.*VSTU.*

Team Reference Document

01/12/2018

Contents

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_"

#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>
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.

```
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);
  }
};
bool cmp(Vec a, Vec b) {
```

```
  return a.x < b.x || (a.x == b.x && a.y < b.y);
}
bool cw(Vec a, Vec b, Vec c) {
  return (b - a).cross(c - b) < 0;
}
bool ccw(Vec a, Vec b, Vec c) {
  return (b - a).cross(c - b) > 0;
}
void convex_hull(vector<Vec> & a) {
  if (a.size() == 1) return;
  sort(a.begin(), a.end(), &cmp);
  Vec p1 = a[0], p2 = a.back();
  vector<Vec> up, down;
  up.push_back(p1);
  down.push_back(p1);
  for (size_t i=1; i<a.size(); ++i) {
    if (i==a.size()-1 || cw(p1, a[i], p2)) {
      while (up.size()>=2
          && !cw(up[up.size()-2], up[up.size()-1], a[i]))
        up.pop_back();
      up.push_back (a[i]);
    }
    if (i == a.size()-1 || ccw(p1, a[i], p2)) {
      while (down.size()>=2
          && !ccw(down[down.size()-2],
            down[down.size()-1], a[i]))
        down.pop_back();
      down.push_back(a[i]);
    }
  }
  a.clear();
  for (size_t i=0; i<up.size(); ++i)
    a.push_back(up[i]);
  for (size_t i=down.size()-2; i>0; --i)
    a.push_back(down[i]);
}
```

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. 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]; }
};
```

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

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.