NANJING UNIVERSITY

ACM-ICPC Codebook 3

Data Structures

July 14, 2017

Contents

1 Segment Tree

1.1 Binary indexed tree

1.1.1 Point update, range query

Usage:

`init(n)`	Initialize the tree with 0.
`add(n, x)`	Add the n-th element by x.
`sum(n)`	Return the sum of the first n elements.

Time complexity: $O(n)$ for initialization; $O(\log n)$ for each update and query.

```
inline int lowbit(int x){return x&-x;}

struct bit_purq{ // point update, range query
    int N;
    vector<LL> tr;

    void init(int n){ // fill the array with 0
        tr.resize(N = n + 5);
    }

    LL sum(int n){
        LL ans = 0;
        while (n){
            ans += tr[n];
            n -= lowbit(n);
        }
        return ans;
    }

    void add(int n, LL x){
        while (n < N){
            tr[n] += x;
            n += lowbit(n);
        }
    }
};
```

1.1.2 Range update, point query

Usage:

`init(n)`	Initialize the tree with 0.
`add(n, x)`	Add the first n element by x.
`query(n)`	Return the value of the n-th element.

Time complexity: $O(n)$ for initialization; $O(\log n)$ for each update and query.

```
inline int lowbit(int x){return x&-x;}

struct bit_rupq{ // range update, point query
    int N;
    vector<LL> tr;

    void init(int n){ // fill the array with 0
        tr.resize(N = n + 5);
    }

    LL query(int n){
        LL ans = 0;
        while (n < N){
            ans += tr[n];
            n += lowbit(n);
        }
        return ans;
    }

    void add(int n, LL x){
        while (n){
            tr[n] += x;
            n -= lowbit(n);
        }
    }
};
```

1.1.3 Range update, range query

Usage:

`init(n)`	Initialize the tree with 0.
`add(l, r, x)`	Add the elements in $[l, r]$ by x.
`query(l, r)`	Return the sum of the elements in $[l, r]$.

Requirement:
1.1.1 Point update, range query

Time complexity: $O(n)$ for initialization; $O(\log n)$ for each update and query.

```
struct bit_rurq{
```

```
    bit_purq d, di;

    void init(int n){
        d.init(n); di.init(n);
    }

    void add(int l, int r, LL x){
        d.add(l, x); d.add(r+1, -x);
        di.add(l, x*l); di.add(r+1, -x*(r+1));
    }

    LL query(int l, int r){
        return (r+1)*d.sum(r) - di.sum(r) - l*d.sum(l-1) + di.sum(l-1);
    }
};
```

2 Miscellaneous Data Structures

2.1 Union-find set

Data structure for disjoint sets with path-compression optimization.

Usage:

`init(n)`	Initialize the sets from 0 to n, each includes one element.
`find(x)`	Return the representative of the set containing x.
`unite(u, v)`	Unite the two sets containing u and v. Return `false` if u and v are already in the same set; otherwise `true`.

Time complexity: $O(n)$ for initialization; $O(\log n)$ for find and union.

```
struct ufs{
    vector<int> p;

    void init(int n){
        p.resize(n + 1);
        for (int i=0; i<n; i++) p[i] = i;
    }

    int find(int x){
        if (p[x] == x) return x;
        return p[x] = find(p[x]);
    }

    bool unite(int u, int v){
```

```
15          u = find(u); v = find(v);
16          p[u] = v;
17          return u != v;
18      }
19  };
```