<int> Heavy;
    vector<int> Head;
    int CurrentPosition = 1; // 0/1 - index based
        segmentTree ST = segmentTree( ?? ) /
        ↪ AnyQueryTree;

        HeavyLightDecomposition(int NN)
        {
            ValueOfNode.resize(NN);
            Position.resize(NN);
            Parent.resize(NN, -1);
            Depth.resize(NN, 0);
            Heavy.resize(NN, -1);
            Head.resize(NN);
        }

        int DFS(int Vertex)
        {
            int TotalSize = 1;
            int MaxChildSize = 0;

            for (int i = 0; i <
            ↪ List[Vertex].size(); ++i)
            {
                int Child = List[Vertex][i];
                if (Child != Parent[Vertex])
                {
                    Parent[Child] = Vertex;
                    Depth[Child] =
                    ↪ Depth[Vertex] + 1;
                    int ChildSize = DFS(Child);
                    TotalSize += ChildSize;
                    if (ChildSize >
                    ↪ MaxChildSize)
                    {
                        MaxChildSize =
                        ↪ ChildSize;
                        Heavy[Vertex] = Child;
                    }
                }
            }

            return TotalSize;
        }

        void TreeDecompose(int Vertex, int Hd)
        {
            Head[Vertex] = Hd;
            ST.A[CurrentPosition] =
            ↪ ValueOfNode[Vertex];
            Position[Vertex] =
            ↪ CurrentPosition++;
            if (Heavy[Vertex] != -1)
                TreeDecompose(Heavy[Vertex],
                ↪ Hd);

            for (int i = 0; i <
            ↪ List[Vertex].size(); ++i)
            {
```

```cpp
            int Child = List[Vertex][i];
            if (Child != Parent[Vertex] &&
                Child != Heavy[Vertex])
                TreeDecompose(Child, Child);
        }
    }

    void MakeQueryTree()
    { // ?? = Number of Node in Tree;
      // Build Query Data Structure
      ? ?
    }

    int Query(int NodeA, int NodeB)
    {
        int Res = 0;
        while (Head[NodeA] != Head[NodeB])
        {
            if (Depth[Head[NodeA]] >
                Depth[Head[NodeB]])
                swap(NodeA, NodeB);
            int CurrentPathResult =
                ST.rangeQuery(L_R,
                Position[Head[NodeB]],
                Position[NodeB]).Value;
            Res = ? ? (Res,
                CurrentPathResult);
            NodeB = Parent[Head[NodeB]];
        }
        if (Depth[NodeA] > Depth[NodeB])
            swap(NodeA, NodeB);
        int LastHeavyPathResult =
            ST.rangeQuery(L_R,
            Position[NodeA],
            Position[NodeB]).Value;
        Res = ? ? (Res,
            LastHeavyPathResult);
        return Res;
    }

    int Update(int NodeA, int NodeB, int X)
    {
        while (Head[NodeA] != Head[NodeB])
        {
            if (Depth[Head[NodeA]] >
                Depth[Head[NodeB]])
                swap(NodeA, NodeB);
            ST.rangeUpdate(L_R,
                Position[Head[NodeB]],
                Position[NodeB], X);
            NodeB = Parent[Head[NodeB]];
        }
        if (Depth[NodeA] > Depth[NodeB])
            swap(NodeA, NodeB);
        ST.rangeUpdate(L_R,
            Position[NodeA],
            Position[NodeB], X);
    }
};
```

3.8 Hopcroft Karp

```cpp
// Maximum biparite matching. Complexity :
   O(E*sqrt(V))
//define NIL (dummy vertex), M and INF
vector<int>g[M];
int Lmatch[M], Rmatch[M], dist[M];

bool bfs(int n){
 queue<int>q;
 for(int u=1; u<=n; u++){
  if(Lmatch[u] == NIL) dist[u] = 0, q.push(u);
  else dist[u] = INF;
 }
 dist[NIL] = INF;
 while(!q.empty()){
  int u = q.front();
  q.pop();
  if(dist[u] < dist[NIL]){
   for(int v : g[u]){
    if(dist[Rmatch[v]] == INF){
     dist[Rmatch[v]] = dist[u] + 1;
     q.push(Rmatch[v]);
    }
   }
  }
 }
 return dist[NIL] != INF;
}

bool dfs(int u){
 if(u == NIL) return true;
 for(int v : g[u]){
  if(dist[Rmatch[v]] == dist[u] + 1 and
    dfs(Rmatch[v])){
   Rmatch[v] = u;
   Lmatch[u] = v;
   return true;
  }
 }
 dist[u] = INF;
 return false;
}

int HopcroftKarp(int n, int m){
 fill(Lmatch, Lmatch+n+1, 0);
 fill(Rmatch, Rmatch+m+1, 0);
 int res = 0;
 while(bfs(n)){
  for(int u=1; u<=n; u++){
   if(Lmatch[u]==NIL and dfs(u)) res++;
  }
 }
 return res;
}
```

3.9 Hungarian

```cpp
/*returns maximum/minimum weighted bipartite
   matching. Complexity : O(N^2 * M)
flag = -1 minimizes, flag = 1 maximizes. */
#define CLR(a) memset(a, 0, sizeof a)

ll weight[N][M];
int used[M], P[M], way[M], match[M];
ll U[M], V[M], minv[M], ara[N][M];

ll hungarian(int n, int m, int flag){
 CLR(U), CLR(V), CLR(P), CLR(ara), CLR(way);
    for(int i=1; i<=n; i++){
        for(int j=1; j<=m; j++){
            ara[i][j] = -flag * weight[i][j];
        }
    }

    if(n > m) m = n;
    int a, b, d;
    ll r, w;
    for(int i=1; i<=n; i++){
        P[0] = i, b = 0;
        for(int j=0; j<=m; j++) minv[j] = INF,
            used[j] = false;
        do{
            used[b] = true;
            a = P[b], d = 0, w = INF;
            for(int j=1; j<=m; j++){
                if(!used[j]){
                    r = ara[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<=m; j++){
                if(used[j]) U[P[j]] += w, V[j] -= w;
                else minv[j] -= w;
            }
            b = d;
        }while(P[b] != 0);
        do{
            d = way[b];
            P[b] = P[d], b = d;
        }while(b != 0);
    }
    for(int j=1; j<=m; j++) match[P[j]] = j;
    return flag*V[0];
}
```

3.10 LCA(sparse table)

```cpp
vector<int>Edges[10000];
int p[10005][17], level[10005], n, lg;
bool vis[10005];
void DFS( int par,int node )
{
    vis[node] = 1;
    if( par != -1 ) level[node] = level[par] + 1;
    p[node][0] = par;
    for( int i = 1; i <= lg; i++ ){
        if( p[node][i-1] != -1 ) p[node][i] = p[
            p[node][i-1] ][i-1];
    }
    for( int i = 0; i < Edges[node].size(); i++ ){
        if( vis[ Edges[node][i] ] == 0 ) DFS(
            node,Edges[node][i] );
    }
    return;
}
int LCA( int u,int v )
{
    if( level[u] < level[v] ) swap(u,v);
    for( int i = lg; i >= 0; i-- ){
```

```
        int par = p[u][i];
        if( level[par] >= level[v] ){
            u = par;
        }
    }
    if( u == v ) return u;
    for( int i = lg; i >= 0; i-- ){
        int U = p[u][i];
        int V = p[v][i];
        if( U != V ){
            u = U;
            v = V;
        }
    }
    return p[u][0];
}
```

3.11 MCMF(SPFA)

```
/***
    * 1 BASED NODE INDEXING
    * call init at the start of every test case
    * Complexity --> E*Flow (A lot less actually,
    not sure)
    * Maximizes the flow first, then minimizes the
    cost
    * The algorithm finds a path with minimum cost
    to send one unit of flow
      and sends flow over the path as much as
    possible. Then tries to find
      another path in the residual graph.
    * SPFA Technique :
      The basic idea of SPFA is the same as
    Bellman Ford algorithm in that each
      vertex is used as a candidate to relax its
    adjacent vertices. The improvement
      over the latter is that instead of trying
    all vertices blindly, SPFA maintains
      a queue of candidate vertices and adds a
    vertex to the queue only if that vertex
      is relaxed. This process repeats until no
    more vertex can be relaxed.
      This doesn't work if there is a negative
    cycle in the graph
***/
namespace mcmf {
    using T = int;
    const T INF = 0x3f3f3f3f; // 0x3f3f3f3f or
     0x3f3f3f3f3f3f3f3fLL
    const int MAX = 204; // maximum number of nodes
    int  n, src, snk;
    T dis[MAX], mCap[MAX];
    int par[MAX], pos[MAX];
    bool vis[MAX];
    struct Edge{
        int to, rev_pos;
        T cap, cost, flow;
    };
    vector <Edge> ed[MAX];
    void init(int _n, int _src, int _snk) {
        n = _n, src = _src, snk = _snk;
        for(int i=1;i<=n;i++) ed[i].clear();
    }
    void addEdge(int u, int v, T cap, T cost) {
        Edge a = {v, (int)ed[v].size(), cap, cost,
         0};
        Edge b = {u, (int)ed[u].size(), 0, -cost,
         0};
        ed[u].pb(a);
        ed[v].pb(b);
    }
    inline bool SPFA(){
        CLR(vis);
        for(int i=1; i<=n; i++) mCap[i] = dis[i] =
         INF;
        queue <int> q;
        dis[src] = 0;
        vis[src] = true; /// src is in the queue now
        q.push(src);
        while(!q.empty()){
            int u = q.front();
            q.pop();
            vis[u] = false; /// u is not in the
             queue now
            for(int i=0; i<(int)ed[u].size(); i++) {
                Edge &e = ed[u][i];
                int v = e.to;
                if(e.cap > e.flow && dis[v] >
                 dis[u] + e.cost){
                    dis[v] = dis[u] + e.cost;
                    par[v] = u;
                    pos[v] = i;
                    mCap[v] = min(mCap[u],e.cap -
                     e.flow);
                    if(!vis[v]) {
                        vis[v] = true;
                        q.push(v);
                    }
                }
            }
        }
        return (dis[snk] != INF);
    }
    inline pair <T, T> solve() {
        T F = 0, C = 0, f;
        int u, v;
        while(SPFA()){
            u = snk;
            f = mCap[u];
            F += f;
            while(u!=src){
                v = par[u];
                ed[v][pos[u]].flow += f; // edge of
                 v-->u increases
                ed[u][ed[v][pos[u]].rev_pos].flow
                 -= f;
                u = v;
            }
            C += dis[snk] * f;
        }
        return make_pair(F,C);
    }
}
ll arr[103]  ;
int main() {
    ios::sync_with_stdio(0);
    ll i,j,k ,l , m, n ;
    cin >> n >>  m ;
    mcmf::init(n +2 , 1 , n + 2)  ;
    for(i = 1 ; i <= n ; i++){
        cin >> arr[i+1] ;
    }
    for(i = 0 ; i < m ;i++){
        ll u,v,c;
        cin >> u >> v >> c ;
        u++;
        v++;
        mcmf::addEdge(u, v , c, -arr[v]);
    }
    mcmf::addEdge(1 , 2 , mcmf::INF , -arr[2] );
    mcmf::addEdge(n+1 , n+2 , mcmf::INF , 0) ;
    mcmf::addEdge(1 , n+1 , mcmf::INF , 0) ;
    //mcmf::addEdge(1 , 2*n , mcmf::INF , 0) ;
    pair < ll , ll > ans = mcmf::solve() ;
    cout << -ans.ss << endl;
 return 0;
}
```

3.12 MCMF(normal)

```
#include<bits/extc++.h>
using T=int;
const T kInf=numeric_limits<T>::max()/4;
struct mcf_graph
{
 struct Edge { int to,from,nxt; T flow,cap,cost; };
 vector<Edge>edges;

 int n;
 vector<T>dist,pi;
 vector<int>par,graph;

 mcf_graph(int _n):
 n(_n),dist(n),pi(n,0),par(n),graph(n,-1) {}
 void _addEdge(int from,int to,T cap,T cost)
 {
  edges.push_back(Edge{to,from,graph[from],0,cap,co
   st});
  graph[from]=edges.size()-1;
 }
 void add_edge(int from,int to,T cap,T cost)
 {
  _addEdge(from,to,cap,cost);
  _addEdge(to,from,0,-cost);
 }
 bool dijkstra(int s,int t)
 {
  fill(dist.begin(),dist.end(),kInf);
  fill(par.begin(),par.end(),-1);

