

# ACM Template

The event of zero and one

January 14, 2022

Contents

<b>1</b>	<b>String</b>	<b>1</b>
1.1	KMP . . . . .	1
1.2	Trie . . . . .	1
1.3	AC 自动机 . . . . .	1
<b>2</b>	<b>Math</b>	<b>1</b>
2.1	线性筛 . . . . .	1
2.2	欧拉函数 . . . . .	1
2.3	扩展欧几里得 . . . . .	2
2.4	逆元 . . . . .	2
<b>3</b>	<b>Structure</b>	<b>2</b>
3.1	线段树 . . . . .	2
3.2	树链剖分 . . . . .	2
<b>4</b>	<b>Graph</b>	<b>3</b>
4.1	dijkstra . . . . .	3
4.2	spfa . . . . .	3
4.3	prim . . . . .	4
4.4	kruskal . . . . .	4
<b>5</b>	<b>Other</b>	<b>4</b>
5.1	VIM . . . . .	4

# 1 String

## 1.1 KMP

```

1 for(int i = 2, j = 0; i <= n; i++){
2     while(j && p[i] != p[j + 1]) j = ne[j];
3     if(p[i] == p[j + 1]) j++;
4     ne[i] = j;
5 }

```

```

29     int res = 0;
30     for (int i = 0, j = 0; i < s.size(); i ++ ){
31         int t = s[i] - 'a';
32         j = tr[j][t];
33         for(int t = j; t && ~cnt[t]; t = fail[t])
34             res += cnt[t], cnt[t] = - 1;
35     }
36     return res;
37 }
38 }ac;

```

## 1.2 Trie

```

1 int son[N * 26][26], cnt[N * 26], idx;
2 void insert(string s){
3     int p = 0;
4     for(int i = 0; i < s.length(); i++){
5         int u = s[i] - 'a';
6         if(!son[p][u]) son[p][u] = ++ idx;
7         p = son[p][u];
8     }
9     cnt[p]++;
10 }
11 int query(string s){
12     int p = 0;
13     for(int i = 0; i < s.length(); i++){
14         int u = s[i] - 'a';
15         if(!son[p][u]) return 0;
16         p = son[p][u];
17     }
18     return cnt[p];
19 }

```

## 1.3 AC 自动机

```

1 struct AC_Automaton{
2     int tr[N][26], cnt[N], fail[N], idx;
3     void insert(string s){
4         int p = 0;
5         for (int i = 0; i < s.size(); i ++ ){
6             int t = s[i] - 'a';
7             if (!tr[p][t]) tr[p][t] = ++ idx;
8             p = tr[p][t];
9         }
10        cnt[p] ++ ;
11    }
12    void getFail(){
13        queue<int> q;
14        for(int i = 0; i < 26; i ++ )
15            if(tr[0][i]) q.push(tr[0][i]);
16        while(q.size()){
17            int t = q.front(); q.pop();
18            for(int i = 0; i < 26; i ++ ){
19                int p = tr[t][i];
20                if(!p) tr[t][i] = tr[fail[t]][i];
21                else{
22                    fail[p] = tr[fail[t]][i];
23                    q.push(p);
24                }
25            }
26        }
27    }
28    int query(string s){

```

# 2 Math

## 2.1 线性筛

```

1 void getPrimes(int n){
2     for(int i = 2; i <= n; i++){
3         if(!st[i]) p[cnt++] = i;
4         for(int j = 0; p[j] <= n / i; j++){
5             st[p[j] * i] = 1;
6             if(i % p[j] == 0) break;
7         }
8     }
9 }

```

## 2.2 欧拉函数

$$\phi(n) = n \prod_{i=1}^m (1 - \frac{1}{p_i})$$

```

1 int phi(int x){
2     int res = x;
3     for (int i = 2; i <= x / i; i ++ ){
4         if (x % i == 0){
5             res = res / i * (i - 1);
6             while (x % i == 0) x /= i;
7         }
8     }
9     if (x > 1) res = res / x * (x - 1);
10    return res;
11 }
12
13 void get_phi(int n){
14     phi[1] = 1;
15     for(int i = 2; i <= n; i++){
16         if(!st[i]){
17             p[cnt++] = i;
18             phi[i] = i - 1;
19         }
20         for(int j = 0; p[j] <= n / i; j++){
21             int t = p[j] * i;
22             st[t] = true;
23             if(i % p[j] == 0){
24                 phi[t] = phi[i] * p[j];
25                 break;
26             }
27             phi[t] = phi[i] * (p[j] - 1);
28         }
29     }
30 }

```

## 2.3 扩展欧几里得

```

1 int exgcd(int a, int b, int& x, int& y){
2     if(!b){
3         x = 1, y = 0; return a;
4     }
5     int d = exgcd(b, a % b, y, x);
6     y -= a / b * x;
7     return d;
8 }

