字符串

```
/**
 * 常规字符串处理     STRING
 * KMP                KMP
 * 字符串哈希         hash
 * trie       字典树 trie
 * manacher 马拉车 manacher
 * 序列自动机    fakeac
*/

#include <vector>
#include <iostream>
#include <algorithm>
#include <cstring>
using namespace std;

namespace golitter {
namespace STRING {
    /**
     * 字符串输入
    */
void String_Input() {
    // 1. string
    string str; getline(cin,str); // 一行
    // 2. 从下标1开始
    char ph[33]; cin>>ph + 1;
}
    /**
     * 子字符串
    */
void subString() {
    string str,sub; int pos, length;
    // 获取字串 从str的pos下标开始获取，子字符串的长度为length
    sub = str.substr(pos,length);
    // 获取substring在string中存在的下标位置，如果不存在为-1
    int pos = str.find(sub); // 注意：如果直接判断 str.find() == ... ，可能错误，因为
str.find返回为size_t，没有进行隐式转换。
}
    /**
     * 字符串转换
    */
void String_Transform() {
    string str;
    // 全部转换为大写字母
    transform(str.begin(), str.end(),str.begin(), ::toupper);
    // 全部转换为小写字母
    transform(str.begin(), str.end(), str.begin(), ::tolower);
    // 翻转
    reverse(str.begin(), str.end());
    /**
```

```
     * 字符串和数字之间的转换
     */
    int i; double d; float f;
    // 数字转字符串 -- to_string
    str = to_string(i); str = to_string(d); str = to_string(f);
    // 字符串转数字 -- sto / 数据类型前缀
    i = stoi(str); d = stod(str); f = stof(str);
    long long ll = stoll(str);
}
}}

namespace golitter {
namespace KMP { // https://ac.nowcoder.com/acm/contest/57358/A
                // https://oi-wiki.org/string/kmp/
/**
 *
 * 给定一个长度为 n 的字符串 s，其 前缀函数 被定义为一个长度为 n 的数组 nxt。 其中 nxt[i] 的定义是：
* 如果子串 s[i] 有一对相等的真前缀与真后缀：s[k-1] 和 s[i - (k - 1)i]，那么 nxt[i] 就是这个相等的真前缀（或者真后缀，因为它们相等））的长度，也就是 nxt[i]=k；
* 如果不止有一对相等的，那么 nxt[i] 就是其中最长的那一对的长度；
* 如果没有相等的，那么 nxt[i]=0。
* 简单来说 nxt[i] 就是，子串 s[i] 最长的相等的真前缀与真后缀的长度。
*/
vector<int> prefix_function(string s) {
    int n = (int)s.length();
    vector<int> nxt(n);
    for (int i = 1; i < n; i++) {
        int j = nxt[i - 1];
        while (j > 0 && s[i] != s[j]) j = nxt[j - 1];
        if (s[i] == s[j]) j++;
        nxt[i] = j;
    }
    return nxt;
}
// 在字符串text中查找pattern字符串
vector<int> find_occurrences(string text, string pattern) {
    string cur = pattern + '#' + text; // 加一个两个字符串中不存在的字符，表示最长前缀为n咯
    int sz1 = text.size(), sz2 = pattern.size();
    vector<int> v;
    vector<int> lps = prefix_function(cur);
    for (int i = sz2 + 1; i <= sz1 + sz2; i++) {
        if (lps[i] == sz2)
        v.push_back(i - 2 * sz2);
    }
    return v;
}

}}

namespace golitter {
namespace hash_ {
/*
    核心思想：将字符串看成P进制数，P的经验值是131或13331，取这两个值的冲突概率低
    小技巧：取模的数用2^64，这样直接用unsigned long long存储，溢出的结果就是取模的结果
*/
typedef unsigned long long ULL;
```

```
ULL h[N], p[N],n; // h[k]存储字符串前k个字母的哈希值, p[k]存储 P^k mod 2^64
ULL P = 111451; // 质数即可
char str[N];
void init() {
    // 初始化
    p[0] = 1;
    for (int i = 1; i <= n; i ++ ) {
        h[i] = h[i - 1] * P + str[i];
        p[i] = p[i - 1] * P;
    }
}

// 计算子串 str[l ~ r] 的哈希值
ULL get(int l, int r) {
    return h[r] - h[l - 1] * p[r - l + 1];
}
// C++ lambda
auto get_pre = [&](int l, int r) -> ULL {
    return h[r] - h[l - 1] * p[r - l + 1];
}; // lambda注意 逗号

// 封装
// https://ac.nowcoder.com/acm/contest/view-submission?submissionId=62962241
class strHash {
    typedef unsigned long long ULL;
public:
