1. Understanding the Problem
   - First, you have to understand the problem
   - *What is the unknown? What are the data? What is the condition?*
   - Is it the conditon possible to satisfy, or sufficient to determine the unknown, or insufficient/redundant/contradictory?
   - Draw a figure. Introduce suitable notation.
2. Devising a Plan
   - *Do you know a related problem?*
   - *Look at the unknown!* Try to think of a familiar problem with the same/similar unknown.
   - Try related problems. More accessible, general, special, analogous problems? Solve a part of a problem.
   - *Try small cases.

1 checksum

Check sum of adler32.py "1526540401 0x5afd2871"

```
import sys
a, b, MOD = 1, 0, 65521
for s in sys.stdin:
  for x in ''.join(s.split()):
    a = (a + ord(x)) % MOD
    b = (b + a) % MOD
h = (b << 16) | a
print h, format(h, '#x')
```

```
setxkbmap -option ctrl:swapcaps
```

2 vimrc

Checksum "3215220533 0xbfa45f35"

```
set expandtab
set tabstop=4
set shiftwidth=4
set autoindent
set smartindent
set number
set ic
set incsearch
setlocal spell spelllang=en_us

au BufNewFile Main.java r .../Main.java
au BufNewFile *.cpp r .../cf.cpp
au BufNewFile gen.sh r .../gen.sh
au BufNewFile c r .../c
au BufNewFile j r .../j
```

3 c.sh

Checksum "1749835965 0x684c60bd"

```
rm -f myout*
rm -f "a.out"

g++ -std=c++11 -Wall -Wextra -Wshadow -O2 -Wconversion -
    fsanitize=address A.cpp

if [[ -f "./a.out" ]]; then

  for ((i=1; i<=1000; i++)); do
    if [[ -f "./in$i" ]]; then
      ./a.out < "in$i" > "myout$i"
          cat "myout$i"
    fi
  done

  echo difference

  for ((i=1; i<=1000; i++)); do
    if [[ -f "./in$i" ]]; then
      diff "myout$i" "out$i"
    fi
  done

fi
```

4 gen.sh

Checksum "1155022484 0x44d83e94"

```
rm -f myout*
rm -f "a.out"
```

```
g++ -std=c++11 -Wall -Wextra -Wshadow -O2 -Wconversion -
    fsanitize=address S.cpp

if [[ -f "./a.out" ]]; then

  for ((i=10; i<=100; i++)); do
    python gen.py > "in$i"
    ./a.out < "in$i" > "out$i"
  done

fi
```

5 cf.cpp

Checksum "1661913663 0x630eca3f"

```
#include <bits/stdc++.h>
using namespace std;
#define rep(i, n) for (int (i) = 0; (i) < (n); (i) ++)
#define rep1(i, n) for (int (i) = 1; (i) <= (n); (i) ++)
#define FOR(i, a, b) for (int (i) = (a); (i) <= (b); (i)++)
#define db(x) {cout << #x << " = " << (x) << endl;}
#define dba(a, x, y) {cout<<#a<<" :";FOR(i123,x,y)cout<<setw
    (4)<<(a)[i123];cout<<endl;}
#define clr(x) memset(x,0,sizeof(x));
#define mp make_pair
#define pb push_back
#define sz(x) int((x).size())
typedef long long ll;
typedef long double ld;
const int INF = INT_MAX;
const ll INFL = LLONG_MAX;
const ld pi = acos(-1);
// const int MOD = ;

int main()
{
    ios_base::sync_with_stdio(0); cout.precision(15); cout <<
        fixed; cout.tie(0); cin.tie(0);
}
```

6 Main.java

```
import java.util.*;
import java.io.*;
import java.math.*;
import java.text.*;
public class Main
{
    static FastScanner in = new FastScanner(System.in);
    static StringBuilder sb = new StringBuilder();
    public static void main(String[] args)
    {
        // Never sort array of primitive type
        System.out.print(sb.toString());
    }
    static BigInteger b(long n) { return BigInteger.valueOf(n)
        ;}
    static BigInteger b(String s) { return new BigInteger(s);
        }
}
class FastScanner
{
    BufferedReader reader;
    StringTokenizer tokenizer;
    FastScanner (InputStream inputStream)
    {
        reader = new BufferedReader(new InputStreamReader(
            inputStream));
    }
    String nextToken()
    {
        while (tokenizer == null || ! tokenizer.hasMoreTokens
            ())
        {
            try {
                tokenizer = new StringTokenizer(reader.
                    readLine());
            } catch (Exception e){}
        }
        return tokenizer.nextToken();
    }
    boolean hasNext()
    {
        if (tokenizer == null || ! tokenizer.hasMoreTokens())
        {
            try {
                tokenizer = new StringTokenizer(reader.
                    readLine());
```

```
            } catch (Exception e){ return false; }
        }
        return true;
    }
    int nextInt()
    {
        return Integer.parseInt(nextToken());
    }
}
```

7 max flow, Dinic

Runtime is $O(|E||V|^2)$, but it's much faster. For $|E| \leq 400,000, |V| \leq 20,000$, this ran in 5 seconds on POJ. $O(|E||V|^2) \times |C|$ ran in 8 seconds, "Surely You congest"($|E| \leq 50,000, |V| \leq 25,000, |C| \leq 10^3$)

```
struct edge { int to, cap, rev; };
struct MF{
  vector< vector<edge> > adj;
  vector<int> level, iter;
  MF(int V) : adj(V), level(V), iter(V) {}

  void add_edge(int from, int to, int cap) {
    adj[from].pb((edge){to,cap,sz(adj[to])});
    adj[to].pb((edge){from,0,sz(adj[from])-1});
  }
  void bfs(int s) {
    fill(level.begin(), level.end(), -1);
    queue<int> q;
    level[s] = 0;
    q.push(s);
    while (!q.empty()) {
      int v = q.front(); q.pop();
      rep(i,sz(adj[v])) {
        edge& e = adj[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 < sz(adj[v]); i++) {
      edge& e = adj[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;
          adj[e.to][e.rev].cap += d;
          return d;
        }
      }
    }
    return 0;
  }
  int max_flow(int s, int t) {
    int flow = 0;
    for(;;) {
      bfs(s);
      if (level[t] < 0) return flow;
      fill(iter.begin(), iter.end(), 0);
      for (int f; (f = dfs(s, t, INF)) > 0; flow += f);
    }
  }
};
```

