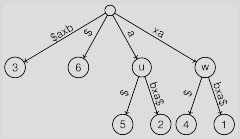
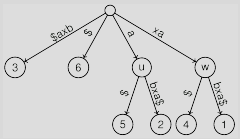


Datenstrukturen und Effiziente Algorithmen

Vorlesung *Datenstrukturen und Effiziente Algorithmen* im WS 18/19

Marc Hellmuth
Institut für Mathematik und Informatik
Universität Greifswald



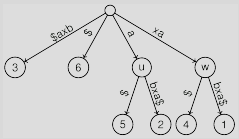
Exact String Matching Problem

Definition 1

Let P and T be strings, called *pattern* and *text*, respectively, and let T be longer than P. The exact matching problem is to find all occurrences, if any, of pattern P in text T.

Example 2

Let $P = au$ and $T = blaukraut$. Then P occurs in T at positions 3 and 7.
Occurrences may overlap as in $P = ata$ and $T = ctatatagc$, where P occurs at 3 and 5.



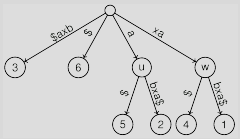
Exact String Matching Problem

Definition 1

Let P and T be strings, called *pattern* and *text*, respectively, and let T be longer than P. The exact matching problem is to find all occurrences, if any, of pattern P in text T.

Example 2

Let $P = au$ and $T = blaukraut$. Then P occurs in T at positions 3 and 7.
Occurrences may overlap as in $P = ata$ and $T = ctatatagc$, where P occurs at 3 and 5.



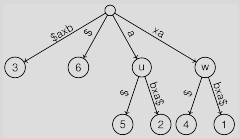
Applications of Exact String Matching

Applications requiring exact string matching

- word processing
- internet search
- Bioinformatics:
  Here, T a biological sequence database, e.g. of DNA sequences
- `fgrep` on UNIX
- search for plagiarism
- ...

Further relevance

- subtask for other problems (e.g. inexact matching)
- example for algorithm techniques, proof techniques



Applications of Exact String Matching

Applications requiring exact string matching

- word processing
- internet search
- Bioinformatics:
  Here, T a biological sequence database, e.g. of DNA sequences
- `fgrep` on UNIX
- search for plagiarism
- ...

Further relevance

- subtask for other problems (e.g. inexact matching)
- example for algorithm techniques, proof techniques

String Definitions

Definition 3

A string S is an ordered list of characters written contiguously from left to right.
For any string S, $S[i..j]$ is the substring of S that starts at position i and ends at position j of S.
If $i > j$, then $S[i..j]$ is the empty string.
$S[1..j]$ is called the prefix of S that ends at position j.
$|S|$ is the length of string S.
$S[i..|S|]$ is the suffix of S that begins at position i. $S[i]$ denotes the i-th character of S.
A proper (German: “echt”) prefix, suffix or substring of S is, respectively, a prefix, suffix, or substring that is not the entire string S, nor the empty string.
We say that two characters match if they are equal, otherwise we say they mismatch.

“Sequence”

Sometimes the word “subsequence” is used synonymously to “substring”, and not, as in mathematics, for a possibly non-contiguous ordered subset.

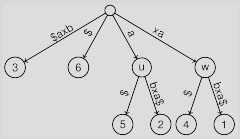
String Definitions

Definition 3

A string S is an ordered list of characters written contiguously from left to right.
For any string S, $S[i..j]$ is the substring of S that starts at position i and ends at position j of S.
If $i > j$, then $S[i..j]$ is the empty string.
$S[1..j]$ is called the prefix of S that ends at position j.
$|S|$ is the length of string S.
$S[i..|S|]$ is the suffix of S that begins at position i. $S[i]$ denotes the i-th character of S.
A proper (German: “echt”) prefix, suffix or substring of S is, respectively, a prefix, suffix, or substring that is not the entire string S, nor the empty string.
We say that two characters match if they are equal, otherwise we say they mismatch.

“Sequence”

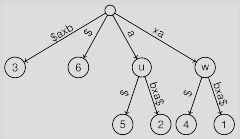
Sometimes the word “subsequence” is used synonymously to “substring”, and not, as in mathematics, for a possibly non-contiguous ordered subset.



