University of Southern California

USC Trojans

ICPC Reference Document - 2020

Team Note of USC

Some Members

Compiled on February 11, 2020

Contents

1 Flow

1.1 Bipartite Matching

Usage: Use `add_edge` to add edges, `bipartite_matching` to get maximal matching.
Time Complexity: $\mathcal{O}(VE)$

```
int V;
vector<int> G[N];
int match[N];
bool used[N];
```

```
void add_edge(int u, int v) {
    G[u].push_back(v);
    G[v].push_back(u);
}

bool dfs(int v) {
    used[v] = true;
    for (int u : G[v]) {
        int w = match[u];
        if (w < 0 || !used[w] && dfs(w)) {
            match[v] = u;
            match[u] = v;
            return true;
        }
    }
    return false;
}

int bipartite_matching() {
    int res = 0;
    memset(match, -1, sizeof match);
    for (int i = 0; i < V; i++) {
        if (match[i] != -1) continue;
        memset(used, 0, sizeof used);
        res += dfs(i);
    }
    return res;
}
```

1.2 Dinic

Usage: Use `init` to init, `add_edge` to add edges, `max_flow` to find max flow from source s to sink t.

Time Complexity: $\mathcal{O}(V^2E)$

```
struct edge {
    int to, cap, rev;
};

vector<edge> G[MAXV];
int level[MAXV], iter[MAXV];
int n;

void add_edge(int from, int to, int cap) {
    G[from].push_back((edge) {
        to, cap, (int)G[to].size()
    });
    G[to].push_back((edge) {
        from, 0, (int)G[from].size() - 1
    });
}

void bfs(int s) {
    memset(level, -1, sizeof(level));
    queue <int> q;
    level[s] = 0;
    q.push(s);
    while (!q.empty()) {
        int v = q.front();
        q.pop();
        for (int i = 0; i < G[v].size(); i++) {
            edge &e = G[v][i];
            if (e.cap > 0 && level[e.to] < 0) {
                level[e.to] = level[v] + 1;
                q.push(e.to);
            }
        }
    }
}

int dfs(int v, int t, int f) {
    if (v == t) return f;
    for (int i = iter[v]; i < G[v].size(); i++) {
        edge &e = G[v][i];
        if (e.cap > 0 && level[v] < level[e.to]) {
            int d = dfs(e.to, t, min(f, e.cap));
            if (d > 0) {
                e.cap -= d;
                G[e.to][e.rev].cap += d;
                return d;
```

```
            } else {
                level[e.to] = -1;
            }
        }
    }
    return 0;
}

int max_flow(int s, int t) {
    int flow = 0;
    for (;;) {
        bfs(s);
        if (level[t] < 0) return flow;
        memset(iter, 0, sizeof iter);
        int f;
        while ((f = dfs(s, t, INF)) > 0) flow += f;
    }
}
```

1.3 Ford Fulkerson

Usage: Use `init` to init, `add_edge` to add edges, `max_flow` to find max flow from source s to sink t.
Time Complexity: $\mathcal{O}(FE)$

```
struct edge {
    int to, cap, rev;
};

vector<edge> G[MAXV];
bool used[MAXV];
int n;

void add_edge(int from, int to, int cap) {
    G[from].push_back((edge) {
        to, cap, (int)G[to].size()
    });
    G[to].push_back((edge) {
        from, 0, (int)G[from].size() - 1
    });
}

int dfs(int v, int t, int f) {
    if (v == t) return f;
    used[v] = true;
    for (edge e : G[v]) {
        if (e.cap > 0 && !used[e.to]) {
            int d = dfs(e.to, t, min(f, e.cap));
            if (d > 0) {
                e.cap -= d;
                G[e.to][e.rev].cap += d;
                return d;
            }
        }
    }
    return 0;
}

int max_flow(int s, int t) {
    int flow = 0;
    for (;;) {
        memset(used, 0, sizeof used);
        int f = dfs(s, t, INF);
        if (f == 0) return flow;
        flow += f;
    }
}
```

1.4 Min Cost Max Flow

Usage: Use `add_edge` to add edges, `min_cost_flow` to get flow from source s to sink t which flow is f, please set `V` to number of vertices in graph.
Time Complexity: $\mathcal{O}(FEV)$

```
struct edge {
    int to, cap, cost, rev;
};

vector<edge> G[MAXV];
int h[MAXV], dist[MAXV], prevv[MAXV], preve[MAXV];
int V; // Please set the number of V (total number of nodes)!
```

```
void add_edge(int from, int to, int cap, int cost) {
    G[from].push_back((edge) {
        to, cap, cost, (int)G[to].size()
    });
    G[to].push_back((edge) {
        from, 0, -cost, (int)G[from].size() - 1
    });
}

int min_cost_flow(int s, int t, int f, int V) {
    int res = 0;
    while (f) {
        fill(dist, dist + V, INF);
        dist[s] = 0;
        bool update = true;
        while (update) {
            update = false;
            for (int v = 0; v < V; v++) {
                if (dist[v] == INF) continue;
                for (int i = 0; i < G[v].size(); i++) {
                    edge &e = G[v][i];
                    if (e.cap > 0 && dist[e.to] > dist[v] + e.cost)
                    {
                        dist[e.to] = dist[v] + e.cost;
                        prevv[e.to] = v;
                        preve[e.to] = i;
                        update = true;
                    }
                }
            }
        }
        if (dist[t] == INF) return -1;
        int d = f;
        for (int v = t; v != s; v = prevv[v])
            d = min(d, G[prevv[v]][preve[v]].cap);
        f -= d;
        res += d * dist[t];
        for (int v = t; v != s; v = prevv[v]) {
            edge &e = G[prevv[v]][preve[v]];
            e.cap -= d;
            G[v][e.rev].cap += d;
        }
    }
    return res;
}

int main() {
    V = 0;// !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
    return 0;
}
```

2 Math

2.1 Extended GCD and others

```
LL ex_gcd(LL a, LL b, LL &x, LL &y){
    if (b == 0)
    {
        x = 1, y = 0;
        return a;
    } else {
        LL g = ex_gcd(b, a % b, x, y);
        LL t = x;
        x = y, y = t - a / b * x;
        return g;
    }
}
ax ≡ b(mod n), n > 0
void modularlinearequationsolver(int a, int b, int n)
{
    int d, x, y, e, i;
    d = ex_gcd(a, n, x, y);
    if (b % d != 0) cout << "No answer !" ;
    else
    {
        e = x (b / d) % n; // x=e is a basic solution
        for (i = 0; i < d; i++)
            cout << (e + i (n / d)) % n << endl;
    }
}
```

```
Given bi, wi, i = 0...len 1 which wi > 0, i = 0...len 1 and (wi,
wj) = 1, i != j Find an x which satisfies : x ≡ bi(mod wi), i =
0...len 1
int china(int b[], int w[], int len)
{
    int i, d, x, y, x, m, n;
    x = 0;
    n = 1;
    for (i = 0; i < len; i++)
        n= w[i];
    for (i = 0; i < len; i++)
    {
        m = n / w[i];
        d = ex_gcd(w[i], m, x, y);
        x = (x + ymb[i]) % n;
    }
    return (n + x % n) % n;
}
```

2.2 FFT

Usage: Simply put polynomial(`vector`) a, b into `multiply`. The variable `res` stores return answer. It is a MOD version.

Time Complexity: $\mathcal{O}(NlogN)$

```
const double PI = acos(-1.0);

struct base {
    double a, b;
    base() { a = 0, b = 0; }
    base(double a, double b) : a(a), b(b) {}
    base operator + (const base& y) const { return (base) {a + y.a,
    b + y.b}; }
    base operator - (const base& y) const { return (base) {a - y.a,
    b - y.b}; }
    base operator * (const base& y) const { return (base) {a * y.a -
    b * y.b, a * y.b + b * y.a}; }
    base operator ! () const { return (base) {a, -b}; }
};

void fft(vector<base> & a, bool invert) {
    int n = (int)a.size(), lg_n = 0;
    for (int i = 1, j = 0; i < n; ++i) {
        int bit = n >> 1;
        for (; j >= bit; bit >>= 1) j -= bit;
        j += bit;
        if (i < j) swap(a[i], a[j]);
    }
    for (int len = 2; len <= n; len <<= 1) {
        double ang = 2 * PI / len * (invert ? -1 : 1);
        base wlen (cos(ang), sin(ang));
        for (int i = 0; i < n; i += len) {
            base w(1, 0);
            for (int j = 0; j < len / 2; j++) {
                base u = a[i + j],  v = a[i + j + len / 2] * w;
                a[i + j] = u + v;
                a[i + j + len / 2] = u - v;
                w = w * wlen;
            }
        }
    }
    if (invert) {
        for (int i = 0; i < n; i++) {
            a[i].a /= n;
            a[i].b /= n;
        }
    }
}

void multiply(const vector<int>& a, const vector<int>& b,
vector<int>& res) {
    vector<base> fa, fb;
    for (auto x : a) fa.push_back(base(x, 0));
    for (auto x : b) fb.push_back(base(x, 0));
    size_t n = 1;
    while (n < max (a.size(), b.size()))  n <<= 1;
    n <<= 1;
    fa.resize(n), fb.resize(n);
    fft(fa, false),  fft(fb, false);
    for (size_t i = 0; i < n; ++i) {
        fa[i] = fa[i] * fb[i];
```

```
    }
    fft(fa, true);
    res.resize(n);
    for (size_t i = 0; i < n; ++i) {
        res[i] = (long long)(fa[i].a + 0.5) % MOD;
    }
}
```

2.3 Mod

Usage: Combine a lot modular operations.
Time Complexity: $\mathcal{O}(logN)$

```
typedef long long ll;

const int MOD = 1e9 + 7;
const int N = 1e5 + 5;

int f[N], inv[N], finv[N];
// inv[]: inverse element

int powMod(int u, int v) {
    int res = 1;
    while (v) {
        if (v & 1) res = (ll)res * u % MOD;
        v >>= 1;
        u = (ll) u * u % MOD;
    }
    return res;
}

void initInv() {
    f[0] = 1;
    for (int i = 1; i < N; i++) {
        f[i] = (ll)f[i - 1] * i % MOD;
    }
    inv[0] = inv[1] = 1;
    for (int i = 2; i < N; i++) {
        inv[i] = (ll)(MOD - MOD / i) * inv[MOD % i] % MOD;
    }
    finv[0] = finv[1] = 1;
    for (int i = 2; i < N; i++) {
        finv[i] = ((ll)finv[i - 1] * inv[i]) % MOD;
    }
}

int C(int x, int y) {
    if (y < 0) return 0;
    return ((ll)f[x] * finv[y] % MOD) * finv[x - y] % MOD;
}

int Lucas(int u, int v) {
    if (v == 0) return 1;
    return (ll)C(u % MOD, v % MOD) * Lucas(u / MOD, v / MOD) % MOD;
}
```

