ACM Template

The event of zero and one

February 28, 2022

Contents

1 String

1.1 KMP

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

1.2 Manacher

```
int manacher(string str){
    int len = str.length();
    vector<char> s(2 * len + 100);
    vector<int> d(2 * len + 100);
    int n = 2 * len + 1, res = 1;
    for(int i = 0; i < len; i ++)
        s[2 * i + 1] = str[i];
    for(int i = 0, l = 0, r = -1; i < n; i ++) {
        int j = l + r - i;
        d[i] = max(min(d[j], j - l + 1), 0);
        if (j - d[j] < l) {
            while (i - d[i] >= 0 && i + d[i] < n && s[i
                 - d[i]] == s[i + d[i]])
                d[i]++;
            l = i - d[i] + 1, r = i + d[i] - 1;
        }
        res = max(res, d[i]);
    }
    return res - 1;
}
```

1.3 Trie

```
int son[N * 26][26], cnt[N * 26], idx;
void insert(string s){
    int p = 0;
    for(int i = 0; i < s.length(); i ++){
        int u = s[i] - 'a';
        if(!son[p][u]) son[p][u] = ++ idx;
        p = son[p][u];
    }
    cnt[p] ++;
}
int query(string s){
    int p = 0;
    for(int i = 0; i < s.length(); i ++){
        int u = s[i] - 'a';
        if(!son[p][u]) return 0;
        p = son[p][u];
    }
    return cnt[p];
}
```

1.4 AC-Automation

```
struct AC_Automaton{
    int tr[N][26], cnt[N],fail[N], idx;
    void insert(string s){
        int p = 0;
        for (int i = 0; i < s.size(); i ++ ){
            int t = s[i] - 'a';
            if (!tr[p][t]) tr[p][t] = ++ idx;
            p = tr[p][t];
        }
        cnt[p] ++ ;
    }
    void getFail(){
        queue<int> q;
        for(int i = 0; i < 26; i ++)
            if(tr[0][i]) q.push(tr[0][i]);
        while(q.size()){
            int t = q.front(); q.pop();
            for(int i = 0; i < 26; i ++){
                int p = tr[t][i];
                if(!p) tr[t][i] = tr[fail[t]][i];
                else{
                    fail[p] = tr[fail[t]][i];
                    q.push(p);
                }
            }
        }
    }
    int query(string s){
        int res = 0;
        for (int i = 0, j = 0; i < s.size(); i ++ ){
            int t = s[i] - 'a';
            j = tr[j][t];
            for(int t = j; t && ~cnt[t]; t = fail[t])
                res += cnt[t], cnt[t] = - 1;
        }
        return res;
    }
}ac;
```

2 Math

2.1 线性筛

```
void getPrimes(int n){
    for(int i = 2; i <= n; i ++){
        if(!st[i]) p[cnt ++] = i;
        for(int j = 0; p[j] <= n / i; j ++){
            st[p[j] * i] = 1;
            if(i % p[j] == 0) break;
        }
    }
}
```

2.2 欧拉函数

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

```
int phi(int x){
    int res = x;
    for (int i = 2; i <= x / i; i ++ ){
        if (x % i == 0){
            res = res / i * (i - 1);
            while (x % i == 0) x /= i;
        }
    }
```

```
   if (x > 1) res = res / x * (x - 1);
   return res;
}

void get_phi(int n){
   phi[1] = 1;
   for(int i = 2; i <= n; i ++){
      if(!st[i]){
         p[cnt ++] = i;
         phi[i] = i - 1;
      }
      for(int j = 0; p[j] <= n / i; j ++){
         int t = p[j] * i;
         st[t] = true;
         if(i % p[j] == 0){
            phi[t] = phi[i] * p[j];
            break;
         }
         phi[t] = phi[i] * (p[j] - 1);
      }
   }
}
```

2.3 扩展欧几里得

```
int exgcd(int a, int b, int& x, int& y){
   if(!b){
      x = 1, y = 0; return a;
   }
   int d = exgcd(b, a % b, y, x);
   y -= a / b * x;
   return d;
}
```