The Naive Method

Naive exact string matching algorithm

```
1: for i = 1 to |T| do
2:     if OCCURSAT(P,T,i) then
3:         output i
4:
5: // check whether P occurs in T starting at pos. i
6: function OCCURSAT(P,T,i)
7:     j ← 1
8:     while j ≤ |P| and i + j − 1 ≤ |T| and P[j] = T[i + j − 1] do
9:         j ← j + 1
10:    if j > |P| then
11:        return true
12:    else
13:        return false
```



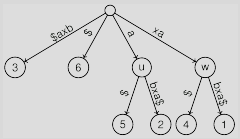
The Naive Method

Running time

Let $n = |P|$ and $m = |T|$ be the lengths of strings P and T, respectively.
The worst-case running time is $\Theta(nm)$.
This is required by above algorithm e.g. when P and T consist of repeats of a single character, e.g.
$P = aaaa$ and $T = aaaaaaaaaaaaaaaaaaaa$.

Want faster algorithm

We will next construct algorithms that run in $O(m + n)$.



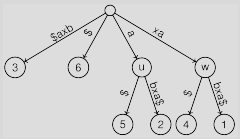
The Naive Method

Running time

Let $n = |P|$ and $m = |T|$ be the lengths of strings P and T, respectively.
The worst-case running time is $\Theta(nm)$.
This is required by above algorithm e.g. when P and T consist of repeats of a single character, e.g.
$P = aaaa$ and $T = aaaaaaaaaaaaaaaaaaaa$.

Want faster algorithm

We will next construct algorithms that run in $O(m + n)$.

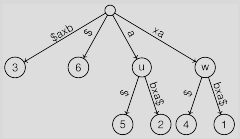


Some Intuition for a Speedup

Example 4 (naive method)

`1`: match, `0`: mismatch

```
T:   xabxyabxyabxz
P:   abxyabxz
```

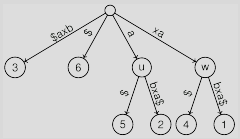


Some Intuition for a Speedup

Example 4 (naive method)

1: match, 0: mismatch

```
T:   xabxyabxyabxz
P:   abxyabxz
     0
```

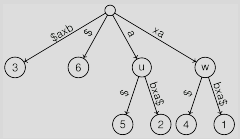


Some Intuition for a Speedup

Example 4 (naive method)

`1`: match, `0`: mismatch

```
T:   xabxyabxyabxz
P:   abxyabxz
     0
      abxyabxz
      1
```

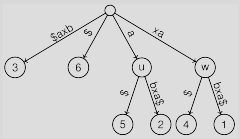


Some Intuition for a Speedup

Example 4 (naive method)

1: match, 0: mismatch

```
T:   xabxyabxyabxz
P:   abxyabxz
     0
      abxyabxz
      11
```

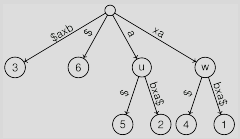


Some Intuition for a Speedup

Example 4 (naive method)

`1`: match, `0`: mismatch

```
T:   xabxyabxyabxz
P:   abxyabxz
     0
      abxyabxz
      111
```

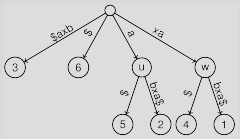


Some Intuition for a Speedup

Example 4 (naive method)

`1`: match, `0`: mismatch

```
T:   xabxyabxyabxz
P:   abxyabxz
     0
      abxyabxz
      1111
```

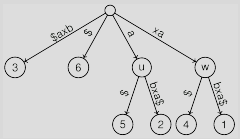


Some Intuition for a Speedup

Example 4 (naive method)

`1`: match, `0`: mismatch

```
T:   xabxyabxyabxz
P:   abxyabxz
     0
      abxyabxz
      11111
```

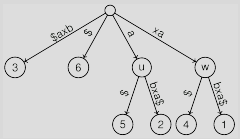


Some Intuition for a Speedup

Example 4 (naive method)

`1`: match, `0`: mismatch

```
T:   xabxyabxyabxz
P:   abxyabxz
     0
      abxyabxz
      111111
```