2.4 Gaussian Elimination

Usage: With both real number and binary version. One have to set n, m before using binary one.
Time Complexity: $\mathcal{O}(N^3)$

```
int gauss (vector < vector<double> > a, vector<double> & ans) {
  int n = (int) a.size(), m = (int) a[0].size() - 1;
  vector<int> where (m, -1);
  for (int col=0, row=0; col<m && row<n; ++col) {
    int sel = row;
    for (int i=row; i<n; ++i)
      if (abs (a[i][col]) > abs (a[sel][col]))
        sel = i;
    if (abs (a[sel][col]) < EPS)
      continue;
    for (int i=col; i<=m; ++i)
      swap (a[sel][i], a[row][i]);
    where[col] = row;

    for (int i=0; i<n; ++i)
      if (i != row) {
        double c = a[i][col] / a[row][col];
        for (int j=col; j<=m; ++j)
          a[i][j] -= a[row][j] * c;
      }
```

```
    ++row;
  }
  ans.assign (m, 0);
  for (int i=0; i<m; ++i)
    if (where[i] != -1)
      ans[i] = a[where[i]][m] / a[where[i]][i];
  for (int i=0; i<n; ++i) {
    double sum = 0;
    for (int j=0; j<m; ++j)
      sum += ans[j] * a[i][j];
    if (abs (sum - a[i][m]) > EPS)
      return 0;
  }
  for (int i=0; i<m; ++i)
    if (where[i] == -1)
      return INF;
  return 1;
}

//SLAE modulo
int gauss(vector < bitset<N> > a, int n, int m, bitset<N> & ans) {
  vector<int> where (m, -1);
  for (int col=0, row=0; col<m && row<n; ++col) {
    for (int i=row; i<n; ++i)
      if (a[i][col]) {
        swap (a[i], a[row]);
        break;
      }
    if (! a[row][col])
      continue;
    where[col] = row;

    for (int i=0; i<n; ++i)
      if (i != row && a[i][col])
        a[i] ^= a[row];
    ++row;
    }
}
```

2.5 Cantor Expansion

Usage: Hash a permutation into a int64.
Time Complexity: $\mathcal{O}(N)$

```
bool vis[N], used[N];
int64 fac[N];

void init() {
    for (int i = 2; i < N; i++) fac[i] = fac[i - 1] * i;
}

int64 KT(vector<int> v, int len) {
    if (len == 0) return 0;
    int64 sum = 0;
    for (int i = 0; i < len; i++) {
        int t = 0;
        for (int j = i + 1; j < len; j++) {
            if (v[j] < v[i]) t++;
        }
        sum += (int64)t * fac[len - i - 1];
    }
    return ++sum;
}

vector<int> invKT(int u, int now) {
    vector<int> res;
    memset(vis, 0, sizeof(vis));
    now--;
    for (int i = 1; i <= u; i++) {
        int64 t = (int64)now / fac[u - i];
        for (int j = 1; j <= u; j++) {
            if (!vis[j]) {
                if (!t) {
                    res.push_back(j);
                    vis[j] = 1;
                    break;
                }
                t--;
            }
        }
```

```
        now %= fac[u - i];
    }
    return res;
}
```

2.6 Simpson Algorithm

Usage: Integration.
Time Complexity: $\mathcal{O}(Dependsonyou)$

```
double f(double x) {
    return x;
}

double cal(double a, double b) {
    double h = (b - a) / N, s = 0;
    for (int i = 0; i <= N; ++i) {
        double x = a + h * i;
        s += f(x) * ((i == 0 || i == N) ? 1 : ((i & 1) == 0) ? 2 :
        4);
    }
    s *= h / 3;
    return s;
}
```

2.7 Euler sieve

Usage: Compute all euler function by using sieve.
Time Complexity: $\mathcal{O}(N)$

```
long long phi[N];

void sieve() {
  memset(phi, 0, sizeof(phi));
  for (int i = 2; i < N; i++) {
    if (!phi[i]) {
      for (int j = i; j < N; j += i) {
        if (!phi[j]) {
          phi[j] = j;
        }
        phi[j] = phi[j] / i * (i - 1);
      }
    }
  }
}
```

3 Graph

3.1 Strong Connected Component (2-SAT)

Usage: For 2-SAT, reduce to CNF like $(x1 \lor x2) \land (y1 \lor y2)$ using `add_edge` to add $(\neg x1 \Rightarrow x2)$ $(\neg x2 \Rightarrow x1)$ into the graph for each clause.
Time Complexity: for each operation $\mathcal{O}(N)$

```
bool used[MAXV];
int V, component_id[MAXV];
vector<int> G[MAXV], rG[MAXV], vs;

void add_edge(int from, int to) {
    G[from].pb(to), rG[to].pb(from);
}

void dfs(int v) {
    used[v] = true;
    for (int i = 0; i < G[v].size(); i++) {
        if (!used[G[v][i]])
            dfs(G[v][i]);
    }
    vs.pb(v);
}

void rdfs(int v, int k) {
    used[v] = true;
    component_id[v] = k;
    for (int i = 0; i < rG[v].size(); i++) {
        if (!used[rG[v][i]]) rdfs(rG[v][i], k);
    }
}

int scc() {
    memset(used, 0, sizeof used);
```

```
    vs.clear();
    for (int i = 0; i < V; i++) if (!used[i]) dfs(i);
    memset(used, 0, sizeof used);
    int k = 0;
    for (int i = vs.size() - 1; i >= 0; i--) {
        if (!used[vs[i]]) rdfs(vs[i], k++);
    }
    return k;
}
```

3.2 Heavy Light Decomposition

Time Complexity: for each operation $\mathcal{O}(logN)$

```
vector<int> G[N];
int n, m, len;
int fa[20][N], sz[N], depth[N], top[N], id[N];
ll t[N << 2], lazy[N << 2];

void dfs1(int u, int f, int d) {
  depth[u] = d;
  fa[0][u] = f;
  sz[u] = 1;
  for (int i = 0; i < G[u].size(); i++) {
    if (f == G[u][i]) continue;
    dfs1(G[u][i], u, d + 1);
    sz[u] += sz[G[u][i]];
  }
}

void dfs2(int u, int to, int f) {
  if (u == 0) return ;
  id[u] = ++len;
  top[u] = to;
  pair<int, int> mx = make_pair(0, 0);
  for (int i = 0; i < G[u].size(); i++) {
    if (f == G[u][i]) continue;
    mx = max(mx, make_pair(sz[G[u][i]], G[u][i]));
  }
  dfs2(mx.second, to, u);
  for (int i = 0; i < G[u].size(); i++) {
    if (f == G[u][i] || mx.second == G[u][i]) continue;
    dfs2(G[u][i], G[u][i], u);
  }
}

void pushup(int rt) {
  t[rt] = t[rt << 1] + t[rt << 1 | 1];
}

void pushdown(int rt, int l) {
  if (lazy[rt]) {
    lazy[rt << 1] += lazy[rt];
    lazy[rt << 1 | 1] += lazy[rt];
    t[rt << 1] += lazy[rt] * ((l + 1) / 2);
    t[rt << 1 | 1] += lazy[rt]* (l - (l + 1) / 2);
    lazy[rt] = 0;
  }
}

void build(int rt, int left, int right) {
  if (left == right) {
    t[rt] = 0;
    return ;
  }
  build(lson), build(rson);
  pushup(rt);
}

ll query(int rt, int left, int right, int l, int r) {
  if (left == l && r == right) {
    return t[rt];
  }
  pushdown(rt, right - left + 1);
  if (mid >= r) return query(lson, l, r);
  else if (mid < l) return query(rson, l, r);
  else return query(lson, l, mid) + query(rson, mid + 1, r);
}

void add(int rt, int left, int right, int l, int r) {
  if (left == l && r == right) {
```

```
    lazy[rt]++;
    t[rt] += r - l + 1;
    return ;
  }
  pushdown(rt, right - left + 1);
  if (mid >= r) add(lson, l, r);
  else if (mid < l) add(rson, l, r);
  else add(lson, l, mid), add(rson, mid + 1, r);
  pushup(rt);
}

void init_HLD() {
  dfs1(1, -1, 1);
  for (int i = 1; i < 20; i++) {
    for (int j = 1; j <= n; j++) {
      if (fa[i - 1][j] == -1) fa[i][j] = -1;
      else fa[i][j] = fa[i - 1][fa[i - 1][j]];
    }
  }
  dfs2(1, 1, -1);
  build(1, 1, len);
}
```

3.3 Articulation Points

Time Complexity: $\mathcal{O}(N)$

```
// Articulation points and Bridges O(V+E)
int par[N], art[N], low[N], num[N], ch[N], cnt;

void articulation(int u) {
  low[u] = num[u] = ++cnt;
  for (int v : adj[u]) {
    if (!num[v]) {
      par[v] = u; ch[u]++;
      articulation(v);
      if (low[v] >= num[u]) art[u] = 1;
      if (low[v] >  num[u]) // u-v bridge
      low[u] = min(low[u], low[v]);
    }
    else if (v != par[u]) low[u] = min(low[u], num[v]);
  }
}

for (int i = 0; i < n; ++i) if (!num[i])
  articulation(i), art[i] = ch[i]>1;
```

