KUET_EVANGELION

Faysal Mahmud, Umar Faruk, Niloy Das

Contents

1 Data Structure

1.1 FenwickTree [35 lines] - 9bd56f72

```
/* * Description: Computes partial sums a[1] + a[2] +
    ... + a[pos], and updates single elements a[i],
 * taking the difference between the old and new value.
 * Time: Both operations are $O(\log N)$.
 * Status: Stress-tested
 * Note: 1-based indexing
 */
#pragma once

struct FT {
    vector<ll> s;
    FT(int n) : s(n + 1) {} // size + 1 for 1-based
        indexing

    void update(int pos, ll dif) { // a[pos] += dif
        for (; pos < sz(s); pos += pos & -pos)
            s[pos] += dif;
    }

    ll query(int pos) { // sum of values in [1, pos]
        ll res = 0;
        for (; pos > 0; pos -= pos & -pos)
            res += s[pos];
        return res;
    }

    int lower_bound(ll sum) { // min pos st sum of [1,
        pos] >= sum
        // Returns n if no sum is >= sum, or 0 if empty
            sum is.
        if (sum <= 0) return 0;
        int pos = 0;
        for (int pw = 1 << 25; pw; pw >>= 1) {
            if (pos + pw < sz(s) && s[pos + pw] < sum)
                pos += pw, sum -= s[pos];
        }
        return pos + 1;
    }
};
```

1.2 FenwickTree2h [31 lines] - d9a9aa5f

```
* Description: Computes sums a[i,j] for all i<I, j<J,
    and increases single elements a[i,j].
 *  Requires that the elements to be updated are known
     in advance (call fakeUpdate() before init()).
 * Time: $O(\log^2 N)$. (Use persistent segment trees
    for $O(\log N)$.)
 * Status: stress-tested
 */
#pragma once

#include "FenwickTree.h"

struct FT2 {
  vector<vi> ys; vector<FT> ft;
  FT2(int limx) : ys(limx) {}
  void fakeUpdate(int x, int y) {
    for (; x < sz(ys); x |= x + 1) ys[x].push_back(y);
  }
  void init() {
    for (vi& v : ys) sort(all(v)),
        ft.emplace_back(sz(v));
  }
  int ind(int x, int y) {
    return (int)(lower_bound(all(ys[x]), y) -
        ys[x].begin()); }
  void update(int x, int y, ll dif) {
    for (; x < sz(ys); x |= x + 1)
      ft[x].update(ind(x, y), dif);
  }
  ll query(int x, int y) {
    ll sum = 0;
    for (; x; x &= x - 1)
      sum += ft[x-1].query(ind(x-1, y));
    return sum;
  }
};
```

1.3 Persistent Segment Tree [64 lines] - 56d90f29

```
const int mxn = 4e5+5;
int root[mxn], leftchild[25*mxn], rightchild[25*mxn],
    value[25*mxn], a[mxn];
int now = 0, n, sz = 1;
int l, r;
```

```
int build(int L, int R){
  int node = ++now;
  if(L == R){
    //initialize
    //value[node] = a[L];
    return node;
  }
  int mid = (L+R)>>1;
  leftchild[node] = build(L, mid);
  rightchild[node] = build(mid+1, R);
  //combine
  //value[node] = value[leftchild[node]] +
    value[rightchild[node]];
  return node;
}

int update(int nownode, int L, int R, int ind, int val){
  int node = ++now;
  if(L == R){
    //value[node] = value[nownode]+val;
    //update value[node]
    return node;
  }
  int mid = (L+R)>>1;
  leftchild[node] = leftchild[nownode];
  rightchild[node] = rightchild[nownode];
  if(mid >= ind){//change condition as required
    leftchild[node] = update(leftchild[nownode], L,
    mid, ind, val);
  }
  else{
    rightchild[node] = update(rightchild[nownode],
    mid+1, R, ind, val);
  }
  //value[node] = value[leftchild[node]] +
    value[rightchild[node]];
  //combine value[node]
  return node;
}


int query(int nownode, int L, int R){
  if(l > R || r < L) return 0;

  if(L>=l && r >= R){
    return value[nownode];
  }
  int mid = (L+R)>>1;
  //change as required
  return query(leftchild[nownode], L, mid) +
    query(rightchild[nownode], mid+1, R);
}

void persistant(){
  root[0] = build(1, n);
  while(m--){
    if(ck == 2){
      cout << query(root[idx], 1, n) << "\n";
    }
    else{
          root[sz++] = update(root[idx], 1, n, ind,
    val);
    }
  }
}
```

1.4 Treap [166 lines] - d98b40d8

```
struct Treap {
  struct Node {
    int val, priority, cnt;  // value, priority, subtree
    size
    Node* l, * r;              // left child,right child
    pointer
    Node() {} //rng from template
    Node(int key) : val(key), priority(rng()),
    l(nullptr), r(nullptr) {}
  };
  typedef Node* node;
  node root;
  Treap() : root(0) {}
  int cnt(node t) { return t ? t->cnt : 0; }  // return
    subtree size
  void updateCnt(node t) {
    if (t) t->cnt = 1 + cnt(t->l) + cnt(t->r);  //
    update subtree size
  }
  void push(node cur) {
    ;  // Lazy Propagation
  }

  void combine(node& cur, node l, node r) {
    if (!l) {
      cur = r;
      return;
    }
    if (!r) {
      cur = l;
      return;
    }
    // Merge Operations like in segment tree
  }

  void reset(node& cur) {
    if (!cur) return;  // To reset other fields of cur
    except value and cnt
  }

  void operation(node& cur) {
    if (!cur) return;
    reset(cur);
    combine(cur, cur->l, cur);
    combine(cur, cur, cur->r);
  }
  // Split(T,key): split the tree in two tree. Left
    pointer contains all value
  // less than or equal to key.Right pointer contains
    the rest.
  void split(node t, node& l, node& r, int key) {
    if (!t)
      return void(l = r = nullptr);
    push(t);
    if (t->val <= key) {
      split(t->r, t->r, r, key), l = t;

    }
    else {
      split(t->l, l, t->l, key), r = t;
    }
    updateCnt(t);
    operation(t);
  }
  void splitPos(node t, node& l, node& r, int k, int add
    = 0) {
    if (!t) return void(l = r = 0);
    push(t);
    int idx = add + cnt(t->l);
    if (idx <= k)
      splitPos(t->r, t->r, r, k, idx + 1), l = t;
    else
      splitPos(t->l, l, t->l, k, add), r = t;
    updateCnt(t);
    operation(t);
  }
  // Merge(T1,T2): merges 2 tree into one.The tree with
    root of higher
  // priority becomes the new root.
  void merge(node& t, node l, node r) {
    push(l);
    push(r);
    if (!l || !r)
      t = l ? l : r;
    else if (l->priority > r->priority)
      merge(l->r, l->r, r), t = l;
    else
      merge(r->l, l, r->l), t = r;
    updateCnt(t);
    operation(t);
  }

  node merge_treap(node l, node r) {
    if (!l) return r;
    if (!r) return l;
    if (l->priority < r->priority) swap(l, r);
    node L, R;
    split(r, L, R, l->val);
    l->r = merge_treap(l->r, R);
    l->l = merge_treap(L, l->l);
    updateCnt(l);
    operation(l);
    return l;
  }
  // insert creates a set.all unique value.
  void insert(int val) {
    if (!root) {
      root = new Node(val);
      return;
    }
    node l, r, mid, mid2, rr;
    mid = new Node(val);
    split(root, l, r, val);
    merge(l, l, mid);  // these 3 lines will create
    multiset.
    merge(root, l, r);
    /*split(root, l, r, val - 1);  // l contains all
    small values.
      merge(l, l, mid);            // l contains new val
    too.
      split(r, mid2, rr, val);     // rr contains all
    greater values.
      merge(root, l, rr);*/
  }
  // removes all similar values.
```

```
  void erase(int val) {
    node l, r, mid;
    /* Removes all similar element*/
    split(root, l, r, val - 1);
    split(r, mid, r, val);
    merge(root, l, r);
    /*Removes single instance*/
    /*split(root, l, r, val - 1);
      split(r, mid, r, val);
      merge(mid,mid->l,mid->r);
      merge(l, l, mid);
      merge(root, l, r);*/
  }
  void clear(node cur) {
    if (!cur) return;
    clear(cur->l), clear(cur->r);
    delete cur;
  }

  void clear() { clear(root); }
  void inorder(node t) {
    if (!t) return;
    inorder(t->l);
    cout << t->val << ' ';
    inorder(t->r);
  }
  void inorder() {
    inorder(root);
    puts("");
  }
  //1 indexed - xth element after sorting.
  int find_by_order(int x) {
    if (!x) return -1;
    x--;
    node l, r, mid;
    splitPos(root, l, r, x - 1);
    splitPos(r, mid, r, 0);
    int ans = -1;
    if (cnt(mid) == 1) ans = mid->val;
    merge(r, mid, r);
    merge(root, l, r);
  }
  // 1 indexed. index of val in sorted array. -1 if not
     found.
  int order_of_key(int val) {
    node l, r, mid;
    split(root, l, r, val - 1);
    split(r, mid, r, val);
    int ans = -1;
    if (cnt(mid) == 1) ans = 1 + cnt(l);
    merge(r, mid, r);
    merge(root, l, r);
    return ans;
  }
};
```

1.5 Zfunc [22 lines] - 6164fab0

```
/**
 * Author: chilli
 * License: CC0
 * Description: z[i] computes the length of the longest
     common prefix of s[i:] and s,
 * except z[0] = 0. (abacaba -> 0010301)
 * Time: O(n)
 * Status: stress-tested
 */
#pragma once

vi Z(const string& S) {
  vi z(sz(S));
  int l = -1, r = -1;
  rep(i,1,sz(S)) {
    z[i] = i >= r ? 0 : min(r - i, z[i - l]);
    while (i + z[i] < sz(S) && S[i + z[i]] == S[z[i]])
      z[i]++;
    if (i + z[i] > r)
      l = i, r = i + z[i];
  }
  return z;
}
```

1.6 fft [50 lines] - c672f576

```
#include <bits/stdc++.h>
using namespace std;
using cd = complex<double>;
const double PI = acos(-1);

void fft(vector<cd> & a, bool invert) {
    int n = a.size();
    if (n == 1)
        return;

    vector<cd> a0(n / 2), a1(n / 2);
    for (int i = 0; 2 * i < n; i++) {
        a0[i] = a[2*i];
        a1[i] = a[2*i+1];
    }
    fft(a0, invert);
    fft(a1, invert);

    double ang = 2 * PI / n * (invert ? -1 : 1);
    cd w(1), wn(cos(ang), sin(ang));
    for (int i = 0; 2 * i < n; i++) {
        a[i] = a0[i] + w * a1[i];
        a[i + n/2] = a0[i] - w * a1[i];
        if (invert) {
            a[i] /= 2;
            a[i + n/2] /= 2;
        }
        w *= wn;
    }
}

vector<int> multiply(vector<int> const& a, vector<int>
    const& b) {
    vector<cd> fa(a.begin(), a.end()), fb(b.begin(),
    b.end());
    int n = 1;
    while (n < a.size() + b.size())
        n <<= 1;
    fa.resize(n);
    fb.resize(n);

    fft(fa, false);
    fft(fb, false);
    for (int i = 0; i < n; i++)
        fa[i] *= fb[i];
    fft(fa, true);
    vector<int> result(n);
    for (int i = 0; i < n; i++)
        result[i] = round(fa[i].real());
    return result;
}
```

1.7 lca [36 lines] - 0421573f

```
vector<vector<int>>g;
int timer;
int stime[N];
int etime[N];
int pp[N][20];
void dfs(int node,int par)
{
    stime[node]=++timer;
    pp[node][0]=par;
    for(int
    i=1;i<20;i++)pp[node][i]=pp[pp[node][i-1]][i-1];
    for(auto it:g[node])
    {
        if(it==par)continue;
        dfs(it,node);
    }
    etime[node]=++timer;
}
int isansistor(int x,int y)
{
    if(x==0)return 1;
    return stime[x]<=stime[y]&&etime[x]>=etime[y];
}
int lca(int x,int y)
{
    if(stime[x]>stime[y])swap(x,y);
    if(isansistor(x,y))return x;
    for(int i=19;i>=0;i--)
    {
        int tem=pp[x][i];
        if(isansistor(tem,y)==0)
        {
            x=tem;
        }
    }
    return pp[x][0];
}
```

1.8 matrixMultiplication [33 lines] - 4fbe16ba

```
vector<vector<ll>> matmul(const vector<vector<ll>> &A,
                          const vector<vector<ll>> &B) {
    vector<vector<ll>> C(3, vector<ll>(3, 0));
    for (int i = 0; i < 3; i++)
        for (int k = 0; k < 3; k++)
            if (A[i][k])
                for (int j = 0; j < 3; j++)
                    C[i][j] = (C[i][j] + A[i][k] *
    B[k][j]) % MOD;
    return C;
}

vector<vector<ll>> matpow(vector<vector<ll>> n, ll k) {
    vector<vector<ll>> ans(3, vector<ll>(3, 0));
    for (int i = 0; i < 3; i++) ans[i][i] = 1;

    while (k > 0) {
```

```
        if (k & 1) ans = matmul(ans, n);
        n = matmul(n, n);
        k >>= 1;
    }
    return ans;
}

ll qp(ll a,ll b)
{
    ll x=1; a%=MOD;
    while(b)
    {
        if(b&1) x=x*a%MOD;
        a=a*a%MOD; b>>=1;
    }
    return x;
}
```

1.9 mergeSortTree [56 lines] - 10087f57

```
#include <bits/stdc++.h>
using namespace std;
const int N=1e5+5;
int n,q,a[N];
multiset<int> seg[4*N+5];
void build(int node,int l,int r){
    if (l==r){
        seg[node].insert(a[l]);
        return;
    }int mid=(l+r)/2;
    build(node*2,l,mid);
    build(node*2+1,mid+1,r);
    for (int i=l;i<=r;i++) seg[node].insert(a[i]);
    return;
}
void edit(int node,int l,int r,int idx,int val){
    if (l==r){
        seg[node].erase(a[idx]);
        seg[node].insert(val);
        return;
    }int mid=(l+r)/2;
    if (idx<=mid) edit(node*2,l,mid,idx,val);
    else edit(node*2+1,mid+1,r,idx,val);
    seg[node].erase(a[idx]);
    seg[node].insert(val);
    return;
}
int query(int node,int l,int r,int lx,int rx,int x){
    if (l>rx || r<lx) return INT_MAX;
    if (l>=lx && r<=rx){
        auto it=seg[node].lower_bound(x);
        if (it==seg[node].end()) return INT_MAX;
        return *it;
    }int mid=(l+r)/2;
    return min(query(node*2,l,mid,lx,rx,x),query(node*
    2+1,mid+1,r,lx,rx,x));
}
int32_t main(){
    ios_base::sync_with_stdio(false);cin.tie(NULL);
    cout.tie(NULL);
    cin>>n>>q;
    for (int i=1;i<=n;i++) cin>>a[i];
    build(1,1,n);
    while (q--){
        bool t;cin>>t;
        if (!t){
            int idx,val;
            cin>>idx>>val;
            edit(1,1,n,idx,val);
            a[idx]=val;
            continue;
        }int l,r,x;cin>>l>>r>>x;
        int y=query(1,1,n,l,r,x);
        if (y==INT_MAX) cout<<-1<<endl;
        else cout<<y<<endl;
    }
    return 0;
}
```

1.10 pbds [9 lines] - be428411

```
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
#define ordered_set tree<int, null_type,less<int>,
    rb_tree_tag,tree_order_statistics_node_update>
ordered_set oset;
for(int i=0;i<n;i++)oset.insert(i);
int nxt=*oset.find_by_order(x+1);
oset.erase(oset.find(nxt));
int q=oset.order_of_key(p);
```