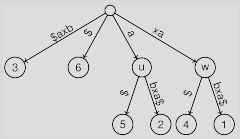


Some Intuition for a Speedup

Example 4 (naive method)

1: match, 0: mismatch

```
T:   xabxyabxyabxz
P:   abxyabxz
     0
      abxyabxz
      1111111
```

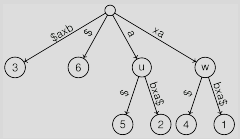


Some Intuition for a Speedup

Example 4 (naive method)

1: match, 0: mismatch

```
T:   xabxyabxyabxz
P:   abxyabxz
     0
      abxyabxz
      11111110
```

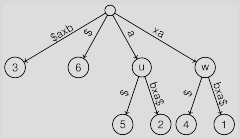


Some Intuition for a Speedup

Example 4 (naive method)

1: match, 0: mismatch

```
T:   xabxyabxyabxz
P:   abxyabxz
     0
      abxyabxz
      11111110
       abxyabxz
       0
```

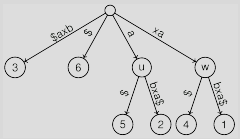


Some Intuition for a Speedup

Example 4 (naive method)

1: match, 0: mismatch

```
T:   xabxyabxyabxz
P:   abxyabxz
     0
      abxyabxz
      11111110
       abxyabxz
       0
        abxyabxz
        0
```

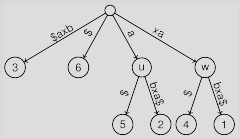


Some Intuition for a Speedup

Example 4 (naive method)

1: match, 0: mismatch

```
T:   xabxyabxyabxz
P:   abxyabxz
     0
      abxyabxz
      11111110
       abxyabxz
       0
        abxyabxz
        0
         abxyabxz
         0
```

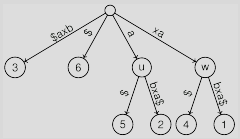


Some Intuition for a Speedup

Example 4 (naive method)

1: match, 0: mismatch

```
T:   xabxyabxyabxz
P:   abxyabxz
     0
      abxyabxz
      11111110
       abxyabxz
       0
        abxyabxz
        0
         abxyabxz
         0
          abxyabxz
          1
```

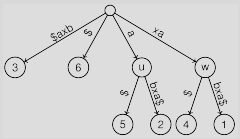


Some Intuition for a Speedup

Example 4 (naive method)

1: match, 0: mismatch

```
T:   xabxyabxyabxz
P:   abxyabxz
     0
      abxyabxz
      11111110
       abxyabxz
       0
        abxyabxz
        0
         abxyabxz
         0
          abxyabxz
          0
           abxyabxz
           11
```

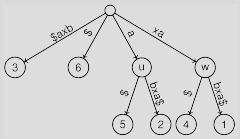


Some Intuition for a Speedup

Example 4 (naive method)

1: match, 0: mismatch

```
T:   xabxyabxyabxz
P:   abxyabxz
     0
      abxyabxz
      11111110
       abxyabxz
       0
        abxyabxz
        0
         abxyabxz
         0
          abxyabxz
          111
```

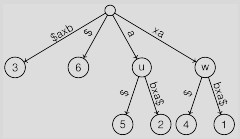


Some Intuition for a Speedup

Example 4 (naive method)

1: match, 0: mismatch

```
T:   xabxyabxyabxz
P:   abxyabxz
     0
      abxyabxz
      11111110
       abxyabxz
       0
        abxyabxz
        0
         abxyabxz
         0
          abxyabxz
          1111
```

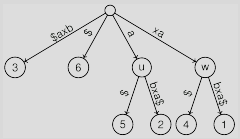


Some Intuition for a Speedup

Example 4 (naive method)

1: match, 0: mismatch

```
T:   xabxyabxyabxz
P:   abxyabxz
     0
      abxyabxz
      11111110
       abxyabxz
       0
        abxyabxz
        0
         abxyabxz
         0
          abxyabxz
          11111
```

