

ACM TEMPLATE



Fibonacci's Rabbit

Last build at May 10, 2019

Contents

1	字符串	2
1.1	字符串最小最大表示	2
1.2	kmp 算法	2
1.3	z 函数	2
1.4	manacher	3
1.5	字典树	3
1.6	ac 自动机	4
1.7	后缀数组	5
1.7.1	后缀数组-倍增法	5
1.7.2	后缀数组-dc3	7
1.7.3	后缀数组-快排	8
1.8	字符串哈希	9
1.9	字符串分割	9
1.9.1	按字符分割	9
1.9.2	按字符串分割	10
1.9.3	按字符分割 (STL)	10

1 字符串

1.1 字符串最小最大表示

```

1 #include <algorithm>
2 using namespace std;
3
4 // T = sec[k..n-1]+sec[0..k-1]
5 // k为返回值,n为sec的大小,T为sec的最小表示法
6 int get_min(const char* sec, int n) {
7     int k = 0, i = 0, j = 1;
8     while (k < n && i < n && j < n) {
9         if (sec[(i + k) % n] == sec[(j + k) % n]) {
10             k++;
11         } else {
12             sec[(i + k) % n] > sec[(j + k) % n] ? i = i + k + 1 : j = j + k + 1;
13             if (i == j) i++;
14             k = 0;
15         }
16     }
17     i = min(i, j);
18     return i;
19 }
20
21 int get_max(const char* sec, int n) {
22     int k = 0, i = 0, j = 1;
23     while (k < n && i < n && j < n) {
24         if (sec[(i + k) % n] == sec[(j + k) % n]) {
25             k++;
26         } else {
27             sec[(i + k) % n] < sec[(j + k) % n] ? i = i + k + 1 : j = j + k + 1;
28             if (i == j) i++;
29             k = 0;
30         }
31     }
32     i = min(i, j);
33     return i;
34 }

```

1.2 kmp 算法

以 i 结尾的最小循环节: $i - f[i]$

```

1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5 const int maxn = 10000 + 5;
6
7 int f[maxn];
8
9 void get_next(const char *P, int n) {
10     f[0] = 0;
11     f[1] = 0; // 递推边界初值
12     for (int i = 1; i < n; i++) {
13         int j = f[i];
14         while (j && P[i] != P[j]) j = f[j];
15         f[i + 1] = (P[i] == P[j] ? j + 1 : 0);
16     }
17 }

```

1.3 z 函数

```

1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5 const int maxn = 1000000 + 5;
6
7 int z[maxn];
8
9 // s 为待匹配的字符串指针
10 // n 为字符串长度
11 // z[i]是s和s+i的最大公共前缀长度。
12 void z_function(const char* s, int n) {
13     fill_n(z, n, 0);
14     for (int i = 1, l = 0, r = 0; i < n; ++i) {

```

```

15     if (i <= r) z[i] = min(r - i + 1, z[i - 1]);
16     while (i + z[i] < n && s[z[i]] == s[i + z[i]]) ++z[i];
17     if (i + z[i] - 1 > r) l = i, r = i + z[i] - 1;
18 }
19 }

```

1.4 manacher

回文匹配算法 (可用后缀数组代替, 但是比后缀数组简洁得多)

```

1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5 const int maxn = 1000000;
6 int d1[maxn], d2[maxn];
7
8 // s 为字符串, 也可以是const string&
9 // n 是字符串长度, 即为s.length()
10 // d1 为奇数回文长度(算上起点), 总长度为d1[*]*2-1
11 // d2 为偶数回文长度(算上起点), 总长度为d2[*]*2
12 void Manacher(const char* s, int n) {
13     for (int i = 0, l = 0, r = -1; i < n; i++) {
14         int k = (i > r) ? 1 : min(d1[l + r - i], r - i);
15         while (0 <= i - k && i + k < n && s[i - k] == s[i + k]) {
16             k++;
17         }
18         d1[i] = k--;
19         if (i + k > r) {
20             l = i - k;
21             r = i + k;
22         }
23     }
24
25     for (int i = 0, l = 0, r = -1; i < n; i++) {
26         int k = (i > r) ? 0 : min(d2[l + r - i + 1], r - i + 1);
27         while (0 <= i - k - 1 && i + k < n && s[i - k - 1] == s[i + k]) {
28             k++;
29         }
30         d2[i] = k--;
31         if (i + k > r) {
32             l = i - k - 1;
33             r = i + k;
34         }
35     }
36 }
37
38 // 判断[l,r)是否回文
39 bool is_palindrome(int l, int r) {
40     if (l == r) return true;
41     if ((r - l) & 1) {
42         return d1[l + (r - l) / 2] >= (r - l + 1) / 2;
43     } else {
44         return d2[l + (r - l) / 2] >= (r - l) / 2;
45     }
46 }