    strHash(const string& s) {
        this->str = "^" + s;
        dispose();
    }
    ULL get(int l, int r) {
        return h[r] - h[l - 1] * p[r - l + 1];
    }
private:
    void dispose() {
        len = str.size();
        h.assign(len + 1, 0); p.assign(len + 1, 0);
        h[0] = p[0] = 1;
        for(int i = 1; i <= len; ++i) {
            h[i] = h[i-1] * P + str[i];
            p[i] = p[i - 1] * P;
        }
    }

    const ULL P = 11451;
    string str;
    ULL len;
    vector<ULL> h,p;
};

typedef long long LL;
/**
 * 二维哈希板子
 * 注意：一定记得初始化 init()
*/
const int B[] = {223333333, 773333333}; // P1 P2
const int P = 1000002233;
```

```
LL *p[2];

void init(int N) { // 初始化
    p[0] = new LL [N];
    p[1] = new LL [N];
    p[0][0] = p[1][0] = 1;
    for (int i = 1; i < N; i++) {
        p[0][i] = p[0][i - 1] * B[0] % P;
        p[1][i] = p[1][i - 1] * B[1] % P;
    }
}

struct StringHash2D { // 字符串二维哈希 板子来源：
https://ac.nowcoder.com/acm/contest/view-submission?submissionId=63032157
    using LL = long long;
    int n, m;// n * m
    vector<vector<LL>> h;
    StringHash2D(const vector<string> &a) {
        n = a.size();
        m = (n == 0 ? 0 : a[0].size());
        h.assign(n + 1, {});
        for (int i = 0; i <= n; i++) { // 分配 n * m
            h[i].assign(m + 1, 0);
        }
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < m; j++) { // 二维哈希
                h[i + 1][j + 1] = (h[i][j + 1] * B[0] % P + h[i + 1][j] * B[1] %
P +
                    (P - h[i][j]) * B[0] % P * B[1] % P + a[i][j]) % P;
            }
        }
    }

    LL get(int x1, int y1, int x2, int y2) { // p1 = (x1, y1), p2 = (x2, y2)
[p1, p2)
        return (h[x2][y2] + h[x1][y1] * p[0][x2 - x1] % P * p[1][y2 - y1] % P +
            (P - h[x1][y2]) * p[0][x2 - x1] % P + (P - h[x2][y1]) * p[1][y2 -
y1] % P) % P;
    }
};

}}

namespace golitter {
namespace trie {

const int N = 2e5 + 21;
int tr[N][26],idx;
int cnt[N];
void insert(string str) {
    int p = 0;
    for(auto t: str) {
        int u = t - '0';
        if(!tr[p][u]) tr[p][u] = ++ idx;
        p = tr[p][u];
    }
    cnt[p]++;
}
```

```
int query(string str) {
    int p = 0;
    for(auto t: str) {
        int u = t - '0';
        if(!tr[p][u]) return 0;
        p = tr[p][u];
    }
    return cnt[p];
}

}
namespace trie_01 { // XOR https://zhuanlan.zhihu.com/p/373477543?utm_id=0

const int N = 1e5 + 21;
const int  M = N * 31;
int tr[M][2], a[N], idx;
int s[N];
void insert(int x) {
    int p = 0;
    for(int i = 30; i >= 0; --i) {
        int u = x >> i & 1; // 获得x二进制表示的第i位数
        if(!tr[p][u]) tr[p][u] = ++idx;
        p = tr[p][u];
    }
}
int query(int x) {
    int p = 0, res = 0;
    for(int i = 30; i >= 0; --i) {
        int u = x >> i & 1;
        if(tr[p][!u]) {
            p = tr[p][!u];
            res += 1<<i;
        } else {
            p = tr[p][u];
            // res = res << 1 + u;
        }
    }
    return res;
}
void solve() {
    int n; //idx = 0;
    for(int i = 0; i <= n; ++i) {
        tr[i][1] = 0; tr[i][0] = 0;
        s[i] = 0;
    }
    cin>>n;
    int res = 0;
    for(int i = 1; i <= n; ++i) cin>>a[i];
    for(int i = 1; i <= n; ++i) {
        s[i] = s[i-1] ^ a[i];
    }
    for(int i = 0; i <= n; ++i) {
