ICPC Notebook

TeamTam - National University of Singapore

Contents

Additional Notes (22 - 25)

1 Data Structure

1.1 Treap with implicit keys

```
// {{{ Treap with Implicit Keys
struct Treap {
  struct node {
    int x, cnt, mn, pos, rev;
    unsigned y;
    node *l, *r;
    node() { l = r = NULL; }
    node(int xx) {
      x = xx;
      y = rand();
      cnt = 1;
      mn = x;
      pos = 0;
      rev = 0;
      l = r = NULL;
    }
  };

  typedef node* Node;

  inline int cnt(Node x) { return x ? x->cnt : 0; }

  Node root, t1, t2, t3;

  Treap() { root = NULL; }

  inline void refresh(Node &x) {
    if (!x) return;
    x->cnt = 1 + cnt(x->l) + cnt(x->r);
    x->mn = x->x;
    x->pos = cnt(x->l);
    if (x->l) {
      push(x->l);
      if (x->mn > x->l->mn) x->pos = x->l->pos;
      MIN(x->mn, x->l->mn);
    }
    if (x->r) {
      push(x->r);
      if (x->mn > x->r->mn) x->pos = cnt(x->l) + 1 + x->r->pos;
      MIN(x->mn, x->r->mn);
    }
  }

  inline void push(Node &x) {
    if (!x) return;
    if (x->rev) {
      if (x->l) x->l->rev ^= 1;
      if (x->r) x->r->rev ^= 1;
      swap(x->l, x->r);
      x->pos = cnt(x) - 1 - x->pos;
      x->rev = 0;
    }
  }

  inline void split(Node x, const int &val, Node &l, Node &r, int add) {
    l = r = NULL;
    push(x);
    if (!x) return;
    int cur = add + cnt(x->l);
    if (val <= cur) {
      split(x->l, val, l, x->l, add);
      r = x;
    } else {
      split(x->r, val, x->r, r, add + 1 + cnt(x->l));
      l = x;
    }
    refresh(x);
  }

  inline Node merge(Node l, Node r) {
    push(l); push(r);
    if (!l || !r) return l ? l : r;
    if (l->y < r->y) {
      l->r = merge(l->r, r);
      refresh(l);
      return l;
    } else {
      r->l = merge(l, r->l);
      refresh(r);
      return r;
    }
  }

  inline void erase(Node &x) {
    push(x);
```

```
    if (x->l) {
      erase(x->l);
      refresh(x);
    } else {
      x = x->r;
      push(x);
    }
  }

  inline void insert(int x) { root = merge(root, new node(x)); }

  inline void update() { erase(root); }

  inline void reverse(int l, int r) {
    split(root, l, t1, t2, 0);
    split(t2, r-l+1, t2, t3, 0);
    t2->rev ^= 1;
    push(t2);
    root = merge(t1, t2);
    root = merge(root, t3);
  }
};
// }}}
```

1.2 Range fenwick tree

```
class FenwickTree {
private: vi ft1, ft2;
  int query(vi &ft, int b) {
    int sum = 0; for (; b; b -= LSOne(b)) sum += ft[b];
    return sum; }
  void adjust(vi &ft, int k, int v) {
    for (; k < (int)ft.size(); k += LSOne(k)) ft[k] += v; }
public:
  FenwickTree() {}
  FenwickTree(int n) { ft1.assign(n + 1, 0); ft2.assign(n+1, 0);}
  int query(int a) { return a * query(ft1, a) - query(ft2, a); }
  int query(int a, int b) { return query(b) - (a == 1 ? 0 : query(a-1)); }
  void adjust(int a, int b, int value){
    adjust(ft1, a, value);
    adjust(ft1, b+1, -value);
    adjust(ft2, a, value * (a-1));
    adjust(ft2, b+1, -1 * value * b);
  }
  int get(int n) {
    return query(n) - query(n-1);  }
};
```

1.3 Link Cut Tree

```
// {{{ Link cut tree
namespace LCT {
struct Node {
  int val;
  int sum;
  bool rev;
  Node *p, *c[2];
  Node(int val) : val(val), sum(val), rev(0), p(), c() {}
};

int getDir(Node *n) { return n->p->c[1] == n; }
int getSum(Node *n) { return n ? n->sum : 0; }
bool isRoot(Node *n) { return !n->p || n->p->c[getDir(n)] != n; }

void connect(Node *p, Node *n, int dir) {
  if (p) p->c[dir] = n;
  if (n) n->p = p;
}

void push(Node *n) {
  if (n->rev) {
    swap(n->c[0], n->c[1]);
    if (n->c[0]) n->c[0]->rev ^= true;
    if (n->c[1]) n->c[1]->rev ^= true;
    n->rev = false;
    n->sum = getSum(n->c[0]) + getSum(n->c[1]) + n->val;
  }
}


void update(Node *n) {
  push(n); push(n->c[0]); push(n->c[1]);
  n->sum = getSum(n->c[0]) + getSum(n->c[1]) + n->val;
}

void rotate(Node *n) {
  int dir = getDir(n);
  Node *p = n->p;
  if (isRoot(p)) n->p = p->p;
  else connect(p->p, n, getDir(p));
  connect(p, n->c[!dir], dir);
  connect(n, p, !dir);
  update(p);
  update(n);
}

Node *splay(Node *n) {
  while (!isRoot(n)) {
    Node *p = n->p;
    if (!isRoot(p)) push(p->p);
    push(p);
    push(n);
    if (!isRoot(p)) rotate(getDir(n) == getDir(p) ? p : n);
    rotate(n);
  }
  push(n);
  return n;
}

Node *access(Node *n) {
  splay(n);
  n->c[1] = nullptr;
  update(n);
  Node *last = n;
  while (n->p) {
    last = n->p;
    splay(n->p); // !!
    n->p->c[1] = n;
    update(n->p);
    splay(n);
  }
  push(n);
  update(n);
  return last;
}

Node *evert(Node *n) {
  access(n);
  n->rev ^= true;
  return n;
}

Node *getRoot(Node *n) {
  access(n);
  while (true) {
    push(n);
    if (n->c[0]) n = n->c[0];
    else break;
  }
  splay(n);
  return n;
}

void link(Node *a, Node *b) {
  evert(a);
  a->p = b;
}

void cut(Node *a, Node *b) { // a is the parent
  evert(a);
  access(b);
  assert(b->c[0] == a);
  b->c[0] = nullptr;
  a->p = nullptr;
  update(b);
}
}
// }}}
```

1.4 Splay Tree

```
// {{{ Splay Tree
struct Node {
  int size;
  int val, mn;
  int sum, lazy;
  bool rev;
  Node *p, *c[2];
  Node(int val) : size(1), val(val), mn(val), sum(val), lazy(0), rev(0), p(), c() {}
};
```

```
inline int getDir(Node *n) { return n->p->c[1] == n; }
inline int getMin(Node *n) { return n ? n->mn : INT_MAX; }
inline int getSum(Node *n) { return n ? n->sum : 0; }
inline int getSize(Node *n) { return n ? n->size : 0; }

inline void push(Node *n) {
  if (!n) return;
  if (n->rev) {
    swap(n->c[0], n->c[1]);
    if (n->c[0]) n->c[0]->rev ^= true;
    if (n->c[1]) n->c[1]->rev ^= true;
    n->rev = false;
  }
  if (n->lazy) {
    if (n->c[0]) n->c[0]->lazy += n->lazy;
    if (n->c[1]) n->c[1]->lazy += n->lazy;
    n->sum += n->lazy;
    n->mn += n->lazy;
    n->val += n->lazy;
    n->lazy = 0;
  }
}

inline void update(Node *n) {
  if (!n) return;
  push(n); push(n->c[0]); push(n->c[1]);
  n->size = getSize(n->c[0]) + getSize(n->c[1]) + 1;
  n->sum = getSum(n->c[0]) + getSum(n->c[1]) + n->val;
  n->mn = min(min(getMin(n->c[0]), getMin(n->c[1])), n->val);
}

inline void connect(Node *p, Node *n, int dir) {
  if (p) p->c[dir] = n;
  if (n) n->p = p;
}

inline void rotate(Node *n) {
  assert(n->p);
  int d = getDir(n);
  Node *p = n->p;
  if (p->p) connect(p->p, n, getDir(p));
  else n->p = nullptr;
  connect(p, n->c[!d], d);
  connect(n, p, !d);
  update(p);
  update(n);
}

Node *splay(Node *n) {
  while (n->p) {
    Node *p = n->p;
    if (p->p) push(p->p);
    push(p);
    push(n);
    if (p->p) rotate(getDir(n) == getDir(p) ? p : n);
    rotate(n);
  }
  push(n);
  return n;
}

Node *accessKey(Node *root, int x) {
  if (!root) return root;
  Node *cur = root;
  while (true) {
    push(cur);
    if (x == cur->val) break;
    int d = x > cur->val;
    if (!cur->c[d]) break;
    cur = cur->c[d];
  }
  return splay(cur);
}

Node *accessKth(Node *root, int k) {
  assert(k <= root->size);
  Node *cur = root;
  while (true) {
    push(cur);
    int upToCur = getSize(cur->c[0]) + 1;
    if (upToCur == k) return splay(cur);
    if (k > upToCur) {
      k -= upToCur;
      cur = cur->c[1];
    } else {
      cur = cur->c[0];
    }
  }
  assert(false);
}

void splitKey(Node *root, int x, Node *&l, Node *&r) {
  l = r = nullptr;
  if (!root) return;
  Node *cur = accessKey(root, x);
  if (cur->val <= x) {
    l = cur;
    r = cur->c[1];
    l->c[1] = nullptr;
    if (r) r->p = nullptr;
  } else {
    l = cur->c[0];
    r = cur;
    if (l) l->p = nullptr;
    r->c[0] = nullptr;
  }
  update(l); update(r);
}

void splitKth(Node *root, int k, Node *&l, Node *&r) {
  l = r = nullptr;
  if (!root) return;
  assert(k <= root->size);
  if (k == 0) {
    push(root);
    r = root;
    return;
  }
  root = accessKth(root, k);
  l = root;
  r = root->c[1];
  l->c[1] = nullptr;
  if (r) r->p = nullptr;
  update(l);
}

Node *join(Node *l, Node *r) {
  if (!l || !r) return l ? l : r;
  while (true) {
    push(l);
    if (!l->c[1]) break;
    l = l->c[1];
  }
  splay(l);
  connect(l, r, 1);
  push(r);
  update(l);
  return l;
}

Node *insert(Node *root, int val) {
  if (!root) return new Node(val);
  Node *l, *r;
  splitKey(root, val, l, r);
  if (!l || l->val != val) {
    Node *tmp = new Node(val);
    connect(tmp, l, 0);
    update(tmp);
    l = tmp;
  }
  return join(l, r);
}

Node *erase(Node *root, int val) {
  if (!root) return nullptr;
  root = accessKey(root, val);
  if (root->val != val) return root;
  else {
    Node *l = root->c[0], *r = root->c[1];
    if (l) l->p = nullptr;
    if (r) r->p = nullptr;
    delete root;
    return join(l, r);
  }
}

pair<Node*, int> queryKth(Node *root, int k) {
  if (!root || root->size < k) return {root, INT_MAX};
  root = accessKth(root, k);
  return {root, root->val};
}

pair<Node*, int> queryLt(Node *root, int x) {
  Node *l, *r;
  splitKey(root, x - 1, l, r);
  int ret = getSize(l);
  return {join(l, r), ret};
}

Node *add(Node *root, int l, int r, int v) {
  Node *t1, *t2, *t3;
  splitKth(root, l - 1, t1, t2);
  splitKth(t2, r - l + 1, t2, t3);
```

```
  if (t2) t2->lazy += v;
  return join(join(t1, t2), t3);
}

Node *reverse(Node *root, int l, int r) {
  Node *t1, *t2, *t3;
  splitKth(root, l - 1, t1, t2);
  splitKth(t2, r - l + 1, t2, t3);
  if (t2) t2->rev ^= true;
  return join(join(t1, t2), t3);
}

// }}}
```

2 String

2.1 Aho-Corasick

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

// {{{ Aho Corasick
namespace AhoCorasick {
  struct Node {
    int next[26];
    int parent;
    int fromParent;
    int fail;
    int wordCount;
  };

  int last;
  Node node[MAXNODE];

  void init() {
    last = 0;
    for (int i = 0; i < MAXNODE; i++) {
      for (int j = 0; j < 26; j++) {
        node[i].next[j] = -1;
      }
      node[i].parent = node[i].fail = -1;
      node[i].wordCount = 0;
    }
    node[0].fail = 0;
  }

  int addString(char *s, int offset = 'a') {
    int len = strlen(s);
    int cur = 0;
    for (int i = 0; i < len; i++) { //no C++11? BibleThump
      if (node[cur].next[s[i] - offset] == -1) {
        node[cur].next[s[i] - offset] = ++last;
        node[last].parent = cur;
        node[last].fromParent = s[i] - offset;
      }
      cur = node[cur].next[s[i] - offset];
    }
    node[cur].wordCount++;
    return cur;
  }

  void constructFail() {
    queue<int> q;
    q.push(0);
    while (!q.empty()) {
      int u = q.front();
      q.pop();
      for (int i = 0; i < 26; i++) {
        if (node[u].next[i] != -1) {
          q.push(node[u].next[i]);
        }
      }
      if (u == 0) {
        continue;
      }
      int v = node[node[u].parent].fail;
      int c = node[u].fromParent;
      while (v != 0 && node[v].next[c] == -1) {
        v = node[v].fail;
      }
      node[u].fail = node[v].next[c] != -1  && node[v].next[c] != u ? node[v].next[c] : 0;
    }
  }

  int search(char *s, int offset) {
    int len = strlen(s);
    int cur = 0;
    int match = 0;
    for (int i = 0; i < len; i++) {
      while (cur != 0 && node[cur].next[s[i] - offset] == -1) {
        assert(node[cur].fail != -1);
        cur = node[cur].fail;
      }
      if (node[cur].next[s[i] - offset] != -1) {
        cur = node[cur].next[s[i] - offset];
      }
      assert(cur != -1);
      match += node[cur].wordCount;
    }
    return match;
  }
}
// }}}
```

2.2 Hash

```
// {{{ Hash
template<int pr, int MOD> class Hash {
  int n;
  vector<int> p, v;
  void init() {
    p.resize(n);
    p[0] = 1;
    for (int i = 1; i < n; i++) {
      p[i] = (int)(1LL * p[i - 1] * pr % MOD);
      v[i] = (int)(1LL * v[i - 1] * pr % MOD) + v[i];
      if (v[i] >= MOD) v[i] -= MOD;
    }
  }
public:
  Hash() {}
  Hash(string s) {
    n = (int)s.length();
    v.resize(n);
    for (int i = 0; i < n; i++) {
      v[i] = s;
    }
    init();
  }
  Hash(const vector<int> &v) {
    this->v = v;
    n = (int)v.size();
    for (auto &x : this->v) {
      x %= MOD;
      if (x < 0) x += MOD;
    }
    init();
  }
  int get(int l, int r) {
    int res = v[r] - (l > 0 ? (int)(1LL * v[l - 1] * p[r - l + 1] % MOD) : 0);
    if (res >= MOD) res -= MOD;
    if (res < 0) res += MOD;
    return res;
  }
};
// }}}
```

2.3 SuffixArray

```
// Source: http://codeforces.com/contest/452/submission/7269543
// Efficient Suffix Array O(N*logN)

// String index from 0
// Usage:
// string s;
// SuffixArray sa(s);
// Now we can use sa.SA and sa.LCP
struct SuffixArray {
    string a;
    int N, m;
    vector<int> SA, LCP, x, y, w, c;

    SuffixArray(string _a, int m) : a(" " + _a), N(a.length()), m(m),
            SA(N), LCP(N), x(N), y(N), w(max(m, N)), c(N) {
        a[0] = 0;
        DA();
        kasaiLCP();
        #define REF(X) { rotate(X.begin(), X.begin()+1, X.end()); X.pop_back(); }
```