1.11 sparse$_t$able[42lines] – 1b2d11b1

```
class SparseTable
{
public:
    int n;
    vector<vector<int>> table;
    int limit;
    vector<int> lg;
    SparseTable(int _n) {
        n = _n;
        limit = __lg(n) + 2;
        lg.assign(n + 5, 0);
        table.assign(n + 5, vector<int>(limit));
        for (int i = 2; i <= n; i++) {
            lg[i] = lg[i / 2] + 1;
        }
    }
    virtual int merge(int u, int v) = 0;
    void build(const vector<int> &a) {
        for (int i = 0; i < n; i++) {
            table[i][0] = a[i];
        }
        for (int j = 1; j < limit; j++) {
            for (int i = 0; i + (1<<j) <= n; i++) {
                table[i][j] = merge(table[i][j - 1],
    table[i + (1<<(j - 1))][j - 1]);
            }
        }
    }
    int query(int l, int r) {
        int len = lg[r - l + 1];
        int ans = merge(table[l][len], table[r -
    (1<<len) + 1][len]);
        return ans;
    }
};

class Max : public SparseTable
{
public:
    Max(int n) : SparseTable(n){}
    int merge(int u, int v) {return max(u, v);}
};
//vector<Max> sp(k + 1, Max(n + 1));
//Max st(5);
```

1.12 suffixArrayBinSearch [168 lines] - 68b791d3

```
struct sparse_table {
private:
    ll maxn;
    ll k;
    vector<vector<ll>> TABLE;
    vector<ll> logs;

public:
    sparse_table (ll n) {
        maxn = n;
        logs.resize(maxn+1, 0);
        for (ll i = 2; i <= maxn; i++) logs[i] = logs[i
    >> 1] + 1;
        k = logs[maxn] + 1;
        TABLE.resize(maxn, vector<ll> (k));
    }
    void create_table (vector<ll> &A) {
        for (ll i = 0; i < maxn; i++) TABLE[i][0] =
    A[i];
        for (ll i = 1; i < k; i++) {
            for (ll j = 0; j + (1 << i) - 1 < maxn; j++)
    {
                TABLE[j][i] = min(TABLE[j][i-1] ,
    TABLE[j + (1 << (i-1))][i-1]);
            }
        }
    }
    // indexing should be zero based

    ll query (ll x, ll y) {
        ll gap = (y - x) + 1;
        ll lg = logs[gap];
        return min(TABLE[x][lg], TABLE[y - (1 << lg) +
    1][lg]);
    }

    ll query_sum (ll x, ll y) {
        long long ans = 0;
        ll gap = y - x + 1;
        ll log_gap = logs[gap];
        for (ll i = log_gap; i >= 0; i--) {
            if ((1 << i) <= (y - x + 1)) {
                ans += (long long) TABLE[x][i];
                x += (1 << i);
            }
        }
        return ans;
    }

};

typedef struct sparse_table sparse;
class SuffixArray {
private:

    void countSort(vector<ll>& p, vector<ll>& c) {
```

```
        ll n = p.size();
        vector<ll> cnt(n);
        for (auto x : c) {
            cnt[x]++;
        }

        vector<ll> p_new(n);
        vector<ll> pos(n);
        pos[0] = 0;
        for (ll i = 1; i < n; i++) {
            pos[i] = pos[i - 1] + cnt[i - 1];
        }
        for (auto x : p) {
            ll i = c[x];
            p_new[pos[i]] = x;
            pos[i]++;
        }
        p = p_new;
    }

public:
    string s;
    ll n;
    int totk = 0;
    vector<ll> p;
    vector<vector<ll> > ck;
    vector<ll>lcp;
    SuffixArray(){}

    void build(string s) {
        s += " ";
        this->s = s;
        n = this->s.size();
        p.resize(n);

        lcp.resize(n);
        ck.resize(32);
        for(auto &row:ck){
            row.resize(n);
        }

        // k = 0
        vector<pair<char, ll>> a(n);
        for (ll i = 0; i < n; i++) a[i] = {s[i], i};
        sort(a.begin(), a.end());

        for (ll i = 0; i < n; i++) p[i] = a[i].second;
        ck[0][p[0]] = 0;

        for (ll i = 1; i < n; i++) {
            if (a[i].first == a[i - 1].first) {
                ck[0][p[i]] = ck[0][p[i - 1]];
            } else {
                ck[0][p[i]] = ck[0][p[i - 1]] + 1;
            }
        }

        ll k = 0;
        while ((1 << k) < n) {
            // k -> k + 1
            for (ll i = 0; i < n; i++) {
                p[i] = (p[i] - (1 << k) + n) % n;
            }
            countSort(p, ck[k]);
            ck[k + 1][p[0]] = 0;
            for (ll i = 1; i < n; i++) {
                pair<ll, ll> prev = {ck[k][p[i - 1]],
    ck[k][(p[i - 1] + (1 << k)) % n]};
                pair<ll, ll> now = {ck[k][p[i]],
    ck[k][(p[i] + (1 << k)) % n]};
                if (now == prev) {
                    ck[k + 1][p[i]] = ck[k + 1][p[i -
    1]];
                } else {
                    ck[k + 1][p[i]] = ck[k + 1][p[i -
    1]] + 1;
                }
            }
            k++;
        }

        totk = k;
        k = 0;

        //p[i] hocche amar sorted array.
        //c[i] hocche p[i] er inv => c[p[i]] = i;
        //c[i] hocche suffix array er kon index
        for(ll i = 0 ; i < n - 1 ; i++){
            ll pi = ck[totk][i];
            ll j = p[pi - 1];
            while(s[i + k] == s[j + k]){

                k++;
            }

            lcp[pi] = k;
            k = max(0LL , k - 1);
        }
        for(int i = 1 ; i < n ; i++){
            debug(lcp[i] , s.substr(p[i]));
        }


    }
}suffix_array;
int cnt = 0;
ll upper(ll lo , ll hi , ll cur , ll val ,
    sparse_table&sp){

    if(lo == hi)return lo;
    ll mid = (lo + hi) / 2;
    if(sp.query(cur , mid) >= val)return upper(mid + 1
    , hi , cur , val , sp);
    return upper(lo , mid , cur , val , sp);
}
ll upper_l(ll lo , ll hi , ll cur , ll val ,
    sparse_table&sp){

    if(lo == hi)return lo;
    ll mid = (lo + hi) / 2;
    if(sp.query(mid + 1 , cur) >= val)return upper_l(lo
    , mid , cur , val , sp);
    return upper_l(mid + 1 , hi , cur , val , sp);
}
```

2 Flow

2.1 Blossom [58 lines] - 4efb5ae3

```
// Finds Maximum matching in General Graph
// Complexity O(NM)
// mate[i] = j means i is paired with j
// source: https://codeforces.com/blog/entry
    /92339?#comment-810242
vector<int> Blossom(vector<vector<int>>& graph) {
  //mate contains matched edge.
  int n = graph.size(), timer = -1;
  vector<int> mate(n, -1), label(n), parent(n),
    orig(n), aux(n, -1), q;
  auto lca = [&](int x, int y) {
    for (timer++; ; swap(x, y)) {
      if (x == -1) continue;
      if (aux[x] == timer) return x;
      aux[x] = timer;
      x = (mate[x] == -1 ? -1 : orig[parent[mate[x]]]);
    }
  };
  auto blossom = [&](int v, int w, int a) {
    while (orig[v] != a) {
      parent[v] = w; w = mate[v];
      if (label[w] == 1) label[w] = 0, q.push_back(w);
      orig[v] = orig[w] = a; v = parent[w];
    }
  };
  auto augment = [&](int v) {
    while (v != -1) {
      int pv = parent[v], nv = mate[pv];
      mate[v] = pv; mate[pv] = v; v = nv;
    }
  };
  auto bfs = [&](int root) {
    fill(label.begin(), label.end(), -1);
    iota(orig.begin(), orig.end(), 0);
    q.clear();
    label[root] = 0; q.push_back(root);
    for (int i = 0; i < (int)q.size(); ++i) {
      int v = q[i];
      for (auto x : graph[v]) {
        if (label[x] == -1) {
          label[x] = 1; parent[x] = v;
          if (mate[x] == -1)
            return augment(x), 1;
          label[mate[x]] = 0; q.push_back(mate[x]);
        }
        else if (label[x] == 0 && orig[v] != orig[x]) {
          int a = lca(orig[v], orig[x]);
          blossom(x, v, a); blossom(v, x, a);
        }
      }
    }
    return 0;
  };
  // Time halves if you start with (any) maximal
    matching.
  for (int i = 0; i < n; i++)
    if (mate[i] == -1)
      bfs(i);
  return mate;
}
```

2.2 Dinic [72 lines] - b82fa5b3

```
/*.Complexity: O(V^2 E)
  .Call Dinic with total number of nodes.
  .Nodes start from 0.
  .Capacity is long long data.
  .make graph with create edge(u,v,capacity).
  .Get max flow with maxFlow(src,des).*/
#define eb emplace_back
struct Dinic {
  struct Edge {
    int u, v;
    ll cap, flow = 0;
    Edge() {}
    Edge(int u, int v, ll cap) :u(u), v(v), cap(cap) {}
  };
  int N;
  vector<Edge>edge;
  vector<vector<int>>adj;
  vector<int>d, pt;
  Dinic(int N) :N(N), edge(0), adj(N), d(N), pt(N) {}
  void addEdge(int u, int v, ll cap) {
    if (u == v) return;
    edge.eb(u, v, cap);
    adj[u].eb(edge.size() - 1);
    edge.eb(v, u, 0);
    adj[v].eb(edge.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 : adj[u]) {
        Edge& e = edge[k];
        if (e.flow<e.cap && 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 || flow == 0) return flow;
    for (int& i = pt[u];i < adj[u].size();i++) {
      Edge& e = edge[adj[u][i]];
      Edge& oe = edge[adj[u][i] ^ 1];
      if (d[e.v] == d[e.u] + 1) {
        ll amt = e.cap - e.flow;
        if (flow != -1 && 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;
  }

};
```

2.3 Flow [6 lines] - cffa5276

```
Covering Problems:
> Maximum Independent Set(Bipartite): Largest set of
    nodes which do not have any edge between them. sol:
    V-(MaxMatching)
> Minimum Vertex Cover(Bipartite): -Smallest set of
    nodes to cover all the edges -sol: MaxMatching
> Minimum Edge Cover(General graph): -Smallest set of
    edges to cover all the nodes -sol: V-(MaxMatching)
    (if edge cover exists, does not exit for isolated
    nodes)
> Minimum Path Cover(Vertex disjoint) DAG: -Minimum
    number of vertex disjoint paths that visit all the
    nodes -sol: make a bipartite graph using same nodes
    in two sides, one side is "from" other is "to", add
    edges from "from" to "to", then ans is
    V-(MaxMatching)
> Minimum Path Cover(Vertex Not Disjoint) General graph:
    -Minimum number of paths that visit all the nodes
    -sol: consider cycles as nodes then it will become a
    path cover problem with vertex disjoint on DAG
```

2.4 HopCroftKarp [67 lines] - bb94f51d

```
/*.Finds Maximum Matching In a bipartite graph
  .Complexity £O(E\sqrt V)£
  .1-indexed
  .No default constructor
  .add single edge for (u, v)*/
struct HK {
  static const int inf = 1e9;
  int n;
  vector<int>matchL, matchR, dist;
  //matchL contains value of matched node for L part.
  vector<vector<int>>adj;
  HK(int n) :n(n), matchL(n + 1),
  matchR(n + 1), dist(n + 1), adj(n + 1) {
  }

  void addEdge(int u, int v) {
    adj[u].push_back(v);
  }
  bool bfs() {
    queue<int>q;
    for (int u = 1;u <= n;u++) {
      if (!matchL[u]) {
        dist[u] = 0;
        q.push(u);
      }
      else dist[u] = inf;
    }
    dist[0] = inf;///unmatched node matches with 0.
    while (!q.empty()) {
      int u = q.front();
      q.pop();
      for (auto v : adj[u]) {
        if (dist[matchR[v]] == inf) {
          dist[matchR[v]] = dist[u] + 1;
          q.push(matchR[v]);
        }
      }
    }
    return dist[0] != inf;
  }

  bool dfs(int u) {
    if (!u) return true;
    for (auto v : adj[u]) {
      if (dist[matchR[v]] == dist[u] + 1
          && dfs(matchR[v])) {
        matchL[u] = v;
        matchR[v] = u;
        return true;
      }
    }
    dist[u] = inf;
    return false;
  }
  int max_match() {
    int matching = 0;
    while (bfs()) {
      for (int u = 1;u <= n;u++) {
        if (!matchL[u])
          if (dfs(u))
            matching++;
      }
    }
    return matching;
  }
};
```

2.5 Hungarian [116 lines] - e1054195

```
/* Complexity: O(n^3) but optimized
   It finds minimum cost maximum matching.
   For finding maximum cost maximum matching
   add -cost and return -matching()
   1-indexed */
struct Hungarian {
  long long c[N][N], fx[N], fy[N], d[N];
  int l[N], r[N], arg[N], trace[N];
  queue<int> q;
  int start, finish, n;
  const long long inf = 1e18;
  Hungarian() {}
  Hungarian(int n1, int n2) : n(max(n1, n2)) {
    for (int i = 1; i <= n; ++i) {
      fy[i] = l[i] = r[i] = 0;
      for (int j = 1; j <= n; ++j) c[i][j] = inf;
    }
  }
  void add_edge(int u, int v, long long cost) {
    c[u][v] = min(c[u][v], cost);
  }
  inline long long getC(int u, int v) {
    return c[u][v] - fx[u] - fy[v];
  }
  void initBFS() {
    while (!q.empty()) q.pop();
    q.push(start);
```

```
  for (int i = 0; i <= n; ++i) trace[i] = 0;
  for (int v = 1; v <= n; ++v) {
    d[v] = getC(start, v);
    arg[v] = start;
  }
  finish = 0;
}
void findAugPath() {
  while (!q.empty()) {
    int u = q.front();
    q.pop();
    for (int v = 1; v <= n; ++v) if (!trace[v]) {
      long long w = getC(u, v);
      if (!w) {
        trace[v] = u;
        if (!r[v]) {
          finish = v;
          return;
        }
        q.push(r[v]);
      }
      if (d[v] > w) {
        d[v] = w;
        arg[v] = u;
      }
    }
  }
}
void subX_addY() {
  long long delta = inf;
  for (int v = 1; v <= n; ++v) if (trace[v] == 0 &&
  d[v] < delta) {
    delta = d[v];
  }
  // Rotate
  fx[start] += delta;
  for (int v = 1; v <= n; ++v) if (trace[v]) {
    int u = r[v];
    fy[v] -= delta;
    fx[u] += delta;
  }
  else d[v] -= delta;
  for (int v = 1; v <= n; ++v) if (!trace[v] && !d[v])
  {
    trace[v] = arg[v];
    if (!r[v]) {
      finish = v;
      return;
    }
    q.push(r[v]);
  }
}
void Enlarge() {
  do {
    int u = trace[finish];
    int nxt = l[u];
    l[u] = finish;
    r[finish] = u;
    finish = nxt;
  } while (finish);
}
long long maximum_matching() {
  for (int u = 1; u <= n; ++u) {
    fx[u] = c[u][1];
    for (int v = 1; v <= n; ++v) {
      fx[u] = min(fx[u], c[u][v]);
    }
  }
  for (int v = 1; v <= n; ++v) {
    fy[v] = c[1][v] - fx[1];
    for (int u = 1; u <= n; ++u) {
      fy[v] = min(fy[v], c[u][v] - fx[u]);
    }
  }
  for (int u = 1; u <= n; ++u) {
    start = u;
    initBFS();
    while (!finish) {
      findAugPath();
      if (!finish) subX_addY();
    }
    Enlarge();
  }
  long long ans = 0;
  for (int i = 1; i <= n; ++i) {
    if (c[i][l[i]] != inf) ans += c[i][l[i]];
    else l[i] = 0;
  }
  return ans;
 }
};
```

