



Standard Code Library

Part3 - String

Jiangxi Normal University
HeartFireY, eroengine, yezzz

May 16, 2023

Standard Code Library

HeartFireY, eroengine, yezzz

Jiangxi Normal University

May 16, 2023

Contents

| | |
|-----------------|----|
| Section.5 字符串 | 2 |
| KMP | 2 |
| Trie | 2 |
| AC 自动机 | 2 |
| 后缀自动机 | 3 |
| 回文自动机 | 8 |
| Manacher | 9 |
| 哈希 | 9 |
| 后缀数组 | 12 |
| $O(n\log n)$ 构造 | 12 |
| Height 数组的应用 | 12 |
| $O(n)$ 构造 | 13 |
| 后缀平衡树 | 14 |
| 做法 1 | 14 |
| 做法 2 | 14 |
| 做法 3 | 15 |

Section.5 字符串

KMP

- 前缀函数（每一个前缀的最长 border）

```
1  int nxt[N];
2
3  void get_pi(int nxt[], string s, int n){
4      int j = nxt[0] = 0;
5      for(int i = 2; i <= n; i++){
6          while(j && s[i] != s[j + 1]) j = nxt[j];
7          nxt[i] = j += s[i] == s[j + 1];
8      }
9  }
10
11 void kmp(string s, string p, int lens, int lenp){
12     for(int i = 1, j = 0; i <= lens; i++){
13         while(j > 0 && s[i] != p[j + 1]) j = nxt[j];
14         j += s[i] == p[j + 1];
15         if(j == lenp){
16             cout << i - lenp + 1 << endl;
17             j = nxt[j];
18         }
19     }
20 }
```

- Z 函数（每一个后缀和该字符串的 LCP 长度）

```
1  void get_z(int a[], char s[], int n) {
2      int l = 0, r = 0; a[0] = n;
3      FOR (i, 1, n) {
4          a[i] = i > r ? 0 : min(r - i + 1, a[i - l]);
5          while (i + a[i] < n && s[a[i]] == s[i + a[i]]) ++a[i];
6          if (i + a[i] - 1 > r) { l = i; r = i + a[i] - 1; }
7      }
8  }
```

Trie

```
1  namespace trie {
2      int t[N][26], sz, ed[N];
3      void init() { sz = 2; memset(ed, 0, sizeof ed); }
4      int _new() { memset(t[sz], 0, sizeof t[sz]); return sz++; }
5      void ins(char* s, int p) {
6          int u = 1;
7          FOR (i, 0, strlen(s)) {
8              int c = s[i] - 'a';
9              if (!t[u][c]) t[u][c] = _new();
10             u = t[u][c];
11         }
12         ed[u] = p;
13     }
14 }
```

AC 自动机

```
1  const int N = 1e6 + 10, M = 26, MOD = 1e9 + 7;
2
3  #define mp(x) (x - 'a')
4
5  namespace ACA{
6      int ch[N][M], fail[N], ed[N];
7      int sz;
8      void init() {
9          sz = 1;
10         memset(ch[0], 0, sizeof(ch));
11     }
12
13     void insert(const string &s, int id) {
```

```

14     int n = s.size(), u = 0, c;
15     for(int i = 0; i < n; i++) {
16         c = mp[s[i]];
17         if(!ch[u][c]) {
18             memset(ch[sz], 0, sizeof(ch[sz]));
19             ch[u][c] = sz++;
20         }
21         u = ch[u][c];
22     }
23     ed[u] = id;
24 }
25
26 void build() {
27     queue<int> Q;
28     fail[0] = 0;
29     for(int c = 0, u; c < M; c++) {
30         u = ch[0][c];
31         if(u) { Q.emplace(u); fail[u] = 0; }
32     }
33     while(!Q.empty()) {
34         int r = Q.front(); Q.pop();
35         for(int c = 0, u; c < M; c++) {
36             u = ch[r][c];
37             if(!u) {
38                 ch[r][c] = ch[fail[r]][c];
39                 continue;
40             }
41             fail[u] = ch[fail[r]][c];
42             Q.emplace(u);
43         }
44     }
45 }
46 }
47

```

后缀自动机

- 广义后缀自动机如果直接使用以下代码的话会产生一些冗余状态（置 last 为 1），所以要用拓扑排序。用 len 基数排序不能。
- 字符集大的话要使用 **map**。
- 树上 dp 时注意边界（root 和 null）。
- rsort 中的数组 a 是拓扑序 [1, sz)

```

1 struct SAM{
2     int ch[N << 1][26], fa[N << 1], len[N << 1], vis[N << 1];
3     int last, tot;
4     SAM(): last(1), tot(1) {}
5     inline void extend(int x){ /* 单字符扩展
6         int p = last, np = last = ++tot;
7         len[np] = len[p] + 1, vis[np] = 1;
8         for(; p && !ch[p][x]; p = fa[p]) ch[p][x] = np;
9         if(!p) fa[np] = 1;
10        else{
11            int q = ch[p][x];
12            if(len[q] == len[p] + 1) fa[np] = q;
13            else {
14                int nq = ++tot;
15                for(int i = 0; i < 26; i++) ch[nq][i] = ch[q][i]; //for(int i = 0; i < 26; i++) ch[nq][i] = ch[q][i];
16                fa[nq] = fa[q], fa[np] = fa[q] = nq, len[nq] = len[p] + 1;
17                for(; ch[p][x] == q; p = fa[p]) ch[p][x] = nq;
18            }
19        }
20    }
21 }sam;

