Sequential Search Algorithm

Prof. Sasmita Kumari Nayak, CSE

Introduction

ALGORITHM SequentialSearch(A[0..n-1], K)
//Output: index of the first element in A, whose //value is equal to K or -1 if no such element is found

```
i <- 0
while i < n and A[i] ≠ K do
        i <- i+1
if i < n
        return i
else
        return -1
```

Input size: n

Basic op: <, ≠

$C_{worst}(n) = n$

Example

Let the elements of array are -

0	1	2	3	4	5	6	7	8
70	40	30	11	57	41	25	14	52

Let the element to be searched is **K = 41**
Now, start from the first element and compare **K** with each element of the array.

0	1	2	3	4	5	6	7	8
70	40	30	11	57	41	25	14	52

↑
K≠70

The value of **K,** i.e., **41,** is not matched with the first element of the array. So, move to the next element. And follow the same process until the respective element is found

Prof. Sasmita Kumari Nayak, CSE

Cont…

0	1	2	3	4	5	6	7	8
70	40	30	11	57	41	25	14	52

K≠40

0	1	2	3	4	5	6	7	8
70	40	30	11	57	41	25	14	52

K≠30

0	1	2	3	4	5	6	7	8
70	40	30	11	57	41	25	14	52

K≠11

0	1	2	3	4	5	6	7	8
70	40	30	11	57	41	25	14	52

K≠57

0	1	2	3	4	5	6	7	8
70	40	30	11	57	41	25	14	52

K=41

Now, the element to be searched is found. So algorithm will return the index of the element matched.

Time Complexity

- Best Case Complexity - In Linear search, best case occurs when the element we are finding is at the first position of the array.
- Average Case Complexity - The average case time complexity of linear search is O(n).
- Worst Case Complexity - In Linear search, the worst case occurs when the element we are looking is present at the end of the array. The worst-case in linear search could be when the target element is not present in the given array, and we have to traverse the entire array.

Case	Time Complexity
Best Case	O(1)
Average Case	O(n)
Worst Case	O(n)

Prof. Sasmita Kumari Nayak, CSE

Brute-Force String Matching Algorithm

Prof. Sasmita Kumari Nayak, CSE

Brute-Force String Matching Algorithm

Given a text array $T[1 \ldots n]$ and a pattern array $P[1 \ldots m]$ such that the elements of T and P are characters taken from alphabet Σ . e.g., $\Sigma = \{0, 1\}$ or $\Sigma = \{a, b, \ldots, z\}$.

The **String Matching Problem** is to find *all* the occurrence of P in T .

A pattern P occurs with **shift** s in T , if $P[1 \ldots m]$ = $T[s + 1 \ldots s + m]$. The String Matching Problem is to find all values of s. Obviously, we must have $0 \leq s \leq n - m$.

Cont…

Examples:

1. String and pattern matching
2. Computing $n!$
3. Multiplying two matrices
4. Searching for a key of a given value in a list

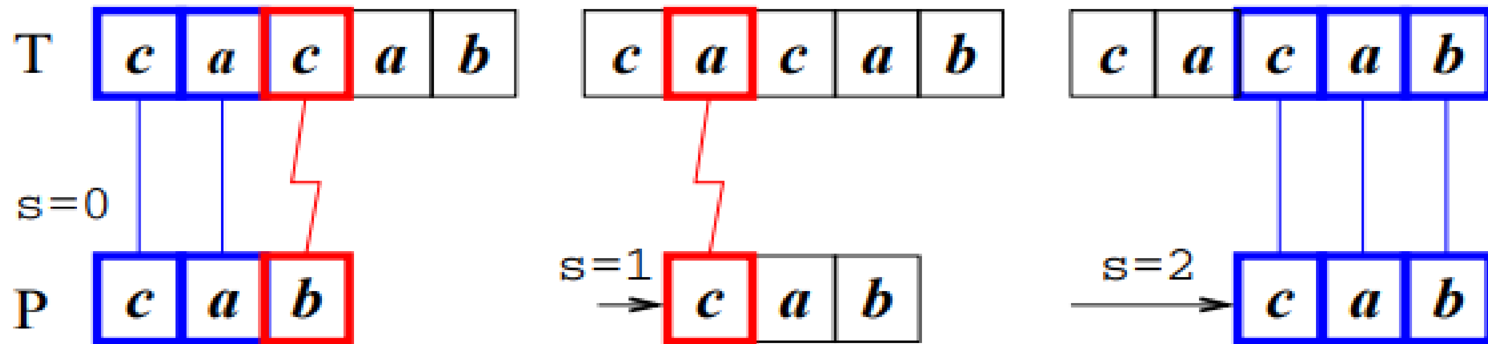
Brute-Force String Matching Algorithm

```
ALGORITHM BruteForceStringMatch(T [0..n − 1], P[0..m − 1])
//Implements brute-force string matching
//Input: An array T [0..n − 1] of n characters representing a text and
// an array P[0..m − 1] of m characters representing a pattern
//Output: The index of the first character in the text that starts a
// matching substring or −1 if the search is unsuccessful

for i ←0 to n − m do
        j ←0
        while j <m and P[j ]= T [i + j ] do
                j ←j + 1
        if j = m return i
return −1
```

Example

Initially, P is aligned with T at the first index position. P is then compared with T from **left-to-right**. If a mismatch occurs, ”slide” P to *right* by 1 position, and start the comparison again.



Efficiency

Brute-force pattern matching runs in time ***O***(***nm***).

Brute-force exact pattern match: worst case

Brute-force algorithm can be slow if text and pattern are repetitive

but this situation is rare in typical applications

Strengths

Strengths:

- It is wide applicable.
- It is not complex in nature unlike other algorithms.
- It yields reasonable algorithms for some important problems
  - searching; string matching; matrix multiplication
- It yields standard algorithms for simple computational tasks
  - sum/product of *n* numbers; finding max/min in a list

Weakness

Weaknesses:

- It rarely yields efficient algorithms
- Some brute force algorithms are unacceptably slow
  - e.g., the recursive algorithm for computing Fibonacci numbers
- It is not as constructive/creative as some other design techniques

Applications

- parsers.
- spam filters.
- digital libraries.
- screen scrapers.
- word processors.
- web search engines.
- natural language processing.
- computational molecular biology.
- feature detection in digitized images. . . .