  __gnu_pbds::priority_queue<pair<T,int>>q;
  vector<decltype(q)::point_iterator>its(n);

  dist[s]=0; q.push({0,s});
  while(!q.empty())
  {
   int node; T d;
   tie(d,node)=q.top(); q.pop();
   if(dist[node]!=-d) continue;
   for(int i=graph[node];i>=0;)
   {
    const auto &e=edges[i];
    T now=dist[node]+pi[node]-pi[e.to]+e.cost;
    if(e.flow<e.cap && now<dist[e.to])
    {
     dist[e.to]=now;
```

```
     par[e.to]=i;
     if(its[e.to]==q.end())
     {
      its[e.to]=q.push({-dist[e.to],e.to});
     }
     else q.modify(its[e.to],{-dist[e.to],e.to});
    }
    i=e.nxt;
   }
  }
  for(int i=0;i<n;i++)
   pi[i]=min(pi[i]+dist[i],kInf);
  return par[t]!=-1;
 }
 pair<T,T> flow(int s,int t)
 {
  T flow=0,cost=0;
  while(dijkstra(s,t))
  {
   T now=kInf;
   for(int node=t;node!=s;)
   {
    int ei=par[node];
    now=min(now,edges[ei].cap-edges[ei].flow);
    node=edges[ei^1].to;
   }
   for(int node=t;node!=s;)
   {
    int ei=par[node];
    edges[ei].flow+=now;
    edges[ei^1].flow-=now;
    cost+=edges[ei].cost*now;
    node=edges[ei^1].to;
   }
   flow+=now;
  }
  return {flow,cost};
 }
};
//use add_edge(from,to,cap,cost) for adding edge
//use flow(s,t) for finding max flow and minimum
 cost
```

3.13 Max_Flow-1

```
int graph[105][105];
int rgraph[105][105];
int par[105];
int n;
int bfs( int s,int d )
{
    bool vis[105];
    memset( vis,0,sizeof(vis) );
    queue<int>Q;
    Q.push(s);
    while( !Q.empty() ){
        int q = Q.front();
        Q.pop();
        for( int i = 1; i <= n; i++ ){
            if( vis[i] == 0 && rgraph[q][i] > 0 ){
                vis[i] = 1;
                par[i] = q;
                if( i == d ) return 1;
                Q.push(i);
            }
        }
    }
    return 0;
}
int max_flow( int s,int d )
{
    int total_flow = 0;
    for( int i = 1; i <= n; i++ ){
        for( int j = 1; j <= n; j++ ) rgraph[i][j]
         = graph[i][j];
    }
    int mn;
    while( bfs( s,d ) == 1 ){
        mn = INT_MAX;
        for( int child = d; child != s; child =
         par[child] ){
            int P = par[child];
            mn = min(mn,rgraph[P][child] );
        }
        for( int child = d; child != s; child =
         par[child] ){
            int P = par[child];
            rgraph[P][child] -= mn;
            rgraph[child][P] += mn;
        }
        total_flow += mn;
    }
    return total_flow;
}
```

3.14 Maximum flow - Edmonds Karp

```
int n;
vector<vector<int>> capacity;
vector<vector<int>> adj;

int bfs(int s, int t, vector<int> &parent)
{
    fill(parent.begin(), parent.end(), -1);
    parent[s] = -2;
    queue<pair<int, int>> q;
    q.push({s, INF});

    while (!q.empty())
    {
        int cur = q.front().first;
        int flow = q.front().second;
        q.pop();

        for (int next : adj[cur])
        {
            if (parent[next] == -1 &&
             capacity[cur][next])
            {
                parent[next] = cur;
                int new_flow = min(flow,
                 capacity[cur][next]);
                if (next == t)
                    return new_flow;
                q.push({next, new_flow});
            }
        }
    }

    return 0;
}
int maxflow(int s, int t)
{
    int flow = 0;
    vector<int> parent(n);
    int new_flow;

    while (new_flow = bfs(s, t, parent))
    {
        flow += new_flow;
        int cur = t;
        while (cur != s)
        {
            int prev = parent[cur];
            capacity[prev][cur] -= new_flow;
            capacity[cur][prev] += new_flow;
            cur = prev;
        }
    }

    return flow;
}
```

3.15 Online Bridge

```
vector<int> par, dsu_2ecc, dsu_cc, dsu_cc_size;
int bridges;
int lca_iteration;
vector<int> last_visit;

void init(int n) {
    par.resize(n);
    dsu_2ecc.resize(n);
    dsu_cc.resize(n);
    dsu_cc_size.resize(n);
    lca_iteration = 0;
    last_visit.assign(n, 0);
    for (int i=0; i<n; ++i) {
        dsu_2ecc[i] = i;
        dsu_cc[i] = i;
        dsu_cc_size[i] = 1;
        par[i] = -1;
    }
    bridges = 0;
}

int find_2ecc(int v) {
    if (v == -1)
        return -1;
    return dsu_2ecc[v] == v ? v : dsu_2ecc[v] =
     find_2ecc(dsu_2ecc[v]);
}

int find_cc(int v) {
    v = find_2ecc(v);
    return dsu_cc[v] == v ? v : dsu_cc[v] =
     find_cc(dsu_cc[v]);
}

void make_root(int v) {
    v = find_2ecc(v);
    int root = v;
    int child = -1;
    while (v != -1) {
```

```
        int p = find_2ecc(par[v]);
        par[v] = child;
        dsu_cc[v] = root;
        child = v;
        v = p;
    }
    dsu_cc_size[root] = dsu_cc_size[child];
}

void merge_path (int a, int b) {
    ++lca_iteration;
    vector<int> path_a, path_b;
    int lca = -1;
    while (lca == -1) {
        if (a != -1) {
            a = find_2ecc(a);
            path_a.push_back(a);
            if (last_visit[a] == lca_iteration){
                lca = a;
                break;
                }
            last_visit[a] = lca_iteration;
            a = par[a];
        }
        if (b != -1) {
            b = find_2ecc(b);
            path_b.push_back(b);
            if (last_visit[b] == lca_iteration){
                lca = b;
                break;
                }
            last_visit[b] = lca_iteration;
            b = par[b];
        }

    }

    for (int v : path_a) {
        dsu_2ecc[v] = lca;
        if (v == lca)
            break;
        --bridges;
    }
    for (int v : path_b) {
        dsu_2ecc[v] = lca;
        if (v == lca)
            break;
        --bridges;
    }
}

void add_edge(int a, int b) {
    a = find_2ecc(a);
    b = find_2ecc(b);
    if (a == b)
        return;

    int ca = find_cc(a);
    int cb = find_cc(b);

    if (ca != cb) {
        ++bridges;
        if (dsu_cc_size[ca] > dsu_cc_size[cb]) {
            swap(a, b);
            swap(ca, cb);
        }
        make_root(a);
        par[a] = dsu_cc[a] = b;
        dsu_cc_size[cb] += dsu_cc_size[a];
    } else {
        merge_path(a, b);
    }
}
```

3.16 SCC

```
/*In a directed graph, an SCC is a connected
 ↪ component where all nodes are pairwise
 ↪ reachable.
condesation graph is the DAG built on a directed
 ↪ graph by compressing each SCC into a node.
define M */

vector<int>g[M], gr[M];
set<int>gc[M];
int vis[M], id[M], sz[M];
vector<int>order, comp, roots;

namespace SCC{

    void addEdge(int u, int v){
        g[u].push_back(v), gr[v].push_back(u);
    }

    void dfs1(int u){
        vis[u] = 1;
        for(int x : g[u]){
            if(!vis[x]) dfs1(x);
        }
        order.push_back(u);
    }

    void dfs2(int u){
        vis[u] = 1;
        comp.push_back(u);
        for(int x : gr[u]){
            if(!vis[x]) dfs2(x);
        }
    }

    void condense(int n){
        fill(vis, vis+n+1, 0);
        for(int i=1; i<=n; i++){
            if(!vis[i]) dfs1(i);
        }
        reverse(order.begin(), order.end());
        fill(vis, vis+n+1, 0);
        for(int u : order){
            if(!vis[u]){
                dfs2(u); //this part of the code
                 ↪ processes components, returns
                 ↪ them in comp
                for(int v : comp) id[v] = u;
                sz[u] = (int)comp.size();
                roots.push_back(u);
                comp.clear();
            }
        }
        fill(vis, vis+n+1, 0);
        for(int u=1; u<=n; u++){
            for(int v : g[u]){
                if(id[u] != id[v]){
                    gc[id[u]].insert(id[v]);
                }
            }
        }
    }

    void reset(int n){
        order.clear(), comp.clear(), roots.clear();
        for(int i=1; i<=n; i++){
            g[i].clear(), gr[i].clear(),
             ↪ gc[i].clear();
            id[i] = vis[i] = sz[i] = 0;
        }
    }

}
```

4 Math

4.1 Geometry

4.1.1 2D Point Line _ Segment

```
const double PI = acos(-1.0);
const double EPS = 1e-12;
/***
    u . v = |u|*|v|*cos(theta)
          = u.x*v.x + u.y*v.y
          = How much parallel they are
          = Dot product does not change if one
 ↪  vector move perpendicular to the other
    u x v = |u|*|v|*sin(theta)
          = u.x*v.y - v.x*u.y
          = How much perpendicular they are
          = Cross product does not change if one
 ↪  vector move parallel to the other
    dot(a-b,a-b) returns squared distance between
 ↪  pt a and pt b

***/
struct pt {
    double x, y;
    pt() {}
    pt(double x, double y) : x(x) , y(y) {}

    pt operator + (const pt &p) const { return pt(
     ↪ x+p.x , y+p.y ); }
    pt operator - (const pt &p) const { return pt(
     ↪ x-p.x , y-p.y ); }
    pt operator * (double c) const { return pt( x*c
     ↪ , y*c ); }
    pt operator / (double c) const { return pt( x/c
     ↪ , y/c ); }

    bool operator == (const pt &p) const { return (
     ↪ fabs( x - p.x ) < EPS && fabs( y - p.y ) <
     ↪ EPS ); }
    bool operator != (const pt &p) const { return
     ↪ !(pt(x,y) == p); }
};
ostream& operator << (ostream& os, pt p) {
    return os << "("<< p.x << "," << p.y << ")";
}
// u.v = |u|*|v|*cos(theta)
inline double dot(pt u, pt v) { return u.x*v.x +
 ↪ u.y*v.y; }
// a x b = |a|*|b|*sin(theta)
```

```cpp
inline double cross(pt u, pt v) {return u.x*v.y - u.y*v.x;}
// returns |u|
inline double norm(pt u) { return sqrt(dot(u,u)); }
// returns angle between two vectors
inline double angle(pt u,pt v) {
    double cosTheta = dot(u,v)/norm(u)/norm(v);
    return acos(max(-1.0, min(1.0, cosTheta))); // keeping cosTheta in [-1,1]
}
// returns ang radian rotated version of vector u
// ccw rotation if angle is positive else cw rotation
inline pt rotate(pt u,double ang) {
    return pt( u.x*cos(ang) - u.y*sin(ang) ,
      u.x*sin(ang) + u.y*cos(ang) );
}
// returns a vector perpendicular to v
inline pt perp(pt u) { return pt( -u.y , u.x ); }
// returns 2*area of triangle
inline double triArea2(pt a,pt b,pt c) { return cross(b-a,c-a); }
// compare function for angular sort around point P0
inline bool comp(pt P0,pt a, pt b) {
    double d = triArea2(P0, a, b);
    if(d < 0) return false;
    if(d == 0 && dot(P0-a, P0-a) > dot(P0-b, P0-b) ) return false;
    return true;
}
/***
    if line equation is, ax + by = c
    then,
        v --> direction vector of the line (b,-a)
        c --> v cross p
        p --> Any point(vector) on the line
    side(p) = ( v cross p) - c )
            = triArea2(origin,v,p)
    if side(p) is,
        positive --> p is above the line
        zero     --> p is on the line
        negative --> p is below the line
***/

struct line {
    pt v;
    double c;
    line(pt v, double c) : v(v), c(c) {}
    // From equation ax + by = c
    line(double a, double b, double c) : v({b,-a}), c(c) {}
    // From points p and q
    line(pt p, pt q) : v(q-p), c(cross(v,p)) {}
    // |v| * dist
    // dist --> distance of p from the line
    double side(pt p) { return cross(v,p)-c; }
    // better to using sqDist than dist
    double dist(pt p) { return abs(side(p)) / norm(v); }
    double sqDist(pt p) {return side(p)*side(p) / dot(v,v);}
    // perpendicular line through point p
    // 90deg ccw rotated line
    line perpThrough(pt p) { return {p, p + perp(v)}; }
    // translates a line by vector t(dx,dy)
    // every point (x,y) of previous line is translated to (x+dx,y+dy)
    line translate(pt t) {return {v, c + cross(v,t)};}
    // for every point
    // distance between previous position and current position is dist
    line shiftLeft(double dist) { return {v, c + dist*norm(v)}; }
    // projection of point p on the line
    pt projection(pt p) { return p - perp(v)*side(p)/dot(v,v); }
    // reflection of point p wrt the line
    pt reflection(pt p) { return p - perp(v)*side(p)*2.0/dot(v,v); }
};
inline bool lineLineIntersection(line l1, line l2, pt &out) {
    double d = cross(l1.v, l2.v);
    if (d == 0) return false;
    out = (l2.v*l1.c - l1.v*l2.c) / d;
    return true;
}
// interior = true for interior bisector
// interior = false for exterior bisector
inline line bisector(line l1, line l2, bool interior) {
    assert(cross(l1.v, l2.v) != 0); // l1 and l2 cannot be parallel!
    double sign = interior ? 1 : -1;
    return {l2.v/norm(l2.v) + (l1.v * sign)/norm(l1.v),
            l2.c/norm(l2.v) + (l1.c * sign)/norm(l1.v)};
}
/*** Segment ***/

