后缀数组

io-wiki

$nlogn$

```cpp
#include <algorithm>
#include <cstdio>
#include <cstring>
#include <iostream>

using namespace std;

const int N = 1000010;

char s[N];
int n, sa[N], rk[N << 1], oldrk[N << 1], id[N], cnt[N];

int main() {
  int i, m, p, w;

  scanf("%s", s + 1);
  n = strlen(s + 1);
  m = max(n, 300);
  for (i = 1; i <= n; ++i) ++cnt[rk[i] = s[i]];
  for (i = 1; i <= m; ++i) cnt[i] += cnt[i - 1];
  for (i = n; i >= 1; --i) sa[cnt[rk[i]]--] = i;

  for (w = 1; w < n; w <<= 1) {
    memset(cnt, 0, sizeof(cnt));
    for (i = 1; i <= n; ++i) id[i] = sa[i];
    for (i = 1; i <= n; ++i) ++cnt[rk[id[i] + w]];
    for (i = 1; i <= m; ++i) cnt[i] += cnt[i - 1];
    for (i = n; i >= 1; --i) sa[cnt[rk[id[i] + w]]--] = id[i];
    memset(cnt, 0, sizeof(cnt));
    for (i = 1; i <= n; ++i) id[i] = sa[i];
    for (i = 1; i <= n; ++i) ++cnt[rk[id[i]]];
    for (i = 1; i <= m; ++i) cnt[i] += cnt[i - 1];
    for (i = n; i >= 1; --i) sa[cnt[rk[id[i]]]--] = id[i];
    memcpy(oldrk, rk, sizeof(rk));
    for (p = 0, i = 1; i <= n; ++i) {
      if (oldrk[sa[i]] == oldrk[sa[i - 1]] &&
          oldrk[sa[i] + w] == oldrk[sa[i - 1] + w]) {
        rk[sa[i]] = p;
      } else {
        rk[sa[i]] = ++p;
      }
    }
  }

  for (i = 1; i <= n; ++i) printf("%d ", sa[i]);

  return 0;
}
```

sa-is

```cpp
// 后缀类型
#define L_TYPE 0
#define S_TYPE 1

// 判断一个字符是否为LMS字符
inline bool is_lms_char(int *type, int x) {
    return x > 0 && type[x] == S_TYPE && type[x - 1] == L_TYPE;
}

// 判断两个LMS子串是否相同
inline bool equal_substring(int *S, int x, int y, int *type) {
    do {
        if (S[x] != S[y])
            return false;
        x++, y++;
    } while (!is_lms_char(type, x) && !is_lms_char(type, y));

    return S[x] == S[y];
}

// 诱导排序(从*型诱导到L型、从L型诱导到S型)
// 调用之前应将*型按要求放入SA中
inline void induced_sort(int *S, int *SA, int *type, int *bucket, int
*lbucket,
                         int *sbucket, int n, int SIGMA) {
    for (int i = 0; i <= n; i++)
        if (SA[i] > 0 && type[SA[i] - 1] == L_TYPE)
            SA[lbucket[S[SA[i] - 1]]++] = SA[i] - 1;
    for (int i = 1; i <= SIGMA; i++)  // Reset S-type bucket
        sbucket[i] = bucket[i] - 1;
    for (int i = n; i >= 0; i--)
        if (SA[i] > 0 && type[SA[i] - 1] == S_TYPE)
            SA[sbucket[S[SA[i] - 1]]--] = SA[i] - 1;
}

// SA-IS主体
// S是输入字符串，length是字符串的长度，SIGMA是字符集的大小
static int *SAIS(int *S, int length, int SIGMA) {
    int n = length - 1;
    int *type = new int[n + 1];  // 后缀类型
    int *position = new int[n + 1];  // 记录LMS子串的起始位置
    int *name = new int[n + 1];  // 记录每个LMS子串的新名称
    int *SA = new int[n + 1];  // SA数组
    int *bucket = new int[SIGMA + 1];  // 每个字符的桶
    int *lbucket = new int[SIGMA + 1];  // 每个字符的L型桶的起始位置
    int *sbucket = new int[SIGMA + 1];  // 每个字符的S型桶的起始位置

    // 初始化每个桶
    memset(bucket, 0, sizeof(int) * (SIGMA + 1));
    for (int i = 0; i <= n; i++)
        bucket[S[i]]++;
    lbucket[0] = sbucket[0] = 0;
    for (int i = 1; i <= SIGMA; i++) {
```

```
        bucket[i] += bucket[i - 1];
        lbucket[i] = bucket[i - 1];
        sbucket[i] = bucket[i] - 1;
    }

    // 确定后缀类型(利用引理2.1)
    type[n] = S_TYPE;
    for (int i = n - 1; i >= 0; i--) {
        if (S[i] < S[i + 1])
            type[i] = S_TYPE;
        else if (S[i] > S[i + 1])
            type[i] = L_TYPE;
        else
            type[i] = type[i + 1];
    }

    // 寻找每个LMS子串
    int cnt = 0;
    for (int i = 1; i <= n; i++)
        if (type[i] == S_TYPE && type[i - 1] == L_TYPE)
            position[cnt++] = i;

    // 对LMS子串进行排序
    fill(SA, SA + n + 1, -1);
    for (int i = 0; i < cnt; i++)
        SA[sbucket[S[position[i]]]--] = position[i];
    induced_sort(S, SA, type, bucket, lbucket, sbucket, n, SIGMA);

    // 为每个LMS子串命名
    fill(name, name + n + 1, -1);
    int lastx = -1, namecnt = 1;  // 上一次处理的LMS子串与名称的计数
    bool flag = false;  // 这里顺便记录是否有重复的字符
    for (int i = 1; i <= n; i++) {
        int x = SA[i];

        if (is_lms_char(type, x)) {
            if (lastx >= 0 && !equal_substring(S, x, lastx, type))
                namecnt++;
            // 因为只有相同的LMS子串才会有同样的名称
            if (lastx >= 0 && namecnt == name[lastx])
                flag = true;

            name[x] = namecnt;
            lastx = x;
        }
    }  // for
    name[n] = 0;

    // 生成S1
    int *S1 = new int[cnt];
    int pos = 0;
    for (int i = 0; i <= n; i++)
        if (name[i] >= 0)
            S1[pos++] = name[i];

    int *SA1;
    if (!flag) {
        // 直接计算SA1
```

```
        SA1 = new int[cnt + 1];

        for (int i = 0; i < cnt; i++)
            SA1[S1[i]] = i;
    } else
        SA1 = SAIS(S1, cnt, namecnt);  // 递归计算SA1

    // 从SA1诱导到SA
    lbucket[0] = sbucket[0] = 0;
    for (int i = 1; i <= SIGMA; i++) {
        lbucket[i] = bucket[i - 1];
        sbucket[i] = bucket[i] - 1;
    }
    fill(SA, SA + n + 1, -1);
    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
        SA[sbucket[S[position[SA1[i]]]]--] = position[SA1[i]];
    induced_sort(S, SA, type, bucket, lbucket, sbucket, n, SIGMA);

    // 后缀数组计算完毕
    return SA;
}
```