```

- 最长公共子串

```

1 /* 最长公共子串
2 string lcs(const string &T) {
3     int v = 0, l = 0, best = 0, bestpos = 0;
4     for (int i = 0; i < T.size(); i++) {

```

```

5         while (v && !sam.ch[v][T[i] - 'a']) {
6             v = sam.fa[v];
7             l = sam.len[v];
8         }
9         if (sam.ch[v][T[i] - 'a']) {
10            v = sam.ch[v][T[i] - 'a'];
11            l++;
12        }
13        if (l > best) {
14            best = l;
15            bestpos = i;
16        }
17    }
18    return T.substr(bestpos - best + 1, best);
19 }

```

- 真·广义后缀自动机

```

1  int t[M][26], len[M] = {-1}, fa[M], sz = 2, last = 1;
2  LL cnt[M][2];
3  void ins(int ch, int id) {
4      int p = last, np = 0, nq = 0, q = -1;
5      if (!t[p][ch]) {
6          np = sz++;
7          len[np] = len[p] + 1;
8          for (; p && !t[p][ch]; p = fa[p]) t[p][ch] = np;
9      }
10     if (!p) fa[np] = 1;
11     else {
12         q = t[p][ch];
13         if (len[p] + 1 == len[q]) fa[np] = q;
14         else {
15             nq = sz++; len[nq] = len[p] + 1;
16             memcpy(t[nq], t[q], sizeof t[0]);
17             fa[nq] = fa[q];
18             fa[np] = fa[q] = nq;
19             for (; t[p][ch] == q; p = fa[p]) t[p][ch] = nq;
20         }
21     }
22     last = np ? np : nq ? nq : q;
23     cnt[last][id] = 1;
24 }

```

- 按字典序建立后缀树注意逆序插入
- rsort2 里的 a 不是拓扑序，需要拓扑序就去树上做

```

1  void ins(int ch, int pp) {
2      int p = last, np = last = sz++;
3      len[np] = len[p] + 1; one[np] = pos[np] = pp;
4      for (; p && !t[p][ch]; p = fa[p]) t[p][ch] = np;
5      if (!p) { fa[np] = 1; return; }
6      int q = t[p][ch];
7      if (len[q] == len[p] + 1) fa[np] = q;
8      else {
9          int nq = sz++; len[nq] = len[p] + 1; one[nq] = one[q];
10         memcpy(t[nq], t[q], sizeof t[0]);
11         fa[nq] = fa[q];
12         fa[q] = fa[np] = nq;
13         for (; p && t[p][ch] == q; p = fa[p]) t[p][ch] = nq;
14     }
15 }
16
17 int up[M], c[256] = {2}, a[M];
18 void rsort2() {
19     FOR (i, 1, 256) c[i] = 0;
20     FOR (i, 2, sz) up[i] = s[one[i] + len[fa[i]]];
21     FOR (i, 2, sz) c[up[i]]++;
22     FOR (i, 1, 256) c[i] += c[i - 1];
23     FOR (i, 2, sz) a[--c[up[i]]] = i;
24     FOR (i, 2, sz) G[fa[a[i]]].push_back(a[i]);
25 }

```

- 广义后缀自动机建后缀树，必须反向插入

```

1  int t[M][26], len[M] = {0}, fa[M], sz = 2, last = 1;
2  char* one[M];
3  void ins(int ch, char* pp) {
4      int p = last, np = 0, nq = 0, q = -1;
5      if (!t[p][ch]) {
6          np = sz++; one[np] = pp;
7          len[np] = len[p] + 1;
8          for (; p && !t[p][ch]; p = fa[p]) t[p][ch] = np;
9      }
10     if (!p) fa[np] = 1;
11     else {
12         q = t[p][ch];
13         if (len[p] + 1 == len[q]) fa[np] = q;
14         else {
15             nq = sz++; len[nq] = len[p] + 1; one[nq] = one[q];
16             memcpy(t[nq], t[q], sizeof t[0]);
17             fa[nq] = fa[q];
18             fa[np] = fa[q] = nq;
19             for (; t[p][ch] == q; p = fa[p]) t[p][ch] = nq;
20         }
21     }
22     last = np ? np : nq ? nq : q;
23 }
24 int up[M], c[256] = {2}, aa[M];
25 vector<int> G[M];
26 void rsort() {
27     FOR (i, 1, 256) c[i] = 0;
28     FOR (i, 2, sz) up[i] = *(one[i] + len[fa[i]]);
29     FOR (i, 2, sz) c[up[i]]++;
30     FOR (i, 1, 256) c[i] += c[i - 1];
31     FOR (i, 2, sz) aa[--c[up[i]]] = i;
32     FOR (i, 2, sz) G[fa[aa[i]]].push_back(aa[i]);
33 }

```

- 匹配

```

1  int u = 1, l = 0;
2  FOR (i, 0, strlen(s)) {
3      int ch = s[i] - 'a';
4      while (u && !t[u][ch]) { u = fa[u]; l = len[u]; }
5      ++l; u = t[u][ch];
6      if (!u) u = 1;
7      if (l) // do something...
8  }

```

- 获取子串状态

```

1  int get_state(int l, int r) {
2      int u = rpos[r], s = r - l + 1;
3      FOR (i, SP - 1, -1) if (len[pa[u][i]] >= s) u = pa[u][i];
4      return u;
5  }