/// C --> A circle which have diameter ab
/// returns true if point p is inside C or on the border of C
inline bool inDisk(pt a, pt b, pt p) { return dot(a-p, b-p) <= 0; }
/// returns true if point p is on the segment
inline bool onSegment(pt a, pt b, pt p) {
    return triArea2(a,b,p) == 0 && inDisk(a,b,p);
}
inline bool segSegIntersection(pt a,pt b,pt c,pt d,pt &out) {
    if(onSegment(a,b,c)) return out = c, true;
    if(onSegment(a,b,d)) return out = d, true;
    if(onSegment(c,d,a)) return out = a, true;
    if(onSegment(c,d,b)) return out = b, true;
    double oa = triArea2(c,d,a);
    double ob = triArea2(c,d,b);
    double oc = triArea2(a,b,c);
    double od = triArea2(a,b,d);
    if (oa*ob < 0 && oc*od < 0) {
        out = (a*ob - b*oa) / (ob-oa);
        return true;
    }
    return false;
}
// returns distance between segment ab and point p
inline double segPointDist(pt a,pt b,pt p) {
    if( norm(a-b) == 0 ) {
        line l(a,b);
        pt pr = l.projection(p);
        if(onSegment(a,b,p)) return l.dist(p);
    }
    return min(norm(a-p),norm(b-p));
}
// returns distance between segment ab and segment cd
inline double segSegDist(pt a, pt b, pt c, pt d) {
    double oa = triArea2(c,d,a);
    double ob = triArea2(c,d,b);
    double oc = triArea2(a,b,c);
    double od = triArea2(a,b,d);
    if (oa*ob < 0 && oc*od < 0) return 0; // proper intersection
    // If the segments don't intersect, the result will be minimum of these four
    return min({segPointDist(a,b,c), segPointDist(a,b,d),
                segPointDist(c,d,a), segPointDist(c,d,b)});
}
```

4.1.2 Circle Line Intersection

```cpp
struct Point {
    double x, y;
    Point(double px, double py) {
        x = px;
        y = py;
    }
    Point sub(Point p2) {
        return Point(x - p2.x, y - p2.y);
    }
    Point add(Point p2) {
        return Point(x + p2.x, y + p2.y);
    }
    double distance(Point p2) {
        return sqrt((x - p2.x)*(x - p2.x) + (y - p2.y)*(y - p2.y));
    }
    Point normal() {
        double length = sqrt(x*x + y*y);
        return Point(x/length, y/length);
    }
    Point scale(double s) {
        return Point(x*s, y*s);
    }
};
struct line   // Creates a line with equation ax + by + c = 0
{
    double a, b, c;
    line() {}
    line( Point p1,Point p2 ) {
        a = p1.y - p2.y;
        b = p2.x - p1.x;
        c = p1.x * p2.y - p2.x * p1.y;
    }
};
inline bool eq(double a, double b) {
    return fabs( a - b ) < eps;
}
struct Circle {
    double x, y, r, left,right;
```

```
    Circle () {}
    Circle(double cx, double cy, double cr) {
        x = cx;
        y = cy;
        r = cr;
        left = x - r;
        right = x + r;
    }
    pair<Point, Point> intersections(Circle c) {
        Point P0(x, y);
        Point P1(c.x, c.y);
        double d, a, h;
        d = P0.distance(P1);
        a = (r*r - c.r*c.r + d*d)/(2*d);
        h = sqrt(r*r - a*a);
        Point P2 = P1.sub(P0).scale(a/d).add(P0);
        double x3, y3, x4, y4;
        x3 = P2.x + h*(P1.y - P0.y)/d;
        y3 = P2.y - h*(P1.x - P0.x)/d;
        x4 = P2.x - h*(P1.y - P0.y)/d;
        y4 = P2.y + h*(P1.x - P0.x)/d;

        return pair<Point, Point>(Point(x3, y3),
         ↪  Point(x4, y4));
    }
};
inline double Distance( Point a, Point b ) {
    return sqrt( ( a.x - b.x ) * ( a.x - b.x ) + (
     ↪  a.y - b.y ) * ( a.y - b.y ) );
}
inline double Distance( Point P, line L ) {
    return fabs( L.a * P.x + L.b * P.y + L.c ) /
     ↪  sqrt( L.a * L.a + L.b * L.b );
}
bool intersection(Circle C,line L,Point &p1,Point
 ↪  &p2) {
    if( Distance( {C.x,C.y}, L ) > C.r + eps )
     ↪  return false;
    double a, b, c, d, x = C.x, y = C.y;
    d = C.r*C.r - x*x - y*y;
    if( eq( L.a, 0) ) {
        p1.y = p2.y = -L.c / L.b;
        a = 1;
        b = 2 * x;
        c = p1.y * p1.y - 2 * p1.y * y - d;
        d = b * b - 4 * a * c;
        d = sqrt( fabs (d) );
        p1.x = ( b + d ) / ( 2 * a );
        p2.x = ( b - d ) / ( 2 * a );
    }
    else {
        a = L.a *L.a + L.b * L.b;
        b = 2 * ( L.a * L.a * y - L.b * L.c - L.a *
         ↪  L.b * x);
        c = L.c * L.c + 2 * L.a * L.c * x - L.a *
         ↪  L.a * d;
        d = b * b - 4 * a * c;
        d = sqrt( fabs(d) );
        p1.y = ( b + d ) / ( 2 * a );
        p2.y = ( b - d ) / ( 2 * a );
        p1.x = ( -L.b * p1.y -L.c ) / L.a;
        p2.x = ( -L.b * p2.y -L.c ) / L.a;
    }
    return true;
}
```

4.1.3 Circle

```
struct circle {
    pt c;
    double r;
    circle() {}
    circle(pt c, double r) : c(c) , r(r) {}
};
/* returns circumcircle of a triangle
   the radius of circumcircle  -->  intersection
 ↪  point of the perpendicular
                                    bisectors of
 ↪  the three sides */
circle circumCircle(pt a, pt b, pt c) {
    b = b-a, c = c-a; // consider coordinates
     ↪  relative to point a
    assert(cross(b,c) != 0); // no circumcircle if
     ↪  A,B,C are co-linear
    // detecting the intersection point using the
     ↪  same technique used in line line
     ↪  intersection
    pt center = a + ( perp( b*dot(c,c) - c*dot(b,b)
     ↪  )/cross(b,c)/2 );
    return {center, norm(center-a)};
}
int sgn(double val) {
    if(val>0) return 1;
    else if(val == 0) return 0;
    else return -1;
}
/*  returns number of intersection points between a
 ↪  line and a circle
    O --> Center
    I,J --> Intersection points
    P -- > Projection of O onto line l
    IP = JP = h , OP = d */
int circleLineIntersection(circle c, line l,
 ↪  pair<pt,pt> &out) {
    double h2 = c.r*c.r - l.sqDist(c.c); // h^2
    if (h2 >= 0) { // the line touches the circle
        pt p = l.proj(c.c); // point P
        pt h = l.v*sqrt(h2)/norm(l.v); // vector
         ↪  parallel to l, of length h
        out = {p-h, p+h}; // {I,J}
    }
    return 1 + sgn(h2); // number of intersection
     ↪  points
}
/*  returns number of intersection points between
 ↪  two circles
    O_i --> Center of circle i
    I,J --> Intersection points
    P -- > Projection of O onto line IJ
    IP = JP = h , O_10_2 = d */
int circleCircleIntersection(circle c1, circle c2,
 ↪  pair<pt,pt> &out) {
    pt d = c2.c - c1.c; double d2 = dot(d,d); // d^2
    if (d2 == 0) { // concentric circle
        assert(c1.r != c2.r); // same circle
        return 0;
    }
    double pd = (d2 + c1.r*c1.r - c2.r*c2.r)/2; //
     ↪  = |O_1P| * d
    double h2 = c1.r*c1.r - pd*pd/d2; // = h^2
    if (h2 >= 0) {
        pt p = c1.c + d*pd/d2, h =
         ↪  perp(d)*sqrt(h2/d2);
        out = {p-h, p+h};
    }
    return 1 + sgn(h2);
}
/*  inner --> if true returns inner tangents

    * if the radius of c2 is 0, returns tangents
 ↪  that go through the center
      of circle c2 (value of inner is does not
 ↪  matter in this case)
    * if there are 2 tangents, it fills out with
 ↪  two pairs of points: the pairs
      of tangency points on each circle (P1; P2),
 ↪  for each of the tangents
    * if there is 1 tangent, the circles are
 ↪  tangent to each other at some point
      P, out just contains P 4 times, and the
 ↪  tangent line can be found as
      line(c1.c,p).perpThrough(p)
    * if there are 0 tangents, it does nothing
    * if the circles are identical, it aborts. */
int tangents(circle c1, circle c2, bool inner,
 ↪  vector < pair <pt,pt> > &out) {
    if (inner) c2.r = -c2.r;
    pt d = c2.c-c1.c;
    double dr = c1.r-c2.r, d2 = dot(d,d), h2 =
     ↪  d2-dr*dr;
    if (d2 == 0 || h2 < 0) {
        //assert(h2 != 0);
        return 0;
    }
    for (double sign : {-1,1}) {
        pt v = (d*dr + perp(d)*sqrt(h2)*sign)/d2;
        out.push_back({c1.c + v*c1.r, c2.c +
         ↪  v*c2.r});
    }
    return 1 + (h2 > 0);
}
```

4.1.4 Closest Point Pair

```
void solve()
{
 int n;
 cin>>n;
 vector<pair<pii,int>>vec(n);
 for(int i=0;i<n;i++)
 {
  cin>>vec[i].first.first>>vec[i].first.second;
  vec[i].second=i;
 }
 sort(vec.begin(),vec.end());
 ll ans=1e18L;
 int a=-1,b=-1;
 set<pair<pii,int>>st;
 for(int i=0,j=0;i<n;i++)
 {
  int d=ceil(sqrt(ans));
  while(j<i &&
   ↪  vec[j].first.first+d<vec[j].first.first)
  {
```

```
  st.erase({{vec[j].first.second,vec[j].first.first},vec[j].second});
  j++;
 }
 auto it1=st.lower_bound({{vec[i].first.second-d,vec[i].first.first},-1});
 auto it2=st.lower_bound({{vec[i].first.second+d,vec[i].first.first},-1});
 for(auto it=it1;it!=it2;it++)
 {
  int dx=vec[i].first.first-it->first.second;
  int dy=vec[i].first.second-it->first.first;
  ll curr=1LL*dx*dx+1LL*dy*dy;
  if(curr<ans)
  {
   ans=curr;
   a=vec[i].second;
   b=it->second;
  }
 }
 st.insert({{vec[i].first.second,vec[i].first.first},vec[i].second});
 }
 if(a>b) swap(a,b);
 cout<<a<<" "<<b<<" ";
 cout<<fixed<<setprecision(6)<<sqrt(ans)<<"\n";
}
```

4.1.5 Convex_Hull