2.4 逆元

费马小定理

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

3 Structure

3.1 DSU

```
struct UF {
   vector<int> fa, sz;
   int n, cnt;
   UF(int x): n(x), cnt(x), fa(x), sz(x, 1) {
      iota(fa.begin(), fa.end(), 0);
   }
   int find(int x) {
      return fa[x] == x ? x : (fa[x] = find(fa[x]));
   }
   bool merge(int x, int y) {
      x = find(x), y = find(y);
      if (x == y) return false;
      if (sz[x] < sz[y]) swap(x, y);
      fa[y] = x; sz[x] += sz[y];
      --cnt;
      return true;
   }
   bool same(int x, int y) {return find(x) == find(y);}
};
```

3.2 Fenwick

```
struct Fenwick {
   const int n;
   vector<int> a;
   Fenwick(int n) : n(n), a(n) {}
   void add(int x, int v) {
      for (int i = x + 1; i <= n; i += i & -i) {
         a[i - 1] += v;
      }
   }
   int sum(int x) {
      T ans = 0;
      for (int i = x; i > 0; i -= i & -i) {
         ans += a[i - 1];
      }
      return ans;
   }
   int rangeSum(int l, int r) {
      return sum(r) - sum(l);
   }
};
```

3.3 SegmentTree

```
struct SegmentTree {
   //modify1 : mul
   //modify2 : add
   struct node{
      int l, r;
      LL mul, add, sum;
   }tr[N << 2];
   void pushup(int u){
      tr[u].sum = tr[u << 1].sum + tr[u << 1 | 1].sum;
   }
   void pushdown(int u){
      auto &root = tr[u];
      auto &left = tr[u << 1];
      auto &right = tr[u << 1 | 1];
      if(root.mul != 1){
         left.mul *= root.mul;
         left.add *= root.mul;
         left.sum *= root.mul;
         right.mul *= root.mul;
         right.add *= root.mul;
         right.sum *= root.mul;
         root.mul = 1;
      }
      if(root.add){
         left.add += root.add;
         left.sum += (left.r - left.l + 1) * root.add;
         right.add += root.add;
         right.sum += (right.r - right.l + 1) * root.add;
         root.add = 0;
      }
   }
   void build(int u, int l, int r){
      tr[u] = {l, r, 1, 0, w[r]};
      if(l == r) return;
      int mid = l + r >> 1;
      build(u << 1, l, mid), build(u << 1 | 1, mid + 1, r);
      pushup(u);
   }
   void modify1(int u, int l, int r, int k){
      if(l <= tr[u].l && tr[u].r <= r){
         tr[u].mul *= k;
         tr[u].add *= k;
         tr[u].sum *= k;
         return;
      }
      else{
         pushdown(u);
         int mid = tr[u].l + tr[u].r >> 1;
         if(l <= mid) modify1(u << 1, l, r, k);
         if(r > mid) modify1(u << 1 | 1, l, r, k);
         pushup(u);
      }
   }
   void modify2(int u, int l, int r, int k){
      if(l <= tr[u].l && tr[u].r <= r){
         tr[u].add += k;
         tr[u].sum += (tr[u].r - tr[u].l + 1) * k;
         return;
      }
      else{
         pushdown(u);
         int mid = tr[u].l + tr[u].r >> 1;
         if(l <= mid) modify2(u << 1, l, r, k);
         if(r > mid) modify2(u << 1 | 1, l, r, k);
         pushup(u);
      }
   }
   LL query(int u, int l, int r){
      if(l <= tr[u].l && tr[u].r <= r) return tr[u].sum;
      pushdown(u);
      int mid = tr[u].l + tr[u].r >> 1;
      LL res = 0;
      if(l <= mid) res += query(u << 1, l, r);
      if(r > mid) res += query(u << 1 | 1, l, r);
      return res;
   }
}t;
```