```
        REF(SA); REF(LCP);
        a = a.substr(1, a.size());
        for(int i = 0; i < SA.size(); ++i) --SA[i];
        #undef REF
    }

    inline bool cmp (const int a, const int b, const int l) { return (y[a] == y[b] && y[a + l] == y[b
        + l]); }

    void Sort() {
        for(int i = 0; i < m; ++i) w[i] = 0;
        for(int i = 0; i < N; ++i) ++w[x[y[i]]];
        for(int i = 0; i < m - 1; ++i) w[i + 1] += w[i];
        for(int i = N - 1; i >= 0; --i) SA[--w[x[y[i]]]] = y[i];
    }

    void DA() {
        for(int i = 0; i < N; ++i) x[i] = a[i], y[i] = i;
        Sort();
        for(int i, j = 1, p = 1; p < N; j <<= 1, m = p) {
            for(p = 0, i = N - j; i < N; i++) y[p++] = i;
            for (int k = 0; k < N; ++k) if (SA[k] >= j) y[p++] = SA[k] - j;
            Sort();
            for(swap(x, y), p = 1, x[SA[0]] = 0, i = 1; i < N; ++i)
                x[SA[i]] = cmp(SA[i - 1], SA[i], j) ? p - 1 : p++;
        }
    }

    void kasaiLCP() {
        for (int i = 0; i < N; i++) c[SA[i]] = i;
        for (int i = 0, j, k = 0; i < N; LCP[c[i++]] = k)
            if (c[i] > 0) for (k ? k-- : 0, j = SA[c[i] - 1]; a[i + k] == a[j + k]; k++);
            else k = 0;
    }
};
```

2.4 Manacher

```
const char DUMMY = '.';

int manacher(string s) {
    // Add dummy character to not consider odd/even length
    // NOTE: Ensure DUMMY does not appear in input
    // NOTE: Remember to ignore DUMMY when tracing

    int n = s.size() * 2 - 1;
    vector <int> f = vector <int>(n, 0);
    string a = string(n, DUMMY);
    for (int i = 0; i < n; i += 2) a[i] = s[i / 2];

    int l = 0, r = -1, center, res = 0;
    for (int i = 0, j = 0; i < n; i++) {
        j = (i > r ? 0 : min(f[l + r - i], r - i)) + 1;
        while (i - j >= 0 && i + j < n && a[i - j] == a[i + j]) j++;
        f[i] = --j;
        if (i + j > r) {
            r = i + j;
            l = i - j;
        }

        int len = (f[i] + i % 2) / 2 * 2 + 1 - i % 2;
        if (len > res) {
            res = len;
            center = i;
        }
    }
    // a[center - f[center]..center + f[center]] is the needed substring
    return res;
}
```

2.5 Z-Function

```
// z[i] =       d i  x u con d i  nh t  b  t      u  t    v   tr  i m  tr ng  v  i   o  n
          u  c  a  v[]
void zfunc(string v, vector<int> &z) {
    int n = v.length(); z.resize(n);
    int l = 0, r = -1;
    z[0] = 0;
    for(int i = 1; i < n-1; ++i) {
        int k = (i > r) ? 0 : min(z[i - l], r - i + 1);
        while (i + k < n && v[k] == v[i + k]) ++k;
        z[i] = k;
        if (i + k - 1 > r) {
            l = i;
            r = i + k - 1;
        }
    }
    z[0] = n;
}
```

2.6 EertreE

```
#include <map>
using std::map;

const int MAX = 3e5 + 5;

// PalindromicTree.init()
// PalindromicTree.addChar(c)
// PalindromicTree::Char
// {{{ PalindromicTree
struct PalindromicTree {
  typedef char Char;

  map<Char, int> to[MAX];
  int link[MAX], len[MAX];
  int occ[MAX];
  Char s[MAX];
  int n, last, sz = 0;

  PalindromicTree() { init(); }

  void init() {
    for_each(to, to + MAX, [] (map<Char, int> &m) { m.clear(); });
    memset(link, -1, sizeof(link));
    memset(len, -1, sizeof(len));
    n = 0;
    link[0] = 0;
    link[1] = 0;
    len[0] = -1;
    len[1] = 0;
    last = 1;
    sz = 2;
  }

  int getLink(int v) {
    while (n - len[v] - 2 < 0 || (s[n - len[v] - 2] != s[n - 1])) {
      v = link[v];
    }
    return v;
  }

  void addChar(Char c) {
    s[n++] = c;
    int lnk = getLink(last);
    if (to[lnk].count(c) == 0) {
      len[sz] = len[lnk] + 2;
      if (len[sz] == 1) {
        link[sz] = 1;
      } else {
        //TODO remove
        int next = getLink(link[lnk]);
        assert(to[next].count(c));
        link[sz] = to[next][c];
      }
      to[lnk][c] = sz;
      sz++;
    }
    last = to[lnk][c];
    occ[last]++;
  }

  ll go() {
    ll ret = 0;
    for (int i = sz - 1; i >= 0; i--) {
      occ[link[i]] += occ[i];
      ret = max(ret, 1LL * len[i] * occ[i]);
    }
    // vector<string> result(sz);
    // queue<int> q;
    // for (auto it : to[0]) {
    //   result[it.second] = string(1, it.first);
    //   q.push(it.second);
    // }
    // for (auto it : to[1]) {
    //   result[it.second] = string(2, it.first);
    //   q.push(it.second);
    // }
    // while (!q.empty()) {
    //   int u = q.front();
    //   q.pop();
```

```
    //   for (auto it : to[u]) {
    //     if (result[it.second] != "") {
    //       continue;
    //     }
    //     result[it.second] = it.first + result[u] + it.first;
    //     q.push(it.second);
    //   }
    // }
    // for (int i = 2; i < sz; i++) {
    //   cout << result[i] << " " << occ[i] << endl;
    // }
    return ret;
  }
};
// }}}
```

3 Graph

3.1 2-SAT

```
// VAR(x), NOT(x)
// TwoSat(int varCount)
// bool TwoSat.solve() -> vector<int> TwoSat.value
// {{{ 2-SAT
using graph = vector<vi>;
inline int VAR(int x) { return 2 * x; }
inline int NOT(int x) { return x ^ 1; }

struct TwoSat {
  int n;
  vi value;
  graph g, grev;
  TwoSat(int n) : n(2 * n), g(2 * n) {}

  void addImplication(int x, int y) {
    g[x].emplace_back(y);
    g[NOT(y)].emplace_back(NOT(x));
  }

  void addOr(int x, int y) {
    g[NOT(x)].emplace_back(y);
    g[NOT(y)].emplace_back(x);
  }

  void dfs(int u, graph &g, vi &visit, vi &order) {
    visit[u] = 1;
    for (int v : g[u]) {
      if (!visit[v]) {
        dfs(v, g, visit, order);
      }
    }
    order.push_back(u);
  }

  bool solve() {
    grev.assign(n, vi());
    for (int i = 0; i < n; i++) {
      for (int j : g[i]) {
        grev[j].push_back(i);
      }
    }
    vi visited(n, false);
    vi order;
    for (int i = 0; i < n; i++) {
      if (!visited[i]) {
        dfs(i, g, visited, order);
      }
    }
    reverse(ALL(order));
    vi repr(n);
    fill(ALL(visited), false);
    value.assign(n, -1);
    for (int u : order) {
      if (!visited[u]) {
        vi tmp;
        dfs(u, grev, visited, tmp);
        for (int v : tmp) {
          repr[v] = tmp[0];
          if (value[v] == -1) {
            value[v] = 1;
            value[NOT(v)] = 0;
          }
        }
      }
    }
    for (int i = 0; i < n; i++) {
      if (repr[i] == repr[NOT(i)]) {
        return false;
      }
    }
    return true;
  }
};
// }}}
```

3.2 Biconnected Component

```
// Input graph: vector< vector<int> > a, int n
// Note: 0-indexed
// Usage: BiconnectedComponent bc; (bc.components is the list of components)

struct BiconnectedComponent {
  vector<int> low, num, s;
  vector< vector<int> > components;
  int counter;

  BiconnectedComponent() : num(n, -1), low(n, -1), counter(0) {
    for (int i = 0; i < n; i++)
      if (num[i] < 0)
        dfs(i, 1);
  }

  void dfs(int x, int isRoot) {
    low[x] = num[x] = ++counter;
    if (a[x].empty()) {
      components.push_back(vector<int>(1, x));
      return;
    }
    s.push_back(x);

    for (int i = 0; i < a[x].size(); i++) {
      int y = a[x][i];
      if (num[y] > -1) low[x] = min(low[x], num[y]);
      else {
        dfs(y, 0);
        low[x] = min(low[x], low[y]);

        if (isRoot || low[y] >= num[x]) {
          components.push_back(vector<int>(1, x));
          while (1) {
            int u = s.back();
            s.pop_back();
            components.back().push_back(u);
            if (u == y) break;
          }
        }
      }
    }
  }
};
```

3.3 Bridge Articulation

```
// Assume already have undirected graph vector< vector<int> > G with V vertices
// Vertex index from 0
// Usage:
// UndirectedDfs tree;
// Then you can use tree.bridges and tree.cuts
struct UndirectedDfs {
    vector<int> low, num, parent;
    vector<bool> articulation;
    int counter, root, children;

    vector< pair<int,int> > bridges;
    vector<int> cuts;

    UndirectedDfs() : low(V, 0), num(V, -1), parent(V, 0), articulation(V, false),
            counter(0), children(0) {
        for(int i = 0; i < V; ++i) if (num[i] == -1) {
            root = i; children = 0;
            dfs(i);
            articulation[root] = (children > 1);
        }
        for(int i = 0; i < V; ++i)
            if (articulation[i]) cuts.push_back(i);
    }
private:
    void dfs(int u) {
```

```
        low[u] = num[u] = counter++;
        for(int j = 0; j < G[u].size(); ++j) {
            int v = G[u][j];
            if (num[v] == -1) {
                parent[v] = u;
                if (u == root) children++;
                dfs(v);
                if (low[v] >= num[u])
                    articulation[u] = true;
                if (low[v] > num[u]) bridges.push_back(make_pair(u, v));
                low[u] = min(low[u], low[v]);
            } else if (v != parent[u])
                low[u] = min(low[u], num[v]);
        }
    }
};
```

3.4 Dinic Max Flow

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

//Add Edge : MaxFlow.addEdge(int from, int to, F capacity)
//Solve : MaxFlow.calcMaxFlow(int source, int sink)
// {{{ Dinic Max Flow<F=int>
template<typename F = int>
struct DinicMaxFlow {
  struct Edge {
    int to, rev;
    F flow, cap;
  };

  const F INF_FLOW = numeric_limits<F>::max() / 2;
  const int INF_DIST = 1e9;

  int source, sink;
  vector<vector<Edge>> graph;
  vector<int> dist, ptr;

  DinicMaxFlow(int N) : graph(N), dist(N), ptr(N) {}

  void addEdge(int from, int to, F cap) {
    if (from == to) {
      return;
    }
    Edge e1 { to, SZ(graph[to]), 0, cap };
    Edge e2 { from, SZ(graph[from]), 0, 0 };
    graph[from].push_back(e1);
    graph[to].push_back(e2);
  }

  bool buildLevelGraph() {
    fill(begin(dist), end(dist), INF_DIST);
    queue<int> q;
    dist[source] = 0;
    q.push(source);
    while (!q.empty()) {
      int u = q.front();
      q.pop();
      for (const auto &e : graph[u]) {
        if (e.flow < e.cap && dist[e.to] > dist[u] + 1) {
          dist[e.to] = dist[u] + 1;
          q.push(e.to);
        }
      }
    }
    return dist[sink] != INF_DIST;
  }

  F dfs(int u, F bottleneck) {
    if (u == sink || bottleneck == 0) {
      return bottleneck;
    }
    for (int &i = ptr[u]; i < SZ(graph[u]); i++) {
      auto &e = graph[u][i];
      auto &b = graph[e.to][e.rev];
      if (dist[e.to] != dist[u] + 1 || e.flow >= e.cap) {
        continue;
      }
      F pushed = dfs(e.to, min(e.cap - e.flow, bottleneck));
      if (pushed > 0) {
        e.flow += pushed;
        b.flow -= pushed;
        return pushed;
      }
    }
    return 0;
  }

  F calcMaxFlow(const int source, const int sink) {
    this->source = source;
    this->sink = sink;
    F mf = 0;
    while (buildLevelGraph()) {
      fill(begin(ptr), end(ptr), 0);
      F pushed;
      while ((pushed = dfs(source, INF_FLOW))) {
        mf += pushed;
      }
    }
    return mf;
  }
};
// }}}
```

3.5 Directed MST

```
#include "../../template.h"

const int maxe = 100111;
const int maxv = 100;

// Index from 0
// Running time O(E*V)
namespace chuliu {
    struct Cost;
    vector<Cost> costlist;

    struct Cost {
        int id, val, used, a, b, pos;
        Cost() { val = -1; used = 0; }
        Cost(int _id, int _val, bool temp) {
            a = b = -1; id = _id; val = _val; used = 0;
            pos = costlist.size(); costlist.push_back(*this);
        }
        Cost(int _a, int _b) {
            a = _a; b = _b; id = -1; val = costlist[a].val - costlist[b].val;
            used = 0; pos = costlist.size(); costlist.push_back(*this);
        }
        void push() {
            if (id == -1) {
                costlist[a].used += used;
                costlist[b].used -= used;
            }
        }
    };

    struct Edge {
        int u, v;
        Cost cost;
        Edge() {}
        Edge(int id, int _u, int _v, int c) {
            u = _u; v = _v; cost = Cost(id, c, 0);
        }
    } edge[maxe];

    int n, m, root, pre[maxv], node[maxv], vis[maxv], best[maxv];

    void init(int _n) {
        n = _n; m = 0;
        costlist.clear();
    }

    void add(int id, int u, int v, int c) {
        edge[m++] = Edge(id, u, v, c);
    }

    int mst(int root) {
        int ret = 0;

        while (true) {
            REP(i, n) best[i] = -1;

            REP(e, m) {
                int u = edge[e].u, v = edge[e].v;
                if ((best[v] == -1 || edge[e].cost.val < costlist[best[v]].val) && u != v) {
                    pre[v] = u;
                    best[v] = edge[e].cost.pos;
                }
            }

            REP(i, n) if (i != root && best[i] == -1) return -1;

            int cntnode = 0;
            memset(node, -1, sizeof node); memset(vis, -1, sizeof vis);
```

```
            REP(i, n) if (i != root) {
                ret += costlist[best[i]].val;
                costlist[best[i]].used++;

                int v = i;
                while (vis[v] != i && node[v] == -1 && v != root) {
                    vis[v] = i;
                    v = pre[v];
                }

                if (v != root && node[v] == -1) {
                    for (int u = pre[v]; u != v; u = pre[u]) node[u] = cntnode;
                    node[v] = cntnode++;
                }
            }

            if (cntnode == 0) break;

            REP(i, n) if (node[i] == -1) node[i] = cntnode++;

            REP(e, m) {
                int v = edge[e].v;
                edge[e].u = node[edge[e].u];
                edge[e].v = node[edge[e].v];
                if (edge[e].u != edge[e].v) edge[e].cost = Cost(edge[e].cost.pos, best[v]);
            }

            n = cntnode;
            root = node[root];
        }

        return ret;
    }

    vector<int> trace() {
        vector<int> ret;
        FORD(i, costlist.size()-1,0) costlist[i].push();
        REP(i, costlist.size()) {
            Cost cost = costlist[i];
            if (cost.id != -1 && cost.used > 0) ret.push_back(cost.id);
        }
        return ret;
    }
}

int main() {
}
```

3.6 Eulerian Cycle

```
// directed!
// careful with empty graph!
// {{{ eulerianTour
template<typename T>
bool eulerianTour(const vector<vector<T>> &g, vi &tour) {
  int n = SZ(g);
  int edgeCount = 0;
  vi inDeg(n), outDeg(n);
  for (int i = 0; i < n; i++) {
    outDeg[i] = SZ(g[i]);
    for (auto &it : g[i]) {
      inDeg[it.fi]++;
      edgeCount++;
    }
  }
  int out = -1, in = -1;
  for (int i = 0; i < n; i++) {
    if (inDeg[i] == outDeg[i] + 1) {
      debug printf("finish vertex %d\n", i);
      if (out == -1) {
        out = i;
      } else {
        return false;
      }
    } else if (outDeg[i] == inDeg[i] + 1) {
      debug printf("start vertex %d\n", i);
      if (in == -1) {
        in = i;
      } else {
        return false;
      }
    } else if (outDeg[i] != inDeg[i]) {
      return false;
    }
  }
  // either zero or both
  if ((in != -1) ^ (out != -1)) {
    return false;
  }
  if (in == -1) {
    for (int i = 0; i < n; i++) {
      if (outDeg[i] != 0) {
        in = out = i;
        break;
      }
    }
    // empty graph?
    if (in == -1) {
      in = out = 0;
    }
  }
  tour.clear();
  stack<int> st;
  vi visit(n);
  vi ptr(n);
  st.push(in);
  while (!st.empty()) {
    int u = st.top();
    visit[u] = true;
    if (ptr[u] == SZ(g[u])) {
      tour.push_back(u);
      st.pop();
    } else {
      st.push(g[u][ptr[u]++].fi);
    }
  }
  reverse(ALL(tour));
  return SZ(tour) == edgeCount + 1;
}
// }}}
```

