

# 后缀数组

io-wiki

*nlogn*

```
1  #include <algorithm>
2  #include <cstdio>
3  #include <cstring>
4  #include <iostream>
5
6  using namespace std;
7
8  const int N = 1000010;
9
10 char s[N];
11 int n, sa[N], rk[N << 1], oldrk[N << 1], id[N], cnt[N];
12
13 int main() {
14     int i, m, p, w;
15
16     scanf("%s", s + 1);
17     n = strlen(s + 1);
18     m = max(n, 300);
19     for (i = 1; i <= n; ++i) ++cnt[rk[i] = s[i]];
20     for (i = 1; i <= m; ++i) cnt[i] += cnt[i - 1];
21     for (i = n; i >= 1; --i) sa[cnt[rk[i]]--] = i;
22
23     for (w = 1; w < n; w <= 1) {
24         memset(cnt, 0, sizeof(cnt));
25         for (i = 1; i <= n; ++i) id[i] = sa[i];
26         for (i = 1; i <= n; ++i) ++cnt[rk[id[i] + w]];
27         for (i = 1; i <= m; ++i) cnt[i] += cnt[i - 1];
28         for (i = n; i >= 1; --i) sa[cnt[rk[id[i] + w]]--] = id[i];
29         memset(cnt, 0, sizeof(cnt));
30         for (i = 1; i <= n; ++i) id[i] = sa[i];
31         for (i = 1; i <= n; ++i) ++cnt[rk[id[i]]];
32         for (i = 1; i <= m; ++i) cnt[i] += cnt[i - 1];
33         for (i = n; i >= 1; --i) sa[cnt[rk[id[i]]]--] = id[i];
34         memcpy(oldrk, rk, sizeof(rk));
35         for (p = 0, i = 1; i <= n; ++i) {
36             if (oldrk[sa[i]] == oldrk[sa[i - 1]] &&
37                 oldrk[sa[i] + w] == oldrk[sa[i - 1] + w]) {
38                 rk[sa[i]] = p;
39             } else {
40                 rk[sa[i]] = ++p;
41             }
42         }
43     }
44
45     for (i = 1; i <= n; ++i) printf("%d ", sa[i]);
46
47     return 0;
48 }
```

```

1 // 后缀类型
2 #define L_TYPE 0
3 #define S_TYPE 1
4
5 // 判断一个字符是否为LMS字符
6 inline bool is_lms_char(int *type, int x) {
7     return x > 0 && type[x] == S_TYPE && type[x - 1] == L_TYPE;
8 }
9
10 // 判断两个LMS子串是否相同
11 inline bool equal_substring(int *s, int x, int y, int *type) {
12     do {
13         if (s[x] != s[y])
14             return false;
15         x++, y++;
16     } while (!is_lms_char(type, x) && !is_lms_char(type, y));
17
18     return s[x] == s[y];
19 }
20
21 // 诱导排序(从*型诱导到L型、从L型诱导到S型)
22 // 调用之前应将*型按要求放入SA中
23 inline void induced_sort(int *s, int *sa, int *type, int *bucket, int
    *lbucket,
24                         int *sbucket, int n, int SIGMA) {
25     for (int i = 0; i <= n; i++)
26         if (sa[i] > 0 && type[sa[i] - 1] == L_TYPE)
27             sa[lbucket[s[sa[i] - 1]]++] = sa[i] - 1;
28     for (int i = 1; i <= SIGMA; i++) // Reset S-type bucket
29         sbucket[i] = bucket[i] - 1;
30     for (int i = n; i >= 0; i--)
31         if (sa[i] > 0 && type[sa[i] - 1] == S_TYPE)
32             sa[sbucket[s[sa[i] - 1]]--] = sa[i] - 1;
33 }
34
35 // SA-IS主体
36 // s是输入字符串, length是字符串的长度, SIGMA是字符集的大小
37 static int *SAIS(int *s, int length, int SIGMA) {
38     int n = length - 1;
39     int *type = new int[n + 1]; // 后缀类型
40     int *position = new int[n + 1]; // 记录LMS子串的起始位置
41     int *name = new int[n + 1]; // 记录每个LMS子串的新名称
42     int *sa = new int[n + 1]; // SA数组
43     int *bucket = new int[SIGMA + 1]; // 每个字符的桶
44     int *lbucket = new int[SIGMA + 1]; // 每个字符的L型桶的起始位置
45     int *sbucket = new int[SIGMA + 1]; // 每个字符的S型桶的起始位置
46
47     // 初始化每个桶
48     memset(bucket, 0, sizeof(int) * (SIGMA + 1));
49     for (int i = 0; i <= n; i++)
50         bucket[s[i]]++;
51     lbucket[0] = sbucket[0] = 0;
52     for (int i = 1; i <= SIGMA; i++) {