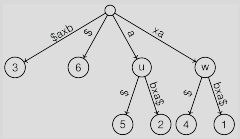


Some Intuition for a Speedup

Example 4 (naive method)

1: match, 0: mismatch

```
T:   xabxyabxyabxz
P:   abxyabxz
     0
      abxyabxz
      11111110
       abxyabxz
       0
        abxyabxz
        0
         abxyabxz
         0
          abxyabxz
          111111
```

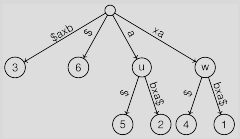


Some Intuition for a Speedup

Example 4 (naive method)

1: match, 0: mismatch

```
T:   xabxyabxyabxz
P:   abxyabxz
     0
      abxyabxz
      11111110
       abxyabxz
       0
        abxyabxz
        0
         abxyabxz
         0
          abxyabxz
          1111111
```

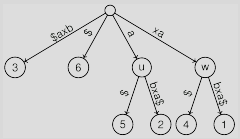


Some Intuition for a Speedup

Example 4 (naive method)

1: match, 0: mismatch

```
T:   xabxyabxyabxz
P:   abxyabxz
     0
      abxyabxz
      11111110
       abxyabxz
       0
        abxyabxz
        0
         abxyabxz
         0
          abxyabxz
          11111111
```



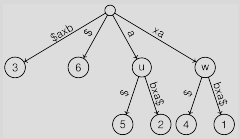
Some Intuition for a Speedup

Example 4 (naive method)

1: match, 0: mismatch

```
T:   xabxyabxyabxz
P:   abxyabxz
     0
      abxyabxz
      11111110
       abxyabxz
       0
        abxyabxz
        0
         abxyabxz
         0
          abxyabxz
          11111111
```

Need 20 comparisons.

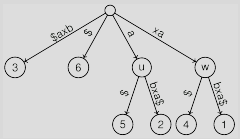


Some Intuition for a Speedup

Example 5 (skip shifts)

1: match, 0: mismatch

```
T:   xabxyabxyabxz
P:   abxyabxz
```

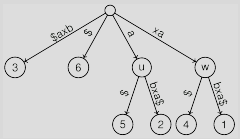


Some Intuition for a Speedup

Example 5 (skip shifts)

1: match, 0: mismatch

```
T:   xabxyabxyabxz
P:   abxyabxz
     0
```

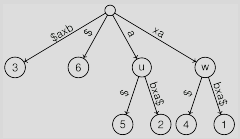


Some Intuition for a Speedup

Example 5 (skip shifts)

1: match, 0: mismatch

```
T:   xabxyabxyabxz
P:   abxyabxz
     0
      abxyabxz
      1
```

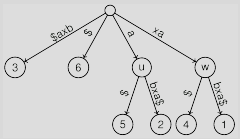


Some Intuition for a Speedup

Example 5 (skip shifts)

1: match, 0: mismatch

```
T:   xabxyabxyabxz
P:   abxyabxz
     0
      abxyabxz
      11
```

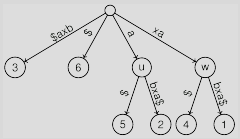


Some Intuition for a Speedup

Example 5 (skip shifts)

1: match, 0: mismatch

```
T:   xabxyabxyabxz
P:   abxyabxz
     0
      abxyabxz
      111
```

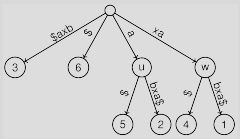


Some Intuition for a Speedup

Example 5 (skip shifts)

1: match, 0: mismatch

```
T:   xabxyabxyabxz
P:   abxyabxz
     0
      abxyabxz
      1111
```

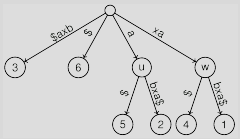


Some Intuition for a Speedup

Example 5 (skip shifts)

1: match, 0: mismatch

```
T:   xabxyabxyabxz
P:   abxyabxz
     0
      abxyabxz
      11111
```

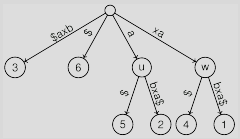


Some Intuition for a Speedup

Example 5 (skip shifts)