```
#include<bits/stdc++.h>
using namespace std;

struct Point
{
 int x, y;
};
Point p0;
Point nextToTop(stack<Point> &S)
{
 Point p = S.top();
 S.pop();
 Point res = S.top();
 S.push(p);
 return res;
}

void swap(Point &p1, Point &p2)
{
 Point temp = p1;
 p1 = p2;
 p2 = temp;
}

int distSq(Point p1, Point p2)
{
 return (p1.x - p2.x)*(p1.x - p2.x) +
  (p1.y - p2.y)*(p1.y - p2.y);
}

int orientation(Point p, Point q, Point r)
{
 int val = (q.y - p.y) * (r.x - q.x) -
   (q.x - p.x) * (r.y - q.y);

 if (val == 0) return 0;
 return (val > 0)? 1: 2;
}
int compare(const void *vp1, const void *vp2)
{
Point *p1 = (Point *)vp1;
Point *p2 = (Point *)vp2;

int o = orientation(p0, *p1, *p2);
if (o == 0)
 return (distSq(p0, *p2) >= distSq(p0, *p1))? -1 : 1;

return (o == 2)? -1: 1;
}

void convexHull(Point points[], int n)
{

int ymin = points[0].y, min = 0;
for (int i = 1; i < n; i++)
{
 int y = points[i].y;

 if ((y < ymin) || (ymin == y &&
  points[i].x < points[min].x))
  ymin = points[i].y, min = i;
}

swap(points[0], points[min]);
p0 = points[0];
qsort(&points[1], n-1, sizeof(Point), compare);
int m = 1;
for (int i=1; i<n; i++)
{
 while (i < n-1 && orientation(p0, points[i],
        points[i+1]) == 0)
  i++;


 points[m] = points[i];
 m++;
}

if (m < 3) return;
stack<Point> S;
S.push(points[0]);
S.push(points[1]);
S.push(points[2]);

for (int i = 3; i < m; i++)
{
 while (S.size()>1 && orientation(nextToTop(S),
  S.top(), points[i]) != 2)
  S.pop();
 S.push(points[i]);
}

while (!S.empty())
{
 Point p = S.top();
 cout << "(" << p.x << ", " << p.y <<")" << endl;
 S.pop();
}
}

int main()
{
 Point points[100005];
   int n;
   scanf("%d",&n);
   for( int i = 0; i < n; i++ )scanf("%d %d",&points[i].x,&points[i].y);
 convexHull(points, n);
 return 0;
}
```

4.1.6 Half Plain Intersection

```
const double eps=1e-9;
template<class T>
struct Point
{
 typedef Point P;
 T x,y;
 Point(T _x=0,T _y=0) { x=_x; y=_y; }
 bool operator<(P p) const { return tie(x,y)<tie(p.x,p.y); }
 bool operator==(P p) const { return tie(x,y)==tie(p.x,p.y); }
 P operator+(P p) const { return P(x+p.x,y+p.y); }
 P operator-(P p) const { return P(x-p.x,y-p.y); }
 P operator*(T d) const { return P(x*d,y*d); }
 P operator/(T d) const { return P(x/d,y/d); }
 T dot(P p) const { return x*p.x+y*p.y; }
 T cross(P p) const { return x*p.y-y*p.x; }
 T cross(P a,P b) const { return (a-*this).cross(b-*this); }
 T dist2() const { return x*x+y*y; }
 double dist() const { return sqrt(double(dist2())); }
 double angle() const { return atan2(y,x); }
 P unit() const { return *this/dist(); }
 P perp() const { return P(-y,x); }
 P normal() const { return perp().unit(); }
 P rotate(double a) const
 {
  return P(x*cos(a)-y*sin(a),x*sin(a)+y*cos(a));
 }
};
template<class P>
int lineIntersection(const P &s1,const P &e1,const P &s2,const P &e2,P &r)
{
 if((e1-s1).cross(e2-s2)) //if not parallel
 {
  r=s2-(e2-s2)*(e1-s1).cross(s2-s1)/(e1-s1).cross(e2-s2);
  return 1;
 }
 return -((e1-s1).cross(s2-s1)==0 || s2==e2);
}
typedef Point<ld> P;
struct Line
{
 P P1,P2;
 //right hand side of the ray P1 --> P2
 Line(P a=P(), P b=P())
 {
  P1=a;
  P2=b;
 }
```

```
 P intpo(Line y)
 {
  P r;
  assert(lineIntersection(P1,P2,y.P1,y.P2,r)==1);
  return r;
 }
 P dir() { return P2-P1; }
 bool contains(P x) { return
 ↪  (P2-P1).cross(x-P1)<eps; }
 bool out(P x) { return !contains(x); }
};
template<class T>
bool mycmp(Point<T>a,Point<T>b)
{
 if(a.x*b.x<0) return a.x<0;
 if(abs(a.x)<eps)
 {
  if(abs(b.x)<eps) return a.y>0  && b.y<0;
  if(b.x<0) return a.y>0;
  if(b.x>0) return true;
 }
 if(abs(b.x)<eps)
 {
  if(a.x<0) return b.y<0;
  if(a.x>0) return false;
 }
 return a.cross(b)>0;
}
bool cmp(Line a,Line b) { return
 ↪  mycmp(a.dir(),b.dir()); }
ld Intersection_Area(vector<Line>b)
{
 sort(b.begin(),b.end(),cmp);
 int n=b.size();
 int q=1,h=0;
 vector<Line>c(b.size()+10);
 for(int i=0;i<n;i++)
 {
  while(q<h && b[i].out(c[h].intpo(c[h-1]))) h--;
  while(q<h && b[i].out(c[q].intpo(c[q+1]))) q++;
  c[++h]=b[i];
  if(q<h && abs(c[h].dir().cross(c[h-1].dir()))<eps)
  {
   h--;
   if(b[i].out(c[h].P1)) c[h]=b[i];
  }
 }
 while(q<h-1 && c[q].out(c[h].intpo(c[h-1]))) h--;
 while(q<h-1 && c[h].out(c[q].intpo(c[q+1]))) q++;
 if(h-q<=1) return 0;
 c[h+1]=c[q];
 vector<P>s;
 for(int i=q;i<=h;i++)
 ↪  s.push_back(c[i].intpo(c[i+1]));
 s.push_back(s[0]);
 ld ans=0;
 for(int i=0;i<int(s.size())-1;i++)
 ↪  ans+=s[i].cross(s[i+1]);
 ans/=2.0;
 return ans;
}
void solve()
{
 int n;
 cin>>n;
 vector<P>vec(n);
 for(int i=0;i<n;i++)
 {
  ld x,y;
  cin>>x>>y;
  vec[i]=P(x,y);
 }
 vector<Line>halfplanes;
 for(int i=0;i<n;i++)
 {
  int j=(i+1)%n;
  halfplanes.push_back(Line(vec[i],vec[j]));
 }
 ld ans=Intersection_Area(halfplanes);
 cout<<fixed<<setprecision(10)<<ans<<"\n";
}
```

4.1.7 Point Inside Poly (Ray Shooting)

```
// if strict, returns false when a is on the
 ↪  boundary
inline bool insidePoly(pt *P, int np, pt a, bool
 ↪  strict = true) {
    int numCrossings = 0;
    for (int i = 0; i < np; i++) {
        if (onSegment(P[i], P[(i+1)%np], a)) return
         ↪  !strict;
        numCrossings += crossesRay(a, P[i],
         ↪  P[(i+1)%np]);
    }
    return (numCrossings & 1); // inside if odd
     ↪  number of crossings
}
```

4.2 Matrices

4.2.1 Gauss-Jordan Elimination in GF(2)

```
const int SZ = 105;
const int MOD = 1e9 + 7;
bitset <SZ> mat[SZ];
int where[SZ];
bitset <SZ> ans;
ll bigMod(ll a,ll b,ll m){
    ll ret = 1LL;
    a %= m;
    while (b){
        if (b & 1LL) ret = (ret * a) % m;
        a = (a * a) % m;
        b >>= 1LL;
    }
    return ret;
}
/// n for row, m for column, modulo 2
int GaussJordan(int n,int m) {
    SET(where); /// sets to -1
    for(int r=0,c=0; c<m && r<n; c++) {
        for(int i=r; i<n; i++)
            if( mat[i][c] ){
                swap(mat[i],mat[r]); break;
            }
        if( !mat[r][c] ) continue;
        where[c] = r;
        for (int i=0; i<n; ++i) if (i != r &&
         ↪  mat[i][c]) mat[i] ^= mat[r];
        r++;
    }
    for(int j=0; j<m; j++) {
        if(where[j]!=-1) ans[j] =
         ↪  mat[where[j]][m]/mat[where[j]][j];
        else ans[j] = 0;
    }
    for(int i=0; i<n; i++){
        int sum = 0;
        for(int j=0; j<m; j++) sum ^=  (ans[j] &
         ↪  mat[i][j]);
        if( sum != mat[i][m] ) return 0;  /// no
         ↪  solution
    }
    int cnt = 0;
    for(int j=0; j<m; j++) if (where[j]==-1) cnt++;
    return bigMod(2,cnt,MOD); /// how many
     ↪  solutions modulo some other MOD
}
```

4.2.2 Gauss-Jordan Elimination in GF(P)

```
const int SZ = 105;
const int MOD = 1e9 + 7;
int mat[SZ][SZ], where[SZ], ans[SZ];
ll bigMod(ll a,ll b,ll m){
    ll ret = 1LL;
    a %= m;
    while (b){
        if (b & 1LL) ret = (ret * a) % m;
        a = (a * a) % m;
        b >>= 1LL;
    }
    return ret;
}
int GaussJordan(int n,int m,int P) {
    SET(where); /// sets to -1
    for(int r=0,c=0; c<m && r<n; c++) {
        int mx = r;
        for(int i=r; i<n; i++) if( mat[i][c] >
         ↪  mat[mx][c] ) mx = i;
        if( mat[mx][c] == 0 ) continue;
        if(r != mx) for(int j=c; j<=m; j++)
         ↪  swap(mat[r][j],mat[mx][j]);
        where[c] = r;
        int mul,minv = bigMod(mat[r][c],P-2,P);
        int temp;
        for(int i=0; i<n; i++){
            if( i!=r && mat[i][c]!=0){
                mul = ( mat[i][c] * (long long)
                 ↪  minv ) % P;
                for(int j=c; j<=m; j++) {
                    temp = mat[i][j];
                    temp -= ( ( mul * (long long)
                     ↪  mat[r][j] ) % P );
                    temp += P;
                    if( temp >= P ) temp -= P;
                    mat[i][j] = temp;
                }
            }
        }
        r++;
    }
```

```
    for(int j=0; j<m; j++) {
        if(where[j]!=-1) ans[j] = (
         mat[where[j]][m] * 1LL *
         bigMod(mat[where[j]][j],P-2,P) ) % P;
        else ans[j] = 0;
    }
    for(int i=0; i<n; i++){
        int sum = 0;
        for(int j=0; j<m; j++) {
            sum += ( ans[j] * 1LL * mat[i][j] ) % P;
            if(sum >= P) sum -= P;
        }
        if( sum != mat[i][m] ) return 0;  /// no
         solution
    }
    int cnt = 0;
    for(int j=0; j<m; j++) if (where[j]==-1) cnt++;
    return bigMod(P,cnt,MOD);
}
```

4.2.3 Gauss-Jordan Elimination

```
/***
    * mat is 0 based
    * In every test case, clear mat first and then
    do the changes
    * For solving problems on graphs with
    probability/expectation, make sure the graph
      is connected and a single component. If not,
    then re-number the vertex and solve
      for each connected component separately.
    * Complexity --> O( min(n,m) * nm )
***/
const int SZ = 105;
const double EPS = 1e-9;
double mat[SZ][SZ], ans[SZ];
int where[SZ];
int GaussJordan(int n,int m) {
    SET(where); /// sets to -1
    for(int r=0,c=0; c<m && r<n; c++) {
        int mx = r;
        for(int i=r; i<n; i++) if( abs(mat[i][c]) >
         abs(mat[mx][c]) ) mx = i;
        if( abs(mat[mx][c]) < EPS ) continue;
        if(r != mx) for(int j=c; j<=m; j++)
         swap(mat[r][j],mat[mx][j]);
        where[c] = r;
        for(int i=0; i<n; i++) if( i!=r )
        {
    double mul = mat[i][c]/mat[r][c];
    for(int j=c; j<=m; j++) mat[i][j] -=
     mul*mat[r][j];
        }
        r++;
    }
    for(int j=0; j<m; j++) {
        if(where[j]!=-1) ans[j] =
         mat[where[j]][m]/mat[where[j]][j];
        else ans[j] = 0;
    }
    for(int i=0; i<n; i++){
        double sum = 0;
        for(int j=0; j<m; j++) sum += ans[j] *
         mat[i][j];
        if( abs(sum - mat[i][m]) > EPS ) return 0;
         /// no solution
    }
    for(int j=0; j<m; j++) if (where[j]==-1) return
     INF;
    return 1;
}
```

4.2.4 Space of Binary Vectors

