



Universidade de Brasília

É só Fazer

Ruan Petrus, Eduardo Freire, Arthur Botelho

2024-06-15

1 Contest

2 Data structures

3 Matching

Contest (1)

```
template.cpp26 lines

#include <bits/stdc++.h>

using namespace std;

#define int long long
#define endl '\n'
#define rep(i, a, b) for(int i = (a); i < (b); ++i)
#define all(x) begin(x), end(x)
#define sz(x) (int)(x).size()
#define debug(var) cout << #var << ": " << var << endl
#define pb push_back
#define eb emplace_back
typedef long long ll;
typedef pair<int, int> pii;
typedef vector<int> vi;

void solve() {
}

int32_t main() {
    ios_base::sync_with_stdio(0); cout.tie(0); cin.tie(0);
    int t = 1;
    while(t--) solve();

    return 0;
}
```

```
.bashrc3 lines

alias c='g++ -Wall -Wconversion -Wfatal-errors -g -std=c++17 \
-fsanitize=undefined,address'
xmodmap -e 'clear lock' -e 'keycode 66=less greater' #caps = <
```

```
.vimrc4 lines

set cin ai is ts=4 sw=4 nu rnu
" Select a region and then type :Hash
ca Hash w !cpp -dD -P -fpreprocessed | tr -d '[:space:]' \
\ | md5sum | cut -c-6
```

```
hash.sh3 lines

# Hashes a file, ignoring all whitespace and comments. Use for
# verifying that code was correctly typed.
cpp -dD -P -fpreprocessed | tr -d '[:space:]' | md5sum |cut -c-6
```

```
troubleshoot.txt52 lines

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

Wrong answer:
Print your solution! Print debug output, as well.
```

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

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

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

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

Data structures (2)

```
SegTree.h731230, 37 lines

Description: Iterative SegTree Can be changed by modifying Spec
Time: O(log N)

template<typename Spec>
struct SegTree {
    using LS = Spec;
    using S = typename LS::S;
    using K = typename LS::K;
    int n;
    vector<S> seg;

    SegTree(const vector<S> & v)
        : n(v.size()), seg(2*n) {
        rep(i, 0, n) seg[i+n] = v[i];
        for(int i = n-1; i >= 1; i--) seg[i] = LS::op(seg[i*2], seg[i*2+1]);
    }

    void update(int no, K val) {
        no += n;
        seg[no] = LS::update(val, seg[no]);
    }
};
```

```
while (no > 1) no /= 2, seg[no] = LS::op(seg[no*2], seg[no*2+1]);
}

S query(int l, int r) { // [l, r)
    S vl = LS::id(), vr = LS::id();
    for (l += n, r += n; l < r; l /= 2, r /= 2) {
        if (l & 1) vl = LS::op(vl, seg[l++]);
        if (r & 1) vr = LS::op(seg[--r], vr);
    }
    return LS::op(vl, vr);
};

struct Spec {
    using S = int;
    using K = int;
    static S op(S a, S b) { return max(a, b); }
    static S update(K f, S a) { return f; }
    static S id() { return 0; }
};
```

LazySeg.h  
Description: Iterative Lazy SegTree Can be changed by modifying Spec  
Time: O(log N)3beda5, 96 lines

```
template<typename Spec>
struct LazySeg {
    using LS = Spec;
    using S = typename LS::S;
    using K = typename LS::K;
    int n;
    vector<S> seg;
    vector<K> lazy;
    vector<bool> has_lazy;
    // vector<int> lx, rx; // Additional info

    LazySeg(vector<S> & v) : n(v.size()), seg(2*n), lazy(n),
        has_lazy(n) {
        rep(no, 0, n) seg[no+n] = v[no];
        for (int no = n-1; no >= 1; no--) pull(no);

        // Additional info, n must be power of two
        /*
        lx.assign(2*n, 0); rx.assign(2*n, 0);
        lx[1] = 0; rx[1] = n;
        rep(no, 1, n) {
            int mid = (lx[no] + rx[no])/2;
            lx[no*2] = lx[no]; rx[no*2] = mid;
            lx[no*2+1] = mid; rx[no*2+1] = rx[no];
        }
        */
    }

    S query(int l, int r) { // [l, r)
        l += n;
        r += n;
        push_to(l); push_to(r-1);
        S vl = LS::id(), vr = LS::id();
        while(l < r) {
            if (l & 1) vl = LS::op(vl, seg[l++]);
            if (r & 1) vr = LS::op(seg[--r], vr);
            l >>= 1; r >>= 1;
        }
        return LS::op(vl, vr);
    }

    void update(int l, int r, K val) {
        l += n;
```

```

    r += n;
    push_to(1); push_to(r-1);
    int lo = 1, ro = 1;
    while(1 < r) {
        if (1 & 1) lo = max(lo, 1), apply(l++, val);
        if (r & 1) ro = max(ro, r), apply(--r, val);
        1 >>= 1; r >>= 1;
    }
    pull_from(lo);
    pull_from(ro-1);
}

void apply(int no, K val) {
    seg[no] = LS::update(val, seg[no]);
    // seg[no] = LS::update(val, seg[no], lx[no], rx[no]);

    if (no < n) {
        if (has_lazy[no]) lazy[no] = LS::compose(val, lazy[no]);
        else lazy[no] = val;
        has_lazy[no] = true;
    }
}

void pull_from(int no) {
    while(no > 1) no >>= 1, pull(no);
}

void pull(int no) {
    seg[no] = LS::op(seg[no*2], seg[no*2+1]);
}

void push_to(int no) {
    int h = 0; int p2 = 1;
    while(p2 < no) p2 *= 2, h++;
    for (int i = h; i >= 1; i--) push(no >> i);
}

void push(int no) {
    if (has_lazy[no]) {
        apply(no*2, lazy[no]);
        apply(no*2+1, lazy[no]);
        has_lazy[no] = false;
    }
}

};

struct Spec {
    using S = int;
    using K = int;
    static S op(S a, S b) { return max(a, b); }
    static S update(K f, S a) { return f + a; }
    static K compose(const K f, const K g) { return f + g; }
    static S id() { return 0; }
};
```