2.6 MCMF [116 lines] - 24fe500f

```
/*Credit: ShahjalalShohag
  .Works for both directed, undirected and with negative
    cost too
  .doesn't work for negative cycles
  .for undirected edges just make the directed flag
    false
  .Complexity: O(min(E^2 *V log V, E logV * flow))*/
using T = long long;
const T inf = 1LL << 61;
struct MCMF {
  struct edge {
    int u, v;
    T cap, cost;
    int id;
    edge(int _u, int _v, T _cap, T _cost, int _id) {
      u = _u;
      v = _v;
      cap = _cap;
      cost = _cost;
      id = _id;
    }
  };
  int n, s, t, mxid;
  T flow, cost;
  vector<vector<int>> g;
  vector<edge> e;
  vector<T> d, potential, flow_through;
  vector<int> par;
  bool neg;
  MCMF() {}
  MCMF(int _n) { // 0-based indexing
    n = _n + 10;
    g.assign(n, vector<int>());
    neg = false;
    mxid = 0;
  }
  void add_edge(int u, int v, T cap, T cost, int id =
    -1, bool directed = true) {
    if (cost < 0) neg = true;
    g[u].push_back(e.size());
    e.push_back(edge(u, v, cap, cost, id));
    g[v].push_back(e.size());
    e.push_back(edge(v, u, 0, -cost, -1));
    mxid = max(mxid, id);
    if (!directed) add_edge(v, u, cap, cost, -1, true);
  }
  bool dijkstra() {
    par.assign(n, -1);
    d.assign(n, inf);
    priority_queue<pair<T, T>, vector<pair<T, T>>,
    greater<pair<T, T>> > q;
    d[s] = 0;
    q.push(pair<T, T>(0, s));
    while (!q.empty()) {
      int u = q.top().second;
      T nw = q.top().first;
      q.pop();
      if (nw != d[u]) continue;
      for (int i = 0; i < (int)g[u].size(); i++) {
        int id = g[u][i];
        int v = e[id].v;
        T cap = e[id].cap;
        T w = e[id].cost + potential[u] - potential[v];
        if (d[u] + w < d[v] && cap > 0) {
          d[v] = d[u] + w;
          par[v] = id;
          q.push(pair<T, T>(d[v], v));
        }
      }
    }
    for (int i = 0; i < n; i++) { // update potential
      if (d[i] < inf) potential[i] += d[i];
    }
    return d[t] != inf;
  }
  T send_flow(int v, T cur) {
    if (par[v] == -1) return cur;
    int id = par[v];
    int u = e[id].u;
    T w = e[id].cost;
    T f = send_flow(u, min(cur, e[id].cap));
    cost += f * w;
    e[id].cap -= f;
    e[id ^ 1].cap += f;
    return f;
  }
  //returns {maxflow, mincost}
  pair<T, T> solve(int _s, int _t, T goal = inf) {
    s = _s;
    t = _t;
    flow = 0, cost = 0;
    potential.assign(n, 0);
    if (neg) {
      // run Bellman-Ford to find starting potential
      d.assign(n, inf);
      for (int i = 0, relax = true; i < n && relax; i++)
    {
        for (int u = 0; u < n; u++) {
          for (int k = 0; k < (int)g[u].size(); k++) {
```

```
            int id = g[u][k];
            int v = e[id].v;
            T cap = e[id].cap, w = e[id].cost;
            if (d[v] > d[u] + w && cap > 0) {
              d[v] = d[u] + w;
              relax = true;
            }
          }
        }
      }
      for (int i = 0; i < n; i++) if (d[i] < inf)
    potential[i] = d[i];
    }
    while (flow < goal && dijkstra()) flow +=
    send_flow(t, goal - flow);
    flow_through.assign(mxid + 10, 0);
    for (int u = 0; u < n; u++) {
      for (auto v : g[u]) {
        if (e[v].id >= 0) flow_through[e[v].id] = e[v ^
    1].cap;
      }
    }
    return make_pair(flow, cost);
  }
};
```

2.7 maxflow [59 lines] - 27b6b6bb

```
ll g[N][N];
int pre[N];
ll val[N];
//Edmonds{Karp BFS - ekbfs
void ekbfs(int st,int n)
{
    val[st]=inf;
    queue<int>str;
    str.push(st);
    while(!str.empty())
    {
        st=str.front();
        str.pop();
        for(int j=1;j<=n;j++)
        {
            if(g[st][j]>0&&pre[j]==0)
            {
                val[j]=min(g[st][j],val[st]);
                pre[j]=st;
                str.push(j);
            }
        }
    }
}
ll maxflow(int st,int ed,int n)
{
    ll sum=0;
    while(1)
    {
        set1(pre,n,0);
        ekbfs(st,n);
        if(pre[ed]==0)break;
        sum+=val[ed];
        int cur=ed;
        while(cur!=st)
        {
            g[pre[cur]][cur]-=val[ed];
            g[cur][pre[cur]]+=val[ed];
            cur=pre[cur];
        }
    }
    return sum;
}
//main
int main()
{
    int n,m;
    cin>>n>>m;
    int st,ed;
    cin>>st>>ed;
    set2(g,n,n,0);
    for(int i=0;i<m;i++)
    {
        int x,y,w;
        cin>>x>>y>>w;
        g[x][y]=w;
    }
    cout<<maxflow(st,ed,n);
}
```

3 string

3.1 Dynamic suffix array [179 lines] - f5cc9ffa

```
struct dyn_sa {
  struct node {
    int sa, lcp;
    node *l, *r, *p;
    int sz, mi;
    node(int sa_, int lcp_, node* p_) : sa(sa_),
    lcp(lcp_),
      l(NULL), r(NULL), p(p_), sz(1), mi(lcp) {}
    void update() {
      sz = 1, mi = lcp;
      if (l) sz += l->sz, mi = min(mi, l->mi);
      if (r) sz += r->sz, mi = min(mi, r->mi);
    }
  };

  node* root;
  vector<ll> tag; // tag of a suffix (reversed id)
  string s; // reversed

  dyn_sa() : root(NULL) {}
  dyn_sa(string s_) : dyn_sa() {
    reverse(s_.begin(), s_.end());
    for (char c : s_) push_front(c);
  }
  ~dyn_sa() {
    vector<node*> q = {root};
    while (q.size()) {
      node* x = q.back(); q.pop_back();
      if (!x) continue;
      q.push_back(x->l), q.push_back(x->r);
      delete x;
    }
  }

  int size(node* x) { return x ? x->sz : 0; }
  int mirror(int i) { return s.size()-1 - i; }
  bool cmp(int i, int j) {
    if (s[i] != s[j]) return s[i] < s[j];
    if (i == 0 or j == 0) return i < j;
    return tag[i-1] < tag[j-1];
  }
  void fix_path(node* x) { while (x) x->update(), x =
    x->p; }
  void flatten(vector<node*>& v, node* x) {
    if (!x) return;
    flatten(v, x->l);
    v.push_back(x);
    flatten(v, x->r);
  }
  void build(vector<node*>& v, node*& x, node* p, int
    L, int R, ll l, ll r) {
    if (L > R) return void(x = NULL);
    int M = (L+R)/2;
    ll m = (l+r)/2;
    x = v[M];
    x->p = p;
    tag[x->sa] = m;
    build(v, x->l, x, L, M-1, l, m-1), build(v, x->r,
    x, M+1, R, m+1, r);
    x->update();
  }
  void fix(node*& x, node* p, ll l, ll r) {
    if (3*max(size(x->l), size(x->r)) <= 2*size(x))
    return x->update();
    vector<node*> v;
    flatten(v, x);
    build(v, x, p, 0, v.size()-1, l, r);
  }
  node* next(node* x) {
    if (x->r) {
      x = x->r;
      while (x->l) x = x->l;
      return x;
    }
    while (x->p and x->p->r == x) x = x->p;
    return x->p;
  }
  node* prev(node* x) {
    if (x->l) {
      x = x->l;
      while (x->r) x = x->r;
      return x;
    }
    while (x->p and x->p->l == x) x = x->p;
    return x->p;
  }

  int get_lcp(node* x, node* y) {
    if (!x or !y) return 0; // change defaut value here
    if (s[x->sa] != s[y->sa]) return 0;
    if (x->sa == 0 or y->sa == 0) return 1;
    return 1 + query(mirror(x->sa-1), mirror(y->sa-1));
  }
  void add_suf(node*& x, node* p, int id, ll l, ll r) {
    if (!x) {
      x = new node(id, 0, p);
      node *prv = prev(x), *nxt = next(x);
      int lcp_cur = get_lcp(prv, x), lcp_nxt =
    get_lcp(x, nxt);
      if (nxt) nxt->lcp = lcp_nxt, fix_path(nxt);
      x->lcp = lcp_cur;
      tag[id] = (l+r)/2;
      x->update();
```

```
    return;
  }
  if (cmp(id, x->sa)) add_suf(x->l, x, id, l,
  tag[x->sa]-1);
  else add_suf(x->r, x, id, tag[x->sa]+1, r);
  fix(x, p, l, r);
}
void push_front(char c) {
  s += c;
  tag.push_back(-1);
  add_suf(root, NULL, s.size() - 1, 0, 1e18);
}

void rem_suf(node*& x, int id) {
  if (x->sa != id) {
    if (tag[id] < tag[x->sa]) return rem_suf(x->l,
  id);
    return rem_suf(x->r, id);
  }
  node* nxt = next(x);
  if (nxt) nxt->lcp = min(nxt->lcp, x->lcp),
  fix_path(nxt);

  node *p = x->p, *tmp = x;
  if (!x->l or !x->r) {
    x = x->l ? x->l : x->r;
    if (x) x->p = p;
  } else {
    for (tmp = x->l, p = x; tmp->r; tmp = tmp->r) p =
  tmp;
    x->sa = tmp->sa, x->lcp = tmp->lcp;
    if (tmp->l) tmp->l->p = p;
    if (p->l == tmp) p->l = tmp->l;
    else p->r = tmp->l;
  }
  fix_path(p);
  delete tmp;
}
void pop_front() {
  if (!s.size()) return;
  s.pop_back();
  rem_suf(root, s.size());
  tag.pop_back();
}

int query(node* x, ll l, ll r, ll a, ll b) {
  if (!x or tag[x->sa] == -1 or r < a or b < l) return
  s.size();
  if (a <= l and r <= b) return x->mi;
  int ans = s.size();
  if (a <= tag[x->sa] and tag[x->sa] <= b) ans =
  min(ans, x->lcp);
  ans = min(ans, query(x->l, l, tag[x->sa]-1, a, b));
  ans = min(ans, query(x->r, tag[x->sa]+1, r, a, b));
  return ans;
}
int query(int i, int j) { // lcp(s[i..], s[j..])
  if (i == j) return s.size() - i;
  ll a = tag[mirror(i)], b = tag[mirror(j)];
  int ret = query(root, 0, 1e18, min(a, b)+1, max(a,
  b));
  return ret;
}
```

```
  // optional: get isa[i], sa[i] and lcp[i] (standard
    index representation)
  int isa(int i) {
    i = mirror(i);
    node* x = root;
    int ret = 0;
    while (x) {
      if (tag[x->sa] < tag[i]) {
        ret += size(x->l)+1;
        x = x->r;
      } else x = x->l;
    }
    return ret;
  }
  // returns a pair with SA[i] and lcp[i]
  pair<int, int> operator[](int i) {
    node* x = root;
    while (1) {
      if (i < size(x->l)) x = x->l;
      else {
        i -= size(x->l);
        if (!i) return {mirror(x->sa), x->lcp};
        i--, x = x->r;
      }
    }
  }
};
```

3.2 customHash [16 lines] - 803dc8a8

```
struct custom_hash {
    static uint64_t splitmix64(uint64_t x) {
        // http://xorshift.di.unimi.it/splitmix64.c
        x += 0x9e3779b97f4a7c15;
        x = (x ^ (x >> 30)) * 0xbf58476d1ce4e5b9;
        x = (x ^ (x >> 27)) * 0x94d049bb133111eb;
        return x ^ (x >> 31);
    }

    size_t operator()(uint64_t x) const {
        static const uint64_t FIXED_RANDOM =
    chrono::steady_clock::now().time_since_epoch().count();

        return splitmix64(x + FIXED_RANDOM);
    }
};
unordered_map<long long, int, custom_hash> safe_map;
gp_hash_table<long long, int, custom_hash>
    safe_hash_table;
```

3.3 hashing [53 lines] - 5e722c99

```
struct SimpleHash
{
    ll len;
    ll base, mod;
    vector<ll> P, H, R;
    SimpleHash(){};
    SimpleHash(string str, ll b, ll m)
    {
        base = b, mod = m, len = str.size();
        P.resize(len + 4, 1), H.resize(len + 3, 0),
    R.resize(len + 3, 0);
        for (ll i = 1; i <= len + 3; i++)
        {
            P[i] = (P[i - 1] * base) % mod;
        }
        for (ll i = 1; i <= len; i++)
        {
            H[i] = (H[i - 1] * base + str[i - 1] + 1007)
    % mod;
        }
        for (ll i = len; i >= 1; i--)
        {
            R[i] = (R[i + 1] * base + str[i - 1] + 1007)
    % mod;
        }
    }
    inline ll range_hash(ll l, ll r)
    {
        ll hashval = H[r + 1] - ((ll)P[r - l + 1] * H[l]
    % mod);
        return (hashval < 0 ? hashval + mod : hashval);
    }
    inline ll reverse_hash(ll l, ll r)
    {
        ll hashval = R[l + 1] - ((ll)P[r - l + 1] * R[r
    + 2] % mod);
        return (hashval < 0 ? hashval + mod : hashval);
    }
};
struct DoubleHash
{
    SimpleHash sh1, sh2;
    DoubleHash() {}
    DoubleHash(string str)
    {
        sh1 = SimpleHash(str, 1949313259, 2091573227);
        sh2 = SimpleHash(str, 1997293877, 2117566807);
    }

    inline long long range_hash(ll l, ll r)
    {
        return ((long long)sh1.range_hash(l, r) << 32) ^
    sh2.range_hash(l, r);
    }
    inline long long reverse_hash(ll l, ll r)
    {
        return ((long long)sh1.reverse_hash(l, r) << 32)
    ^ sh2.reverse_hash(l, r);
    }
};
```

3.4 kmp [13 lines] - 040d082c

```
vector<int> KMP(string s) {
    int n = (int)s.length();
    vector<int> pi(n);
    for (int i = 1; i < n; i++) {
        int j = pi[i-1];
        while (j > 0 && s[i] != s[j])
            j = pi[j-1];
        if (s[i] == s[j])
            j++;
        pi[i] = j;
    }
    return pi;
}
```

3.5 mancharsAlgorithm [32 lines] - 663d53a4

```
int p[2*N];
void manacher (string s)
{
```

```
    int n=s.size();
    char ss[2*n+2];
    ss[2*n+1]='\0';
    for(int i=0; i<2*n+1; i++)
    {
        p[i]=0;
        if(i%2==0)ss[i]='#';
        else ss[i]=s[i/2];
    }
    int l = 0, r = 0;
    for(int i = 0; i <=2*n; i++)
    {
        p[i] = max(0, min(r - i, p[l + (r - i)]));
        while((p[i]+i<=2*n)&&(i-p[i]>=0)&&(ss[i - p[i]]
    == ss[i + p[i]]))
        {
            p[i]++;
        }
        if(i + p[i] > r)
        {
            l = i - p[i], r = i + p[i];
        }
    }
}
bool check(int l,int r)
{
    return p[l+r+1]>r-l;
}
```