```

1.5 字典树

```

1 #include <cstring>
2 #include <vector>
3 using namespace std;
4
5 const int wordnum = 100;
6 const int wordlen = 4000;
7 const int maxnode = wordnum * wordlen + 10;
8 const int sigma_size = 26;
9
10 // 字母表为全体小写字母的Trie
11 struct Trie {
12     int ch[maxnode][sigma_size];
13     int val[maxnode];
14     int sz; // 结点总数
15     void clear() {
16         sz = 1;
17         memset(ch[0], 0, sizeof(ch[0]));
18     }
19     // 初始时只有一个根结点
20     int idx(char c) { return c - 'a'; } // 字符c的编号

```

```

20
21 // 插入字符串s, 附加信息为v。注意v必须非0, 因为0代表“本结点不是单词结点”
22 void insert(const char *s, int v) {
23     int u = 0, n = strlen(s);
24     for (int i = 0; i < n; i++) {
25         int c = idx(s[i]);
26         if (!ch[u][c]) { // 结点不存在
27             memset(ch[sz], 0, sizeof(ch[sz]));
28             val[sz] = 0; // 中间结点的附加信息为0
29             ch[u][c] = sz++; // 新建结点
30         }
31         u = ch[u][c]; // 往下走
32     }
33     val[u] = v; // 字符串的最后一个字符的附加信息为v
34 }
35 };

```

1.6 ac 自动机

```

1 #include <cstring>
2 #include <queue>
3
4 using namespace std;
5
6 const int SIGMA_SIZE = 128;
7 const int WORD_SIZE = 55;
8 const int WORD_NUM = 1005;
9 const int MAXNODE = WORD_SIZE * WORD_NUM + 10;
10
11 struct AhoCorasickAutomata {
12     int ch[MAXNODE][SIGMA_SIZE];
13     int f[MAXNODE]; // fail函数
14     int val[MAXNODE]; // 每个字符串的结尾结点都有一个非0的val
15     int last[MAXNODE]; // 输出链表的下一个结点
16     bool vis[MAXNODE];
17     int cnt[WORD_NUM];
18     int sz;
19
20     void init() {
21         sz = 1;
22         memset(ch[0], 0, sizeof(ch[0]));
23         memset(vis, 0, sizeof(vis));
24         memset(cnt, 0, sizeof(cnt));
25     }
26
27     // 字符c的编号
28     int idx(char c) const { return c; }
29
30     // 插入字符串。v必须非0
31     void insert(char* s, int v) {
32         int u = 0, n = strlen(s);
33         for (int i = 0; i < n; i++) {
34             int c = idx(s[i]);
35             if (!ch[u][c]) {
36                 memset(ch[sz], 0, sizeof(ch[sz]));
37                 val[sz] = 0;
38                 ch[u][c] = sz++;
39             }
40             u = ch[u][c];
41         }
42         val[u] = v;
43     }
44
45     // 递归打印以结点j结尾的所有字符串
46     void print(int j) {
47         int ret = 0;
48         if (j) {
49             cnt[val[j]]++;
50             print(last[j]);
51         }
52     }
53
54     // 在T中找模板
55     void find(const char* T) {
56         int n = strlen(T);
57         int j = 0; // 当前结点编号, 初始为根结点
58         for (int i = 0; i < n; i++) { // 文本串当前指针
59             int c = idx(T[i]);
60             while (j && !ch[j][c]) j = f[j]; // 顺着细边走, 直到可以匹配

