

Backlight's Code Template

Backlight @ CSU

2021 年 4 月 30 日

目录

1	ds	1
1.1	SGT	1
2	graph	3
2.1	Dijkstra	3
3	math	4
3.1	Lucas	4
4	other	6
4.1	BFprt	6
5	string	7
5.1	KMP	7

1 ds

1.1 SGT

```
1 template<typename T>
2 struct SGTTree {
3     static constexpr double alpha = 0.75; // alpha \in (0.5, 1)
4     int root, tot, buf_size;
5     T v[N];
6     int s[N], sz[N], sd[N], cnt[N], l[N], r[N], buf[N];
7
8
9     SGTTree()
10    {
11        root = tot = 0;
12    }
13
14    int new_node(T _v)
15    {
16        ++tot;
17        v[tot] = _v;
18        s[tot] = sz[tot] = sd[tot] = cnt[tot] = 1;
19        l[tot] = r[tot] = 0;
20        return tot;
21    }
22
23    void push_up(int x)
24    {
25        if (!x) return;
26        int lc = l[x], rc = r[x];
27        s[x] = s[lc] + 1 + s[rc];
28        sz[x] = sz[lc] + cnt[x] + sz[rc];
29        sd[x] = sd[lc] + (cnt[x] != 0) + sd[rc];
30    }
31
32    bool balance(int x)
33    {
34        int lc = l[x], rc = r[x];
35        if (alpha * s[x] <= max(s[lc], s[rc])) return false;
36        if (alpha * s[x] >= sd[x]) return false;
37        return true;
38    }
39
40    void flatten(int x)
41    {
42        if (!x) return;
43        flatten(l[x]);
44        if (cnt[x]) buf[++buf_size] = x;
45        flatten(r[x]);
46    }
47
48    void build(int& x, int L, int R)
49    {
50        if (L > R) {
51            x = 0;
52            return;
53        }
54        int mid = (L + R) >> 1;
55        x = buf[mid];
56        build(l[x], L, mid - 1);
57        build(r[x], mid + 1, R);
58        push_up(x);
59    }
60
61    void rebuild(int& x)
```

```
62     {
63         buf_size = 0;
64         flatten(x);
65         build(x, 1, buf_size);
66     }
67
68 void ins(int& rt, T val)
69 {
70     if (!rt) {
71         rt = new_node(val);
72         return;
73     }
74     if (val == v[rt]) {
75         ++cnt[rt];
76     } else if (val < v[rt]) {
77         ins(l[rt], val);
78     } else {
79         ins(r[rt], val);
80     }
81     push_up(rt);
82     if (!balance(rt)) rebuild(rt);
83 }
84
85 void del(int &rt, T val)
86 {
87     if (!rt) return;
88
89     if (val == v[rt]) {
90         if (cnt[rt] -- cnt[rt];
91     } else if (val < v[rt]) {
92         del(l[rt], val);
93     } else {
94         del(r[rt], val);
95     }
96     push_up(rt);
97     if (!balance(rt)) rebuild(rt);
98 }
99
100 int getPrevRank(int rt, T val)
101 {
102     if (!rt) return 0;
103     if (v[rt] == val && cnt[rt]) return sz[l[rt]];
104     if (v[rt] < val) return sz[l[rt]] + cnt[rt] + getPrevRank(r[rt], val);
105     return getPrevRank(l[rt], val);
106 }
107
108 int getSuccRank(int rt, T val)
109 {
110     if (!rt) return 1;
111     if (v[rt] == val && cnt[rt]) return sz[l[rt]] + cnt[rt] + 1;
112     if (v[rt] < val) return sz[l[rt]] + cnt[rt] + getSuccRank(r[rt], val);
113     return getSuccRank(l[rt], val);
114 }
115
116
117 T getKth(int rt, int k)
118 {
119     if (!rt) return 0;
120     if (k <= sz[l[rt]]) return getKth(l[rt], k);
121     if (k - sz[l[rt]] <= cnt[rt]) return v[rt];
122     return getKth(r[rt], k - sz[l[rt]] - cnt[rt]);
123 }
124
125 void ins(T val)
126 {
```

```

127     ins(root, val);
128 }
129
130 void del(T val)
131 {
132     del(root, val);
133 }
134
135 int getRank(T val)
136 {
137     return getPrevRank(root, val) + 1;
138 }
139
140 T getKth(int k)
141 {
142     return getKth(root, k);
143 }
144
145 T getPrev(T val)
146 {
147     return getKth(getPrevRank(root, val));
148 }
149
150 T getSucc(T val)
151 {
152     return getKth(getSuccRank(root, val));
153 }
154
155 void debug(int x)
156 {
157     if (!x) return;
158     debug(l[x]);
159     cerr << v[x] << " ";
160     debug(r[x]);
161 }
162
163 void debug()
164 {
165     cerr << "SGTree:" << endl;
166     debug(root);
167     cerr << endl;
168 }
169 };