3.6 suffix$_array$[98$_{lines}$] – c56e57cb

```
class SuffixArray {
private:
    string s;
    int n;
    vector<int> p, c;
    vector<int> lcp;

    vector<int> sort_cyclic_shifts() {
        const int alphabet = 256;
        vector<int> p(n), c(n), cnt(max(alphabet, n),
    0);

        // Count sort for the first character
        for(int i = 0; i < n; i++) cnt[s[i]]++;
        for(int i = 1; i < alphabet; i++) cnt[i] +=
    cnt[i-1];
        for(int i = 0; i < n; i++) p[--cnt[s[i]]] = i;

        c[p[0]] = 0;
        int classes = 1;
        for(int i = 1; i < n; i++) {
            if(s[p[i]] != s[p[i-1]]) classes++;
            c[p[i]] = classes - 1;
        }

        // Iterate through powers of 2
        vector<int> pn(n), cn(n);
        for(int h = 0; (1 << h) < n; h++) {
            for(int i = 0; i < n; i++) {
                pn[i] = p[i] - (1 << h);
                if(pn[i] < 0) pn[i] += n;
            }
            fill(cnt.begin(), cnt.begin() + classes, 0);
            for(int i = 0; i < n; i++) cnt[c[pn[i]]]++;
            for(int i = 1; i < classes; i++) cnt[i] +=
    cnt[i-1];
            for(int i = n-1; i >= 0; i--)
    p[--cnt[c[pn[i]]]] = pn[i];

            cn[p[0]] = 0;
            classes = 1;
            for(int i = 1; i < n; i++) {
                pair<int,int> cur = {c[p[i]], c[(p[i] +
    (1 << h)) % n]};
                pair<int,int> prev = {c[p[i-1]],
    c[(p[i-1] + (1 << h)) % n]};
                if(cur != prev) ++classes;
                cn[p[i]] = classes - 1;
            }
            c.swap(cn);
        }
        return p;
    }

    int cmp(int i, string &pat) {
        for(int j = i; j < n - 1; j++) {
            if(j - i == pat.size()) return 0;
            if(s[j] > pat[j - i]) return 1;
            if(s[j] < pat[j - i]) return -1;
        }
        if(pat.size() == (n - i - 1)) return 0;
        return -1;
    }

public:
    SuffixArray() {}

    void build(string &input) {
        s = input;
        n = s.size();
        p = sort_cyclic_shifts();
        c.resize(n);
        lcp.resize(n);

        for(int i = 0; i < n; i++) c[p[i]] = i;
        int k = 0;
        for(int i = 0 ; i < n - 1 ; i++){ //0123£
            int pi = c[i];
            int j = p[pi - 1];
            while(s[i + k] == s[j + k])k++;
            lcp[pi] = k;
            k = max(0 , k - 1);
        }
        // for(int i = 1 ; i < n ; i++){
        //     debug(i , s.substr(p[i]) ,lcp[i]);
        // }
    }

    int upper_bound(int lo, int hi, string &pat) {
        if(lo == hi) return lo;
        int mid = (lo + hi) / 2;
        int cc = cmp(p[mid], pat);
        if(cc > 0) return upper_bound(lo, mid, pat);
        return upper_bound(mid + 1, hi, pat);
    }

    int lower_bound(int lo, int hi, string &pat) {
        if(lo == hi) return lo;
        int mid = (lo + hi) / 2;
        if(cmp(p[mid], pat) >= 0) return
    lower_bound(lo, mid, pat);
        return lower_bound(mid + 1, hi, pat);
    }
};
```

3.7 trie [143 lines] - d1f30347

```
#include <iostream>
#include <string>
#include <memory>

class TrieNode {
public:
    std::unique_ptr<TrieNode> children[26];  // Array
    for lowercase English letters
    bool isEndOfWord;

    TrieNode() : isEndOfWord(false) {
        // Initialize all children to nullptr
        for (int i = 0; i < 26; i++) {
            children[i] = nullptr;
        }
    }
};

class Trie {
private:
    std::unique_ptr<TrieNode> root;

    // Helper function for prefix search
    bool hasChildren(TrieNode* node) {
        for (int i = 0; i < 26; i++) {
            if (node->children[i] != nullptr) {
                return true;
            }
        }
        return false;
    }

    // Helper function for deletion
    bool deleteHelper(TrieNode* node, const std::string&
    key, int depth = 0) {
        if (node == nullptr) {
            return false;
        }

        // Base case: if we've processed all characters
        if (depth == key.length()) {
            if (node->isEndOfWord) {
                node->isEndOfWord = false;
                return !hasChildren(node);
            }
            return false;
        }

        int index = key[depth] - 'a';
        if (node->children[index] != nullptr) {
            bool shouldDeleteChild =
    deleteHelper(node->children[index].get(), key, depth
    + 1);
```

```
            if (shouldDeleteChild) {
                node->children[index] = nullptr;
                return !hasChildren(node) &&
    !node->isEndOfWord;
            }
        }
        return false;
    }

public:
    Trie() : root(std::make_unique<TrieNode>()) {}

    // Insert a word into the trie
    void insert(const std::string& word) {
        TrieNode* current = root.get();

        for (char c : word) {
            int index = c - 'a';
            if (current->children[index] == nullptr) {
                current->children[index] =
    std::make_unique<TrieNode>();
            }
            current = current->children[index].get();
        }
        current->isEndOfWord = true;
    }

    // Search for a word in the trie
    bool search(const std::string& word) const {
        TrieNode* node = root.get();

        for (char c : word) {
            int index = c - 'a';
            if (node->children[index] == nullptr) {
                return false;
            }
            node = node->children[index].get();
        }
        return node->isEndOfWord;
    }

    // Check if there is any word that starts with the
    given prefix
    bool startsWith(const std::string& prefix) const {
        TrieNode* node = root.get();

        for (char c : prefix) {
            int index = c - 'a';
            if (node->children[index] == nullptr) {
                return false;
            }
            node = node->children[index].get();
        }
        return true;
    }

    // Delete a word from the trie
    bool remove(const std::string& word) {
        return deleteHelper(root.get(), word);
    }

    // Check if trie is empty
    bool isEmpty() {
        return !hasChildren(root.get());
    }
    }
    TrieNode *getRoot(){
        return root.get();
    }
};

// Example usage
int main() {
    Trie trie;

    // Insert some words
    trie.insert("apple");
    trie.insert("app");
    trie.insert("apricot");

    // Search demonstrations
    std::cout << "Search 'apple': " <<
    (trie.search("apple") ? "Found" : "Not found") <<
    std::endl;
    std::cout << "Search 'app': " << (trie.search("app")
    ? "Found" : "Not found") << std::endl;
    std::cout << "Search 'apt': " << (trie.search("apt")
    ? "Found" : "Not found") << std::endl;

    // Prefix search demonstrations
    std::cout << "Prefix 'app': " <<
    (trie.startsWith("app") ? "Found" : "Not found") <<
    std::endl;
    std::cout << "Prefix 'apt': " <<
    (trie.startsWith("apt") ? "Found" : "Not found") <<
    std::endl;

    // Delete demonstration
    std::cout << "Removing 'apple'" << std::endl;
    trie.remove("apple");
    std::cout << "Search 'apple': " <<
    (trie.search("apple") ? "Found" : "Not found") <<
    std::endl;
    std::cout << "Search 'app': " << (trie.search("app")
    ? "Found" : "Not found") << std::endl;

    return 0;
}
```

4 Geometry

4.1 Rotation Matrix [39 lines] - da59ff1d

```
struct { double x; double y; double z; } Point;
double rMat[4][4];
double inMat[4][1] = {0.0, 0.0, 0.0, 0.0};
double outMat[4][1] = {0.0, 0.0, 0.0, 0.0};
void mulMat() {
  for(int i = 0; i < 4; i++ ){
    for(int j = 0; j < 1; j++){
      outMat[i][j] = 0;
      for(int k = 0; k < 4; k++)
        outMat[i][j] += rMat[i][k] * inMat[k][j];
    }
  }
}
void setMat(double ang, double u, double v, double w){
  double L = (u * u + v * v + w * w);
  ang = ang * PI / 180.0; /*converting to radian
    value*/
  double u2 = u*u; double v2 = v*v; double w2 = w*w;
  rMat[0][0]=(u2+(v2+w2)*cos(ang))/L;
  rMat[0][1]=(u*v*(1-cos(ang))-w*sqrt(L)*sin(ang))/L;
  rMat[0][2]=(u*w*(1-cos(ang))+v*sqrt(L)*sin(ang))/L;
  rMat[0][3]=0.0;
  rMat[1][0]=(u*v*(1-cos(ang))+w*sqrt(L)*sin(ang))/L;
  rMat[1][1]=(v2+(u2+w2)*cos(ang))/L;
  rMat[1][2]=(v*w*(1-cos(ang))-u*sqrt(L)*sin(ang))/L;
  rMat[1][3]=0.0;
  rMat[2][0]=(u*w*(1-cos(ang))-v*sqrt(L)*sin(ang))/L;
  rMat[2][1]=(v*w*(1-cos(ang))+u*sqrt(L)*sin(ang))/L;
  rMat[2][2]=(w2 + (u2 + v2) * cos(ang)) / L;
  rMat[2][3]=0.0; rMat[3][0]=0.0; rMat[3][1]=0.0;
  rMat[3][2]=0.0; rMat[3][3]=1.0;
}
/*double ang;
  double u, v, w; //points = the point to be rotated
  Point point, rotated; //u,v,w=unit vector of line
  inMat[0][0] = points.x; inMat[1][0] = points.y;
  inMat[2][0] = points.z; inMat[3][0] = 1.0;
  setMat(ang, u, v, w); mulMat();
  rotated.x = outMat[0][0]; rotated.y = outMat[1][0];
  rotated.z = outMat[2][0];*/
```

4.2 hull [48 lines] - d6b6f949

```
struct Point {
    ll x, y;
    bool operator<(const Point &other) const {
        if (x != other.x) return x < other.x;
        return y < other.y;
    }
    bool operator==(const Point &other) const {
        return x == other.x && y == other.y;
    }
};

static inline ll dot (const Point &a, const Point &b) {
    return a.x * 1LL * b.x + a.y * 1LL * b.y;
}

static inline ll cross(const Point &a, const Point &b,
    const Point &c) {
    return (b.x - a.x) * 1LL * (c.y - a.y) - (b.y - a.y)
    * 1LL * (c.x - a.x);
}

vector<Point>
    convex_hull_including_collinear(vector<Point> pts) {
    int n = (int)pts.size();
    if (n <= 1) return pts;
    sort(pts.begin(), pts.end());
    vector<Point> lower, upper;

    for (const auto &p : pts) {
        while (lower.size() >= 2 &&
    cross(lower[lower.size()-2], lower.back(), p) < 0)
    lower.pop_back();
        lower.push_back(p);
    }

    for (int i = n - 1; i >= 0; --i) {
        const auto &p = pts[i];
        while (upper.size() >= 2 &&
    cross(upper[upper.size()-2], upper.back(), p) < 0)
    upper.pop_back();
```

```
        upper.push_back(p);
    }

    lower.pop_back(); upper.pop_back();
    lower.insert(lower.end(), upper.begin(),
    upper.end());
    return lower;
}

ll area_of_pts(auto pts){
    ll area = 0 ;
    ll n = pts.size() ;
    for(int i = 0 ; i < n-1 ; i++) area += pts[i].x *
    pts[i+1].y - pts[i+1].x * pts[i].y ;
    area += pts[n-1].x * pts[0].y - pts[0].x *
    pts[n-1].y ;
    return area ;
}
```

4.3 minkowski [44 lines] - 9387db0c

```
struct pt{
    long long x, 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};
    }
    long long cross(const pt & p) const {
        return x * p.y - y * p.x;
    }
};

void reorder_polygon(vector<pt> & P){
    size_t pos = 0;
    for(size_t i = 1; i < P.size(); i++){
        if(P[i].y < P[pos].y || (P[i].y == P[pos].y &&
    P[i].x < P[pos].x))
            pos = i;
    }
    rotate(P.begin(), P.begin() + pos, P.end());
}

vector<pt> minkowski(vector<pt> P, vector<pt> Q){
    // the first vertex must be the lowest
    reorder_polygon(P);
    reorder_polygon(Q);
    // we must ensure cyclic indexing
    P.push_back(P[0]);
    P.push_back(P[1]);
    Q.push_back(Q[0]);
    Q.push_back(Q[1]);
    // main part
    vector<pt> result;
    size_t i = 0, j = 0;
    while(i < P.size() - 2 || j < Q.size() - 2){
        result.push_back(P[i] + Q[j]);
        auto cross = (P[i + 1] - P[i]).cross(Q[j + 1] -
    Q[j]);
        if(cross >= 0 && i < P.size() - 2)
            ++i;
        if(cross <= 0 && j < Q.size() - 2)
            ++j;
    }
    return result;
}
```

5 miscellaneous

5.1 BM [96 lines] - 9c0a4511

```
const int MOD = 998244353;
ll qp(ll a,ll b)
{
  ll x=1; a%=MOD;
  while(b)
  {
    if(b&1) x=x*a%MOD;
    a=a*a%MOD; b>>=1;
  }
  return x;
}
namespace linear_seq {
inline vector<int> BM(vector<int> x)
{
  //ls: (shortest) relation sequence (after filling
    zeroes) so far
  //cur: current relation sequence
  vector<int> ls,cur;
  //lf: the position of ls (t')
  //ld: delta of ls (v')
  int lf,ld;
  for(int i=0;i<int(x.size());++i)
  {
    ll t=0;
    //evaluate at position i
    for(int j=0;j<int(cur.size());++j)
      t=(t+x[i-j-1]*(ll)cur[j])%MOD;
    if((t-x[i])%MOD==0) continue; //good so far
    //first non-zero position
    if(!cur.size())
    {
      cur.resize(i+1);
      lf=i; ld=(t-x[i])%MOD;
      continue;
    }
    //cur=cur-c/ld*(x[i]-t)
    ll k=-(x[i]-t)*qp(ld,MOD-2)%MOD/*1/ld*/;
    vector<int> c(i-lf-1); //add zeroes in front
    c.pb(k);
    for(int j=0;j<int(ls.size());++j)
      c.pb(-ls[j]*k%MOD);
    if(c.size()<cur.size()) c.resize(cur.size());
    for(int j=0;j<int(cur.size());++j)
      c[j]=(c[j]+cur[j])%MOD;
    //if cur is better than ls, change ls to cur
    if(i-lf+(int)ls.size()>=(int)cur.size())
      ls=cur,lf=i,ld=(t-x[i])%MOD;
    cur=c;
  }
  for(int i=0;i<int(cur.size());++i)
    cur[i]=(cur[i]%MOD+MOD)%MOD;
  return cur;
}
int m; //length of recurrence
//a: first terms
//h: relation
ll a[SZ],h[SZ],t_[SZ],s[SZ],t[SZ];
//calculate p*q mod f
inline void mull(ll*p,ll*q)
{
  for(int i=0;i<m+m;++i) t_[i]=0;
  for(int i=0;i<m;++i) if(p[i])
    for(int j=0;j<m;++j)
      t_[i+j]=(t_[i+j]+p[i]*q[j])%MOD;
  for(int i=m+m-1;i>=m;--i) if(t_[i])
    //miuns t_[i]x^{i-m}(x^m-\sum_{j=0}^{m-1}
    x^{m-j-1}h_j)
    for(int j=m-1;~j;--j)
      t_[i-j-1]=(t_[i-j-1]+t_[i]*h[j])%MOD;
  for(int i=0;i<m;++i) p[i]=t_[i];
}
inline ll calc(ll K)
{
  for(int i=m;~i;--i)
    s[i]=t[i]=0;
  //init
  s[0]=1; if(m!=1) t[1]=1; else t[0]=h[0];
  //binary-exponentiation
  while(K)
  {
    if(K&1) mull(s,t);
    mull(t,t); K>>=1;
  }
  ll su=0;
  for(int i=0;i<m;++i) su=(su+s[i]*a[i])%MOD;
  return (su%MOD+MOD)%MOD;
}
inline int work(vector<int> x,ll n)
{
  if(n<int(x.size())) return x[n];
  vector<int> v=BM(x); m=v.size(); if(!m) return 0;
  for(int i=0;i<m;++i) h[i]=v[i],a[i]=x[i];
  return calc(n);
}
}
using linear_seq::work;
//cout<<work({1,1,2,3,5,8,13,21},10)<<"\n";
```

5.2 Sqrt Tricks [8 lines] - 8f8b610b

1. Size of the block is not always Sqrt, adjust it as necessary. if o(n/b+b) then take n/b = b and calculate b.
2. MO's Algorithm
   *it is possible to solve a Mo problem without any remove operation. For L in one block R only increases, for every range we can start L from the last of that block
3. Sqrt Decomposition by time of queries.
   *keep overall solution and sqrt(n) updates in a vector and for a query iterate over all of them, when the vector size exceeds sqrt(n) you can add these updates with overall solution using o(n)
4. If sum of N positive numbers are S, there are at most sqrt(S) distinct values.
5. Randomization
6. Baby step, gaint step

5.3 XOR$_{VECTOR_{BASIS}}$[31lines] – e8d99179

```
#include <bits/stdc++.h>

using namespace std;
```

```
const int N = 1e5 + 10, LOG_A = 20;

int basis[LOG_A];

void insertVector(int mask) {
  for (int i = LOG_A - 1; i >= 0; i--) {
    if ((mask & 1 << i) == 0) continue;

    if (!basis[i]) {
      basis[i] = mask;
      return;
    }

    mask ^= basis[i];
  }
}

int main() {
  for (int i = LOG_A - 1; i >= 0; i--) {
    if (!basis[i]) continue;

    if (ans & 1 << i) continue;

    ans ^= basis[i];
  }
}
```