8 Min-Cost-Max-Flow

Find the flow f of given $\mathbf{f}$ such that $\sum_e (f(e) \times e.cost)$ is minimized. $O(F(|E| \log |V| + |V|^2))$ Checksum "1333104336 0x4f758ed0" ** (Shouldn't int be long long when declaring INF?)

```
typedef ll cost_type;
const cost_type INF = (int) 1e15;
typedef pair<cost_type,int> P;
struct edge{
  int to, cap;
  cost_type cost;
  int rev;
};

struct MF {
  vector<vector<edge> > adj;
  vector<cost_type> h, dist;
  vector<int> prevv, preve;
  MF(int V) : adj(V), h(V), dist(V), prevv(V), preve(V) {}
  void add_edge(int from, int to, int cap, cost_type cost) {
    adj[from].pb((edge){to,cap,cost,sz(adj[to])});
    adj[to].pb((edge){from,0,-cost,sz(adj[from])-1});
```

```
  }
  cost_type min_cost_flow(int s, int t, int f) {
    cost_type res = 0;
    fill(h.begin(), h.end(), 0);
    while (f > 0) {
      priority_queue< P, vector<P>, greater<P> > q;
      fill(dist.begin(), dist.end(), INF);
      dist[s] = 0;
      q.push(P(0,s));
      while (!q.empty()) {
        P p = q.top();
        q.pop();
        int v = p.second;
        if (dist[v] < p.first) continue;
        rep(i,sz(adj[v])) {
          edge& e = adj[v][i];
          if (e.cap > 0 && dist[e.to] > dist[v] + e.cost + h[v
              ] - h[e.to]) {
            dist[e.to] = dist[v] + e.cost + h[v] - h[e.to];
            prevv[e.to] = v;
            preve[e.to] = i;
            q.push(P(dist[e.to], e.to));
          }
        }
      }
      if ((double) dist[t] + 1.0 > INF) return -1;
      rep(v,sz(h)) h[v] += dist[v];

      int d = f;
      for (int v = t; v != s; v = prevv[v]) {
        d = min(d, adj[prevv[v]][preve[v]].cap);
      }
      f -= d;
      res += d * h[t];
      for (int v = t; v != s; v = prevv[v]) {
        edge& e = adj[prevv[v]][preve[v]];
        e.cap -= d;
        adj[v][e.rev].cap += d;
      }
    }
    return res;
  }
};
```

9 LCA

Find the Lowest Common Ancestor in $O(log(n))$. Initializing is $O(n \times log(n))$.

```
const int MAX_V = 200100;
const int MAX_LOG_V = 25;
vector<int>  adj[MAX_V];
int root = 1;
int edgefrom[MAX_V];
int edgeto[MAX_V];

int parent[MAX_LOG_V][MAX_V];
int depth[MAX_V];

void dfs(int v, int p, int d) {
  parent[0][v] = p;
  depth[v] = d;
  rep(i,sz(adj[v]))
  if (adj[v][i] != p) dfs(adj[v][i], v, d + 1);
}

void init(int V) {
  dfs(root, -1, 0);
  for (int k = 0; k + 1 < MAX_LOG_V; k++) {
    for (int v = 1; v <= V; v++) {
      if (parent[k][v] < 0) parent[k + 1][v] = -1;
      else parent[k + 1][v] = parent[k][parent[k][v]];
    }
  }
}

int lca(int u, int v) {
  if (depth[u] > depth[v]) swap(u, v);
  for (int k = 0; k < MAX_LOG_V; k++) {
    if ((depth[v] - depth[u]) >> k & 1) {
      v = parent[k][v];
    }
  }
  if (u == v) return u;
  for (int k = MAX_LOG_V - 1; k >= 0; k--) {
    if (parent[k][u] != parent[k][v]) {
      u = parent[k][u];
      v = parent[k][v];
    }
  }
  return parent[0][u];
```

```
}
```

10 LCA RMQ

Find the Lowest Common Ancestor in $O(1)$. Initializing is $O(n \log n)$. For some reason, this seems slower? more test needed.

```
// add an empty constructor in SPT
const int MAX_V = 1001;
struct LCA {
  vector<int> adj[MAX_V], depth, vs;
  int root, id[MAX_V];
  SPT rmq;
  LCA(int inproot) : root(inproot){
    clr(adj); clr(id); vs.clear(); depth.clear();
  }
  void dfs(int u, int p, int d) {
    id[u] = sz(depth); vs.pb(u); depth.pb(d);
    rep(i,sz(adj[u])) {
      int v = adj[u][i];
      if (v != p) {
        dfs(v, u, d + 1);
        depth.pb(d); vs.pb(u);
      }
    }
  }
  void add_edge(int u, int v) {
    adj[u].pb(v); adj[v].pb(u);
  }
  void init() { dfs(root, -1, 0); rmq = SPT(depth); }
  int lca (int u, int v) {
    return vs[rmq.minPos(min(id[u],id[v]),max(id[u],id[v]))];
  }
};
```

11 SCC

SCC scc(|V|); Must use index from 1. $cmp[u] < cmp[v]$ implies no path from v to u. **scc()** returns # of SCC's. checksum = "1757526187 0x68c1b8ab"

```
const int MAX_V = 100;
struct SCC {
  int V, used[MAX_V], cmp[MAX_V];
  vector<int> G[MAX_V], rG[MAX_V], vs;
  SCC(int n) : V(n) {}
  void add_edge(int from, int to) {
    G[from].pb(to);
    rG[to].pb(from);
  }
  void dfs(int v) {
    used[v] = true;
    rep(i,sz(G[v])) if (!used[G[v][i]]) dfs(G[v][i]);
    vs.pb(v);
  }
  void rdfs(int v, int k) {
    used[v] = true; cmp[v] = k;
    rep(i,sz(rG[v])) if (!used[rG[v][i]]) rdfs(rG[v][i], k);
  }
  int scc() {
    memset(used, 0, sizeof(used));
    vs.clear();
    rep1(v,V) if (!used[v]) dfs(v);
    memset(used, 0, sizeof(used));
    int k = 0;
    for (int i = sz(vs) - 1; i >= 0; i--)
      if (!used[vs[i]]) rdfs(vs[i], k++);
    return k;
  }
};
```