```

- 维护区间本质不同字符串数目

- 给你一个长度为 n 的字符串 s , m 次询问, 第 i 次询问 s 上的一个区间 $[l_i, r_i]$ 上有多少个本质不同的子串
- 将每个本质不同的字符串视为一个连续的区间 $[l, r]$, 我们只需要维护左端点最后一次出现的位置即可

因为需要知道每个子串最后一次出现的位置, 所以我们选择对字符串构造后缀自动机, 对于某个右端点 r SAM $[1, r]$ parent r pre[i] i 上一次出现时的右端点位置, 我们只需要暴跳 $father$, 每次将之前出现过的位置, 在线段树上区间更新成 -1 $[1, r], [2, r] \dots [r, r]$ r $[1, r]$ 在线段树上 $+1$ 就好了

不过问题是, 这样暴跳 $father$ 的时间复杂度是不正确的, 考虑优化, 因为每次选择一条链, 自下而上去更新, 其本质就是:

1. 令这条链每个节点相应的位置在线段树上区间更新
2. 令每个节点都被端点 r pre r

这个过程其实就是 LCT access $\log n$ splay $\log^2 n$ $n \log^2 n$ 了。

```

1  #include <bits/stdc++.h>
2  #pragma gcc optimize("O2")
3  #pragma g++ optimize("O2")
4  #define int long long
5  #define endl '\n'
6  using namespace std;
7
8  const int N = 2e5 + 10, MOD = 1e9 + 7;
9
10 int n = 0;
11
12 namespace DS{
13     namespace SAM{
14         int ch[N << 1][26], fa[N << 1], len[N << 1], vis[N << 1], pos[N << 1];
15         int last, tot;
16         inline void extend(int x){
17             int p = last, np = last = ++tot;
18             len[np] = len[p] + 1, vis[np] = 1;
19             for(; p && !ch[p][x]; p = fa[p]) ch[p][x] = np;
20             if(!p) fa[np] = 1;
21             else{
22                 int q = ch[p][x];
23                 if(len[q] == len[p] + 1) fa[np] = q;
24                 else {
25                     int nq = ++tot;
26                     for(int i = 0; i < 26; i++) ch[nq][i] = ch[q][i];
27                     fa[nq] = fa[q], fa[q] = nq, len[nq] = len[p] + 1;
28                     for(; ch[p][x] == q; p = fa[p]) ch[p][x] = nq;
29                 }
30             }
31         }
32     }
33
34     void build(string s) {
35         last = tot = 1;
36         int len = s.size();
37         s = '@' + s;
38         for(int i = 1; i <= len; i++) extend(s[i] - 'a'), pos[i] = last;
39     }
40
41     namespace SegTree{
42         #define ls rt << 1
43         #define rs rt << 1 | 1
44         #define lson ls, l, mid
45         #define rson rs, mid + 1, r
46         int tree[N << 2], lazy[N << 2];
47
48         inline void push_up(int rt) { tree[rt] = tree[ls] + tree[rs]; }
49
50         inline void push(int rt, int val, int c) { tree[rt] += val * c, lazy[rt] += val; }
51
52         inline void push_down(int rt, int c) {
53             if(lazy[rt]) {
54                 push(ls, lazy[rt], (c - (c >> 1)));
55                 push(rs, lazy[rt], (c >> 1));
56                 lazy[rt] = 0;
57             }
58         }
59
60         void build(int rt, int l, int r){
61             tree[rt] = lazy[rt] = 0;
62             if(l == r) return;
63             int mid = l + r >> 1;
64             build(lson), build(rson);
65         }
66
67         void update(int rt, int l, int r, int L, int R, int val) {
68             if(l >= L && r <= R) return push(rt, val, r - l + 1);
69             push_down(rt, r - l + 1);
70             int mid = l + r >> 1;
71             if(mid >= L) update(lson, L, R, val);

```



```

72     if(mid < R) update(rson, L, R, val);
73     push_up(rt);
74 }
75
76 int query(int rt, int l, int r, int L, int R) {
77     if(l >= L && r <= R) return tree[rt];
78     push_down(rt, r - l + 1);
79     int mid = l + r >> 1, sum = 0;
80     if(mid >= L) sum += query(lson, L, R);
81     if(mid < R) sum += query(rson, L, R);
82     return sum;
83 }
84 #undef ls
85 #undef rs
86 #undef lson
87 #undef rson
88 }
89
90 namespace LCT{
91     #define ls ch[x][0]
92     #define rs ch[x][1]
93
94     struct Info{
95         int len, minn, pre, tag_chg;
96     }tree[N];
97
98     int ch[N][2], f[N], tag[N];
99
100     inline void push_up(int x) { tree[x].minn = min({tree[x].len, tree[ls].minn, tree[rs].minn}); }
101
102     inline void push(int x) { swap(ls, rs), tag[x] ^= 1; }
103
104     inline void push_chg(int x, int v) { tree[x].pre = tree[x].tag_chg = v; }
105
106     inline void push_down(int x) {
107         if(tag[x]) {
108             if(ls) push(ls);
109             if(rs) push(rs);
110             tag[x] = 0;
111         }
112         if(tree[x].tag_chg) {
113             if(ls) push_chg(ls, tree[x].tag_chg);
114             if(rs) push_chg(rs, tree[x].tag_chg);
115             tree[x].tag_chg = 0;
116         }
117     }
118
119     #define get(x) (ch[f[x]][1] == x)
120     #define isRoot(x) (ch[f[x]][0] != x && ch[f[x]][1] != x)
121
122     inline void rotate(int x) {
123         int y = f[x], z = f[y], k = get(x);
124         if(!isRoot(y)) ch[z][ch[z][1] == y] = x;
125         ch[y][k] = ch[x][!k], f[ch[x][!k]] = y;
126         ch[x][!k] = y, f[y] = x, f[x] = z;
127         push_up(y); push_up(x);
128     }
129
130     inline void update(int x) {
131         if(!isRoot(x)) update(f[x]);
132         push_down(x);
133     }
134
135     inline void splay(int x) {
136         update(x);
137         for(int fa = f[x]; !isRoot(x); rotate(x), fa = f[x]){
138             if(!isRoot(fa)) rotate(get(fa) == get(x) ? fa : x);
139         }
140         push_up(x);
141     }
142 }