欧拉回路

```
#include <bits/stdc++.h>

using namespace std;
typedef long long ll;
const int N = 1e6 + 10;


int stk[N], top;
struct edge {
    int to, idx;
};

vector<edge> g[N];

namespace Euler1 {  //鏃犲悜鍥炬□鎷夊洖璺?
    bool vis[N];
    int cur[N];

    void dfs(int u, const int &w) {
        vis[abs(w)] = true;
        for (int &i = cur[u]; i < g[u].size();) {
            int idx = g[u][i].idx, v = g[u][i].to;
            i++;
            if (!vis[abs(idx)]) dfs(v, idx);
        }
        stk[++top] = w;
    }

```

```cpp
    bool solve(int n) {
        // init();
        for (int i = 0; i <= n; i++) cur[i] = 0;
        for (int i = 0; i <= n; i++) vis[i] = 0;
        // calculate degree
        for (int i = 1; i <= n; i++) {
            if (g[i].size() & 1) return false;
        }
        // Hierholzer
        for (int i = 1; i <= n; i++)
            if (!g[i].empty()) {
                dfs(i, 0);
                break;
            }
        return true;
    }
}  // namespace Euler1

namespace Euler2 {  // 鏈攵悜鍥炬□鎷攵涙璧?
    int deg[N], cur[N];

    void dfs(int u, const int &w) {
        for (int &i = cur[u]; i < g[u].size();) {
            int idx = g[u][i].idx, v = g[u][i].to;
            i++;
            dfs(v, idx);
        }
        stk[++top] = w;
    }

    bool solve(int n) {
        // init
        for (int i = 0; i <= n; i++) deg[i] = 0;
        for (int i = 0; i <= n; i++) cur[i] = 0;
        // calculate degree
        for (int i = 1; i <= n; ++i) {
            for (auto x: g[i]) deg[i]++, deg[x.to]--;
        }
        for (int i = 1; i <= n; ++i)
            if (deg[i]) return false;
        // Hierholzer
        for (int i = 1; i <= n; ++i)
            if (!g[i].empty()) {
                dfs(i, 0);
                break;
            }
        return true;
    }
}  // namespace Euler2

int main() {
    int t, n, m;
    cin >> t >> n >> m;
    for (int u, v, i = 1; i <= m; i++) {
        cin >> u >> v;
        g[u].push_back({v, i});
        if (t == 1) g[v].push_back({u, -i});
    }
```

```
    // solve
    bool flag = t == 1 ? Euler1::solve(n) : Euler2::solve(n);
    // output
    if (!flag || (m > 0 && top - 1 < m))
        puts("NO");
    else {
        puts("YES");
        for (int i = top - 1; i > 0; --i) printf("%d%c", stk[i], " \n"[i ==
1]);
    }
    return 0;
}
```