12 Range Tree

Given n points on $X-Y$ plane, find a number of points within a rectangle in $O(log(n))$. -std=c++11

```
typedef int Ptype;
typedef pair<Ptype, Ptype> Pt;
const Pt Pinf = Pt(numeric_limits<Ptype>::max(),
    numeric_limits<Ptype>::max());
class RangeTree {
public:
    int n;
    vector<Pt> xsorted;

    vector<vector<Pt> > dat;
    vector<vector<pair<int,int> > > bsearch_speedup;
    RangeTree(vector<Pt>& a) {
        n = 1;
        while (n < sz(a)) n <<= 1;
```

```
        dat.resize(2 * n - 1);

        //sort by first
        sort(a.begin(), a.end());
        xsorted = a;
        xsorted.resize(n, Pinf);
        bsearch_speedup.resize(n);

        rep(i, n) {
            int k = n - 1 + i;
            if (i < sz(a)) dat[k].push_back(a[i]);
            else dat[k].push_back(Pinf);
        }
        for (int i = n - 2; i >= 0; i--) {
            auto& cur_dat = dat[i], &lchild = dat[2 * i + 1],
                rchild = dat[2 * i + 2];
            cur_dat.resize(sz(lchild) + sz(rchild));
            //sort by second
            merge(lchild.begin(), lchild.end(), rchild.begin()
                , rchild.end(),
                cur_dat.begin(),
                [](const Pt& l, const Pt&  r) { return l.
                    second != r.second ? l.second < r.second :
                    l.first < r.first; }
            );
            //binary_search speed up with fractional cascading
            auto& bs = bsearch_speedup[i];
            bs.resize(sz(cur_dat));
            int a1 = 0, a2 = 0;
            rep(k, sz(cur_dat)) {
                while (a1 < sz(lchild) && lchild[a1].second <
                    cur_dat[k].second) a1++;
                bs[k].first = a1;
                while (a2 < sz(rchild) && rchild[a2].second <
                    cur_dat[k].second) a2++;
                bs[k].second = a2;
            }
            bs.emplace_back(sz(lchild), sz(rchild));
        }
    }
    // The number of verticies in [lx,rx) * [ly,ry) (O(log n))
    int query(Ptype lx, Ptype rx, Ptype ly, Ptype ry) {
#define CMP [](const Pt& l, const Ptype val) { return l.second
    < val; }
        int ly_index = lower_bound(dat[0].begin(), dat[0].end
            (), ly, CMP) - dat[0].begin();
        int ry_index = lower_bound(dat[0].begin(), dat[0].end
            (), ry, CMP) - dat[0].begin();
#undef  CMP
        return query(lx, rx, ly_index, ry_index, 0, 0, n);
    }
    int query(Ptype lx, Ptype rx, int ly_index, int ry_index,
        int k, int a, int b) {
        if (rx <= xsorted[a].first || xsorted[b - 1].first <
            lx) return 0;
        if (lx <= xsorted[a].first && xsorted[b - 1].first <
            rx) {
            return ry_index - ly_index;
        }

        int nly_idx_f, nly_idx_s, nry_idx_f, nry_idx_s;
        tie(nly_idx_f, nly_idx_s) = bsearch_speedup[k][
            ly_index];
        tie(nry_idx_f, nry_idx_s) = bsearch_speedup[k][
            ry_index];
        int lval = query(lx, rx, nly_idx_f, nry_idx_f, 2 * k +
            1, a, (a + b) / 2);
        int rval = query(lx, rx, nly_idx_s, nry_idx_s, 2 * k +
            2, (a + b) / 2, b);
        return lval + rval;
    }
};

void range_tree_test() {
  vector<Pt> vpt = { Pt(0, 0),
    Pt(1, 0), Pt(2, 0), Pt(3, 0), Pt(0, 10),
    Pt(0, 20), Pt(0, 30), Pt(5, 5), Pt(-3, -3)
  };

  RangeTree rt(vpt);
  assert(rt.query(0, 1, 0, 1) == 1);
  assert(rt.query(0, 4, 0, 4) == 4);
  assert(rt.query(-10, 1, -10, 1) == 2);
  assert(rt.query(-100, 100, -100, 100) == 9);
}
```

13 bipartite matching

Time complexity: $O(|V||E|)$(upper bound)
$match[i]$ is the node matched with i, and -1 if not matched Checksum : "281848587

0x10ccab0b"

```
struct BipartiteMatching {
  vector<vector<int> > adj;
  vector<int> match, used;
  BipartiteMatching(int V) : adj(V), match(V), used(V) {}
  void add_edge(int u, int v) {
    adj[u].pb(v); adj[v].pb(u);
  }
  int dfs(int v) {
    used[v] = true;
    rep(i,sz(adj[v])) {
      int u = adj[v][i], w = match[u];
      if (w < 0 || (!used[w] && dfs(w))) {
        match[v] = u;
        match[u] = v;
        return 1;
      }
    }
    return 0;
  }
  int bipartite_matching() {
    int res = 0;
    fill(match.begin(), match.end(), -1);
    rep(v,sz(match)) {
      if (match[v] < 0) {
        fill(used.begin(), used.end(), 0);
        if (dfs(v)) res++;
      }
    }
    return res;
  }
};
```

14 Cut Point

A *cut point* is a vertex whose removal makes the graph disconnected. Input: A connected undirected graph, a root (any node in G) Output: a list of all cut points Runtime: $O(|V|+|E|)$ Note: An articulation point exists = the global min vertex cut is 1 No re-initialization required when called multiple times.(run resets whatever that needs to be reset) Checksum "1353057223 0x50a603c7"

```
const int MAXN = 1010;
struct CutPoint {
  vector<int> g[MAXN], cut_points;
  int timer, used[MAXN], tin[MAXN], fup[MAXN];
  CutPoint() {
    clr(g); clr(used); clr(tin); clr(fup); timer = 0;
  }

  void add_edge(int u, int v) {
    g[u].pb(v); g[v].pb(u);
  }

  void dfs (int v, int p = -1) {
    used[v] = true;
    tin[v] = fup[v] = timer++;
    int children = 0;
    rep(i,sz(g[v])) {
      int to = g[v][i];
      if (to == p)  continue;
      if (used[to]) {
        fup[v] = min (fup[v], tin[to]);
      } else {
        dfs (to, v);
        fup[v] = min (fup[v], fup[to]);
        if (fup[to] >= tin[v] && p != -1) {
          cut_points.pb(v);
        }
        ++children;
      }
    }
    if (p == -1 && children > 1) {
      cut_points.pb(v);
    }
  }
  vector<int> run(int root) {
    clr(used); cut_points.clear();
    dfs(root);
    sort(cut_points.begin(), cut_points.end());
    cut_points.erase(unique(cut_points.begin(), cut_points.end
        ()), cut_points.end());
    return cut_points;
  }
};
```

