AC 自动机

Aho-Corasick Automaton

Idea：在 Trie 树上做 fail 匹配。两个关键：1. 子节点的 `fail` 是 `fail` 的对应子节点；2. 若无子节点，则子节点设为 `fail` 的对应子节点（Trie 树化为 Trie 图，这是一种对 AC 自动机的优化）。

ATT：在 `getfail` 时，先预处理第二层的 `fail[]` 作为初始条件。

Complexity：$O(N+M)$，其中 N 是模式串长度和，M 是主串长度。

What's more：fail 树——由 `fail` 指针构成的树形结构，在统计模式串出现次数时可以拿出来拓扑或 dfs 统计以减少复杂度。

Code：

```cpp
struct Trie{
    int ch[26], cntEnd; // cntEnd can be changed according to different problem
}tr[NODEN];
#define root 0
int id;
void insertTrie(char s[]){
    int len = (int)strlen(s);
    int now = root;
    for(int i = 0; i < len; i++){
        if(!tr[now].ch[s[i]-'a'])
            tr[now].ch[s[i]-'a'] = ++id;
        now = tr[now].ch[s[i]-'a'];
    }
    tr[now].cntEnd++;
}

int fail[NODEN];
void getFail(){
    queue<int> q;
    for(int i = 0; i < 26; i++){
        if(tr[root].ch[i]){
            fail[tr[root].ch[i]] = root;
            q.push(tr[root].ch[i]);
        }
    }
    while(!q.empty()){
        int cur = q.front(); q.pop();
        for(int i = 0; i < 26; i++){
            if(!tr[cur].ch[i])
                tr[cur].ch[i] = tr[fail[cur]].ch[i];
            else{
                fail[tr[cur].ch[i]] = tr[fail[cur]].ch[i];
                q.push(tr[cur].ch[i]);
            }
        }
    }
}

void ACauto(char s[]){
    int len = (int)strlen(s);
    int now = root;
    for(int i = 0; i < len; i++){
        now = tr[now].ch[s[i]-'a'];
        for(int t = now; t && tr[t].cntEnd != -1; t = fail[t]){
            ans += tr[t].cntEnd;
            tr[t].cntEnd = -1;
            // in this example, if a sting appears many times, we only count it once
        }
    }
}
```