# .*VSTU.*

Team Reference Document

01/12/2018

Contents


1. Code Templates 2
1.1. Basic Configuration 2
1.2. Vector 3
1.3. FFT 4
1.4. Matrix 5
1.5. SegmTree 5
1.6. Aho 6
1.7. Suffix Automaton 7
1.8. Stoer Wagner 8
1.9. Flow 8
1.10. Prefix function 9
1.11. BPWS 9
1.12. Bridge search 11
1.13. Lca 11
1.14. 2-SAT 12
2. Misc 13
2.1. Debugging Tips 13
2.2. Solution Ideas 13
2.3. The Twelvefold Way 15
Practice Contest Checklist 16

# 1. Code Templates

## 1.1. Basic Configuration.

### 1.1.1. *.vimrc.*

```
set cin nu ts=2 sw=2 sts=2 mouse=a
syn on

function! Compile()
    :!g++ -std=gnu++11 -g % -o %<.exe
endfunction

function! Run()
    :!time ./%<.exe
endfunction

map <F4> :call Compile()<cr>
map <F5> :call Run()<cr>
map <C-A> ggVG"+y
```

### 1.1.2. *stress and template.*

```
// g++ -std=c++11 main.cpp -o main -D"_DEBUG_TEMICH_"
// -Wall -Wextra -pedantic -std=c++11
// -O2 -Wshadow -Wformat=2 -Wfloat-equal
// -Wconversion -Wlogical-op -Wshift-overflow=2
// -Wduplicated-cond -Wcast-qual -Wcast-align
// -D_GLIBCXX_DEBUG -D_GLIBCXX_DEBUG_PEDANTIC
// -D_FORTIFY_SOURCE=2 -fsanitize=address
// -fsanitize=undefined -fno-sanitize-recover
// -fstack-protector
#pragma GCC optimize("O3")
#pragma GCC target(
    "sse,sse2,sse3,ssse3,sse4,popcnt,abm,mmx")

#include <algorithm>
#include <cmath>
#include <functional>
#include <iostream>
#include <map>
#include <queue>
#include <set>
#include <sstream>
#include <string>
#include <vector>

using namespace std;

using LL = long long;
using pii = pair<int, int>;

#define X first
#define Y second

template<typename T>
ostream& operator<<(ostream& out, const vector<T>& v);

template<typename U, typename V>
ostream& operator<<(ostream& out, const map<U, V>& v);

template<typename U, typename V>
ostream& operator<<(ostream& out, const pair<U, V>& v);

template<typename U, typename V>
ostream& operator<<(ostream& out, const pair<U, V>& v) {
  return out << "(" << v.first << ", " << v.second << ")";
}

template<typename U, typename V>
ostream& operator<<(ostream& out, const map<U, V>& v) {
  out << "{";
  bool f = false;
  for (const auto& p : v) {
    out << (!f ? "" : ", ") << p;
    f = true;
  }
  return out << "}";
}

template<typename T>
ostream& operator<<(ostream& out, const vector<T>& v) {
  out << "{";
  for (int i = 0; i < int(v.size()); ++i)
    out << (i == 0 ? "" : ", ") << v[i];
  return out << "}";
}

void cerr_printer(bool start) {}
template<typename T, typename ... Args>
void cerr_printer(bool start, const T& x, const Args& ... args) {
  if (!start) cerr << ", ";
  cerr << x;
  cerr_printer(false, args...);
}

template<typename ... Args>
```

```cpp
void dbg(const char * name, int line, const Args& ... args) {
  cerr << "[" << line << "] (" << name << ") = (";
  cerr_printer(true, args...);
  cerr << ")" << endl;
}

#define DBG(...) { dbg(#__VA_ARGS__, __LINE__, __VA_ARGS__); }

struct Solver {
  void solve(istream& cin, ostream& cout) {
    int a, b;
    cin >> a >> b;
    cout << a + b << endl;
  }
};

struct Brute {
  void solve(istream& cin, ostream& cout) {
    int a, b;
    cin >> a >> b;
    while (b--) ++a;
    cout << a << endl;
  }
};

template <typename Solution>
struct SolutionStr {
  string solve(string input) {
    istringstream is(input);
    ostringstream os;
    Solution().solve(is, os);
    return os.str();
  }
};

string gen_input(int it) {
  (void)it;
  return "10 20";
}

void stress() {
  for (int it = 0; it < 1000; ++it) {
    auto input = gen_input(it);
    auto brute_out = SolutionStr<Solver>().solve(input);
    auto sol_out = SolutionStr<Brute>().solve(input);
    if (sol_out != brute_out) {
      cerr << "WA #" << it << endl;
      cerr << "input: " << endl;
      cerr << input << endl;
      cerr << "expected: " << brute_out << endl;
      cerr << "got: " << sol_out << endl;
      exit(1);
    }
  }

  cerr << "OK" << endl;
}

int main() {
  #ifdef _DEBUG_TEMICH_
  stress();
  #endif
  Solver().solve(cin, cout);
}
```