15 Minimum Cut

Given undirected graph, returns (min cut value, nodes in half of min cut) 0-based index. Nodes are numbered $0, \cdots, n-1$. Run time: $O(|V|^3)$. Memory $O(|V|^2)$. Check sum "1780091446 0x6a1a0a36"

```
struct MinCut {
  vector<vector<int> > weights;
  MinCut(int n) : weights(n, vector<int>(n)) {}
  void add_edge(int u, int v, int w) { weights[u][v] += w;
      weights[v][u] += w; }
  pair<int, vector<int> > GetMinCut() {
    int N = sz(weights), best_weight = -1, prev, last;
    vector<int> used(N), cut, best_cut, w, added;
    for (int phase = N-1; phase >= 0; phase--) {
      w = weights[0]; added = used; last = 0;
      rep(i,phase) {
        prev = last; last = -1;
        rep1(j,N-1) if (!added[j] && (last == -1 || w[j] > w[
            last])) last = j;
        if (i == phase-1) {
          rep(j,N) weights[prev][j] += weights[last][j];
          rep(j,N) weights[j][prev] = weights[prev][j];
          used[last] = true;
          cut.push_back(last);
          if (best_weight == -1 || w[last] < best_weight) {
            best_cut = cut;
            best_weight = w[last];
          }
        } else {
          rep(j,N) w[j] += weights[last][j];
          added[last] = true;
        }
      }
    }
    return mp(best_weight, best_cut);
  }
};
```

16 Choose

NOT TESTED YET

```
const int MAX_N = 200100;
struct Choose {
  int fac[MAX_N + 1], inv[MAX_N + 1];
  int inverse(int a) {
    return a==1?1:(int)((MOD-MOD/a)*1ll*inverse(MOD%a)%MOD);
  }
  void init() {
    fac[0] = inv[0] = 1;
    rep1(i,MAX_N) fac[i] = (int) ((fac[i-1]*1ll*i)%MOD);
    rep1(i,MAX_N) inv[i] = inverse(fac[i]);
  }
  int choose(int a, int b) {
    return (int) ((fac[a]*1ll*inv[b]%MOD)*inv[a-b]%MOD);
  }
};
```

17 inverse

$extgcd(a,b,x,y)$ returns $gcd(a,b)$ and $a*x+y*b = gcd(a,b)$ $inv(a,M)$ return $0 \le b < M$ s.t. $b*a \equiv 1 (mod M)$ Take care of the case $M = 1$. Checksum "2063221188 0x7afa41c4"

```
typedef ll Int;
Int extgcd(Int a, Int b, Int& x, Int& y) {
  Int d = a;
  if (b != 0) {
    d = extgcd(b, a % b, y, x);
    y -= (a / b) * x;
  } else {
    x = 1; y = 0;
  }
  return d;
}
Int inv(Int a, Int mod) {
  Int x, y, g;
  g = extgcd(a, mod, x, y);
  if (x < 0) x += mod;
  return x;
}
```

17.1 modular equation

Checksum: "2319074103 0x8a3a4337" Solve $ax \equiv b \pmod m$ All a, b, m have to be positive. -1 if no solution exists. **soln** is the "minimum" solution. Put any k to get other solution.

```
Int solve(Int a, Int b, Int m) {
  Int x, y, g = extgcd(a, m, x, y);
```

```
  if (g == 1) {
    return (Int) ((inv(a, m) * (__int128) b) % m);
  }
  if (b % g != 0) {
    return -1;
  }
  __int128 soln = (inv(a / g, m / g) * (__int128) (b / g)) % (
      m / g);
  __int128 k = 0;
  return (Int) (soln + k * (m / g)) % m;
}
```

18 FFT

Multiply two polynomials in $O(n \times log(n))$.

```
typedef complex<double> base;

 void fft (vector<base> & a, bool invert) {
    int n = (int) a.size();

      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);
          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 *= wlen;
          }
        }
      }
      if (invert)
        for (int i=0; i<n; ++i)
          a[i] /= n;
 }

void multiply (const vector<ll> & a, const vector<ll> & b,
    vector<ll> & res) {
  vector<base> fa (a.begin(), a.end());
  vector<base> fb (b.begin(), b.end());
  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] *= fb[i];
  fft (fa, true);

  res.resize (n);
  for (size_t i=0; i<n; ++i)
    res[i] = ll (fa[i].real() + 0.5);
}
```

19 Geometry

****TEST NEEDED ON EACH FUNCTION AND CHECKSUM!!!!**** ****MARK THOSE THAT ARE TESTED!!!!!***** Declaration of PT. Always necessary. For the sake of simplicity, just write the whole thing. Checksum "3078928143 0xb784b70f"

```
double INF = 1e100;
double EPS = 1e-12;

struct PT {
  double x, y;
  PT() {}
  PT(double x, double y) : x(x), y(y) {}
  PT(const PT &p) : x(p.x), y(p.y)    {}
  PT operator + (const PT &p)  const { return PT(x+p.x, y+p.y)
      ; }
  PT operator - (const PT &p)  const { return PT(x-p.x, y-p.y)
      ; }
  PT operator * (double c)     const { return PT(x*c,   y*c   )
      ; }
  PT operator / (double c)     const { return PT(x/c,   y/c   )
      ; }
```

```
};

double dot(PT p, PT q)     { return p.x*q.x+p.y*q.y; }
double dist2(PT p, PT q)   { return dot(p-q,p-q); }
double cross(PT p, PT q)   { return p.x*q.y-p.y*q.x; }
ostream &operator<<(ostream &os, const PT &p) {
  os << "(" << p.x << "," << p.y << ")";
  return os;
}
```

Rotate a point CCW or CW around the origin. Checksum "1937257670 0x737834c6"

```
PT RotateCCW90(PT p)   { return PT(-p.y,p.x); }
PT RotateCW90(PT p)    { return PT(p.y,-p.x); }
PT RotateCCW(PT p, double t) {
  return PT(p.x*cos(t)-p.y*sin(t), p.x*sin(t)+p.y*cos(t));
}
```

Project point c onto line through a and b assuming $a \neq b$. Checksum "2839025675 0xa938180b"

```
PT ProjectPointLine(PT a, PT b, PT c) {
  return a + (b-a)*dot(c-a, b-a)/dot(b-a, b-a);
}
```

Project point c onto line segment through a and b. Checksum "4121440480 0xf5a830e0"

```
PT ProjectPointSegment(PT a, PT b, PT c) {
  double r = dot(b-a,b-a);
  if (fabs(r) < EPS) return a;
  r = dot(c-a, b-a)/r;
  if (r < 0) return a;
  if (r > 1) return b;
  return a + (b-a)*r;
}
```