```

```

143     int access(int x, int pos) {
144         int p;
145         for(p = 0; x; x = f[p = x]){
146             splay(x), ch[x][1] = p, push_up(x);
147             if(tree[x].pre) {
148                 int upl = tree[x].pre - SAM::len[x] + 1;
149                 int upr = tree[x].pre - tree[x].minn + 1;
150                 // cout << "LCT Operation SegTree -> Part(" << upl << ", " << upr << ")", add value -1\n";
151                 SegTree::update(1, 1, n, upl, upr, -1);
152             }
153         }
154         splay(p);
155         push_chg(p, pos);
156         SegTree::update(1, 1, n, 1, pos, 1);
157         // cout << endl;
158         return p;
159     }
160
161     void build() {
162         tree[0].minn = 1e18;
163         for(int i = 1; i <= SAM::tot; i++) {
164             f[i] = SAM::fa[i];
165             tree[i].len = tree[i].minn = SAM::len[SAM::fa[i]] + 1;
166             tree[i].pre = tree[i].tag_chg = ch[i][0] = ch[i][1] = 0;
167         }
168     }
169     #undef ls
170     #undef rs
171 }
172
173 struct query{ int l, id; };
174 vector<query> qr[N];
175 int ans[N];
176
177 inline void solve(){
178     string s; cin >> s, n = s.size();
179     DS::SAM::build(s);
180     DS::SegTree::build(1, 1, n);
181     DS::LCT::build();
182     int m = 0; cin >> m;
183     for(int i = 1; i <= m; i++) {
184         int l, r; cin >> l >> r;
185         qr[r].emplace_back(query{l, i});
186     }
187     for(int i = 1; i <= n; i++) {
188         DS::LCT::access(DS::SAM::pos[i], i);
189         for(auto &[l, id] : qr[i]) ans[id] = DS::SegTree::query(1, 1, n, l, i);
190     }
191     for(int i = 1; i <= m; i++) cout << ans[i] << endl;
192 }
193
194 signed main(){
195     ios_base::sync_with_stdio(false), cin.tie(0);
196     cout << fixed << setprecision(12);
197     int t = 1; // cin >> t;
198     while(t--) solve();
199     return 0;
200 }
201

```

回文自动机

- num 是该结点表示的前缀的回文后缀个数
- cnt 是该结点表示的回文串在原串中的出现次数（使用前需要向父亲更新）

```

1 namespace pam {
2     int t[N][26], fa[N], len[N], rs[N], cnt[N], num[N];
3     int sz, n, last;
4     int _new(int l) {
5         len[sz] = l; cnt[sz] = num[sz] = 0;

```

```

6     return sz++;
7 }
8 void init() {
9     memset(t, 0, sz * sizeof t[0]);
10    rs[n = sz = 0] = -1;
11    last = _new(0);
12    fa[last] = _new(-1);
13 }
14 int get_fa(int x) {
15     while (rs[n - 1 - len[x]] != rs[n]) x = fa[x];
16     return x;
17 }
18 void ins(int ch) {
19     rs[++n] = ch;
20     int p = get_fa(last);
21     if (!t[p][ch]) {
22         int np = _new(len[p] + 2);
23         num[np] = num[fa[np]] = t[get_fa(fa[p])][ch] + 1;
24         t[p][ch] = np;
25     }
26     ++cnt[last = t[p][ch]];
27 }
28 }

```

Manacher

$P[i]$ 的计算方法: 假设 j 是 i 关于 C 的镜像点, $P[j]$ 已经求解完毕。

- 若 $i \geq R$, 由于 R 右侧的字符都没有检查过, 因此只能初始化 $P[i] = 1$ 然后暴力中心扩展
- 若 $i < R$, 分两种情况:
 1. j 的回文串被 C 的回文串包含, 即 j 的回文串左端点比 C 回文串的左端点大, 按照镜像原理, 镜像 i 的回文不会超过 C 的右端点 R , 因此根据 $(i + j)/2 = C$ 得 $j = 2C - i$, 故 $P[i] = P[j] = P[2C - i]$ 。然后继续用暴力中心扩展法完成 $P[i]$ 的计算。
 2. j 的回文串不被 C 的回文串包含, 即 j 的回文串左端点比 C 回文串的左端点小。此时 i 回文串的右端点比 R 大, 但是由于 R 右边的字符还没有检查过, 只能先让 $P[i]$ 被限制在 R 之内, 有 $P[i] = w = R - i = C + P[i] - i$, 然后继续用暴力中心扩展法完成 $P[i]$ 的计算。

```

1  int n, p[N << 1]; // p[i]: 以 s[i] 为中心的回文串半径
2  char a[N], s[N << 1]; // a 为原始串, s 为修改后的串
3
4  void change() {
5      n = strlen(a);
6      int k = 0; s[k++] = '$', s[k++] = '#';
7      for(int i = 0; i < n; i++) s[k++] = a[i], s[k++] = '#';
8      s[k++] = '&', n = k;
9  }
10
11 void manacher() {
12     int R = 0, C = 0;
13     for(int i = 1; i < n; i++) {
14         if(i < R) p[i] = min(p[(C << 1) - i], p[C] + C - i); // 1. 合并处理两种情况
15         else p[i] = 1; //
16         while(s[i + p[i]] == s[i - p[i]]) p[i]++; // 2. 暴力中心扩展
17         if(p[i] + i > R) R = p[i] + i, C = i; // 3. 更新最大的 R
18     }
19 }
20
21 inline void solve() {
22     cin >> a;
23     change(), manacher();
24     int ans = 1;
25     for(int i = 0; i < n; i++) ans = max(ans, p[i]);
26     cout << ans - 1 << endl;
27 }