### 1.2. Vector.

```cpp
struct Vec {
  LL x, y;
  explicit Vec(LL x = 0 , LL y = 0) : x(x), y(y) {}
  Vec operator+(const Vec& o) const {
    return Vec(x + o.x, y + o.y); }
  Vec operator-(const Vec& o) const {
    return Vec(x - o.x, y - o.y); }
  Vec operator*(const LL p) const {
    return Vec(x * p, y * p); }
  double len() const { return sqrt(x * x + y * y); }
  LL cross(const Vec& o) const { return x * o.y - y * o.x; }
  LL dot(const Vec& o) const { return x * o.x + y * o.y; }
  static Vec read(istream& cin) {
    LL x, y;
    cin >> x >> y;
    return Vec(x, y);
  }
};

// CONVEX HULL: last point == first point
vector<Vec> convex_hull(vector<Vec> a) {
  int n = a.size(), k = 0;
  vector<Vec> p(n * 2);
  sort(a.begin(), a.end());

  for(int i = 0; i < n; p[k++] = a[i++])
    while(k > 1 && (p[k - 1] - p[k - 2])
        % (p[k - 1] - a[i]) >= 0) --k;
```

```
  for(int i = n - 2, w = k; i >= 0; p[k++] = a[i--])
    while(k > w && (p[k - 1] - p[k - 2])
        % (p[k - 1] - a[i]) >= 0) --k;
  p.resize(k);
  return p;
}
```

### 1.3. FFT.

```
struct Complex {
  long double re, im;
  explicit Complex(long double re = 0,
      long double im = 0) : re(re), im(im) {}
  Complex operator+(const Complex& o) const {
    return Complex(re + o.re, im + o.im); }
  Complex operator-(const Complex& o) const {
    return Complex(re - o.re, im - o.im); }
  Complex operator*(const Complex& o) const {
    return Complex(re * o.re - im * o.im, re * o.im + im * o.re); }
};

const int MAX_SHIFT = 22;
const int MAX_N     = 1 << MAX_SHIFT;

const double Pi = acos(-1);

Complex roots[MAX_N / 2];
int bit_reverse[MAX_N];

void prep() {
  bit_reverse[0] = 0;
  for (int i = 1; i < MAX_N; ++i)
    bit_reverse[i] = (bit_reverse[i >> 1]
        | ((i & 1) << MAX_SHIFT)) >> 1;

  for (int i = 0; i + i < MAX_N; ++i) {
    double angle = 2 * i * Pi / MAX_N;
    roots[i] = Complex(cos(angle), sin(angle));
  }
}

Complex arr[MAX_N];
void fft(int k) {
  assert(k <= MAX_SHIFT);

  const int n = 1 << k;
  for (int i = 0; i < n; ++i) {
    int rv = bit_reverse[i] >> (MAX_SHIFT - k);
    if (rv < i) swap(arr[i], arr[rv]);
  }

  for (int bs = 2; bs <= n; bs *= 2) {
    const int hbs = bs / 2;
    const int factor = (MAX_N / 2) / hbs;
    for (int i = 0; i < n; i += bs) {
      for (int j = 0; j < hbs; ++j) {
        auto a = arr[i + j];
        auto b = arr[i + j + hbs] * roots[factor * j];
        arr[i + j] = a + b;
        arr[i + j + hbs] = a - b;
      }
    }
  }
}

const int Base = 100;

void square(vector<int>& number) {
  int sz = number.size() * 2;
  int k = 1;
  {
    int rsz = 2;
    while (rsz < sz) {
      rsz *= 2;
      ++k;
    }

    sz = rsz;
  }

  assert(sz <= MAX_N);

  for (int i = 0; i < sz; ++i)
    arr[i] = Complex(i < number.size() ? number[i] : 0);

  fft(k);
  for (int i = 0; i < sz; ++i)
    arr[i] = arr[i] * arr[i];
  fft(k);
  reverse(arr + 1, arr + sz);

  number.resize(sz);
  int cr = 0;
  for (int i = 0; i < sz; ++i) {
    number[i] = cr + int(arr[i].re / sz + 0.5);
```

```
    cr = number[i] / Base;
    number[i] %= Base;
  }

  while (number.back() == 0) number.pop_back();
}
```

### 1.4. Matrix.

```
struct Matrix {
  ULL vals[N][N];
  Matrix() {
    for (int i = 0; i < N; ++i)
      fill(vals[i], vals[i] + N, 0);
  }

  ULL* operator[](const int idx) {
    return vals[idx];
  }

  const ULL* operator[](const int idx) const {
    return vals[idx];
  }

  static Matrix Ident() {
    Matrix res;
    for (int i = 0; i < N; ++i)
      res[i][i] = 1;

    return res;
  }

  Matrix operator*(const Matrix& o) const {
    Matrix res;

    for (int i = 0; i < N; ++i) {
      for (int j = 0; j < N; ++j) {
        for (int k = 0; k < N; ++k) {
          res[i][j] += vals[i][k] * o[k][j];
          if (k == 7)
            res[i][j] %= MOD;
        }
        res[i][j] %= MOD;
      }
    }

    return res;
  }
};
```