1: match, 0: mismatch

```
T:   xabxyabxyabxz
P:   abxyabxz
     0
      abxyabxz
      111111
```

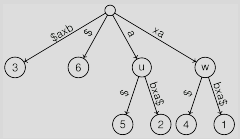


Some Intuition for a Speedup

Example 5 (skip shifts)

1: match, 0: mismatch

```
T:   xabxyabxyabxz
P:   abxyabxz
     0
      abxyabxz
      1111111
```

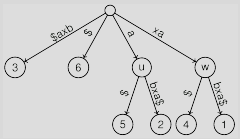


Some Intuition for a Speedup

Example 5 (skip shifts)

1: match, 0: mismatch

```
T:   xabxyabxyabxz
P:   abxyabxz
     0
      abxyabxz
      11111110
```



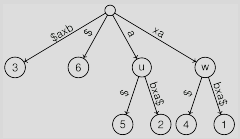
Some Intuition for a Speedup

Example 5 (skip shifts)

1: match, 0: mismatch

```
T:   xabxyabxyabxz
P:   abxyabxz
     0
      abxyabxz
      11111110
          abxyabxz
          1
```

Property of P: Whenever the first mismatch is at the 8th position in P, then we can skip 3 shifts, because the second a in P is at position 5.



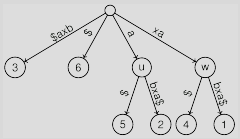
Some Intuition for a Speedup

Example 5 (skip shifts)

1: match, 0: mismatch

```
T:   xabxyabxyabxz
P:   abxyabxz
     0
      abxyabxz
      11111110
          abxyabxz
          11
```

Property of P: Whenever the first mismatch is at the 8th position in P, then we can skip 3 shifts, because the second a in P is at position 5.



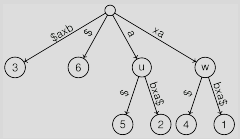
Some Intuition for a Speedup

Example 5 (skip shifts)

1: match, 0: mismatch

```
T:   xabxyabxyabxz
P:   abxyabxz
     0
      abxyabxz
      11111110
          abxyabxz
          111
```

Property of P: Whenever the first mismatch is at the 8th position in P, then we can skip 3 shifts, because the second a in P is at position 5.



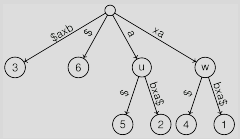
Some Intuition for a Speedup

Example 5 (skip shifts)

1: match, 0: mismatch

```
T:   xabxyabxyabxz
P:   abxyabxz
     0
      abxyabxz
      11111110
          abxyabxz
          1111
```

Property of P: Whenever the first mismatch is at the 8th position in P, then we can skip 3 shifts, because the second a in P is at position 5.



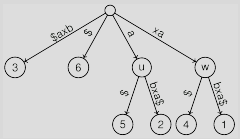
Some Intuition for a Speedup

Example 5 (skip shifts)

1: match, 0: mismatch

```
T:   xabxyabxyabxz
P:   abxyabxz
     0
      abxyabxz
      11111110
          abxyabxz
          11111
```

Property of P: Whenever the first mismatch is at the 8th position in P, then we can skip 3 shifts, because the second a in P is at position 5.



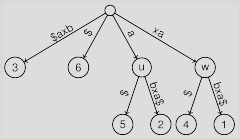
Some Intuition for a Speedup

Example 5 (skip shifts)

1: match, 0: mismatch

```
T:   xabxyabxyabxz
P:   abxyabxz
     0
      abxyabxz
      11111110
          abxyabxz
          111111
```

Property of P: Whenever the first mismatch is at the 8th position in P, then we can skip 3 shifts, because the second a in P is at position 5.



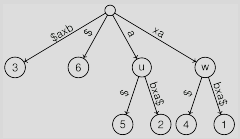
Some Intuition for a Speedup

Example 5 (skip shifts)

1: match, 0: mismatch

```
T:   xabxyabxyabxz
P:   abxyabxz
     0
      abxyabxz
      11111110
          abxyabxz
          1111111
```

Property of P: Whenever the first mismatch is at the 8th position in P, then we can skip 3 shifts, because the second a in P is at position 5.