```

```

61         j = ch[j][c];
62         if (val[j])
63             print(j);
64         else if (last[j])
65             print(last[j]); // 找到了!
66     }
67 }
68
69 // 计算fail函数
70 void getFail() {
71     queue<int> q;
72     f[0] = 0;
73     // 初始化队列
74     for (int c = 0; c < SIGMA_SIZE; c++) {
75         int u = ch[0][c];
76         if (u) {
77             f[u] = 0;
78             q.push(u);
79             last[u] = 0;
80         }
81     }
82     // 按BFS顺序计算fail
83     while (!q.empty()) {
84         int r = q.front();
85         q.pop();
86         for (int c = 0; c < SIGMA_SIZE; c++) {
87             int u = ch[r][c];
88             if (!u) continue;
89             q.push(u);
90             int v = f[r];
91             while (v && !ch[v][c]) v = f[v];
92             f[u] = ch[v][c];
93             last[u] = val[f[u]] ? f[u] : last[f[u]];
94         }
95     }
96 }
97 }; // namespace AhoCorasickAutomata
98
99 AhoCorasickAutomata ac;

```

1.7 后缀数组

全字符串找循环节 要有长度为 i 的循环节，就要满足以下条件：

$$\begin{aligned}
 rank[0] - rank[i] &= 1 \\
 height[rank[0]] &= len - i \\
 len \% i &= 0
 \end{aligned}$$

找字符串循环节最大重复次数 枚举长度 len ，枚举起点 j ，求 $lcp(j, j + len)$

$$\begin{aligned}
 ans &= lcp / len + 1 \\
 k &= j - (len - ans \% len) \\
 if (k > 0 \&\& lcp(k, k + len) \geq len) \{ ans ++; \}
 \end{aligned}$$

求 ans 最大值

复杂度

- * 后缀数组倍增法（时间 $O(n \log n)$ ，空间 $O(4n)$ ）
- * 后缀数组 dc3 法（时间 $O(n)$ ，空间 $O(10n)$ ）
- * 后缀数组快排（适用于最大值很大的情况，除了 sa 数组，其他暂未测试）

1.7.1 后缀数组-倍增法

```

1 #include <algorithm>
2 #include <cstdio>
3 #include <cstring>
4 using namespace std;
5
6 namespace SuffixArray {
7     using std::printf;
8
9     const int maxn = 1e7 + 5; // max(字符串长度, 最大字符值加1)