### 1.5. SegmTree.

```
class SegmTreeSum {
  vector<int> tree;
  int n;

  int get(int v, int l, int r, int L, int R) const {
    if (L > R) return 0;
    if (l == L && r == R) return tree[v];

    int mid = (l + r) / 2;

    int a = get(2 * v + 1, l, mid, L, min(R, mid));
    int b = get(2 * v + 2, mid + 1, r, max(L, mid + 1), R);

    return a + b;
  }

  void set(int v, int l, int r, int pos, int val) {
    if (l == r) {
      tree[pos] = val;
      return;
    }

    int mid = (l + r) / 2;

    if (pos <= mid) set(2 * v + 1, l, mid, pos, val);
    else set(2 * v + 2, mid + 1, r, pos, val);

    tree[v] = tree[2 * v + 1] + tree[2 * v + 2];
  }

public:
  void init(int n_) {
    n = n_;
    tree.assign(4 * n, 0);
  }

  int get(int l, int r) const {
    return get(0, 0, n - 1, l, r);
  }

  void set(int pos, int val) {
    set(0, 0, n - 1, pos, val);
  }
};
```

```
};

class SegmTreeMax {
  vector<Pair> tree;
  vector<int> psh;
  int n;

  void build(int v, int l, int r, const vector<int>& dp) {
    if (l == r) {
      tree[v] = Pair(dp[l], l);
      return;
    }

    int mid = (l + r) / 2;

    build(2 * v + 1, l, mid, dp);
    build(2 * v + 2, mid + 1, r, dp);

    tree[v] = max(tree[2 * v + 1], tree[2 * v + 2]);
  }

  void push(int v, int l, int r) {
    if (l != r) {
      psh[2 * v + 1] += psh[v];
      psh[2 * v + 2] += psh[v];
    }
    tree[v].X += psh[v];
    psh[v] = 0;
  }

  Pair getMax(int v, int l, int r, int L, int R) {
    push(v, l, r);
    if (L > R) return Pair(-INF, -INF);

    if (l == L && r == R)
      return tree[v];

    int mid = (l + r) / 2;

    Pair a = getMax(2 * v + 1, l, mid, L, min(R, mid));
    Pair b = getMax(2 * v + 2, mid + 1, r, max(L, mid + 1), R);

    return max(a, b);
  }

  void add(int v, int l, int r, int L, int R, int val) {
    push(v, l, r);

    if (L > R) return;
    if (l == L && r == R) {
      psh[v] += val;
      push(v, l, r);
      return;
    }

    int mid = (l + r) / 2;

    add(2 * v + 1, l, mid, L, min(R, mid), val);
    add(2 * v + 2, mid + 1, r, max(L, mid + 1), R, val);

    tree[v] = max(tree[2 * v + 1], tree[2 * v + 2]);
  }

public:
  void init(const vector<int>& dp) {
    n = dp.size();
    tree.resize(4 * n);
    psh.assign(4 * n, 0);

    build(0, 0, n - 1, dp);
  }

  Pair getMax(int l, int r) {
    return getMax(0, 0, n - 1, l, r);
  }

  void add(int l, int r, int val) {
    add(0, 0, n - 1, l, r, val);
  }
};
```

### 1.6. Aho.

```
struct Matcher {
  static const int LETTERS_COUNT = 'z' - 'a' + 1;
  struct Next {
    int nxt[LETTERS_COUNT];
    Next() { fill(nxt, nxt + LETTERS_COUNT, -1); }
    int& operator[](char c) { return nxt[c - 'a']; }
  };

  vector<Next> next;
  vector<int> link;
  vector<char> p_char;
  vector<int> p;
```

```
  vector<int> id;

  void build(const set<string>& strings) {
    int total_size = 0;
    for (const auto& s : strings)
      total_size += s.size();
    next.reserve(total_size);
    link.reserve(total_size);
    p_char.reserve(total_size);
    p.reserve(total_size);

    push();

    int _id = 0;
    for (const auto& s : strings) {
      add(s, _id);
      ++_id;
    }
  }

  void push() {
    next.push_back(Next());
    link.push_back(-1);
    p_char.push_back('#');
    p.push_back(-1);
    id.push_back(-1);
  }

  void add(const string& s, int _id) {
    int state = 0;

    for (char c : s) {
      int next_state = next[state][c];
      if (next_state == -1) {
        push();
        p_char.back() = c;
        p.back() = state;
        next_state = p.size() - 1;
        next[state][c] = next_state;
      }

      state = next_state;
    }

    id[state] = _id;
  }

  int get_next(int state, char c) {
    int x = _get_next(state, c);
    // cerr << "get next " << state << " " << c << " = " << x << endl;
    return x;
  }

  int _get_next(int state, char c) {
    if (next[state][c] == -1 && state == 0)
      return 0;
    if (next[state][c] == -1)
      next[state][c] = get_next(get_link(state), c);
    return next[state][c];
  }

  int get_link(int state) { int x = _get_link(state);
    // cerr << "get link " << state << " = " << x << endl;
    return x;
  }

  int _get_link(int state) {
    if (state == 0)
      return 0;
    if (p[state] == 0)
      return 0;

    int& l = link[state];
    if (l == -1)
      l = get_next(get_link(p[state]), p_char[state]);
    return l;
  }

  int get_id(int state) { return id[state]; }
};
```