```

## 2.4 逆元

### 费马小定理

若  $\gcd(a, p) = 1$ , 且  $p$  为质数, 则有  $\text{inv}(a) \equiv a^{p-2} \pmod{p}$

## 3 Structure

### 3.1 线段树

```

1 struct SegmentTree {
2     //modify1 : mul
3     //modify2 : add
4     struct node{
5         int l, r;
6         LL mul, add, sum;
7     }tr[N << 2];
8     void pushup(int u){
9         tr[u].sum = tr[u << 1].sum + tr[u << 1 | 1].sum;
10    }
11    void pushdown(int u){
12        auto &root = tr[u];
13        auto &left = tr[u << 1];
14        auto &right = tr[u << 1 | 1];
15        if(root.mul != 1){
16            left.mul *= root.mul;
17            left.add *= root.mul;
18            left.sum *= root.mul;
19            right.mul *= root.mul;
20            right.add *= root.mul;
21            right.sum *= root.mul;
22            root.mul = 1;
23        }
24        if(root.add){
25            left.add += root.add;
26            left.sum += (left.r - left.l + 1) * root.add;
27            right.add += root.add;
28            right.sum += (right.r - right.l + 1) * root.add;
29            root.add = 0;
30        }
31    }
32    void build(int u, int l, int r){
33        tr[u] = {l, r, 1, 0, w[r]};
34        if(l == r) return;
35        int mid = l + r >> 1;
36        build(u << 1, l, mid), build(u << 1 | 1, mid + 1, r);
37        pushup(u);
38    }

```

```

39 void modify1(int u, int l, int r, int k){
40     if(l <= tr[u].l && tr[u].r <= r){
41         tr[u].mul *= k;
42         tr[u].add *= k;
43         tr[u].sum *= k;
44         return;
45     }
46     else{
47         pushdown(u);
48         int mid = tr[u].l + tr[u].r >> 1;
49         if(l <= mid) modify1(u << 1, l, r, k);
50         if(r > mid) modify1(u << 1 | 1, l, r, k);
51         pushup(u);
52     }
53 }
54 void modify2(int u, int l, int r, int k){
55     if(l <= tr[u].l && tr[u].r <= r){
56         tr[u].add += k;
57         tr[u].sum += (tr[u].r - tr[u].l + 1) * k;
58         return;
59     }
60     else{
61         pushdown(u);
62         int mid = tr[u].l + tr[u].r >> 1;
63         if(l <= mid) modify2(u << 1, l, r, k);
64         if(r > mid) modify2(u << 1 | 1, l, r, k);
65         pushup(u);
66     }
67 }
68 LL query(int u, int l, int r){
69     if(l <= tr[u].l && tr[u].r <= r) return tr[u].sum;
70     pushdown(u);
71     int mid = tr[u].l + tr[u].r >> 1;
72     LL res = 0;
73     if(l <= mid) res += query(u << 1, l, r);
74     if(r > mid) res += query(u << 1 | 1, l, r);
75     return res;
76 }
77 }t;

```

### 3.2 树链剖分

```

1 const int N = 1e5 + 10, M = 2 * N;
2 int n, m;
3 int w[N], e[M], ne[M], h[N], idx;
4 int id[N], nw[N], cnt;
5 int top[N], fa[N], sz[N], son[N], dep[N];
6 void add(int a, int b){
7     e[idx] = b, ne[idx] = h[a], h[a] = idx++;
8 }
9 void dfs1(int u, int father, int depth){
10     dep[u] = depth, fa[u] = father, sz[u] = 1;
11     for(int i = h[u]; ~i; i = ne[i]){
12         int j = e[i];
13         if(j == father) continue;
14         dfs1(j, u, depth + 1);
15         sz[u] += sz[j];
16         if(sz[son[u]] < sz[j]) son[u] = j;
17     }
18 }
19 void dfs2(int u, int t){
20     id[u] = ++cnt, nw[cnt] = w[u], top[u] = t;
21     if(!son[u]) return;