## Matching (3)

### OnlineMatching.h

**Description:** Modified khun developed for specific question able to run  $2 * 10^6$  queries, in  $2 * 10^6 \times 10^6$  graph in 3 seconds codeforces  
**Time:**  $\mathcal{O}(confia)$

6ac539, 42 lines

```

struct OnlineMatching {
    int n = 0, m = 0;
    vector<int> vis, match, dist;
    vector<vector<int>> g;
    vector<int> last;
    int t = 0;
```

```

    OnlineMatching(int n_, int m_) : n(n_), m(m_),
    vis(n, 0), match(m, -1), dist(n, n+1), g(n), last(n, -1)
    {}

    void add(int a, int b) {
        g[a].pb(b);
    }

    bool kuhn(int a) {
        vis[a] = t;
        for(int b: g[a]) {
            int c = match[b];
            if (c == -1) {
                match[b] = a;
                return true;
            }
            if (last[c] != t || (dist[a] + 1 < dist[c]))
                dist[c] = dist[a] + 1, last[c] = t;
        }
        for (int b: g[a]) {
            int c = match[b];
            if (dist[a] + 1 == dist[c] && vis[c] != t && kuhn(c)) {
                match[b] = a;
                return true;
            }
        }
        return false;
    }

    bool can_match(int a) {
        t++;
        last[a] = t;
        dist[a] = 0;
        return kuhn(a);
    }
};
```

# Techniques (A)

techniques.txt	159 lines
Recursion	
Divide and conquer	
Finding interesting points in N log N	
Algorithm analysis	
Master theorem	
Amortized time complexity	
Greedy algorithm	
Scheduling	
Max contiguous subvector sum	
Invariants	
Huffman encoding	
Graph theory	
Dynamic graphs (extra book-keeping)	
Breadth first search	
Depth first search	
* Normal trees / DFS trees	
Dijkstra's algorithm	
MST: Prim's algorithm	
Bellman-Ford	
Konig's theorem and vertex cover	
Min-cost max flow	
Lovasz toggle	
Matrix tree theorem	
Maximal matching, general graphs	
Hopcroft-Karp	
Hall's marriage theorem	
Graphical sequences	
Floyd-Warshall	
Euler cycles	
Flow networks	
* Augmenting paths	
* Edmonds-Karp	
Bipartite matching	
Min. path cover	
Topological sorting	
Strongly connected components	
2-SAT	
Cut vertices, cut-edges and biconnected components	
Edge coloring	
* Trees	
Vertex coloring	
* Bipartite graphs (=> trees)	
* 3^n (special case of set cover)	
Diameter and centroid	
K'th shortest path	
Shortest cycle	
Dynamic programming	
Knapsack	
Coin change	
Longest common subsequence	
Longest increasing subsequence	
Number of paths in a dag	
Shortest path in a dag	
Dynprog over intervals	
Dynprog over subsets	
Dynprog over probabilities	
Dynprog over trees	
3^n set cover	
Divide and conquer	
Knuth optimization	
Convex hull optimizations	
RMQ (sparse table a.k.a 2^k-jumps)	
Bitonic cycle	
Log partitioning (loop over most restricted)	
Combinatorics	

Computation of binomial coefficients
Pigeon-hole principle
Inclusion/exclusion
Catalan number
Pick's theorem
Number theory
Integer parts
Divisibility
Euclidean algorithm
Modular arithmetic
* Modular multiplication
* Modular inverses
* Modular exponentiation by squaring
Chinese remainder theorem
Fermat's little theorem
Euler's theorem
Phi function
Frobenius number
Quadratic reciprocity
Pollard-Rho
Miller-Rabin
Hensel lifting
Vieta root jumping
Game theory
Combinatorial games
Game trees
Mini-max
Nim
Games on graphs
Games on graphs with loops
Grundy numbers
Bipartite games without repetition
General games without repetition
Alpha-beta pruning
Probability theory
Optimization
Binary search
Ternary search
Unimodality and convex functions
Binary search on derivative
Numerical methods
Numeric integration
Newton's method
Root-finding with binary/ternary search
Golden section search
Matrices
Gaussian elimination
Exponentiation by squaring
Sorting
Radix sort
Geometry
Coordinates and vectors
* Cross product
* Scalar product
Convex hull
Polygon cut
Closest pair
Coordinate-compression
Quadtrees
KD-trees
All segment-segment intersection
Sweeping
Discretization (convert to events and sweep)
Angle sweeping
Line sweeping
Discrete second derivatives
Strings
Longest common substring
Palindrome subsequences

Knuth-Morris-Pratt
Tries
Rolling polynomial hashes
Suffix array
Suffix tree
Aho-Corasick
Manacher's algorithm
Letter position lists
Combinatorial search
Meet in the middle
Brute-force with pruning
Best-first (A*)
Bidirectional search
Iterative deepening DFS / A*
Data structures
LCA (2^k-jumps in trees in general)
Pull/push-technique on trees
Heavy-light decomposition
Centroid decomposition
Lazy propagation
Self-balancing trees
Convex hull trick (wcipeg.com/wiki/Convex_hull_trick)
Monotone queues / monotone stacks / sliding queues
Sliding queue using 2 stacks
Persistent segment tree