```

2 graph

2.1 Dijkstra

```

1 namespace Backlight {
2
3 template<typename T>
4 struct Wraph {
5     struct Edge {
6         int u, v;
7         T w;
8         Edge(){}
9         Edge(int _u, int _v, T _w): u(_u), v(_v), w(_w) {}
10    };
11
12    int V;
13    vector<vector<Edge>> G;
14
15    Wraph() : V(0) {}

```

```

16  Wraph(int _V) : V(_V), G(_V + 1) {}
17
18  inline void addarc(int u, int v, T w) {
19      assert(1 <= u && u <= V);
20      assert(1 <= v && v <= V);
21      G[u].push_back(Edge(u, v, w));
22  }
23
24  inline void addedge(int u, int v, T w) {
25      addarc(u, v, w);
26      addarc(v, u, w);
27  }
28
29  /*****
30  vector<T> dijkstra(int S, T T_MAX) {
31      typedef pair<T, int> Node;
32      priority_queue<Node, vector<Node>, greater<Node>> q;
33      vector<T> dis(V + 1);
34      for (int i = 1; i <= V; i++) dis[i] = T_MAX;
35      dis[S] = 0; q.push(Node(0, S));
36      while (!q.empty()){
37          Node p = q.top(); q.pop();
38          T cost = p.first; int u = p.second;
39          if (dis[u] != cost) continue;
40
41          for (Edge e: G[u]){
42              int v = e.v;
43              T w = e.w;
44              if (dis[v] > dis[u] + w) {
45                  dis[v] = dis[u] + w;
46                  q.push(Node(dis[v], v));
47              }
48          }
49      }
50      return dis;
51  }
52  };
53
54  }

```

3 math

3.1 Lucas

```

1  namespace Backlight {
2
3  // use this when n, m is really large and p is small
4  namespace Lucas {
5      inline ll pow(ll a, ll b, ll p) {
6          ll res = 1;
7          a %= p;
8          while(b) {
9              if (b & 1) res = res * a % p;
10             a = a * a % p;
11             b >>= 1;
12         }
13         return res;
14     }
15
16     inline ll inv1(ll n, ll p) { return pow(n, p - 2, p); }
17
18     inline ll C1(ll n, ll m, ll p) {
19         if (m > n) return 0;

```

```

20     if (m > n - m) m = n - m;
21     ll u = 1, d = 1;
22     for (ll i = 1; i <= m; ++i) {
23         u = u * (n - i + 1) % p;
24         d = d * i % p;
25     }
26     return u * inv1(d, p) % p;
27 }
28
29 // solve n choose m (mod p) while p is a prime
30 ll lucas(ll n, ll m, ll p) {
31     if (m == 0) return 1;
32     return C1(n % p, m % p, p) * lucas(n / p, m / p, p) % p;
33 }
34
35
36 ll exgcd(ll a, ll b, ll& x, ll& y) {
37     if (b == 0) {
38         x = 1; y = 0;
39         return a;
40     }
41     ll d = exgcd(b, a % b, x, y);
42     ll z = x; x = y; y = z - y * (a / b);
43     return d;
44 }
45
46 inline ll inv2(ll n, ll p) {
47     ll x, y;
48     ll d = exgcd(n, p, x, y);
49     return d == 1 ? (p + x % p) % p : -1;
50 }
51
52 // n! mod pk without pi^x
53 ll f(ll n, ll pi, ll pk) {
54     if (!n) return 1;
55     ll res = 1;
56     if (n / pk) {
57         for (ll i = 2; i <= pk; ++i)
58             if (i % pi) res = res * i % pk;
59         res = pow(res, n / pk, pk);
60     }
61     for (ll i = 2; i <= n % pk; ++i)
62         if (i % pi) res = res * i % pk;
63     return res * f(n / pi, pi, pk) % pk;
64 }
65
66 ll C2(ll n, ll m, ll p, ll pi, ll pk) {
67     if (m > n) return 0;
68     ll a = f(n, pi, pk), b = f(m, pi, pk), c = f(n - m, pi, pk);
69     ll k = 0;
70     for (ll i = n; i; i /= pi) k += i / pi;
71     for (ll i = m; i; i /= pi) k -= i / pi;
72     for (ll i = n - m; i; i /= pi) k -= i / pi;
73     ll ans = a * inv2(b, pk) % pk * inv2(c, pk) % pk * pow(pi, k, pk) % pk;
74     ans = ans * (p / pk) % p * inv2(p / pk, pk) % p;
75     return ans;
76 }
77
78 // solve n choose m (mod p) while p might not be a prime
79 ll exlucas(ll n, ll m, ll p) {
80     ll x = p;
81     ll ans = 0;
82     for (ll i = 2; i <= p; ++i) {
83         if (x % i == 0) {
84             ll pk = 1;