```
// A vector can be added to the space at any moment
// Following queries can be made on the current
 basis at any moment
const int B = ?;
struct space {

    int base[B];
    int sz;
    void init() {
        CLR(base); sz = 0;
    }
    // if the vector val is not currently in the
     vector space,
    // then adds val as a basis vector
    void add(int val) {
        for(int i = B-1; i >= 0; i--) {
            if( val & (1<<i) ) {
                if(!base[i]) {
                    base[i] = val; ++sz; return;
                }
                else val ^= base[i];
            }
        }
    }
    int getSize() { return sz; }
    // returns maximum possible ret such that, ret
     = (val ^ x)
    // and x is also in the vector space of the
     current basis
    int getMax(int val) {
        int ret = val;
        for(int i = B - 1; i >= 0; i--) {
            if(ret & (1<<i)) continue;
            ret ^= base[i];
        }
        return ret;
    }
    // returns minimum possible ret such that, ret
     = (val ^ x)
    // and x is also in the vector space of the
     current basis
    int getMin(int val) {
        int ret = val;
        for(int i = B - 1; i >= 0; i--) {
            if( !(ret & (1<<i)) ) continue;
            ret ^= base[i];
        }
        return ret;
    }
    // returns true if val is in the vector space
    bool possible(int val) {
        for(int i = B - 1; i >= 0; i--) {
            if(val & (1<<i)) val ^= base[i];
        }
        return (val == 0);
    }
    // returns the k'th element of the current
     vector space
    // if we sort all the elements according to
     their values
    int query(int k) {
        int ret = 0;
        int tot = 1 << getSize();
        for(int i = B - 1; i >= 0; i--) {
            if(!base[i]) continue;
            int low = tot >> 1;
            if ((low < k && (ret & 1 << i) == 0) ||
             (low >= k && (ret & 1 << i) > 0))
                ret ^= base[i];
            if (low < k) k -= low;
            tot /= 2;
        }
        return ret;
    }
};
```

4.2.5 matrix_exponentiation

```
#include<bits/stdc++.h>
using namespace std;
long long a, b, n, m, F[2][2], f[2][2];
long long p = 1e9 + 7;
void multiply( long long a[2][2],long long b[2][2])
{
    long long g[2][2];
    for( int i = 0; i < 2; i++ ){
        for( int j = 0; j < 2; j++ ){
            g[i][j] = 0; for( int k = 0; k < 2; k++
             ) g[i][j] =( (g[i][j]%p) +
             ((a[i][k]%p)*(b[k][j]%p))%p )%p;
        }
    }
    for( int i = 0; i < 2; i++ ){ for( int j = 0; j
     < 2; j++ ) F[i][j] = g[i][j];}
}
void power( long long N )
{
    if( N == 1 ) return;
    if( N%2 == 0 ){ power( N/2 ); multiply(F,F); }
    else{power(N-1); multiply(F,f);}
    return;
}
```

4.3 Modular Arithmatic

4.3.1 Chinese Remainder Theorem

```
/***
    X = a_1 % m_1
    X = a_2 % m_2
    X = a_3 % m_3

    m_1,m_2,m_3 are pair wise co-prime

    M = m_1*m_2*m_3

    u_i = Modular inverse of (M/m_i) with respect
 m_i
```

```
    X = ( a_1 * (M/m_1) * u_1 + a_2 * (M/m_2) * u_2
 ↪  + a_3 * (M/m_3) * u_3 ) % M
***/
ll inv(ll a, ll m)
{
    ll m0 = m, t, q;
    ll x0 = 0, x1 = 1;
    if (m == 1) return 0;
    while (a > 1)
    {
        q = a / m; t = m; m = a % m, a = t; t = x0;
         ↪  x0 = x1 - q * x0; x1 = t;
    }
    if (x1 < 0) x1 += m0;
    return x1;
}
ll findMinX(ll num[], ll rem[], ll k)
{
    ll prod = 1;
    for (ll i = 0; i < k; i++) prod *= num[i];
    ll result = 0;
    for (ll i = 0; i < k; i++)
    {
        ll pp = prod / num[i];
        result += rem[i] * inv(pp, num[i]) * pp;
    }
    return result % prod;
}
int main()
{
    ll num[15],rem[15],n,t,i,j;
    scanf("%lld",&t);
    for(i=1;i<=t;i++)
    {
        scanf("%lld",&n);
        for(j=0;j<n;j++)
        {
            scanf("%lld %lld",&num[j],&rem[j]);
        }
        printf("Case %lld:
         ↪  %lld\n",i,findMinX(num,rem,n));
    }
}
```

4.3.2 Discrete log

```
int solve(int a, int b, int m) {
    a %= m, b %= m;
    int k = 1, add = 0, g;
    while ((g = gcd(a, m)) > 1) {
        if (b == k)
            return add;
        if (b % g)
            return -1;
        b /= g, m /= g, ++add;
        k = (k * 1ll * a / g) % m;
    }

    int n = sqrt(m) + 1;
    int an = 1;
    for (int i = 0; i < n; ++i)
        an = (an * 1ll * a) % m;

    unordered_map<int, int> vals;
    for (int q = 0, cur = b; q <= n; ++q) {
        vals[cur] = q;
        cur = (cur * 1ll * a) % m;
    }

    for (int p = 1, cur = k; p <= n; ++p) {
        cur = (cur * 1ll * an) % m;
        if (vals.count(cur)) {
            int ans = n * p - vals[cur] + add;
            return ans;
        }
    }
    return -1;
}
```

4.3.3 Modular_inverse_EGCD

```
#include<bits/stdc++.h>
using namespace std;
int gcdExtended(int a, int b, int* x, int* y)
{
    if (a == 0)
    {
        *x = 0, *y = 1;
        return b;
    }

    int x1, y1;
    int gcd = gcdExtended(b % a, a, &x1, &y1);
    *x = y1 - (b / a) * x1;
    *y = x1;

    return gcd;
}

void modInverse(int a, int m)
{
    int x, y;
    int g = gcdExtended(a, m, &x, &y);
    if (g != 1)
        printf("Inverse doesn't exist");
    else
    {
        int res = (x % m + m) % m;
        printf("Modular multiplicative inverse is
         ↪  %d\n", res);
    }
}

int main()
{
    int a, m;
    cin >> a >> m;
    modInverse(a, m);
    return 0;
}
```

4.3.4 nCr Lucas

```
/*use this to calculate nCr modulo mod, when mod is
 ↪  smaller than n and m. define MOD
Complexity : O(mod + log mod n) */
ll fact[MOD];

ll bigmod(int x, int p){
    ll res = 1;
    while(p){
        if(p & 1) res = res * x % MOD;
        x = x * x % MOD;
        p >>= 1;
    }
    return res;
}

ll modinv(ll x){
 return bigmod(x, MOD-2);
}

void precalc(){ //run this
 fact[0] = 1;
 for(int i=1; i<MOD; i++){
  fact[i] = fact[i-1] * i % MOD;
 }
}

int C(int n, int m){
 if(m > n) return 0;
 if(m == 0 or m == n) return 1;
 ll ret = fact[n] * modinv(fact[m]) % MOD;
 return ret * modinv(fact[n-m]) % MOD;
}

int nCr(int n, int m){
 if(m > n) return 0;
 if(m == 0) return 1;
 return nCr(n/MOD, m/MOD)*C(n%MOD, m%MOD) % MOD;
}
```

4.4 Polynomial Multiplication

4.4.1 FFT

```
typedef cplx cd;

//define N as a power of two greater than the size
 ↪  of any possible polynomial

using cd = complex<double>;
const double PI = acosl(-1);

int rev[N]; cd w[N];
static cd f[N];

void prepare(int &n){
    int sz = __builtin_ctz(n);
    for(int i=1; i<n; i++) rev[i] = (rev[i>>1] >>
     ↪  1) | ((i & 1) << (sz - 1));
    w[0] = 0, w[1] = 1, sz = 1;
    while(1 << sz < n){
        cd w_n = cd(cos(2*PI / (1<<(sz+1))),
         ↪  sin(2*PI / (1 << (sz+1))));
        for(int i=1<<(sz-1); i<(1<<sz); i++){
            w[i<<1] = w[i], w[i<<1 | 1] = w[i] *
             ↪  w_n;
        }
        sz++;
    }
}

void fft(cd *a, int n){
    for(int i=1; i<n-1; i++){
        if(i < rev[i]) swap(a[i], a[rev[i]]);
    }
    for(int h=1; h<n; h<<=1){
        for(int s=0; s<n; s+=h<<1){
```

```
            for(int i=0; i<h; i++){
                cd &u = a[s+i], &v = a[s+i+h], t = v*w[h+i];
                v = u-t, u = u+t;
            }
        }
    }
}

vector<ll>multiply(vector<ll>a, vector<ll>b){
    int n = a.size(), m = b.size(), sz = 1;
    if(!n or !m) return {};
    while(sz < n+m-1) sz <<= 1;
    prepare(sz);
    for(int i=0; i<sz; i++) f[i] = cd(i < n ? a[i] : 0, i < m ? b[i] : 0);
    fft(f, sz);
    for(int i=0; i<=(sz>>1); i++){
        int j = (sz - i) & (sz - 1);
        cd x = (f[i]*f[i] - conj(f[j]*f[j])) * cd(0, -0.25);
        f[j] = x, f[i] = conj(x);
    }
    fft(f, sz);
    vector<ll>c(n+m-1);
    for(int i=0; i<n+m-1; i++) c[i] = round(f[i].real()/sz);
    return c;
}
```

4.4.2 NTT

```
const int G = 3;
const int MOD = 998244353;
const int N = ?; // (1 << 20) + 5; greater than maximum possible degree of any polynomial
int rev[N], w[N], inv_n;
int bigMod(int a, int e, int mod) {
    if(e == -1) assert(false);
    if(e == -1) e = mod - 2;
    int ret = 1;
    while (e) {
        if (e & 1) ret = (ll) ret * a % mod;
        a = (ll) a * a % mod; e >>= 1;
    }
    return ret;
}
void prepare(int &n) {
    int sz = abs(31 - __builtin_clz(n));
    int r = bigMod(G, (MOD - 1) / n, MOD);
    inv_n = bigMod(n, MOD - 2, MOD), w[0] = w[n] = 1;
    for (int i = 1; i < n; ++i) w[i] = (ll) w[i - 1] * r % MOD;
    for (int i = 1; i < n; ++i) rev[i] = (rev[i >> 1] >> 1) | ((i & 1) << (sz - 1));
}
void ntt (int *a, int n, int dir) {
    for (int i = 1; i < n - 1; ++i) {
        if (i < rev[i]) swap(a[i], a[rev[i]]);
    }
    for (int m = 2; m <= n; m <<= 1) {
        for (int i = 0; i < n; i += m) {
            for (int j = 0; j < (m >> 1); ++j) {
                int &u = a[i + j], &v = a[i + j + (m >> 1)];
                int t = (ll) v * w[dir ? n - n / m * j : n / m * j] % MOD;
                v = u - t < 0 ? u - t + MOD : u - t;
                u = u + t >= MOD ? u + t - MOD : u + t;
            }
        }
    }
    if (dir) for (int i = 0; i < n; ++i) a[i] = (ll) a[i] * inv_n % MOD;
}
int f_a[N], f_b[N];
vector <int> multiply (vector <int> a, vector <int> b) {
    int sz = 1, n = a.size(), m = b.size();
    while (sz < n + m - 1) sz <<= 1; prepare(sz);
    for (int i = 0; i < sz; ++i) f_a[i] = i < n ? a[i] : 0;
    for (int i = 0; i < sz; ++i) f_b[i] = i < m ? b[i] : 0;
    ntt(f_a, sz, 0); ntt(f_b, sz, 0);
    for (int i = 0; i < sz; ++i) f_a[i] = (ll) f_a[i] * f_b[i] % MOD;
    ntt(f_a, sz, 1); return vector <int> (f_a, f_a + n + m - 1);
}
// G = primitive_root(MOD)
int primitive_root (int p) {
    vector <int> factor;
    int tmp = p - 1;
    for (int i = 2; i * i <= tmp; ++i) {
        if (tmp % i == 0) {
            factor.emplace_back(i);
            while (tmp % i == 0) tmp /= i;
        }
    }
    if (tmp != 1) factor.emplace_back(tmp);
    for (int root = 1; ; ++root) {
        bool flag = true;
        for (int i = 0; i < (int) factor.size(); ++i) {
            if (bigMod(root, (p - 1) / factor[i], p) == 1) {
                flag = false; break;
            }
        }
        if (flag) return root;
    }
}
int main() {
    // (x + 2)(x + 3) = x^2 + 5x + 6
    vector <int> a = {2, 1};
    vector <int> b = {3, 1};
    vector <int> c = multiply(a, b);
    for (int x : c) cout << x << " "; cout << endl;
    return 0;
}
```

4.5 2-SAT