```

```

10
11 int s[maxn]; // 原始字符数组（最后一个字符应必须是0，而前面的字符必须非0）
12 int sa[maxn]; // 后缀数组
13 int rank[maxn]; // 名次数组. rank[0]一定是n-1，即最后一个字符
14 int height[maxn]; // height数组
15 int t[maxn], t2[maxn], c[maxn]; // 辅助数组
16 int n; // 字符个数（包括最后一个0字符）
17
18 void init() { n = 0; }
19
20 // m为最大字符值加1。调用之前需设置好s和n
21 void build_sa(int m) {
22     int i, *x = t, *y = t2;
23     for (i = 0; i < m; i++) c[i] = 0;
24     for (i = 0; i < n; i++) c[x[i]]++;
25     for (i = 1; i < m; i++) c[i] += c[i - 1];
26     for (i = n - 1; i >= 0; i--) sa[--c[x[i]]] = i;
27     for (int k = 1; k <= n; k <= 1) {
28         int p = 0;
29         for (i = n - k; i < n; i++) y[p++] = i;
30         for (i = 0; i < n; i++)
31             if (sa[i] >= k) y[p++] = sa[i] - k;
32         for (i = 0; i < m; i++) c[i] = 0;
33         for (i = 0; i < n; i++) c[x[y[i]]]++;
34         for (i = 0; i < m; i++) c[i] += c[i - 1];
35         for (i = n - 1; i >= 0; i--) sa[--c[x[y[i]]]] = y[i];
36         swap(x, y);
37         p = 1;
38         x[sa[0]] = 0;
39         for (i = 1; i < n; i++) x[sa[i]] = y[sa[i - 1]] == y[sa[i]] && y[sa[i - 1] + k] == y[sa[i] + k] ? p - 1 : p++;
40         if (p >= n) break;
41         m = p;
42     }
43 }
44
45 void build_height() {
46     int i, k = 0;
47     for (i = 0; i < n; i++) rank[sa[i]] = i;
48     for (i = 0; i < n; i++) {
49         if (k) k--;
50         int j = sa[rank[i] - 1];
51         while (s[i + k] == s[j + k]) k++;
52         height[rank[i]] = k;
53     }
54 }
55 } // namespace SuffixArray
56
57 // 编号辅助
58 namespace SuffixArray {
59     int idx[maxn];
60
61     // 给字符串加上一个字符，属于字符串i
62     void add(int ch, int i) {
63         idx[n] = i;
64         s[n++] = ch;
65     }
66 } // namespace SuffixArray
67
68 // LCP 模板
69 namespace SuffixArray {
70     using std::min;
71     int dp[maxn][20];
72     void initRMQ(int n) {
73         for (int i = 1; i <= n; i++) dp[i][0] = height[i];
74         for (int j = 1; (1 << j) <= n; j++)
75             for (int i = 1; i + (1 << j) - 1 <= n; i++) dp[i][j] = min(dp[i][j - 1], dp[i + (1 << (j - 1))]
76                 [j - 1]);
77         return;
78     }
79
80     void initRMQ() { initRMQ(n - 1); }
81
82     int lcp(int a, int b) {
83         int ra = rank[a], rb = rank[b];
84         if (ra > rb) swap(ra, rb);
85         int k = 0;
86         while ((1 << (k + 1)) <= rb - ra) k++;
87         return min(dp[ra + 1][k], dp[rb - (1 << k) + 1][k]);
88     }
89 }

```

```

88 } // namespace SuffixArray
89
90 // 调试信息
91 namespace SuffixArray {
92     using std::printf;
93     void debug() {
94         printf("n:%d\n", n);
95
96         printf("%8s", "");
97         for (int i = 0; i < n; i++) {
98             printf("%4d", i);
99         }
100         printf("\n");
101
102         printf("%8s", "s:");
103         for (int i = 0; i < n; i++) {
104             printf("%4d", s[i]);
105         }
106         printf("\n");
107
108         printf("%8s", "sa:");
109         for (int i = 0; i < n; i++) {
110             printf("%4d", sa[i]);
111         }
112         printf("\n");
113
114         printf("%8s", "rank:");
115         for (int i = 0; i < n; i++) {
116             printf("%4d", rank[i]);
117         }
118         printf("\n");
119
120         printf("%8s", "height:");
121         for (int i = 0; i < n; i++) {
122             printf("%4d", height[i]);
123         }
124         printf("\n");
125     }
126 } // namespace SuffixArray

```

1.7.2 后缀数组-dc3

```

1  #include <algorithm>
2
3  using namespace std;
4
5  /*
6  注意:
7  1.maxn开n的十倍大小;
8  2.dc3(r,sa,n+1,Max+1);r为待后缀处理的数组,sa为存储排名位置的数组,n+1和Max+1都和倍增一样
9  3.calheight(r,sa,n);和倍增一样
10 */
11 // DC3 算法
12 namespace SuffixArray {
13     #define F(x) ((x) / 3 + ((x) % 3 == 1 ? 0 : tb))
14     #define G(x) ((x) < tb ? (x)*3 + 1 : ((x)-tb) * 3 + 2)
15
16     const int maxn = 1e7 + 5;
17
18     int wa[maxn], wb[maxn], wv[maxn], ws[maxn];
19     int s[maxn], sa[maxn];
20     int rank[maxn], height[maxn];
21     int n;
22
23     void init() { n = 0; }
24
25     int c0(int *r, int a, int b) { return r[a] == r[b] && r[a + 1] == r[b + 1] && r[a + 2] == r[b + 2]; }
26
27     int c12(int k, int *r, int a, int b) {
28         if (k == 2)
29             return r[a] < r[b] || (r[a] == r[b] && c12(1, r, a + 1, b + 1));
30         else
31             return r[a] < r[b] || (r[a] == r[b] && wv[a + 1] < wv[b + 1]);
32     }
33
34     void sort(int *r, int *a, int *b, int n, int m) {
35         int i;
36         for (i = 0; i < n; i++) wv[i] = r[a[i]];
37         for (i = 0; i < m; i++) ws[i] = 0;