```

哈希

内置了自动双哈希开关 (小心 TLE)。

```

1  #include <bits/stdc++.h>
2  using namespace std;
3
4  #define ENABLE_DOUBLE_HASH
5
6  typedef long long LL;
7  typedef unsigned long long ULL;
8
9  const int x = 135;
10 const int N = 4e5 + 10;
11 const int p1 = 1e9 + 7, p2 = 1e9 + 9;
12 ULL xp1[N], xp2[N], xp[N];
13
14 void init_xp() {
15     xp1[0] = xp2[0] = xp[0] = 1;
16     for (int i = 1; i < N; ++i) {
17         xp1[i] = xp1[i - 1] * x % p1;
18         xp2[i] = xp2[i - 1] * x % p2;
19         xp[i] = xp[i - 1] * x;
20     }
21 }
22
23 struct String {
24     char s[N];
25     int length, subsize;
26     bool sorted;
27     ULL h[N], hl[N];
28
29     ULL hash() {
30         length = strlen(s);
31         ULL res1 = 0, res2 = 0;
32         h[length] = 0; // ATTENTION!
33         for (int j = length - 1; j >= 0; --j) {
34             #ifdef ENABLE_DOUBLE_HASH
35                 res1 = (res1 * x + s[j]) % p1;
36                 res2 = (res2 * x + s[j]) % p2;
37                 h[j] = (res1 << 32) | res2;
38             #else
39                 res1 = res1 * x + s[j];
40                 h[j] = res1;
41             #endif
42             // printf("%llu\n", h[j]);
43         }
44         return h[0];
45     }
46
47     // 获取子串哈希, 左闭右开区间
48     ULL get_substring_hash(int left, int right) const {
49         int len = right - left;
50         #ifdef ENABLE_DOUBLE_HASH
51             // get hash of s[left...right-1]
52             unsigned int mask32 = ~(0u);
53             ULL left1 = h[left] >> 32, right1 = h[right] >> 32;
54             ULL left2 = h[left] & mask32, right2 = h[right] & mask32;
55             return (((left1 - right1 * xp1[len] % p1 + p1) % p1) << 32) |
56                 (((left2 - right2 * xp2[len] % p2 + p2) % p2));
57         #else
58             return h[left] - h[right] * xp[len];
59         #endif
60     }
61
62     void get_all_subs_hash(int sublen) {
63         subsize = length - sublen + 1;
64         for (int i = 0; i < subsize; ++i)
65             hl[i] = get_substring_hash(i, i + sublen);
66         sorted = 0;
67     }
68
69     void sort_substring_hash() {
70         sort(hl, hl + subsize);
71         sorted = 1;

```

```

72     }
73
74     bool match(ULL key) const {
75         if (!sorted) assert (0);
76         if (!subsize) return false;
77         return binary_search(hl, hl + subsize, key);
78     }
79
80     void init(const char *t) {
81         length = strlen(t);
82         strcpy(s, t);
83     }
84 };
85
86 int LCP(const String &a, const String &b, int ai, int bi) {
87     // Find LCP of a[ai...] and b[bi...]
88     int l = 0, r = min(a.length - ai, b.length - bi);
89     while (l < r) {
90         int mid = (l + r + 1) / 2;
91         if (a.get_substring_hash(ai, ai + mid) == b.get_substring_hash(bi, bi + mid))
92             l = mid;
93         else r = mid - 1;
94     }
95     return l;
96 }
97
98 int check(int ans) {
99     if (T.length < ans) return 1;
100    T.get_all_subs_hash(ans); T.sort_substring_hash();
101    for (int i = 0; i < S.length - ans + 1; ++i)
102        if (!T.match(S.get_substring_hash(i, i + ans)))
103            return 1;
104    return 0;
105 }
106
107 int main() {
108     init_xp(); // DON'T FORGET TO DO THIS!
109
110     for (int tt = 1; tt <= kases; ++tt) {
111         scanf("%d", &n); scanf("%s", str);
112         S.init(str);
113         S.hash(); T.hash();
114     }
115 }

```

二维哈希

```

1  struct Hash2D { // 1-index
2      static const LL px = 131, py = 233, MOD = 998244353;
3      static LL pwx[N], pwy[N];
4      int a[N][N];
5      LL hv[N][N];
6      static void init_xp() {
7          pwx[0] = pwy[0] = 1;
8          FOR (i, 1, N) {
9              pwx[i] = pwx[i - 1] * px % MOD;
10             pwy[i] = pwy[i - 1] * py % MOD;
11         }
12     }
13     void init_hash(int n, int m) {
14         FOR (i, 1, n + 1) {
15             LL s = 0;
16             FOR (j, 1, m + 1) {
17                 s = (s * py + a[i][j]) % MOD;
18                 hv[i][j] = (hv[i - 1][j] * px + s) % MOD;
19             }
20         }
21     }
22     LL h(int x, int y, int dx, int dy) {
23         --x; --y;
24         LL ret = hv[x + dx][y + dy] + hv[x][y] * pwx[dx] % MOD * pwy[dy]
25             - hv[x][y + dy] * pwx[dx] - hv[x + dx][y] * pwy[dy];