```

```

85         while(x % i == 0) pk *= i, x /= i;
86         ans = (ans + C2(n, m, p, i, pk)) % p;
87     }
88 }
89 return ans;
90 }
91
92 } // namespace Lucas
93
94 } // namespace Backlight

```

4 other

4.1 BFPRT

```

1  /**
2   * BFPRT: find the kth element of an array in  $O(n)$  using Divide and Conquer method.
3   * you can use std::nth_element(a, a + k, a + n) instead
4   */
5  namespace BFPRT {
6      template<typename T, typename Cmp>
7      T kth_index(T* a, int l, int r, int k, Cmp cmp);
8
9      template<typename T, typename Cmp>
10     int insert_sort(T* a, int l, int r, Cmp cmp) {
11         for (int i = l + 1; i <= r; ++i) {
12             int tmp = a[i];
13             int j = i - 1;
14             while(j >= l && a[j] > tmp) {
15                 a[j + 1] = a[j];
16                 --j;
17             }
18             a[j + 1] = tmp;
19         }
20         return l + (r - l) / 2;
21     }
22
23     template<typename T, typename Cmp>
24     int pivot(T* a, int l, int r, Cmp cmp) {
25         if (r - l < 5) return insert_sort(a, l, r, cmp);
26         int lst = l - 1;
27         for (int i = l; i + 4 <= r; i += 5) {
28             int p = insert_sort(a, i, i + 4, cmp);
29             swap(a[++lst], a[p]);
30         }
31         return kth_index<T>(a, l, lst, (lst - l + 1) / 2 + 1, cmp);
32     }
33
34     template<typename T, typename Cmp>
35     int partition(T* a, int l, int r, Cmp cmp) {
36         int p = pivot(a, l, r, cmp);
37         swap(a[p], a[r]);
38         int lst = l - 1;
39         for (int i = l; i < r; ++i) {
40             if (cmp(a[i], a[r])) swap(a[++lst], a[i]);
41         }
42         swap(a[++lst], a[r]);
43         return lst;
44     }
45
46     template<typename T, typename Cmp>
47     T kth_index(T* a, int l, int r, int k, Cmp cmp) {
48         int p = partition(a, l, r, cmp);

```



```
49     int d = p - 1 + 1;
50     if (d == k) return p;
51     else if (d < k) return kth_index(a, p + 1, r, k - d, cmp);
52     else return kth_index(a, l, p - 1, k, cmp);
53 }
54
55 template<typename T>
56 T kth_index(T* a, int l, int r, int k) {
57     return kth_index(a, l, r, k, less<T>());
58 }
59 };
```

5 string

5.1 KMP

```
1 namespace KMP {
2     // pi_i = s[0...i] 最长 border
3     void getPi(char* s, int n, int* pi) {
4         pi[0] = 0;
5         for (int i = 1; i < n; ++i) {
6             int j = pi[i - 1];
7             while(j > 0 && s[j] != s[i]) j = pi[j - 1];
8             if (s[i] == s[j]) ++j;
9             pi[i] = j;
10        }
11    }
12
13    vector<int> getAllMatchPosition(char* s, int n, int* pi, char* t, int m) {
14        s[n] = '#'; s[n + 1] = 0; ++n;
15        KMP::getPi(s, n, pi);
16
17        vector<int> ans;
18
19        int p = 0;
20        for (int i = 0; i < m; ++i) {
21            while(p > 0 && t[i] != s[p]) p = pi[p - 1];
22            if (t[i] == s[p]) {
23                ++p;
24                if (p == n - 1) {
25                    ans.push_back(i + 2 - n);
26                }
27            }
28        }
29
30        return ans;
31    }
32
33    int getPeriod(int n, int* pi) {
34        return n - pi[n - 1];
35    }
36 }
```
