KTH Royal Institute of Technology

Omogen Heap

Simon Lindholm, Johan Sannemo, Mårten Wiman

2020-11-23

Contest (1)

template.cpp

10 lines

```
#include <bits/stdc++.h>
#define all(x) begin(x), end(x)
using namespace std;

using ll = long long;

int main() {
  cin.tie(0)->sync_with_stdio(0);
  cin.exceptions(cin.failbit);
}
```

Data structures (2)

BIT.h

Description: Query [l, r] sums, and point updates. kth() returns the smallest index i s.t. query(0, i) >= k
Time: $\mathcal{O}(\log n)$ for all ops.

33f78c, 22 lines

```
template <typename T>
struct BIT {
  vector<T> s;
  int n;
  BIT(int n) : s(n + 1), n(n) {}
  void update(int i, T v) {
    for (i++; i <= n; i += i & -i) s[i] += v;
  }
  T query(int i) {
    T ans = 0;
    for (i++; i; i -= i & -i) ans += s[i];
    return ans;
  }
  T query(int l, int r) { return query(r) - query(l - 1); }
  int kth(T k) { // returns n if k > sum of tree
    if (k <= 0) return -1;
    int i = 0;
    for (int pw = 1 << __lg(n); pw; pw >>= 1)
      if (i + pw <= n && s[i + pw] < k) k -= s[i += pw];
    return i;
  }
};
```

KDBIT.h

Description: k-dimensional BIT. `BIT<int, N, M>` gives an $N \times M$ BIT. Query: `bit.query(x1, x2, y1, y2)` Update: `bit.update(x, y, delta)`
Time: $O(\log^k n)$ Status: Tested

3b9692, 28 lines

```
template <class T, int... Ns>
struct BIT {
  T val = 0;
  void update(T v) { val += v; }
  T query() { return val; }
};
template <class T, int N, int... Ns>
struct BIT<T, N, Ns...> {
  BIT<T, Ns...> bit[N + 1];
  // map<int, BIT<T, Ns...>> bit;
  // if the memory use is too high
  template <class... Args>
  void update(int i, Args... args) {
    for (i++; i <= N; i += i & -i) bit[i].update(args...);
  }
  template <class... Args>
  T query(int i, Args... args) {
    T ans = 0;
    for (i++; i; i -= i & -i) ans += bit[i].query(args...);
    return ans;
  }
  template <class... Args,
            enable_if_t<(sizeof...(Args) ==
                         2 * sizeof...(Ns))>* = nullptr>
  T query(int l, int r, Args... args) {
    return query(r, args...) - query(l - 1, args...);
  }
};
```

DSU.h

Description: Maintains union of disjoint sets
Time: $\mathcal{O}(\alpha(N))$

c22586, 14 lines

```
struct DSU {
  vector<int> s;
  DSU(int n) : s(n, -1) {}
  int find(int i) { return s[i] < 0 ? i : s[i] = find(s[i]); }
  bool join(int a, int b) {
    a = find(a), b = find(b);
    if (a == b) return false;
    if (s[a] > s[b]) swap(a, b);
    s[a] += s[b], s[b] = a;
    return true;
  }
  int size(int i) { return -s[find(i)]; }
  bool same(int a, int b) { return find(a) == find(b); }
};
```

RMQ.h

Description: Constant time subarray min/max queries for a fixed array
Time: $O(nlogn)$ initialization and $O(1)$ queries. Status: Tested

536eac, 15 lines

```
template <typename T, class Compare = less<T>>
struct RMQ {
  vector<vector<T>> t;
  Compare cmp;
  RMQ(vector<T>& a) : t(__lg(a.size()) + 1, a) {
    int n = a.size(), lg = __lg(n);
    for (int k = 1, len = 1; k <= lg; k++, len <<= 1)
      for (int i = 0; i + 2 * len - 1 < n; i++)
        t[k][i] = min(t[k - 1][i], t[k - 1][i + len], cmp);
  }
  T query(int a, int b) {
    int k = __lg(b - a + 1), len = 1 << k;
    return min(t[k][a], t[k][b - len + 1], cmp);
  }
```

```
};
```

Splay.h

Description: An implicit balanced BST. You only need to change `update()` and `prop()`.
If used for link-cut tree, code everything up to `splay()`. Time: amortized $O(\log n)$ for all operations