```

```

26     return (ret % MOD + MOD) % MOD;
27 }
28 } ha, hb;
29 LL Hash2D::pwx[N], Hash2D::pwy[N];

```

后缀数组

$O(n \log n)$ 构造

构造时间: $O(L \log L)$; 查询时间 $O(\log L)$ 。 **suffix** 数组是排好序的后缀下标, **suffix** 的反数组是后缀数组。

```

1  const int N = 1e6 + 10;
2
3  namespace SA {
4      int sa[N], tax[N], pool1[N], pool2[N], height[N];
5      int *rnk = pool1, *tp = pool2;
6      void sort(int n, int m) {
7          for(int i = 0; i <= m; i++) tax[i] = 0;
8          for(int i = 1; i <= n; i++) tax[rnk[i]]++;
9          for(int i = 1; i <= m; i++) tax[i] += tax[i - 1];
10         for(int i = n; i >= 1; i--) sa[tax[rnk[tp[i]]]--] = tp[i];
11     }
12
13     void build(string str, int n, int m) {
14         for(int i = 1; i <= n; i++) rnk[i] = str[i] - '0' + 1, tp[i] = i;
15         sort(n, m);
16         for(int w = 1, p = 0; p < n; w <= 1, m = p) {
17             p = 0;
18             for(int i = 1; i <= w; i++) tp[++p] = n - w + i;
19             for(int i = 1; i <= n; i++) if(sa[i] > w) tp[++p] = sa[i] - w;
20             sort(n, m);
21             swap(tp, rnk);
22             rnk[sa[1]] = p = 1;
23             for(int i = 2; i <= n; i++) rnk[sa[i]] = (tp[sa[i - 1]] == tp[sa[i]] && tp[sa[i - 1] + w] == tp[sa[i] + w]) ?
                ↪ p : ++p;
24         }
25         int k = 0;
26         for(int i = 1; i <= n; i++) {
27             if(rnk[i] == 1) continue;
28             if(k) --k;
29             for(int j = sa[rnk[i] - 1]; str[i + k] == str[j + k]; ++k);
30             height[rnk[i]] = k;
31         } // Get the height array (height[i] = lcp(sa[i], sa[i - 1]))
32     }
33
34     void reset() {
35         memset(sa, 0, sizeof(sa));
36         memset(rnk, 0, sizeof(pool1));
37         memset(tp, 0, sizeof(pool2));
38         memset(tax, 0, sizeof(tax));
39         rnk = pool1, tp = pool2;
40     }
41 } // Suffix Array (init pos = 1)
42
43 using SA::sa;
44
45 inline void solve() {
46     string s; cin >> s;
47     int len = s.size();
48     SA::build('@' + s, len, 75);
49     for(int i = 1; i <= len; i++) cout << sa[i] << " \n"[i == len];
50 }

```

Height 数组的应用

(1). 两个串的最长公共前缀

$$lcp(sa[i], sa[j]) = \min\{height[i + 1..j]\}$$

如果 *height* 一直大于某个数, 前这么多位就一直没变过; 反之, 由于后缀已经排好序了, 不可能变了之后变回来。

```

1 namespace SA {...}
2
3

```

(2). 比较一个字符串的两个子串的大小关系

假设需要比较的是 $A = S[a..b]$ 和 $B = S[c..d]$ 的大小关系。

若 $lcp(a, c) \geq \min(|A|, |B|)$, $A < B \iff |A| < |B|$ 。

否则, $A < B \iff rk[a] < rk[c]$ 。

(3). 不同子串的数目

子串就是后缀的前缀, 所以可以枚举每个后缀, 计算前缀总数, 再减掉重复。

“前缀总数” 其实就是子串个数, 为 $n(n+1)/2$ 。

如果按后缀排序的顺序枚举后缀, 每次新增的子串就是除了与上一个后缀的 LCP 剩下的前缀。这些前缀一定是新增的, 否则会破坏 $lcp(sa[i], sa[j]) = \min\{height[i+1..j]\}$ 的性质。只有这些前缀是新增的, 因为 LCP 部分在枚举上一个前缀时计算过了。