3.4 Eulerian Path

Time Complexity: $\mathcal{O}(Nlogn)$

```
vector<int> ans, adj[N];
int in[N];
void dfs(int v){
  while(adj[v].size()){
    int x = adj[v].back();
    adj[v].pop_back(); dfs(x);
  }
  ans.pb(v);
}
// Verify if there is an eulerian path or circuit
vector<int> v;
for(int i = 0; i < n; i++) if(adj[i].size() != in[i]){
  if(abs((int)adj[i].size() - in[i]) != 1) //-> There is no valid
  eulerian circuit/path
  v.pb(i);
}
if(v.size()){
  if(v.size() != 2) //-> There is no valid eulerian path
  if(in[v[0]] > adj[v[0]].size()) swap(v[0], v[1]);
  if(in[v[0]] > adj[v[0]].size()) //-> There is no valid eulerian
  path
  adj[v[1]].pb(v[0]); // Turn the eulerian path into a eulerian
  circuit
}
dfs(0);
for(int i = 0; i < cnt; i++)
  if(adj[i].size()) //-> There is no valid eulerian circuit/path in
  this case because the graph is not conected
ans.pop_back(); // Since it's a curcuit, the first and the last are
repeated
reverse(ans.begin(), ans.end());
```

```
int bg = 0; // Is used to mark where the eulerian path begins
if(v.size()){
  for(int i = 0; i < ans.size(); i++)
    if(ans[i] == v[1] and ans[(i + 1)%ans.size()] == v[0]) { bg = i
    + 1; break;}
}
```

4 Data Structure

4.1 Convex Hull Trick

Usage: Lines adding into the vector have to be **sorted**. Default is for increasing slope and get the maximum.
Time Complexity: for each query $\mathcal{O}(logN)$

```
struct Line {
      long long a, b;
      long long get(long long x) {
            return a * x + b;
      }
};

struct ConvexHull {
    int size;
    vector<Line> hull;

    ConvexHull() { this->clear(); }
    void clear() {
        hull.clear(), size = 0;
    }
    bool is_bad(long long curr, long long prev, long long next) {
        Line c = hull[curr], p = hull[prev], n = hull[next];
        return (c.b - n.b) * (c.a - p.a) <= (p.b - c.b) * (n.a -
        c.a);
    }
    void add_line(long long a, long long b) {
        hull.push_back((Line){a, b}), size += 1;
        while (size > 2 && is_bad(size - 2, size - 3, size - 1))
            hull[size - 2] = hull[size - 1], size--,
            hull.pop_back();
    }
    long long query(long long x) {
        int l = -1, r = size - 1;
        while (r - l > 1) {
                int m = (l + r) / 2;
                if (hull[m].get(x) <= hull[m + 1].get(x)) l = m;
                else r = m;
        }
        return hull[r].get(x);
    }
};
```

4.2 Sqrt Decomposition

Usage: Just in case you forget it.
Time Complexity: for each operation $\mathcal{O}(\sqrt{N})$

```
int B, a[N], b[N];

void init( int n) {
    B = (int) sqrt (n + .0) + 1;
    for (int i=0; i < n; ++i)
        b[i / B] += a[i];
}

int cal(int l, int r) {
    int sum = 0, c_l = l / B,   c_r = r / B;
    if (c_l == c_r)
        for (int i=l; i<=r; ++i) sum += a[i];
    else {
        for (int i=l, end=(c_l+1)*B-1; i<=end; ++i) sum += a[i];
        for (int i=c_l+1; i<=c_r-1; ++i) sum += b[i];
        for (int i=c_r*B; i<=r; ++i) sum += a[i];
    }
}
```

4.3 Centroid Decomposition

Time Complexity: $\mathcal{O}(NLogn)$

```
vector<int> adj[N];
int forb[N], sz[N], par[N], n, m;
```

```
unordered_map<int, int> dist[N];

void dfs(int u, int p) {
  sz[u] = 1;
  for(int v : adj[u]) {
    if(v != p and !forb[v]) {
      dfs(v, u);
      sz[u] += sz[v];
}}}

int find_cen(int u, int p, int qt) {
  for(int v : adj[u]) {
    if(v == p or forb[v]) continue;
    if(sz[v] > qt / 2) return find_cen(v, u, qt);
  }
  return u;
}

void getdist(int u, int p, int cen) {
  for(int v : adj[u]) {
    if(v != p and !forb[v]) {
      dist[cen][v] = dist[v][cen] = dist[cen][u] + 1;
      getdist(v, u, cen);
}}}

void decomp(int u, int p) {
  dfs(u, -1);
  int cen = find_cen(u, -1, sz[u]);
  forb[cen] = 1;
  par[cen] = p;
  dist[cen][cen] = 0;
  getdist(cen, -1, cen);
  for(int v : adj[cen]) if(!forb[v]) decomp(v, cen);
}
decomp(1, -1);
```

4.4 BIT 2D

Time Complexity: for each operation $\mathcal{O}(log^2(n))$

```
int bit[N][N];

void add(int i, int j, int v) {
  for (; i < N; i+=i&-i)
    for (int jj = j; jj < N; jj+=jj&-jj)
      bit[i][jj] += v;
}
int query(int i, int j) {
  int res = 0;
  for (; i; i-=i&-i)
    for (int jj = j; jj; jj-=jj&-jj)
      res += bit[i][jj];
  return res;
}
// Whole BIT 2D set to 1
void init() {
  cl(bit,0);
  for (int i = 1; i <= r; ++i) for (int j = 1; j <= c; ++j)
      add(i, j, 1);
}

// Return number of positions set
int query(int imin, int jmin, int imax, int jmax) {
  return query(imax, jmax) - query(imax, jmin-1) - query(imin-1,
  jmax) + query(imin-1, jmin-1);
}

// Find all positions inside rect (imin, jmin), (imax, jmax) where
position is set
void proc(int imin, int jmin, int imax, int jmax, int v, int tot) {
  if (tot < 0) tot = query(imin, jmin, imax, jmax);
  if (!tot) return;
  int imid = (imin+imax)/2, jmid = (jmin+jmax)/2;
  if (imin != imax) {
    int qnt = query(imin, jmin, imid, jmax);
    if (qnt) proc(imin, jmin, imid, jmax, v, qnt);
    if (tot-qnt) proc(imid+1, jmin, imax, jmax, v, tot-qnt);
  } else if (jmin != jmax) {
    int qnt = query(imin, jmin, imax, jmid);
    if (qnt) proc(imin, jmin, imax, jmid, v, qnt);
```

```
    if (tot-qnt) proc(imin, jmid+1, imax, jmax, v, tot-qnt);
  } else {
    // single position set!
    // now process position!!!
    add(imin, jmin, -1);
  }
}
```

4.5 Treap

Usage: Implicit treap, with reversal
Time Complexity: for each operation $\mathcal{O}(log(n))$

```
typedef struct item * pitem;
struct item {
    int prior, value, cnt;
    bool rev;
    pitem l, r;
};

int cnt (pitem it) {
    return it ? it->cnt : 0;
}

void upd_cnt (pitem it) {
    if (it)
        it->cnt = cnt(it->l) + cnt(it->r) + 1;
}

void push (pitem it) {
    if (it && it->rev) {
        it->rev = false;
        swap (it->l, it->r);
        if (it->l)  it->l->rev ^= true;
        if (it->r)  it->r->rev ^= true;
    }
}

void merge (pitem & t, pitem l, pitem r) {
    push (l);
    push (r);
    if (!l || !r)
        t = l ? l : r;
    else if (l->prior > r->prior)
        merge (l->r, l->r, r),  t = l;
    else
        merge (r->l, l, r->l),  t = r;
    upd_cnt (t);
}

void split (pitem t, pitem & l, pitem & r, int key, int add = 0) {
    if (!t)
        return void( l = r = 0 );
    push (t);
    int cur_key = add + cnt(t->l);
    if (key <= cur_key)
        split (t->l, l, t->l, key, add),  r = t;
    else
        split (t->r, t->r, r, key, add + 1 + cnt(t->l)),  l = t;
    upd_cnt (t);
}

void reverse (pitem t, int l, int r) {
    pitem t1, t2, t3;
    split (t, t1, t2, l);
    split (t2, t2, t3, r-l+1);
    t2->rev ^= true;
    merge (t, t1, t2);
    merge (t, t, t3);
}
```

5 String

5.1 KMP

```
char P[N], T[N];
int pi[N], n, m; // n = len(T), m = len(P)
vector<int> prefix_function (string) {
  pi[0] = -1;
    int k = -1;
    for(int i = 1; i <= m; i++) {
```

```
        while(k >= 0 && P[k+1] != P[i]) k = pi[k];
        pi[i] = ++k;
    }
}
void match() {
    int k = 0;
    for(int i = 1; i <= n; i++) {
        while(k >= 0 && P[k+1] != T[i]) k = pi[k];
        k++;
        if(k == m) k = pi[k];
    }
}
```

5.2 Suffix Array

```
int sa[N], wa[N], wb[N], cnt[N], rank[N], height[N];
void getSA(int *sz, int len){
    int *x = wa, *y = wb, m = 27;
    for (int i = 1; i <= len; i++) cnt[x[i] = sz[i]]++;
    for (int i = 1; i <= m; i++) cnt[i] += cnt[i - 1];
    for (int i = len; i; i--) sa[cnt[x[i]]--] = i;
    for (int h = 1; h <= len; h <<= 1){
        int pos = 0;
        for (int i = len - h + 1; i <= len; i++) y[++pos] = i;
        for (int i = 1; i <= len; i++) if (sa[i] > h) y[++pos] =
        sa[i] - h;
        memset(cnt, 0, sizeof cnt);
        for (int i = 1; i <= len; i++) cnt[x[i]]++;
        for (int i = 1; i <= m; i++) cnt[i] += cnt[i - 1];
        for (int i = len; i; i--) sa[cnt[x[y[i]]]--] = y[i];
        swap(x, y); pos = 0; x[sa[1]] = ++pos;
        for (int i = 2; i <= len; i++)
          x[sa[i]] = y[sa[i]] == y[sa[i - 1]] && y[sa[i] + h] ==
          y[sa[i - 1] + h] ? pos : ++pos;
        m = pos;
        if (m == len) break;
    }
}
void getHeight(int *sz, int len){
    for (int i = 1; i <= len; i++) rank[sa[i]] = i;
    int k = 0;
    for (int i = 1; i <= len; i++){
        if (k) k--;
        int p = sa[rank[i] - 1];
        while (sz[p + k] == sz[i + k]) k++;
        height[rank[i]] = k;
    }
}
```