0d0cee, 92 lines

```
struct node {
  node *ch[2] = {0}, *p = 0;
  int cnt = 1, val;

  node(int val, node* l = 0, node* r = 0)
    : ch{l, r}, val(val) {}
};

int cnt(node* x) { return x ? x->cnt : 0; }
int dir(node* p, node* x) { return p && p->ch[0] != x; }
void setLink(node* p, node* x, int d) {
  if (p) p->ch[d] = x;
  if (x) x->p = p;
}

node* update(node* x) {
  if (!x) return 0;
  x->cnt = 1 + cnt(x->ch[0]) + cnt(x->ch[1]);
  setLink(x, x->ch[0], 0);
  setLink(x, x->ch[1], 1);
  return x;
}

void prop(node* x) {
  if (!x) return;
  // update(x); // needed if prop() can change subtree sizes
}

void rotate(node* x, int d) {
  if (!x || !x->ch[d]) return;
  node *y = x->ch[d], *z = x->p;
  setLink(x, y->ch[d ^ 1], d);
  setLink(y, x, d ^ 1);
  setLink(z, y, dir(z, x));
  update(x);
  update(y);
}

node* splay(node* x) {
  while (x && x->p) {
    node *y = x->p, *z = y->p;
    // prop(z), prop(y), prop(x); // needed for LCT
    int dy = dir(y, x), dz = dir(z, y);
    if (!z)
      rotate(y, dy);
    else if (dy == dz)
      rotate(z, dz), rotate(y, dy);
    else
      rotate(y, dy), rotate(z, dz);
  }
  return x;
}

// the returned node becomes the new root, update the root
// pointer!
node* nodeAt(node* x, int pos) {
  if (!x) return 0;
  while (prop(x), cnt(x->ch[0]) != pos)
    if (pos < cnt(x->ch[0]))
      x = x->ch[0];
    else
```

```
    pos -= cnt(x->ch[0]) + 1, x = x->ch[1];
  return splay(x);
}

node* merge(node* l, node* r) {
  if (!l || !r) return l ?: r;
  l = nodeAt(l, cnt(l) - 1);
  setLink(l, r, 1);
  return update(l);
}

// first is everything < pos, second is >= pos
pair<node*, node*> split(node* t, int pos) {
  if (pos <= 0 || !t) return {0, t};
  if (pos > cnt(t)) return {t, 0};
  node *l = nodeAt(t, pos - 1), *r = l->ch[1];
  if (r) l->ch[1] = r->p = 0;
  return {update(l), update(r)};
}

// insert a new node between pos-1 and pos
node* insert(node* t, int pos, int val) {
  auto [l, r] = split(t, pos);
  return update(new node(val, l, r));
}

// apply lambda to all nodes in an inorder traversal
template <class F>
void each(node* x, F f) {
  if (x) prop(x), each(x->ch[0], f), f(x), each(x->ch[1], f);
}
```

Geometry (3)

Graphs (4)

SCCTarjan.h

Description: Finds strongly connected components of a directed graph. Visits/indexes SCCs in reverse topological order.
Usage: `scc(graph)` returns an array that has the ID of each node's SCC. `scc(graph, [&](vector<int>& v) { ... })` calls the lambda on each SCC, and returns the same array.
Time: $\mathcal{O}(|V| + |E|)$

358d18, 37 lines

```
namespace SCCTarjan {
  vector<int> val, comp, z, cont;
  int Time, ncomps;
  template <class G, class F>
  int dfs(int j, G& g, F& f) {
    int low = val[j] = ++Time, x;
    z.push_back(j);
    for (auto e : g[j])
      if (comp[e] < 0) low = min(low, val[e] ?: dfs(e, g, f));
    if (low == val[j]) {
      do {
        x = z.back();
        z.pop_back();
        comp[x] = ncomps;
        cont.push_back(x);
      } while (x != j);
      f(cont);
      cont.clear();
      ncomps++;
    }
    return val[j] = low;
  }
  template <class G, class F>
  vector<int> scc(G& g, F f) {
    int n = g.size();
    val.assign(n, 0);
    comp.assign(n, -1);
    Time = ncomps = 0;
    for (int i = 0; i < n; i++)
      if (comp[i] < 0) dfs(i, g, f);
    return comp;
  }
  template <class G> // convenience function w/o lambda
  vector<int> scc(G& g) {
    return scc(g, [](auto& v) {});
  }
} // namespace SCCTarjan
```