```

```

53     bucket[i] += bucket[i - 1];
54     lbucket[i] = bucket[i - 1];
55     sbucket[i] = bucket[i] - 1;
56 }
57
58 // 确定后缀类型(利用引理2.1)
59 type[n] = S_TYPE;
60 for (int i = n - 1; i >= 0; i--) {
61     if (S[i] < S[i + 1])
62         type[i] = S_TYPE;
63     else if (S[i] > S[i + 1])
64         type[i] = L_TYPE;
65     else
66         type[i] = type[i + 1];
67 }
68
69 // 寻找每个LMS子串
70 int cnt = 0;
71 for (int i = 1; i <= n; i++)
72     if (type[i] == S_TYPE && type[i - 1] == L_TYPE)
73         position[cnt++] = i;
74
75 // 对LMS子串进行排序
76 fill(SA, SA + n + 1, -1);
77 for (int i = 0; i < cnt; i++)
78     SA[sbucket[S[position[i]]]--] = position[i];
79 induced_sort(S, SA, type, bucket, lbucket, sbucket, n, SIGMA);
80
81 // 为每个LMS子串命名
82 fill(name, name + n + 1, -1);
83 int lastx = -1, namecnt = 1; // 上一次处理的LMS子串与名称的计数
84 bool flag = false; // 这里顺便记录是否有重复的字符
85 for (int i = 1; i <= n; i++) {
86     int x = SA[i];
87
88     if (is_lms_char(type, x)) {
89         if (lastx >= 0 && !equal_substring(S, x, lastx, type))
90             namecnt++;
91         // 因为只有相同的LMS子串才会有同样的名称
92         if (lastx >= 0 && namecnt == name[lastx])
93             flag = true;
94
95         name[x] = namecnt;
96         lastx = x;
97     }
98 } // for
99 name[n] = 0;
100
101 // 生成S1
102 int *S1 = new int[cnt];
103 int pos = 0;
104 for (int i = 0; i <= n; i++)
105     if (name[i] >= 0)
106         S1[pos++] = name[i];
107
108 int *SA1;
109 if (!flag) {
110     // 直接计算SA1

```

```

111         SA1 = new int[cnt + 1];
112
113         for (int i = 0; i < cnt; i++)
114             SA1[S1[i]] = i;
115     } else
116         SA1 = SAIS(S1, cnt, namecnt); // 递归计算SA1
117
118     // 从SA1诱导到SA
119     lbucket[0] = sbucket[0] = 0;
120     for (int i = 1; i <= SIGMA; i++) {
121         lbucket[i] = bucket[i - 1];
122         sbucket[i] = bucket[i] - 1;
123     }
124     fill(SA, SA + n + 1, -1);
125     for (int i = cnt - 1; i >= 0; i--) // 这里是逆序扫描SA1, 因为SA中S型桶是倒
序的
126         SA[sbucket[S[position[SA1[i]]]]--] = position[SA1[i]];
127     induced_sort(S, SA, type, bucket, lbucket, sbucket, n, SIGMA);
128
129     // 后缀数组计算完毕
130     return SA;
131 }

```

## 欧拉回路

```

1  #include <bits/stdc++.h>
2
3  using namespace std;
4  typedef long long ll;
5  const int N = 1e6 + 10;
6
7
8  int stk[N], top;
9  struct edge {
10     int to, idx;
11 };
12
13 vector<edge> g[N];
14
15 namespace Euler1 { // 獵豺惺鋈炬□鋤女誤璽?
16     bool vis[N];
17     int cur[N];
18
19     void dfs(int u, const int &w) {
20         vis[abs(w)] = true;
21         for (int &i = cur[u]; i < g[u].size(); i++) {
22             int idx = g[u][i].idx, v = g[u][i].to;
23             i++;
24             if (!vis[abs(idx)]) dfs(v, idx);
25         }
26         stk[++top] = w;
27     }
28

```

```

29     bool solve(int n) {
30         // init();
31         for (int i = 0; i <= n; i++) cur[i] = 0;
32         for (int i = 0; i <= n; i++) vis[i] = 0;
33         // calculate degree
34         for (int i = 1; i <= n; i++) {
35             if (g[i].size() & 1) return false;
36         }
37         // Hierholzer
38         for (int i = 1; i <= n; i++)
39             if (!g[i].empty()) {
40                 dfs(i, 0);
41                 break;
42             }
43         return true;
44     }
45 } // namespace Euler1
46
47 namespace Euler2 { // 鏈文惺鏷炬□鍋文誤璽?
48     int deg[N], cur[N];
49
50     void dfs(int u, const int &w) {
51         for (int &i = cur[u]; i < g[u].size(); i++) {
52             int idx = g[u][i].idx, v = g[u][i].to;
53             i++;
54             dfs(v, idx);
55         }
56         stk[++top] = w;
57     }
58
59     bool solve(int n) {
60         // init
61         for (int i = 0; i <= n; i++) deg[i] = 0;
62         for (int i = 0; i <= n; i++) cur[i] = 0;
63         // calculate degree
64         for (int i = 1; i <= n; ++i) {
65             for (auto x: g[i]) deg[i]++, deg[x.to]--;
66         }
67         for (int i = 1; i <= n; ++i)
68             if (deg[i]) return false;
69         // Hierholzer
70         for (int i = 1; i <= n; ++i)
71             if (!g[i].empty()) {
72                 dfs(i, 0);
73                 break;
74             }
75         return true;
76     }
77 } // namespace Euler2
78
79 int main() {
80     int t, n, m;
81     cin >> t >> n >> m;
82     for (int u, v, i = 1; i <= m; i++) {
83         cin >> u >> v;
84         g[u].push_back({v, i});
85         if (t == 1) g[v].push_back({u, -i});
86     }

```

```
87     // solve
88     bool flag = t == 1 ? Euler1::solve(n) : Euler2::solve(n);
89     // output
90     if (!flag || (m > 0 && top - 1 < m))
91         puts("NO");
92     else {
93         puts("YES");
94         for (int i = top - 1; i > 0; --i) printf("%d%c", stk[i], " \n"[i ==
95 1]);
96     }
97     return 0;
98 }
```