Compute distance from c to segment between a and b. Checksum "3888980086 0xe7cd2076"

```
double DistancePointSegment(PT a, PT b, PT c) {
  return sqrt(dist2(c, ProjectPointSegment(a, b, c)));
}
```

Compute distance between point (x, y, z) and plane $ax + by + cz = d$. Checksum "849489516 0x32a22e6c"

```
double DistancePointPlane(double x, double y, double z,
                          double a, double b, double c, double d) {
  return fabs(a*x+b*y+c*z-d)/sqrt(a*a+b*b+c*c);
}
```

Determine if lines from a to b and c to d are parallel or collinear. Checksum "913839923 0x36781733"

```
bool LinesParallel(PT a, PT b, PT c, PT d) {
  return fabs(cross(b-a, c-d)) < EPS;
}
```

Checksum "2546804568 0x97cd2758"

```
bool LinesCollinear(PT a, PT b, PT c, PT d) {
  return LinesParallel(a, b, c, d)
      && fabs(cross(a-b, a-c)) < EPS
      && fabs(cross(c-d, c-a)) < EPS;
}
```

Determine if line segment from a to b intersects with line segment from c to d. Checksum "2456579558 0x926c6de6"

```
bool SegmentsIntersect(PT a, PT b, PT c, PT d) {
  if (LinesCollinear(a, b, c, d)) {
    if (dist2(a, c) < EPS || dist2(a, d) < EPS ||
      dist2(b, c) < EPS || dist2(b, d) < EPS) return true;
    if (dot(c-a, c-b) > 0 && dot(d-a, d-b) > 0 && dot(c-b, d-b
      ) > 0)
      return false;
    return true;
  }
  if (cross(d-a, b-a) * cross(c-a, b-a) > 0) return false;
  if (cross(a-c, d-c) * cross(b-c, d-c) > 0) return false;
  return true;
}
```

Compute intersection of line passing through a and b with line passing through c and d, assuming that unique intersection exists; for segment intersection, check if segments intersect first. Checksum "951724821 0x38ba2b15"

```
PT ComputeLineIntersection(PT a, PT b, PT c, PT d) {
  b=b-a; d=c-d; c=c-a;
  assert(dot(b, b) > EPS && dot(d, d) > EPS);
  return a + b*cross(c, d)/cross(b, d);
}
```

Compute center of circle given three points. Checksum "3497404977 0xd0762a31"

```
PT ComputeCircleCenter(PT a, PT b, PT c) {
```

```
  b=(a+b)/2; c=(a+c)/2;
  return ComputeLineIntersection(b, b+RotateCW90(a-b), c, c+
      RotateCW90(a-c));
}
```

Determine if point is in a possibly non-convex polygon; returns 1 for strictly interior points, 0 for strictly exterior points, and 0 or 1 for the remaining points. Note that it is possible to convert this into an *exact* test using integer arithmetic by taking care of the division appropriately (making sure to deal with signs properly) and then by writing exact tests for checking point on polygon boundary. changed for loop. not tested yet. Checksum "2766030140 0xa4de453c"

```
bool PointInPolygon(const vector<PT> &p, PT q) {
  bool c = 0;
  rep(i,sz(p)) {
    int j = (i+1)%sz(p);
    if ((p[i].y <= q.y && q.y < p[j].y ||
      p[j].y <= q.y && q.y < p[i].y) &&
      q.x < p[i].x + (p[j].x - p[i].x) * (q.y - p[i].y) / (p[j
          ].y - p[i].y))
      c = !c;
  }
  return c;
}
```

Determine if point is on the boundary of a polygon. changed for loop. not tested yet. Checksum "1777676600 0x69f53138"

```
bool PointOnPolygon(const vector<PT> &p, PT q) {
  rep(i,sz(p))
    if (dist2(ProjectPointSegment(p[i], p[(i+1)%sz(p)], q), q)
        < EPS)
      return true;
    return false;
}
```

Compute intersection of line through points a and b with circle centered at c with radius $r > 0$. Checksum "344938771 0x148f5913"

```
vector<PT> CircleLineIntersection(PT a, PT b, PT c, double r)
    {
  vector<PT> ret;
  b = b-a;
  a = a-c;
  double A = dot(b, b);
  double B = dot(a, b);
  double C = dot(a, a) - r*r;
  double D = B*B - A*C;
  if (D < -EPS) return ret;
  ret.push_back(c+a+b*(-B+sqrt(D+EPS))/A);
  if (D > EPS)
    ret.push_back(c+a+b*(-B-sqrt(D))/A);
  return ret;
}
```

Compute intersection of circle centered at a with radius r with circle centered at b with radius R. Checksum "3623707949 0xd7fd652d"

```
vector<PT> CircleCircleIntersection(PT a, PT b, double r,
    double R) {
  vector<PT> ret;
  double d = sqrt(dist2(a, b));
  if (d > r+R || d+min(r, R) < max(r, R)) return ret;
  double x = (d*d-R*R+r*r)/(2*d);
  double y = sqrt(r*r-x*x);
  PT v = (b-a)/d;
  ret.push_back(a+v*x + RotateCCW90(v)*y);
  if (y > 0)
    ret.push_back(a+v*x - RotateCCW90(v)*y);
  return ret;
}
```

This code computes the area or centroid of a (possibly nonconvex) polygon, assuming that the coordinates are listed in a clockwise or counterclockwise fashion. Note that the centroid is often known as the "center of gravity" or "center of mass". Checksum "3145551487 0xbb7d4e7f"

```
double ComputeSignedArea(const vector<PT> &p) {
  double area = 0;
  for(int i = 0; i < sz(p); i++) {
    int j = (i+1) % sz(p);
    area += p[i].x*p[j].y - p[j].x*p[i].y;
  }
  return area / 2.0;
}
double ComputeArea(const vector<PT> &p) {
  return fabs(ComputeSignedArea(p));
}
```

Checksum "3613080572 0xd75b3bfc"

```
PT ComputeCentroid(const vector<PT> &p) {
  PT c(0,0);
  double scale = 6.0 * ComputeSignedArea(p);
  rep(i,sz(p)) {
    int j = (i+1) % sz(p);
```

```
    c = c + (p[i]+p[j])*(p[i].x*p[j].y - p[j].x*p[i].y);
  }
  return c / scale;
}
```