所以答案为: $\frac{n(n+1)}{2} - \sum_{i=2}^n height[i]$

O(n) 构造

- SA-IS
- 仅在后缀自动机被卡内存或者卡常且需要 O(1) LCA 的情况下使用 (比赛中敲这个我觉得不行)
- UOJ 35

```

1 // rk [0..n-1] -> [1..n], sa/ht [1..n]
2 // s[i] > 0 && s[n] = 0
3 // b: normally as bucket
4 // c: normally as bucket1
5 // d: normally as bucket2
6 // f: normally as cntbuf
7
8 template<size_t size>
9 struct SuffixArray {
10     bool t[size << 1];
11     int b[size], c[size];
12     int sa[size], rk[size], ht[size];
13     inline bool isLMS(const int i, const bool *t) { return i > 0 && t[i] && !t[i - 1]; }
14     template<class T>
15     inline void inducedSort(T s, int *sa, const int n, const int M, const int bs,
16                             bool *t, int *b, int *f, int *p) {
17         fill(b, b + M, 0); fill(sa, sa + n, -1);
18         FOR (i, 0, n) b[s[i]]++;
19         f[0] = b[0];
20         FOR (i, 1, M) f[i] = f[i - 1] + b[i];
21         FORD (i, bs - 1, -1) sa[--f[s[p[i]]]] = p[i];
22         FOR (i, 1, M) f[i] = f[i - 1] + b[i - 1];
23         FOR (i, 0, n) if (sa[i] > 0 && !t[sa[i] - 1]) sa[f[s[sa[i] - 1]]++] = sa[i] - 1;
24         f[0] = b[0];
25         FOR (i, 1, M) f[i] = f[i - 1] + b[i];
26         FORD (i, n - 1, -1) if (sa[i] > 0 && t[sa[i] - 1]) sa[--f[s[sa[i] - 1]]] = sa[i] - 1;
27     }
28     template<class T>
29     inline void sais(T s, int *sa, int n, bool *t, int *b, int *c, int M) {
30         int i, j, bs = 0, cnt = 0, p = -1, x, *r = b + M;
31         t[n - 1] = 1;
32         FORD (i, n - 2, -1) t[i] = s[i] < s[i + 1] || (s[i] == s[i + 1] && t[i + 1]);
33         FOR (i, 1, n) if (t[i] && !t[i - 1]) c[bs++] = i;
34         inducedSort(s, sa, n, M, bs, t, b, r, c);
35         for (i = bs = 0; i < n; i++) if (isLMS(sa[i], t)) sa[bs++] = sa[i];
36         FOR (i, bs, n) sa[i] = -1;
37         FOR (i, 0, bs) {
38             x = sa[i];
39             for (j = 0; j < n; j++) {

```

```

40         if (p == -1 || s[x + j] != s[p + j] || t[x + j] != t[p + j]) { cnt++; p = x; break; }
41         else if (j > 0 && (isLMS(x + j, t) || isLMS(p + j, t))) break;
42     }
43     x = (~x & 1 ? x >> 1 : x - 1 >> 1), sa[bs + x] = cnt - 1;
44 }
45 for (i = j = n - 1; i >= bs; i--) if (sa[i] >= 0) sa[j--] = sa[i];
46 int *s1 = sa + n - bs, *d = c + bs;
47 if (cnt < bs) sais(s1, sa, bs, t + n, b, c + bs, cnt);
48 else FOR (i, 0, bs) sa[s1[i]] = i;
49 FOR (i, 0, bs) d[i] = c[sa[i]];
50 inducedSort(s, sa, n, M, bs, t, b, r, d);
51 }
52 template<typename T>
53 inline void getHeight(T s, const int n, const int *sa) {
54     for (int i = 0, k = 0; i < n; i++) {
55         if (rk[i] == 0) k = 0;
56         else {
57             if (k > 0) k--;
58             int j = sa[rk[i] - 1];
59             while (i + k < n && j + k < n && s[i + k] == s[j + k]) k++;
60         }
61         ht[rk[i]] = k;
62     }
63 }
64 template<class T>
65 inline void init(T s, int n, int M) {
66     sais(s, sa, ++n, t, b, c, M);
67     for (int i = 1; i < n; i++) rk[sa[i]] = i;
68     getHeight(s, n, sa);
69 }
70 };
71
72 const int N = 2E5 + 100;
73 SuffixArray<N> sa;
74
75 int main() {
76     string s; cin >> s; int n = s.length();
77     sa.init(s, n, 128);
78     FOR (i, 1, n + 1) printf("%d%c", sa.sa[i] + 1, i == _i - 1 ? '\n' : ' ');
79     FOR (i, 2, n + 1) printf("%d%c", sa.ht[i], i == _i - 1 ? '\n' : ' ');
80 }

```

后缀平衡树

后缀之间的大小由字典序定义，后缀平衡树就是一个维护这些后缀顺序的平衡树，即字符串 T 的后缀平衡树是 T 所有后缀的有序集合。后缀平衡树上的一个节点相当于原字符串的一个后缀。

特别地，后缀平衡树的中序遍历即为后缀数组。

对长度为 n 的字符串 T 建立其后缀平衡树，考虑逆序将其后缀加入后缀平衡树。

记后缀平衡树维护的集合为 X ，当前添加的后缀为 S ，则添加下一个后缀就是向 X 中加入 $\mathbf{c}S$ （亦可理解为后缀平衡树维护的字符串为 S ，下一步往 S 前加入一个字符 \mathbf{c} ）。这一操作其实就是向平衡树中插入节点。这里使用期望树高为 $O(\log n)$ 的平衡树，例如替罪羊树或 Treap 等。

做法 1

插入时，暴力比较两个后缀之间的大小关系，从而判断之后是往哪一个子树添加。这样子，单次插入至多比较 $O(\log n)$ 次，单次比较的时间复杂度至多为 $O(n)$ ，一共 $O(n \log n)$ 。

一共会插入 n 次，所以该做法的时间复杂度存在上界 $O(n^2 \log n)$ 。

做法 2

注意到 $\mathbf{c}S$ 与 S 的区别仅在于 \mathbf{c} ，且 S 已经属于 X 了，可以利用这一点来优化插入操作。

假设当前要比较 $\mathbf{c}S$ 与 A 两个字符串的大小，且 $A, S \in X$ 。每次比较时，首先比较两串的首字符。若首字符不等，则两串的大小关系就已经确定了；若首字符相等，那么就只需要判断去除首字符后两字符串的大小关系。而两串去除首字符后都已经属于 X 了，这时候可

