AC 自动机

主体

```
int getid(char c) {
    if ('A' <= c && c <= 'Z') return c - 'A';
    if ('a' <= c && c <= 'z') return c - 'a' + 26;
    return c - '0' + 26 + 26;
}

struct AC {
    const int root = 1;
    int t[N][siz], fail[N], deg[N];
    bool vis[N];
    int cnt = root;

    void init() {
        for (int i = 1; i <= cnt; i++) memset(t[i], 0, sizeof(t[i]));
        memset(fail, 0, sizeof(int) * (cnt + 1));
        memset(vis, 0, sizeof(bool) * (cnt + 1));
        memset(deg, 0, sizeof(int) * (cnt + 1));
        cnt = root;
    }

    int insert(string &s) {
        int p = root;
        for (auto c: s) {
            int nex = getid(c);
            if (!t[p][nex]) t[p][nex] = ++cnt;
            p = t[p][nex];
        }
        return p;
    }

    void getfail() {
        queue<int> Q;
        for (int i = 0; i < siz; i++) {
            if (t[root][i]) {
                fail[t[root][i]] = root;
                Q.push(t[root][i]);
                deg[root]++;
```

```
            } else t[root][i] = root;
        }
        while (Q.size()) {
            int v = Q.front();
            Q.pop();
            for (int i = 0; i < siz; i++) {
                int &u = t[v][i];
                if (u) {
                    fail[u] = t[fail[v]][i];
                    deg[t[fail[v]][i]]++;
                    Q.push(u);
                } else u = t[fail[v]][i];
            }
        }
    }

    void query(string &s) {
        int p = root;
        for (auto c: s) {
            int nex = getid(c);
            p = t[p][nex];
            vis[p] = 1;
        }
    }

    void getans() {
        queue<int> Q;
        for (int i = 1; i <= cnt; i++) if (!deg[i]) Q.push(i);
        while (Q.size()) {
            int v = Q.front();
            Q.pop();
            deg[fail[v]]--;
            if (vis[v]) vis[fail[v]] = 1;
            if (!deg[fail[v]]) Q.push(fail[v]);
        }
    }
} ac;
```

调用顺序

```
ac.insert(s);
ac.getfail();
ac.query(t);
ac.getans();
```

tarjan

强连通分量

```
int dfn[N], low[N], col[N], tot, cnt;
stack<int> S;
void tarjan(int v) {
    low[v] = dfn[v] = ++tot;
    S.push(v);
    for (auto u: E[v]) {
        if (!dfn[u]) {
            tarjan(u);
            low[v] = min(low[v], low[u]);
        } else if (!col[u]) {
            low[v] = min(low[v], dfn[u]);
        }
    }
    if (low[v] == dfn[v]) {
        col[v] = ++cnt;
        while (S.top() != v) {
            col[S.top()] = cnt;
            S.pop();
        }
        S.pop();
    }
}
```

割点

```
void dfs(int v, int fa) {
    dfn[v] = low[v] = ++tot;
    int son = 0;
    for (int i = 0; i < E[v].size(); i++) {
        int x = E[v][i];
        if (!dfn[x]) {
            son++;
            dfs(x, v);
            low[v] = min(low[v], low[x]);
            if (v != 1 && low[x] >= dfn[v]) {
                mark[v] = true;
            }
            if (v == 1 && son == 2) {
                mark[1] = true;
            }
            // mark 了的是割点
        } else if (x != fa)
            low[v] = min(low[v], dfn[x]);
    }
}

dfs(1, 1);
```

桥

```
vector<int> E[N];
int dfn[N], low[N], tot, n, m;
vector<pair<int, int>> ans;
void dfs(int v, int fa) {
    dfn[v] = low[v] = ++tot;
    for (int i = 0; i < E[v].size(); i++) {
        int x = E[v][i];
        if (!dfn[x]) {
            dfs(x, v);
            low[v] = min(low[v], low[x]);
            if (low[x] > dfn[v]) {
                ans.emplace_back(x, v);
                // 加入 ans 中的就是桥
            }
        } else if (x != fa)
            low[v] = min(low[v], dfn[x]);
    }
}

dfs(1, 1);
```

KMP

```
int nex[N];
void getnex(const string &s) {
    int n = s.size();
    int i = 0, j = -1;
    nex[0] = -1;
    while (i < n) {
        if (s[i] == s[j] || j == -1)
            nex[++i] = ++j;
        else j = nex[j];
    }
}
int kmp(const string &a, const string &b) {
    int i = 0, j = 0, res = 0;
    getnex(b);
    while (i < a.size()) {
        if (j == -1 || a[i] == b[j]) {
            i++;
            j++;
        } else j = nex[j];
        if (j == b.size()) {
            j = nex[j];
            res++;
        }
    }
    return res;
}
```

Manacher

对于一个位置 i，$[i - f[i] + 1, i + f[i] - 1]$ 是最长的以 i 为中心的奇回文串（对应扩充串），$g[i]$ 是最长的以 i 为开头的回文串长度（对应原串）。

`max(f) - 1` 是原字符串的最长回文子串长度。

`g` 是答案数组，表示从原串每个位置出发，最长的回文前缀长度。

`t` 是辅助数组，用来记录“某个起点能到的最远右边界”。

```cpp
void manacher(std::string a, std::vector<int> &f, std::vector<int> &g) {
    int n = a.size(), m;
    std::vector<char> s(n * 2 + 3);
    for (int i = 1; i <= n; ++i) s[i << 1] = a[i - 1], s[i << 1 | 1] = '#';
    s[0] = '$', s[1] = '#', s[m = (n + 1) << 1] = '\0';

    f.resize(m + 1, 0); g.resize(n + 1, 0); f[1] = 1;
    for (int r = 0, p = 0, i = 2; i < m; ++i) {
        for (f[i] = r > i ? std::min(r - i, f[p * 2 - i]) : 1;
            s[i - f[i]] == s[i + f[i]]; ++f[i]);
        if (i + f[i] > r) r = i + f[i], p = i;
    }

    std::vector<int> t(m + 1, 0);
    auto update = [] (int &a, int b) { if (a < b) a = b; };
    for (int i = 0; i <= m; ++i) t[i] = 0;
    for (int i = 2; i < m; ++i) update(t[i - f[i] + 1], i + 1);
    for (int i = 1; i < m; ++i) update(t[i], t[i - 1]);
    for (int i = 2; i < m; ++i) g[i >> 1] = t[i] - i;
}
```

欧拉筛

```cpp
std::vector<int> prime;
bool notprime[N] = {1, 1};
void Euler(int n) {
    for (int i = 2; i <= n; i++) {
        if (!notprime[i]) prime.push_back(i);
        for (int j: prime) {
            if (i * j > n) break;
            notprime[i * j] = true;
            if (i % j == 0) break;
        }
    }
}
```