```



```

38     for (i = 0; i < n; i++) ws[wv[i]]++;
39     for (i = 1; i < m; i++) ws[i] += ws[i - 1];
40     for (i = n - 1; i >= 0; i--) b[ws[wv[i]]] = a[i];
41     return;
42 }
43
44 void dc3(int *r, int *sa, int n, int m) {
45     int i, j, *rn = r + n, *san = sa + n, ta = 0, tb = (n + 1) / 3, tbc = 0, p;
46     r[n] = r[n + 1] = 0;
47     for (i = 0; i < n; i++)
48         if (i % 3 != 0) wa[tbc++] = i;
49     sort(r + 2, wa, wb, tbc, m);
50     sort(r + 1, wb, wa, tbc, m);
51     sort(r, wa, wb, tbc, m);
52     for (p = 1, rn[F(wb[0])] = 0, i = 1; i < tbc; i++) rn[F(wb[i])] = c0(r, wb[i - 1], wb[i]) ? p - 1
53         : p++;
54     if (p < tbc)
55         dc3(rn, san, tbc, p);
56     else
57         for (i = 0; i < tbc; i++) san[rn[i]] = i;
58     for (i = 0; i < tbc; i++)
59         if (san[i] < tb) wb[ta++] = san[i] * 3;
60     if (n % 3 == 1) wb[ta++] = n - 1;
61     sort(r, wb, wa, ta, m);
62     for (i = 0; i < tbc; i++) wv[wb[i] = G(san[i])] = i;
63     for (i = 0, j = 0, p = 0; i < ta && j < tbc; p++) sa[p] = c12(wb[j] % 3, r, wa[i], wb[j]) ? wa[i]
64         : wb[j++];
65     for (; i < ta; p++) sa[p] = wa[i++];
66     for (; j < tbc; p++) sa[p] = wb[j++];
67     return;
68 }
69
70 void build_height(int n) {
71     int i, j, k = 0;
72     for (i = 1; i <= n; i++) rank[sa[i]] = i;
73     for (i = 0; i < n; height[rank[i++]] = k)
74         for (k ? k-- : 0, j = sa[rank[i] - 1]; s[i + k] == s[j + k]; k++);
75     return;
76 }
77
78 void build_height() { build_height(n - 1); }
79
80 void build_sa(int m) { dc3(s, sa, n, m); }
81 } // namespace SuffixArray

```

1.7.3 后缀数组-快排

```

1  #include <cstdio>
2  #include <algorithm>
3  #include <cstring>
4  using namespace std;
5
6  namespace SuffixArray {
7      using std::printf;
8
9      const int maxn = 1e7 + 5; // max(字符串长度, 最大字符值加1)
10
11     int s[maxn]; // 原始字符数组 (最后一个字符应必须是0, 而前面的字符必须非0)
12     int sa[maxn]; // 后缀数组
13     int t[maxn], rank[maxn], c[maxn]; // 辅助数组
14     int n; // 字符个数 (包括最后一个0字符)
15
16     void init() { n = 0; }
17
18     int k;
19     bool compare_sa(int i, int j) {
20         if (rank[i] != rank[j]) {
21             return rank[i] < rank[j];
22         } else {
23             int ri = i + k < n ? rank[i + k] : -1;
24             int rj = j + k < n ? rank[j + k] : -1;
25             return ri < rj;
26         }
27     }
28
29     void build_sa(int _) {
30         for (int i = 0; i < n; i++) {