3.7 Edmonds Blossom

```
// General matching on graph

const int maxv = 1000;
const int maxe = 50000;

// Index from 1
// Directed
struct EdmondsLawler {
    int n, E, start, finish, newRoot, qsize, adj[maxe], next[maxe], last[maxv], mat[maxv], que[maxv],
        dad[maxv], root[maxv];
    bool inque[maxv], inpath[maxv], inblossom[maxv];

    void init(int _n) {
        n = _n; E = 0;
        for(int x=1; x<=n; ++x) { last[x] = -1; mat[x] = 0; }
    }
    void add(int u, int v) {
        adj[E] = v; next[E] = last[u]; last[u] = E++;
    }
    int lca(int u, int v) {
        for(int x=1; x<=n; ++x) inpath[x] = false;
        while (true) {
            u = root[u];
            inpath[u] = true;
            if (u == start) break;
            u = dad[mat[u]];
        }
        while (true) {
            v = root[v];
            if (inpath[v]) break;
            v = dad[mat[v]];
        }
        return v;
    }
    void trace(int u) {
        while (root[u] != newRoot) {
            int v = mat[u];

            inblossom[root[u]] = true;
            inblossom[root[v]] = true;

            u = dad[v];
            if (root[u] != newRoot) dad[u] = v;
        }
    }
    void blossom(int u, int v) {
        for(int x=1; x<=n; ++x) inblossom[x] = false;

        newRoot = lca(u, v);
        trace(u); trace(v);

        if (root[u] != newRoot) dad[u] = v;
```

```
        if (root[v] != newRoot) dad[v] = u;

        for(int x=1; x<=n; ++x) if (inblossom[root[x]]) {
            root[x] = newRoot;
            if (!inque[x]) {
                inque[x] = true;
                que[qsize++] = x;
            }
        }
    }
    bool bfs() {
        for(int x=1; x<=n; ++x){
            inque[x] = false;
            dad[x] = 0;
            root[x] = x;
        }
        qsize = 0;
        que[qsize++] = start;
        inque[start] = true;
        finish = 0;

        for(int i=0; i<qsize; ++i) {
            int u = que[i];
            for (int e = last[u]; e != -1; e = next[e]) {
                int v = adj[e];
                if (root[v] != root[u] && v != mat[u]) {
                    if (v == start || (mat[v] > 0 && dad[mat[v]] > 0)) blossom(u, v);
                    else if (dad[v] == 0) {
                        dad[v] = u;
                        if (mat[v] > 0) que[qsize++] = mat[v];
                        else {
                            finish = v;
                            return true;
                        }
                    }
                }
            }
        }
        return false;
    }
    void enlarge() {
        int u = finish;
        while (u > 0) {
            int v = dad[u], x = mat[v];
            mat[v] = u;
            mat[u] = v;
            u = x;
        }
    }
    int maxmat() {
        for(int x=1; x<=n; ++x) if (mat[x] == 0) {
            start = x;
            if (bfs()) enlarge();
        }

        int ret = 0;
        for(int x=1; x<=n; ++x) if (mat[x] > x) ++ret;
        return ret;
    }
} edmonds;
```

3.8 Hungarian Assignment

```
const int MN = 3210;
const int inf = 1000111000;

// Vertex: 1 --> nx, 1 --> ny
// cost >= 0
// fx[x] + fy[y] <= cost[x][y] for all x, y
// Min cost matching

struct Hungary {
    int nx, ny, cost[MN][MN], fx[MN], fy[MN], matx[MN], which[MN], dist[MN];
    bool used[MN];

    void init(int _nx, int _ny) {
        nx = _nx; ny = _ny;
        memset(fx, 0, sizeof fx);
        memset(fy, 0, sizeof fy);
        memset(used, false, sizeof used);
        memset(matx, 0, sizeof matx);
        for(int i=0; i<=nx; ++i) for(int j=0; j<=ny; ++j) cost[i][j] = inf;
    }

    void add(int x, int y, int c) { cost[x][y] = min(cost[x][y],c); }

    int mincost() {
        for(int x=1; x<=nx; ++x) {
            int y0 = 0; matx[0] = x;
            for(int y=0; y<=ny; ++y) { dist[y] = inf + 1; used[y] = false; }

            do {
                used[y0] = true;
                int x0 = matx[y0], delta = inf + 1, y1;

                for(int y=1; y<=ny; ++y) if (!used[y]) {
                    int curdist = cost[x0][y] - fx[x0] - fy[y];
                    if (curdist < dist[y]) {
                        dist[y] = curdist;
                        which[y] = y0;
                    }
                    if (dist[y] < delta) {
                        delta = dist[y];
                        y1 = y;
                    }
                }

                for(int y=0; y<=ny; ++y) if (used[y]) {
                    fx[matx[y]] += delta;
                    fy[y] -= delta;
                } else dist[y] -= delta;

                y0 = y1;
            } while (matx[y0] != 0);

            do {
                int y1 = which[y0];
                matx[y0] = matx[y1];
                y0 = y1;
            } while (y0);
        }
//      return -fy[0]; // n  u  lu n     m  b  o  c   b   gh p     y
        int ret = 0;
        for(int y=1; y<=ny; ++y) {
            int x = matx[y];
            if (cost[x][y] < inf) ret += cost[x][y];
        }
        return ret;
    }
} hungary;
```

3.9 Min Cost Max Flow

```
// Min Cost Max Flow - SPFA
// Index from 0
// Lots of double comparison --> likely to fail for double
// Example:
// MinCostFlow mcf(n);
// mcf.addEdge(1, 2, 3, 4);
// cout << mcf.minCostFlow() << endl;

template<class Flow=int, class Cost=int>
struct MinCostFlow {
    const Flow INF_FLOW = 1000111000;
    const Cost INF_COST = 1000111000111000LL;

    int n, t, S, T;
    Flow totalFlow;
    Cost totalCost;
    vector<int> last, visited;
    vector<Cost> dis;
    struct Edge {
        int to;
        Flow cap;
        Cost cost;
        int next;
        Edge(int to, Flow cap, Cost cost, int next) :
                to(to), cap(cap), cost(cost), next(next) {}
    };
    vector<Edge> edges;

    MinCostFlow(int n) : n(n), t(0), totalFlow(0), totalCost(0), last(n, -1), visited(n, 0), dis(n, 0)
            {
        edges.clear();
    }

    int addEdge(int from, int to, Flow cap, Cost cost) {
        edges.push_back(Edge(to, cap, cost, last[from]));
        last[from] = t++;
        edges.push_back(Edge(from, 0, -cost, last[to]));
        last[to] = t++;
        return t - 2;
    }

    pair<Flow, Cost> minCostFlow(int _S, int _T) {
```

```
        S = _S; T = _T;
        SPFA();
        while (1) {
            while (1) {
                REP(i,n) visited[i] = 0;
                if (!findFlow(S, INF_FLOW)) break;
            }
            if (!modifyLabel()) break;
        }
        return make_pair(totalFlow, totalCost);
    }

private:
    void SPFA() {
        REP(i,n) dis[i] = INF_COST;
        priority_queue< pair<Cost,int> > Q;
        Q.push(make_pair(dis[S]=0, S));
        while (!Q.empty()) {
            int x = Q.top().second;
            Cost d = -Q.top().first;
            Q.pop();
            // For double: dis[x] > d + EPS
            if (dis[x] != d) continue;
            for(int it = last[x]; it >= 0; it = edges[it].next)
                if (edges[it].cap > 0 && dis[edges[it].to] > d + edges[it].cost)
                    Q.push(make_pair(-(dis[edges[it].to] = d + edges[it].cost), edges[it].to));
        }
        REP(i,n) dis[i] = dis[T] - dis[i];
    }

    Flow findFlow(int x, Flow flow) {
        if (x == T) {
            totalCost += dis[S] * flow;
            totalFlow += flow;
            return flow;
        }
        visited[x] = 1;
        Flow now = flow;
        for(int it = last[x]; it >= 0; it = edges[it].next)
            // For double: fabs(dis[edges[it].to] + edges[it].cost - dis[x]) < EPS
            if (edges[it].cap && !visited[edges[it].to] && dis[edges[it].to] + edges[it].cost == dis[x
                ]) {
                Flow tmp = findFlow(edges[it].to, min(now, edges[it].cap));
                edges[it].cap -= tmp;
                edges[it ^ 1].cap += tmp;
                now -= tmp;
                if (!now) break;
            }
        return flow - now;
    }

    bool modifyLabel() {
        Cost d = INF_COST;
        REP(i,n) if (visited[i])
            for(int it = last[i]; it >= 0; it = edges[it].next)
                if (edges[it].cap && !visited[edges[it].to])
                    d = min(d, dis[edges[it].to] + edges[it].cost - dis[i]);

        // For double: if (d > INF_COST / 10)      INF_COST = 1e20
        if (d == INF_COST) return false;
        REP(i,n) if (visited[i])
            dis[i] += d;
        return true;
    }
};
```

3.10 Strongly Connected Component

```
// Assume that already have directed graph vector< vector<int> > G with V vertices
// Index from 0
// Usage:
// DirectedDfs tree;
// Now you can use tree.scc
struct DirectedDfs {
    vector<int> num, low, current, S;
    int counter;
    vector< vector<int> > scc;

    DirectedDfs() : num(V, -1), low(V, 0), current(V, 0), counter(0) {
        REP(i,V) if (num[i] == -1) dfs(i);
    }

    void dfs(int u) {
        low[u] = num[u] = counter++;
        S.push_back(u);
        current[u] = 1;
        REP(j, G[u].size()) {
            int v = G[u][j];
            if (num[v] == -1) dfs(v);
            if (current[v]) low[u] = min(low[u], low[v]);
        }
        if (low[u] == num[u]) {
            scc.push_back(vector<int>());
            while (1) {
                int v = S.back(); S.pop_back(); current[v] = 0;
                scc.back().push_back(v);
                if (u == v) break;
            }
        }
    }
};
```

3.11 LRFlow

```
#include "Dinic.cpp"

//Construct : LRFlow(int N, int originalSource, int originalSink)
//Add Edge : LRFlow.addEdge(int from, int to, F lowerBound, F upperBound) []
//Solve: LRFlow.calcLRFlow()
// {{{ LR FLow<F=int, MF=DinicMaxFlow<F>>
template<typename F = int, typename MF = DinicMaxFlow<F>>
struct LRFlow {
  MF mf;
  F lowerBoundSum = 0;
  int n, realSource, realSink;

  LRFlow(int n, int src, int snk) : mf(n + 2), n(n + 2) {
    realSource = this->n - 2;
    realSink = this->n - 1;
    mf.addEdge(snk, src, mf.INF_FLOW);
  }

  void addEdge(int from, int to, F lowerBound, F upperBound) {
    assert(lowerBound <= upperBound);
    lowerBoundSum += lowerBound;
    mf.addEdge(from, to, upperBound - lowerBound);
    mf.addEdge(realSource, to, lowerBound);
    mf.addEdge(from, realSink, lowerBound);
  }

  pair<F, bool> calcLRFlow() {
    F maxFlow = mf.calcMaxFlow(realSource, realSink);
    return make_pair(maxFlow, maxFlow == lowerBoundSum);
  }
};
// }}}
```

3.12 Hopcroft Karp

```
#include <algorithm>
#include <iostream>

using namespace std;

const int MAXN1 = 50000;
const int MAXN2 = 50000;
const int MAXM = 150000;

int n1, n2, edges, last[MAXN1], prev[MAXM], head[MAXM];
int matching[MAXN2], dist[MAXN1], Q[MAXN1];
bool used[MAXN1], vis[MAXN1];

void init(int _n1, int _n2) {
    n1 = _n1;
    n2 = _n2;
    edges = 0;
    fill(last, last + n1, -1);
}

void addEdge(int u, int v) {
    head[edges] = v;
    prev[edges] = last[u];
    last[u] = edges++;
}

void bfs() {
    fill(dist, dist + n1, -1);
    int sizeQ = 0;
    for (int u = 0; u < n1; ++u) {
        if (!used[u]) {
```

```
            Q[sizeQ++] = u;
            dist[u] = 0;
        }
    }
    for (int i = 0; i < sizeQ; i++) {
        int u1 = Q[i];
        for (int e = last[u1]; e >= 0; e = prev[e]) {
            int u2 = matching[head[e]];
            if (u2 >= 0 && dist[u2] < 0) {
                dist[u2] = dist[u1] + 1;
                Q[sizeQ++] = u2;
            }
        }
    }
}

bool dfs(int u1) {
    vis[u1] = true;
    for (int e = last[u1]; e >= 0; e = prev[e]) {
        int v = head[e];
        int u2 = matching[v];
        if (u2 < 0 || !vis[u2] && dist[u2] == dist[u1] + 1 && dfs(u2)) {
            matching[v] = u1;
            used[u1] = true;
            return true;
        }
    }
    return false;
}

int maxMatching() {
    fill(used, used + n1, false);
    fill(matching, matching + n2, -1);
    for (int res = 0;;) {
        bfs();
        fill(vis, vis + n1, false);
        int f = 0;
        for (int u = 0; u < n1; ++u)
            if (!used[u] && dfs(u))
                ++f;
        if (!f)
            return res;
        res += f;
    }
}

int main() {
    init(2, 2);

    addEdge(0, 0);
    addEdge(0, 1);
    addEdge(1, 1);

    cout << (2 == maxMatching()) << endl;
}
```

3.13 Heavy Light Decomposition

```
template <class T, int V>
class HeavyLight {
  int parent[V], heavy[V], depth[V];
  int root[V], treePos[V];
  SegmentTree<T> tree;

  template <class G>
  int dfs(const G& graph, int v) {
    int size = 1, maxSubtree = 0;
    for (int u : graph[v]) if (u != parent[v]) {
      parent[u] = v;
      depth[u] = depth[v] + 1;
      int subtree = dfs(graph, u);
      if (subtree > maxSubtree) heavy[v] = u, maxSubtree = subtree;
      size += subtree;
    }
    return size;
  }

  template <class BinaryOperation>
  void processPath(int u, int v, BinaryOperation op) {
    for (; root[u] != root[v]; v = parent[root[v]]) {
      if (depth[root[u]] > depth[root[v]]) swap(u, v);
      op(treePos[root[v]], treePos[v] + 1);
    }
    if (depth[u] > depth[v]) swap(u, v);
    op(treePos[u], treePos[v] + 1);
  }

public:
  template <class G>
  void init(const G& graph) {
    int n = graph.size();
    fill_n(heavy, n, -1);
    parent[0] = -1;
    depth[0] = 0;
    dfs(graph, 0);
    for (int i = 0, currentPos = 0; i < n; ++i)
      if (parent[i] == -1 || heavy[parent[i]] != i)
        for (int j = i; j != -1; j = heavy[j]) {
          root[j] = i;
          treePos[j] = currentPos++;
        }
    tree.init(n);
  }

  void set(int v, const T& value) {
    tree.set(treePos[v], value);
  }

  void modifyPath(int u, int v, const T& value) {
    processPath(u, v, [this, &value](int l, int r) { tree.modify(l, r, value); });
  }

  T queryPath(int u, int v) {
    T res = T();
    processPath(u, v, [this, &res](int l, int r) { res.add(tree.query(l, r)); });
    return res;
  }
};
```

3.14 Gomory-Hu

```
// tree which represents all-pair min-cut
// min-cut of u-v = minimum edge in path from u to v in tree
// combine the following function with usual max flow routine
// also because we need to reset the flow graph during each max flow, // don't forget to add variable
      in struct to record initial capacity
void buildGomoryHu() {
  for(int i = 2 ; i <= n ; i++)
    parent[i] = 1;
  for(int i = 2 ; i <= n ; i++) {
    int minCut = maxFlow(i,parent[i]);
    tree[parent[i]].pb({i,minCut});
    // the one below is not necessary if we want to make a rooted tree tree[i].pb({parent[i],minCut});
    for(int j = i + 1 ; j <= n ; j++) {
      if(dist[j] != -1 && parent[j] == parent[i]) {
        parent[j] = i;
      }
    }
  }
}
```

