Standard Code Library

ONGLU

North Eastern University

August 2021

Contents

初始化

数据结构

数学

图论

计算几何

字符串

字串哈希

```
namespace String {
    const int x = 135;
    const int p1 = 1e9 + 7, p2 = 1e9 + 9;
    ull xp1[N], xp2[N], xp[N];
    void init_xp() {
        xp1[0] = xp2[0] = xp[0] = 1;
        for(int i = 1; i < N; i++) {
            xp1[i] = xp1[i - 1] * x % p1;
            xp2[i] = xp2[i - 1] * x % p2;
            xp[i] = xp[i - 1] * x;
        }
    }
    struct HashString {
        char s[N];
        int length, subsize;
        bool sorted;
        ull h[N], hl[N];
        ull init(const char *t) {
            if(xp[0] != 1) init_xp();
            length = strlen(t);
            strcpy(s, t);
            ull res1 = 0, res2 = 0;
            h[length] = 0;
            for(int j = length - 1; j >= 0; j--) {
            #ifdef ENABLE_DOUBLE_HASH
                res1 = (res1 * x + s[j]) % p1;
                res2 = (res2 * x + s[j]) % p2;
                h[j] = (res1 << 32) | res2;
            #else
                res1 = res1 * x + s[j];
                h[j] = res1;
            #endif
            }
            return h[0];
        }
        //获取子串哈希，左闭右开
        ull get_substring_hash(int left, int right)  {
            int len = right - left;
        #ifdef ENABLE_DOUBLE_HASH
            unsigned int mask32 = ~(0u);
            ull left1 = h[left] >> 32, right1 = h[right] >> 32;
            ull left2 = h[left] & mask32, right2 = h[right] & mask32;
            return (((left1 - right1 * xp1[len] % p1 + p1) % p1) << 32) |
                   (((left2 - right2 * xp2[len] % p2 + p2) % p2));
        #else
            return h[left] - h[right] * xp[len];
        #endif
        }
        void get_all_subs_hash(int sublen) {
            subsize = length - sublen + 1;
            for (int i = 0; i < subsize; ++i)
                hl[i] = get_substring_hash(i, i + sublen);
            sorted = 0;
        }

        void sort_substring_hash() {
            sort(hl, hl + subsize);
            sorted = 1;
        }

        bool match(ull key) const {
            if (!sorted) assert (0);
            if (!subsize) return false;
            return binary_search(hl, hl + subsize, key);
        }
    };
}
```

Trie

```
namespace trie {
    int t[N][26], sz, ed[N];
    int _new() {
        sz++;
        memset(t[sz], 0, sizeof(t[sz]));
        return sz;
    }
    void init() {
        sz = 0;
        _new();
        memset(ed, 0, sizeof(ed));
    }
    void Insert(char *s, int n) {
        int u = 1;
        for(int i = 0; i < n; i++) {
            int c = s[i] - 'a';
            if(!t[u][c]) t[u][c] = _new();
            u = t[u][c];
        }
        ed[u]++;
    }
    int find(char *s, int n) {
        int u = 1;
        for(int i = 0; i < n; i++) {
            int c = s[i] - 'a';
            if(!t[u][c]) return -1;
            u = t[u][c];
        }
        return u;
    }
}
```

KMP 算法

```
namespace KMP {
    void get_next(char *t, int m, int *nxt) {
        int j = nxt[0] = 0;
        for(int i = 1; i < m; i++) {
            while(j && t[i] != t[j]) j = nxt[j - 1];
            nxt[i] = j += (t[i] == t[j]);
        }
    }
    vector<int> find(char *t, int m, int *nxt, char *s, int n) {
        vector<int> ans;
        int j = 0;
        for(int i = 0; i < n; i++) {
            while(j && s[i] != t[j]) j = nxt[j - 1];
            j += s[i] == t[j];
            if(j == m) {
                ans.push_back(i - m + 1);
                j = nxt[j - 1];
            }
        }
        return ans;
    }
}
```