```
/***
    * 1 based index for variables
    * F = (a op b) and (c op d) and ...... (y op z)
      a, b, c ... are the variables
      sat::satisfy() returns true if there is some assignment(True/False)
      for all the variables that make F = True
    * init() at the start of every case
***/
namespace sat{
    #define CLR(ara,n) fill(ara+1,ara+n+1,0)
 const int MAX = 200010; /// number of variables * 2
    bool vis[MAX];
    vector <int> ed[MAX], rev[MAX];
    int n, m, ptr, dfs_t[MAX], ord[MAX], par[MAX];

    inline int inv(int x){
        return ((x) <= n ? (x + n) : (x - n));
    }
    /// Call init once
    void init(int vars){
        n = vars, m = vars << 1;
        for (int i = 1; i <= m; i++){
            ed[i].clear();
            rev[i].clear();
        }
    }
    /// Adding implication, if a then b ( a --> b )
    inline void add(int a, int b){
        ed[a].push_back(b);
        rev[b].push_back(a);
    }
    /// (a or b) is true --> OR(a,b)
    /// (a or b) is true --> OR(inv(a),b)
    /// (a or b) is true --> OR(a,inv(b))
    /// (a or b) is true --> OR(inv(a),inv(b))
    inline void OR(int a, int b){
        add(inv(a), b);
        add(inv(b), a);
    }
    /// same rule as or
    inline void AND(int a, int b){
        add(a, b);
        add(b, a);
    }
    /// same rule as or
    void XOR(int a,int b){
        add(inv(b), a);
        add(a, inv(b));
        add(inv(a), b);
        add(b, inv(a));
    }
    /// same rule as or
    inline void XNOR(int a, int b){
        add(a,b);
        add(b,a);
        add(inv(a), inv(b));
        add(inv(b), inv(a));
    }
    /// (x <= n) means forcing variable x to be true
    /// (x = n + y) means forcing variable y to be false
    inline void force_true(int x){
        add(inv(x), x);
    }
    inline void topsort(int s){
        vis[s] = true;
```

```
        for(int x : rev[s]) if(!vis[x]) topsort(x);
        dfs_t[s] = ++ptr;
    }
    inline void dfs(int s, int p){
        par[s] = p;
        vis[s] = true;
        for(int x : ed[s]) if (!vis[x]) dfs(x, p);
    }
    void build(){
        CLR(vis,m);
        ptr = 0;
        for(int i=m;i>=1;i--) {
            if (!vis[i]) topsort(i);
            ord[dfs_t[i]] = i;
        }
        CLR(vis,m);
        for (int i = m; i >= 1; i--){
            int x = ord[i];
            if (!vis[x]) dfs(x, x);
        }
    }
    /// Returns true if the system is 2-satisfiable
    ↪ and returns the solution (vars set to true)
    ↪ in vector res
    bool satisfy(vector <int>& res){
        build();
        CLR(vis,m);

        for (int i = 1; i <= m; i++){
            int x = ord[i];
            if (par[x] == par[inv(x)]) return false;
            if (!vis[par[x]]){
                vis[par[x]] = true;
                vis[par[inv(x)]] = false;
            }
        }
        res.clear();
        for (int i = 1; i <= n; i++){
            if (vis[par[i]]) res.push_back(i);
        }
        return true;
    }
    #undef CLR
}
```

4.6 Catalan_number

```
unsigned long int binomialCoeff(unsigned int n,
                                unsigned int k)
{
    unsigned long int res = 1;
    if (k > n - k)
        k = n - k;
    for (int i = 0; i < k; ++i) {
        res *= (n - i);
        res /= (i + 1);
    }
    return res;
}

unsigned long int catalan(unsigned int n)
{
    unsigned long int c = binomialCoeff(2 * n, n);
    return c / (n + 1);
}
```

4.7 Diophantine_Equation

```
int gcd_extend(int a, int b,int& x, int& y)
{
    if (b == 0) {
        x = 1;
        y = 0;
        return a;
    }
    else {
        int g = gcd_extend(b,a % b, x, y);
        int x1 = x, y1 = y;
        x = y1;
        y = x1 - (a / b) * y1;
        return g;
    }
}
void print_solution(int a, int b, int c)
{
    int x, y;
    if (a == 0 && b == 0) {
        if (c == 0) {
            cout << "Infinite Solutions Exist" <<
            ↪ endl;
        }
        else {
            cout << "No Solution exists" << endl;
        }
    }
    int gcd = gcd_extend(a, b, x, y);
    if (c % gcd != 0) {
        cout<< "No Solution exists"<< endl;
    }
    else {
        cout << "x = " << x * (c / gcd)<< ", y = "
        ↪ << y * (c / gcd)<< endl;
    }
}
```

4.8 Euler_Totient

```
int phi(int n) {
    int result = n;
    for (int i = 2; i * i <= n; i++) {
        if (n % i == 0) {
            while (n % i == 0)
                n /= i;
            result -= result / i;
        }
    }
    if (n > 1)
        result -= result / n;
    return result;
}
void phi_1_to_n(int n) {
    vector<int> phi(n + 1);
    phi[0] = 0;
    phi[1] = 1;
    for (int i = 2; i <= n; i++)
        phi[i] = i;

    for (int i = 2; i <= n; i++) {
        if (phi[i] == i) {
            for (int j = i; j <= n; j += i)
                phi[j] -= phi[j] / i;
            }
        }
    }
}
```

4.9 Extended Euclid

```
/***
    c = gcd(a,b);

    ax + by = c;
    (bq + r)x + by = c;
    bqx + rx + by = c;
    b(qx + y) + rx = c;
    bx' + ry' = c;          [r = a % b]

    We get,
    x' = qx + y;
    y' = x

    So,
    y = x' - qx;
    y = x' - qy';           [y' = x]
    and
    x = y'

    If c is not the gcd then,

    actual x = x * (c/gcd)
    actual y = y * (c/gcd)
    But if gcd doesn't divide c, there is no
↪   solution.
***/
/// returns (x,y) for ax + by = gcd(a,b)
/// keep in mind that if a or b or both are
↪ negative,  gcd(a,b) will be negative
PLL extEuclid(ll a,ll b)
{
    if(b==0LL) return mp(1LL,0LL);
    PLL ret,got;
    got = extEuclid(b,a%b);
    ret = mp(got.yy,got.xx-(a/b)*got.yy);
    return ret;
}
/***
    From one solution (x0,y0), we can obtain all
 ↪  the solutions of the given equation.
    Let g = gcd(a,b) and let x0,y0 be integers
 ↪  which satisfy the following:
    a*x0+b*y0 = c
    Now, we should see that adding b/g to x0 and at
 ↪  the same time subtracting a/g
    from y0 will not break the equality:

      a*(x0 + b/g) + b*(y0 - a/g)
    = a*x0 + b*y0 + (a*b)/g - (b*a)/g
    = c
    Obviously, this process can be repeated again,
 ↪  so all the numbers of the form:

    x = x0 + k * (b/g)
    y = y0 - k * (a/g)
    are solutions of the given Diophantine equation.

    In the solution returned by extEuclid :
    |x| and |y| is minimized
    |x| <= b/2g
```

```
    |y| <= a/2g
    Because we get a new x after every b/g amount
 ↪  of jump
    and we get a new y after every a/g amount of
 ↪  jump

    Solution with minimum (x+y):
    x + y = x0 + y0 + k*(b/g - a/g)
    x + y = x0 + y0 + k*((b-a)/g)

    If b>a, we need to find the k with the minimum
 ↪  value
    else we need to find the k with the maximum
 ↪  value
***/
/// Iterative Implementation
PLL extEuclid(ll a,ll b){
    ll s = 1,t = 0,st = 0,tt = 1;
    while(b) {
        s = s - (a/b)*st;
        swap(s,st);
        t = t - (a/b)*tt;
        swap(t,tt);
        a = a % b;
        swap(a,b);
    }
    return mp(s,t);
}
/// returns number of solutions for the equation ax
 ↪  + by = c
/// where minx <= x <= maxx and miny <= y <= maxy
ll numberOfSolutions(ll a,ll b,ll c,ll minx,ll
 ↪  maxx,ll miny,ll maxy)
{
    if(a==0 && b==0){
        if(c!=0) return 0;
        else return (maxx-minx+1)*(maxy-miny+1);
         ↪  /// all possible (x,y) within the
         ↪  ranges can be a solution
    }
    ll gcd = __gcd(a,b);
    if(c%gcd!=0) return 0;/// no solution ,
     ↪  gcd(a,b) doesn't divide c

    /// If b==0, x will be fixed, any y in the
     ↪  range can form a pair with that x
    if(b==0){
        c /= a;
        if(c>=minx && c<=maxx) return maxy-miny+1;
        else return 0;
    }

    /// If a==0, x will be fixed, any x in the
     ↪  range can form a pair with that y
    if(a==0){
        c /= b;
        if(c>=miny && c<=maxy) return maxx-minx+1;
        else return 0;
    }
    /// gives a particular solution to the equation
     ↪  ax + by = gcd(a,b) {gcd(a,b) can be
     ↪  negative also}
    PLL sol = extEuclid(a,b);
    a /= gcd;
    b /= gcd;
    c /= gcd;
    ll x,y;
    x = sol.xx*c;
    y = sol.yy*c;
    ll lx,ly,rx,ry;
    /// lx -> minimum value of k such that sol.xx +
     ↪  k * (b/g) is in range[minx,maxx]
    /// rx -> maximum value of k such that sol.xx +
     ↪  k * (b/g) is in range[minx,maxx]
    if(x<minx) lx = ceil( (minx-x) / (double)abs(b)
     ↪  );
    else lx = -floor( (x-minx) / (double)abs(b) );
    if(x<maxx) rx = floor((maxx-x) / (double)abs(b)
     ↪  );
    else rx = -ceil((x-maxx) / (double)abs(b) );
    /// Doing this I because I ignored sign of b
     ↪  before passing to getCeil/getFloor
    if(b<0){
        lx *= -1;
        rx *= -1;
        swap(lx,rx);
    }
    if(lx>rx) return 0;
    /// ly -> minimum value of k such that sol.yy -
     ↪  k * (a/g) is in range[miny,maxy]
    /// ry -> maximum value of k such that sol.yy -
     ↪  k * (a/g) is in range[miny,maxy]
    if(y<miny) ly = ceil( (miny-y) / (double)abs(a)
     ↪  );
    else ly = -floor( (y-miny) / (double)abs(a) );
    if(y<maxy) ry = floor( (maxy-y) /
     ↪  (double)abs(a) );
    else ry = -ceil( (y-maxy) / (double)abs(a) );
    /// Doing this because I ignored sign of a
     ↪  before passing to getCeil/getFloor
    if(a<0){
        ly *= -1;
        ry *= -1;
        swap(ly,ry);
    }
    if(ly>ry) return 0;
    ly *= -1;
    ry *= -1;
    swap(ly,ry);
    /// getting the intersection between (x range)
     ↪  and (y range) of k
    ll li = max(lx,ly);
    ll ri = min(rx,ry);
    return max( ri - li + 1 , 0LL );
}
```

4.10 Mobius Function

```
/***
    mu[1] = 1,
    mu[n] = 0 if n has a squared prime factor,
    mu[n] = 1 if n is square-free with even number
 ↪  of prime factors
    mu[n] = -1 if n is square-free with odd number
 ↪  of prime factors

    *** sum of mu[d] where d | n is 0 ( For n=1,
 ↪  sum is 1 )***
***/
int mu[MAX] = {0};
void Mobius(int N){
    int i, j;
    mu[1] = 1;
    for (i = 1; i <= N; i++){
        if (mu[i]){
            for (j = i + i; j <= N; j += i){
                mu[j] -= mu[i];
            }
        }
    }
}
```

4.11 Primes_stuff

```
const int N = 1000000+6;
vector<long long>primes;
bitset<N>flag;
vector<long long>v;
void siv()
{
    flag[1] = 1;
    for( int i = 2; i*i <= N; i++ ){
        if( flag[i] == 0 ){
            for( int j = i*i; j < N; j+= i )
             ↪  flag[j] = 1;
        }
    }
    for( int i = 2; i < N; i++ ){
        if( flag[i] == 0 ) primes.push_back(i);
    }
}
long long mul(long long a,long long b,long long mod)
{
    long long res = 0;
    a %= mod;
    while (b)
    {
        if (b & 1) res = (res + a) % mod;
        a = (2 * a) % mod;
        b >>= 1; // b = b / 2
    }
    return res;
}
long long gcd( long long x,long long y )
{
    if( x < 0 ) x = -x;
    if( y < 0 ) y = -y;
    if( !x || !y ) return x+y;
    long long temp;
    while( x%y ){
        temp = x;
        x = y;
        y = temp%y;
    }
    return y;
}
long long mod_inverse( long long n,long long p )
{
    long long x, y, g;
    g = gcd_extended( n,p,x,y );
    if( g < 0 ) x = -x;
    return (x%p +p)%p;
}
long long mpow( long long x,long long y,long long
 ↪  mod )
```