Tests whether a given polygon (in CW or CCW order) is simple. Checksum "4094118100 0xf40748d4" changed size. not tested yet.

```
bool IsSimple(const vector<PT> &p) {
  for (int i = 0; i < sz(p); i++) {
    for (int k = i+1; k < sz(p); k++) {
      int j = (i+1) % sz(p);
      int l = (k+1) % sz(p);
      if (i == l || j == k) continue;
      if (SegmentsIntersect(p[i], p[j], p[k], p[l]))
        return false;
    }
  }
  return true;
}
```

20 degree

have not tested much also pay close attention to the precision error

```
// returns [0, 360)
double EPS = 1e-9;
struct point { int x, y; };
double mydeg(point p)
{
  double x = p.x;
  double y = p.y;
  double a = asin(y / sqrt(x * x + y * y)) * 180 / pi;
  double ret = 0;
  if (x >= 0 && y >= 0) ret = a;
  if (x <= 0 && y >= 0) ret = 180 - a;
  if (x >= 0 && y <= 0) ret = 360 + a;
  if (x <= 0 && y <= 0) ret = 180 + abs(a);
  while (ret >= 360) ret -= 360;
  return ret;
}
```

21 Convex hull

$O(n \log n)$ From an unordered vector of points to a vector of points in the convex hull, counterclockwise, starting with bottommost/leftmost points. A convex hull of a set of grid points s.t. $|x|, |y| < |M|$ has $O(\sqrt{M})$ points. Checksum "2298843660 0x8905920c"

```
double area2(PT a, PT b, PT c) { return cross(a,b) + cross(b,c
    ) + cross(c,a); }
int mycomp(PT p, PT q) { return mp(p.y,p.x) < mp(q.y,q.x); }
void ConvexHull(vector<PT> &pts) {
  sort(pts.begin(), pts.end(), mycomp);
  vector<PT> up, dn;
  rep(i,sz(pts)) {
    while (sz(up) > 1 && area2(up[sz(up)-2], up.back(), pts[i
        ]) >= 0) up.pop_back();
    while (sz(dn) > 1 && area2(dn[sz(dn)-2], dn.back(), pts[i
        ]) <= 0) dn.pop_back();
    up.pb(pts[i]); dn.pb(pts[i]);
  }
  pts = dn;
  rep1(i,sz(up)-2) pts.pb(up[sz(up)-i-1]);
}
```

22 Rotating Calipers

Input **must** be a result of ConvexHull. Returns all antipodal pairs by their indexes. **Don't forget to consider the case of only 1 point**. Checksum : "97420220 0x5ce83bc" tested only once.

```
vector< pair<int,int> > calipers(const vector<PT>& v) {
  vector<pair<int,int> > ret;
  if (sz(v) == 2)  {
    ret.pb(mp(0, 1));
  } else if (sz(v) >= 3) {
    int i = 0, j = 0, ii, jj, si, sj;
    rep(k,sz(v)) {
      if (mp(v[k].x, v[k].y) >= mp(v[i].x, v[i].y)) i = k;
      if (mp(v[k].x, v[k].y) <  mp(v[j].x, v[j].y)) j = k;
    }
    for (si = i, sj = j; i != sj || j != si;) {
      ret.pb(mp(i, j));
      ii = (i + 1) % sz(v); jj = (j + 1) % sz(v);
      if (cross(v[ii] - v[i], v[jj] - v[j]) < 0)  {
        i = ii;
      } else {
        j = jj;
```

```
      }
    }
  }
  return ret;
}
```

23 trie

prefix tree TODO: add more comments and explanation

```
int nextNode[200001][26];
int z = 1;
int add(string s)
{
  int node = 1;
  for (int i = 0; i < sz(s); i++)
  {
    int which = nextNode[node][s[i]-'a'];
    if (which == 0)
    {
      z++;
      nextNode[node][s[i]-'a'] = z;
      which = z;
    }
    node = which;
  }
  return 0;
}
```

24 Z-algorithm

Given a string s of length n, $k = z[i]$ is the largest number such that $s[0]...s[k-1]$ is the same as $z[i]..z[i+k-1]$. In other words, $z[i]$ is the longest substring that starts at i which is also a prefix of s. $O(n)$. Checksum "826949247 0x314a3e7f"

```
vector<int> make_z(string s)
{
  int N = sz(s);
  vector<int> z(N);
  for (int i = 1, l = 0, r = 0, k; i < N; i++) {
    if (i <= r) z[i] = min (r - i + 1, z[i - l]);
    while ((k = i + z[i]) < N && s[z[i]] == s[k]) z[i]++;
    k--;
    if (k > r) l = i, r = k;
  }
  return z;
}
```

25 Suffix Array

Everything here is 0-based index. $S[sa[i]\cdots]$ is the i-th smallest suffix. The smallest (0-th) is an empty suffix. Runtime: $O(n \log^2(n))$. Checksum "1802937852 0x6b76a5fc"

```
struct SuffixArray {
  int n, k;
  vector<int> rank, tmp, sa;
  pair<int,int> f(int i) {
    return mp(rank[i], i + k <= n ? rank[i + k] : -1);
  }
  vector<int> get_sa(const string& s) {
    n = sz(s);
    rank.resize(n + 1); tmp.resize(n + 1); sa.resize(n + 1);
    rep(i,n + 1) sa[i] = i, rank[i] = i < n ? s[i] : -1;
    vector< pair<pair<int,int>,int> > v;
    for (k = 1; k <= n; k *= 2) {
      v.clear();
      rep(i,n + 1) v.pb(mp(f(sa[i]),sa[i]));
      sort(v.begin(), v.end());
      rep(i,n + 1) sa[i] = v[i].second;
      tmp[sa[0]] = 0;
      rep1(i,n) tmp[sa[i]] = tmp[sa[i - 1]] + (f(sa[i - 1]) <
          f(sa[i]) ? 1 : 0);
      rank = tmp;
    }
    return sa;
  }
};
```

26 LCP Array

Runtime: $O(n)$. $lcp[i]$ = the length of the longest common prefix of $S[sa[i]\cdots]$ and $S[sa[i+1]\cdots]$. Construct $rank$ array s.t. $rank[sa[i]] = i$. Let $rank[i] < rank[j]$. The length of LCP of suffixes at i, j are $\min_{rank[i] \le k \le rank[j]-1} lcp[k]$. Checksum: "3448589706 0xcd8d4d8a"

```
vector<int> LCP(const string& s, const vector<int>& sa) {
  int n = sz(s), h = 0;
```

```
  vector<int> rank(n + 1), lcp(n);
  rep(i,n + 1) rank[sa[i]] = i;
  rep(i,n) {
    int j = sa[rank[i] - 1];
    if (h > 0) h--;
    for (; max(i, j) + h < n && s[j + h] == s[i + h]; h++);
    lcp[rank[i] - 1] = h;
  }
  return lcp;
}
```

27 Segment Tree

Initialization.

```
seg_tree_lazy<node, update> st(N, zero_node, zero_update);
vector<node> leaves(N, {zero_value, 1});
st.set_leaves(leaves);
```

Update type = 0 for add, 1 for reset. The default code supports the following operations:

1. Add amount V to the values in range $[L, R]$.
2. Reset the values in range $[L, R]$ to value V.
3. Find the sum of the values in range $[L, R]$.