5.3 Aho Corasick

```
struct AhoCorasick {
    enum{alpha = 26, first = 'A'};
    struct Node {
        int back, next[SZ], start = -1, end = -1, nmatches = 0;
        Node(int v) {
            memset(next, v, sizeof next);
        }
    };
    vector<Node> N;
    vector<int> backp;
    void insert(string& s, int j) {
        int n = 0;
        for (int i = 0; i < s.length(); i++) {
            char c = s[i];
            int& m = N[n].next[c - first];
            if (m == -1) {n = m = N.size(); N.emplace_back(-1);}
            else n = m;
        }
        if (N[n].end == -1) N[n].start = j;
        backp.push_back(N[n].end);
        N[n].end = j;
        N[n].nmatches++;
    }
    AhoCorasick(vector<string> &pat) {
        N.emplace_back(-1);
        for (int i = 0; i < pat.size(); i++) {
            insert(pat[i], i);
        }
        N[0].back = N.size();
```

```
    N.emplace_back(0);
    queue<int> q;
    for (q.push(0); !q.empty(); q.pop()) {
        int n = q.front(), prev = N[n].back;
        for (int i = 0; i < alpha; i++) {
            int& ed = N[n].next[i], y = N[prev].next[i];
            if (ed == -1) ed = y;
            else {
                N[ed].back = y;
                (N[ed].end == -1 ? N[ed].end :
                backp[N[ed].start]) = N[y].end;
                N[ed].nmatches += N[y].nmatches;
                q.push(ed);
            }
        }
    }
}
vector<int> find(string word) {
    int n = 0;
    vector<int> res; // ll count = 0;
    for (int i = 0; i < word.length(); i++) {
        char c = word[i];
        n = N[n].next[c - first];
        res.push_back(N[n].end);
        // count += N[n].nmatches;
    }
    return res;
}
vector<vector<int>> findAll(vector<string>& pat, string word) {
    vector<int> r = find(word);
    vector<vector<int>> res(word.length();
    for (int i = 0; i < word.length(); i++) {
        int ind = r[i];
        while (ind != -1) {
            res[i - pat[ind].size() + 1].push_back(ind);
            ind = backp[ind];
        }
    }
    return res;
}
};
```

6 Geometry

6.1 Point & Segment

Usage: Contain a lot of functions for points and segment.

```
struct Point {
  double x, y;
  Point(double x=0, double y=0) : x(x), y(y) {}
  Point operator+(const Point &o) const { return Point(x+o.x,
  y+o.y); }
  Point operator-(const Point &o) const { return Point(x-o.x,
  y-o.y); }
  Point operator*(const double m) const { return Point(x*m, y*m); }
  Point operator/(const double d) const { return Point(x/d, y/d); }
  bool operator<(const Point &o) const { return x != o.x ? x < o.x :
  y < o.y; }
  bool operator==(const Point &o) const { return fabs(x-o.x) < EPS
  && fabs(y-o.y) < EPS; }
  double cross(const Point &o) const { return x * o.y - y * o.x; }
  double dot(const Point &o) const { return x * o.x + y * o.y; }
  double atan() const { return atan2(y, x); }
  double norm() const { return sqrt(dot(*this)); }
  double distance(const Point &o) const { return (o -
  (*this)).norm(); }
  double area(const Point &a,const Point &b) {
    Point p = a - (*this), p2 = b - (*this);
    return p.cross(p2);
  }
  double area_abs(const Point &a,const Point &b) const {
    Point p = a - (*this), p2 = b - (*this);
    return fabs(p.cross(p2)) / 2.0;
  }
  //線分abが自身に含まれているのかどうか判する
  int between(const Point &a,const Point &b) {
    if(area(a,b) != 0) return 0;
    if(a.x != b.x)  return ((a.x <= x) && (x <= b.x) || (a.x >= x)
    && (x >= b.x));
```

```
    else return ((a.y <= y) && (y <= b.y) || (a.y >= y) && (y >=
    b.y));
  }
  double distance_seg(const Point& a,const Point& b) {
    if((b-a).dot(*this-a) < EPS) return (*this-a).norm();
    if((a-b).dot(*this-b) < EPS) return (*this-b).norm();
    return abs((b-a).cross(*this-a)) / (b-a).norm();
  }
  bool hitPolygon(const Point& a,const Point& b,const Point& c) {
    double t = (b-a).cross(*this-b);
    double t2 = (c-b).cross(*this-c);
    double t3 = (a-c).cross(*this-a);
    if((t > 0 && t2 > 0 && t3 > 0) || ( t < 0 && t2 < 0 && t3 < 0))
    {
      return true;
    }
    return false;
  }
  // counter-clockwise
    // [cos,  sin]   [x]
    // [-sin, cos]   [y]
  void rotate(double theta) {
        double tx = x * cos(theta) - y * sin(theta);
        double ty = x * sin(theta) + y * cos(theta);
        x = tx, y = ty;
    }
};

struct Seg {
  Point a,b;
  Seg () : a(Point(0, 0)), b(Point(0, 0)) {}
  Seg (Point a, Point b) : a(a),b(b) {}
  bool isOrthogonal(Seg &s) { return equals((b - a).dot(s.b -
  s.a),0.0); }
  bool isParallel(Seg &s) { return equals((b-a).cross(s.b -
  s.a),0.0); }
  bool isIntersect(Seg &s) {
    if(s.a.between(a,b) || s.b.between(a,b) || a.between(s.a,s.b) ||
    b.between(s.a,s.b)) return true;
    return ((a-b).cross(s.a-a) * (a-b).cross(s.b-a) < EPS) &&
    ((s.b-s.a).cross(a-s.a)*(s.b-s.a).cross(b-s.a) < EPS);
  }
  bool distance(Seg &s) {
    if((*this).isIntersect(s)) return 0.0;
    return min(min(a.distance_seg(s.a,s.b),b.distance_seg(s.a,s.b)),
           min(s.a.distance_seg(a,b),s.b.distance_seg(a,b)));
  }
  Point getCrossPoint(Seg &s) {
    Point p = s.b - s.a;
    double d = abs(p.cross(a-s.a));
    double d2 = abs(p.cross(b-s.a));
    double t = d / (d+d2);
    return a + (b-a)*t;
  }
  Point project(Point &p) {
    Point base = b - a;
    double t = base.dot(p-a) / base.dot(base);
    return a + base * t;
  }
  Point reflect(Point &p) {
    return p + (project(p) - p) * 2.0;
  }
};
```

6.2 Triangle Center

Usage: Conclusions.

```
Point gravity(Point a, Point b, Point c) {
  double x=(a.x+b.x+c.x)/3, y=(a.y+b.y+c.y)/3;
  return Point(x,y);
}
Point incenter(Point a, Point b, Point c) {
     double A=dis(b,c), B=dis(a,c), C=dis(a,b);
     double S=A+B+C;
     double x=(A*a.x+B*b.x+C*c.x)/S, y=(A*a.y+B*b.y+C*c.y)/S;
   return Point(x,y);
}
Point Circum(Point a, Point b, Point c) {
  double x1=a.x,y1=a.y, x2=b.x,y2=b.y, x3=c.x,y3=c.y;
```

```
  double a1=2*(x2-x1), b1=2*(y2-y1), c1=x2*x2+y2*y2-x1*x1-y1*y1;
  double a2=2*(x3-x2), b2=2*(y3-y2), c2=x3*x3+y3*y3-x2*x2-y2*y2;
  double x=(c1*b2-c2*b1)/(a1*b2-a2*b1),
  y=(a1*c2-a2*c1)/(a1*b2-a2*b1);
  return Point(x,y);
}
Point ortho(Point a, Point b, Point c){
  double A1=b.x-c.x, B1=b.y-c.y, C1=A1*a.y-B1*a.x;
  double A2=a.x-c.x, B2=a.y-c.y, C2=A2*b.y-B2*b.x;
  double x=(A1*C2-A2*C1)/(A2*B1-A1*B2),
  y=(B1*C2-B2*C1)/(A2*B1-A1*B2);
  return Point(x,y);
}
```

6.3 Line

Usage: To get function coefficient for every line (normalized)

```
struct Line {
    int a, b, c;
    Line(int x1, int y1, int x2, int y2) {
        a = y1 - y2, b = -(x1 - x2), c = -a * x1 - b * y1;
        int tmp = gcd(abs(c), gcd(abs(a), abs(b)));
        a /= tmp, b /= tmp, c /= tmp;
        if (a < 0 || (a == 0 && b < 0)) a *= -1, b *= -1, c *= -1;
    }
    bool operator < (struct Line l) const {
        return make_tuple(a, b, c) < make_tuple(l.a, l.b, l.c);
    }
};
```

6.4 Convex Hull

Usage: `p` is a pair(STL)-like array (with x, y as its variables) to store all the geometry points.
Time Complexity: for each operation $\mathcal{O}(N \log N)$

```
double cross(struct Point a, struct Point b) {
    return a.x * b.y - a.y * b.x;
}
void convex_hull() {
    sort(p + 1, p + 1 + n);
    for (int i = 1; i <= n; i++) {
        while (l >= 2 && cross(p[lower_hull[l - 1]] - p[lower_hull[l
        - 2]], p[i] - p[lower_hull[l - 1]]) <= 0) l--;
        lower_hull[l++] = i;
    }
    for (int i = n; i >= 1; i--) {
        while (r >= 2 && cross(p[upper_hull[r - 1]] - p[upper_hull[r
        - 2]], p[i] - p[upper_hull[r - 1]]) <= 0) r--;
        upper_hull[r++] = i;
    }
}
```