5.4 debug [37 lines] - 299652fb

```
#include "bits/stdc++.h"
#pragma GCC optimize ("O3")
#pragma GCC target ("sse4")
using namespace std;
mt19937
    rng(chrono::steady_clock::now().time_since_epoch().count());

#define uid(a, b) uniform_int_distribution<ll>(a,
    b)(rng)
#define tc ll tt, qq=0; cin>>tt; while(qq++<tt)
#define cs cout<<"Case "<<qq<<": "
#define csl cout<<"Case "<<qq<<"\n"
#define uniq(vec) vec.resize(distance(vec.begin(),
    unique(vec.begin(),vec.end())))
typedef long long ll;
#define all(x) x.begin() , x.end()
void __print(int x) {cerr << x;}
void __print(long x) {cerr << x;}
void __print(long long x) {cerr << x;}
void __print(unsigned x) {cerr << x;}
void __print(unsigned long x) {cerr << x;}
void __print(unsigned long long x) {cerr << x;}
void __print(float x) {cerr << x;}
void __print(double x) {cerr << x;}
void __print(long double x) {cerr << x;}
void __print(char x) {cerr << '\'' << x << '\'';}
void __print(const char *x) {cerr << '\"' << x << '\"';}
void __print(const string &x) {cerr << '\"' << x <<
    '\"';}
void __print(bool x) {cerr << (x ? "true" : "false");}
template<typename T, typename V>
void __print(const pair<T, V> &x) {cerr << '{';
    __print(x.first); cerr << ','; __print(x.second);
    cerr << '}';}
template<typename T>
void __print(const T &x) {int f = 0; cerr << '{'; for
    (auto &i: x) cerr << (f++ ? "," : ""), __print(i);
    cerr << "}";}
void _print() {cerr << "]\n";}
template <typename T, typename... V>
void _print(T t, V... v) {__print(t); if (sizeof...(v))
    cerr << ", "; _print(v...);}
#ifndef ONLINE_JUDGE
#define debug(x...) cerr << "[" << #x << "] = [";
    _print(x)
#else
#define debug(x...)
#endif
```

5.5 floyed [14 lines] - 96e2562f

```
ll dp[N][N];
void floyed(int n)
{
    for(int k=1; k<=n; k++)
    {
        for(int i=1; i<=n; i++)
        {
            for(int j=1; j<=n; j++)
            {

    dp[i][j]=min(dp[i][j],dp[i][k]+dp[k][j]);
            }
        }
    }
}
```

5.6 geometry [18 lines] - 905dd994

```
bool is_cross(ld a,ld b,ld c,int x1,int y1,int x2,int
    y2)
{
    ld f1=a*x1+b*y1+c;
    ld f2=a*x2+b*y2+c;
    if(f1<0&&f2<0)return 0;
    if(f1>0&&f2>0)return 0;
    return 1;
}
pair<ld,ld>cross_point(ld a ,ld b,ld c,ld aa,ld bb,ll
    cc)
{
    ld xx=(cc*b/bb-c)/(a-aa*b/bb);
    ld yy=(-c-a*xx)/b;3
    return {xx,yy};
}
ld dis(ld x1,ld y1,ld x2,ld y2)
{
    return ((x1-x2)*(x1-x2)+(y1-y2)*(y1-y2));
}
```

5.7 hopcroftKarp [75 lines] - bdd603f5

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

const int N = 3e5 + 9;

struct HopcroftKarp {
  static const int inf = 1e9;
  int n;
  vector<int> l, r, d;
  vector<vector<int>> g;
  HopcroftKarp(int _n, int _m) {
    n = _n;
    int p = _n + _m + 1;
    // n = no of nodes, m = extension for v + n;
    g.resize(p);
    l.resize(p, 0);
    r.resize(p, 0);
    d.resize(p, 0);
  }
  void add_edge(int u, int v) {
    g[u].push_back(v + n); //right id is increased by n,
    so is l[u]
  }
  bool bfs() {
    queue<int> q;
    for (int u = 1; u <= n; u++) {
      if (!l[u]) d[u] = 0, q.push(u);
      else d[u] = inf;
    }
    d[0] = inf;
    while (!q.empty()) {
      int u = q.front();
      q.pop();
      for (auto v : g[u]) {
        if (d[r[v]] == inf) {
          d[r[v]] = d[u] + 1;
          q.push(r[v]);
        }
      }
    }
    return d[0] != inf;
  }
  bool dfs(int u) {
    if (!u) return true;
    for (auto v : g[u]) {
      if(d[r[v]] == d[u] + 1 && dfs(r[v])) {
        l[u] = v;
        r[v] = u;
        return true;
      }
    }
    d[u] = inf;
    return false;
  }
  int maximum_matching() {
    int ans = 0;
    while (bfs()) {
      for(int u = 1; u <= n; u++) if (!l[u] && dfs(u))
    ans++;
    }
    return ans;
  }
};
int32_t main() {
  ios_base::sync_with_stdio(0);
  cin.tie(0);
  int n, m, q;
  cin >> n >> m >> q;
  HopcroftKarp M(n, m);
  while (q--) {
    int u, v;
    cin >> u >> v;
    M.add_edge(u, v);
  }
  cout << M.maximum_matching() << '\n';
```

```
  return 0;
}
```

5.8 inverseMatrix [95 lines] - 562baf66

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

vector<vector<int>> find_cofactor(vector<vector<int>>
    a, int r, int c)
{
    vector<vector<int>> mat;
    int n = a.size();
    for (int i = 0; i < n; i++) {
        vector<int> row;
        if (r == i) continue;
        for (int j = 0; j < n; j++) {
            if (j == c) continue;
            row.push_back(a[i][j]);
        }
        mat.push_back(row);
    }
    return mat;
}

int determinant(vector<vector<int>> a)
{
    if (a.size() == 1) {
        return a[0][0];
    }
    int n = a.size();
    int sign = +1;
    int det = 0;
    for (int i = 0; i < n; i++) {
        vector<vector<int>> cf_mat = find_cofactor(a,
    0, i);
        int cofactor = determinant(cf_mat);
        det += cofactor * sign * a[0][i];
        sign = -sign;
    }
    return det;
}

vector<vector<int>> transpose(vector<vector<int>> a)
{
    int n = a.size();
    vector<vector<int>> res(n, vector<int>(n));
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            res[i][j] = a[j][i];
        }
    }
    return res;
}

vector<vector<double>> find_inverse(vector<vector<int>>
    a)
{
    int n = a.size();
    int det = determinant(a);
    if (det == 0) {
        cout << "Inverse Impossible\n";
        return {{}};
    }
    vector<vector<int>> cofactor_matrix(n,
    vector<int>(n));
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            int sign = (i + j)%2 ? -1 : +1;
            cofactor_matrix[i][j] = sign *
    determinant(find_cofactor(a, i, j));
        }
    }
    auto adj_matrix = transpose(cofactor_matrix);
    auto inverse_mat = vector<vector<double>>(n,
    vector<double>(n));
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            inverse_mat[i][j] = 1.0 * adj_matrix[i][j] /
    det;
        }
    }
    return inverse_mat;
}

signed main()
{
    vector<vector<int>> mat = {
        {2, -3, 1},
        {2, 0, -1},
        {1, 4, 5}
    };

    int det = determinant(mat);

    cout << det << "\n";

    auto inverse = find_inverse(mat);

    int n = inverse.size();

    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            cout << inverse[i][j] << " \n"[j + 1 == n];
        }
    }
}
```

5.9 maxXor [68 lines] - 383bfa4d

```
class trie
{
public:
    trie * next[2];
    int val,co;
    trie()
    {
        next[0]=next[1]=0;
        co=val=0;
    }
};
void insertt(trie * root, int val)
{
    trie* cur=root;
    cur->co++;
    for(int i=31;i>=0;i--)
    {
        bool bit=((1<<i)&val);
        if(cur->next[bit]==NULL)
        {
            cur->next[bit]=new trie();
        }
        cur=cur->next[bit];
        cur->co++;
    }
    cur->val=val;
}
void delnode(trie *root)
{
    if(root->next[0]!=NULL)delnode(root->next[0]);
    if(root->next[1]!=NULL)delnode(root->next[1]);
    delete root;
}
void delett(trie * root, int val)
{
    trie* cur=root;
    cur->co--;
    for(int i=31;i>=0;i--)
    {
        bool bit=((1<<i)&val);
        if(cur->next[bit]->co==1)
        {
            delnode(cur->next[bit]);
            cur->next[bit]=NULL;
            return ;
        }
        cur=cur->next[bit];
        cur->co--;
    }
}
int max_xor(trie * root, int val)
{
    trie* cur=root;
    if(cur->co==0)return -1;
    for(int i=31;i>=0;i--)
    {
        bool bit=((1<<i)&val);
        if(cur->next[1-bit]!=NULL)
        {
           cur=cur->next[1-bit];
        }else
        {
            cur=cur->next[bit];
        }
    }
    return val^cur->val;
}
```

5.10 scc [59 lines] - 0514b290

```
vector<vector<int>>g,rg;
vector<bool>vis;
int timer;
int stime[N];
int etime[N];
void dfs(int node)
{
    if(vis[node])return;
    vis[node]=1;
    stime[node]=++timer;
    for(auto it:g[node])
    {
        dfs(it);
    }
    etime[node]=++timer;
```

```
}
void rdfs(int node,vector<int>&tem)
{
    if(vis[node])return;
    vis[node]=1;
    tem.push_back(node);
    for(auto it:rg[node])
    {
        rdfs(it,tem);
    }
}
//main
int main()
{
    int n,m;
    cin>>n>>m;
    g.assign(n+1,vector<int>(0));
    rg.assign(n+1,vector<int>(0));
    vis.assign(n+1,0);
    timer=0;
    for(int i=0;i<m;i++)
    {
        int x,y;
        cin>>x>>y;
        g[x].push_back(y);
        rg[y].push_back(x);
    }
    vector<pair<int,int>>str;
    for(int i=1;i<=n;i++)
    {
        dfs(i);
        str.push_back({etime[i],i});
    }
    sort(all(str));
    reverse(all(str));
    vis.assign(n+1,0);
    for(auto [ww,i]:str)
    {
        if(vis[i])continue;
        vector<int>tem;
        rdfs(i,tem);
        for(auto it:tem)cout<<it<<" "; cout<<"\n";
    }
}
```

5.11 script [9 lines] - 3ba1f1d4

```
mkdir -p contest
for dir in {a..h}; do
    mkdir -p contest/$dir
    cp solve.cpp contest/$dir/
    cp brute.cpp contest/$dir/
    cp gen.cpp contest/$dir/
    cp stress.sh contest/$dir/
    chmod +x contest/$dir/stress.sh
done
```

5.12 stress [17 lines] - 6927ab2e

```
set -e
g++ code.cpp -o code
g++ gen.cpp -o gen
g++ brute.cpp -o brute
for((i = 1; ; ++i)); do
    ./gen $i > input_file
    ./code < input_file > myAnswer
    ./brute < input_file > correctAnswer
    diff -Z myAnswer correctAnswer > /dev/null || break
    echo "Passed test: "  $i
done
echo "WA on the following test:"
cat input_file
echo "Your answer is:"
cat myAnswer
echo "Correct answer is:"
cat correctAnswer
```

5.13 vectorAddMultiplicaton [40 lines] - c20b8d12

```
vector<int>mul(vector<int>ddd,int k)  //k<=10
{
    int pre=0;
    vector<int>ans;
    for(int i=0; i<ddd.size(); i++)
    {
        int tem=ddd[i]*k+pre;
        ans.push_back(tem%10);
        pre=tem/10;
    }
    if(pre>0)
    {
        ans.push_back(pre);
    }
    return ans;
}
vector<int>plu(vector<int>ddd,vector<int>eee)
{
    vector<int>ans;
    int pre=0;
    for(int i=0;i<max(ddd.size(),eee.size());i++)
    {
        int cur=pre;
        if(i<ddd.size())
        {
            cur+=ddd[i];
        }
        if(i<eee.size())
        {
            cur+=eee[i];
        }
        ans.push_back(cur%10);
        pre=cur/10;
    }
    if(pre>0)
    {
        ans.push_back(pre);
    }
    return ans;
}
```

6 Game Theory

6.1 Points to be noted [14 lines] - 85ddc883

```
>[First Write a Brute Force solution]
>Nim = all xor
>Misere Nim = Nim + corner case: if all piles are 1,
    reverse(nim)
>Bogus Nim = Nim
>Staircase Nim = Odd indexed pile Nim (Even indexed pile
    doesnt matter, as one player can give bogus moves to
    drop all even piles to ground)
>Sprague Grundy: [Every impartial game under the normal
    play convention is equivalent to a one-heap game of
    nim]
Every tree = one nim pile = tree root value; tree leaf
    value = 0; tree node value = mex of all child nodes.
[Careful: one tree node can become multiple new tree
    roots(multiple elements in one node), then the value
    of that node = xor of all those root values]
>Hackenbush(Given a rooted tree; cut an edge in one
    move; subtree under that edge gets removed; last
    player to cut wins):
Colon: //£G(u) = (G(v1)+1) \oplus (G(v2)+1) \oplus
    \cdots [v1, v2,\cdots£ are childs of u]
For multiple trees ans is their xor
>Hackenbush on graph (instead of tree given an rooted
    graph):
fusion: All edges in a cycle can be fused to get a tree
    structure; build a super node, connect some single
    nodes with that super node, number of single nodes
    is the number of edges in the cycle.
Sol: [Bridge component tree] mark all bridges, a group
    of edges that are not bridges, becomes one component
    and contributes number of edges to the hackenbush.
    (even number of edges contributes 0, odd number of
    edges contributes 1)
```

7 dp

7.1 CHT [46 lines] - ae81b51a

```
struct CHT {
vector<ll> m, b;
int ptr = 0;

bool bad(int l1, int l2, int l3) {
    return 1.0 * (b[l3] - b[l1]) * (m[l1] - m[l2])  <=
    1.0 * (b[l2] - b[l1]) * (m[l1] - m[l3]); //(slope
    dec+query min),(slope inc+query max)
    return 1.0 * (b[l3] - b[l1]) * (m[l1] - m[l2])  >
    1.0 * (b[l2] - b[l1]) * (m[l1] - m[l3]); //(slope
    dec+query max), (slope inc+query min)
}

void add(ll _m, ll _b) {
    m.push_back(_m);
    b.push_back(_b);
    int s = m.size();
    while(s >= 3 && bad(s - 3, s - 2, s - 1)) {
    s--;
    m.erase(m.end() - 2);
    b.erase(b.end() - 2);
    }
}

ll f(int i, ll x) {
    return m[i] * x + b[i];
}

//(slope dec+query min), (slope inc+query max) -> x
    increasing
//(slope dec+query max), (slope inc+query min) -> x
    decreasing
ll query(ll x) {
    if(ptr >= m.size()) ptr = m.size() - 1;
    while(ptr < m.size() - 1 && f(ptr + 1, x) < f(ptr,
    x)) ptr++;
    return f(ptr, x);
```

```
}

ll bs(int l, int r, ll x) {
    int mid = (l + r) / 2;
    if(mid + 1 < m.size() && f(mid + 1, x) < f(mid, x))
    return bs(mid + 1, r, x); // > for max
    if(mid - 1 >= 0 && f(mid - 1, x) < f(mid, x)) return
    bs(l, mid - 1, x); // > for max
    return f(mid, x);
}
};

ll a[N], b[N];
CHT cht;
// for(int i = 1; i < n; i++) {
//     ans = cht.query(a[i]);
//     cht.add(b[i], ans);
//   }
```

7.2 divideAndConquer [28 lines] - b9de33ed

```
void yo(int i, int l, int r, int optl, int optr) {
  if(l > r) return;
  int mid = (l + r) / 2;
  dp[i][mid] = inf; // for maximum cost change it to 0
  int opt = -1;
  for(int k = optl; k <= min(mid - 1, optr); k++) {
    int c = dp[i - 1][k] + cost(k + 1, mid);
    if(c < dp[i][mid]) { // for maximum cost just change
    < to > only and rest of the algo should not be
    changed
      dp[i][mid] = c;
      opt = k;
    }
  }
  // for opt[1..j] <= opt[1...j+1]
  if (opt == -1) {
    // if we can't divide into k parts, then go right
    yo(i, mid + 1, r, optl, optr);
    return;
  }
  yo(i, l, mid - 1, optl, opt);
  yo(i, mid + 1, r, opt, optr);

  // for opt[1...j] >= opt[1...j+1]
  // yo(i, l, mid-1, opt, optr);
  // yo(i, mid+1, r, optl, opt);

}
//for(i = 1; i <= n; i++) dp[1][i] = cost(1, i);
//for(i = 2; i <= k; i++) yo(i, 1, n, 1, n);
```