```

```

31         sa[i] = i;
32         rank[i] = i < n ? s[i] : -1;
33     }
34     for (k = 1; k < n; k <= 1) {
35         sort(sa, sa + n, compare_sa);
36         t[sa[0]] = 0;
37         for (int i = 1; i < n; i++) {
38             t[sa[i]] = t[sa[i - 1]] + (compare_sa(sa[i - 1], sa[i]) ? 1 : 0);
39         }
40         for (int i = 0; i < n; i++) {
41             rank[i] = t[i];
42         }
43     }
44 }
45
46 int height[maxn]; // height数组
47 void build_height() {
48     int i, k = 0;
49     for (i = 0; i < n; i++) {
50         if (k) k--;
51         int j = sa[rank[i] - 1];
52         while (s[i + k] == s[j + k]) k++;
53         height[rank[i]] = k;
54     }
55 }
56 } // namespace SuffixArray

```

1.8 字符串分割

1.8.1 按字符分割

```

1 #include <iostream>
2 #include <cstring>
3 #include <vector>
4 using namespace std;
5
6 // 字符串分割，分隔符为字符，可为多字符，前后不留空字符串
7 // *a,b*c,d, 按*分割 -> {"a","b","c","d"}
8 // 注意：源字符串s将会被改变，请勿使用string.c_str()
9 // s源字符串 t传出结果 sep分隔符字符串(分隔符为每个单字符)
10 void split(char *s, vector<string> &v, const char *sep) {
11     char *p = strtok(s, sep);
12     while (p) {
13         v.push_back(string(p));
14         p = strtok(NULL, sep);
15     }
16 }

```

1.8.2 按字符串分割

```

1 #include <iostream>
2 #include <string>
3 #include <vector>
4
5 using namespace std;
6
7 // 字符串分割，分隔符为字符串，前后留空字符串
8 // cabcacac 按c分割 -> {"", "ab", "a", "a", ""}
9 // s源字符串 v传出结果 c分隔符字符串
10 void split(const string& s, vector<string>& v, const string& c) {
11     string::size_type pos1, pos2;
12     pos2 = s.find(c);
13     pos1 = 0;
14     while (string::npos != pos2) {
15         v.push_back(s.substr(pos1, pos2 - pos1));
16
17         pos1 = pos2 + c.size();
18         pos2 = s.find(c, pos1);
19     }
20     if (pos1 <= s.length()) v.push_back(s.substr(pos1));
21     // 如果要去除最后空串,用下方语句替代上一条
22     // if (pos1 != s.length()) v.push_back(s.substr(pos1));
23 }

```

1.8.3 按字符分割 (STL)

```

1  #include <iostream>
2  #include <string>
3  #include <vector>
4  using namespace std;
5
6  // 字符串分割，分隔符为字符，可为多字符，前后不留空字符串
7  // **a,b*c,d, 按,*分割 -> {"a","b","c","d"}
8  // strtok 的实现
9  // s源字符串 t传出结果 sep分隔符字符串(分隔符为每个单字符)
10 void split(const string &s, vector<string> &v, const string &sep) {
11     typedef string::size_type string_size;
12     string_size i = 0;
13     while (i != s.size()) {
14         //找到字符串中首个不等于分隔符的字母;
15         int flag = 0;
16         while (i != s.size() && flag == 0) {
17             flag = 1;
18             for (string_size x = 0; x < sep.size(); ++x) {
19                 if (s[i] == sep[x]) {
20                     ++i;
21                     flag = 0;
22                     break;
23                 }
24             }
25         }
26
27         //找到又一个分隔符，将两个分隔符之间的字符串取出;
28         flag = 0;
29         string_size j = i;
30         while (j != s.size() && flag == 0) {
31             for (string_size x = 0; x < sep.size(); ++x) {
32                 if (s[j] == sep[x]) {
33                     flag = 1;
34                     break;
35                 }
36             }
37             if (flag == 0) ++j;
38         }
39         if (i != j) {
40             v.push_back(s.substr(i, j - i));
41             i = j;
42         }
43     }
44 }

```