字符串哈希习题

2791 兔子与兔子

Solution

子串哈希板子题

Code

```
void solve() {
    string str; cin >> str;
    int n = str.length();
    vector<Hash> vec(n + 1);
    for (int i = 1; i <= n; i++) {
        vec[i] = vec[i - 1].add(str[i - 1]);
    }

    auto unify = [&](int l, int r) {
        auto res = vec[r];
        auto pfx = vec[l - 1];
        pfx.x1 *= qpow(Hash::B, r - l + 1, Hash::M1); pfx.x1 %= Hash::M1;
        pfx.x2 *= qpow(Hash::B, r - l + 1, Hash::M2); pfx.x2 %= Hash::M2;
        return res - pfx;
    };

    int q; cin >> q;
    while (q--) {
        int l1, r1, l2, r2;
        cin >> l1 >> r1 >> l2 >> r2;
        if (unify(l1, r1) == unify(l2, r2)) cout << "Yes\n";
        else cout << "No\n";
    }
}
```

8307 最长回文子串问题

Solution

回文串分为两类：

- 有中心字符，即长度为奇数
- 没有中心字符，即长度为偶数（我们可以规定第 $\lfloor \frac{len}{2} \rfloor$ 个字符为中心）

显然，若 $str[i...j]$ 为回文串，$str[i+x...j-x]$ 也为回文串，反之，若 $str[i...j]$ 不为回文串，$str[i-x...j+x]$ 也不为回文串，因此，在中心字符的情况下，**是不是回文串**对字符串长度而言，是**单调**的，因此我们可以考虑枚举中心字符，二分长度求解

但是，对于串 abcdcba 而言，左边的 abcd 和右边的 dcba 很明显哈希值不一样，因此，我们可以计算一个**从左到右**的哈希值和一个**从右到左**的哈希值，这样相当于把字符串反转过来做了一遍哈希，此时右边的串 dcba 相当于被反转成了 abcd ，此时左右哈希值就相等了

注意二分的上下界选择

Bonus：对于回文串而言，我们可以选用 Manacher 算法

Code

```
void solve() {
    string str; cin >> str;
    int n = str.length();
    vector h1(n + 2, Hash<M1>()), h2(n + 2, Hash<M1>());
    vector b(n + 2, 0ll);

    b[0] = 1;
    for (int i = 1; i <= n; i++) {
        h1[i] = h1[i - 1].add(str[i - 1]);
        h2[n - i + 1] = h2[n - i + 2].add(str[n - i]);
        b[i] = b[i - 1] * B % M1;
    }

    auto modify = [&](int type, int l, int r) {
        if (type == 1)
            return h1[r] - h1[l - 1] * b[r - l + 1];
        else
            return h2[l] - h2[r + 1] * b[r - l + 1];
    };

    auto check = [&](int type, int mid, int len) {
        if (type ==  1)
            return modify(1, mid - len + 1, mid) == modify(2, mid, mid + len -
1);
        else
            return modify(1, mid - len + 1, mid) == modify(2, mid + 1, mid +
len);
    };

    pair ans_o{1, n}, ans_e{0, n};
    for (int i = n; i >= 1; i--) {
        int l = 1, r = min(n - i + 1, i), res = 1;
        while (l <= r) {
            int mid = (l + r) >> 1;
            if (check(1, i, mid)) l = mid + 1, res = mid;
            else r = mid - 1;
        }
        if (res > ans_o.first) ans_o = {res, i};

        l = 1, r = min(n - i, i), res = 0;
        while (l <= r) {
            int mid = (l + r) >> 1;
            if (check(2, i, mid)) l = mid + 1, res = mid;
            else r = mid - 1;
        }
        if (res > ans_e.first) ans_e = {res, i};
    }

    int L, R;
    if (ans_o.first * 2 - 1 > ans_e.first * 2)
```

```
        L = ans_o.second - ans_o.first, R = ans_o.second + ans_o.first - 1;
    else
        L = ans_e.second - ans_e.first, R = ans_e.second + ans_e.first;
    for (int i = L; i < R; i++) cout << str[i];
}
```

10321 「SPOJ LCS」求最长公共子串的长度

Solution

我们观察LCS的性质，发现如果有某个LCS长度为 len ，那么一定存在所有处于区间 $[1, len]$ 之间的LCS

那么，我们就可以二分答案

对于某个确定的长度 len ，我们将第一个字符串的所有等于该长度的子串的哈希值全部塞入 `std::set` 中，再利用第二个字符串的等于该长度的子串的哈希值去查找即可