7.3 dynamicCht [30 lines] - 497d2f53

```
struct Line {
  mutable ll k, m, p;
  bool operator<(const Line& o) const { return k < o.k;
    }
  bool operator<(ll x) const { return p < x; }
};

struct LineContainer : multiset<Line, less<>> {
  // (for doubles, use inf = 1/.0, div(a,b) = a/b)
  static const ll inf = LLONG_MAX;
  ll div(ll a, ll b) { // floored division
    return a / b - ((a ^ b) < 0 && a % b); }
  bool isect(iterator x, iterator y) {
    if (y == end()) return x->p = inf, 0;
    if (x->k == y->k) x->p = x->m > y->m ? inf : -inf;
    else x->p = div(y->m - x->m, x->k - y->k);
    return x->p >= y->p;
  }
  void add(ll k, ll m) {
    auto z = insert({k, m, 0}), y = z++, x = y;
    while (isect(y, z)) z = erase(z);
    if (x != begin() && isect(--x, y)) isect(x, y =
    erase(y));
    while ((y = x) != begin() && (--x)->p >= y->p)
      isect(x, erase(y));
  }
  ll query(ll x) {
    assert(!empty());
    auto l = *lower_bound(x);
    return l.k * x + l.m;
  }
};
```

8 Graph

8.1 Bipartite [89 lines] - 073e8191

```
template <bool ToShuffle = false>
struct bipartite_matching {
    int n_left, n_right, flow = 0;
    std::vector<std::vector<int>> g;
    std::vector<int> match_from_left, match_from_right;

    bipartite_matching(int _n_left, int _n_right)
        : n_left(_n_left),
          n_right(_n_right),
          g(_n_left),
          match_from_left(_n_left, -1),
          match_from_right(_n_right, -1),
          dist(_n_left) {}

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

    std::vector<int> dist;

    void bfs() {
        std::queue<int> q;
        for (int u = 0; u < n_left; ++u) {
            if (!~match_from_left[u])
                q.push(u), dist[u] = 0;
            else
                dist[u] = -1;
        }
        while (!q.empty()) {
            int u = q.front();
            q.pop();
            for (auto v : g[u])
                if (~match_from_right[v] &&
    !~dist[match_from_right[v]]) {
                    dist[match_from_right[v]] = dist[u]
    + 1;
                    q.push(match_from_right[v]);
                }
        }
    }

    bool dfs(int u) {
        for (auto v : g[u])
            if (!~match_from_right[v]) {
                match_from_left[u] = v,
    match_from_right[v] = u;
                return true;
            }
        for (auto v : g[u])
            if (dist[match_from_right[v]] == dist[u] + 1
    &&
                dfs(match_from_right[v])) {
                match_from_left[u] = v,
    match_from_right[v] = u;
                return true;
            }
        return false;
    }

    int get_max_matching() {
        if constexpr (ToShuffle) {
            Random rng;
            for (int i = 0; i < n_left; ++i)
                std::random_shuffle(std::begin(g[i]),
    std::end(g[i]), rng);
        }
        while (true) {
            bfs();
            int augment = 0;
            for (int u = 0; u < n_left; ++u)
                if (!~match_from_left[u]) augment +=
    dfs(u);
            if (!augment) break;
            flow += augment;
        }
        return flow;
    }

    std::pair<std::vector<int>, std::vector<int>>
    minimum_vertex_cover() {
        std::vector<int> L, R;
        for (int u = 0; u < n_left; ++u) {
            if (!~dist[u])
                L.push_back(u);
            else if (~match_from_left[u])
                R.push_back(match_from_left[u]);
        }
        return {L, R};
    }

    std::vector<std::pair<int, int>> get_edges() {
        std::vector<std::pair<int, int>> ans;
        for (int u = 0; u < n_left; ++u)
            if (match_from_left[u] != -1)
                ans.emplace_back(u, match_from_left[u]);
        return ans;
    }
};
```

8.2 Euler Path Directed [61 lines] - d5887c16

```
const int N = 4e5 + 9;
/*
all the edges should be in the same connected component
#directed graph: euler path: for all -> indeg = outdeg
    or nodes having indeg > outdeg = outdeg > indeg = 1
    and for others in = out
```

```cpp
#directed graph: euler circuit: for all -> indeg =
    outdeg
*/

//euler path in a directed graph
//it also finds circuit if it exists
vector<int> g[N], ans;
int done[N];
void dfs(int u) {
  while (done[u] < g[u].size()) dfs(g[u][done[u]++]);
  ans.push_back(u);
}
int solve(int n) {
  int edges = 0;
  vector<int> in(n + 1, 0), out(n + 1, 0);
  for (int u = 1; u <= n; u++) {
    for (auto v : g[u]) in[v]++, out[u]++, edges++;
  }
  int ok = 1, cnt1 = 0, cnt2 = 0, root = 0;
  for (int i = 1; i <= n; i++) {
    if (in[i] - out[i] == 1) cnt1++;
    if (out[i] - in[i] == 1) cnt2++, root = i;
    if (abs(in[i] - out[i]) > 1) ok = 0;
  }
  if (cnt1 > 1 || cnt2 > 1) ok = 0;
  if (!ok) return 0;
  if (root == 0) {
    for (int i = 1; i <= n; i++) if (out[i]) root = i;
  }
  if (root == 0) return 1; //empty graph
  dfs(root);
  if (ans.size() != edges + 1) return 0; //connectivity
  reverse(ans.begin(), ans.end());
  return 1;
}
map<string, int> mp;
string id[N];
int T = 0;
int32_t main() {
  int n;
  cin >> n;
  for (int i = 1; i <= n; i++) {
    string s;
    cin >> s;
    string a = s.substr(0, 2);
    string b = s.substr(1);
    if (!mp.count(a)) mp[a] = ++T, id[T] = a;
    if (!mp.count(b)) mp[b] = ++T, id[T] = b;
    g[mp[a]].push_back(mp[b]);
  }
  int ok = solve(T);
  if (!ok) return cout << "NO\n", 0;
  cout << "YES\n";
  string res = id[ans.front()];
  for (int i = 1; i < ans.size(); i++) res +=
    id[ans[i]][1];
  cout << res << '\n';
  return 0;
}
```

8.3 Euler Path Undirected [92 lines] - 3e514bda

```cpp
const int N = 420;

/*
all the edges should be in the same connected component
#undirected graph: euler path: all degrees are even or
    exactly two of them are odd.
#undirected graph: euler circuit: all degrees are even
*/

//euler path in an undirected graph
//it also finds circuit if it exists
vector<pair<int, int>> g[N];
vector<int> ans;
int done[N];
int vis[N * N]; //number of edges
void dfs(int u) {
  while (done[u] < g[u].size()) {
    auto e = g[u][done[u]++];
    if (vis[e.second]) continue;
    vis[e.second] = 1;
    dfs(e.first);
  }
  ans.push_back(u);
}

int solve(int n) {
  int edges = 0;
  ans.clear();
  memset(done, 0, sizeof done);
  memset(vis, 0, sizeof vis);
  vector<int> deg(n + 1, 0);
  for (int u = 1; u <= n; u++) {
    for (auto e : g[u]) {
      deg[e.first]++, deg[u]++, edges++;
    }
  }
  int odd = 0, root = 0;
  for (int i = 1; i <= n; i++) {
    if (deg[i] & 1) odd++, root = i;
  }
  if (odd > 2) return 0;
  if (root == 0) {
    for (int i = 1; i <= n; i++) if (deg[i]) root = i;
  }
  if (root == 0) return 1; //empty graph
  dfs(root);
  if (ans.size() != edges / 2 + 1) return 0;
  reverse(ans.begin(), ans.end());
  return 1;
}

int32_t main() {
  ios_base::sync_with_stdio(0);
  cin.tie(0);

  int t;
  cin >> t;
  while (t--) {
    int n, m;
    cin >> n >> m;
    vector<int> deg(n + 1, 0);
    for (int i = 1; i <= m; i++) {
      int u, v;
      cin >> u >> v;
      g[u].push_back({v, i});
      g[v].push_back({u, i});
      deg[u]++, deg[v]++;
    }
    int sz = m;
    for (int i = 1; i <= n; i++) {
      if (deg[i] & 1) {
        ++sz;
        g[n + 1].push_back({i, sz});
        g[i].push_back({n + 1, sz});
      }
    }
    int ok = solve(n + 1);
    assert(ok);
    vector<int> in(n + 2, 0), out(n + 2, 0);
    for (int i = 0; i + 1 < ans.size(); i++) {
      if (ans[i] != n + 1 && ans[i + 1] != n + 1) {
        in[ans[i + 1]]++;
        out[ans[i]]++;
      }
    }
    int res = 0;
    for (int i = 1; i <= n; i++) res += in[i] == out[i];
    cout << res << '\n';
    for (int i = 0; i + 1 < ans.size(); i++) if (ans[i]
    != n + 1 &&  ans[i + 1] != n + 1) cout << ans[i] <<
    ' ' << ans[i + 1] << '\n';
    for (int i = 0; i <= n + 1; i++) g[i].clear();
  }
  return 0;
}
```

8.4 HopCroft Karp Algorithm [64 lines] - 0eac5635

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

struct HopcroftKarp {
    vector<int> g, l, r; int ans;
    HopcroftKarp(int n, int m, const
    vector<pair<int,int>> &e)
    : g(e.size()), l(n, -1), r(m, -1), ans(0) {
        vector<int> deg(n + 1), a, p, q(n);
        for (auto &[x, y] : e) deg[x]++;
        for (int i = 1; i <= n; i++) deg[i] += deg[i -
    1];
        for (auto &[x, y] : e) g[--deg[x]] = y;
        for (bool match=true; match;) {
            a.assign(n,-1), p.assign(n,-1); int t=0;
    match=false;
            for (int i = 0; i < n; i++)
                if (l[i] == -1) q[t++] = a[i] = p[i] =
    i;
            for (int i = 0; i < t; i++) {
                int x = q[i];
                if (~l[a[x]]) continue;
                for (int j = deg[x]; j < deg[x + 1];
    j++) {
                    int y = g[j];
                    if (r[y] == -1) {
                        while (~y) r[y] = x, swap(l[x],
    y), x = p[x];
                        match = true; ans++; break;
                    }

    if(p[r[y]]==-1)q[t++]=y=r[y],p[y]=x,a[y]=a[x];
                }
            }
```

```
        }
    }
};

void solve(int t){
    int l,r,m;
    cin>>l>>r>>m;

    vector<pair<int,int>> edges;
    for(int i=0;i<m;i++){
        int a,b;
        cin>>a>>b;
        edges.push_back({a,b});
    }
    mt19937 rng(0);
    shuffle(edges.begin(),edges.end(),rng);

    HopcroftKarp ds(l,r,edges);
    cout<<ds.ans<<"\n";
    for(int i=0;i<l;i++){
        if(ds.l[i]!=-1){
            cout<<i<<" "<<ds.l[i]<<"\n";
            assert(ds.r[ds.l[i]]==i);
        }
    }
}

int main(){
    ios::sync_with_stdio(false);
    cin.tie(0);

    int t=1;
    //cin>>t;
    for(int i=1;i<=t;i++)solve(i);

    return 0;
}
```

8.5 MCBM [29 lines] - 7d0529db

```
vector<int>graph[MX];
bool vis[MX];
int match[MX];
bool dfs(int node){
  if(vis[node])return 0;
  vis[node] = 1;
  for(auto nx:graph[node]){
    if(match[nx]==-1 || dfs(match[nx])){
      match[node] = nx;
      match[nx] = node;
      return 1;
    }
  }
  return 0;
}
// inside main()
memset(match, -1, sizeof match);
while(1){
  memset(vis, 0, sizeof vis);
  bool cont = 0;
  for(int i=1;i<=n;i++){
    if(match[i]==-1)cont|=dfs(i);
  }
  if(cont==0)break;
}
int MCBM = 0;
for(int i=1;i<=n;i++){
  if(match[i]!=-1)MCBM++;
}
```

8.6 Maximum Independant Set [105 lines] - 2b659f45

```
// amiD
#include <bits/stdc++.h>
#define ll long long int
#define pb push_back
#define endl '\n'
#define Endl '\n'
#define fi first
#define ii pair<ll,ll>
#define se second
#define ld long double
#define mod 1000000007
#define popcount __builtin_popcountll
#define PI acos(-1.0)
#define MX 555
using namespace std;
vector<int>graph[MX];
int match[MX];
bool vis[MX];
int n, m;

bool dfs(int node){
  if(vis[node])return 0;
  vis[node] = 1;
  for(auto nx:graph[node]){
    if(match[nx]==-1 || dfs(match[nx])){
      match[node] = nx;
      match[nx] = node;
      return 1;
    }
  }
  return 0;
}
void cal(int node){
    if(vis[node])return;
    vis[node] = 1;
    if(node>n){     // node from the right side, can
    only traverse matched edge
        cal(match[node]);
        return;
    }
    for(auto nx:graph[node]){
        if(nx==match[node])continue;
        cal(nx);
    }
}
int main(){
    cin>>n>>m;
    for(int i=1;i<=n;i++){
        int k;
        scanf("%d", &k);
        for(int j=0;j<k;j++){
            char c;
            scanf(" %c", &c);
            int idx = (c-'A') + n + 1;
            graph[i].pb(n+c-'A' + 1);
            graph[n+c-'A' + 1].pb(i);
        }
    }
    memset(match, -1, sizeof match);
    while(1){
        memset(vis, 0, sizeof vis);
        bool cont = 0;
        for(int i=1;i<=n;i++){
            if(match[i]==-1)cont|=dfs(i);
        }
        if(cont==0)break;
    }
    memset(vis,0,sizeof vis);
    for(int i=1;i<=n;i++){
        if(match[i]!=-1)continue;       // matched node
    from the left side
        cal(i);
    }
    vector<int>mvc, MaxIS;
    for(int i=1;i<=n;i++){
        // Left side nodes
        // Visited nodes are part of the mvc
        // Unvisited nodes are part of the MaxIS
        if(vis[i])MaxIS.pb(i);
        else mvc.pb(i);
    }
    for(int i=n+1;i<=n+m;i++){
        // Right side nodes
        // Visited nodes are part of the MaxIS
        // Unvisited nodes are part of the mvc
        if(!vis[i])MaxIS.pb(i);
        else mvc.pb(i);
    }
    cout<<"MVC nodes:\n";
    for(auto x:mvc){
        if(x<=n)cout<<x<<" ";
        else cout<<char(x-n+'A'-1)<<" ";
    }
    cout<<endl;
    cout<<"MaxIS nodes:\n";
    for(auto x:MaxIS){
        if(x<=n)cout<<x<<" ";
        else cout<<char(x-n+'A'-1)<<" ";
    }
}
/*
4 5
2 A B
3 A C E
1 D
1 D
*/
```

8.7 articulationBridge [45 lines] - 4af66da3

```
l n,m;
ll ans;
vector<int>g[N];
int co[N];
vector<bool> visited;
vector<int> tin, low;
int timer;

void dfs(int v, int p = -1) {
    co[v]=1;
    visited[v] = true;
    tin[v] = low[v] = timer++;
    bool parent_skipped = false;
    for (int to : g[v]) {
```

```
        if (to == p && !parent_skipped) {
            parent_skipped = true;
            continue;
        }
        if (visited[to]) {
            low[v] = min(low[v], tin[to]);
        } else {
            dfs(to, v);
            co[v]+=co[to];
            low[v] = min(low[v], low[to]);

            if (low[to] > tin[v])
            {
                ll rem=co[to];
                ll cur=n-co[to];

    ans=min(ans,(rem*(rem-1))/2+(cur*(cur-1))/2);
            }
        }
    }
}

void find_bridges() {
    timer = 0;
    visited.assign(n+1, false);
    tin.assign(n+1, -1);
    low.assign(n+1, -1);
    for (int i = 1; i <=n; ++i) {
        if (!visited[i])
            dfs(i);
    }
}
```

8.8 articulationPoint [33 lines] - 0a056b91

