Chapter 1

字符串

1.1 Shift-And 算法

基于位并行的算法，如 Shift-And 算法的基本思想是：将模式串集合与文本串的匹配状态用位向量存储，匹配过程就是用位操作更新位向量的过程。它的优点是所需存储空间小，匹配速度快；缺点是算法性能会随模式串个数的增多而下降，只适合中小规模的模式串集合。

Shift-And 算法维护一个字符串的集合，集合中的每个字符串既是模式串 p 的前缀，同时也是已读入文本的后缀。每读入一个新的文本字符，该算法采用位并行的方法更新该集合，该集合用一个位掩码 $D = d_m...d_1$ 来表示。D 的第 j 位被置为 1，当且仅当 $p_1...p_j$ 是 $t_1...t_i$ 的后缀。

Shift-And 算法首先构造一个 m 位 (m 是模式串的长度) 的向量表 $B[]$，用来记录字符在模式串的出现位置。如果 p_j 为字符 c，掩码 $B[c]$ 的第 j 位被置为 1，否则为 0。首先置 $D = 0^m$，对于每个新读入的文本字符 t_{i+1}，用如下公式对 D 进行更新：$D[i+1] = ((D[i] << 1)|0^{m+1}1)\&B[t_{i+1}]$。在匹配时，逐个扫描文本字符并更新向量 D，当 $D[i]\&10^{m-1}$ 0^m 时，在文本位置 i 处匹配成功。

Shift-And 算法扩展到多模式串时，将所有模式串的位向量 D 包装到一个机器字里，用位并行技术同时对 r 个位向量进行更新，初始化字和匹配掩码分别是所有初始化字和所有匹配掩码的连接。

设机器字的长度为 w，文本串的长度为 n，模式串的个数为 r，最短模式串长度为 m，那么 Shift-And 算法的时间复杂度为 $O(n\lceil\frac{m*r}{w}\rceil)$。由于采用了位并行技术，Shift-And 算法的匹配速度是很快的。但一旦模式串的长度和超出机器字的长度，算法的性能都会发生明显下降。
[2016 大连 B]

给一个 $n \leq 1000$，代表数字长度，以及每位上候选数字集合，再给一个数字字符串 $s(|s| \leq 5 * 10^6)$，输出 s 中所有匹配的 n 位数字子串。
样例输入：
4 （一共四位）
3 0 9 7 （第一位有三个候选数字分别为：0 9 7）
2 5 7 （第二位有两个候选数字分别为：5 7）
2 2 5 （第三位有两个候选数字分别为：2 5）
2 4 5 （第四位有两个候选数字分别为：4 5）
09755420524 （数字字符串 s）
样例输出：（所有匹配的四位数字子串）
9755
7554
0524

```
#include <inttypes.h>

const int MAX_N = 1005;
const int MAX_LEN = 10000005;
const int MAX_ARR_LEN = ((MAX_N >> 6) + 5);
const int MASK = 63;

int n;
```

```
9  ll num[10][MAX_ARR_LEN], ret_n[MAX_ARR_LEN];
10 ll ind_x_arr[MAX_N], ind_y_arr[MAX_N];
11 char str[MAX_LEN];
12
13 void init() {
14     memset(num, 0, sizeof (num));
15     memset(ret_n, 0, sizeof (ret_n));
16     for (int i = 0; i < n; ++i) {
17         ind_x_arr[i] = (i >> 6) + 1; // i 位置属于哪一段
18         ind_y_arr[i] = 1ll << (i & MASK); // 二进制中对应的位置
19     }
20 }
21
22 inline void set_one(ll *arr, int pos) {
23     arr[ind_x_arr[pos]] |= ind_y_arr[pos];
24 }
25
26 inline bool seek_one(ll *arr, int pos) {
27     return arr[ind_x_arr[pos]] & ind_y_arr[pos];
28 }
29
30 inline void left_move_one(ll *arr, int pos) {
31     for (int i = pos; i >= 1; --i) {
32         arr[i] <<= 1;
33         arr[i] |= (!!(arr[i - 1] & 0x8000000000000000ll));
34     }
35     set_one(arr, 0);
36 }
37
38 inline void and_opt(ll *arr1, ll *arr2, int pos) {
39     for (int i = 1; i <= pos; ++i) {
40         arr1[i] &= arr2[i];
41     }
42 }
43
44 int main() {
45     scanf("%d", &n);
46     init();
47     for (int i = 0; i < n; ++i) {
48         int x, y;
49         scanf("%d", &x);
50         for (int j = 0; j < x; ++j) {
51             scanf("%d", &y);
52             set_one(num[y], i);
53         }
54     }
55     scanf("%s", str);
56     int len = strlen(str);
57     int L = n >> 6;
58     if (n & MASK) L++; // 分成若干段数
59     for (int i = 0; i < len; ++i) {
60         left_move_one(ret_n, L);
61         and_opt(ret_n, num[str[i] - '0'], L);
62         if (seek_one(ret_n, n - 1)) {
63             char ch = str[i + 1];
64             str[i + 1] = '\0';
65             printf("%s\n", str + i - n + 1);
66             str[i + 1] = ch;
67         }
68     }
69     return 0;
70 }
```