### 1.7. Suffix Automaton.

```
struct State {
  map<char, int> nxt;
  int link;
  int len;
  bool added;
  int cnt;
};

State st[N];
int lst;
int sz;

```

```
void init() {
  lst = 0;
  sz = 1;
  st[0].link = -1;
  st[0].len = 1;
}

void ext(char c) {
  // cerr << "ext : " << c << endl;
  int cur = sz++;
  st[cur].len = st[lst].len + 1;

  int p;
  for (p = lst; p != -1 && !st[p].nxt.count(c); p = st[p].link)
    st[p].nxt[c] = cur;

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

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

      for (; p != -1 && st[p].nxt[c] == q; p = st[p].link)
        st[p].nxt[c] = clone;
    }
  }

  lst = cur;
  st[cur].cnt = st[st[cur].link].cnt;
}
```

### 1.8. Stoer Wagner.

```
const int MAXN = 500;
int n, g[MAXN][MAXN];
int best_cost = 1000000000;
vector<int> best_cut;

void mincut() {
  vector<int> v[MAXN];
  for (int i=0; i<n; ++i)
    v[i].assign (1, i);
  int w[MAXN];
  bool exist[MAXN], in_a[MAXN];
  memset (exist, true, sizeof exist);
  for (int ph=0; ph<n-1; ++ph) {
    memset (in_a, false, sizeof in_a);
    memset (w, 0, sizeof w);
    for (int it=0, prev; it<n-ph; ++it) {
      int sel = -1;
      for (int i=0; i<n; ++i)
        if (exist[i] && !in_a[i] && (sel == -1
              || w[i] > w[sel]))
          sel = i;
      if (it == n-ph-1) {
        if (w[sel] < best_cost)
          best_cost = w[sel],  best_cut = v[sel];
        v[prev].insert (v[prev].end(),
            v[sel].begin(), v[sel].end());
        for (int i=0; i<n; ++i)
          g[prev][i] = g[i][prev] += g[sel][i];
        exist[sel] = false;
      }
      else {
        in_a[sel] = true;
        for (int i=0; i<n; ++i)
          w[i] += g[sel][i];
        prev = sel;
      }
    }
  }
}
```

### 1.9. Flow.

```
struct Edge {
  int u, v, flow, cap;
  Edge() : u(0), v(0), flow(0), cap(0) {}
  Edge(int u, int v, int c) : u(u), v(v), flow(0), cap(c) {}
};

const int N = 666;
const int T = 1111;

const int MAXN = N + 2 * T + 100;


vector<int> g[500000];
```

```
vector<Edge> edges;

int flow, s, t;
int start[MAXN], used[MAXN], dist[MAXN];

bool bfs() {
  memset(start, 0, sizeof(start));
  memset(dist, -1, sizeof(dist));
  dist[s] = 0;

  queue<int> q;
  q.push(s);

  while (q.size()) {
    int u = q.front();
    q.pop();

    for (int id : g[u]) {
      Edge &e = edges[id];
      int v = e.v;

      if (dist[v] == -1 && e.flow < e.cap) {
        dist[v] = dist[u] + 1;
        q.push(v);
      }
    }
  }
  return dist[t] != -1;
}

int dfs(int u, int fl = -1) {
  if (fl == -1) memset(used, false, sizeof(used));
  used[u] = true;

  if (u == t) return fl;

  for (int &i = start[u]; i < g[u].size(); ++i) {
    int id = g[u][i];
    Edge &e = edges[id];
    int v = e.v;
    if (!used[v] && dist[v] == dist[u] + 1 && e.flow < e.cap) {
      int can = e.cap - e.flow;
      int df = dfs(v, fl == -1 ? can : min(fl, can));
      if (df > 0) {
        edges[id ^ 0].flow += df;
        edges[id ^ 1].flow -= df;
        return df;
      }
    }
  }
  return 0;
}

void add_edge(int u, int v, int c) {
  //cout << "add (" << u << " " << v << " " << c << ") " << endl;
  g[u].push_back(edges.size());
  edges.emplace_back(u, v, c);
  g[v].push_back(edges.size());
  edges.emplace_back(v, u, 0);
}

int calc(int ss, int tt) {
  //cout << "calc (" << ss << ", " << tt << ")" << endl;
  flow = 0, s = ss, t = tt;
  while (bfs()) {
    while (int add = dfs(ss)) {
      flow += add;
    }
  }
  return flow;
}
```

### 1.10. Prefix function.

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

### 1.11. BPWS.

```
const int trivial_limit = 50;
int p[1000];

int gcd (int a, int b) {
  return a ? gcd (b%a, a) : b;
}