It takes a type T for vertex values, and a type U for update operations. Type T should have an operator + specifying how to combine vertices. Type U should have an operator () specifying how to apply updates to vertices, and an operator + for combining two updates. Checksum "3695777235 0xdc4915d3"

```
struct node {
  int sum, width;
  node operator+(const node &n) const {
    return { sum + n.sum, width + n.width };
  }
};
struct update {
  bool type; int value;
  node operator()(const node &n) const {
    if (type) return { n.width * value, n.width };
    else return { n.sum + n.width * value, n.width };
  }
  update operator+(const update &u) const {
    if (u.type) return u;
    return { type, value + u.value };
  }
};
template<typename T, typename U> struct seg_tree_lazy {
  int S, H;
  T zero;
  vector<T> value;
  U noop;
  vector<bool> dirty;
  vector<U> prop;
  seg_tree_lazy<T, U>(int _S, T _zero = T(), U _noop = U()) {
    zero = _zero, noop = _noop;
    for (S = 1, H = 1; S < _S; ) S *= 2, H++;
    value.resize(2*S, zero);
    dirty.resize(2*S, false);
    prop.resize(2*S, noop);
  }
  void set_leaves(vector<T> &leaves) {
    copy(leaves.begin(), leaves.end(), value.begin() + S);
    for (int i = S - 1; i > 0; i--)
      value[i] = value[2 * i] + value[2 * i + 1];
  }
  void apply(int i, U &update) {
    value[i] = update(value[i]);
    if(i < S) {
      prop[i] = prop[i] + update;
      dirty[i] = true;
    }
  }
  void rebuild(int i) {
    for (int l = i/2; l; l /= 2) {
      T combined = value[2*l] + value[2*l+1];
      value[l] = prop[l](combined);
    }
  }
  void propagate(int i) {
    for (int h = H; h > 0; h--) {
      int l = i >> h;
      if (dirty[l]) {
        apply(2*l, prop[l]);
        apply(2*l+1, prop[l]);
        prop[l] = noop;
        dirty[l] = false;
      }
    }
  }
```

```
  void upd(int i, int j, U update) {
    i += S, j += S;
    propagate(i), propagate(j);
    for (int l = i, r = j; l <= r; l /= 2, r /= 2) {
      if((l&1) == 1) apply(l++, update);
      if((r&1) == 0) apply(r--, update);
    }
    rebuild(i), rebuild(j);
  }
  T query(int i, int j){
    i += S, j += S;
    propagate(i), propagate(j);
    T res_left = zero, res_right = zero;
    for(; i <= j; i /= 2, j /= 2){
      if((i&1) == 1) res_left = res_left + value[i++];
      if((j&1) == 0) res_right = value[j--] + res_right;
    }
    return res_left + res_right;
  }
};
```

28 BIT

The index is from 1 to n. Each operations is $O(log(n))$. checksum "2735027546 0xa305355a" Extensible to 2-dimension in a very obvious way.

```
struct BIT{
  int N;
  vector<int> v;
  BIT(int n) : N(n), v(n + 1) {}
  int sum(int i) {
    int s = 0;
    for (;i > 0; s += v[i], i -= i & -i);
    return s;
  }
  void add(int i, int x) {
    for (; i <= N; v[i] += x, i += i & -i);
  }
};
```

29 Sparse table

SPT rmq(a); where **a** is a vector. Index from 0. **minPos(i,j)** returns $k \in [i, j]$ such that $\forall x \in [i, j] a_k \leq a_x$.

```
struct SPT {
  vector<int> logTable, a;
  vector< vector<int> > rmq;
  SPT () {}
  SPT(vector<int> inpv) {
    this->a = inpv;
    int n = sz(a);
    logTable.resize(n + 1);
    for (int i = 2; i <= n; i++)
      logTable[i] = logTable[i >> 1] + 1;
    rmq.resize(logTable[n] + 1, vector<int>(n));
    rep(i,n) rmq[0][i] = i;
    for (int k = 1; (1 << k) < n; k++) {
      rep(i, n - (1 << k) + 1) {
        int x = rmq[k - 1][i];
        int y = rmq[k - 1][i + (1 << (k - 1))];
        rmq[k][i] = a[x] <= a[y] ? x : y;
      }
    }
  }
  int minPos(int i, int j) {
    int k = logTable[j - i];
    int x = rmq[k][i], y = rmq[k][j - (1 << k) + 1];
    return a[x] <= a[y] ? x : y;
  }
};
```

30 meet in the middle

```
vector< pair<key, val> > v;
void gen1(index) {
  if (index == N / 2) {
    v.add(new pair)
  }
  else {
    gen1(index + 1)
  }
}

void gen2(index) {
  if (index == N - N / 2){
    for (i = lower_bound(v.begin(), v.end(), mp(key, minval));
        i < sz(v) && v[i].first == key; i++) {
```

```
      answer.pb(v[i].second);
    }
  }
}
```

31 SQRT decomposition

Use index from 0

```
const int MAX_N = 100100;
const int bsize = 1000;
int A[MAX_N + 1];
vector<int> B[MAX_N / bsize + 1];

// kind of tested
// check the interval [from, to)
void f(int from, int to)
{
  for (int ub = min((from/bsize+1)*bsize,to);
       from < ub; from++)
  {
    // do something at from
  }
  for (int lb = max((to/bsize)*bsize,from);
       lb < to;)
  {
    to--;
    // do something at to
  }
  for (int fromb = from/bsize, tob = to/bsize;
    fromb < tob; fromb++)
  {
    // do something at fromb
  }
}
```

32 LR Flow

More specifically, for each edge $e = (u, v)$, $0 \le b(e) \le c(e)$ is defined and we want to find a flow f such that $b(e) \le f(e) \le c(e)$.