3.15 Online-Bridge

```
// O(N + M)
int psetBridge[N], psetComp[N]; // UF super vertice, UF tree int par[N];
bool seen[N];
int size[N];
int bridge;
int n;
void reset() {
  for(int i = 0 ; i <= n ; i++) {
    psetBridge[i] = psetComp[i] = i;
    par[i] = -1;
    seen[i] = 0;
    size[i] = 1;
  }
  bridge = 0;
}
int findsB(int x) {
  if(x == -1) return -1;
  return x == psetBridge[x] ? x : psetBridge[x] = findsB(psetBridge[x]);
}
int findsC(int x) {
  x = findsB(x);
  return x == psetComp[x] ? x : psetComp[x] = findsC(psetComp[x]);
}
void makeRoot(int x) { // make super vertice which contains x root of its tree
  x = findsB(x);
  int root = x;
  int chld = -1;
```

```
  while(x != -1) {
    int papa = findsB(par[x]);
    par[x] = chld;
    psetComp[x] = root;
    chld = x;
    x = papa;
  }
  size[root] = size[chld];
}
void mergePath(int u,int v) { // remove bridge between u and v in tree vector<int> vu,vv;
  int lca = -1;
  while(1) {
    if(u != -1) {
      u = findsB(u);
      vu.push_back(u);
      if(seen[u]) {
        lca = u;
        break;
      }
      seen[u] = 1;
      u = par[u];
    }
    if(v != -1) {
      v = findsB(v);
      vv.push_back(v);
      if(seen[v]) {
        lca = v;
        break;
      }
      seen[v] = 1;
      v = par[v];
    }
  }
  for(int i = 0 ; i < 2 ; i++) {
    vector<int> &proc = i ? vv : vu;
    for(int x : proc) {
      psetBridge[x] = lca;
      if(x == lca) {
        break;
      }
      bridge--;
    }
    for(int x : proc)
      seen[x] = 0;
  }
}
void addEdge(int u,int v) {
  u = findsB(u);
  v = findsB(v);
  if(u != v) {
    int uu = findsC(u);
    int vv = findsC(v);
    if(uu != vv) {
      bridge++;
      if(size[uu] > size[vv]) {
        swap(u,v);
        swap(vv,uu);
      }
      makeRoot(u);
      par[u] = psetComp[u] = v;
      size[vv] += size[u];
    } else
      mergePath(u,v);
  }
}
```

4 Math

4.1 Chinese Remainder Theorem

```
bool linearCongruences(const vector<int> &a, const vector<int> &b,
        const vector<int> &m, int &x, int &M) {
    int n =     a.size();
    x = 0; M = 1;
    REP(i, n) {
        int a_ = a[i] % M, b_ = b[i] - a[i] * x, m_ = m[i];
        int y, t, g = extgcd(a_, m_, y, t);
        if (b_ % g) return false;
        b_ /= g; m_ /= g;
        x += M * (y * b_ % m_);
        M *= m_;
    }
    x = (x + M) % M;
    return true;
}
```

4.2 Fast Fourier Transform

```
// {{{ FFT Generic
template<typename T>
struct FFTTrait {
  static T getRoot(int, bool) { assert(false); }
};

template<>
struct FFTTrait<complex<double>> {
  static complex<double> getRoot(int len, bool rev) {
    double ang = 2 * M_PI / len;
    return rev ? complex<double>{cos(ang), sin(ang)} : complex<double>{cos(-ang), sin(-ang)};
  }
};

template<typename T = complex<double>, typename Trait = FFTTrait<T>>
struct FFT {
  const static int MAXLEN = 1 << 20;
  static void reorder(vector<T> &data) {
    int n = SZ(data);
    assert(n > 0 && (n & (n - 1)) == 0);
    int bitCount = __builtin_ctz(n);
    for (int i = 0; i < n; i++) {
      int j = 0;
      for (int b = 0; b < bitCount; b++) {
        if (i & (1 << b)) {
          j |= 1 << (bitCount - 1 - b);
        }
      }
      if (i < j) {
        swap(data[i], data[j]);
      }
    }
  }
  static void fix(vector<T> &data) {
    int n = 1;
    while (n < SZ(data)) n *= 2;
    n *= 2;
    data.resize(n);
  }
  static void fft(vector<T> &data, bool rev) {
    reorder(data);
    int n = SZ(data);
    for (int len = 2; len <= n; len <<= 1) {
      static T roots[MAXLEN];
      int len2 = len >> 1;
      T w = Trait::getRoot(len, rev);
      roots[0] = T(1);
      for (int i = 1; i < len2; i++) {
        roots[i] = roots[i - 1] * w; // binary? can be precomputed
      }
      for (int i = 0; i < n; i += len) {
        for (int j = 0; j < len2; j++) {
          T p = data[i + j];
          T q = roots[j] * data[i + j + len2];
          data[i + j] = p + q;
          data[i + j + len2] = p - q;
        }
      }
    }
    if (rev) {
      for (int i = 0; i < n; i++) {
        data[i] /= n;
      }
    }
  }
};
// }}}
```

4.3 Gauss Jordan

```
// Gauss-Jordan elimination with full pivoting.
//
// Uses:
//   (1) solving systems of linear equations (AX=B)
//   (2) inverting matrices (AX=I)
//   (3) computing determinants of square matrices
//
// Running time: O(n^3)
//
// INPUT:    a[][] = an nxn matrix
//           b[][] = an nxm matrix
//
```

```
// OUTPUT:   X      = an nxm matrix (stored in b[][])
//           A^{-1} = an nxn matrix (stored in a[][])
//           returns determinant of a[][]

#include <iostream>
#include <vector>
#include <cmath>

using namespace std;

const double EPS = 1e-10;

typedef vector<int> VI;
typedef double T;
typedef vector<T> VT;
typedef vector<VT> VVT;

T GaussJordan(VVT &a, VVT &b) {
  const int n = a.size();
  const int m = b[0].size();
  VI irow(n), icol(n), ipiv(n);
  T det = 1;

  for (int i = 0; i < n; i++) {
    int pj = -1, pk = -1;
    for (int j = 0; j < n; j++) if (!ipiv[j])
      for (int k = 0; k < n; k++) if (!ipiv[k])
        if (pj == -1 || fabs(a[j][k]) > fabs(a[pj][pk])) { pj = j; pk = k; }
    if (fabs(a[pj][pk]) < EPS) { cerr << "Matrix is singular." << endl; exit(0); }
    ipiv[pk]++;
    swap(a[pj], a[pk]);
    swap(b[pj], b[pk]);
    if (pj != pk) det *= -1;
    irow[i] = pj;
    icol[i] = pk;

    T c = 1.0 / a[pk][pk];
    det *= a[pk][pk];
    a[pk][pk] = 1.0;
    for (int p = 0; p < n; p++) a[pk][p] *= c;
    for (int p = 0; p < m; p++) b[pk][p] *= c;
    for (int p = 0; p < n; p++) if (p != pk) {
      c = a[p][pk];
      a[p][pk] = 0;
      for (int q = 0; q < n; q++) a[p][q] -= a[pk][q] * c;
      for (int q = 0; q < m; q++) b[p][q] -= b[pk][q] * c;
    }
  }

  for (int p = n-1; p >= 0; p--) if (irow[p] != icol[p]) {
    for (int k = 0; k < n; k++) swap(a[k][irow[p]], a[k][icol[p]]);
  }

  return det;
}

int main() {
  const int n = 4;
  const int m = 2;
  double A[n][n] = { {1,2,3,4},{1,0,1,0},{5,3,2,4},{6,1,4,6} };
  double B[n][m] = { {1,2},{4,3},{5,6},{8,7} };
  VVT a(n), b(n);
  for (int i = 0; i < n; i++) {
    a[i] = VT(A[i], A[i] + n);
    b[i] = VT(B[i], B[i] + m);
  }

  double det = GaussJordan(a, b);

  // expected: 60
  cout << "Determinant: " << det << endl;

  // expected: -0.233333 0.166667 0.133333 0.0666667
  //           0.166667 0.166667 0.333333 -0.333333
  //           0.233333 0.833333 -0.133333 -0.0666667
  //           0.05 -0.75 -0.1 0.2
  cout << "Inverse: " << endl;
  for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++)
      cout << a[i][j] << ' ';
    cout << endl;
  }

  // expected: 1.63333 1.3
  //           -0.166667 0.5
  //           2.36667 1.7
  //           -1.85 -1.35
  cout << "Solution: " << endl;
  for (int i = 0; i < n; i++) {
    for (int j = 0; j < m; j++)
      cout << b[i][j] << ' ';
    cout << endl;
  }
}
```

4.4 Simplex

```
// Two-phase simplex algorithm for solving linear programs of the form
//
//     maximize     c^T x
//     subject to   Ax <= b
//                  x >= 0
//
// INPUT: A -- an m x n matrix
//        b -- an m-dimensional vector
//        c -- an n-dimensional vector
//        x -- a vector where the optimal solution will be stored
//
// OUTPUT: value of the optimal solution (infinity if unbounded
//         above, nan if infeasible)
//
// To use this code, create an LPSolver object with A, b, and c as
// arguments.  Then, call Solve(x).

#include <iostream>
#include <iomanip>
#include <vector>
#include <cmath>
#include <limits>

using namespace std;

typedef long double DOUBLE;
typedef vector<DOUBLE> VD;
typedef vector<VD> VVD;
typedef vector<int> VI;

const DOUBLE EPS = 1e-9;

struct LPSolver {
  int m, n;
  VI B, N;
  VVD D;

  LPSolver(const VVD &A, const VD &b, const VD &c) :
    m(b.size()), n(c.size()), N(n + 1), B(m), D(m + 2, VD(n + 2)) {
    for (int i = 0; i < m; i++) for (int j = 0; j < n; j++) D[i][j] = A[i][j];
    for (int i = 0; i < m; i++) { B[i] = n + i; D[i][n] = -1; D[i][n + 1] = b[i]; }
    for (int j = 0; j < n; j++) { N[j] = j; D[m][j] = -c[j]; }
    N[n] = -1; D[m + 1][n] = 1;
  }

  void Pivot(int r, int s) {
    double inv = 1.0 / D[r][s];
    for (int i = 0; i < m + 2; i++) if (i != r)
      for (int j = 0; j < n + 2; j++) if (j != s)
        D[i][j] -= D[r][j] * D[i][s] * inv;
    for (int j = 0; j < n + 2; j++) if (j != s) D[r][j] *= inv;
    for (int i = 0; i < m + 2; i++) if (i != r) D[i][s] *= -inv;
    D[r][s] = inv;
    swap(B[r], N[s]);
  }

  bool Simplex(int phase) {
    int x = phase == 1 ? m + 1 : m;
    while (true) {
      int s = -1;
      for (int j = 0; j <= n; j++) {
        if (phase == 2 && N[j] == -1) continue;
        if (s == -1 || D[x][j] < D[x][s] || D[x][j] == D[x][s] && N[j] < N[s]) s = j;
      }
      if (D[x][s] > -EPS) return true;
      int r = -1;
      for (int i = 0; i < m; i++) {
        if (D[i][s] < EPS) continue;
        if (r == -1 || D[i][n + 1] / D[i][s] < D[r][n + 1] / D[r][s] ||
          (D[i][n + 1] / D[i][s]) == (D[r][n + 1] / D[r][s]) && B[i] < B[r]) r = i;
      }
      if (r == -1) return false;
      Pivot(r, s);
    }
  }

  DOUBLE Solve(VD &x) {
    int r = 0;
    for (int i = 1; i < m; i++) if (D[i][n + 1] < D[r][n + 1]) r = i;
    if (D[r][n + 1] < -EPS) {
      Pivot(r, n);
      if (!Simplex(1) || D[m + 1][n + 1] < -EPS) return -numeric_limits<DOUBLE>::infinity();
```

```
      for (int i = 0; i < m; i++) if (B[i] == -1) {
        int s = -1;
        for (int j = 0; j <= n; j++)
          if (s == -1 || D[i][j] < D[i][s] || D[i][j] == D[i][s] && N[j] < N[s]) s = j;
        Pivot(i, s);
      }
    }
    if (!Simplex(2)) return numeric_limits<DOUBLE>::infinity();
    x = VD(n);
    for (int i = 0; i < m; i++) if (B[i] < n) x[B[i]] = D[i][n + 1];
    return D[m][n + 1];
  }
};

int main() {

  const int m = 4;
  const int n = 3;
  DOUBLE _A[m][n] = {
    { 6, -1, 0 },
    { -1, -5, 0 },
    { 1, 5, 1 },
    { -1, -5, -1 }
  };
  DOUBLE _b[m] = { 10, -4, 5, -5 };
  DOUBLE _c[n] = { 1, -1, 0 };

  VVD A(m);
  VD b(_b, _b + m);
  VD c(_c, _c + n);
  for (int i = 0; i < m; i++) A[i] = VD(_A[i], _A[i] + n);

  LPSolver solver(A, b, c);
  VD x;
  DOUBLE value = solver.Solve(x);

  cerr << "VALUE: " << value << endl; // VALUE: 1.29032
  cerr << "SOLUTION:"; // SOLUTION: 1.74194 0.451613 1
  for (size_t i = 0; i < x.size(); i++) cerr << " " << x[i];
  cerr << endl;
  return 0;
}
```

4.5 Sqrt Mod

```
// Jacobi Symbol (m/n), m, n  0   and n is odd
// (m/n)==1    x^2 == m (mod n) solvable, -1 unsolvable
#define NEGPOW(e) ((e) % 2 ? -1 : 1)
int jacobi(int a, int m) {
   if (a == 0) return m == 1 ? 1 : 0;
   if (a % 2)  return NEGPOW((a-1)*(m-1)/4)*jacobi(m%a, a);
   else return NEGPOW((m*m-1)/8)*jacobi(a/2, m);
}
int invMod(int a, int m) {
   int x, y;
   if (extgcd(a, m, x, y) == 1) return (x + m) % m;
   else                         return 0; // unsolvable
}
// No solution when: n(p-1)/2 = -1 mod p
int sqrtMod(int n, int p) { //find x: x2 = n (mod p) p is prime
   int S, Q, W, i, m = invMod(n, p);
   for (Q = p - 1, S = 0; Q % 2 == 0; Q /= 2, ++S);
   do { W = rand() % p; } while (W == 0 || jacobi(W, p) != -1);
   for (int R = powMod(n, (Q+1)/2, p), V = powMod(W, Q, p); ;) {
      int z = R * R * m % p;
      for (i = 0; i < S && z % p != 1; z *= z, ++i);
      if (i == 0) return R;
      R = (R * powMod(V, 1 << (S-i-1), p)) % p;
   }
}
int powMod (int a, int b, int p) {
        int res = 1;
        while (b)
                if (b & 1)
                        res = int (res * 1ll * a % p),  --b;
                else
                        a = int (a * 1ll * a % p),  b >>= 1;
        return res;
}
```

4.6 Brent, Miller Rabin

```
#include "../Prime/RabinMiller.h"

long long mul(long long a, long long b, long long mod) {
    if (b == 0) return 0;
    if (b == 1) return a % mod;
    long long mid = mul(a, b >> 1, mod);
    mid = (mid + mid) % mod;
    if (b & 1) return (mid + a) % mod;
    else return mid;
}

long long brent(long long n) {
    if (n == 1) return 1;
    if (!(n & 1)) return 2;
    if (!(n % 3)) return 3;

    const int p[3] = {1, 3, 5};
    long long y, q, x, ys, g, my = 3;
    int i, j, k, m, r, c;

    for (i = 0; i < my; ++i) {
        y = 1; r = 1; q = 1; m = 111; c = p[i];

        do {
            x = y; k = 0;
            for (j = 1; j <= r; ++j) y = (mul(y, y, n) + c) % n;
            do {
                ys = y;
                for (j = 1; j <= min(m, r-k); ++j) {
                    y = (mul(y, y, n) + c) % n;
                    q = mul(q, abs(x - y), n);
                }
                g = __gcd(q, n); k += m;
            } while (k < r && g < 2);
            r <<= 1;
        } while (g < 2);

        if (g == n)
            do {
                ys = (mul(ys, ys, n) + c) % n;
                g = __gcd(abs(x - ys), n);
            } while (g < 2);

        if (g != n) return g;
    }
    return n;
}
```