```

```
int powmod (int a, int b, int m) {
  int res = 1;
  while (b)
    if (b & 1)
      res = (res * 1ll * a) % m,  --b;
    else
      a = (a * 1ll * a) % m,  b >>= 1;
  return res;
}

bool miller_rabin (int n) {
  int b = 2;
  for (int g; (g = gcd (n, b)) != 1; ++b)
    if (n > g)
      return false;
  int p=0, q=n-1;
  while ((q & 1) == 0)
    ++p,  q >>= 1;
  int rem = powmod (b, q, n);
  if (rem == 1 || rem == n-1)
    return true;
  for (int i=1; i<p; ++i) {
    rem = (rem * 1ll * rem) % n;
    if (rem == n-1)  return true;
  }
  return false;
}

int jacobi (int a, int b)
{
  if (a == 0)  return 0;
  if (a == 1)  return 1;
  if (a < 0)
    if ((b & 2) == 0)
      return jacobi (-a, b);
    else
      return - jacobi (-a, b);
  int a1=a,  e=0;
  while ((a1 & 1) == 0)
    a1 >>= 1,  ++e;
  int s;
  if ((e & 1) == 0 || (b & 7) == 1 || (b & 7) == 7)
    s = 1;
  else
    s = -1;
  if ((b & 3) == 3 && (a1 & 3) == 3)
    s = -s;
  if (a1 == 1)
    return s;
  return s * jacobi (b % a1, a1);
}

bool bpsw (int n) {
  if ((int)sqrt(n+0.0) *
      (int)sqrt(n+0.0) == n)  return false;
  int dd=5;
  for (;;) {
    int g = gcd (n, abs(dd));
    if (1<g && g<n)  return false;
    if (jacobi (dd, n) == -1)  break;
    dd = dd<0 ? -dd+2 : -dd-2;
  }
  int p=1,  q=(p*p-dd)/4;
  int d=n+1,  s=0;
  while ((d & 1) == 0)
    ++s,  d>>=1;
  long long u=1, v=p, u2m=1, v2m=p, qm=q, qm2=q*2, qkd=q;
  for (int mask=2; mask<=d; mask<<=1) {
    u2m = (u2m * v2m) % n;
    v2m = (v2m * v2m) % n;
    while (v2m < qm2)   v2m += n;
    v2m -= qm2;
    qm = (qm * qm) % n;
    qm2 = qm * 2;
    if (d & mask) {
      long long t1 = (u2m * v) % n,
           t2 = (v2m * u) % n,
        t3 = (v2m * v) % n,
        t4 = (((u2m * u) % n) * dd) % n;
      u = t1 + t2;
      if (u & 1)  u += n;
      u = (u >> 1) % n;
      v = t3 + t4;
      if (v & 1)  v += n;
      v = (v >> 1) % n;
      qkd = (qkd * qm) % n;
    }
  }
  if (u==0 || v==0)  return true;
  long long qkd2 = qkd*2;
  for (int r=1; r<s; ++r) {
    v = (v * v) % n - qkd2;
    if (v < 0)  v += n;
    if (v < 0)  v += n;
```

```
    if (v >= n)  v -= n;
    if (v >= n)  v -= n;
    if (v == 0)  return true;
    if (r < s-1) {
      qkd = (qkd * 1ll * qkd) % n;
      qkd2 = qkd * 2;
    }
  }
  return false;
}

bool prime (int n) {
  // Call for prime check
  for (int i=0; i<trivial_limit && p[i]<n; ++i)
    if (n % p[i] == 0)
      return false;
  if (p[trivial_limit-1]*p[trivial_limit-1] >= n)
    return true;
  if (!miller_rabin (n))
    return false;
  return bpsw (n);
}

void prime_init() {
  // Call before prime check
  for (int i=2, j=0; j<trivial_limit; ++i) {
    bool pr = true;
    for (int k=2; k*k<=i; ++k)
      if (i % k == 0)
        pr = false;
    if (pr)
      p[j++] = i;
  }
}
```

### 1.12. Bridge search.

```
const int MAXN = ...;
vector<int> g[MAXN];
bool used[MAXN];
int timer, tin[MAXN], fup[MAXN];

void dfs (int v, int p = -1) {
  used[v] = true;
  tin[v] = fup[v] = timer++;
  for (size_t i=0; i<g[v].size(); ++i) {
    int to = g[v][i];
    if (to == p)  continue;
    if (used[to])
      fup[v] = min (fup[v], tin[to]);
    else {
      dfs (to, v);
      fup[v] = min (fup[v], fup[to]);
      if (fup[to] > tin[v])
        IS_BRIDGE(v,to);
    }
  }
}

void find_bridges() {
  timer = 0;
  for (int i=0; i<n; ++i)
    used[i] = false;
  for (int i=0; i<n; ++i)
    if (!used[i])
      dfs (i);
}
```

### 1.13. Lca.

```
int n, l;
vector < vector<int> > g;
vector<int> tin, tout;
int timer;
vector < vector<int> > up;

void dfs (int v, int p = 0) {
  tin[v] = ++timer;
  up[v][0] = p;
  for (int i=1; i<=l; ++i)
    up[v][i] = up[up[v][i-1]][i-1];
  for (size_t i=0; i<g[v].size(); ++i) {
    int to = g[v][i];
    if (to != p)
      dfs (to, v);
  }
  tout[v] = ++timer;
}

bool upper (int a, int b) {
  return tin[a] <= tin[b] && tout[a] >= tout[b];
}

int lca (int a, int b) {
  if (upper (a, b))  return a;
  if (upper (b, a))  return b;
```

```
  for (int i=l; i>=0; --i)
    if (! upper (up[a][i], b))
      a = up[a][i];
  return up[a][0];
}

int main() {
  // read
  tin.resize (n),  tout.resize (n),  up.resize (n);
  l = 1;
  while ((1<<l) <= n)  ++l;
  for (int i=0; i<n; ++i)  up[i].resize (l+1);
  dfs (0);

  for (;;) {
    int a, b; // query
    int res = lca (a, b); // answer
  }

}
```

