0.1 bitset 优化

使 bitset 优化，利用 bitset 的位移特性和每一位 01 表示匹配状态。例如对于模式串：*abc* 和读入文本：*abcabcabc*。先根据读入文本得到每个字母向量表示 (从右往左看第 i 位为 1 表示读入文本的第 i 位（从左往右看）为该字母)：
bs[a]:001001001
bs[b]:010010010
bs[c]:100100100
用 dp 表示匹配状态，初始时：dp=111111111，扫描模式串:*abc*。

对于第一个字母 a，(dp « 1) & bs[a] 可得：dp = (111111110 & 001001001) = 001001001，这表示字母 a 可以在文本串中的哪些位置作为前缀。

对于第二个字母 b，(dp « 1) & bs[b] 可得：dp = (010010010 & 010010010) = 010010010，这表示 ab 可以在文本串中的哪些位置作为前缀。如果想要以当前 b 结尾作为前缀的话，那么必然需要以前一个字母 a 作为上一个字母的前缀，所以需要先 dp 左移一位，然后与上 当前 b 可以匹配的位置。

对于第三个字母 c，(dp « 1) & bs[c] 可得：dp = (100100100 & 100100100) = 100100100。此时状态 1 的位置就表示可以和模式串匹配的结尾位置。

[HDU 5745]
给一个长度 $n \leq 10^5$ 的文本串和长度为 $m \leq 5000$ 的模式串，对于文本串的每个字母可以选择相邻位置字母交换但是不允许交叉交换。例如 abcd 可以变换成 bacd,abdc,acbd，badc, 但是不能变成 bcad,bcda 等。对于文本串的每个位置判断以它为起始的子串能否变换成模式串。

其实就是限制了每个位置字母的交换位置只能是相邻的两个。用 $dp[i][j][]$ 表示文本串的第 i 个位置和模式串的第 j 个位置的匹配状态。第三维用 0,1,2 分别表示文本串的第 i 个字母和第 $i-1$ 个字母交换，不动以及和第 $i+1$ 个字母交换三种状态。状态转移：

$$dp[i][j][0] = dp[i-1][j-1][2] \;\&\&\; a[i] == b[j-1]$$
$$dp[i][j][1] = (dp[i-1][j-1][0] \mid dp[i-1][j][1]) \;\&\&\; a[i] == b[j]$$
$$dp[i][j][2] = (dp[i-1][j-1][0] \mid dp[i-1][j-1][1]) \;\&\&\; a[i] == b[j+1]$$

先处理出文本串中每个字母出现的位置，相当于状压第一维，然后枚举模式串的每个位置借助 bitset 左移操作模拟匹配并且滚动数组。
时间复杂度：$O(\frac{n*m}{w}), w$ 是机器字节数

```
const int MAX_N = 100010;
const int MAX_M = 5010;

int T, n, m;
int ans[MAX_N];
char str1[MAX_N], str2[MAX_M];
bitset<MAX_N> dp[2][3], bs[30];

void init() {
    for (int i = 0; i < 2; ++i) {
        for (int j = 0; j < 3; ++j) {
            dp[i][j].reset();
            dp[i][j][0] = 1;
        }
    }
    for (int i = 0; i < 26; ++i) { bs[i].reset(); }
    for (int i = 1; i <= n; ++i) {
        bs[str1[i] - 'a'][i] = 1;
    }
}

void solve() {
    init();
    int now = 0;
    dp[0][1].set(); // 初始置为1
```

```
    for (int i = 1; i <= m; ++i) {
        now ^= 1;
        if (i > 1) dp[now][0] = (dp[now ^ 1][2] << 1) & bs[str2[i - 1] - 'a'];
        dp[now][1] = ((dp[now ^ 1][1] | dp[now ^ 1][0]) << 1) & bs[str2[i] - 'a'];
        if (i <= m - 1)
            dp[now][2] = ((dp[now ^ 1][0] | dp[now ^ 1][1]) << 1) & bs[str2[i + 1] - 'a'];
        dp[now][0][0] = dp[now][1][0] = dp[now][2][0] = 1;
    }
    for (int i = 1; i <= n - m + 1; ++i) {
        if (dp[now][0][i + m - 1] || dp[now][1][i + m - 1]) printf("1");
        else printf("0");
    }
    for (int i = n - m + 2; i <= n; ++i) { // 最后的 m-1 个位置肯定不符
        printf("0");
    }
    printf("\n");
}

int main() {
    scanf("%d", &T);
    while (T--) {
        scanf("%d%d", &n, &m);
        scanf("%s%s", str1 + 1, str2 + 1);
        solve();
    }
    return 0;
}
```

[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

把 n 位数字看成模式串，先处理处每个数字可以在模式串中的匹配位置，然后扫描文本串。用 $dp[i][j]$ 表示文本串的第 i 个位置能否和模式串的第 j 个位置匹配（前缀），状态转移：

$$dp[i][j] = dp[i-1][j-1] \ \&\& \ a[i] \in b[j]$$

时间复杂度：$O(\frac{n*m}{w}), w$ 是机器字节数

```
const int MAX_M = 5000010;
const int MAX_N = 1010;

int n, len;
char str[MAX_M];
bitset<MAX_N> bs[10], dp[2];

void solve() {
    len = strlen(str + 1);
    dp[0].reset(), dp[1].reset();
    dp[0][0] = 1;
    int now = 0;
```

```
13      for (int i = 1; i <= len; ++i) {
14          now ^= 1;
15          dp[now] = (dp[now ^ 1] << 1) & bs[str[i] - '0'];
16          dp[now][0] = 1;
17          if (dp[now][n]) {
18              char ch = str[i + 1];
19              str[i + 1] = '\0';
20              printf("%s\n", str + (i - n + 1));
21              str[i + 1] = ch;
22          }
23      }
24  }
25
26  int main() {
27      while (~scanf("%d", &n)) {
28          for (int i = 0; i < 10; ++i) { bs[i].reset(); }
29          for (int i = 1; i <= n; ++i) {
30              int x, y;
31              scanf("%d", &x);
32              for (int j = 0; j < x; ++j) {
33                  scanf("%d", &y);
34                  bs[y][i] = 1;
35              }
36          }
37          scanf("%s", str + 1);
38          solve();
39      }
40      return 0;
41  }
```