4.7 FFT Integer

```
//#include <iostream>
#include <fstream>
#include <vector>
#include <set>
#include <map>
#include <cstring>
#include <string>
#include <cmath>
#include <cassert>
#include <ctime>
#include <algorithm>
#include <sstream>
#include <list>
#include <queue>
#include <deque>
#include <stack>
#include <cstdlib>
#include <cstdio>
#include <iterator>
#include <functional>
#include <bitset>
#define mp make_pair
#define pb push_back

//#define DEBUG

#ifdef LOCAL
#define eprintf(...) fprintf(stderr,__VA_ARGS__)
#else
#define eprintf(...)
#endif

#define TIMESTAMP(x) eprintf("["#x"] Time : %.3lf s.\n", clock()*1.0/CLOCKS_PER_SEC)
#define TIMESTAMPf(x,...) eprintf("[" x "] Time : %.3lf s.\n", __VA_ARGS__, clock()*1.0/CLOCKS_PER_SEC
    )

#if ( ( _WIN32 || __WIN32__ ) && __cplusplus < 201103L)
  #define LLD "%I64d"
```

```
#else
  #define LLD "%lld"
#endif

using namespace std;

#define TASKNAME "F"

typedef vector<int> vi;
#define sz(a) ((int)(a).size())

#ifdef LOCAL
static struct __timestamper {
  string what;
  __timestamper(const char* what) : what(what){};
  __timestamper(const string& what) : what(what){};
  ~__timestamper(){
    TIMESTAMPf("%s", what.data());
  }
} __TIMESTAMPER("end");
#else
struct __timestamper {};
#endif

typedef long long ll;
typedef long double ld;


const int MOD = 998244353;
const int ROOT = 3;

int mmul(int a, int b) {
        return (a * 1LL * b) % MOD;
}
void madd(int& a, int b) {
    if ((a += b) >= MOD) a -= MOD;
}

int mpow(int a, int b){
  if (!b) return 1;
  if (b & 1) return (mpow(a, b-1) * 1LL * a) % MOD;
  int temp = mpow(a, b/2);
  return (temp * 1LL * temp) % MOD;
}

bool checkRoot(int x){
  return mpow(x, (MOD - 1) / 7) != 1 && mpow(x, (MOD - 1) / 17) != 1 && mpow(x, (MOD - 1) / 2) != 1;
}

// BEGIN ALGO
const int MROOT = 19;
const int MROOTP = 1<<MROOT;
int rts[MROOTP + 10], brev[MROOTP + 10];
// \emph{Don't forget to call before}
void PreCalcRoots(){
  rts[0] = 1; // ROOT is primary root for MOD
  rts[1] = mpow(ROOT, (MOD-1) / MROOTP);
  for (int i = 2; i < MROOTP; i++)
    rts[i] = mmul(rts[i-1], rts[1]);
  for (int i = 0; i < MROOTP; i++) /*BOXNEXT*/
    brev[i] = (brev[i>>1]>>1) | ((i&1) << (MROOT-1));
}
inline void butterfly(int &a, int &b, int x){
  int temp = mmul(x, b); b = a;
  madd(a, temp); madd(b, MOD - temp);
}
void fft(vi &a, bool inv){
  int n = __builtin_ctz(sz(a));
  for (int i = 0; i < (1<<n); i++){
    int temp = brev[i] >> (MROOT - n);
    if (temp > i) swap(a[i], a[temp]);
  }
  for (int step = 0; step < n; step++){
    int pos = 0; /*BOXNEXT*/
    int dlt = (inv ? -1 : 1) * (1 << (MROOT - step - 1));
    for (int i = 0; i < (1<<n); i++){
      if ((i ^ (1<<step)) > i) /*BOXNEXT*/
        butterfly(a[i], a[i ^ (1<<step)], rts[pos]);
      pos = (pos + dlt) & (MROOTP-1);
    }
  }
}
vi multiply(vi a, vi b){
  int rsz = sz(a) + sz(b), rsz2 = 1;
  while (rsz2 < rsz) rsz2 *= 2;
  a.resize(rsz2); b.resize(rsz2);
  fft(a, false); fft(b, false);
  for (int i = 0; i < sz(a); i++)
    a[i] = mmul(a[i], b[i]);
  fft(a, true);
  int in = mpow(sz(a), MOD - 2);
  for (int i = 0; i < sz(a); i++)
    a[i] = mmul(a[i], in);
  return a;
}
vi inverse(vi a){
  assert(a[0] != 0);
  vi x(1, mpow(a[0], MOD - 2));
  while (sz(x) < sz(a)) { /*BOXNEXT*/
    vi temp(a.begin(), a.begin() + min(sz(a), 2*sz(x)));
    vi nx = multiply(multiply(x, x), temp);
    x.resize(2*sz(x));
    for (int i = 0; i < sz(x); i++) /*BOXNEXT*/
      madd(x[i], x[i]), madd(x[i], MOD - nx[i]);
  }
  return x;
}
// END ALGO

const int MAXN = 257000;

int facs[MAXN];
int ifacs[MAXN];

void PreCalcFacs(){
  facs[0] = ifacs[0] = 1;
  for (int i = 1; i < MAXN; i++){
    facs[i] = (facs[i-1] * 1LL * i) % MOD;
    ifacs[i] = mpow(facs[i], MOD - 2);
  }
}

int cnk(int n, int k){
  if (n < 0 || k < 0 || k > n) return false;
  int res = facs[n];
  res = (res * 1LL * ifacs[k]) % MOD;
  res = (res * 1LL * ifacs[n-k]) % MOD;
  return res;
}

ll get2(int x1, int x2, int t){
  int dlt = t - abs(x1 - x2);
  if (dlt < 0 || dlt % 2) return 0;
  return cnk(t, dlt / 2);
}

ll get1(int x1, int x2, int y1, int y2, int t){
  return (get2(x1 + y1, x2 + y2, t) * 1LL * get2(x1 - y1, x2 - y2, t)) % MOD;
}

int get(int x1, int x2, int y1, int y2, int t){
  return (get1(x1, x2, y1, y2, t) -
      get1(x1,-x2, y1, y2, t) -
      get1(x1, x2, y1,-y2, t) +
      get1(x1,-x2, y1,-y2, t) +
      3LL * MOD) % MOD;
}

int main(){
  assert(checkRoot(ROOT));
  PreCalcRoots();
  PreCalcFacs();
  #ifdef LOCAL
  assert(freopen(TASKNAME".in","r",stdin));
  assert(freopen(TASKNAME".out","w",stdout));
  #endif

  int x1, y1, x2, y2, t;
  scanf("%d %d %d %d %d",&x1,&y1,&x2,&y2, &t);

  vi res(t+1);
  vi res0(t+1);

  for (int i = 0; i <= t; i++){
    res[i] = get(x1, x2, y1, y2, i);
    res0[i] = get(x2, x2, y2, y2, i);
  }

  res = multiply(res, inverse(res0));

  printf("%d\n", res[t]);

  return 0;
}
```

5 Geometry

5.1 Rectangle in Rectangle

```
// Checks if rectangle of sides x,y fits inside one of sides X,Y
// Not tested with doubles but should work fine :)
// Code as written rejects rectangles that just touch.
bool rect_in_rect(int X, int Y, int x, int y) {
   if (Y > X) swap(Y, X);
   if (y > x) swap(y, x);
   double diagonal = sqrt(double(X)*X + double(Y)*Y);
   if (x < X && y < Y) return true;
   else if (y >= Y || x >= diagonal) return false;
   else {
      double w, theta, tMin = PI/4, tMax = PI/2;
      while (tMax - tMin > EPS) {
         theta = (tMax + tMin)/2.0;
         w = (Y-x*cos(theta))/sin(theta);
         if (w < 0 || x * sin(theta) + w * cos(theta) < X) tMin = theta;
         else tMax = theta;
      }
      return (w > y);
   }
}
```

5.2 Geometry Basic

```
#define EPS 1e-6

double DEG_to_RAD(double d) { return d * M_PI / 180.0; }
double RAD_to_DEG(double r) { return r * 180.0 / M_PI; }

inline int cmp(double a, double b) {
    return (a < b - EPS) ? -1 : ((a > b + EPS) ? 1 : 0);
}

struct Point {
    double x, y;
    Point(double x = 0.0, double y = 0.0) : x(x), y(y) {}

    Point operator + (Point a) { return Point(x+a.x, y+a.y); }
    Point operator - (Point a) { return Point(x-a.x, y-a.y); }
    Point operator * (double k) { return Point(x*k, y*k); }
    Point operator / (double k) { return Point(x/k, y/k); }

    double operator * (Point a) { return x*a.x + y*a.y; } // dot product
    double operator % (Point a) { return x*a.y - y*a.x; } // cross product

    int cmp(Point q) const { if (int t = ::cmp(x,q.x)) return t; return ::cmp(y,q.y); }

    #define Comp(x) bool operator x (Point q) const { return cmp(q) x 0; }
    Comp(>) Comp(<) Comp(==) Comp(>=) Comp(<=) Comp(!=)
    #undef Comp

    Point conj() { return Point(x, -y); }
    double norm() { return x*x + y*y; }

    // Note: There are 2 ways for implementing len():
    // 1. sqrt(norm()) --> fast, but inaccurate (produce some values that are of order X^2)
    // 2. hypot(x, y) --> slow, but much more accurate
    double len() { return sqrt(norm()); }

    Point rotate(double alpha) {
        double cosa = cos(alpha), sina = sin(alpha);
        return Point(x * cosa - y * sina, x * sina + y * cosa);
    }
};

int ccw(Point a, Point b, Point c) {
    return cmp((b-a)%(c-a),0);
}

double angle(Point a, Point o, Point b) { // angle AOB
    a = a - o; b = b - o;
    return acos((a * b) / sqrt(a.norm() * b.norm()));
}

// Distance from p to Line ab (closest Point --> c)
double distToLine(Point p, Point a, Point b, Point &c) {
    Point ap = p - a, ab = b - a;
    double u = (ap * ab) / ab.norm();
    c = a + (ab * u);
    return (p-c).len();
}

// Distance from p to segment ab (closest Point --> c)
double distToLineSegment(Point p, Point a, Point b, Point &c) {
    Point ap = p - a, ab = b - a;
    double u = (ap * ab) / ab.norm();
    if (u < 0.0) {
        c = Point(a.x, a.y);
        return (p - a).len();
    }
    if (u > 1.0) {
        c = Point(b.x, b.y);
        return (p - b).len();
    }
    return distToLine(p, a, b, c);
}

struct Line {
    double a, b, c;
    Point A, B; // Added for polygon intersect line. Do not rely on assumption that these are valid

    Line(double a, double b, double c) : a(a), b(b), c(c) {}

    Line(Point A, Point B) : A(A), B(B) {
        a = B.y - A.y;
        b = A.x - B.x;
        c = - (a * A.x + b * A.y);
    }
    Line(Point P, double m) {
        a = -m; b = 1;
        c = -((a * P.x) + (b * P.y));
    }
    double f(Point A) {
        return a*A.x + b*A.y + c;
    }
};

bool areParallel(Line l1, Line l2) {
    return cmp(l1.a*l2.b, l1.b*l2.a) == 0;
}

bool areSame(Line l1, Line l2) {
    return areParallel(l1 ,l2) && cmp(l1.c*l2.a, l2.c*l1.a) == 0;
}

bool areIntersect(Line l1, Line l2, Point &p) {
    if (areParallel(l1, l2)) return false;
    double dx = l1.b*l2.c - l2.b*l1.c;
    double dy = l1.c*l2.a - l2.c*l1.a;
    double d  = l1.a*l2.b - l2.a*l1.b;
    p = Point(dx/d, dy/d);
    return true;
}

void closestPoint(Line l, Point p, Point &ans) {
    if (fabs(l.b) < EPS) {
        ans.x = -(l.c) / l.a; ans.y = p.y;
        return;
    }
    if (fabs(l.a) < EPS) {
        ans.x = p.x; ans.y = -(l.c) / l.b;
        return;
    }
    Line perp(l.b, -l.a, - (l.b*p.x - l.a*p.y));
    areIntersect(l, perp, ans);
}

void reflectionPoint(Line l, Point p, Point &ans) {
    Point b;
    closestPoint(l, p, b);
    ans = p + (b - p) * 2;
}
```

5.3 Geometry Circle

```
struct Circle : Point {
    double r;
    Circle(double x = 0, double y = 0, double r = 0) : Point(x, y), r(r) {}
    Circle(Point p, double r) : Point(p), r(r) {}

    bool contains(Point p) { return (*this - p).len() <= r + EPS; }
};

// Find common tangents to 2 circles
// Helper method
```

```
void tangents(Point c, double r1, double r2, vector<Line> & ans) {
    double r = r2 - r1;
    double z = sqr(c.x) + sqr(c.y);
    double d = z - sqr(r);
    if (d < -EPS)  return;
    d = sqrt(fabs(d));
    Line l((c.x * r + c.y * d) / z,
           (c.y * r - c.x * d) / z,
           r1);
    ans.push_back(l);
}
// Actual method: returns vector containing all common tangents
vector<Line> tangents(Circle a, Circle b) {
    vector<Line> ans; ans.clear();
    for (int i=-1; i<=1; i+=2)
        for (int j=-1; j<=1; j+=2)
            tangents(b-a, a.r*i, b.r*j, ans);
    for(int i = 0; i < ans.size(); ++i)
        ans[i].c -= ans[i].a * a.x + ans[i].b * a.y;

    vector<Line> ret;
    for(int i = 0; i < (int) ans.size(); ++i) {
        bool ok = true;
        for(int j = 0; j < i; ++j)
            if (areSame(ret[j], ans[i])) {
                ok = false;
                break;
            }
        if (ok) ret.push_back(ans[i]);
    }
    return ret;
}

// Circle & line intersection
vector<Point> intersection(Line l, Circle cir) {
    double r = cir.r, a = l.a, b = l.b, c = l.c + l.a*cir.x + l.b*cir.y;
    vector<Point> res;

    double x0 = -a*c/(a*a+b*b),  y0 = -b*c/(a*a+b*b);
    if (c*c > r*r*(a*a+b*b)+EPS) return res;
    else if (fabs(c*c - r*r*(a*a+b*b)) < EPS) {
        res.push_back(Point(x0, y0) + Point(cir.x, cir.y));
        return res;
    }
    else {
        double d = r*r - c*c/(a*a+b*b);
        double mult = sqrt (d / (a*a+b*b));
        double ax,ay,bx,by;
        ax = x0 + b * mult;
        bx = x0 - b * mult;
        ay = y0 - a * mult;
        by = y0 + a * mult;

        res.push_back(Point(ax, ay) + Point(cir.x, cir.y));
        res.push_back(Point(bx, by) + Point(cir.x, cir.y));
        return res;
    }
}

double commonCircleArea(Circle c1, Circle c2) { //return the common area of two circle
    double d = hypot(c1.x-c2.x, c1.y-c2.y), area;
    if (c1.r+c2.r <= d) area = 0;
    else if (c2.r+d <= c1.r) area = (c1.r * c1.r - c2.r * c2.r) * M_PI;
    else if (c1.r+d <= c2.r) area = (c2.r * c2.r - c1.r * c1.r) * M_PI;
    else {
        double p1 = c2.r * c2.r * acos((d*d + c2.r*c2.r - c1.r*c1.r) / (2*d*c2.r));
        double p2 = c1.r * c1.r * acos((d*d + c1.r*c1.r - c2.r*c2.r) / (2*d*c1.r));
        double p3 = 0.5 * sqrt((-d+c2.r+c1.r) * (d+c2.r-c1.r) * (d+c1.r-c2.r) * (d+c1.r+c2.r));
        area = p1 + p2 - p3;
    }
    return area;
}

// Given 2 circles: (0, 0, r) and (c2.x, c2.y, c2.r)
// Intersections are intersections with line Ax + By + C where
// A = -2 * c2.x
// B = -2 * c2.y
// C = c2.x^2 + c2.y^2 + r^2 - c2.r^2
```