3.4 树链剖分

```
const int N = 1e5 + 10, M = 2 * N;
int n, m;
int w[N], e[M], ne[M], h[N], idx;
int id[N], nw[N], cnt;
int top[N], fa[N], sz[N], son[N], dep[N];
void add(int a, int b){
   e[idx] = b, ne[idx] = h[a], h[a] = idx ++;
}
void dfs1(int u, int father, int depth){
   dep[u] = depth, fa[u] = father, sz[u] = 1;
   for(int i = h[u]; ~i; i = ne[i]){
      int j = e[i];
      if(j == father) continue;
      dfs1(j, u, depth + 1);
      sz[u] += sz[j];
      if(sz[son[u]] < sz[j]) son[u] = j;
   }
}
void dfs2(int u, int t){
   id[u] = ++ cnt, nw[cnt] = w[u], top[u] = t;
   if(!son[u]) return;
   dfs2(son[u], t);
   for(int i = h[u]; ~i; i = ne[i]){
      int j = e[i];
      if(j == fa[u] || j == son[u]) continue;
      dfs2(j, j);
   }
}
struct node{
   int l, r;
   int add, sum;
}tr[N * 4];
void pushup(int u){
   tr[u].sum = tr[u << 1].sum + tr[u << 1 | 1].sum;
}
void pushdown(int u){
   auto &root = tr[u];
   auto &left = tr[u << 1], &right = tr[u << 1 | 1];
   if(root.add){
      left.add += root.add;
      left.sum += (left.r - left.l + 1) * root.add;
      right.add += root.add;
      right.sum += (right.r - right.l + 1) * root.add;
      root.add = 0;
   }
}
void build(int u, int l, int r){
   tr[u] = {l, r, 0, nw[r]};
   if(l == r) return;
   int mid = l + r >> 1;
   build(u << 1, l, mid), build(u << 1 | 1, mid + 1, r);
   pushup(u);
}
void modify(int u, int l, int r, int k){
   if(l <= tr[u].l && tr[u].r <= r){
      tr[u].add += k;
      tr[u].sum += (tr[u].r - tr[u].l + 1) * k;
      return;
   }
   else{
      pushdown(u);
      int mid = tr[u].l + tr[u].r >> 1;
      if(l <= mid) modify(u << 1, l, r, k);
      if(r > mid) modify(u << 1 | 1, l, r, k);
      pushup(u);
   }
}
void modify_path(int u, int v, int k){
   while(top[u] != top[v]){
      if(dep[top[u]] < dep[top[v]]) swap(u, v);
      modify(1, id[top[u]], id[u], k);
      u = fa[top[u]];
   }
   if(dep[u] < dep[v]) swap(u, v);
   modify(1, id[v], id[u], k);
}
void modify_tree(int u, int k){
    modify(1, id[u], id[u] + sz[u] - 1, k);
}
int query(int u, int l, int r){
    if(l <= tr[u].l && tr[u].r <= r) return tr[u].sum;
    pushdown(u);
    int mid = tr[u].l + tr[u].r >> 1;
    int res = 0;
    if(l <= mid) res += query(u << 1, l, r);
    if(r > mid) res += query(u << 1 | 1, l, r);
    return res;
}
int query_path(int u, int v){
    int res = 0;
    while(top[u] != top[v]){
        if(dep[top[u]] < dep[top[v]]) swap(u, v);
        res += query(1, id[top[u]], id[u]);
        u = fa[top[u]];
    }
    if(dep[u] < dep[v]) swap(u, v);
    res += query(1, id[v], id[u]);
    return res;
}
int query_tree(int u){
    return query(1, id[u], id[u] + sz[u] - 1);
}
```