manacher 算法

```
namespace manacher {
    char s[N];
    int p[N], len;
    void getp(string tmp) {
        len = 0;
        for(auto x : tmp) {
            s[len++] = '#';
            s[len++] = x;
        }
        s[len++] = '#';
        memset(p, 0, sizeof(int) * (len + 10));
        int c = 0, r = 0;
        for(int i = 0; i < len; i++) {
            if(i <= r) p[i] = min(p[2 * c - i], r - i);
            else p[i] = 1;
            while(i - p[i] >= 0 && i + p[i] < len && s[i - p[i]] == s[i + p[i]])
                p[i]++;
            if(i + p[i] - 1 > r) {
                r = i + p[i] - 1;
                c = i;
            }
        }
        for(int i = 0; i < len; i++) p[i]--;
    }
    void getp(char *tmp, int n) {
        len = 0;
        for(int i = 0; i < n; i++) {
            s[len++] = '#';
            s[len++] = tmp[i];
        }
        s[len++] = '#';
        memset(p, 0, sizeof(int) * (len + 10));
        int c = 0, r = 0;
        for(int i = 0; i < len; i++) {
            if(i <= r) p[i] = min(p[2 * c - i], r - i);
            else p[i] = 1;
            while(i - p[i] >= 0 && i + p[i] < len && s[i - p[i]] == s[i + p[i]])
                p[i]++;
            if(i + p[i] - 1 > r) {
                r = i + p[i] - 1;
                c = i;
            }
        }
        for(int i = 0; i < len; i++) p[i]--;
    }
    int getlen() {
        return *max_element(p, p + len);
    }
    int getlen(string s) {
        getp(s);
        return getlen();
    }
}
```

AC 自动机

```
struct ac_automaton {
    int t[N][4], danger[N], tot, fail[N];
    int dp[N][N];
    void init() {
        tot = -1;
        _new();
    }
    int _new() {
        tot++;
        memset(t[tot], 0, sizeof(t[tot]));
        danger[tot] = 0;
        fail[tot] = 0;
        return tot;
    }
    void Insert(const char *s) {
        int u = 0;
        for(int i = 0; s[i]; i++) {
            if(!t[u][mp[s[i]]]) t[u][mp[s[i]]] = _new();
            u = t[u][mp[s[i]]];
        }
        danger[u] = 1;
    }
    void build() {
        queue<int> q;
        for(int i = 0; i < 4; i++) {
            if(t[0][i]) {
                fail[i] = 0;
                q.push(t[0][i]);
            }
        }
        while(q.size()) {
            int u = q.front(); q.pop();
            danger[u] |= danger[fail[u]];
            for(int i = 0; i < 4; i++) {
                if(t[u][i]) {
                    fail[t[u][i]] = t[fail[u]][i];
                    q.push(t[u][i]);
                } else t[u][i] = t[fail[u]][i];
            }
        }
    }
    int query(const char *s) {
        memset(dp, 0x3f, sizeof(dp));
        int n = strlen(s);
        dp[0][0] = 0;
        for(int i = 0; i < n; i++) {
            for(int j = 0; j <= tot; j++) if(!danger[j]) {
                for(int k = 0; k < 4; k++) if(!danger[t[j][k]]) {
                    dp[i + 1][t[j][k]] = min(dp[i + 1][t[j][k]], dp[i][j] + (mp[s[i]] != k));
                }
            }
        }
        int ans = 0x3f3f3f3f;
        for(int i = 0; i <= tot; i++) if(!danger[i]) {
            ans = min(ans, dp[n][i]);
        }
        return ans == 0x3f3f3f3f ? -1 : ans;
    }
};
```

杂项

int128

```
typedef __uint128_t u128;
inline u128 read() {
    static char buf[100];
    scanf("%s", buf);
    // std::cin >> buf;
    u128 res = 0;
    for(int i = 0;buf[i];++i) {
        res = res << 4 | (buf[i] <= '9' ? buf[i] - '0' : buf[i] - 'a' + 10);
    }
    return res;
}
inline void output(u128 res) {
    if(res >= 16)
```

```
        output(res / 16);
    putchar(res % 16 >= 10 ? 'a' + res % 16 - 10 : '0' + res % 16);
    //std::cout.put(res % 16 >= 10 ? 'a' + res % 16 - 10 : '0' + res % 16);
}
```