Uzhhorod National University

3yagoda

Vasyl Merenych, Dmytro Mayor, Roman Pitsura

2022-09-09

Contest (1)

template.cpp

88 lines

```cpp
// #pragma comment(linker, "/stack:200000000")
// #pragma GCC optimize("Ofast")
// #pragma GCC optimize("O3,unroll-loops")
// #pragma GCC target("sse,sse2,sse3,ssse3,sse4,avx2")

// #define _GLIBCXX_DEBUG
// #define _GLIBCXX_DEBUG_PEDANTIC

#include <iostream>
#include <vector>
#include <algorithm>
#include <map>
#include <set>
#include <queue>
#include <deque>
#include <cmath>
#include <climits>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>

using namespace __gnu_pbds;
using namespace std;

const int MOD = 998244353;
using ll = long long;
const ll INF = 1e18;

// #define int ll

template<typename T>
using ordered_set = tree<T,null_type,less<T>,rb_tree_tag,
     tree_order_statistics_node_update>;

template<typename T>
using graph = vector<vector<T>>;

template<typename T>
istream& operator>>(istream& in, vector<T>& a) {
    for (auto& i: a) {
        in >> i;
    }
    return in;
}

template<typename T>
ostream& operator<<(ostream& out, vector<T>& a) {
    for (auto& i: a) {
        out << i << " ";
    }
    return out;
}

int fast_pow(int a, int b, int mod) {
    if (b == 0)
        return 1;
    if (b % 2) {
        return (a * fast_pow(a, b - 1, mod)) % mod;
    }
    int k = fast_pow(a, b / 2, mod);
    return (k * k) % mod;
}

int fast_pow(int a, int b) {
    if (b == 0)
        return 1;
    if (b % 2) {
        return (a * fast_pow(a, b - 1));
    }
    int k = fast_pow(a, b / 2);
    return (k * k);
}

void solve() {

}

int32_t main(int32_t argc, const char * argv[]) {
  cin.tie(0);
  cout.tie(0);
  ios_base::sync_with_stdio(0);
    // insert code here...
    int tt= 1;
    // std::cin >> tt;
    while (tt--) {
        solve();
    }
    return 0;
}
```

.bashrc

3 lines

```bash
alias c='g++ -Wall -Wconversion -Wfatal-errors -g -std=c++14 \
  -fsanitize=undefined,address'
xmodmap -e 'clear lock' -e 'keycode 66=less greater' #caps = <>
```

.vimrc

6 lines

```vim
set cin aw ai is ts=4 sw=4 tm=50 nu noeb bg=dark ru cul
sy on   |   im jk <esc>   |   im kj <esc>   |   no ; :
" Select region and then type :Hash to hash your selection.
" Useful for verifying that there aren't mistypes.
ca Hash w !cpp -dD -P -fpreprocessed \| tr -d '[:space:]' \
 \| md5sum \| cut -c-6
```

hash.sh

3 lines

```bash
# 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
```

Data structures (2)

SegmentTree.h

Description: Zero-indexed sum-tree. Bounds are inclusive to the left and to the right.

Time: update - $\mathcal{O}(\log N)$, get - $\mathcal{O}(\log N)$

<vector> 71fa87, 54 lines

```cpp
struct segment_tree {
  struct node{
    int val = 0;
    node *left = nullptr;
    node *right = nullptr;
  };

  node* new_node() {
    const int SZ = 100000;
    static node *block;
    static int count = SZ;
    if (count == SZ) {
      block= new node[SZ];
      count = 0;
    }
    return (block + count++);
  };

  const int N = 100000;
  void update(int pos, int val) {
    update(root, 0, N, pos, val);
  }

  int get(int l, int r) {
    return get(root, 0, N, l, r);
  }
  node *root = new_node();

  void update(node*& v, int tl, int tr, int pos, int value) {
    if (!v)
      v = new_node();
    if (tl == tr) {
      v->val += value;
      return;
    }
    int mid = (tl + tr) / 2;
    if (pos <= mid)
      update(v->left, tl, mid, pos, value);
    else
      update(v->right, mid + 1, tr, pos, value);
    v->val = (v->left ? v->left->val : 0) + (v->right ? v->
        right->val : 0);
  }

  int get(node*& v, int tl, int tr, int l, int r)
  {
    if (!v || r < tl || tr < l)
      return 0;
    if (l <= tl && tr <= r)
      return v->val;
    int mid = (tl + tr) / 2;
    return get(v->left, tl, mid, l, r) + get(v->right, mid + 1,
         tr, l, r);
  }
};
```

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
```