注意本题可能卡单Hash（可以自己计算哈希冲突率）

Code

```
void solve() {
    vector s(2, string()); for (auto& str: s) cin >> str;
    vector h1(2, vector(max(s[0].size(), s[1].size()) + 2, Hash<M1>()));
    vector h2(2, vector(max(s[0].size(), s[1].size()) + 2, Hash<M2>()));
    vector b1(max(s[0].length(), s[1].length()) + 2, 1ull);
    vector b2(max(s[0].length(), s[1].length()) + 2, 1ull);

    for (int idx = 0; idx <= 1; idx++) {
        for (int i = 1; i <= s[idx].size(); i++) {
            h1[idx][i] = h1[idx][i - 1].add(s[idx][i - 1]);
            h2[idx][i] = h2[idx][i - 1].add(s[idx][i - 1]);
            b1[i] = b1[i - 1] * B % M1;
            b2[i] = b2[i - 1] * B % M2;
        }
    }

    int l = 1, r = min(s[0].size(), s[1].size()), ans = 0;
    set<pair<Hash<M1>, Hash<M2>>> st;

    auto modify = [&](int i, int l, int r) -> pair<Hash<M1>, Hash<M2>> {
        return {h1[i][r] - h1[i][l - 1] * b1[r - l + 1], h2[i][r] - h2[i][l - 1]
* b2[r - l + 1]};
    };

    auto check = [&](int mid) {
        st.clear();
        for (int i = mid; i <= s[0].size(); i++) st.insert(modify(0, i - mid +
1, i));
        for (int i = mid; i <= s[1].size(); i++)
            if (st.find(modify(1, i - mid + 1, i)) != st.end()) return 1;
        return 0;
    };
```

```
    while (l <= r) {
        auto mid = (l + r) >> 1;
        if (check(mid)) ans = mid, l = mid + 1;
        else r = mid - 1;
    }

    cout << ans << endl;
}
```

28821 最长公共子串

Solution

和两个串的LCS类似，不过要注意此时是 n 个串共同的LCS，因此在存储的时候，我们可以使用 `std::map` 来存储，遇到某个字符串有某个哈希值 k，就执行 `mp[k]++` （注意去重，即单个字符串不能多次对同一个 key 进行增操作），最后查看 `map` 内是否有 val 为 n 即可

Code

```
void solve() {
    int n; cin >> n;
    vector s(n, string()); for (auto& str: s) cin >> str;
    vector h1(n, vector(0, Hash<M1>()));
    vector h2(n, vector(0, Hash<M2>()));
    int maxx = 0;
    for (auto& str: s) cmax(maxx, (int)str.length());
    vector b1(maxx + 2, 1ull);
    vector b2(maxx + 2, 1ull);

    for (int idx = 0; idx < n; idx++) {
        h1[idx].resize(s[idx].length() + 2);
        h2[idx].resize(s[idx].length() + 2);
        for (int i = 1; i <= s[idx].size(); i++) {
            h1[idx][i] = h1[idx][i - 1].add(s[idx][i - 1]);
            h2[idx][i] = h2[idx][i - 1].add(s[idx][i - 1]);
            b1[i] = b1[i - 1] * B % M1;
            b2[i] = b2[i - 1] * B % M2;
        }
    }

    int l = 1, r = inf, ans = 0;
    for (auto& str: s) cmin(r, (int)str.length());
    map<pair<Hash<M1>, Hash<M2>>, int> mp;
    set<pair<Hash<M1>, Hash<M2>>> st;

    auto modify = [&](int i, int l, int r) -> pair<Hash<M1>, Hash<M2>> {
        return {h1[i][r] - h1[i][l - 1] * b1[r - l + 1], h2[i][r] - h2[i][l - 1]
* b2[r - l + 1]};
    };

    auto check = [&](int mid) {
        mp.clear();
        for (int idx = 0; idx < n; idx++) {
```

```
            st.clear();
            for (int i = mid; i <= s[idx].size(); i++) st.insert(modify(idx, i -
mid + 1, i));
            for (auto v: st) mp[v]++;
        }

        for (auto [f, s]: mp) if (s == n) return 1;
        return 0;
    };

    while (l <= r) {
        auto mid = (l + r) >> 1;
        if (check(mid)) ans = mid, l = mid + 1;
        else r = mid - 1;
    }

    cout << ans << endl;
}
```