```
{
    long long ret = 1;
    while( y ){
        if( y&1 ) ret = mul(ret,x,mod);
        y >>= 1, x = mul(x,x,mod);
    }
    return ret%mod;
}
int isPrime( long long p )
{
    if( p < 2 || !(p&1) ) return 0;
    if( p == 2 ) return 1;
    long long q = p-1, a, t;
    int k = 0, b = 0;
    while( !(q&1) ) q >>=1, k++;
    for( int it = 0; it < 2; it++ ){
        a = rand()% (p-4) + 2;
        t = mpow( a,q,p );
        b = (t==1) || ( t == p-1 );
        for( int i = 1; i < k && !b; i++  ){
            t = mul(t,t,p);
            if( t == p-1 ) b = 1;
        }
        if( b == 0 ) return 0;
    }
    return 1;
}
long long pollard_rho( long long n,long long c )
{
    long long x = 2, y = 2, i = 1, k = 2, d;
    while( 1 ){
        x = ( mul(x,x,n) + c );
        if( x >= n ) x-= n;
        d = gcd(x-y,n);
        if( d > 1 ) return d;
        if( ++i == k ) y = x, k <<= 1;
    }
    return n;
}
map<long long,int>mp;
void factorize( long long n )
{
    int l = primes.size();
    for( int i = 0; primes[i]*primes[i] <= n && i <
     ↪  l; i++ ){
        if( n%primes[i] == 0 ){
            mp[primes[i]] = 1;
            while( n%primes[i] == 0 ) n/= primes[i];
        }
    }
    if( n != 1 ) mp[n] = 1;
}
void lfactorize( long long n )
{
    if( n == 1 ) return;
    if( n < 1e9 ){
        factorize(n);
        return;
    }
    if( isPrime(n) ){
        mp[n] = 1;
        return;
    }
    long long d = n;
    for( int i = 2; d == n; i++ ) d =
     ↪  pollard_rho(n,i);
    lfactorize(d);
    lfactorize(n/d);
}
long long f(long long r,vector<long long> v1){
    int sz=v1.size();
    long long res=0;

    for(long long i=1;i<(1<<sz);i++){
        int ct=0;
        long long mul=1;
        for(int j = 0; j < sz; j++ ) {
            if(i&(1<<j)){
                ct++;
                mul *= v1[j];
            }
        }

        long long sign=-1;
        if(ct&1)sign=1;

        res += sign*(r/mul);
    }

    return r-res;
}
```

4.12 Xor basis

```
int basis[d]; // basis[i] keeps the mask of the
 ↪  vector whose f value is i

int sz; // Current size of the basis

void insertVector(int mask) {
 for (int i = 0; i < d; i++) {
  if ((mask & 1 << i) == 0) continue; // continue
   ↪  if i != f(mask)

  if (!basis[i]) { // If there is no basis vector
   ↪  with the i'th bit set, then insert this
   ↪  vector into the basis
   basis[i] = mask;
   ++sz;

   return;
  }

  mask ^= basis[i]; // Otherwise subtract the basis
   ↪  vector from this vector
 }
}
```

4.13 stirling_number_of_2nd_kind

```
long long p = 1e9 + 7;
long long fact[1000005];
int n, m, k;
long long s( long long N,long long R )
{
    if( N == 0 && R == 0 ) return 1;
    if( N == 0 || R == 0 ) return 0;
    long long ans = 0;
    for( int i = 1; i <= R; i++ ){
        long long par;
        if( (R-i)%2 == 0 ) par = 1;
        else par = -1;
        par = (par+p)%p;
        long long temp = (ncr(R,i) * bm(i,N))%p;
        temp = (temp%p * par%p)%p;
        ans = (ans%p + temp%p)%p;
    }
    return (ans*bm( fact[R],p-2 ))%p;
}
```

5 Misc

5.1 Compilation

```
//pragma
#pragma GCC optimize("O3")
#pragma GCC optimize("unroll-loops")

compile: g++  -std=c++17 -I . -Dakifpathan -o  "%e"
 ↪  "%f"
build: g++ -std=c++17 -DHFTF -Wshadow -o "%e" "%f"
 ↪  -g -fsanitize=address -fsanitize=undefined
 ↪  -D_GLIBCXX_DEBUG
run: "./%e"
//for sublime
{
"cmd" : ["g++ -std=c++14 $file_name -o
 ↪  $file_base_name && timeout 6s
 ↪  ./$file_base_name<in>out"],
"selector" : "source.c, source.cpp, source.Cc",
"shell": true,
"working_dir" : "$file_path"
}
//windows
{
"cmd": ["g++.exe", "-std=c++14", "${file}", "-o",
 ↪  "${file_base_name}.exe", "&&",
 ↪  "${file_base_name}.exe<in>out"],
"shell":true,
"working_dir":"$file_path",
"selector":"source.cpp, source.c, source.c++,
 ↪  source.cc"
}
```

5.2 Ternary Search

```
while(hi>=lo)
{
 int mid1=lo+(hi-lo)/3; int mid2=hi-(hi-lo)/3;
 if(f(mid1)>f(mid2)) { }//change
 else //change
}//ittehad
double x1, why1, z1, x2, y2, z2, x, y , z;
double f( double t )
{
    double xt = x1 + (x2-x1)t;
    double yt = why1 + (y2-why1)t;
    double zt = z1 + (z2-z1)t;
    return ((xt-x)(xt-x) + (yt-y)(yt-y) +
     ↪  (zt-z)(zt-z));
}
double Tsearch()
{
    double low = 0, high = 1, mid;
    int step = 64;
    while( step-- ){
```

```
        double t1 = (2low + high)/3;
        double t2 = (low + 2high)/3;
        double d1 = f(t1);
        double d2 = f(t2);
        if( d1 < d2 ) high = t2;
        else low = t1;
    }
    return low;
}
```

6 String

6.1 Aho-Corasick

```
struct vartex
{
    int next[30];
    int endmark;
    int link;
    vector<int>dlink;
    vartex()
    {
        memset(next,-1,sizeof(next));
        endmark=-1;
        link=0;
    }
};

void addstring(string& s,vector<vartex>&trie)
{
    int v=0;
    for(auto x : s)
    {
        if(trie[v].next[x-'a']==-1)
        {
            trie[v].next[x-'a']=trie.size();
            trie.emplace_back();
        }
        v=trie[v].next[x-'a'];
    }
    trie[v].endmark=0;
}

void fail(vector<vartex>&trie)
{
    int v=0;
    trie[v].link=0;
    queue<int>q;
    q.push(0);
    while(!q.empty())
    {
        v=q.front();
        q.pop();
        for(int i=0;i<26;i++)
        {
            if(trie[v].next[i]!=-1)
            {
                if(v==0)
                {
                    trie[trie[v].next[i]].link=0;
                }
                else
                {
                    int x=trie[v].link;
                    while (x!=0 &&
                    ↪  trie[x].next[i]==-1)
                    {
                        x=trie[x].link;
                    }
                    if(trie[x].next[i]==-1)
                    {
                        trie[trie[v].next[i]].link=
                        ↪  0;
                    }
                    else
                    {
                        trie[trie[v].next[i]].link=
                        ↪  trie[x].next[i];
                    }
                }
                q.push(trie[v].next[i]);
            }
        }
    }
}

void dictionary_link(vector<vartex>&trie)
{
    queue<int>q;
    q.push(0);
    while (!q.empty())
    {
        int u=q.front();
        q.pop();
        for(int i=0;i<26;i++)
        {
            if(trie[u].next[i]!=-1)
            {
                q.push(trie[u].next[i]);
            }
        }
        int k=u;
        while (k!=0)
        {
            if(trie[k].endmark!=-1 && k!=u)
            {
                trie[u].dlink.push_back(k);
            }
            k=trie[k].link;
        }
        debug(u,trie[u].dlink);
    }

}
int search(string& s,vector<vartex>&trie)
{
    int v=0;
    for(auto x : s)
    {
        v=trie[v].next[x-'a'];
    }
    return trie[v].endmark;
}
```

6.2 Hashing_without_inv

```
#include<bits/stdc++.h>
using namespace std;
long long h[400005];
long long MOD[400005];
int L;
void pre_hash( string s )
{
    long long p = 31;
    long long m = 1e9 + 9;
    long long power = 1;
    long long hash = 0;
    int z = 0;
    for( int i = s.size()-1; i >= 0; i-- ){
        hash = ( hash*p+ (s[i]-'A'+1) )%m;
        h[i] = hash;
        MOD[z] = power;
        z++;
        power = (power*p)%m;
    }
}
long long f( int l,int r )
{
    long long val = h[r], m = 1e9 + 9;
    if( l != L-1 )
    {
        long long val2 = (h[l+1]%m  *MOD[l-r+1]%m
        ↪  )%m;
        val -= val2;
        val += m;
        val %= m;
    }
    if( val < 0 ) val = (val+m)%m;
    return val;
}
int main()
{
    string s;
    cin >> s;
    L = s.size();
    pre_hash(s);
    int q;
    cin >> q;
    while( q-- ){
        int l, r;
        cin >> l >> r;
        cout << f(l,r) << endl;
    }
    return 0;
}
```

6.3 KMP

```
#include<bits/stdc++.h>
using namespace std;
#define pii pair<int,int>
vector<int> prefix_function (string Z) {

    int n = (int) Z.length();

    vector<int> F (n);

     F[0]=0;

    for (int i=1; i<n; ++i) {

        int j = F[i-1];

        while (j > 0 && Z[i] != Z[j])
```

```
            j = F[j-1];

        if (Z[i] == Z[j])  ++j;

        F[i] = j;

    }

    return F;

}
int main()
{
    string s;
    cin >> s;
    vector<int> v = prefix_function(s);
    for( int i = 0; i < v.size(); i++ ) cout << v[i] << " ";
    cout << endl;
    return 0;
}
```

6.4 Manacher

```
/// When i is even, pal[i] = largest palindromic substring centered from str[i / 2]
/// When i is odd, pal[i] = largest palindromic substring centered between str[i / 2] and str[i / 2] + 1
vector <int> manacher(char *str){
    int i, j, k, l = strlen(str), n = l << 1;
    vector <int> pal(n);
    for (i = 0, j = 0, k = 0; i < n; j = max(0, j - k), i += k){
        while (j <= i && (i + j + 1) < n && str[(i - j) >> 1] == str[(i + j + 1) >> 1]) j++;
        for (k = 1, pal[i] = j; k <= i && k <= pal[i] && (pal[i] - k) != pal[i - k]; k++){
            pal[i + k] = min(pal[i - k], pal[i] - k);
        }
    }
    pal.pop_back();
    return pal;
}
int main(){
    char str[100];
    while (scanf("%s", str)){
        auto v = manacher(str);
        for (auto it: v) printf("%d ", it);
        puts("");
    }
    return 0;
}
```

6.5 Palindromic Tree

```
#define CLR(a) memset(a,0,sizeof(a))
/***
    * str is 1 based

    Each node in the palindromic tree denotes a STRING
    Node 1 denotes an imaginary string of size -1
    Node 2 denotes a string of size 0
    They are the two roots
    There can be maximum of (string_length + 2) nodes in total

    It's a directed tree. If we reverse the direction of the suffix links, we get a dag. In this DAG, if node v is reachable from node u iff, u is a substring of v.

    * if ( tree[A].next[x] == B )
      then, B = xAx
    * if ( tree[A].suffixLink == B )
      Then B is the longest possible palindrome which is a proper suffix  of A
      (node 1 is an exception)

    * occ[i] contains the number of occurrences of the corresponding palindrome
    * st[i] denotes starting index of the first occurrence of the corresponding palindrome
    * st[] or occ[] or both can be ignored if not needed

    * If memory limit is compact, a map has to be used instead of
      ed[MAXN][MAXC]. Swapping row and column of the matrix will
      save more memory.
      Example :
      map <int,int> ed[MAXC];
      ed[c][u] = v means, there is an edge from node u to
      node v that is labeled character c.