Finding a such flow (check its existence as well)

```
Make 2 more nodes s2, t2
bsum = 0
for each edge e = (u, v, b, c)
  bsum += b
  add_edge(s2, v, b)
  add_edge(u, t2, b)
  add_edge(u, v, c - b)
end
add_edge(t, s, INF)

flow = max_flow(s2, t2)

if flow != bsum: return "NOT SUCH FLOW"

for each edge e
  flow_of_e
    = e.c - (rem cap of a corresp. edge of e)
end
```

Find the max flow satisfying the lower bound (assuming we know the existence of such flow) **NOTE:** Though this is out of the book, not too certain if this works.

```
Make 2 more nodes s2, t2
bsum = 0
for each edge e = (u, v, b, c)
  bsum += b
  add_edge(s2, v, b)
  add_edge(u, t2, b)
  add_edge(u, v, c - b)
end
add_edge(s2, s, INF)
add_edge(t, t2, INF)
flow = max_flow(s2, t2) - bsum
return flow
```

33 Calendar

0 : Sunday, 1 : Monday, $\cdots$ Checksum: "2688897997 0xa04553cd"

```
int toJulian(int y, int m, int d) {
  if (m < 3) y--, m += 12;
  return 365*y + y/4 - y/100 + y/400 + (153*m - 457)/5 + d +
      1721119;
}
int date(int j) { return (j + 1) % 7; }
void fromJulian(int j, int& y, int& m, int& d) {
  int f = j + 1363 + (((4*j + 274277) / 146097) * 3) / 4;
  int e = 4 * f + 3;
```

```
  int h = 5 * ((e % 1461) / 4) + 2;
  d = (h % 153) / 5 + 1;
  m = (h / 153 + 2) % 12 + 1;
  y = (e / 1461) + (14 - m) / 12 - 4716;
}
```

34 C++

Useful built-in functions

The number of ys in A such that $y \leq x$.

```
upper_bound(A,A+N,x) - A
```

The number of ys in A such that $y < x$.

```
lower_bound(A,A+N,x) - A
```

When I need to find maximum $y \in S$ such that $y \leq x$

```
multiset<int, greater<int> > S;
if (S.lower_bound(x) != S.end())
  cout << "y = " << *S.lower_bound(x) << endl;
int max_elem = *S.begin();
int min_elem = *(--S.end());
```

35 Math formula

35.1 Euler's Totient Function

Let $\phi(n)$ be he number of positive integers less than or equal to n that are relatively prime to n.

$$\phi(n) = n \prod_{p|n} (1 - \frac{1}{p}) \tag{1}$$

$a^{\phi(m)} \equiv 1 (\text{mod } m)$ if and only if a, m are relatively prime,

35.2 Catalan Number

$$C_n = \frac{\binom{2n}{n}}{n+1} = \frac{(2n)!}{(n+1)!n!} \tag{2}$$

$$= \prod_{k=2}^{n} \frac{n+k}{k} = \binom{2n}{n} - \binom{2n}{n+1} \tag{3}$$

$$= \frac{1}{n+1} \sum_{i=0}^{n} \binom{n}{i}^2 \tag{4}$$

$$C_0 = 1, C_{n+1} = \sum_{i=0}^{n} C_i C_{n-i} \tag{5}$$

35.3 Inclusion-exclusion principle

$$|\cup_{i=1}^{n} A_i| = \sum_{k=1}^{n} (-1)^{k+1} \Big(\sum_{1 \leq i_1 < \cdots < i_k \leq n} |A_{i_1} \cup \cdots \cup A_{i_k}| \Big) \tag{6}$$

or...

$$|\cup_{i=1}^{n} A_i| = \sum_{\emptyset \neq J \subset \{1,2,\cdots,n\}} (-1)^{|J|-1} \Big(\cup_{j \in J} A_j \Big) \tag{7}$$

```
// not tested. kind of out of book
int ans = 0;
rep(mask,1<<m) if (mask > 0) {
  int curans = 0;
  // calculate curans
  if (__builtin_popcount(mask) & 1) ans += curans;
  else ans -= curans;
}
```

35.4 Mőbuis function

$f(n) = \sum_{d|n} g(d) \leftrightarrow g(n) = \sum_{d|n} \mu(n/d) f(d)$. μ can be found in $O(\sqrt{n})$as following:
** out of the book and not tested **

```
map<int,int> moebius(int n) {
  map<int,int> res;
  vector<int> primes;

  // list all prime factors
  for (int i = 2; i * i <= n; i++) {
    if (n % i == 0) {
```

```
      primes.push_back(i);
      while (n % i == 0) n /= i;
    }
  }
  if (n != 1) primes.push_back(n);

  int m = primes.size();
  for (int i = 0; i < (1 << m); i++) {
    int mu = 1, d = 1;
    for (int j = 0; j < m; j++) {
      if (i >> j & 1) {
        mu *= -1;
        d *= primes[j];
      }
    }
    res[d] = mu;
  }
  return res;
}
void solve() {
  map<int,int> mu = moebius(n);
  for (map<int,int>::iterator it = mu.begin(); it != mu.end();
       it++){
    // do something
  }
}
```

36 Grundy

Grundy number to convert a game to Nim

```
int grundy(current state) {
  S = {}
  for (each option)
    Add (grundy(next state)) to S
  return min x >= 0 that is not in S
}
```

37 Thms in Graph Theory

Menger's theorem Let G be a finite undirected graph and x and y two nonadjacent vertices. Then the theorem states that the size of the minimum vertex cut for x and y (the minimum number of vertices whose removal disconnects x and y) is equal to the maximum number of pairwise vertex-independent paths from x to y. **Minimum cost flow** A flow f has the minimum cost iff the residual graph has no negative cycle.

Matching etc

- Matching: $M \subset E$ w/o common vertices.
- Edge Cover: $F \subset E$ s.t. every vertex is incident to at least one edge.
- Independent Set: $S \subset V$ s.t. no two of which are adjacent.
- Vertex Cover: $S \subset V$ s.t. each edge is incident to at least one vertex.

For any graph,

- |Max. Matching| + |Min. Edge Cover| = |V|,
- |Max. Indep. Set| + |Min. Vertex Cover| = |V|.

In bipartite graph, |Max. Matching| = |Min. Vertex Cover|.

38 2-SAT

1. Make a problem into $(a \vee b) \wedge (c \vee d)$...
2. For each $(a \vee b)$, add an edge $\neg a \implies b$, $\neg b \implies a$.
3. If $\exists x$, x and $\neg x$ are in the same SCC, then impossible
4. x is true iff x's SCC comes after $\neg x$'s SCC in topological sorting(or $cmp[x] > cmp[\neg x]$)