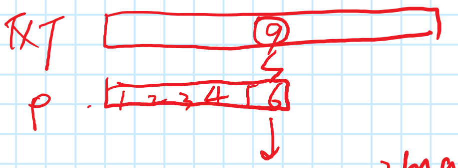
Boyer-Moore Algorithm

(more)

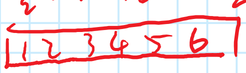
best : sub-linear. better than Knuth-Morris-Pratt

Worst : O(NM). as bad as brute-force.

Main idea is that compare from end of patten.



9和6 mismatch. 而9又不在pat中. 故9不可能对齐pat中的任何一位. 所以整体移动 pat 去对齐 9之后的一格. 这样一下子跳



过了M步. 而 brute force 和 K-M-P 都是1步步. 实际上 应用中经常都是这样的情况, 因此可能达到 ~(N/M) 复杂度. 所以这个算法很快. 可以是最 efficient 的算法. 而且 pat 越长 算法越快.

以上是 case 1. case 2 是 mismatch. 但这个 TXT[i+j] 还在 pat中, 那么找到这个 C 在PAT中的最右出现位置. 用这个位置去对准 C. (pat) 也就是跳 j - right[c] 格. 而 rightmost 保证跳得最小. 不会跳过头. 但如果 right[c] 在j之后. 那就为负了, 即要回退. 这是不必要的. 此时 类似 Brute Force 只跳一格 即 max(1, j - right[c]).

根据这样的规则. 可以 preprocess 建立 right 表格.

R:

	NEEDLE	right[ ]
A		-1
B		-1
C		-1
D	2 3	3
E	1 5	5

```
right = new int[R]
for (int i=0; i<R; i++)
    r[i] = -1;
```

D	2 3	3
E	1 5	5
⋮		⋮
L	4	4
⋮		⋮
N	0	0

```
for (int i=0; i<R; i++)
    right[i] = -1;
for (int p=0; p<M; p++)
    right[~~p~~ → pat[p]] = p;
```

以后这个程序类似于 Brute-Force 的第二版.

```
skip=0;
for( int i=0; i<N; i+=skip) {
    skip=0;
    for(int j=M-1; j>=0; j--) {
        if( TXT[i+j] != PAT[j]) {
            ~~skip = j - right[TXT[i+j]];~~
            skip=max(1, j-right[TXT[i+j]]);
            break;
        }
    }
    if (skip==0) return i;  // found
}
return N;  // not found.
```

Mark: 程序选择要的 KMP 的字里面
∴ 但 pattern 时较长 也的 kmp 好.