6.5 Half Plane Intersaction

7 Others

7.1 PBDS

Usage: Policy based data structures

```
#include<ext/pb_ds/assoc_container.hpp>
#include<ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
tree<int,char,less<int>,rb_tree_tag,tree_order_statistics_node_update>
st;
void test() {
  pdl(st.order_of_key(3));//find 0 indexed order of a value

}
//tree_order_statistics_join.cc
typedef
tree<int,char,less<int>,
  splay_tree_tag,
  tree_order_statistics_node_update>
  tree_map_t;

int main()
{
  // Insert some entries into s0.
```

```
  tree_map_t s0;
  s0.insert(make_pair(12,'a'));
  s0.insert(make_pair(505,'b'));
  s0.insert(make_pair(30,'c'));
  // The order of the keys should be: 12, 30, 505.
  assert(s0.find_by_order(0)->first==12);
  // Insert some entries into s1.
  tree_map_t s1;
  s1.insert(make_pair(506,'a'));
  s1.insert(make_pair(1222,'b'));
  s1.insert(make_pair(3004,'a'));

  // Now join s0 and s1. ALL IN s0<s1
  s0.join(s1);
  // The order of the keys should be: 12, 30, 505, 506, 1222, 3004.
  assert(s0.find_by_order(0)->first==12);
}
template<class Node_CItr,
   class Node_Itr,
   class Cmp_Fn,
   class _Alloc>
struct my_node_update
{
    typedef my_type metadata_type;

    void operator()(Node_Itr it, Node_CItr end_it)
    {
        auto l=it.get_l_child();
        auto r=it.get_r_child();
        int left=0,right=0;
        if(l!=end_it) left =l.get_metadata();
        if(r!=end_it) right=r.get_metadata();
        const_cast<int&>(it.get_metadata())=left+right+1;
    }
};

int order_of_key(int x){
    int ans=0;
    auto it=node_begin();
    while(it!=node_end())
    {
        auto l=it.get_l_child();
        auto r=it.get_r_child();
        if(Cmp_Fn()(x,**it))
        {
            it=l;
        }
        else
        {
            ans++;
            if(l!=node_end()) ans+=l.get_metadata();
            it=r;
        }
    }
    return ans;
}
```

7.2 rope

Usage: rope

```
#include<bits/stdc++.h>
#include<bits/extc++.h>
using namespace std;
using __gnu_cxx::crope;
int n,t,v,p,c,d,vcnt;
string s;
crope rp,history[50111];
int main() {
  s.reserve(66666);
  ios::sync_with_stdio(0);
  cin>>n;
  while (n--) {
    cin>>t;
    switch (t) {
    case 1:cin>>p>>s;
      p-=d;
      rp.insert(p,s.data());
      history[++vcnt]=rp;
      break;
    case 2:cin>>p>>c;p-=d;c-=d;
```

```
      rp.erase(--p,c);
      history[++vcnt]=rp;
      break;
    case 3:cin>>v>>p>>c;
      v-=d;p-=d;c-=d;
      auto tt=history[v].substr(--p,c);
      for (auto&&i:tt)if (i=='c')++d;
      cout<<tt<<'\n';
    }
  }
}
```

7.3 DnC DP

Usage: Divide and Conquer DP Optimization
Time Complexity: O(k*n^2) $=> O(k * n * logn)$

```
// dp[i][j] = min k<i { dp[k][j-1] + C[k][i] }
// Condition: A[i][j] <= A[i+1][j]
// A[i][j] is the smallest k that gives an optimal answer to
dp[i][j]
int n, maxj, dp[N][J], a[N][J];
// declare the cost function
void calc(int l, int r, int j, int kmin, int kmax) {
  int m = (l+r)/2;
  dp[m][j] = LINF;
  for (int k = kmin; k <= kmax; ++k) {
    ll v = dp[k][j-1] + cost(k, m);
    // store the minimum answer for d[m][j], in case of maximum, use
    v > dp[m][j]
    if (v < dp[m][j]) a[m][j] = k, dp[m][j] = v;
  }
  if (l < r) { calc(l,   m, j, kmin,    a[m][k]); calc(m+1, r, j,
  a[m][k], kmax   ); }
}
// run for every j
for (int j = 2; j <= maxj; ++j) calc(1, n, j, 1, n);
```

7.4 Knuth Optimization

Time Complexity: O(N^3)$- > O(N^2)$

```
// 1) dp[i][j] = min i<k<j { dp[i][k] + dp[k][j] } + C[i][j]
// 2) dp[i][j] = min k<i { dp[k][j-1] + C[k][i] }
// Condition: A[i][j-1] <= A[i][j] <= A[i+1][j]
// A[i][j] is the smallest k that gives an optimal answer to
dp[i][j]
// 1) dp[i][j] = min i<k<j { dp[i][k] + dp[k][j] } + C[i][j]
int n, dp[N][N], a[N][N];
int cost(int i, int j)  // declare the cost function
void knuth() {
  // calculate base cases
  memset(dp, 63, sizeof(dp));
  for (int i = 1; i <= n; i++) dp[i][i] = 0;
  // set initial a[i][j]
  for (int i = 1; i <= n; i++) a[i][i] = i;
  for (int j = 2; j <= n; ++j)
    for (int i = j; i >= 1; --i)
      for (int k = a[i][j-1]; k <= a[i+1][j]; ++k) {
        ll v = dp[i][k] + dp[k][j] + cost(i, j);
        // store the minimum answer for d[i][k]
        // in case of maximum, use v > dp[i][k]
        if (v < dp[i][j])
          a[i][j] = k, dp[i][j] = v;
      }
}
// 2) dp[i][j] = min k<i { dp[k][j-1] + C[k][i] }
int n, maxj, dp[N][J], a[N][J];
void knuth() {
  // calculate base cases
  memset(dp, 63, sizeof(dp));
  for (int i = 1; i <= n; i++) dp[i][1] = // ...
  // set initial a[i][j]
  for (int i = 1; i <= n; i++) a[i][0] = 0, a[n+1][i] = n;
  for (int j = 2; j <= maxj; j++) for (int i = n; i >= 1; i--)
      for (int k = a[i][j-1]; k <= a[i+1][j]; k++) {
        ll v = dp[k][j-1] + cost(k, i);
        // store the minimum answer for d[i][k]
        // in case of maximum, use v > dp[i][k]
        if (v < dp[i][j]) a[i][j] = k, dp[i][j] = v;
      }
}
```

Contest (1)

template.cpp

15 lines

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

#define rep(i, a, b) for(int i = a; i < (b); ++i)
#define trav(a, x) for(auto& a : x)
#define all(x) x.begin(), x.end()
#define sz(x) (int)(x).size()
typedef long long ll;
typedef pair<int, int> pii;
typedef vector<int> vi;

int main() {
  cin.sync_with_stdio(0); cin.tie(0);
  cin.exceptions(cin.failbit);
}
```

.bashrc

3 lines

```
alias c='g++ -Wall -Wconversion -Wfatal-errors -g -std=c++14 \
  -fsanitize=undefined,address'
xmodmap -e 'clear lock' -e 'keycode 66=less greater' #caps = <>
```

.vimrc

2 lines

```
set cin aw ai is ts=4 sw=4 tm=50 nu noeb bg=dark ru cul
sy on   |   im jk <esc>   |   im kj <esc>   |   no ; :
```

troubleshoot.txt

52 lines

```
Pre-submit:
Write a few simple test cases, if sample is not enough.
Are time limits close? If so, generate max cases.
Is the memory usage fine?
Could anything overflow?
Make sure to submit the right file.

Wrong answer:
Print your solution! Print debug output, as well.
Are you clearing all datastructures between test cases?
Can your algorithm handle the whole range of input?
Read the full problem statement again.
Do you handle all corner cases correctly?
Have you understood the problem correctly?
Any uninitialized variables?
Any overflows?
Confusing N and M, i and j, etc.?
Are you sure your algorithm works?
What special cases have you not thought of?
Are you sure the STL functions you use work as you think?
Add some assertions, maybe resubmit.
Create some testcases to run your algorithm on.
Go through the algorithm for a simple case.
Go through this list again.
Explain your algorithm to a team mate.
Ask the team mate to look at your code.
Go for a small walk, e.g. to the toilet.
Is your output format correct? (including whitespace)
Rewrite your solution from the start or let a team mate do it.

Runtime error:
Have you tested all corner cases locally?
Any uninitialized variables?
Are you reading or writing outside the range of any vector?
Any assertions that might fail?
Any possible division by 0? (mod 0 for example)
Any possible infinite recursion?
Invalidated pointers or iterators?
Are you using too much memory?
Debug with resubmits (e.g. remapped signals, see Various).

Time limit exceeded:
Do you have any possible infinite loops?
What is the complexity of your algorithm?
Are you copying a lot of unnecessary data? (References)
How big is the input and output? (consider scanf)
Avoid vector, map. (use arrays/unordered_map)
What do your team mates think about your algorithm?