以借助平衡树 $O(\log n)$ 求排名的操作来完成后续的比较。这样，单次插入的操作至多 $O(\log^2 n)$ 。

一共会插入 n 次，所以该做法的时间复杂度存在上界 $O(n \log^2 n)$ 。

做法 3

根据做法 2，如果能够 $O(1)$ 判断平衡树中两个节点之间的大小关系，那么就可以在 $O(n \log n)$ 的时间内完成后缀平衡树的构造。

记 val_i 表示节点 i 的值。如果在建平衡树时，每个节点多维护一个标记 tag_i ，使得若 $tag_i > tag_j \iff val_i > val_j$ ，那么就可以根据 tag_i 的大小 $O(1)$ 判断平衡树中两个节点的大小。

不妨令平衡树中每个节点对应一个实数区间，令根节点对应 $(0, 1)$ 。对于节点 i ，记其对应的实数区间为 (l, r) ，则 $tag_i = \frac{l+r}{2}$ ，其左子树对应实数区间 (l, tag_i) ，其右子树对应实数区间 (tag_i, r) 。易证 tag_i 满足上述要求。

由于使用了期望树高为 $O(\log n)$ 的平衡树，所以精度是有一定保证的。实际实现时也可以用一个较大的区间来做，例如让根对应 $(0, 10^{18})$ 。

```
1  #include <bits/stdc++.h>
2  #define int long long
3  #define endl '\n'
4  using namespace std;
5
6  const int N = 1e6 + 10, INF = 1e18;
7  const double alpha = 0.75;
8
9  string t;
10
11 namespace SBT{
12     #define ls tree[rt].lc
13     #define rs tree[rt].rc
14
15     struct node {
16         int lc, rc;
17         int sz;
18     } tree[N];
19
20     int cnt, root;
21     double tag[N];
22
23     void calc(int rt) {
24         if(!rt) return;
25         tree[rt].sz = tree[ls].sz + tree[rs].sz + 1;
26     }
27
28     bool can_rebuild(int rt) {
29         return alpha * tree[rt].sz <= (double)max(tree[ls].sz, tree[rs].sz);
30     }
31
32     int ldr[N];
33
34     void flatten(int &lrc, int rt) {
35         if(!rt) return;
36         flatten(lrc, ls);
37         ldr[lrc++] = rt;
38         flatten(lrc, rs);
39     }
40
41     int rebuild_bd(int l, int r, double lv, double rv) {
42         if(l >= r) return 0;
43         int mid = l + r >> 1;
44         double mv = (lv + rv) / 2;
45         tag[ldr[mid]] = mv;
46         tree[ldr[mid]].lc = rebuild_bd(l, mid, lv, mv);
47         tree[ldr[mid]].rc = rebuild_bd(mid + 1, r, mv, rv);
48         calc(ldr[mid]);
49         return ldr[mid];
50     }
51
52     void rebuild(int &rt, double lv, double rv) {
53         int ldc = 0;
```

```

54     flatten(ldc, rt);
55     rt = rebuild_bd(0, ldc, lv, rv);
56 }
57
58 bool cmp(int x, int y) {
59     if(t[x] != t[y]) return t[x] < t[y];
60     return tag[x + 1] < tag[y + 1];
61 }
62
63 void insert(int &rt, int k, double lv, double rv) {
64     if(!rt) {
65         rt = k;
66         tree[rt].sz = 1;
67         tag[rt] = (lv + rv) / 2;
68         ls = rs = 0;
69         return;
70     }
71     if(cmp(k, rt)) insert(ls, k, lv, tag[rt]);
72     else insert(rs, k, tag[rt], rv);
73     calc(rt);
74     if(can_rebuild(rt)) rebuild(rt, lv, rv);
75 }
76
77 void del(int &rt, int k, double lv, double rv) {
78     if(!rt) return;
79     if(rt == k) {
80         if(!ls || !rs) {
81             rt = ls | rs;
82         } else {
83             int nrt = ls, fa = rt;
84             while(tree[nrt].rc) fa = nrt, tree[fa].sz--, nrt = tree[nrt].rc;
85             if(fa == rt) tree[nrt].rc = rs;
86             else tree[nrt].lc = ls, tree[nrt].rc = rs, tree[fa].rc = 0;
87             rt = nrt;
88             tag[rt] = (lv + rv) / 2;
89         }
90     } else {
91         double mv = (lv + rv) / 2;
92         if(cmp(k, rt)) del(ls, k, lv, mv);
93         else del(rs, k, mv, rv);
94     }
95     calc(rt);
96     if(can_rebuild(rt)) rebuild(rt, lv, rv);
97 }
98 } // Suffix Balanced Tree
99
100 int sa[N], tot = 0;
101
102 void get_sa(int rt) {
103     using SBT::tree;
104     if(!rt) return;
105     get_sa(tree[rt].lc);
106     sa[++tot] = rt;
107     get_sa(tree[rt].rc);
108 }
109
110 inline void solve() {
111     cin >> t; int n = t.size();
112     t = '@' + t;
113     for(int i = n; i >= 1; i--) SBT::insert(SBT::root, i, 0, INF);
114     get_sa(SBT::root);
115     for(int i = 1; i <= n; i++) cout << sa[i] << " \n"[i == n];
116 }
117
118 signed main() {
119     ios_base::sync_with_stdio(false), cin.tie(0);
120     int t = 1; // cin >> t;
121     while(t--) solve();
122     return 0;
123 }

```