SCCKosaraju.h

Description: Finds strongly connected components of a directed graph. Visits/indexes SCCs in topological order.
Usage: `scc(graph)` returns an array that has the ID of each node's SCC.
Time: $\mathcal{O}(|V| + |E|)$

9b78e7, 29 lines

```
namespace SCCKosaraju {
  vector<vector<int>> adj, radj;
  vector<int> todo, comp;
  vector<bool> vis;
  void dfs1(int x) {
    vis[x] = 1;
    for (int y : adj[x])
      if (!vis[y]) dfs1(y);
    todo.push_back(x);
  }
  void dfs2(int x, int i) {
    comp[x] = i;
    for (int y : radj[x])
      if (comp[y] == -1) dfs2(y, i);
  }
  vector<int> scc(vector<vector<int>>& _adj) {
    adj = _adj;
    int time = 0, n = adj.size();
    comp.resize(n, -1), radj.resize(n), vis.resize(n);
    for (int x = 0; x < n; x++)
      for (int y : adj[x]) radj[y].push_back(x);
    for (int x = 0; x < n; x++)
      if (!vis[x]) dfs1(x);
    reverse(todo.begin(), todo.end());
    for (int x : todo)
      if (comp[x] == -1) dfs2(x, time++);
    return comp;
  }
}; // namespace SCCKosaraju
```

LCA.h

Description: Answers lowest common ancestor queries on a rooted tree using RMQ. Works with both directed and undirected adjacency lists.
Time: $\mathcal{O}(1)$ queries with an $\mathcal{O}(n \log n)$ precomp (RMQ).

"../../content/data-structures/RMQ.h" 7e028d, 20 lines

```
struct LCA {
  int T = 0;
  vector<int> pre, path, times;
  RMQ<int> rmq;
  LCA(vector<vector<int>>& adj, int root = 0)
    : pre(adj.size()), rmq((dfs(root, -1, adj), times)) {}
  void dfs(int u, int p, vector<vector<int>>& adj) {
    pre[u] = T++;
    for (int v : adj[u])
      if (v != p) {
        path.push_back(u), times.push_back(pre[u]);
        dfs(v, u, adj);
      }
  }
  int lca(int u, int v) {
    if (u == v) return u;
    tie(u, v) = minmax(pre[u], pre[v]);
    return path[rmq.query(u, v - 1)];
  }
};
```

Mathematics (5)

Fraction.h

Description: Struct for representing fractions/rationals. All ops are $O(\log N)$ due to GCD in constructor. Uses cross multiplication.

a1de34, 27 lines

```
template <typename T>
struct Q {
  T a, b;
  Q(T p, T q = 1) {
    T g = gcd(p, q);
    a = p / g;
    b = q / g;
    if (b < 0) a = -a, b = -b;
  }
  T gcd(T x, T y) const { return __gcd(x, y); }
  Q operator+(const Q& o) const {
    return {a * o.b + o.a * b, b * o.b};
  }
  Q operator-(const Q& o) const {
    return *this + Q(-o.a, o.b);
  }
  Q operator*(const Q& o) const { return {a * o.a, b * o.b}; }
  Q operator/(const Q& o) const { return *this * Q(o.b, o.a); }
  Q recip() const { return {b, a}; }
  int signum() const { return (a > 0) - (a < 0); }
  bool operator<(const Q& o) const {
    return a * o.b < o.a * b;
  }
  friend ostream& operator<<(ostream& cout, const Q& o) {
    return cout << o.a << "/" << o.b;
  }
};
```

FractionOverflow.h

Description: Safer struct for representing fractions/rationals. Comparison is 100% overflow safe; other ops are safer but can still overflow. All ops are $O(\log N)$.