5.4 Geometry Polygon

```
typedef vector< Point > Polygon;

// Convex Hull: returns minimum number of vertices. Should work when N <= 1 and when all points
// are collinear
struct comp_hull {
    Point pivot;
    bool operator() (Point q,Point w) {
        Point Q = q - pivot, W = w - pivot;
        double R = Q % W;
        if (cmp(R,0)) return R < 0;
        return cmp(Q*Q,W*W) < 0;
    }
};
Polygon convex_hull(Polygon p) { // minimum vertices
    int j = 0, k, n = p.size();
    Polygon r(n);
    if (!n) return r;
    comp_hull comp;
    comp.pivot = *min_element(p.begin(),p.end());
    sort(p.begin(),p.end(),comp);
    for(int i = 0; i < n; ++i) {
        while (j > 1 && ccw(r[j-1],r[j-2],p[i]) <= 0) j--;
        r[j++] = p[i];
    }
    r.resize(j);
    if (r.size() >= 2 && r.back() == r.front()) r.pop_back();
    return r;
}

// Area, perimeter, centroid
double signed_area(Polygon p) {
    double area = 0;
    for(int i = 0; i < p.size(); i++) {
        int j = (i+1) % p.size();
        area += p[i].x*p[j].y - p[j].x*p[i].y;
    }
    return area / 2.0;
}
double area(const Polygon &p) {
    return fabs(signed_area(p));
}
Point centroid(Polygon p) {
    Point c(0,0);
    double scale = 6.0 * signed_area(p);
    for (int i = 0; i < p.size(); i++){
        int j = (i+1) % p.size();
        c = c + (p[i]+p[j])*(p[i].x*p[j].y - p[j].x*p[i].y);
    }
    return c / scale;
}
double perimeter(Polygon P) {
    double res = 0;
    for(int i = 0; i < P.size(); ++i) {
        int j = (i + 1) % P.size();
        res += (P[i] - P[j]).len();
    }
    return res;
}
// Is convex: checks if polygon is convex. Assume there are no 3 collinear points
bool is_convex(const Polygon &P) {
    int sz = (int) P.size();
    if (sz <= 2) return false;
    int isLeft = ccw(P[0], P[1], P[2]);
    for (int i = 1; i < sz; i++)
        if (ccw(P[i], P[(i+1) % sz], P[(i+2) % sz]) * isLeft < 0)
            return false;
    return true;
}

// Inside polygon: O(N). Works with any polygon
// Does not work when point is on edge. Should check separately
bool in_polygon(const Polygon &P, Point pt) {
    if ((int)P.size() == 0) return false;
    double sum = 0;
    for (int i = 0; i < (int)P.size(); i++) {
        Point Pj = P[(i+1) % P.size()];
        if (ccw(pt, P[i], Pj) > 0)
            sum += angle(P[i], pt, Pj);
        else sum -= angle(P[i], pt, Pj);
    }
    return fabs(fabs(sum) - 2*M_PI) < EPS;
}

// Check point in convex polygon, O(logN)
// Source: http://codeforces.com/contest/166/submission/1392387
// On edge --> false
#define Det(a,b,c) ((double)(b.x-a.x)*(double)(c.y-a.y)-(double)(b.y-a.y)*(c.x-a.x))
bool in_convex(vector<Point>& l, Point p){
    int a = 1, b = l.size()-1, c;
    if (Det(l[0], l[a], l[b]) > 0) swap(a,b);
    // Allow on edge --> if (Det... > 0 || Det ... < 0)
    if (Det(l[0], l[a], p) >= 0 || Det(l[0], l[b], p) <= 0) return false;
    while(abs(a-b) > 1) {
        c = (a+b)/2;
        if (Det(l[0], l[c], p) > 0) b = c; else a = c;
    }
    // Alow on edge --> return Det... <= 0
    return Det(l[a], l[b], p) < 0;
```

```
}

// Cut a polygon with a line. Returns one half.
// To return the other half, reverse the direction of Line l (by negating l.a, l.b)
// The line must be formed using 2 points
Polygon polygon_cut(Polygon P, Line l) {
    Polygon Q;
    for(int i = 0; i < P.size(); ++i) {
        Point A = P[i], B = (i == P.size()-1) ? P[0] : P[i+1];
        if (ccw(l.A, l.B, A) != -1) Q.push_back(A);
        if (ccw(l.A, l.B, A)*ccw(l.A, l.B, B) < 0) {
            Point p; areIntersect(Line(A, B), l, p);
            Q.push_back(p);
        }
    }
    return Q;
}

// Find intersection of 2 polygons
// Helper method
bool intersect_lpt(Point a, Point b,
    Point c, Point d, Point &r) {
    double D =  (b - a) % (d - c);
    if (cmp(D, 0) == 0) return false;
    double t =  ((c - a) % (d - c)) / D;
    double s = -((a - c) % (b - a)) / D;
    r = a + (b - a) * t;
    return cmp(t, 0) > 0 && cmp(t, 1) < 0 && cmp(s, 0) > 0 && cmp(s, 1) < 0;
}
Polygon convex_intersect(Polygon P, Polygon Q) {
    const int n = P.size(), m = Q.size();
    int a = 0, b = 0, aa = 0, ba = 0;
    enum { Pin, Qin, Unknown } in = Unknown;
    Polygon R;
    do {
        int a1 = (a+n-1) % n, b1 = (b+m-1) % m;
        double C = (P[a] - P[a1]) % (Q[b] - Q[b1]);
        double A = (P[a1] - Q[b]) % (P[a] - Q[b]);
        double B = (Q[b1] - P[a]) % (Q[b] - P[a]);
        Point r;
        if (intersect_lpt(P[a1], P[a], Q[b1], Q[b], r)) {
            if (in == Unknown) aa = ba = 0;
            R.push_back( r );
            in = B > 0 ? Pin : A > 0 ? Qin : in;
        }
        if (C == 0 && B == 0 && A == 0) {
            if (in == Pin) { b = (b + 1) % m; ++ba; }
            else           { a = (a + 1) % m; ++aa; }
        } else if (C >= 0) {
            if (A > 0) { if (in == Pin) R.push_back(P[a]); a = (a+1)%n; ++aa; }
            else       { if (in == Qin) R.push_back(Q[b]); b = (b+1)%m; ++ba; }
        } else {
            if (B > 0) { if (in == Qin) R.push_back(Q[b]); b = (b+1)%m; ++ba; }
            else       { if (in == Pin) R.push_back(P[a]); a = (a+1)%n; ++aa; }
        }
    } while ( (aa < n || ba < m) && aa < 2*n && ba < 2*m );
    if (in == Unknown) {
        if (in_convex(Q, P[0])) return P;
        if (in_convex(P, Q[0])) return Q;
    }
    return R;
}

// Find the diameter of polygon.
// Rotating callipers
double convex_diameter(Polygon pt) {
    const int n = pt.size();
    int is = 0, js = 0;
    for (int i = 1; i < n; ++i) {
        if (pt[i].y > pt[is].y) is = i;
        if (pt[i].y < pt[js].y) js = i;
    }
    double maxd = (pt[is]-pt[js]).norm();
    int i, maxi, j, maxj;
    i = maxi = is;
    j = maxj = js;
    do {
        int jj = j+1; if (jj == n) jj = 0;
        if ((pt[i] - pt[jj]).norm() > (pt[i] - pt[j]).norm()) j = (j+1) % n;
        else i = (i+1) % n;
        if ((pt[i]-pt[j]).norm() > maxd) {
            maxd = (pt[i]-pt[j]).norm();
            maxi = i; maxj = j;
        }
    } while (i != is || j != js);
    return maxd; /* farthest pair is (maxi, maxj). */
}

// Closest pair
// Source: e-maxx.ru
#define upd_ans(x, y) {}
#define MAXN 100
double mindist = 1e20; // will be the result
void rec(int l, int r, Point a[]) {
    if (r - l <= 3) {
        for (int i=l; i<=r; ++i)
            for (int j=i+1; j<=r; ++j)
                upd_ans(a[i], a[j]);
        sort(a+l, a+r+1); // compare by y
        return;
    }

    int m = (l + r) >> 1;
    int midx = a[m].x;
    rec(l, m, a), rec(m+1, r, a);
    static Point t[MAXN];
    merge(a+l, a+m+1, a+m+1, a+r+1, t); // compare by y
    copy(t, t+r-l+1, a+l);

    int tsz = 0;
    for (int i=l; i<=r; ++i)
        if (fabs(a[i].x - midx) < mindist) {
            for (int j=tsz-1; j>=0 && a[i].y - t[j].y < mindist; --j)
                upd_ans(a[i], t[j]);
            t[tsz++] = a[i];
        }
}

// Pick theorem
// Given non-intersecting polygon.
// S = area
// I = number of integer points strictly Inside
// B = number of points on sides of polygon
// S = I + B/2 - 1
```

5.5 Smallest Enclosing Circle

```
// Smallest enclosing circle:
// Given N points. Find the smallest circle enclosing these points.
// Amortized complexity: O(N)

struct SmallestEnclosingCircle {
    Circle getCircle(vector<Point> points) {
        assert(!points.empty());

        random_shuffle(points.begin(), points.end());
        Circle c(points[0], 0);
        int n = points.size();

        for (int i = 1; i < n; i++)
            if ((points[i] - c).len() > c.r + EPS)
            {
                c = Circle(points[i], 0);
                for (int j = 0; j < i; j++)
                    if ((points[j] - c).len() > c.r + EPS)
                    {
                        c = Circle((points[i] + points[j]) / 2, (points[i] - points[j]).len() / 2);
                        for (int k = 0; k < j; k++)
                            if ((points[k] - c).len() > c.r + EPS)
                                c = getCircumcircle(points[i], points[j], points[k]);
                    }
            }

        return c;
    }

    Circle getCircumcircle(Point a, Point b, Point c) {
        double d = 2.0 * (a.x * (b.y - c.y) + b.x * (c.y - a.y) + c.x * (a.y - b.y));
        assert(fabs(d) > EPS);
        double x = (a.norm() * (b.y - c.y) + b.norm() * (c.y - a.y) + c.norm() * (a.y - b.y)) / d;
        double y = (a.norm() * (c.x - b.x) + b.norm() * (a.x - c.x) + c.norm() * (b.x - a.x)) / d;
        Point p(x, y);
        return Circle(p, (p - a).len());
    }
};
```

5.6 Geometry Complex

```
// add : a + b
// scalar multi : r * a
// dot product : (conj(a) * b).x
// cross product : (conj(a) * b).y
// squared distance : norm(a - b)
// euclidean distance : abs(a - b)
```

```
// angle of elevation : arg(b - a)
// slope of line (a, b) : tan(arg(b - a))
// polar to cartesian : polar(r, theta)
// rotation about the origin : a * polar(1.0, theta)
// rotation about pivot p : (a - p) * polar(1.0, theta) + p
// angle ABC: abs(remainder(arg(a - b) - arg(c - b), 2.0 * M_PI)) distance: [-PI, PI]
// project p onto vector v : v * dot(p, v) / norm(v)
// project p onto line(a, b) : a + (b - a) * dot(p - a, b - a) / norm(b - a)
// reflect p across line (a, b) : a + conj((p - a) / (b - a)) * (b - a)
// relect p across line (a, b) : a + conj((p - a) / (b - a)) * (b - a)
// intersection of line(a, b) and (p, q) :
point intersection(point a, point b, point c) {
  double c1 = cross(p - a, b - a), c2 = cross(q - a, b - a);
  return (c1 * q - c2 * p) / (c1 - c2); //undefined if parallel
}
// read:
template<class T>
istream &operator>>(istream &is, complex<T> &p) {
  T value;
  is >> value;
  p.real(value);
  is >> value;
  p.imag(value);
  return is;
}
```

5.7 Latitude Longitude

```
/*
Converts from rectangular coordinates to latitude/longitude and vice
versa. Uses degrees (not radians).
*/

#include <iostream>
#include <cmath>

using namespace std;

struct ll
{
  double r, lat, lon;
};

struct rect
{
  double x, y, z;
};

ll convert(rect& P)
{
  ll Q;
  Q.r = sqrt(P.x*P.x+P.y*P.y+P.z*P.z);
  Q.lat = 180/M_PI*asin(P.z/Q.r);
  Q.lon = 180/M_PI*acos(P.x/sqrt(P.x*P.x+P.y*P.y));

  return Q;
}

rect convert(ll& Q)
{
  rect P;
  P.x = Q.r*cos(Q.lon*M_PI/180)*cos(Q.lat*M_PI/180);
  P.y = Q.r*sin(Q.lon*M_PI/180)*cos(Q.lat*M_PI/180);
  P.z = Q.r*sin(Q.lat*M_PI/180);

  return P;
}

int main()
{
  rect A;
  ll B;

  A.x = -1.0; A.y = 2.0; A.z = -3.0;

  B = convert(A);
  cout << B.r << " " << B.lat << " " << B.lon << endl;

  A = convert(B);
  cout << A.x << " " << A.y << " " << A.z << endl;
}
```

5.8 Delaunay

```
#include <iostream>
#include <cstdio>
#include <algorithm>
#include <vector>
#include <complex>
#include <list>
#include <array>
#include <set>
using namespace std;

typedef long double ld;
typedef complex<ld> Pnt;
typedef array<int, 2> Edge;
typedef array<int, 3> Tri;
typedef pair<Pnt, ld> Circ;
const ld oo = 1e9;
const ld eps = 1e-6;
int n;

Circ compute_circumcircle(Pnt a, Pnt b, Pnt c) {
  b -= a;
  c -= a;
  auto center = (b*norm(c)-c*norm(b)) / (b*conj(c)-c*conj(b));
  return {center+a, norm(center)};
}

vector<Tri> delaunay(vector<Pnt> pnts) {
  pnts.push_back({-oo, -oo});
  pnts.push_back({oo, -oo});
  pnts.push_back({oo, oo});
  pnts.push_back({-oo, oo});
  list<pair<Tri, Circ>> tcs;
  tcs.push_back({{n, n+1, n+2}, {0, 2*oo*oo}});
  tcs.push_back({{n+2, n+3, n}, {0, 2*oo*oo}});
  for (int i = 0; i < n; i++) {
    set<Edge> edges;
    for (auto it = begin(tcs); it != end(tcs); ) {
      auto circ = it->second;
      ld d = norm(pnts[i] - circ.first) - circ.second;
      if (d < -eps) {
        auto tri = it->first;
        auto j = *prev(end(tri));
        for (int k: tri) {
          auto ite = edges.find({k, j});
          if (ite == end(edges)) {
            edges.insert({j, k});
          } else {
            edges.erase(ite);
          }
          j = k;
        }
        it = tcs.erase(it);
      } else {
        it++;
      }
    }
    int j0 = (*begin(edges))[0];
    for (int j = j0; ; ) {
      int k = (*edges.lower_bound({j, 0}))[1];
      Circ circ = compute_circumcircle(pnts[i], pnts[j], pnts[k]);
      tcs.push_back({{i, j, k}, circ});
      j = k;
      if (j == j0) break;
    }
  }
  vector<Tri> tris;
  for (auto& tc: tcs) {
    auto& tri = tc.first;
    bool flag = false;
    for (int j: tri) {
      flag = flag || j >= n;
    }
    if (flag) continue;
    tris.push_back(tri);
  }
  return tris;
}
```

6 Miscellaneaous

6.1 Calendar Java

```
/*
Constructors:
GregorianCalendar()
```

```
GregorianCalendar(int year, int month, int dayOfMonth)
GregorianCalendar(int year, int month, int dayOfMonth, int hour, int minute)
Methods:
add(int field, int amount)
get(int field)
set(int field, int value)
Fields:
GregorianCalendar.YEAR
GregorianCalendar.MONTH
GregorianCalendar.WEEK_OF_YEAR
GregorianCalendar.DAY_OF_MONTH
GregorianCalendar.DAY_OF_WEEK
GregorianCalendar.DATE
*/
```

6.2 Dynamic Convex Hull DP (Max)

```
const ll is_query = -(1LL<<62);
struct Line {
    ll m, b;
    mutable function<const Line*()> succ;
    bool operator<(const Line& rhs) const {
        if (rhs.b != is_query) return m < rhs.m;
        const Line* s = succ();
        if (!s) return 0;
        ll x = rhs.m;
        return b - s->b < (s->m - m) * x;
    }
};
struct HullDynamic : public multiset<Line> { // will maintain upper hull for maximum
    bool bad(iterator y) {
        auto z = next(y);
        if (y == begin()) {
            if (z == end()) return 0;
            return y->m == z->m && y->b <= z->b;
        }
        auto x = prev(y);
        if (z == end()) return y->m == x->m && y->b <= x->b;
        return (x->b - y->b)*(z->m - y->m) >= (y->b - z->b)*(y->m - x->m);
    }
    void insert_line(ll m, ll b) {
        auto y = insert({ m, b });
        y->succ = [=] { return next(y) == end() ? 0 : &*next(y); };
        if (bad(y)) { erase(y); return; }
        while (next(y) != end() && bad(next(y))) erase(next(y));
        while (y != begin() && bad(prev(y))) erase(prev(y));
    }
    ll eval(ll x) {
        auto l = *lower_bound((Line) { x, is_query });
        return l.m * x + l.b;
    }
};
```