***/

namespace pt {
    const int MAXN = 100010; /// maximum possible string size
    const int MAXC = 26; /// Size of the character set

    int n; /// length of str
    char str[MAXN];
    int len[MAXN], link[MAXN], ed[MAXN][MAXC], occ[MAXN], st[MAXN];

    int nc, suff, pos;
    /// nc -> node count
    /// suff -> Index of the node denoting the longest palindromic proper suffix of the current prefix

    void init() {
        str[0] = -1;
        nc = 2; suff = 2;
        len[1] = -1, link[1] = 1;
        len[2] =  0, link[2] = 1;
        CLR(ed[1]), CLR(ed[2]);
        occ[1] = occ[2] = 0;
    }
    inline int scale(char c) { return c-'a'; }
    inline int nextLink(int cur) {
        while (str[pos - 1 - len[cur]] != str[pos]) cur = link[cur];
        return cur;
    }
    inline bool addLetter(int p) {
        pos = p;
        int let = scale(str[pos]);
        int cur = nextLink(suff);

        if (ed[cur][let]) {
            suff = ed[cur][let];
            occ[suff]++;
            return false;
        }

        suff = ++nc;
        CLR(ed[nc]);
        len[nc] = len[cur] + 2;
        ed[cur][let] = nc;
        occ[nc] = 1;

        if (len[nc] == 1) {
            st[nc] = pos;
            link[nc] = 2;
            return true;
        }
        link[nc] = ed[nextLink(link[cur])][let];
        st[nc] = pos-len[nc] + 1;
        return true;
    }
    void build(int _n) {
        n = _n;
        init();
        for(int i=1;i<=n;i++) addLetter(i);
        for(int i=nc;i>=3;i--) occ[link[i]] += occ[i];
        occ[2] = occ[1] = 0;
    }
    void printTree() {
        puts(str);
        cout << "Node\tStart\tLength\tOcc\n";
        for(int i=3;i<=nc;i++) {
            cout << i << "\t" << st[i] << "\t" << len[i] << "\t" << occ[i] << "\n";
        }
    }
}

int main() {
    scanf("%s",pt::str+1);
    pt::build(strlen(pt::str+1));
    return 0;
}
```

6.6 String Hashing

```
ll bigmod(ll x, ll p, ll md){
    ll res = 1;
    while(p){
        if(p & 1) res = res * x % md;
        x = x * x % md;
        p >>= 1;
    }
    return res;
}

ll modinv(ll x, ll md){
    return bigmod(x, md-2, md);
}
```

```cpp
namespace Hash{
    ll pw[M][2];
    ll invpw[M][2];
    const int pr[] = {37, 53};
    const int md[] = {1000000007, 1000000009};
    void precalc(){
        pw[0][0] = pw[0][1] = 1;
        for(int i=1; i<M; i++){
            pw[i][0] = pw[i-1][0] * pr[0] % md[0];
            pw[i][1] = pw[i-1][1] * pr[1] % md[1];
        }
        invpw[M-1][0] = modinv(pw[M-1][0], md[0]);
        invpw[M-1][1] = modinv(pw[M-1][1], md[1]);
        for(int i=M-2; i>=0; i--){
            invpw[i][0] = invpw[i+1][0] * pr[0] % md[0];
            invpw[i][1] = invpw[i+1][1] * pr[1] % md[1];
        }
    }
    pii get_hash(const string &s){
        pii ret = {0, 0};
        for(int i=0; i<s.size(); i++){
            ret.first += (s[i]-'a'+1)*pw[i][0] % md[0];
            ret.second += (s[i]-'a'+1)*pw[i][1] % md[1];
            if(ret.first >= md[0]) ret.first -= md[0];
            if(ret.second >= md[1]) ret.second -= md[1];
        }
        return ret;
    }
    void prefix(const string &s, pii *H){
        H[0] = {0, 0};
        for(int i=1; i<=s.size(); i++){
            H[i].first = H[i-1].first + (s[i-1]-'a'+1)*pw[i-1][0] % md[0];
            H[i].second = H[i-1].second + (s[i-1]-'a'+1)*pw[i-1][1] % md[1];
            if(H[i].first >= md[0]) H[i].first -= md[0];
            if(H[i].second >= md[1]) H[i].second -= md[1];
        }
    }
    void reverse_prefix(const string &s, pii *H){
        int n = s.size();
        for(int i=1; i<=s.size(); i++){
            H[i].first = H[i-1].first + (s[i-1]-'a'+1)*pw[n-i][0] % md[0];
            H[i].second = H[i-1].second + (s[i-1]-'a'+1)*pw[n-i][1] % md[1];
            if(H[i].first >= md[0]) H[i].first -= md[0];
            if(H[i].second >= md[1]) H[i].second -= md[1];
        }
    }
    pii range_hash(int L, int R, pii H[]){
        pii ret;
        ret.first = (H[R].first - H[L-1].first + md[0]) % md[0];
        ret.second = (H[R].second - H[L-1].second + md[1]) % md[1];
        ret.first = ret.first * invpw[L-1][0] % md[0];
        ret.second = ret.second * invpw[L-1][1] % md[1];
        return ret;
    }
    pii reverse_hash(int L, int R, pii H[], int n){
        pii ret;
        ret.first = (H[R].first - H[L-1].first + md[0]) % md[0];
        ret.second = (H[R].second - H[L-1].second + md[1]) % md[1];
        ret.first = ret.first * invpw[n-R][0] % md[0];
        ret.second = ret.second * invpw[n-R][1] % md[1];
        return ret;
    }
}
```

6.7 Suffix Array (nlog^2n)

```cpp
///if finding suffix array of vector, use const vector<int> &s
vector<int> sa_doubling(const string& s) {
    int n = int(s.size());
    vector<int> sa(n),rnk(n),tmp(n);
    for(int i=0;i<n;i++) rnk[i]=s[i];
    iota(sa.begin(), sa.end(), 0);
    for (int k = 1; k < n; k *= 2) {
        auto cmp = [&](int x, int y) {
            if (rnk[x] != rnk[y]) return rnk[x] < rnk[y];
            int rx = x + k < n ? rnk[x + k] : -1;
            int ry = y + k < n ? rnk[y + k] : -1;
            return rx < ry;
        };
        sort(sa.begin(), sa.end(), cmp);
        tmp[sa[0]] = 0;
        for (int i = 1; i < n; i++) {
            tmp[sa[i]] = tmp[sa[i - 1]] + (cmp(sa[i - 1], sa[i]) ? 1 : 0);
        }
        swap(tmp, rnk);
    }
    return sa;
}

template <class T>
std::vector<int> lcp_array(const std::vector<T>& s,
                           const std::vector<int>& sa) {
    int n = int(s.size());
    assert(n >= 1);
    std::vector<int> rnk(n);
    for (int i = 0; i < n; i++) {
        rnk[sa[i]] = i;
    }
    std::vector<int> lcp(n - 1);
    int h = 0;
    for (int i = 0; i < n; i++) {
        if (h > 0) h--;
        if (rnk[i] == 0) continue;
        int j = sa[rnk[i] - 1];
        for (; j + h < n && i + h < n; h++) {
            if (s[j + h] != s[i + h]) break;
        }
        lcp[rnk[i] - 1] = h;
    }
    return lcp;
}
```

6.8 Suffix Array (nlogn)

```cpp
struct suffix_array
{
    vector<int> sa, lcp;
    suffix_array(string &s, int lim = 256)
    {
        int n = s.size() + 1, k = 0, a, b;
        vector<int> x(s.begin(), s.end() + 1), y(n), ws(max(n, lim)), rank(n);
        sa = lcp = y, iota(sa.begin(), sa.end(), 0);
        for (int j = 0, p = 0; p < n; j = max(1, j * 2), lim = p)
        {
            p = j, iota(y.begin(), y.end(), n - j);
            for (int i = 0; i < n; i++)
            {
                if (sa[i] >= j)
                {
                    y[p++] = sa[i] - j;
                }
            }
            fill(ws.begin(), ws.end(), 0);
            for (int i = 0; i < n; i++)
            {
                ws[x[i]]++;
            }
            for (int i = 1; i < lim; i++)
            {
                ws[i] += ws[i - 1];
            }
            for (int i = n; i--;)
            {
                sa[--ws[x[y[i]]]] = y[i];
            }
            swap(x, y), p = 1, x[sa[0]] = 0;
            for (int i = 1; i < n; i++)
            {
                a = sa[i - 1];
                b = sa[i];
                x[b] = (y[a] == y[b] && y[a + j] == y[b + j]) ? p - 1 : p++;
            }
        }
        for (int i = 1; i < n; i++)
        {
            rank[sa[i]] = i;
        }
        for (int i = 0, j; i < n - 1; lcp[rank[i++]] = k)
        {
```

```
            for (k &&k--, j = sa[rank[i] - 1]; s[i
            ↪ + k] == s[j + k]; k++)
                ;
        }
    }
};
```

6.9 Suffix Automaton

```
// collected from cp algorithm
struct state {
    int len, link, cnt, firstpos; // cnt -> endpos
    ↪ set size, link -> suffix link
    map <char, int> next;
};

const int MAXLEN = 100002;
state st[MAXLEN * 2];

struct SuffixAutomata {  // 0-based
    int sz, last;

    SuffixAutomata() { // init
        st[0].cnt = st[0].len = 0;
        st[0].link = -1;
        sz = 1, last = 0;
    }

    void add(char c) {  // add new char in automata
        int cur = sz++;
        st[cur].len = st[last].len + 1;
        st[cur].firstpos = st[cur].len - 1;
        st[cur].cnt = 1;
        int p = last;

        while (p != -1 && !st[p].next.count(c)) {
            st[p].next[c] = cur;
            p = st[p].link;
        }

        if (p == -1) {
            st[cur].link = 0;
        }
        else {
            int q = st[p].next[c];
            if (st[p].len + 1 == st[q].len) {
                st[cur].link = q;
            }
            else {  // clone state
                int clone = sz++;
                st[clone].len = st[p].len + 1;
                st[clone].next = st[q].next;
                st[clone].link = st[q].link;
                st[clone].firstpos = st[q].firstpos;
                st[clone].cnt = 0;

                while (p != -1 && st[p].next[c] ==
                ↪ q) {
                    st[p].next[c] = clone;
                    p = st[p].link;
                }

                st[q].link = st[cur].link = clone;
            }
        }

        last = cur;
    }

    void occurrence() {  // calculate number of
    ↪ occurrences of all possible substring
        vector <int> rank(sz);
        iota(all(rank), 0);
        sort(all(rank), [&](int i, int j) {
            return st[i].len > st[j].len;
        });

        for (int ii : rank) if (st[ii].link != -1)
            st[st[ii].link].cnt += st[ii].cnt;
    }

    int count(string s) {  // number of occurrences
    ↪ of string s. #prerequisite ->  call
    ↪ occurrence()
        int node = 0;
        for (char ch : s) {
            if (!st[node].next.count(ch)) return 0;
            node = st[node].next[ch];
        }
        return st[node].cnt;
    }

    int firstOcc(string s) {  // first
    ↪ position(occurence) of string s
        int node = 0;
        for (char ch : s) {
            if (!st[node].next.count(ch)) return -1;
            node = st[node].next[ch];
        }
        return st[node].firstpos + 2 -
        ↪ (int)s.size();
    }

    void build(string S) {  // build suffix
    ↪ automata
        for (char ch : S) add(ch);
    }

    bool find(string s) {   // find string s in
    ↪ automata
        int node = 0;
        for (char ch : s) {
            if (!st[node].next.count(ch)) return
            ↪ false;
            node = st[node].next[ch];
        }
        return true;
    }
};
```

6.10 Trie

```
//define M, K = alphabet size
int trie[M][K], word[M*K+3], cnt[M*K+3], sz;

void Insert(string s){
    int node = 0;

    for(int i=0; i<s.size(); i++){
        int c = s[i]-'a';
        if(!trie[node][c]){
            trie[node][c] = ++sz;
        }
        node = trie[node][c];
        cnt[node]++;
    }
    word[node]++;
}
bool Search(string s){
    int node = 0, ret = 0;

    for(int i=0; i<s.size(); i++){
        int c = s[i]-'a';
        if(!trie[node][c]) return false;
        node = trie[node][c];
    }
    return (word[node] > 0);
}
void Delete(string s){
    int node = 0;
    vector<int>v(1, 0);

    for(int i=0; i<s.size(); i++){
        int c = s[i]-'a';
        node = trie[node][c];
        cnt[node]--;
        v.push_back(node);
    }

    word[node]--;

    for(int i=1; i<v.size(); i++){
        int c = s[i-1]-'a';
        if(!cnt[v[i]]){
            trie[v[i-1]][c] = 0;
        }
    }
}
```

6.11 Z algo

```
/*
 * z[i] denotes the maximum length of substring
 * starting from position(i) which is also a prefix
 * of the string
 * call with Z zf(x) where x is the desired string
*/
struct Z
{
 int n;
 string s;
 vector<int>z;
 Z(const string &a)
 {
  n=a.size(); s=a; z.assign(n,0);
 }
 void z_function()
 {
  for (int i = 1, l = 0, r = 0; i < n; ++i)
  {
   if (i <= r) z[i] = min (r - i + 1, z[i - l]);
   while (i + z[i] < n && s[z[i]] == s[i + z[i]])
   ↪ ++z[i];
   if (i + z[i] - 1 > r) l = i, r = i + z[i] - 1;
  }
 }
};
```