        insert(s[i]);
        int t = query(s[i]);
        res = max(res,  t);
    }
    cout<<res<<'\n';
}
```

```
}
}


namespace golitter {
namespace manacher {
// https://zhuanlan.zhihu.com/p/549242325

vector<int> manacher(string &a) {// max(vector<int> P.size() ) - 1;
    string b = "$|";
    for(auto t: a) {
        b += t;
        b += '|';
    }
    int len = b.size();
    vector<int> hw(len);
    int maxright(0), mid(0);
    for(int i = 1; i < len; ++i) {
        if(i < maxright) hw[i] = min(hw[mid*2 - i], hw[mid] + mid - i);
        else hw[i] = 1;
        while(b[i - hw[i]] == b[i + hw[i]]) hw[i]++;
        if(i + hw[i] > maxright) {
            maxright = i + hw[i];
            mid = i;
        }
    }
    a = b;
    return hw;
}

}}

namespace golitter {
namespace fakeac { // 序列自动机

int fake[N][27];
char s[N], t[N];
void NC23053() { // https://ac.nowcoder.com/acm/problem/23053
    cin>>s + 1;
    int n; cin>>n;
    int len = strlen(s + 1);
    for(int i = len; i >= 0; --i) {
        for(int j = 0; j < 26; ++j) fake[i][j] = fake[i+1][j];
        if(i < len) fake[i][s[i+1] - 'a'] = i + 1;
    }
    while(n--) {
        cin>>t + 1;
        int tlen = strlen(t + 1);
        bool fg = false;
        int pos = 0;
        for(int i = 1; i <= tlen; ++i) {
            if(fake[pos][t[i] - 'a']) pos = fake[pos][t[i] - 'a'];
            else {
                fg = true;
                break;
            }
        }
    puts(!fg ? "Yes" : "No");
```

```
    }
}

}}

namespace golitter {
namespace ACAM {
// https://www.luogu.com.cn/problem/P5357
// https://zhuanlan.zhihu.com/p/546999184
struct acam {
    static const int M = 300005;
    struct node {
        int fail;
        int ch[28] = {0};
    }tr[M];
    int cnt = 0;
    map<int,int> ref;
    vector<int> g[M];
    int ans[M];
    void insert(string &s, int id) {
        int u = 0;
        for(auto c: s) {
            int last = c == '_' ? 26 : c - 'a';
            if(!tr[u].ch[last]) {
                tr[u].ch[last] = ++cnt;
            }
            u = tr[u].ch[last];
        }
        ref[id] = u;
    }
    void build() {
        queue<int> que;
        for(int i = 0; i < 27; ++i) {
            if(tr[0].ch[i]) que.push(tr[0].ch[i]);
        }
        while(que.size()) {
            int u = que.front(); que.pop();
            for(int i = 0; i < 27; ++i) {
                if(tr[u].ch[i]) {
                    tr[tr[u].ch[i]].fail = tr[tr[u].fail].ch[i];
                    que.push(tr[u].ch[i]);
                } else tr[u].ch[i] = tr[tr[u].fail].ch[i];
            }
        }
        for(int i = 1; i <= cnt; ++i) g[tr[i].fail].push_back(i);
    }
    void query(string &S) {
        int u = 0;
        for(auto c: S) {
            int last = c == '_' ? 26 : c - 'a';
            u = tr[u].ch[last];
            ans[u]++;
        }
    }
    void get_ans() {
        dfs(0);
    }
    void dfs(int u) {
```

```
        for(auto y: g[u]) {
            dfs(y);
            ans[u] += ans[y];
        }
    }
    void clear() {
        for(int i = 0; i < M; ++i) {
            tr[i] = {};
            g[i].clear();
            ans[i] = 0;
        }
        ref.clear();
        cnt = 0;
    }
}trie;

}}
```