```
vector<int>g[N];
bool vis[N];
set<int>str;
int tin[N],low[N],T;
void dfs(int v,int pa)
{
    vis[v]=1;
    tin[v]=low[v]=++T;
    int child=0;
    for(auto it:g[v])
    {
        if(it==pa)continue;
        if(vis[it]==1)
        {
            low[v]=min(low[v],tin[it]);
        }
        else
        {
            dfs(it,v);
            low[v]=min(low[v],low[it]);
            child++;
            if(tin[v]<=low[it]&&pa!=0)
            {
                str.insert(v);
            }
        }
    }
    if(pa==0&&child>1)
    {
        str.insert(v);
    }
}
```

8.9 dsuWithRollback [57 lines] - 19da151f

```
struct DSU
{
    vector<int>tra;
    vector<int>tot;
    vector<pair<int,int>>str;
    bool rb;
    int cc;
    DSU(int n,bool rb=0)
    {
        cc=n;
        this->rb=rb;
        tra.resize(n+1);
        tot.resize(n+1,1);
        for(int i=0; i<=n; i++)tra[i]=i;
    }
    int par(int chi)
    {
        if(tra[chi]==chi)return chi;
        else if(rb)par(tra[chi]);
        else return tra[chi]=par(tra[chi]);
    }
    bool isjoint(int x,int y)
    {
        return par(x)==par(y);
    }
    int len()
    {
        set<int>tem;
        for(int i=1;i<tra.size();i++)tem.insert(par(i));
        return tem.size();
    }
    void join(int x,int y)
    {
        x=par(x);
        y=par(y);
        if(x==y)return;
        cc--;
        if(tot[x]>=tot[y])
        {
            tra[y]=x,tot[x]+=tot[y];
            if(rb)str.push_back({x,y});
        }
        else
        {
            tra[x]=y,tot[y]+=tot[x];
            if(rb)str.push_back({y,x});
        }
    }
    void pop()
    {
        auto [x,y]=str.back();
        str.pop_back();
        tot[x]-=tot[y];
        tra[y]=y;
        cc++;
    }
};
```

8.10 eulerianPathInAllConnected [28 lines] - b0312712

```
for(int i=0; i<ans; i++)
        {
            cal[i]=i;
            for(int j=0; j<ans; j++)tra[i][j]=0;
        }
inds.clear();
inds.push_back(0);
fun(0,ans);
//fun
vector<int>inds;
int cal[P];
void fun(int cur,int ans)
{
    while(1)
    {
        if(tra[cur][cal[cur]]==0)
        {
            int i=cal[cur];
            inds.push_back(i);
            tra[cur][i]=1;
            tra[i][cur]=1;
            fun(i,ans);
            return ;
        }
        cal[cur]=(cal[cur]+1)%ans;
        if(cal[cur]==cur)break;
    }
}
```

8.11 heavyLightDecomposition [95 lines] - 7d512e94

```
vector<int>g[N];
int b[N],a[N];
int parent[N], depth[N], heavy[N], head[N], pos[N];
int timer=0;
int dfs(int node)
{
    int msi=0,si=1;
    heavy[node]=0;
    for(auto it:g[node])
    {
        if(it==parent[node])continue;
        parent[it]=node;
        depth[it]=depth[node]+1;
        int csi=dfs(it);
        si+=csi;
        if(csi>msi)
        {
            msi=csi;
            heavy[node]=it;
        }
    }
    return si;
}
void decompose(int node,int h)
{
    head[node]=h;
    timer++;
    pos[node]=timer;
    b[timer]=a[node];
    if(heavy[node]!=0)
    {
        decompose(heavy[node],h);
    }
    for(auto it:g[node])
    {
```

```
        if(it==parent[node]||it==heavy[node])continue;
        decompose(it,it);
    }
}
//segemtn tree start
int tree[4*N];
int  make(int node,int i,int n)
{
    if(i==n)
    {
        tree[node]=b[i];
        //cout<<tree[node]<<" ";
        return tree[node];
    }
    int mid=(i+n)/2;
    int p=make(node*2,i,mid);
    int q=make(node*2+1,mid+1,n);
    tree[node]=(p+q);
    return tree[node];
}
int upd(int node,int i,int n,int ind)
{
    if(i==n&&ind==i)
    {
        return tree[node]=b[ind];
    }
    if(n<ind||i>ind)return tree[node];
    int mid=(i+n)/2;
    return tree[node]=(upd(node*2,i,mid,ind)+upd(node*
    2+1,mid+1,n,ind));
}
int ans(int node,int i,int n,int p,int q)
{
    if(n<p||i>q)return 0;
    if(i>=p&&n<=q)
    {
        return tree[node];
    }
    int mid=(i+n)/2;
    int pp=ans(node*2,i,mid,p,q);
    int qq=ans(node*2+1,mid+1,n,p,q);
    //cout<<min(pp,qq)<<" ";
    return (pp+qq);
}
//segment tree end
int query(int x,int y,int n)
{
    int res = 0;
    for (; head[x] != head[y]; y = parent[head[y]])
    {
        if (depth[head[x]] > depth[head[y]])
            swap(x, y);
        int cur_heavy_path_max =
    ans(1,1,n,pos[head[y]], pos[y]);
        res = (res+ cur_heavy_path_max);
    }
    if (depth[x] > depth[y])
        swap(x, y);
    int last_heavy_path_max = ans(1,1,n,pos[x], pos[y]);
    res = (res+ last_heavy_path_max);
    return res;
}
```

8.12 maxDistanceFromAllNode [25 lines] - 8bfd24d7

```
vector<int>g[N];
int ma2[N][2];
int madis[N];
int dfs1(int node,int par)
{
    ma2[node][0]=ma2[node][1]=0;
    for(auto it:g[node])
    {
        if(it==par)continue;
        ma2[node][1]=max(ma2[node][1],1+dfs1(it,node));
        if(ma2[node][1]>ma2[node][0])swap(ma2[node][1],
    ma2[node][0]);
    }
    return ma2[node][0];
}
void dfs2(int node,int par,int pre)
{
    madis[node]=pre;
    for(auto it:g[node])
    {
        if(it==par)continue;
        madis[node]=max(madis[node],ma2[it][0]+1);
        if(ma2[node][0]==ma2[it][0]+1)dfs2(it,node,
    max(pre+1,ma2[node][1]+1));
        else dfs2(it,node,max(pre+1,ma2[node][0]+1));
    }
}
```

8.13 mst [44 lines] - c3578a9f

```
struct edge
{
    int x,y;
    ll w;
    edge() {}
    edge(int x,int y,ll w)
    {
        this->x=x;
        this->y=y;
        this->w=w;
    }
    bool operator <(edge b)
    {
        return w<b.w;
    }
};
ll MST(vector<edge>&str,int n)
{
    sort(str.begin(),str.end());
    ll sum=0;
    DSU dsu(n);
    for(auto it:str)
    {
        if(dsu.isjoint(it.x,it.y)==0)
        {
            sum+=it.w;
            dsu.join(it.x,it.y);
        }
    }
    if(dsu.cc>1)sum=-1;
    return  sum;
}
//main
int main()
{
    int n,m;
    cin>>n>>m;
    vector<edge>str(m);
    for(int i=0; i<m; i++)
    {
        cin>>str[i].x>>str[i].y>>str[i].w;
    }
    cout<<MST(str,n);
}
```

9 Number Theory

9.1 Diophantine [77 lines] - 40c31ff8

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

int gcd(int a, int b, int& x, int& y) {
    if (b == 0) {
        x = 1;
        y = 0;
        return a;
    }
    int x1, y1;
    int d = gcd(b, a % b, x1, y1);
    x = y1;
    y = x1 - y1 * (a / b);
    return d;
}

bool find_any_solution(int a, int b, int c, int &x0, int
    &y0, int &g) {
    g = gcd(abs(a), abs(b), x0, y0);
    if (c % g) {
        return false;
    }

    x0 *= c / g;
    y0 *= c / g;
    if (a < 0) x0 = -x0;
    if (b < 0) y0 = -y0;
    return true;
}

void shift_solution(int & x, int & y, int a, int b, int
    cnt) {
    x += cnt * b;
    y -= cnt * a;
}

int find_all_solutions(int a, int b, int c, int minx,
    int maxx, int miny, int maxy) {
    int x, y, g;
    if (!find_any_solution(a, b, c, x, y, g))
        return 0;
    a /= g;
    b /= g;

    int sign_a = a > 0 ? +1 : -1;
    int sign_b = b > 0 ? +1 : -1;

    shift_solution(x, y, a, b, (minx - x) / b);
    if (x < minx)
        shift_solution(x, y, a, b, sign_b);
    if (x > maxx)
```

```
        return 0;
    int lx1 = x;

    shift_solution(x, y, a, b, (maxx - x) / b);
    if (x > maxx)
        shift_solution(x, y, a, b, -sign_b);
    int rx1 = x;

    shift_solution(x, y, a, b, -(miny - y) / a);
    if (y < miny)
        shift_solution(x, y, a, b, -sign_a);
    if (y > maxy)
        return 0;
    int lx2 = x;

    shift_solution(x, y, a, b, -(maxy - y) / a);
    if (y > maxy)
        shift_solution(x, y, a, b, sign_a);
    int rx2 = x;

    if (lx2 > rx2)
        swap(lx2, rx2);
    int lx = max(lx1, lx2);
    int rx = min(rx1, rx2);

    if (lx > rx)
        return 0;
    return (rx - lx) / abs(b) + 1;
}
```

9.2 convolution [201 lines] - f44c0e25

```
/*
  OR convolution (indices are bitmasks 0..(1<<nbits)-1)
*/
struct ORConv {
    static void zeta(vector<ll>& a, int nbits){
        int N = 1<<nbits;
        for(int b=0;b<nbits;b++){
            for(int mask=0;mask<N;mask++){
                if(mask & (1<<b)) a[mask] += a[mask ^
    (1<<b)];
            }
        }
    }
    static void mobius(vector<ll>& a, int nbits){
        int N = 1<<nbits;
        for(int b=0;b<nbits;b++){
            for(int mask=0;mask<N;mask++){
                if(mask & (1<<b)) a[mask] -= a[mask ^
    (1<<b)];
            }
        }
    }
    static vector<ll> convolve(vector<ll> A, vector<ll>
    B, int nbits){
        int N = 1<<nbits;
        if((int)A.size() != N || (int)B.size() != N)
    throw runtime_error("ORConv: size mismatch");
        zeta(A, nbits); zeta(B, nbits);
        vector<ll> C(N);
        for(int i=0;i<N;i++) C[i] = A[i] * B[i];
        mobius(C, nbits);
        return C;
    }
};

/*
  AND convolution
*/
struct ANDConv {
    static void zeta_superset(vector<ll>& a, int nbits){
        int N = 1<<nbits;
        for(int b=0;b<nbits;b++){
            for(int mask=0;mask<N;mask++){
                if(!(mask & (1<<b))) a[mask] += a[mask |
    (1<<b)];
            }
        }
    }
    static void mobius_superset(vector<ll>& a, int
    nbits){
        int N = 1<<nbits;
        for(int b=0;b<nbits;b++){
            for(int mask=0;mask<N;mask++){
                if(!(mask & (1<<b))) a[mask] -= a[mask |
    (1<<b)];
            }
        }
    }
    static vector<ll> convolve(vector<ll> A, vector<ll>
    B, int nbits){
        int N = 1<<nbits;
        if((int)A.size() != N || (int)B.size() != N)
    throw runtime_error("ANDConv: size mismatch");
        zeta_superset(A, nbits); zeta_superset(B,
    nbits);
        vector<ll> C(N);
        for(int i=0;i<N;i++) C[i] = A[i] * B[i];
        mobius_superset(C, nbits);
        return C;
    }
};

/*
  XOR convolution (FWHT)
*/
struct XORConv {
    static void fwht(vector<ll>& a, bool inverse=false){
        int N = (int)a.size();
        for(int len=1; len<N; len<<=1){
            for(int i=0;i<N;i += (len<<1)){
                for(int j=0;j<len;j++){
                    ll u = a[i+j];
                    ll v = a[i+j+len];
                    a[i+j] = u + v;
                    a[i+j+len] = u - v;
                }
            }
        }
        if(inverse){
            // divide by N
            for(int i=0;i<N;i++) {
                // for integer arrays, result should be
    divisible by N if original convolution is integral.
                a[i] /= N;
            }
        }
    }
    static vector<ll> convolve(vector<ll> A, vector<ll>
    B){
        int N = (int)A.size();
        if(N != (int)B.size()) throw
    runtime_error("XORConv: size mismatch");
        vector<ll> fa=A, fb=B;
        fwht(fa, false);
        fwht(fb, false);
        for(int i=0;i<N;i++) fa[i] = fa[i] * fb[i];
        fwht(fa, true);
        return fa;
    }
};


/*
  GCD Convolution
*/
struct GCDConv {
    static vector<int> mobius_sieve(int M){
        vector<int> mu(M+1, 1), primes; vector<int>
    mind(M+1,0);
        mu[0]=0; if(M>=1) mu[1]=1;
        for(int i=2;i<=M;i++){
            if(!mind[i]) { mind[i]=i;
    primes.push_back(i); mu[i] = -1; }
            for(int p: primes){
                long long v = 1LL * p * i;
                if(v> M) break;
                mind[v] = p;
                if(i % p == 0){ mu[v]=0; break;}
                else mu[v] = -mu[i];
            }
        }
        return mu;
    }
    static vector<ll> convolve(const vector<ll>& A,
    const vector<ll>& B){
        int M = (int)A.size()-1;
        if((int)B.size()-1 != M) throw
    runtime_error("GCDConv: size mismatch");
        vector<ll> tA(M+1,0), tB(M+1,0);
        for(int d=1; d<=M; d++){
            for(int k=d; k<=M; k+=d){
                tA[d] += A[k];
                tB[d] += B[k];
            }
        }
        vector<ll> tC(M+1);
        for(int d=1; d<=M; d++) tC[d] = tA[d] * tB[d];
        // Möbius for multiples inversion: C[i] =
    sum_{t: i*t <= M} mu[t] * tC[i*t]
        vector<int> mu = mobius_sieve(M);
        vector<ll> C(M+1,0);
        for(int i=1;i<=M;i++){
            ll val = 0;
            for(int t=1; i*t<=M; t++){
                val += (ll)mu[t] * tC[i*t];
            }
            C[i] = val;
        }
        return C;
    }
```

```
};

/*
  LCM Convolution
*/
struct LCMConv {
    static vector<int> mobius_sieve(int M){
        vector<int> mu(M+1, 1), primes; vector<int>
    mind(M+1,0);
        mu[0]=0; if(M>=1) mu[1]=1;
        for(int i=2;i<=M;i++){
            if(!mind[i]) { mind[i]=i;
    primes.push_back(i); mu[i] = -1; }
            for(int p: primes){
                long long v = 1LL * p * i;
                if(v> M) break;
                mind[v] = p;
                if(i % p == 0){ mu[v]=0; break;}
                else mu[v] = -mu[i];
            }
        }
        return mu;
    }
    static vector<ll> convolve(const vector<ll>& A,
    const vector<ll>& B){
        int M = (int)A.size()-1;
        if((int)B.size()-1 != M) throw
    runtime_error("LCMConv: size mismatch");
        vector<ll> tA(M+1,0), tB(M+1,0);
        for(int m=1;m<=M;m++){
            for(int d=1; d*d<=m; d++){
                if(m % d == 0){
                    tA[m] += A[d];
                    if(d*d != m) tA[m] += A[m/d];
                }
            }
            for(int d=1; d*d<=m; d++){
                if(m % d == 0){
                    tB[m] += B[d];
                    if(d*d != m) tB[m] += B[m/d];
                }
            }
        }
        vector<ll> tC(M+1);
        for(int m=1;m<=M;m++) tC[m] = tA[m] * tB[m];
        vector<int> mu = mobius_sieve(M);
        vector<ll> C(M+1,0);
        for(int m=1;m<=M;m++){
            ll val = 0;
            // sum_{d|m} mu[d] * tC[m/d]
            for(int d=1; d*d<=m; d++){
                if(m % d == 0){
                    val += (ll)mu[d] * tC[m/d];
                    int d2 = m/d;
                    if(d2 != d) val += (ll)mu[d2] *
    tC[m/d2];
                }
            }
            C[m] = val;
        }
        return C;
    }
};
```