Memory limit exceeded:
What is the max amount of memory your algorithm should need?
Are you clearing all datastructures between test cases?
```

Mathematics (2)

2.1 Equations

$$ax^2 + bx + c = 0 \Rightarrow x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

The extremum is given by $x = -b/2a$.

$$\begin{aligned} ax + by &= e \\ cx + dy &= f \end{aligned} \Rightarrow \begin{aligned} x &= \frac{ed - bf}{ad - bc} \\ y &= \frac{af - ec}{ad - bc} \end{aligned}$$

In general, given an equation $Ax = b$, the solution to a variable x_i is given by

$$x_i = \frac{\det A_i'}{\det A}$$

where A_i' is A with the i'th column replaced by b.

2.2 Recurrences

If $a_n = c_1 a_{n-1} + \cdots + c_k a_{n-k}$, and $r_1, \ldots, r_k$ are distinct roots of $x^k + c_1 x^{k-1} + \cdots + c_k$, there are $d_1, \ldots, d_k$ s.t.

$$a_n = d_1 r_1^n + \cdots + d_k r_k^n.$$

Non-distinct roots r become polynomial factors, e.g. $a_n = (d_1 n + d_2) r^n$.

2.3 Trigonometry

$$\sin(v + w) = \sin v \cos w + \cos v \sin w$$
$$cos(v + w) = \cos v \cos w - \sin v \sin w$$

$$\tan(v + w) = \frac{\tan v + \tan w}{1 - \tan v \tan w}$$
$$\sin v + \sin w = 2 \sin \frac{v + w}{2} \cos \frac{v - w}{2}$$
$$\cos v + \cos w = 2 \cos \frac{v + w}{2} \cos \frac{v - w}{2}$$
$$(V + W) \tan(v - w)/2 = (V - W) \tan(v + w)/2$$

where V, W are lengths of sides opposite angles v, w.

$$a \cos x + b \sin x = r \cos(x - \phi)$$
$$a \sin x + b \cos x = r \sin(x + \phi)$$

where $r = \sqrt{a^2 + b^2}, \phi = \operatorname{atan2}(b, a)$.

2.4 Geometry

2.4.1 Triangles

Side lengths: a, b, c

Semiperimeter: $p = \dfrac{a + b + c}{2}$

Area: $A = \sqrt{p(p - a)(p - b)(p - c)}$

Circumradius: $R = \dfrac{abc}{4A}$

Inradius: $r = \dfrac{A}{p}$

Length of median (divides triangle into two equal-area triangles): $m_a = \frac{1}{2}\sqrt{2b^2 + 2c^2 - a^2}$

Length of bisector (divides angles in two):

$$s_a = \sqrt{bc \left[1 - \left(\frac{a}{b + c}\right)^2\right]}$$

Law of sines: $\dfrac{\sin \alpha}{a} = \dfrac{\sin \beta}{b} = \dfrac{\sin \gamma}{c} = \dfrac{1}{2R}$

Law of cosines: $a^2 = b^2 + c^2 - 2bc \cos \alpha$

Law of tangents: $\dfrac{a + b}{a - b} = \dfrac{\tan \frac{\alpha + \beta}{2}}{\tan \frac{\alpha - \beta}{2}}$

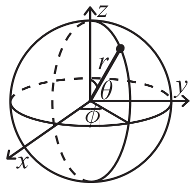
2.4.2 Quadrilaterals

With side lengths a, b, c, d, diagonals e, f, diagonals angle θ, area A and magic flux $F = b^2 + d^2 - a^2 - c^2$:

$$4A = 2ef \cdot \sin \theta = F \tan \theta = \sqrt{4e^2 f^2 - F^2}$$

For cyclic quadrilaterals the sum of opposite angles is 180°, $ef = ac + bd$, and $A = \sqrt{(p - a)(p - b)(p - c)(p - d)}$.

2.4.3 Spherical coordinates



$$\begin{aligned} x &= r\sin\theta\cos\phi & r &= \sqrt{x^2+y^2+z^2} \\ y &= r\sin\theta\sin\phi & \theta &= \text{acos}(z/\sqrt{x^2+y^2+z^2}) \\ z &= r\cos\theta & \phi &= \text{atan2}(y,x) \end{aligned}$$

2.5 Derivatives/Integrals

$$\frac{d}{dx}\arcsin x = \frac{1}{\sqrt{1-x^2}} \qquad \frac{d}{dx}\arccos x = -\frac{1}{\sqrt{1-x^2}}$$

$$\frac{d}{dx}\tan x = 1+\tan^2 x \qquad \frac{d}{dx}\arctan x = \frac{1}{1+x^2}$$

$$\int \tan ax = -\frac{\ln|\cos ax|}{a} \qquad \int x\sin ax = \frac{\sin ax - ax\cos ax}{a^2}$$

$$\int e^{-x^2} = \frac{\sqrt{\pi}}{2}\text{erf}(x) \qquad \int xe^{ax}dx = \frac{e^{ax}}{a^2}(ax-1)$$

Integration by parts:

$$\int_a^b f(x)g(x)dx = [F(x)g(x)]_a^b - \int_a^b F(x)g'(x)dx$$

2.6 Sums

$$c^a + c^{a+1} + \cdots + c^b = \frac{c^{b+1}-c^a}{c-1}, c \neq 1$$

$$\begin{aligned} 1+2+3+\cdots+n &= \frac{n(n+1)}{2} \\ 1^2+2^2+3^2+\cdots+n^2 &= \frac{n(2n+1)(n+1)}{6} \\ 1^3+2^3+3^3+\cdots+n^3 &= \frac{n^2(n+1)^2}{4} \\ 1^4+2^4+3^4+\cdots+n^4 &= \frac{n(n+1)(2n+1)(3n^2+3n-1)}{30} \end{aligned}$$

2.7 Series

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots,\ (-\infty < x < \infty)$$

$$\ln(1+x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \ldots,\ (-1 < x \leq 1)$$

$$\sqrt{1+x} = 1 + \frac{x}{2} - \frac{x^2}{8} + \frac{2x^3}{32} - \frac{5x^4}{128} + \ldots,\ (-1 \leq x \leq 1)$$

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \ldots,\ (-\infty < x < \infty)$$

$$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \ldots,\ (-\infty < x < \infty)$$

2.8 Probability theory

Let X be a discrete random variable with probability $p_X(x)$ of assuming the value x. It will then have an expected value (mean) $\mu = \mathbb{E}(X) = \sum_x xp_X(x)$ and variance $\sigma^2 = V(X) = \mathbb{E}(X^2) - (\mathbb{E}(X))^2 = \sum_x (x-\mathbb{E}(X))^2 p_X(x)$ where σ is the standard deviation. If X is instead continuous it will have a probability density function $f_X(x)$ and the sums above will instead be integrals with $p_X(x)$ replaced by $f_X(x)$.

Expectation is linear:

$$\mathbb{E}(aX+bY) = a\mathbb{E}(X) + b\mathbb{E}(Y)$$

For independent X and Y,

$$V(aX+bY) = a^2V(X) + b^2V(Y).$$

2.8.1 Discrete distributions

Binomial distribution

The number of successes in n independent yes/no experiments, each which yields success with probability p is $\text{Bin}(n,p)$, $n = 1, 2, \ldots$, $0 \leq p \leq 1$.

$$p(k) = \binom{n}{k} p^k (1-p)^{n-k}$$

$$\mu = np,\ \sigma^2 = np(1-p)$$

$\text{Bin}(n,p)$ is approximately $\text{Po}(np)$ for small p.

First success distribution

The number of trials needed to get the first success in independent yes/no experiments, each wich yields success with probability p is $\text{Fs}(p)$, $0 \leq p \leq 1$.

$$p(k) = p(1-p)^{k-1},\ k = 1, 2, \ldots$$

$$\mu = \frac{1}{p},\ \sigma^2 = \frac{1-p}{p^2}$$

Poisson distribution

The number of events occurring in a fixed period of time t if these events occur with a known average rate κ and independently of the time since the last event is $\text{Po}(\lambda)$, $\lambda = t\kappa$.

$$p(k) = e^{-\lambda}\frac{\lambda^k}{k!}, k = 0, 1, 2, \ldots$$

$$\mu = \lambda,\ \sigma^2 = \lambda$$

2.8.2 Continuous distributions

Uniform distribution

If the probability density function is constant between a and b and 0 elsewhere it is $\text{U}(a,b)$, $a < b$.

$$f(x) = \begin{cases} \frac{1}{b-a} & a < x < b \\ 0 & \text{otherwise} \end{cases}$$

$$\mu = \frac{a+b}{2},\ \sigma^2 = \frac{(b-a)^2}{12}$$

Exponential distribution

The time between events in a Poisson process is $\text{Exp}(\lambda)$, $\lambda > 0$.

$$f(x) = \begin{cases} \lambda e^{-\lambda x} & x \geq 0 \\ 0 & x < 0 \end{cases}$$

$$\mu = \frac{1}{\lambda},\ \sigma^2 = \frac{1}{\lambda^2}$$

```
  d c = (a+b) / 2;
  d S1 = simpson(f, a, c);
  d S2 = simpson(f, c, b), T = S1 + S2;
  if (abs (T - S) <= 15*eps || b-a < 1e-10)
    return T + (T - S) / 15;
  return rec(f, a, c, eps/2, S1) + rec(f, c, b, eps/2, S2);
}
d quad(d (*f)(d), d a, d b, d eps = 1e-8) {
  return rec(f, a, b, eps, simpson(f, a, b));
}
```

Determinant.h

Description: Calculates determinant of a matrix. Destroys the matrix.
Time: $\mathcal{O}(N^3)$

15 lines

```
double det(vector<vector<double>>& a) {
  int n = sz(a); double res = 1;
  rep(i,0,n) {
    int b = i;
    rep(j,i+1,n) if (fabs(a[j][i]) > fabs(a[b][i])) b = j;
    if (i != b) swap(a[i], a[b]), res *= -1;
    res *= a[i][i];
    if (res == 0) return 0;
    rep(j,i+1,n) {
      double v = a[j][i] / a[i][i];
      if (v != 0) rep(k,i+1,n) a[j][k] -= v * a[i][k];
    }
  }
  return res;
}
```

IntDeterminant.h

Description: Calculates determinant using modular arithmetics. Modulos can also be removed to get a pure-integer version.
Time: $\mathcal{O}(N^3)$

18 lines

```
const ll mod = 12345;
ll det(vector<vector<ll>>& a) {
  int n = sz(a); ll ans = 1;
  rep(i,0,n) {
    rep(j,i+1,n) {
      while (a[j][i] != 0) { // gcd step
        ll t = a[i][i] / a[j][i];
        if (t) rep(k,i,n)
          a[i][k] = (a[i][k] - a[j][k] * t) % mod;
        swap(a[i], a[j]);
        ans *= -1;
      }
    }
    ans = ans * a[i][i] % mod;
    if (!ans) return 0;
  }
  return (ans + mod) % mod;
}
```

Simplex.h

Description: Solves a general linear maximization problem: maximize $c^T x$ subject to $Ax \le b$, $x \ge 0$. Returns -inf if there is no solution, inf if there are arbitrarily good solutions, or the maximum value of $c^T x$ otherwise. The input vector is set to an optimal x (or in the unbounded case, an arbitrary solution fulfilling the constraints). Numerical stability is not guaranteed. For better performance, define variables such that $x = 0$ is viable.
Usage: vvd A = {{1,-1}, {-1,1}, {-1,-2}};
vd b = {1,1,-4}, c = {-1,-1}, x;
T val = LPSolver(A, b, c).solve(x);
Time: $\mathcal{O}(NM * \#pivots)$, where a pivot may be e.g. an edge relaxation. $\mathcal{O}(2^n)$ in the general case.

68 lines

```
typedef double T; // long double, Rational, double + mod<P>...
typedef vector<T> vd;
typedef vector<vd> vvd;

const T eps = 1e-8, inf = 1/.0;
#define MP make_pair
#define ltj(X) if(s == -1 || MP(X[j],N[j]) < MP(X[s],N[s])) s=j

struct LPSolver {
  int m, n;
  vi N, B;
  vvd D;

