

序列自动机

算法思想

序列自动机是接受且仅接受一个字符串的子序列的自动机。字符串设为 s

若 s 包含 n 个字符，那么序列自动机包括 $n+1$ 个状态。

令 t 是 s 的一个子序列，则 $d(\text{start}, t)$ 是 t 在 s 第一次出现时末端的位置。

也就是说，一个状态 i 表示前缀 $s[1\dots i]$ 的子序列与前缀 $s[1\dots i-1]$ 的子序列的差集。

序列自动机上的所有状态都是接受状态。

$d(u, c) = \min\{i | i > u, s[i] = c\}$ 也就是字符 c 下一次出现的位置。因为若 i 后缀 $s[i\dots |s|]$ 的子序列是后缀 $s[j\dots |s|]$ 的子序列的子集，一定要选尽量靠前的，才能最优。

构建复杂度是 $O(n \sum |s|)$

问题导入：

给定一个字符串 S q 次询问，每次给定另一个字符串 T 询问 T 是否是 S 的子序列
 $\text{len}(S) \leq 10^5, q \leq 10^5, \sum \text{len}(T) \leq 10^6$

最暴力的做法：建一个 $t_{\{0\}}$ 指向所有点，然后每个点的后继是后面的点，最后每个点打结束标记，这样光建边就要 $O(\text{len}^2)$ 再加上查询是 $O(\sum |T|)$ 显然爆炸。显然对于相同的字符贪心取前面的就好了，记录每种字符在哪些位置出现过。这个复杂度是 $O(\text{len})$ 的，然后对于查找串的当前字符 $\text{upper_bound}(\text{pos}[c].\text{begin}(), \text{pos}[c].\text{end}(), \text{now_pos})$ 如果查找到 end 说明不存在，如果一直找到最后说明存在。这个的复杂度是 $O(|S| + \sum |T| \log(|S|))$ 的。

```
const int MAXN=1001000;
const int MAXM=26;
int q;
char s[MAXN], t[MAXN];
vector<int> pos[MAXM];
bool work() {
    scanf("%s", t);
    int len=strlen(t);
    int nowpos=-1;
    for(int i=0; i<len; i++) {
        vector<int>::iterator y=upper_bound(pos[t[i]-'a'].begin(), pos[t[i]-'a'].end(), nowpos);
        if(y==pos[t[i]-'a'].end()) return false;
        nowpos=*y;
    }
    return true;
}

void solve() {
```

```
scanf ("%s",s);
scanf ("%d",&q);
int len=strlen(s);
for(int i=0;i<len;i++) {
    pos[s[i]-'a'].push_back(i);
}
while(q--) {
    if(work()) puts("Yes");
    else puts("No");
}

int main() {
    solve();
    return 0;
}
```

如果要支持带修，把 `$vector$` 换成平衡树即可（修改变成 `$O(\log n)$` 的）。

From:
<https://wiki.cvbbacm.com/> - CVBB ACM Team

Permanent link:
https://wiki.cvbbacm.com/doku.php?id=2020-2021:teams:legal_string:%E7%8E%8B%E6%99%BA%E5%BD%AA:%E5%BA%8F%E5%88%97%E8%87%AA%E5%8A%A8%E6%9C%BA&rev=1627751622

Last update: 2021/08/01 01:13