6.3 Floyd Cycle Finding

```
pii floyd(int x0) {
  int tortoise = f(x0), hare = f(f(x0));
  while (tortoise != hare) {
    tortoise = f(tortoise);
    hare = f(f(hare));
  }
  int mu = 0; hare = 0;
  while (tortoise != hare) {
    tortoise = f(tortoise);
    hare = f(hare);
    mu++;
  }
  int lambda = 1;
  hare = f(tortoise);
  while (tortoise != hare) {
    hare = f(hare);
    lambda++;
  }
  return mp(mu, lambda);
}
```

6.4 Josephus

```
/*
case k = 2 -> if n = 2^m + k and 0 <= k < 2^m, then f(n) =
2k+1
general case (0-based) -> f(n, k) = (f(n - 1, k) + k) mod n,
f(1, k) = 0
*/
```

6.5 Knight Shortest Path

```
// knight shortest path
int f(int x1, int y1, int x2, int y2)
{
 int dx=abs(x2-x1);
 int dy=abs(y2-y1);
 int lb=(dx+1)/2;
 lb>?=(dy+1)/2;
 lb>?=(dx+dy+2)/3;
 while ((lb%2)!=(dx+dy)%2) lb++;
 if (abs(dx)==1 && dy==0) return 3;
 if (abs(dy)==1 && dx==0) return 3;
 if (abs(dx)==2 && abs(dy)==2) return 4;
 return lb;
}
```

6.6 DP Knuth

```
// http://codeforces.com/blog/entry/8219
// Original Recurrence:
//   dp[i][j] = min(dp[i][k] + dp[k][j]) + C[i][j]   for k = i+1..j-1
// Necessary & Sufficient Conditions:
//   A[i][j-1] <= A[i][j] <= A[i+1][j]
//   with A[i][j] = smallest k that gives optimal answer
// Also applicable if the following conditions are met:
//   1. C[a][c] + C[b][d] <= C[a][d] + C[b][c]  (quadrangle inequality)
//   2. C[b][c] <= C[a][d]                      (monotonicity)
//   for all a <= b <= c <= d
// To use:
//   Calculate dp[i][i] and A[i][i]
//
//   FOR(len = 1..n-1)
//     FOR(i = 1..n-len) {
//       j = i + len
//       FOR(k = A[i][j-1]..A[i+1][j])
//         update(dp[i][j])
//     }

// OPTCUT
#include "../template.h"

const int MN = 2011;
int a[MN], dp[MN][MN], C[MN][MN], A[MN][MN];
int n;

int main() {
    while (cin >> n) {
        FOR(i,1,n) {
            cin >> a[i];
            a[i] += a[i-1];
        }
        FOR(i,1,n) FOR(j,i,n) C[i][j] = a[j] - a[i-1];

        FOR(i,1,n) dp[i][i] = 0, A[i][i] = i;

        FOR(len,1,n-1)
            FOR(i,1,n-len) {
                int j = i + len;
                dp[i][j] = 2000111000;
                FOR(k,A[i][j-1],A[i+1][j]) {
                    int cur = dp[i][k-1] + dp[k][j] + C[i][j];
                    if (cur < dp[i][j]) {
                        dp[i][j] = cur;
                        A[i][j] = k;
                    }
                }
            }
        cout << dp[1][n] << endl;
    }
}
```

6.7 vimrc

```
filetype plugin indent on
syntax on
set nocompatible              " be iMproved
set nu bs=2 sw=2 et ts=2 sts=2 rnu showcmd
set background=dark autoread autowrite mouse=a clipboard=unnamed nohls
setlocal cindent cino=j1,(0,ws,Ws
set timeoutlen=3000 ttimeoutlen=100
set fdm=marker
set commentstring=\ \"\ %s
let mapleader = ","
```

6.8 C++ Template

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

#define mt make_tuple
#define mp make_pair
#define pb push_back
#define fi first
#define se second
#define ALL(a) begin(a), end(a)
#define SZ(a) ((int)(a).size())

#ifdef __DEBUG
#define debug if (true)
#else
#define debug if (false)
#endif

typedef long long ll;
typedef pair<int, int> pii;
typedef vector<int> vi;

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

  return 0;
}
```

6.9 Java Template

```java
import java.util.*;
import java.io.*;

class Main {
  FastScanner in;
  PrintWriter out;

  public void solve() throws IOException {

  }

  public void run() {
    try {
      in = new FastScanner(System.in);
      out = new PrintWriter(System.out);

      solve();

      out.close();
    } catch (IOException e) {
      e.printStackTrace();
    }
  }

  class FastScanner {
    BufferedReader br;
    StringTokenizer st;

    FastScanner(InputStream is) {
      br = new BufferedReader(new InputStreamReader(is));
    }

    FastScanner(File f) {
      try {
        br = new BufferedReader(new FileReader(f));
      } catch (FileNotFoundException e) {
        e.printStackTrace();
      }
    }

    String next() {
      while (st == null || !st.hasMoreTokens()) {
        try {
          st = new StringTokenizer(br.readLine());
        } catch (IOException e) {
          e.printStackTrace();
        }
      }
      return st.nextToken();
    }

    int nextInt() {
      return Integer.parseInt(next());
    }
  }

  public static void main(String[] arg) {
    new Main().run();
  }
}
```

6.10 pika

```
#!/usr/bin/env bash
set -e
if [ "$2" == "-d" ]; then
        g++ -o "$1" "$1.cpp" -std=c++14 -D_GLIBCXX_DEBUG -D__DEBUG -g
else
        g++ -o "$1" "$1.cpp" -std=c++14 -O2 -Wall
fi
echo "COMPILE OK"
time "./$1" < "$1.in"
```

Combinatorics

Sums

$\sum_{k=0}^{n} k = n(n+1)/2$
$\sum_{k=0}^{n} k^2 = n(n+1)(2n+1)/6$
$\sum_{k=0}^{n} k^4 = (6n^5+15n^4+10n^3-n)/30$
$\sum_{k=0}^{n} x^k = (x^{n+1}-1)/(x-1)$
$1+x+x^2+\cdots = 1/(1-x)$
$\sum_{k=a}^{b} k = (a+b)(b-a+1)/2$
$\sum_{k=0}^{n} k^3 = n^2(n+1)^2/4$
$\sum_{k=0}^{n} k^5 = (2n^6+6n^5+5n^4-n^2)/12$
$\sum_{k=0}^{n} kx^k = (x-(n+1)x^{n+1}+nx^{n+2})/(x-1)^2$

Binomial coefficients

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	1												
1	1	1											
2	1	2	1										
3	1	3	3	1									
4	1	4	6	4	1								
5	1	5	10	10	5	1							
6	1	6	15	20	15	6	1						
7	1	7	21	35	35	21	7	1					
8	1	8	28	56	70	56	28	8	1				
9	1	9	36	84	126	126	84	36	9	1			
10	1	10	45	120	210	252	210	120	45	10	1		
11	1	11	55	165	330	462	462	330	165	55	11	1	
12	1	12	66	220	495	792	924	792	495	220	66	12	1
	0	1	2	3	4	5	6	7	8	9	10	11	12

$\binom{n}{k} = \frac{n!}{(n-k)!k!}$

$\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1}$

$\binom{n+1}{k} = \frac{n+1}{n-k+1}\binom{n}{k}$

$\binom{n}{k+1} = \frac{n-k}{k+1}\binom{n}{k}$

$\binom{n}{k} = \frac{n}{n-k}\binom{n-1}{k}$

$\binom{n}{k} = \frac{n-k+1}{k}\binom{n}{k-1}$

$12! \approx 2^{28.8}$
$20! \approx 2^{61.1}$

Number of ways to pick a multiset of size k from n elements: $\binom{n+k-1}{k}$
Number of n-tuples of non-negative integers with sum s: $\binom{s+n-1}{n-1}$, at most s: $\binom{s+n}{n}$
Number of n-tuples of positive integers with sum s: $\binom{s-1}{n-1}$
Number of lattice paths from $(0,0)$ to (a,b), restricted to east and north steps: $\binom{a+b}{a}$

Multinomial theorem. $(a_1+\cdots+a_k)^n = \sum \binom{n}{n_1,\ldots,n_k} a_1^{n_1}\ldots a_k^{n_k}$, where $n_i \geqslant 0$ and $\sum n_i = n$.
$\binom{n}{n_1,\ldots,n_k} = M(n_1,\ldots,n_k) = \frac{n!}{n_1!\ldots n_k!}$. $M(a,\ldots,b,c,\ldots) = M(a+\cdots+b,c,\ldots)M(a,\ldots,b)$

Catalan numbers. $C_n = \frac{1}{n+1}\binom{2n}{n}$. $C_0 = 1$, $C_n = \sum_{i=0}^{n-1} C_i C_{n-1-i}$. $C_{n+1} = C_n\frac{4n+2}{n+2}$.
$C_0, C_1, \ldots = 1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, 208012, 742900, \ldots$
C_n is the number of: properly nested sequences of n pairs of parentheses; rooted ordered binary trees with $n+1$ leaves; triangulations of a convex $(n+2)$-gon.

Derangements. Number of permutations of $n = 0, 1, 2, \ldots$ elements without fixed points is $1, 0, 1, 2, 9, 44, 265, 1854, 14833, \ldots$ Recurrence: $D_n = (n-1)(D_{n-1}+D_{n-2}) = nD_{n-1}+(-1)^n$. Corollary: number of permutations with exactly k fixed points is $\binom{n}{k}D_{n-k}$.

Stirling numbers of 1^{st} kind. $s_{n,k}$ is $(-1)^{n-k}$ times the number of permutations of n elements with exactly k permutation cycles. $|s_{n,k}| = |s_{n-1,k-1}| + (n-1)|s_{n-1,k}|$. $\sum_{k=0}^{n} s_{n,k}\, x^k = x^{\underline{n}}$

Stirling numbers of 2^{nd} kind. $S_{n,k}$ is the number of ways to partition a set of n elements into exactly k non-empty subsets. $S_{n,k} = S_{n-1,k-1} + kS_{n-1,k}$. $S_{n,1} = S_{n,n} = 1$. $x^n = \sum_{k=0}^{n} S_{n,k}\, x^{\underline{k}}$

Bell numbers. B_n is the number of partitions of n elements. $B_0, \ldots = 1, 1, 2, 5, 15, 52, 203, \ldots$
$B_{n+1} = \sum_{k=0}^{n}\binom{n}{k}B_k = \sum_{k=1}^{n} S_{n,k}$. Bell triangle: $B_r = a_{r,1} = a_{r-1,r-1}$, $a_{r,c} = a_{r-1,c-1} + a_{r,c-1}$.

Graph Theory

Euler's theorem. For any planar graph, $V - E + F = 1 + C$, where V is the number of graph's vertices, E is the number of edges, F is the number of faces in graph's planar drawing, and C is the number of connected components. Corollary: $V - E + F = 2$ for a 3D polyhedron.

Vertex covers and independent sets. Let M, C, I be a max matching, a min vertex cover, and a max independent set. Then $|M| \leqslant |C| = N - |I|$, with equality for bipartite graphs. Complement of an MVC is always a MIS, and vice versa. Given a bipartite graph with partitions (A, B), build a network: connect source to A, and B to sink with edges of capacities, equal to the corresponding nodes' weights, or 1 in the unweighted case. Set capacities of the original graph's edges to the infinity. Let (S, T) be a minimum s-t cut. Then a maximum(-weighted) independent set is $I = (A \cap S) \cup (B \cap T)$, and a minimum(-weighted) vertex cover is $C = (A \cap T) \cup (B \cap S)$.

Matrix-tree theorem. Let matrix $T = [t_{ij}]$, where t_{ij} is the number of multiedges between i and j, for $i \neq j$, and $t_{ii} = -\deg_i$. Number of spanning trees of a graph is equal to the determinant of a matrix obtained by deleting any k-th row and k-th column from T.

Euler tours. Euler tour in an undirected graph exists iff the graph is connected and each vertex has an even degree. Euler tour in a directed graph exists iff in-degree of each vertex equals its out-degree, and underlying undirected graph is connected. Construction:

```
doit(u):
  for each edge e = (u, v) in E, do: erase e, doit(v)
  prepend u to the list of vertices in the tour
```

Stable marriages problem. While there is a free man m: let w be the most-preferred woman to whom he has not yet proposed, and propose m to w. If w is free, or is engaged to someone whom she prefers less than m, match m with w, else deny proposal.

Stoer-Wagner's min-cut algorithm. Start from a set A containing an arbitrary vertex. While $A \neq V$, add to A the most tightly connected vertex ($z \notin A$ such that $\sum_{x \in A} w(x, z)$ is maximized.) Store cut-of-the-phase (the cut between the last added vertex and rest of the graph), and merge the two vertices added last. Repeat until the graph is contracted to a single vertex. Minimum cut is one of the cuts-of-the-phase.

Tarjan's offline LCA algorithm. (Based on DFS and union-find structure.)

```
DFS(x):
  ancestor[Find(x)] = x
  for all children y of x:
     DFS(y); Union(x, y); ancestor[Find(x)] = x
  seen[x] = true
  for all queries {x, y}:
    if seen[y] then output "LCA(x, y) is ancestor[Find(y)]"
```

Strongly-connected components. Kosaraju's algorithm.
1. Let G^T be a transpose G (graph with reversed edges.)
1. Call DFS(G^T) to compute finishing times $f[u]$ for each vertex u.
3. For each vertex u, in the order of decreasing $f[u]$, perform DFS(G, u).
4. Each tree in the 3rd step's DFS forest is a separate SCC.

2-SAT. Build an implication graph with 2 vertices for each variable – for the variable and its inverse; for each clause $x \vee y$ add edges $(\overline{x}, y)$ and $(\overline{y}, x)$. The formula is satisfiable iff x and $\overline{x}$ are in distinct

SCCs, for all x. To find a satisfiable assignment, consider the graph's SCCs in topological order from sinks to sources (i.e. Kosaraju's last step), assigning 'true' to all variables of the current SCC (if it hasn't been previously assigned 'false'), and 'false' to all inverses.

Randomized algorithm for non-bipartite matching. Let G be a simple undirected graph with even $|V(G)|$. Build a matrix A, which for each edge $(u,v) \in E(G)$ has $A_{i,j} = x_{i,j}$, $A_{j,i} = -x_{i,j}$, and is zero elsewhere. Tutte's theorem: G has a perfect matching iff $\det G$ (a multivariate polynomial) is identically zero. Testing the latter can be done by computing the determinant for a few random values of $x_{i,j}$'s over some field. (e.g. Z_p for a sufficiently large prime p)

Prufer code of a tree. Label vertices with integers 1 to n. Repeatedly remove the leaf with the smallest label, and output its only neighbor's label, until only one edge remains. The sequence has length $n-2$. Two isomorphic trees have the same sequence, and every sequence of integers from 1 and n corresponds to a tree. Corollary: the number of labelled trees with n vertices is n^{n-2}.

Erdos-Gallai theorem. A sequence of integers $\{d_1, d_2, \ldots, d_n\}$, with $n-1 \geqslant d_1 \geqslant d_2 \geqslant \cdots \geqslant d_n \geqslant 0$ is a degree sequence of some undirected simple graph iff $\sum d_i$ is even and $d_1 + \cdots + d_k \leqslant k(k-1) + \sum_{i=k+1}^{n} \min(k, d_i)$ for all $k = 1, 2, \ldots, n-1$.

Games

Grundy numbers. For a two-player, normal-play (last to move wins) game on a graph (V, E): $G(x) = \text{mex}(\{G(y) : (x,y) \in E\})$, where $\text{mex}(S) = \min\{n \geqslant 0 : n \notin S\}$. x is losing iff $G(x) = 0$.

Sums of games.

- *Player chooses a game and makes a move in it.* Grundy number of a position is xor of grundy numbers of positions in summed games.
- *Player chooses a non-empty subset of games (possibly, all) and makes moves in all of them.* A position is losing iff each game is in a losing position.
- *Player chooses a proper subset of games (not empty and not all), and makes moves in all chosen ones.* A position is losing iff grundy numbers of all games are equal.
- *Player must move in all games, and loses if can't move in some game.* A position is losing if any of the games is in a losing position.

Misère Nim. A position with pile sizes $a_1, a_2, \ldots, a_n \geqslant 1$, not all equal to 1, is losing iff $a_1 \oplus a_2 \oplus \cdots \oplus a_n = 0$ (like in normal nim.) A position with n piles of size 1 is losing iff n is *odd*.

Geometry

Pick's theorem. $I = A - B/2 + 1$, where A is the area of a lattice polygon, I is number of lattice points inside it, and B is number of lattice points on the boundary. Number of lattice points minus one on a line segment from $(0,0)$ and (x,y) is $\gcd(x,y)$.

$a \cdot b = a_x b_x + a_y b_y = |a| \cdot |b| \cdot \cos(\theta)$
$a \times b = a_x b_y - a_y b_x = |a| \cdot |b| \cdot \sin(\theta)$
3D: $a \times b = (a_y b_z - a_z b_y,\ a_z b_x - a_x b_z,\ a_x b_y - a_y b_x)$

Line $ax + by = c$ through $A(x_1, y_1)$ and $B(x_2, y_2)$: $a = y_1 - y_2$, $b = x_2 - x_1$, $c = ax_1 + by_1$.
Half-plane to the left of the directed segment AB: $ax + by \geqslant c$.
Normal vector: (a, b). Direction vector: $(b, -a)$. Perpendicular line: $-bx + ay = d$.
Point of intersection of $a_1 x + b_1 y = c_1$ and $a_2 x + b_2 y = c_2$ is $\frac{1}{a_1 b_2 - a_2 b_1}(c_1 b_2 - c_2 b_1, a_1 c_2 - a_2 c_1)$.
Distance from line $ax + by + c = 0$ to point (x_0, y_0) is $|ax_0 + by_0 + c| / \sqrt{a^2 + b^2}$.
Distance from line AB to P (for any dimension): $\frac{|(A-P) \times (B-P)|}{|A-B|}$.
Point-line segment distance:

