Text Processing

→Pattern-Matching Algorithms: in the classic pattern-matching problem, we are given text string T of length n and a pattern string P of length m, and want to know if P is a substring of T.

→ we may want to find lowest index j within T at which P begins, such that T[j:j+m] = P, or perhaps find all indices of T where pattern P begins

→Brute Force:

→algorithmic design pattern which is a powerful technique when we have something we want to search for or want to optimize some function

→In applying brute force to pattern matching, we derive a simple algorithm where we simply test all possible placements of P relative to T

```
def find_brute(T, P):
    n, m = len(T), len(P)
    for i in range(n-m+1):          // try every potential starting index within T
        k = 0                       // at index into pattern P
        while k < m and T[i+k] == P[k]:  // kth char of P matches
            k += 1
        if k == m:                  // if end of pattern
            return i                // T[i:i+m] = P
    return -1                       // failed to find any matches
```

→Boyer-Moore Algorithm

→pattern-matching algorithm which can sometimes avoid comparisons between P and a sizable fraction of characters in T

→looks to improve brute force algorithm by implementing 2 heuristics:

→looking glass: when testing a possible placement of P against T, begin comparisons from the end of P and move backward to front of P

→character jump: during testing of a possible placement of P within T, a mismatch of text character T[i] = c w/ the corresponding pattern character P[k] is handled as follows:

→ if c is not contained anywhere in P, shift P completely past T[i]

→ otherwise, shift P until an occurrence of character c in P gets aligned with T[i]