a42e99, 43 lines

```
template <typename T>
struct QO {
  T a, b;
  QO(T p, T q = 1) {
    T g = gcd(p, q);
    a = p / g;
    b = q / g;
    if (b < 0) a = -a, b = -b;
  }
  T gcd(T x, T y) const { return __gcd(x, y); }
  QO operator+(const QO& o) const {
    T g = gcd(b, o.b), bb = b / g, obb = o.b / g;
    return {a * obb + o.a * bb, o.b * obb};
  }
  QO operator-(const QO& o) const {
    return *this + QO(-o.a, o.b);
  }
  QO operator*(const QO& o) const {
    T g1 = gcd(a, o.b), g2 = gcd(o.a, b);
    return {(a / g1) * (o.a / g2), (b / g2) * (o.b / g1)};
```

```
  }
  QO operator/(const QO& o) const {
    return *this * QO(o.b, o.a);
  }
  QO recip() const { return {b, a}; }
  int signum() const { return (a > 0) - (a < 0); }
  static bool lessThan(T a, T b, T x, T y) {
    if (a / b != x / y) return a / b < x / y;
    if (x % y == 0) return false;
    if (a % b == 0) return true;
    return lessThan(y, x % y, b, a % b);
  }
  bool operator<(const QO& o) const {
    if (this->signum() != o.signum() || a == 0) return a < o.a;
    if (a < 0)
      return lessThan(abs(o.a), o.b, abs(a), b);
    else
      return lessThan(a, b, o.a, o.b);
  }
  friend ostream& operator<<(ostream& cout, const QO& o) {
    return cout << o.a << "/" << o.b;
  }
};
```

PrimeSieve.h

Description: Prime sieve for generating all primes up to a certain limit. isprime[i] is true iff i is a prime.
Time: lim=100'000'000 $\approx$ 0.8 s. Runs 30% faster if only odd indices are stored.

dc4f55, 14 lines

```
const int MAX_PR = 5'000'000;
bitset<MAX_PR> isprime;
vector<int> primeSieve(int lim) {
  isprime.set();
  isprime[0] = isprime[1] = 0;
  for (int i = 4; i < lim; i += 2) isprime[i] = 0;
  for (int i = 3; i * i < lim; i += 2)
    if (isprime[i])
      for (int j = i * i; j < lim; j += i * 2) isprime[j] = 0;
  vector<int> pr;
  for (int i = 2; i < lim; i++)
    if (isprime[i]) pr.push_back(i);
  return pr;
}
```

PrimeSieveFast.h

Description: Prime sieve for generating all primes smaller than LIM.
Time: LIM=1e9 $\approx$ 1.5s

a1933d, 23 lines

```
const int LIM = 1e8;
bitset<LIM> isPrime;
vector<int> primeSieve() {
  const int S = round(sqrt(LIM)), R = LIM / 2;
  vector<int> pr = {2}, sieve(S + 1);
  pr.reserve(int(LIM / log(LIM) * 1.1));
  vector<pair<int, int>> cp;
  for (int i = 3; i <= S; i += 2)
    if (!sieve[i]) {
      cp.push_back({i, i * i / 2});
      for (int j = i * i; j <= S; j += 2 * i) sieve[j] = 1;
    }
  for (int L = 1; L <= R; L += S) {
    array<bool, S> block{};
    for (auto& [p, idx] : cp)
      for (int i = idx; i < S + L; idx = (i += p))
        block[i - L] = 1;
    for (int i = 0; i < min(S, R - L); i++)
      if (!block[i]) pr.push_back((L + i) * 2 + 1);
  }
  for (int i : pr) isPrime[i] = 1;
  return pr;
}
```

Miscellaneous (6)

NDimensionalVector.h

3c0f61, 12 lines

```
template <int D, typename T>
struct Vec : public vector<Vec<D - 1, T>> {
  static_assert(D >= 1,
                "Vector dimension must be greater than zero!");
  template <typename... Args>
  Vec(int n = 0, Args... args)
    : vector<Vec<D - 1, T>>(n, Vec<D - 1, T>(args...)) {}
};
template <typename T>
struct Vec<1, T> : public vector<T> {
  Vec(int n = 0, const T& val = T()) : vector<T>(n, val) {}
};
```

Submasks.h

35424b, 3 lines

```
for (int mask = 0; mask < (1 << n); mask++)
  for (int sub = mask; sub; sub = (sub - 1) & mask)
// do thing
```

Strings (7)

ZValues.h

151ee3, 10 lines

```
vector<int> zValues(string& s) {
  int n = ( int )s.length();
  vector<int> z(n);
  for (int i = 1, l = 0, r = 0; i < n; ++i) {
    if (i <= r) z[i] = min(r - i + 1, z[i - l]);
    while (i + z[i] < n && s[z[i]] == s[i + z[i]]) ++z[i];
    if (i + z[i] - 1 > r) l = i, r = i + z[i] - 1;
  }
  return z;
}
```