```
if (dot(B-A, P-A) < 0) return dist(A,P);
if (dot(A-B, P-B) < 0) return dist(B,P);
return fabs(cross(P,A,B) / dist(A,B));
```

Projection of point C onto line AB is $\frac{AB \cdot AC}{AB \cdot AB} AB$.
Projection of (x_0, y_0) onto line $ax + by = c$ is $(x_0, y_0) + \frac{1}{a^2+b^2}(ad, bd)$, where $d = c - ax_0 - by_0$.
Projection of the origin is $\frac{1}{a^2+b^2}(ac, bc)$.

Segment-segment intersection. Two line segments intersect if one of them contains an endpoint of the other segment, or each segment straddles the line, containing the other segment (AB straddles line l if A and B are on the opposite sides of l.)

Circle-circle and circle-line intersection.

```
a = x2 - x1;   b = y2 - y1;   c = [(r1^2 - x1^2 - y1^2) - (r2^2 - x2^2 - y2^2)] / 2;
d = sqrt(a^2 + b^2);
if not |r1 - r2| <= d <= |r1 + r2|, return "no solution"
if d == 0, circles are concentric, a special case
// Now intersecting circle (x1,y1,r1) with line ax+by=c
Normalize line: a /= d; b /= d; c /= d;     // d=sqrt(a^2+b^2)
e = c - a*x1 - b*y1;
h = sqrt(r1^2 - e^2);                       // check if r1<e for circle-line test
return (x1, y1) + (a*e, b*e) +/- h*(-b, a);
```

Circle from 3 points (circumcircle). Intersect two perpendicular bisectors. Line perpendicular to $ax + by = c$ has the form $-bx + ay = d$. Find d by substituting midpoint's coordinates.

Angular bisector of angle ABC is line BD, where $D = \frac{BA}{|BA|} + \frac{BC}{|BC|}$.
Center of incircle of triangle ABC is at the intersection of angular bisectors, and is $\frac{a}{a+b+c}A + \frac{b}{a+b+c}B + \frac{c}{a+b+c}C$, where a, b, c are lengths of sides, opposite to vertices A, B, C. Radius $= \frac{2S}{a+b+c}$.

Counter-clockwise rotation around the origin. $(x, y) \mapsto (x\cos\phi - y\sin\phi, x\sin\phi + y\cos\phi)$.
90-degrees counter-clockwise rotation: $(x, y) \mapsto (-y, x)$. Clockwise: $(x, y) \mapsto (y, -x)$.

3D rotation by ccw angle ϕ around axis $\mathbf{n}$: $\mathbf{r}' = \mathbf{r}\cos\phi + \mathbf{n}(\mathbf{n} \cdot \mathbf{r})(1 - \cos\phi) + (\mathbf{n} \times \mathbf{r})\sin\phi$

Plane equation from 3 points. $N \cdot (x, y, z) = N \cdot A$, where N is normal: $N = (B - A) \times (C - A)$.

3D figures

Sphere	Volume $V = \frac{4}{3}\pi r^3$, surface area $S = 4\pi r^2$
	$x = \rho\sin\theta\cos\phi,\ y = \rho\sin\theta\sin\phi,\ z = \rho\cos\theta,\ \phi \in [-\pi, \pi],\ \theta \in [0, \pi]$
Spherical section	Volume $V = \pi h^2(r - h/3)$, surface area $S = 2\pi r h$
Pyramid	Volume $V = \frac{1}{3} h S_{base}$
Cone	Volume $V = \frac{1}{3}\pi r^2 h$, lateral surface area $S = \pi r\sqrt{r^2 + h^2}$

Area of a simple polygon. $\frac{1}{2}\sum_{i=0}^{n-1}(x_i y_{i+1} - x_{i+1} y_i)$, where $x_n = x_0, y_n = y_0$.
Area is negative if the boundary is oriented clockwise.

Bernoulli numbers. $\sum_{k=0}^{m-1} k^n = \frac{1}{n+1}\sum_{k=0}^{n}\binom{n+1}{k}B_k m^{n+1-k}$.
$\sum_{j=0}^{m}\binom{m+1}{j}B_j = 0$. $B_0 = 1$, $B_1 = -\frac{1}{2}$. $B_n = 0$, for all odd $n \neq 1$.

Eulerian numbers. $E(n,k)$ is the number of permutations with exactly k descents ($i : \pi_i < \pi_{i+1}$) / ascents ($\pi_i > \pi_{i+1}$) / excedances ($\pi_i > i$) / $k+1$ weak excedances ($\pi_i \geqslant i$).
Formula: $E(n,k) = (k+1)E(n-1,k) + (n-k)E(n-1,k-1)$. $x^n = \sum_{k=0}^{n-1} E(n,k)\binom{x+k}{n}$.

Burnside's lemma. The number of orbits under group G's action on set X:
$|X/G| = \frac{1}{|G|}\sum_{g\in G}|X_g|$, where $X_g = \{x \in X : g(x) = x\}$. ("Average number of fixed points.")
Let $w(x)$ be weight of x's orbit. Sum of all orbits' weights: $\sum_{o\in X/G} w(o) = \frac{1}{|G|}\sum_{g\in G}\sum_{x\in X_g} w(x)$.

Number Theory

Linear diophantine equation. $ax + by = c$. Let $d = \gcd(a,b)$. A solution exists iff $d|c$. If (x_0, y_0) is any solution, then all solutions are given by $(x,y) = (x_0 + \frac{b}{d}t, y_0 - \frac{a}{d}t)$, $t \in \mathbb{Z}$. To find some solution (x_0, y_0), use extended GCD to solve $ax_0 + by_0 = d = \gcd(a,b)$, and multiply its solutions by $\frac{c}{d}$.

Linear diophantine equation in n variablesss: $a_1x_1 + \cdots + a_nx_n = c$ has solutions iff $\gcd(a_1,\ldots,a_n)|c$. To find some solution, let $b = \gcd(a_2,\ldots,a_n)$, solve $a_1x_1 + by = c$, and iterate with $a_2x_2 + \cdots = y$.

Extended GCD

```
// Finds g = gcd(a,b) and x, y such that ax+by=g. Bounds: |x|<=b+1, |y|<=a+1.
void gcdext(int &g, int &x, int &y, int a, int b)
{ if (b == 0) { g = a; x = 1; y = 0; }
  else        { gcdext(g, y, x, b, a % b); y = y - (a / b) * x; } }
```

Multiplicative inverse of a modulo m: x in $ax + my = 1$, or $a^{\phi(m)-1} \pmod m$.

Chinese Remainder Theorem. System $x \equiv a_i \pmod{m_i}$ for $i = 1,\ldots,n$, with pairwise relatively-prime m_i has a unique solution modulo $M = m_1m_2\ldots m_n$: $x = a_1b_1\frac{M}{m_1} + \cdots + a_nb_n\frac{M}{m_n} \pmod M$, where b_i is modular inverse of $\frac{M}{m_i}$ modulo m_i.

System $x \equiv a \pmod m$, $x \equiv b \pmod n$ has solutions iff $a \equiv b \pmod g$, where $g = \gcd(m,n)$. The solution is unique modulo $L = \frac{mn}{g}$, and equals: $x \equiv a + T(b-a)m/g \equiv b + S(a-b)n/g \pmod L$, where S and T are integer solutions of $mT + nS = \gcd(m,n)$.

Prime-counting function. $\pi(n) = |\{p \leqslant n : p \text{ is prime}\}|$. $n/\ln(n) < \pi(n) < 1.3n/\ln(n)$. $\pi(1000) = 168$, $\pi(10^6) = 78498$, $\pi(10^9) = 50\ 847\ 534$. n-th prime $\approx n\ln n$.

Miller-Rabin's primality test. Given $n = 2^r s + 1$ with odd s, and a random integer $1 < a < n$. If $a^s \equiv 1 \pmod n$ or $a^{2^j s} \equiv -1 \pmod n$ for some $0 \leqslant j \leqslant r-1$, then n is a probable prime. With bases 2, 7 and 61, the test indentifies all composites below 2^{32}. Probability of failure for a random a is at most $1/4$.

Pollard-ρ. Choose random x_1, and let $x_{i+1} = x_i^2 - 1 \pmod n$. Test $\gcd(n, x_{2^k+i} - x_{2^k})$ as possible n's factors for $k = 0, 1, \ldots$ Expected time to find a factor: $O(\sqrt{m})$, where m is smallest prime power in n's factorization. That's $O(n^{1/4})$ if you check $n = p^k$ as a special case before factorization.

Fermat primes. A Fermat prime is a prime of form $2^{2^n} + 1$. The only known Fermat primes are 3, 5, 17, 257, 65537. A number of form $2^n + 1$ is prime only if it is a Fermat prime.

Perfect numbers. $n > 1$ is called perfect if it equals sum of its proper divisors and 1. Even n is perfect iff $n = 2^{p-1}(2^p - 1)$ and $2^p - 1$ is prime (Mersenne's). No odd perfect numbers are yet found.

Carmichael numbers. A positive composite n is a Carmichael number ($a^{n-1} \equiv 1 \pmod{n}$ for all a: $\gcd(a,n) = 1$), iff n is square-free, and for all prime divisors p of n, $p-1$ divides $n-1$.

Number/sum of divisors. $\tau(p_1^{a_1} \dots p_k^{a_k}) = \prod_{j=1}^{k}(a_j+1)$. $\sigma(p_1^{a_1} \dots p_k^{a_k}) = \prod_{j=1}^{k} \frac{p_j^{a_j+1}-1}{p_j-1}$.

Euler's phi function. $\phi(n) = |\{m \in \mathbb{N}, m \leqslant n, \gcd(m,n) = 1\}|$.
$\phi(mn) = \frac{\phi(m)\phi(n)\gcd(m,n)}{\phi(\gcd(m,n))}$. $\phi(p^a) = p^{a-1}(p-1)$. $\sum_{d|n}\phi(d) = \sum_{d|n}\phi(\frac{n}{d}) = n$.

Euler's theorem. $a^{\phi(n)} \equiv 1 \pmod{n}$, if $\gcd(a,n) = 1$.
Wilson's theorem. p is prime iff $(p-1)! \equiv -1 \pmod{p}$.

Mobius function. $\mu(1) = 1$. $\mu(n) = 0$, if n is not squarefree. $\mu(n) = (-1)^s$, if n is the product of s distinct primes. Let f, F be functions on positive integers. If for all $n \in N$, $F(n) = \sum_{d|n} f(d)$, then $f(n) = \sum_{d|n}\mu(d)F(\frac{n}{d})$, and vice versa. $\phi(n) = \sum_{d|n}\mu(d)\frac{n}{d}$. $\sum_{d|n}\mu(d) = 1$.
If f is multiplicative, then $\sum_{d|n}\mu(d)f(d) = \prod_{p|n}(1-f(p))$, $\sum_{d|n}\mu(d)^2 f(d) = \prod_{p|n}(1+f(p))$.

Legendre symbol. If p is an odd prime, $a \in \mathbb{Z}$, then $\left(\frac{a}{p}\right)$ equals 0, if $p|a$; 1 if a is a quadratic residue modulo p; and -1 otherwise. Euler's criterion: $\left(\frac{a}{p}\right) = a^{\left(\frac{p-1}{2}\right)} \pmod{p}$.

Jacobi symbol. If $n = p_1^{a_1} \cdots p_k^{a_k}$ is odd, then $\left(\frac{a}{n}\right) = \prod_{i=1}^{k}\left(\frac{a}{p_i}\right)^{k_i}$.

Primitive roots. If the order of g modulo m ($\min n > 0$: $g^n \equiv 1 \pmod{m}$) is $\phi(m)$, then g is called a primitive root. If Z_m has a primitive root, then it has $\phi(\phi(m))$ distinct primitive roots. Z_m has a primitive root iff m is one of 2, 4, p^k, $2p^k$, where p is an odd prime. If Z_m has a primitive root g, then for all a coprime to m, there exists unique integer $i = \mathrm{ind}_g(a)$ modulo $\phi(m)$, such that $g^i \equiv a \pmod{m}$. $\mathrm{ind}_g(a)$ has logarithm-like properties: $\mathrm{ind}(1) = 0$, $\mathrm{ind}(ab) = \mathrm{ind}(a) + \mathrm{ind}(b)$.

If p is prime and a is not divisible by p, then congruence $x^n \equiv a \pmod{p}$ has $\gcd(n, p-1)$ solutions if $a^{(p-1)/\gcd(n,p-1)} \equiv 1 \pmod{p}$, and no solutions otherwise. (Proof sketch: let g be a primitive root, and $g^i \equiv a \pmod{p}$, $g^u \equiv x \pmod{p}$. $x^n \equiv a \pmod{p}$ iff $g^{nu} \equiv g^i \pmod{p}$ iff $nu \equiv i \pmod{p}$.)

Discrete logarithm problem. Find x from $a^x \equiv b \pmod{m}$. Can be solved in $O(\sqrt{m})$ time and space with a meet-in-the-middle trick. Let $n = \lceil\sqrt{m}\rceil$, and $x = ny - z$. Equation becomes $a^{ny} \equiv ba^z \pmod{m}$. Precompute all values that the RHS can take for $z = 0, 1, \dots, n-1$, and brute force y on the LHS, each time checking whether there's a corresponding value for RHS.

Pythagorean triples. Integer solutions of $x^2 + y^2 = z^2$ All relatively prime triples are given by: $x = 2mn, y = m^2 - n^2, z = m^2 + n^2$ where $m > n, \gcd(m,n) = 1$ and $m \not\equiv n \pmod{2}$. All other triples are multiples of these. Equation $x^2 + y^2 = 2z^2$ is equivalent to $(\frac{x+y}{2})^2 + (\frac{x-y}{2})^2 = z^2$.

Postage stamps/McNuggets problem. Let a, b be relatively-prime integers. There are exactly $\frac{1}{2}(a-1)(b-1)$ numbers *not* of form $ax+by$ ($x, y \geqslant 0$), and the largest is $(a-1)(b-1)-1 = ab-a-b$.

Fermat's two-squares theorem. Odd prime p can be represented as a sum of two squares iff $p \equiv 1 \pmod{4}$. A product of two sums of two squares is a sum of two squares. Thus, n is a sum of two squares iff every prime of form $p = 4k+3$ occurs an even number of times in n's factorization.

RSA. Let p and q be random distinct large primes, $n = pq$. Choose a small odd integer e, relatively prime to $\phi(n) = (p-1)(q-1)$, and let $d = e^{-1} \pmod{\phi(n)}$. Pairs (e,n) and (d,n) are the public and secret keys, respectively. Encryption is done by raising a message $M \in Z_n$ to the power e or d, modulo n.