### 1.14. 2-SAT.

```
int n;
vector < vector<int> > g, gt;
vector<bool> used;
vector<int> order, comp;

void dfs1 (int v) {
  used[v] = true;
  for (size_t i=0; i<g[v].size(); ++i) {
    int to = g[v][i];
    if (!used[to])
      dfs1 (to);
  }
  order.push_back (v);
}

void dfs2 (int v, int cl) {
  comp[v] = cl;
  for (size_t i=0; i<gt[v].size(); ++i) {
    int to = gt[v][i];
    if (comp[to] == -1)
      dfs2 (to, cl);
  }
}

int main() {
      // read
  used.assign (n, false);
  for (int i=0; i<n; ++i)
    if (!used[i])
      dfs1 (i);

  comp.assign (n, -1);
  for (int i=0, j=0; i<n; ++i) {
    int v = order[n-i-1];
    if (comp[v] == -1)
      dfs2 (v, j++);
  }

  for (int i=0; i<n; ++i)
    if (comp[i] == comp[i^1]) {
      puts ("NO SOLUTION");
      return 0;
    }
  for (int i=0; i<n; ++i) {
    int ans = comp[i] > comp[i^1] ? i : i^1;
    printf ("%d ", ans);
  }

}
```

## 2. Misc

### 2.1. Debugging Tips.

- Stack overflow? Recursive DFS on tree that is actually a long path?
- Floating-point numbers
  - Getting `NaN`? Make sure `acos` etc. are not getting values out of their range (perhaps `1+eps`).
  - Rounding negative numbers?
  - Outputting in scientific notation?
- Wrong Answer?
  - Read the problem statement again!
  - Are multiple test cases being handled correctly? Try repeating the same test case many times.
  - Integer overflow?
  - Think very carefully about boundaries of all input parameters
  - Try out possible edge cases:
    - $n = 0, n = -1, n = 1, n = 2^{31} - 1$ or $n = -2^{31}$
    - List is empty, or contains a single element
    - $n$ is even, $n$ is odd
    - Graph is empty, or contains a single vertex
    - Graph is a multigraph (loops or multiple edges)
    - Polygon is concave or non-simple
  - Is initial condition wrong for small cases?
  - Are you sure the algorithm is correct?
  - Explain your solution to someone.
  - Are you using any functions that you don't completely understand? Maybe STL functions?
  - Maybe you (or someone else) should rewrite the solution?
  - Can the input line be empty?
- Run-Time Error?
  - Is it actually Memory Limit Exceeded?

### 2.2. Solution Ideas.

- Dynamic Programming
  - Parsing CFGs: CYK Algorithm
  - Drop a parameter, recover from others
  - Swap answer and a parameter
  - When grouping: try splitting in two
  - $2^k$ trick
  - When optimizing
    - Convex hull optimization
      - $\mathrm{dp}[i] = \min_{j<i}\{\mathrm{dp}[j] + b[j] \times a[i]\}$
      - $b[j] \geq b[j+1]$
      - optionally $a[i] \leq a[i+1]$
      - $O(n^2)$ to $O(n)$
    - Divide and conquer optimization
      - $\mathrm{dp}[i][j] = \min_{k<j}\{\mathrm{dp}[i-1][k] + C[k][j]\}$
      - $A[i][j] \leq A[i][j+1]$
      - $O(kn^2)$ to $O(kn \log n)$
      - sufficient: $C[a][c] + C[b][d] \leq C[a][d] + C[b][c]$, $a \leq b \leq c \leq d$ (QI)
    - Knuth optimization
      - $\mathrm{dp}[i][j] = \min_{i<k<j}\{\mathrm{dp}[i][k] + \mathrm{dp}[k][j] + C[i][j]\}$
      - $A[i][j-1] \leq A[i][j] \leq A[i+1][j]$
      - $O(n^3)$ to $O(n^2)$
      - sufficient: QI and $C[b][c] \leq C[a][d]$, $a \leq b \leq c \leq d$
- Greedy
- Randomized
- Optimizations
  - Use bitset (/64)
  - Switch order of loops (cache locality)
- Process queries offline
  - Mo's algorithm
- Square-root decomposition
- Precomputation
- Efficient simulation
  - Mo's algorithm
  - Sqrt decomposition
  - Store $2^k$ jump pointers
- Data structure techniques
  - Sqrt buckets
  - Store $2^k$ jump pointers
  - $2^k$ merging trick
- Counting
  - Inclusion-exclusion principle
  - Generating functions
- Graphs
  - Can we model the problem as a graph?
  - Can we use any properties of the graph?
  - Strongly connected components
  - Cycles (or odd cycles)
  - Bipartite (no odd cycles)
    - Bipartite matching
    - Hall's marriage theorem
    - Stable Marriage
  - Cut vertex/bridge
  - Biconnected components
  - Degrees of vertices (odd/even)
  - Trees
    - Heavy-light decomposition
    - Centroid decomposition
    - Least common ancestor
    - Centers of the tree
  - Eulerian path/circuit
  - Chinese postman problem