  LPSolver(const vvd& A, const vd& b, const vd& c) :
    m(sz(b)), n(sz(c)), N(n+1), B(m), D(m+2, vd(n+2)) {
      rep(i,0,m) rep(j,0,n) D[i][j] = A[i][j];
      rep(i,0,m) { B[i] = n+i; D[i][n] = -1; D[i][n+1] = b[i];}
      rep(j,0,n) { N[j] = j; D[m][j] = -c[j]; }
      N[n] = -1; D[m+1][n] = 1;
    }

  void pivot(int r, int s) {
    T *a = D[r].data(), inv = 1 / a[s];
    rep(i,0,m+2) if (i != r && abs(D[i][s]) > eps) {
      T *b = D[i].data(), inv2 = b[s] * inv;
      rep(j,0,n+2) b[j] -= a[j] * inv2;
      b[s] = a[s] * inv2;
    }
    rep(j,0,n+2) if (j != s) D[r][j] *= inv;
    rep(i,0,m+2) if (i != r) D[i][s] *= -inv;
    D[r][s] = inv;
    swap(B[r], N[s]);
  }

  bool simplex(int phase) {
    int x = m + phase - 1;
    for (;;) {
      int s = -1;
      rep(j,0,n+1) if (N[j] != -phase) ltj(D[x]);
      if (D[x][s] >= -eps) return true;
      int r = -1;
      rep(i,0,m) {
        if (D[i][s] <= eps) continue;
        if (r == -1 || MP(D[i][n+1] / D[i][s], B[i])
                     < MP(D[r][n+1] / D[r][s], B[r])) r = i;
      }
      if (r == -1) return false;
      pivot(r, s);
    }
  }

  T solve(vd &x) {
    int r = 0;
    rep(i,1,m) if (D[i][n+1] < D[r][n+1]) r = i;
    if (D[r][n+1] < -eps) {
      pivot(r, n);
      if (!simplex(2) || D[m+1][n+1] < -eps) return -inf;
      rep(i,0,m) if (B[i] == -1) {
        int s = 0;
        rep(j,1,n+1) ltj(D[i]);
        pivot(i, s);
      }
    }
    bool ok = simplex(1); x = vd(n);
    rep(i,0,m) if (B[i] < n) x[B[i]] = D[i][n+1];
    return ok ? D[m][n+1] : inf;
  }
};
```

SolveLinear.h

Description: Solves $A * x = b$. If there are multiple solutions, an arbitrary one is returned. Returns rank, or -1 if no solutions. Data in A and b is lost.
Time: $\mathcal{O}(n^2 m)$

38 lines

```
typedef vector<double> vd;
const double eps = 1e-12;

int solveLinear(vector<vd>& A, vd& b, vd& x) {
  int n = sz(A), m = sz(x), rank = 0, br, bc;
  if (n) assert(sz(A[0]) == m);
  vi col(m); iota(all(col), 0);

  rep(i,0,n) {
    double v, bv = 0;
    rep(r,i,n) rep(c,i,m)
      if ((v = fabs(A[r][c])) > bv)
        br = r, bc = c, bv = v;
    if (bv <= eps) {
      rep(j,i,n) if (fabs(b[j]) > eps) return -1;
      break;
    }
    swap(A[i], A[br]);
    swap(b[i], b[br]);
    swap(col[i], col[bc]);
    rep(j,0,n) swap(A[j][i], A[j][bc]);
    bv = 1/A[i][i];
    rep(j,i+1,n) {
      double fac = A[j][i] * bv;
      b[j] -= fac * b[i];
      rep(k,i+1,m) A[j][k] -= fac*A[i][k];
    }
    rank++;
  }

  x.assign(m, 0);
  for (int i = rank; i--;) {
    b[i] /= A[i][i];
    x[col[i]] = b[i];
    rep(j,0,i) b[j] -= A[j][i] * b[i];
  }
  return rank; // (multiple solutions if rank < m)
}
```

SolveLinear2.h

Description: To get all uniquely determined values of x back from SolveLinear, make the following changes:

"SolveLinear.h" 7 lines

```
rep(j,0,n) if (j != i) // instead of rep(j,i+1,n)
// ... then at the end:
x.assign(m, undefined);
rep(i,0,rank) {
  rep(j,rank,m) if (fabs(A[i][j]) > eps) goto fail;
  x[col[i]] = b[i] / A[i][i];
fail:; }
```

SolveLinearBinary.h

Description: Solves $Ax = b$ over $\mathbb{F}_2$. If there are multiple solutions, one is returned arbitrarily. Returns rank, or -1 if no solutions. Destroys A and b.
Time: $\mathcal{O}(n^2 m)$

34 lines

```
typedef bitset<1000> bs;

int solveLinear(vector<bs>& A, vi& b, bs& x, int m) {
  int n = sz(A), rank = 0, br;
  assert(m <= sz(x));
  vi col(m); iota(all(col), 0);
  rep(i,0,n) {
```

ModSum.h

Description: Sums of mod'ed arithmetic progressions.
`modsum(to, c, k, m)` $= \sum_{i=0}^{to-1} (ki + c)\%m$. `divsum` is similar but for floored division.
Time: $\log(m)$, with a large constant.

21 lines

```
typedef unsigned long long ull;
ull sumsq(ull to) { return to / 2 * ((to-1) | 1); }

ull divsum(ull to, ull c, ull k, ull m) {
  ull res = k / m * sumsq(to) + c / m * to;
  k %= m; c %= m;
  if (k) {
    ull to2 = (to * k + c) / m;
    res += to * to2;
    res -= divsum(to2, m-1 - c, m, k) + to2;
  }
  return res;
}

ll modsum(ull to, ll c, ll k, ll m) {
  c %= m;
  k %= m;
  if (c < 0) c += m;
  if (k < 0) k += m;
  return to * c + k * sumsq(to) - m * divsum(to, c, k, m);
}
```

ModMulLL.h

Description: Calculate $a \cdot b \bmod c$ (or $a^b \bmod c$) for large c.
Time: $\mathcal{O}(64/bits \cdot \log b)$, where $bits = 64 - k$, if we want to deal with k-bit numbers.

19 lines

```
typedef unsigned long long ull;
const int bits = 10;
// if all numbers are less than 2^k, set bits = 64-k
const ull po = 1 << bits;
ull mod_mul(ull a, ull b, ull &c) {
  ull x = a * (b & (po - 1)) % c;
  while ((b >>= bits) > 0) {
    a = (a << bits) % c;
    x += (a * (b & (po - 1))) % c;
  }
  return x % c;
}
ull mod_pow(ull a, ull b, ull mod) {
  if (b == 0) return 1;
  ull res = mod_pow(a, b / 2, mod);
  res = mod_mul(res, res, mod);
  if (b & 1) return mod_mul(res, a, mod);
  return res;
}
```

ModSqrt.h

Description: Tonelli-Shanks algorithm for modular square roots.
Time: $\mathcal{O}(\log^2 p)$ worst case, often $\mathcal{O}(\log p)$

"ModPow.h" 30 lines

```
ll sqrt(ll a, ll p) {
  a %= p; if (a < 0) a += p;
  if (a == 0) return 0;
  assert(modpow(a, (p-1)/2, p) == 1);
  if (p % 4 == 3) return modpow(a, (p+1)/4, p);
  // a^(n+3)/8 or 2^(n+3)/8 * 2^(n-1)/4 works if p % 8 == 5
  ll s = p - 1;
  int r = 0;
  while (s % 2 == 0)
    ++r, s /= 2;
  ll n = 2; // find a non-square mod p
  while (modpow(n, (p - 1) / 2, p) != p - 1) ++n;
  ll x = modpow(a, (s + 1) / 2, p);
  ll b = modpow(a, s, p);
  ll g = modpow(n, s, p);
  for (;;) {
    ll t = b;
    int m = 0;
    for (; m < r; ++m) {
      if (t == 1) break;
      t = t * t % p;
    }
    if (m == 0) return x;
    ll gs = modpow(g, 1 << (r - m - 1), p);
    g = gs * gs % p;
    x = x * gs % p;
    b = b * g % p;
    r = m;
  }
}
```

5.2 Number theoretic transform

NTT.h

Description: Number theoretic transform. Can be used for convolutions modulo specific nice primes of the form $2^a b+1$, where the convolution result has size at most 2^a. For other primes/integers, use two different primes and combine with CRT. May return negative values.
Time: $\mathcal{O}(N \log N)$

"ModPow.h" 38 lines

```
const ll mod = (119 << 23) + 1, root = 3; // = 998244353
// For p < 2^30 there is also e.g. (5 << 25, 3), (7 << 26, 3),
// (479 << 21, 3) and (483 << 21, 5). The last two are > 10^9.

typedef vector<ll> vl;
void ntt(ll* x, ll* temp, ll* roots, int N, int skip) {
  if (N == 1) return;
  int n2 = N/2;
  ntt(x     , temp, roots, n2, skip*2);
  ntt(x+skip, temp, roots, n2, skip*2);
  rep(i,0,N) temp[i] = x[i*skip];
  rep(i,0,n2) {
    ll s = temp[2*i], t = temp[2*i+1] * roots[skip*i];
    x[skip*i] = (s + t) % mod; x[skip*(i+n2)] = (s - t) % mod;
  }
}
void ntt(vl& x, bool inv = false) {
  ll e = modpow(root, (mod-1) / sz(x));
  if (inv) e = modpow(e, mod-2);
  vl roots(sz(x), 1), temp = roots;
  rep(i,1,sz(x)) roots[i] = roots[i-1] * e % mod;
  ntt(&x[0], &temp[0], &roots[0], sz(x), 1);
}
vl conv(vl a, vl b) {
  int s = sz(a) + sz(b) - 1; if (s <= 0) return {};
  int L = s > 1 ? 32 - __builtin_clz(s - 1) : 0, n = 1 << L;
  if (s <= 200) { // (factor 10 optimization for |a|,|b| = 10)
    vl c(s);
    rep(i,0,sz(a)) rep(j,0,sz(b))
      c[i + j] = (c[i + j] + a[i] * b[j]) % mod;
    return c;
  }
  a.resize(n); ntt(a);
  b.resize(n); ntt(b);
  vl c(n); ll d = modpow(n, mod-2);
  rep(i,0,n) c[i] = a[i] * b[i] % mod * d % mod;
  ntt(c, true); c.resize(s); return c;
}
```

5.3 Primality

eratosthenes.h

Description: Prime sieve for generating all primes up to a certain limit. isprime$[i]$ is true iff i is a prime.
Time: lim=100'000'000 $\approx$ 0.8 s. Runs 30% faster if only odd indices are stored.

11 lines