```

```

22     dfs2(son[u], t);
23     for(int i = h[u]; ~i; i = ne[i]){
24         int j = e[i];
25         if(j == fa[u] || j == son[u]) continue;
26         dfs2(j, j);
27     }
28 }
29 struct node{
30     int l, r;
31     int add, sum;
32 }tr[N * 4];
33 void pushup(int u){
34     tr[u].sum = tr[u << 1].sum + tr[u << 1 | 1].sum;
35 }
36 void pushdown(int u){
37     auto &root = tr[u];
38     auto &left = tr[u << 1], &right = tr[u << 1 | 1];
39     if(root.add){
40         left.add += root.add;
41         left.sum += (left.r - left.l + 1) * root.add;
42         right.add += root.add;
43         right.sum += (right.r - right.l + 1) * root.add;
44         root.add = 0;
45     }
46 }
47 void build(int u, int l, int r){
48     tr[u] = {l, r, 0, nw[r]};
49     if(l == r) return;
50     int mid = l + r >> 1;
51     build(u << 1, l, mid), build(u << 1 | 1, mid + 1,
52         r);
53     pushup(u);
54 }
55 void modify(int u, int l, int r, int k){
56     if(l <= tr[u].l && tr[u].r <= r){
57         tr[u].add += k;
58         tr[u].sum += (tr[u].r - tr[u].l + 1) * k;
59         return;
60     }
61     else{
62         pushdown(u);
63         int mid = tr[u].l + tr[u].r >> 1;
64         if(l <= mid) modify(u << 1, l, r, k);
65         if(r > mid) modify(u << 1 | 1, l, r, k);
66         pushup(u);
67     }
68 }
69 void modify_path(int u, int v, int k){
70     while(top[u] != top[v]){
71         if(dep[top[u]] < dep[top[v]]) swap(u, v);
72         modify(1, id[top[u]], id[u], k);
73         u = fa[top[u]];
74     }
75     if(dep[u] < dep[v]) swap(u, v);
76     modify(1, id[v], id[u], k);
77 }
78 void modify_tree(int u, int k){
79     modify(1, id[u], id[u] + sz[u] - 1, k);
80 }
81 int query(int u, int l, int r){
82     if(l <= tr[u].l && tr[u].r <= r) return tr[u].sum;
83     pushdown(u);
84     int mid = tr[u].l + tr[u].r >> 1;
85     int res = 0;

```

```

85     if(l <= mid) res += query(u << 1, l, r);
86     if(r > mid) res += query(u << 1 | 1, l, r);
87     return res;
88 }
89 int query_path(int u, int v){
90     int res = 0;
91     while(top[u] != top[v]){
92         if(dep[top[u]] < dep[top[v]]) swap(u, v);
93         res += query(1, id[top[u]], id[u]);
94         u = fa[top[u]];
95     }
96     if(dep[u] < dep[v]) swap(u, v);
97     res += query(1, id[v], id[u]);
98     return res;
99 }
100 int query_tree(int u){
101     return query(1, id[u], id[u] + sz[u] - 1);
102 }

```

## 4 Graph

### 4.1 dijkstra

```

1 int dijkstra(){
2     memset(dist, 0x3f, sizeof dist);
3     priority_queue<PII, vector<PII>, greater<PII>> heap;
4     heap.push({0, 1});
5     while(heap.size()){
6         auto t = heap.top(); heap.pop();
7         int ver = t.second, distance = t.first;
8         if(st[ver]) continue;
9         st[ver] = 1;
10        for(int i = h[ver]; i != -1; i = ne[i]){
11            int j = e[i];
12            if(distance + w[i] < dist[j]){
13                dist[j] = distance + w[i];
14                heap.push({dist[j], j});
15            }
16        }
17    }
18    if(dist[n] == 0x3f3f3f3f) return -1;
19    else return dist[n];
20 }

```

### 4.2 spfa

```

1 bool spfa(){
2     queue<int> q;
3     for(int i = 1; i <= n; i++){
4         st[i] = 1, q.push(i);
5     }
6     while(q.size()){
7         int t = q.front(); q.pop(); st[t] = 0;
8         for(int i = h[t]; i != -1; i = ne[i]){
9             int j = e[i];
10            if(dist[j] > dist[t] + w[i]){
11                dist[j] = dist[t] + w[i];
12                cnt[j] = cnt[t] + 1;
13                if(cnt[j] >= n) return true;
14                if(!st[j]){
15                    q.push(j); st[j] = 1;
16                }

```

```
17     }
18     }
19 }
20 return false;
21 }
```

### 4.3 prim

```
1 int prim(){
2     memset(dist, 0x3f, sizeof dist);
3     int res = 0; dist[1] = 0;
4     for(int i = 0; i < n; i ++){
5         int t = -1;
6         for(int j = 1; j <= n; j ++){
7             if(!st[j] && (t == -1 || dist[t] > dist[j]))
8                 t = j;
9         }
10        if(dist[t] == 0x3f3f3f3f) return 0x3f3f3f3f;
11        res += dist[t];
12        st[t] = true;
13        for(int j = 1; j <= n; j ++){
14            dist[j] = min(dist[j], g[t][j]);
15        }
16    }
17    return res;
18 }
```

### 4.4 kruskal

```
1 int kruskal(){
2     sort(arr, arr + m, cmp);
3     int res = 0, cnt = 1;
4     for(int i = 0; i < m; i ++){
5         if(find(arr[i].a) == find(arr[i].b)) continue;
6         merge(arr[i].a, arr[i].b);
7         cnt ++; res += arr[i].c;
8     }
9     if(cnt < n) return 0x3f3f3f3f;
10    else return res;
11 }
```

## 5 Other

### 5.1 VIM

```
1 syntax on
2 set nu
3 set tabstop=4
4 set shiftwidth=4
5 set cin
6 colo evening
7 set mouse=a
```