  - Topological sort
  - (Min-Cost) Max Flow
  - Min Cut
    - Maximum Density Subgraph
  - Huffman Coding
  - Min-Cost Arborescence
  - Steiner Tree
  - Kirchoff's matrix tree theorem
  - Prüfer sequences
  - Lovász Toggle
  - Look at the DFS tree (which has no cross-edges)
  - Is the graph a DFA or NFA?
    - Is it the Synchronizing word problem?
- Mathematics
  - Is the function multiplicative?
  - Look for a pattern
  - Permutations
    - Consider the cycles of the permutation
  - Functions
    - Sum of piecewise-linear functions is a piecewise-linear function
    - Sum of convex (concave) functions is convex (concave)
  - Modular arithmetic
    - Chinese Remainder Theorem
    - Linear Congruence
  - Sieve
  - System of linear equations
  - Values too big to represent?
    - Compute using the logarithm
    - Divide everything by some large value
  - Linear programming
    - Is the dual problem easier to solve?
  - Can the problem be modeled as a different combinatorial problem? Does that simplify calculations?
- Logic
  - 2-SAT
  - XOR-SAT (Gauss elimination or Bipartite matching)
- Meet in the middle
- Only work with the smaller half ($\log(n)$)
- Strings
  - Trie (maybe over something weird, like bits)
  - Suffix array
  - Suffix automaton (+DP?)
  - Aho-Corasick
  - eerTree
  - Work with $S + S$
- Hashing
- Euler tour, tree to array
- Segment trees
  - Lazy propagation
  - Persistent
  - Implicit
  - Segment tree of X
- Geometry
  - Minkowski sum (of convex sets)
  - Rotating calipers
  - Sweep line (horizontally or vertically?)
  - Sweep angle
  - Convex hull
- Fix a parameter (possibly the answer).
- Are there few distinct values?
- Binary search
- Sliding Window (+ Monotonic Queue)
- Computing a Convolution? Fast Fourier Transform
- Computing a 2D Convolution? FFT on each row, and then on each column
- Exact Cover (+ Algorithm X)
- Cycle-Finding
- What is the smallest set of values that identify the solution? The cycle structure of the permutation? The powers of primes in the factorization?
- Look at the complement problem
  - Minimize something instead of maximizing
- Immediately enforce necessary conditions. (All values greater than 0? Initialize them all to 1)
- Add large constant to negative numbers to make them positive
- Counting/Bucket sort

| | | |
|---|---|---|
| Catalan | $C_0 = 1,\ C_n = \frac{1}{n+1}\binom{2n}{n} = \sum_{i=0}^{n-1} C_i C_{n-i-1} = \frac{4n-2}{n+1} C_{n-1}$ | |
| Stirling 1st kind | $\left[\begin{smallmatrix}0\\0\end{smallmatrix}\right] = 1,\ \left[\begin{smallmatrix}n\\0\end{smallmatrix}\right] = \left[\begin{smallmatrix}0\\n\end{smallmatrix}\right] = 0,\ \left[\begin{smallmatrix}n\\k\end{smallmatrix}\right] = (n-1)\left[\begin{smallmatrix}n-1\\k\end{smallmatrix}\right] + \left[\begin{smallmatrix}n-1\\k-1\end{smallmatrix}\right]$ | #perms of $n$ objs with exactly $k$ cycles |
| Stirling 2nd kind | $\left\{\begin{smallmatrix}n\\1\end{smallmatrix}\right\} = \left\{\begin{smallmatrix}n\\n\end{smallmatrix}\right\} = 1,\ \left\{\begin{smallmatrix}n\\k\end{smallmatrix}\right\} = k\left\{\begin{smallmatrix}n-1\\k\end{smallmatrix}\right\} + \left\{\begin{smallmatrix}n-1\\k-1\end{smallmatrix}\right\}$ | #ways to partition $n$ objs into $k$ nonempty sets |
| Euler | $\left\langle\begin{smallmatrix}n\\0\end{smallmatrix}\right\rangle = \left\langle\begin{smallmatrix}n\\n-1\end{smallmatrix}\right\rangle = 1,\ \left\langle\begin{smallmatrix}n\\k\end{smallmatrix}\right\rangle = (k+1)\left\langle\begin{smallmatrix}n-1\\k\end{smallmatrix}\right\rangle + (n-k)\left\langle\begin{smallmatrix}n-1\\k-1\end{smallmatrix}\right\rangle$ | #perms of $n$ objs with exactly $k$ ascents |
| Euler 2nd Order | $\left\langle\!\left\langle\begin{smallmatrix}n\\k\end{smallmatrix}\right\rangle\!\right\rangle = (k+1)\left\langle\!\left\langle\begin{smallmatrix}n-1\\k\end{smallmatrix}\right\rangle\!\right\rangle + (2n-k-1)\left\langle\!\left\langle\begin{smallmatrix}n-1\\k-1\end{smallmatrix}\right\rangle\!\right\rangle$ | #perms of $1,1,2,2,...,n,n$ with exactly $k$ ascents |
| Bell | $B_1 = 1,\ B_n = \sum_{k=0}^{n-1} B_k\binom{n-1}{k} = \sum_{k=0}^{n}\left\{\begin{smallmatrix}n\\k\end{smallmatrix}\right\}$ | #partitions of $1..n$ (Stirling 2nd, no limit on k) |