9.3 crt [77 lines] - fcdd52c1

```
ll pow(ll base,ll po,int M)
{
    ll ans=1;
    while(po)
    {
        if(po%2)
        {
            ans=(ans*base)%M;
        }
        base=(base*base)%M;
        po/=2;
    }
    return ans;
}
int phi(int n)
{
    int ans=1;
    int q=sqrt(n);
    for(int i=2;i<=q;i++)
    {
        if(n%i==0)
        {
            int tem=1;
            while(n%i==0)
            {
                tem*=i;
                n/=i;
            }
            ans=ans*tem/i*(i-1);
            q=sqrt(n);
        }
    }
    if(n>1)ans=ans*(n-1);
    return ans;
}
bool issolvable(vector<pair<int,int>>&str)
{
    int n=str.size();
    ll mm=1;
    for(int i=0;i<n;i++)
    {
        if(__gcd(mm,(ll)str[i].second)!=1)return 0;
        mm*=str[i].second;
    }
    return 1;
}
ll CRT(vector<pair<int,int>>&str)
{
    ll mm=1;
    int n=str.size();
    for(int i=0;i<n;i++)mm=mm*str[i].second;
    ll x=0;
    for(int i=0;i<n;i++)
    {
        x=x+str[i].first*mm/str[i].second*pow(mm
    /str[i].second,phi(str[i].second)-1,str[i].second);
    }
    return x=((x%mm)+mm)%mm;
}
//main
int main()
{
    fastio
    int n;
    cin>>n;
    vector<pair<int,int>>str;
    for(int i=0;i<n;i++)
    {
        int a,m;
        cin>>a>>m;
        str.push_back({a,m});
    }
    if(issolvable(str))
    {
        yes(1);
        cout<<CRT(str);
    }else yes(0);
}
```

9.4 determinantOfMatrix [26 lines] - fa7731ea

```
//const int N=24;
ll mat[N][N];
ll dp[1<<N];
ll det(int mask,int n)
{
    if(mask==0)return 1;
    if(dp[mask]!=-1)return dp[mask];
    int used=n-__builtin_popcount(mask);
    dp[mask]=0;
    int co=0;
    for(int i=0;i<n;i++)
    {
        if(mask&(1<<i))
        {
            co++;
            if(co%2)
            {

    dp[mask]+=mat[used][i]*det(mask^(1<<i),n);
            }else
            {

    dp[mask]-=mat[used][i]*det(mask^(1<<i),n);
            }
        }
    }
    return dp[mask];
}
```

9.5 egcd [14 lines] - 0e34856e

```
int ext_gcd(int a, int b, int& x, int& y)
{
    if (b == 0)
    {
        x = 1;
        y = 0;
        return a;
    }
    int x1, y1;
    int d = ext_gcd(b, a % b, x1, y1);
    x = y1;
    y = x1 - y1 * (a / b);
    return d;
}
```

9.6 gauss [70 lines] - 65f7a22c

```
const ld M=1e-5;
const int N=102;
//tem
long double ans[N];
long double mat[N][N];
int n;
void rswap(int i,int j)
{
    if(i==j)return ;
    for(int k=1; i<=n+1; k++)
    {
        long double tem=mat[i][k];
        mat[i][k]=mat[j][k];
        mat[j][k]=tem;
        //swap(mat[i][k],mat[j][k]);
    }
}
int fun()
{
    int tot=1;
    for(int j=1; j<=n; j++)
    {
        int p=0;
        for(int i=tot; i<=n; i++)
        {
            if(abs(mat[i][j])>M)
            {
                p=i;
                break;
            }
        }
        if(p)
        {
            rswap(tot,p);
            for(int i=tot+1; i<=n; i++)
            {
                if(abs(mat[i][j])>M)
                {
                    long double
mag=mat[i][j]/mat[tot][j];
                    for(int k=j; k<=n+1; k++)
                    {

mat[i][k]=mat[i][k]-mag*mat[tot][k];
                        if(abs(mat[i][k])<M)mat[i][k]=0;
                    }
                }
            }
            tot++;
        }
    }
    ans[n+1]=-1;
    for(int i=n; i>=1; i--)
    {
        for(int j=i; j<=n; j++)
        {
            if(abs(mat[i][j])>M)
            {
                long double sum=0;
                for(int k=j+1; k<=n+1; k++)
                {
                    sum=sum+mat[i][k]*ans[k];
                }
                sum=-sum;
                ans[j]=sum/mat[i][j];
                break;
            }
        }
    }
    return tot;
}
```

9.7 hi [0 lines] - d41d8cd9

9.8 inverse [3 lines] - ef9c1acd

```
for (int i = 2; i <= maxV; ++i)
  invVal[i] = (MOD - (MOD / i) * invVal[MOD % i] % MOD)
    % MOD;
```

9.9 linear$_{s}$ystem[83lines] − 062d3731

```
#include <iostream>
#include <vector>
#include <cmath>
#include <stdexcept>
using namespace std;

const double EPS = 1e-9; // Small value to handle
    floating-point precision issues

class LinearSystem {
private:
    vector<vector<double>> matrix; // Augmented matrix
    int n; // Number of equations

public:
    // Constructor to initialize the system
    LinearSystem(const vector<vector<double>>
    &augmentedMatrix) {
        matrix = augmentedMatrix;
        n = matrix.size();
    }

    // Function to perform Gaussian elimination
    vector<double> solve() {
        // Forward elimination
        for (int i = 0; i < n; i++) {
            // Find the row with the maximum element in
    the current column
            int maxRow = i;
            for (int k = i + 1; k < n; k++) {
                if (abs(matrix[k][i]) >
    abs(matrix[maxRow][i])) {
                    maxRow = k;
                }
            }

            // Swap the current row with the max row
            swap(matrix[i], matrix[maxRow]);

            // If the pivot element is zero, the system
    may have no unique solution
            if (abs(matrix[i][i]) < EPS) {
                throw runtime_error("No unique solution
    exists.");
            }

            // Make all rows below this one 0 in the
    current column
            for (int k = i + 1; k < n; k++) {
                double factor = matrix[k][i] /
    matrix[i][i];
                for (int j = i; j <= n; j++) {
                    matrix[k][j] -= factor *
    matrix[i][j];
                }
            }
        }

        // Back substitution
        vector<double> solution(n);
        for (int i = n - 1; i >= 0; i--) {
            solution[i] = matrix[i][n] / matrix[i][i];
            for (int k = i - 1; k >= 0; k--) {
                matrix[k][n] -= matrix[k][i] *
    solution[i];
            }
        }

        return solution;
    }

    // Function to display the augmented matrix
    void displayMatrix() const {
        for (const auto &row : matrix) {
            for (double val : row) {
                cout << val << "\t";
            }
            cout << endl;
        }
    }
};

int main() {
    // Example system of equations:
    // 3x + 2y - 4z = 3
    // 2x + 3y + 3z = 15
    // 5x - 3y + z = 14
    vector<vector<double>> augmentedMatrix = {
        {3, 2, -4, 3},
        {2, 3, 3, 15},
        {5, -3, 1, 14}
    };
}
```

9.10 movius [10 lines] - 5191fda7

```
int movious[N];
void make()
{
    for(int i=0;i<N;i++)movious[i]=0;
    movious[1]=1;
    for(int i=1;i<N;i++)
    {
        for(int j=i+i;j<N;j+=i)movious[j]-=movious[i];
    }
}
```

9.11 phi [34 lines] - dda0b159

```
*****************phi-1-**********(10^6)********
int phi[N];
void make()
{
    for(int i=0;i<N;i++)phi[i]=i;
    for(int i=1;i<N;i++)
    {
        for(int j=i+i;j<N;j+=i)phi[j]-=phi[i];
    }
}

*****************phi-2-*********10^16***********
int phi(int n)
{
    int ans=1;
    int q=sqrt(n);
    for(int i=2;i<=q;i++)
    {
        if(n%i==0)
        {
            int tem=1;
            while(n%i==0)
            {
                tem*=i;
                n/=i;
            }
            ans=ans*tem/i*(i-1);
            q=sqrt(n);
        }
    }
    if(n>1)ans=ans*(n-1);
    return ans;
}
```

9.12 pollardRho [195 lines] - 7b5dd641

```
uint64_t gcd_stein_impl( uint64_t x, uint64_t y ) {
    if( x == y ) { return x; }
    const uint64_t a = y - x;
    const uint64_t b = x - y;
    const int n = __builtin_ctzll( b );
    const uint64_t s = x < y ? a : b;
    const uint64_t t = x < y ? x : y;
    return gcd_stein_impl( s >> n, t );
}

uint64_t gcd_stein( uint64_t x, uint64_t y ) {
    if( x == 0 ) { return y; }
    if( y == 0 ) { return x; }
    const int n = __builtin_ctzll( x );
    const int m = __builtin_ctzll( y );
    return gcd_stein_impl( x >> n, y >> m ) << ( n < m ?
    n : m );
}

// ---- is_prime ----

uint64_t mod_pow( uint64_t x, uint64_t y, uint64_t mod )
    {
    uint64_t ret = 1;
    uint64_t acc = x;
    for( ; y; y >>= 1 ) {
        if( y & 1 ) {
            ret = __uint128_t(ret) * acc % mod;
        }
        acc = __uint128_t(acc) * acc % mod;
    }
    return ret;
}

bool miller_rabin( uint64_t n, const
    std::initializer_list<uint64_t>& as ) {
    return std::all_of( as.begin(), as.end(), [n](
    uint64_t a ) {
        if( n <= a ) { return true; }

        int e = __builtin_ctzll( n - 1 );
        uint64_t z = mod_pow( a, ( n - 1 ) >> e, n );
        if( z == 1 || z == n - 1 ) { return true; }

        while( --e ) {
            z = __uint128_t(z) * z % n;
            if( z == 1 ) { return false; }
            if( z == n - 1 ) { return true; }
        }

        return false;
    });
}

bool is_prime( uint64_t n ) {
    if( n == 2 ) { return true; }
    if( n % 2 == 0 ) { return false; }
    if( n < 4759123141 ) { return miller_rabin( n, { 2,
    7, 61 } ); }
    return miller_rabin( n, { 2, 325, 9375, 28178,
    450775, 9780504, 1795265022 } );
}

// ---- Montgomery ----

class Montgomery {
    uint64_t mod;
    uint64_t R;
public:
    Montgomery( uint64_t n ) : mod(n), R(n) {
       for( size_t i = 0; i < 5; ++i ) {
          R *= 2 - mod * R;
       }
    }

    uint64_t fma( uint64_t a, uint64_t b, uint64_t c )
    const {
        const __uint128_t d = __uint128_t(a) * b;
        const uint64_t    e = c + mod + ( d >> 64 );
        const uint64_t    f = uint64_t(d) * R;
        const uint64_t    g = ( __uint128_t(f) * mod )
    >> 64;
        return e - g;
    }

    uint64_t mul( uint64_t a, uint64_t b ) const {
        return fma( a, b, 0 );
    }
};

// ---- Pollard's rho algorithm ----

uint64_t pollard_rho( uint64_t n ) {
    if( n % 2 == 0 ) { return 2; }
    const Montgomery m( n );

    constexpr uint64_t C1 = 1;
    constexpr uint64_t C2 = 2;
    constexpr uint64_t M = 512;

    uint64_t Z1 = 1;
    uint64_t Z2 = 2;
retry:
    uint64_t z1 = Z1;
    uint64_t z2 = Z2;
    for( size_t k = M; ; k *= 2 ) {
        const uint64_t x1 = z1 + n;
        const uint64_t x2 = z2 + n;
        for( size_t j = 0; j < k; j += M ) {
            const uint64_t y1 = z1;
            const uint64_t y2 = z2;

            uint64_t q1 = 1;
            uint64_t q2 = 2;
            z1 = m.fma( z1, z1, C1 );
            z2 = m.fma( z2, z2, C2 );
            for( size_t i = 0; i < M; ++i ) {
                const uint64_t t1 = x1 - z1;
                const uint64_t t2 = x2 - z2;
                z1 = m.fma( z1, z1, C1 );
                z2 = m.fma( z2, z2, C2 );
                q1 = m.mul( q1, t1 );
                q2 = m.mul( q2, t2 );
            }
            q1 = m.mul( q1, x1 - z1 );
            q2 = m.mul( q2, x2 - z2 );

            const uint64_t q3 = m.mul( q1, q2 );
            const uint64_t g3 = gcd_stein( n, q3 );
            if( g3 == 1 ) { continue; }
            if( g3 != n ) { return g3; }

            const uint64_t g1 = gcd_stein( n, q1 );
            const uint64_t g2 = gcd_stein( n, q2 );

            const uint64_t C = g1 != 1 ? C1 : C2;
            const uint64_t x = g1 != 1 ? x1 : x2;
            uint64_t       z = g1 != 1 ? y1 : y2;
            uint64_t       g = g1 != 1 ? g1 : g2;

            if( g == n ) {
                do {
                    z = m.fma( z, z, C );
                    g = gcd_stein( n, x - z );
                } while( g == 1 );
            }
            if( g != n ) {
                return g;
            }

            Z1 += 2;
            Z2 += 2;
            goto retry;
        }
    }
```

```
}

void factorize_impl( uint64_t n, std::vector<uint64_t>&
    ret ) {
    if( n <= 1 ) { return; }
    if( is_prime( n ) ) { ret.push_back( n ); return; }

    const uint64_t p = pollard_rho( n );

    factorize_impl( p, ret );
    factorize_impl( n / p, ret );
}

std::vector<uint64_t> factorize( uint64_t n ) {
    std::vector<uint64_t> ret;
    factorize_impl( n, ret );
    std::sort( ret.begin(), ret.end() );
    return ret;
}
vector<pii> generate_divisor(vector<uint64_t> primes)
{
    vector<pii> divisors;
    sort(primes.begin(), primes.end());
    // debug(primes);
    vector<pair<long long, long long>> pfreq;
    for (int i = 0, j; i < primes.size(); i = j) {
        j = i;
        while (j < primes.size() && primes[i] ==
    primes[j]) j++;
        pfreq.push_back({primes[i], j - i});
    }
    divisors.push_back({1, 1});

    for (auto [prime, count] : pfreq) {
        long long result = 1;
        vector<pii> temp;
        for (int i = 1; i <= count; i++) {
            result *= prime;
            int x = result / prime * (prime - 1);
            for (auto [d, phi] : divisors) {
                temp.push_back({d * result, phi * x});
            }
        }
        for (auto d : temp) {
            divisors.push_back(d);
        }
    }
    // sort(divisors.begin(), divisors.end());
    return divisors;
}
```

9.13 prime [48 lines] - cd32f7ee

```
**********Linear**********
std::vector <int> prime;
bool is_composite[MAXN];

void sieve (int n) {
  std::fill (is_composite, is_composite + n, false);
  for (int i = 2; i < n; ++i) {
    if (!is_composite[i]) prime.push_back (i);
    for (int j = 0; j < prime.size () && i * prime[j] <
    n; ++j) {
      is_composite[i * prime[j]] = true;
      if (i % prime[j] == 0) break;
    }
  }
}



void largepri(int N)
{
    int n=sqrt(N);
    normalsieve(n);
    int st=n+1;
    int ed=st+n;
    bool vis[n+1];
    while(st<N)
    {
        if(ed>N)ed=N;
        for(int i=0;i<=n;i++)vis[i]=1;
        for(auto it:pri)
        {
            for(int i=((st-1)/it+1)*it-st;i<=n;i+=it)
            {
                //cout<<i+st<<" ";
                vis[i]=0;
            }
        }
        for(int i=0;i<=ed-st;i++)
        {
            if(vis[i])
            {
                //cout<<i+st<<" ";
                allpri.push_back(i+st);
            }
        }
        st=ed+1;
        ed=st+n;
    }
}
```

9.14 sieve [23 lines] - 1a683527

```
std::vector <int> prime;
bool is_composite[MAXN];
int phi[MAXN];

void sieve (int n) {
  std::fill (is_composite, is_composite + n, false);
  phi[1] = 1;
  for (int i = 2; i < n; ++i) {
    if (!is_composite[i]) {
      prime.push_back (i);
      phi[i] = i - 1;           //i is prime
    }
    for (int j = 0; j < prime.size () && i * prime[j] <
    n; ++j) {
      is_composite[i * prime[j]] = true;
      if (i % prime[j] == 0) {
        phi[i * prime[j]] = phi[i] *
    prime[j];  //prime[j] divides i
        break;
      } else {
        phi[i * prime[j]] = phi[i] *
    phi[prime[j]];  //prime[j] does not divide i
      }
    }
  }
}
```