```
const int MAX_PR = 5000000;
bitset<MAX_PR> isprime;
vi eratosthenes_sieve(int lim) {
  isprime.set(); isprime[0] = isprime[1] = 0;
  for (int i = 4; i < lim; i += 2) isprime[i] = 0;
  for (int i = 3; i*i < lim; i += 2) if (isprime[i])
    for (int j = i*i; j < lim; j += i*2) isprime[j] = 0;
  vi pr;
  rep(i,2,lim) if (isprime[i]) pr.push_back(i);
  return pr;
}
```

MillerRabin.h

Description: Miller-Rabin primality probabilistic test. Probability of failing one iteration is at most 1/4. 15 iterations should be enough for 50-bit numbers.
Time: 15 times the complexity of $a^b \mod c$.

"ModMulLL.h" 16 lines

```
bool prime(ull p) {
  if (p == 2) return true;
  if (p == 1 || p % 2 == 0) return false;
  ull s = p - 1;
  while (s % 2 == 0) s /= 2;
  rep(i,0,15) {
    ull a = rand() % (p - 1) + 1, tmp = s;
    ull mod = mod_pow(a, tmp, p);
    while (tmp != p - 1 && mod != 1 && mod != p - 1) {
      mod = mod_mul(mod, mod, p);
      tmp *= 2;
    }
    if (mod != p - 1 && tmp % 2 == 0) return false;
  }
  return true;
}
```

factor.h

Description: Pollard's rho algorithm. It is a probabilistic factorisation algorithm, whose expected time complexity is good. Before you start using it, run `init(bits)`, where bits is the length of the numbers you use.
Time: Expected running time should be good enough for 50-bit numbers.

"MillerRabin.h", "eratosthenes.h", "euclid.h" 37 lines

```
vector<ull> pr;
ull f(ull a, ull n, ull &has) {
  return (mod_mul(a, a, n) + has) % n;
}
vector<ull> factor(ull d) {
  vector<ull> res;
  for (size_t i = 0; i < pr.size() && pr[i]*pr[i] <= d; i++)
    if (d % pr[i] == 0) {
      while (d % pr[i] == 0) d /= pr[i];
      res.push_back(pr[i]);
    }
  //d is now a product of at most 2 primes.
  if (d > 1) {
    if (prime(d))
      res.push_back(d);
    else while (true) {
      ull has = rand() % 2321 + 47;
      ull x = 2, y = 2, c = 1;
```

```
      for (; c==1; c = gcd((y > x ? y - x : x - y), d)) {
        x = f(x, d, has);
        y = f(f(y, d, has), d, has);
      }
      if (c != d) {
        res.push_back(c); d /= c;
        if (d != c) res.push_back(d);
        break;
      }
    }
  }
  return res;
}
void init(int bits) {//how many bits do we use?
  vi p = eratosthenes_sieve(1 << ((bits + 2) / 3));
  vector<ull> pr(p.size());
  for (size_t i=0; i<pr.size(); i++)
    pr[i] = p[i];
}
```

5.4 Divisibility

euclid.h
Description: Finds the Greatest Common Divisor to the integers a and b. Euclid also finds two integers x and y, such that $ax + by = \gcd(a, b)$. If a and b are coprime, then x is the inverse of $a \pmod b$.
7 lines

```
ll gcd(ll a, ll b) { return __gcd(a, b); }

ll euclid(ll a, ll b, ll &x, ll &y) {
  if (b) { ll d = euclid(b, a % b, y, x);
    return y -= a/b * x, d; }
  return x = 1, y = 0, a;
}
```

Euclid.java
Description: Finds {x, y, d} s.t. ax + by = d = gcd(a, b).
11 lines

```
static BigInteger[] euclid(BigInteger a, BigInteger b) {
  BigInteger x = BigInteger.ONE, yy = x;
  BigInteger y = BigInteger.ZERO, xx = y;
  while (b.signum() != 0) {
    BigInteger q = a.divide(b), t = b;
    b = a.mod(b); a = t;
    t = xx; xx = x.subtract(q.multiply(xx)); x = t;
    t = yy; yy = y.subtract(q.multiply(yy)); y = t;
  }
  return new BigInteger[]{x, y, a};
}
```

5.4.1 Bézout's identity

For $a \neq$, $b \neq 0$, then $d = gcd(a, b)$ is the smallest positive integer for which there are integer solutions to

$$ax + by = d$$

If (x, y) is one solution, then all solutions are given by

$$\left(x + \frac{kb}{\gcd(a,b)}, y - \frac{ka}{\gcd(a,b)}\right), \quad k \in \mathbb{Z}$$

phiFunction.h
Description: *Euler's totient* or *Euler's phi* function is defined as $\phi(n) :=$ # of positive integers $\leq n$ that are coprime with n. The *cototient* is $n - \phi(n)$. $\phi(1) = 1$, p prime $\Rightarrow \phi(p^k) = (p-1)p^{k-1}$, m, n coprime $\Rightarrow \phi(mn) = \phi(m)\phi(n)$. If $n = p_1^{k_1} p_2^{k_2} ... p_r^{k_r}$ then $\phi(n) = (p_1 - 1)p_1^{k_1 - 1} ... (p_r - 1)p_r^{k_r - 1}$. $\phi(n) = n \cdot \prod_{p|n}(1 - 1/p)$.
$\sum_{d|n} \phi(d) = n$, $\sum_{1 \leq k \leq n, \gcd(k,n)=1} k = n\phi(n)/2, n > 1$
Euler's thm: a, n coprime $\Rightarrow a^{\phi(n)} \equiv 1 \pmod n$.
Fermat's little thm: p prime $\Rightarrow a^{p-1} \equiv 1 \pmod p$ $\forall a$.
10 lines

```
const int LIM = 5000000;
int phi[LIM];

void calculatePhi() {
  rep(i,0,LIM) phi[i] = i&1 ? i : i/2;
  for(int i = 3; i < LIM; i += 2)
    if(phi[i] == i)
      for(int j = i; j < LIM; j += i)
        (phi[j] /= i) *= i-1;
}
```

5.5 Chinese remainder theorem

chinese.h
Description: Chinese Remainder Theorem.
chinese(a, m, b, n) returns a number x, such that $x \equiv a \pmod m$ and $x \equiv b \pmod n$. For not coprime n, m, use chinese_common. Note that all numbers must be less than 2^{31} if you have Z = unsigned long long.
Time: $\log(m + n)$
"euclid.h" 13 lines

```
template <class Z> Z chinese(Z a, Z m, Z b, Z n) {
  Z x, y; euclid(m, n, x, y);
  Z ret = a * (y + m) % m * n + b * (x + n) % n * m;
  if (ret >= m * n) ret -= m * n;
  return ret;
}

template <class Z> Z chinese_common(Z a, Z m, Z b, Z n) {
  Z d = gcd(m, n);
  if (((b -= a) %= n) < 0) b += n;
  if (b % d) return -1; // No solution
  return d * chinese(Z(0), m/d, b/d, n/d) + a;
}
```

5.6 Pythagorean Triples

The Pythagorean triples are uniquely generated by

$$a = k \cdot (m^2 - n^2), \;\; b = k \cdot (2mn), \;\; c = k \cdot (m^2 + n^2),$$

with $m > n > 0$, $k > 0$, $m \perp n$, and either m or n even.

5.7 Primes

$p = 962592769$ is such that $2^{21} \mid p - 1$, which may be useful. For hashing use 970592641 (31-bit number), 31443539979727 (45-bit), 3006703054056749 (52-bit). There are 78498 primes less than 1 000 000.

Primitive roots exist modulo any prime power p^a, except for $p = 2, a > 2$, and there are $\phi(\phi(p^a))$ many. For $p = 2, a > 2$, the group $\mathbb{Z}_{2^a}^{\times}$ is instead isomorphic to $\mathbb{Z}_2 \times \mathbb{Z}_{2^{a-2}}$.

5.8 Estimates

$\sum_{d|n} d = O(n \log \log n)$.

The number of divisors of n is at most around 100 for $n < 5e4$, 500 for $n < 1e7$, 2000 for $n < 1e10$, 200 000 for $n < 1e19$.

Combinatorial (6)

6.1 The Twelvefold Way

Counts the # of functions $f : N \to K$, $|N| = n$, $|K| = k$. The elements in N and K can be distinguishable or indistinguishable, while f can be injective (one-to-one) of surjective (onto).

N	K	none	injective	surjective
dist	dist	k^n	$\frac{k!}{(k-n)!}$	$k!S(n,k)$
indist	dist	$\binom{n+k-1}{n}$	$\binom{k}{n}$	$\binom{n-1}{n-k}$
dist	indist	$\sum_{t=0}^{k} S(n,t)$	$[n \leq k]$	$S(n,k)$
indist	indist	$\sum_{t=1}^{k} p(n,t)$	$[n \leq k]$	$p(n,k)$

Here, $S(n, k)$ is the Stirling number of the second kind, and $p(n, k)$ is the partition number.

6.2 Permutations

6.2.1 Factorial

n	1	2	3	4	5	6	7	8	9	10
$n!$	1	2	6	24	120	720	5040	40320	362880	3628800

n	11	12	13	14	15	16	17
$n!$	4.0e7	4.8e8	6.2e9	8.7e10	1.3e12	2.1e13	3.6e14

n	20	25	30	40	50	100	150	171
$n!$	2e18	2e25	3e32	8e47	3e64	9e157	6e262	>DBL_MAX

intperm.h
Description: Permutations to/from integers. The bijection is order preserving.
Time: $\mathcal{O}(n^2)$
20 lines

```
int factorial[] = {1, 1, 2, 6, 24, 120, 720, 5040}; // etc.
template <class Z, class It>
void perm_to_int(Z& val, It begin, It end) {
  int x = 0, n = 0;
  for (It i = begin; i != end; ++i, ++n)
    if (*i < *begin) ++x;
  if (n > 2) perm_to_int<Z>(val, ++begin, end);
  else val = 0;
```