4 Graph

4.1 Dijkstra

```
int dijkstra(){
    memset(dist, 0x3f, sizeof dist);
    priority_queue<PII,vector<PII>,greater<PII>> heap;
    heap.push({0, 1});
    while(heap.size()){
        auto t = heap.top(); heap.pop();
        int ver = t.second, distance = t.first;
        if(st[ver]) continue;
        st[ver] = 1;
        for(int i = h[ver]; i != -1; i = ne[i]){
            int j = e[i];
            if(distance + w[i] < dist[j]){
                dist[j] = distance + w[i];
                heap.push({dist[j], j});
            }
        }
    }
    if(dist[n] == 0x3f3f3f3f) return -1;
    else return dist[n];
}
```

4.2 Spfa

```
bool spfa(){
    queue<int> q;
    for(int i = 1; i <= n; i ++){
        st[i] = 1, q.push(i);
    }
    while(q.size()){
        int t = q.front(); q.pop(); st[t] = 0;
        for(int i = h[t]; i != -1; i = ne[i]){
            int j = e[i];
            if(dist[j] > dist[t] + w[i]){
                dist[j] = dist[t] + w[i];
                cnt[j] = cnt[t] + 1;
                if(cnt[j] >= n) return true;
                if(!st[j]){
                    q.push(j); st[j] = 1;
                }
            }
        }
    }
    return false;
}
```

4.3 Prim

```
int prim(){
    memset(dist, 0x3f, sizeof dist);
    int res = 0; dist[1] = 0;
    for(int i = 0; i < n; i ++){
        int t = -1;
        for(int j = 1; j <= n; j ++){
            if(!st[j] && (t == -1 || dist[t] > dist[j])
                ) t = j;
        }
        if(dist[t] == 0x3f3f3f3f) return 0x3f3f3f3f;
        res += dist[t];
        st[t] = true;
        for(int j = 1; j <= n; j ++){
            dist[j] = min(dist[j], g[t][j]);
        }
    }
    return res;
}
```

4.4 Kruskal

```
int kruskal(){
    sort(arr, arr + m, cmp);
    int res = 0, cnt = 1;
    for(int i = 0; i < m; i ++){
        if(find(arr[i].a) == find(arr[i].b)) continue;
        merge(arr[i].a, arr[i].b);
        cnt ++; res += arr[i].c;
    }
    if(cnt < n) return 0x3f3f3f3f;
    else return res;
}
```

4.5 LCA

```
void bfs(int root){
    memset(dep, 0x3f, sizeof dep);
    queue<int> q;
    dep[0] = 0, dep[root] = 1;
    q.push(root);
    while(q.size()){
        int t = q.front(); q.pop();
        for(int i = h[t]; ~i; i = ne[i]){
         int j = e[i];
         if(dep[j] > dep[t] + 1){
            dep[j] = dep[t] + 1;
            fa[j][0] = t;
            q.push(j);
            for(int k = 1; k <= 15; k ++)
               fa[j][k] = fa[fa[j][k - 1]][k - 1];
         }
      }
   }
}
int lca(int a, int b){
   if(dep[a] < dep[b]) swap(a, b);
   for(int k = 15; k >= 0; k --){
      if(dep[fa[a][k]] >= dep[b])
         a = fa[a][k];
   }
   if(a == b) return a;
   for(int k = 15; k >= 0; k --){
      if(fa[a][k] != fa[b][k])
         a = fa[a][k], b = fa[b][k];
   }
   return fa[a][0];
}
```

5 DP

5.1 LIS

```
int LIS(vector<int>& a){
   vector<int> stk; int n = a.size();
   stk.push_back(a[0]);
   for(int i = 1; i < n; i ++){
      if(a[i] > s1.back()) stk.push_back(a[i]);
      else *lower_bound(all(stk), a[i]) = a[i];
   }
   return stk.size();
}
```

6 Other

6.1 离散化

```
sort(all(v))
v.erase(unique(all(v)), v.end());
int find(VI& v, int& x){
   return lower_bound(all(v), x) - v.begin();
}
```

6.2 VIM

```
   syntax on
   set nu
   set tabstop=4
   set shiftwidth=4
   set cin
   colo evening
   set mouse=a
```