| | |
|---|---|
| #labeled rooted trees | $n^{n-1}$ |
| #labeled unrooted trees | $n^{n-2}$ |
| #forests of $k$ rooted trees | $\frac{k}{n}\binom{n}{k}n^{n-k}$ |
| $\sum_{i=1}^{n} i^2 = n(n+1)(2n+1)/6$ | $\sum_{i=1}^{n} i^3 = n^2(n+1)^2/4$ |
| $!n = n\times!(n-1) + (-1)^n$ | $!n = (n-1)(!(n-1)+!(n-2))$ |
| $\sum_{i=1}^{n}\binom{n}{i}F_i = F_{2n}$ | $\sum_i \binom{n-i}{i} = F_{n+1}$ |
| $\sum_{k=0}^{n}\binom{k}{m} = \binom{n+1}{m+1}$ | $x^k = \sum_{i=0}^{k} i!\left\{\begin{smallmatrix}k\\i\end{smallmatrix}\right\}\binom{x}{i} = \sum_{i=0}^{k}\left\langle\begin{smallmatrix}k\\i\end{smallmatrix}\right\rangle\binom{x+i}{k}$ |
| $a \equiv b \pmod{x,y} \Rightarrow a \equiv b \pmod{\operatorname{lcm}(x,y)}$ | $\sum_{d\mid n}\phi(d) = n$ |
| $ac \equiv bc \pmod{m} \Rightarrow a \equiv b \pmod{\frac{m}{\gcd(c,m)}}$ | $(\sum_{d\mid n}\sigma_0(d))^2 = \sum_{d\mid n}\sigma_0(d)^3$ |
| $p$ prime $\Leftrightarrow (p-1)! \equiv -1 \pmod{p}$ | $\gcd(n^a-1, n^b-1) = n^{\gcd(a,b)} - 1$ |
| $\sigma_x(n) = \prod_{i=0}^{r}\frac{p_i^{(a_i+1)x}-1}{p_i^x-1}$ | $\sigma_0(n) = \prod_{i=0}^{r}(a_i+1)$ |
| $\sum_{k=0}^{m}(-1)^k\binom{n}{k} = (-1)^m\binom{n-1}{m}$ | |
| $2^{\omega(n)} = O(\sqrt{n})$ | $\sum_{i=1}^{n} 2^{\omega(i)} = O(n\log n)$ |
| $d = v_i t + \frac{1}{2}at^2$ | $v_f^2 = v_i^2 + 2ad$ |
| $v_f = v_i + at$ | $d = \frac{v_i+v_f}{2}t$ |

2.3. **The Twelvefold Way.** Putting $n$ balls into $k$ boxes.

| Balls<br>Boxes | same<br>same | distinct<br>same | same<br>distinct | distinct<br>distinct | Remarks |
|---|---|---|---|---|---|
| - | $\mathrm{p}_k(n)$ | $\sum_{i=0}^{k}\left\{\begin{smallmatrix}n\\i\end{smallmatrix}\right\}$ | $\binom{n+k-1}{k-1}$ | $k^n$ | $\mathrm{p}_k(n)$: #partitions of $n$ into $\le k$ positive parts |
| size $\ge 1$ | $\mathrm{p}(n,k)$ | $\left\{\begin{smallmatrix}n\\k\end{smallmatrix}\right\}$ | $\binom{n-1}{k-1}$ | $k!\left\{\begin{smallmatrix}n\\k\end{smallmatrix}\right\}$ | $\mathrm{p}(n,k)$: #partitions of $n$ into $k$ positive parts |
| size $\le 1$ | $[n \le k]$ | $[n \le k]$ | $\binom{k}{n}$ | $n!\binom{k}{n}$ | $[cond]$: 1 if $cond = true$, else 0 |

## Practice Contest Checklist

- How many operations per second? Compare to local machine.
- What is the stack size?
- How to use printf/scanf with long long/long double?
- Are `__int128` and `__float128` available?
- Does MLE give RTE or MLE as a verdict? What about stack overflow?
- What is `RAND_MAX`?
- How does the judge handle extra spaces (or missing newlines) in the output?
- Look at documentation for programming languages.
- Try different programming languages: C++, Java and Python.
- Try the submit script.
- Try local programs: i?python[23], factor.
- Try submitting with `assert(false)` and `assert(true)`.
- Return-value from `main`.
- Look for directory with sample test cases.
- Make sure printing works.
- Remove this page from the notebook.