BM267 - Introduction to Data Structures

12. Searching and BST

Ankara University

Computer Engineering Department

Bulent Tugrul

Searching

Sequential search

The algorithm simply compares successive elements of a given array or list with a given key until either a match is encountered (successful search) or the list is exhausted without finding the a match(unsuccessful search).

```
Algorithm Sequential Search( A[0…n-1], Key)
for( int i =0; i < n; i++)
       if( A[i] == Key )
              return 1;
return 0;
```

Searching

- An improvement can be incorporated in sequential search if the given list or array is sorted: searching in such list can be stopped as soon as an element greater than or equal to the search key is encountered.
- Analysis of sequential search
  - Average case: the sequential search compares ***`n/2`*** element of the list
  - Worst case: worst case occurs in two cases, either the search is unsuccessful or the key is found at the end of the list. Therefore in these cases sequential search has to make ***`n`*** comparisons

Searching

String matching

- String matching problem is searching a string of ***m*** characters called ***pattern*** in a given string of ***n*** characters called ***text***. (m <= n)
- To put it more precisely, we want to find i-the index of the leftmost character of the first matching substring in the text-such that $t_i = p_0, \ldots, t_{i+j} = p_j, \ldots, t_{i+m-1} = p_{m-1}$.

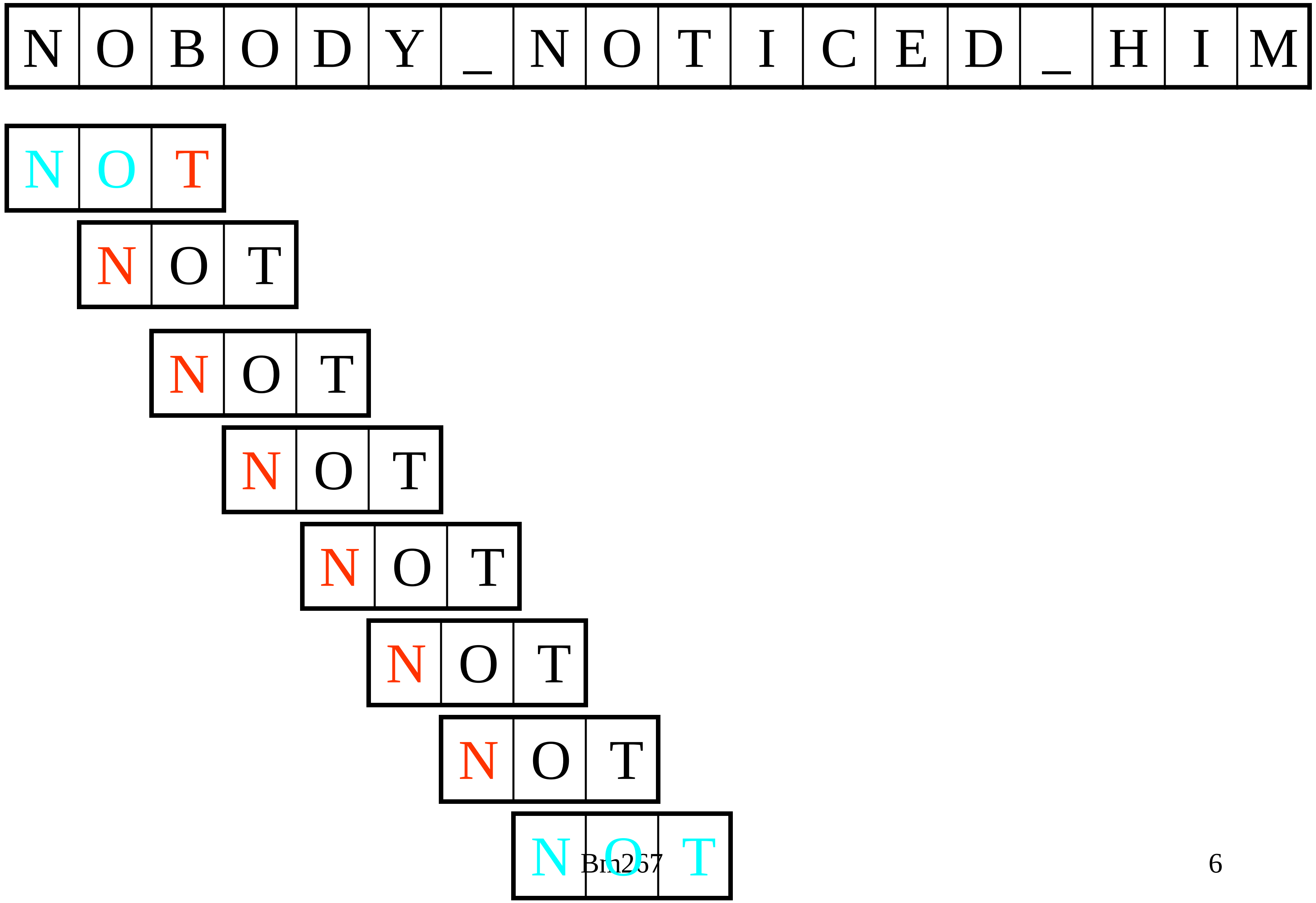
$$\begin{tikzcd}
t_0 & \ldots & t_i & \ldots & t_{i+j} & \ldots & t_{i+m-1} & \ldots & t_{n-1} & \text{text } T \\
 & & p_0 \arrow[u, leftrightarrow] & \ldots & p_j \arrow[u, leftrightarrow] & \ldots & p_{m-1} \arrow[u, leftrightarrow] & & & \text{pattern } P
\end{tikzcd}$$

Searching

- A brute-force algorithm for the string-matching problem is quite obvious: align the pattern against the first m characters of the text and start matching the corresponding pairs of characters from left to right until either all m pairs of the characters match or a mismatching pair is encountered.
- If there is a mismatch, the pattern is shifted one position to the right and character comparisons are resumed.

```
Algorithm BruteForceStringMatching( T[0…n-1], P[0…m-1])
for( i =0; i < n-m; i++)
      for( j =0; j < m && P[j] == T[i+j]; j++);
      if( j == m )
            return i;
return -1;
```

Searching



Searching

- Analysis of string matching: in the worst case the algorithm has to make Θ(n*m) comparisons.

Binary search

- Binary search is an efficient algorithm for searching in a sorted array. It works comparing a search key with the array’s middle element a[m]. If they match, the algorithm stops; Otherwise , the same operation is repeated recursively for first half of the array if key < a[m] and for the second half if key > a[m].

$$\underbrace{a[0] \ldots a[m-1]}_{\text{Search here if key< a[m]}} \;\; \overset{\overset{\text{key}}{\updownarrow}}{a[m]} \;\; \underbrace{a[m+1] \ldots a[n-1]}_{\text{search here if key>a[m]}}$$

-

Searching

Searching Key = 70

index	0	1	2	3	4	5	6	7	8	9	10	11	12
value	3	14	27	31	39	42	55	70	74	81	85	93	98
iteration 1	l						m						r
iteration 2								l		m			r
iteration 3								l,m	r				

Searching

```
Algorithm BinarySearch( A[0…n-1], Key)
l = 0; r = n-1;
while(l <= r ){
      m = (l+r)/2;
      if(K = A[m])
            return m;
      else if ( K < A[m])
            r = m-1;
      else
            l = m+1
}
return -1;
```

Searching

Analysis of binary search

The average case analysis of binary search depends on the key and values in the array.

The worst case occur if the search is unsuccessful. In this case algorithm has to make:

$$T(n) = T(n/2) + 1 \text{ comparisons}$$

$$= \log_2 n$$