→ the efficiency of Boyer-Moore algorithm relies on creating a lookup table that quickly determines where a mismatched character occurs elsewhere in the pattern
function last(c): if c is in P, last(c) is index of last occurrence of c in P.
Otherwise, last(c) = -1

→ instead, can use hash table to represent last function with only those characters from the pattern occurring in the structure. Space usage is proportional to num. of distinct alphabet symbols, thus O(m).

```
def find_boyer_moore(T, P):
  n, m = len(T), len(P)
  if m == 0: return 0
  last = { }
  for k in range(m):
    last[P[k]] = k                  // later occurrence overwrites
  // align end of pattern at index m-1 of text
  i = m-1, k = m-1
  while i < n:
    if T[i] == P[k]:                // a matching character
      if k == 0:
        return i                    // pattern begins at index i of text
      else
        i -= 1                      // examine prev character
        k -= 1                      // of P and T
    else
      j = last.get(T[i], -1)        // last(T[i]) is -1 if not found
      i += m - min(k, j+1)          // case analysis for jump step
      k = m-1                       // restart at end of pattern
  return -1
```

→ the correctness of the algorithm follows from the fact that each time the method makes a shift, it is guaranteed not to skip over any possible matches