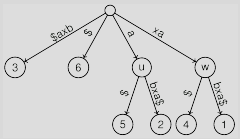
Some Intuition for a Speedup

Example 5 (skip shifts)

1: match, 0: mismatch

```
T:   xabxyabxyabxz
P:   abxyabxz
     0
      abxyabxz
      11111110
          abxyabxz
          11111111
```

Property of P: Whenever the first mismatch is at the 8th position in P, then we can skip 3 shifts, because the second a in P is at position 5.



Some Intuition for a Speedup

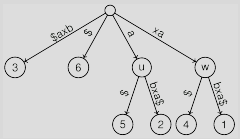
Example 5 (skip shifts)

1: match, 0: mismatch

```
T:   xabxyabxyabxz
P:   abxyabxz
     0
      abxyabxz
      11111110
          abxyabxz
          11111111
```

Need 17 comparisons.

Property of P: Whenever the first mismatch is at the 8th position in P, then we can skip 3 shifts, because the second a in P is at position 5.

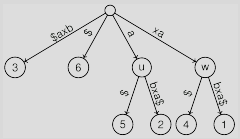


Some Intuition for a Speedup

Example 6 (skip shifts and comparisons)

1: match, 0: mismatch

```
T:   xabxyabxyabxz
P:   abxyabxz
```

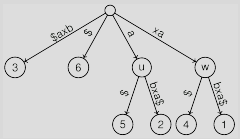


Some Intuition for a Speedup

Example 6 (skip shifts and comparisons)

1: match, 0: mismatch

```
T:   xabxyabxyabxz
P:   abxyabxz
     0
```

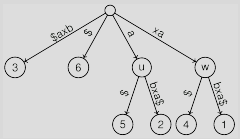


Some Intuition for a Speedup

Example 6 (skip shifts and comparisons)

1: match, 0: mismatch

```
T:   xabxyabxyabxz
P:   abxyabxz
     0
      abxyabxz
      1
```

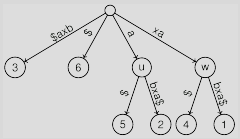


Some Intuition for a Speedup

Example 6 (skip shifts and comparisons)

1: match, 0: mismatch

```
T:   xabxyabxyabxz
P:   abxyabxz
     0
      abxyabxz
      11
```

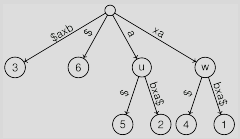


Some Intuition for a Speedup

Example 6 (skip shifts and comparisons)

1: match, 0: mismatch

```
T:   xabxyabxyabxz
P:   abxyabxz
     0
      abxyabxz
      111
```

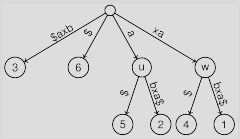


Some Intuition for a Speedup

Example 6 (skip shifts and comparisons)

1: match, 0: mismatch

```
T:   xabxyabxyabxz
P:   abxyabxz
     0
      abxyabxz
      1111
```

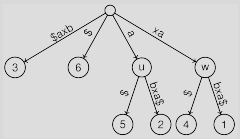


Some Intuition for a Speedup

Example 6 (skip shifts and comparisons)

1: match, 0: mismatch

```
T:   xabxyabxyabxz
P:   abxyabxz
     0
      abxyabxz
      11111
```

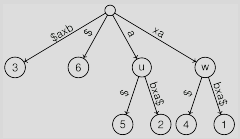


Some Intuition for a Speedup

Example 6 (skip shifts and comparisons)

1: match, 0: mismatch

```
T:   xabxyabxyabxz
P:   abxyabxz
     0
      abxyabxz
      111111
```

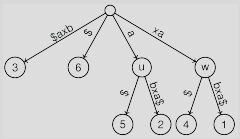


Some Intuition for a Speedup

Example 6 (skip shifts and comparisons)

1: match, 0: mismatch

```
T:   xabxyabxyabxz
P:   abxyabxz
     0
      abxyabxz
      1111111
```

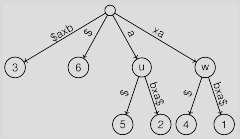


Some Intuition for a Speedup

Example 6 (skip shifts and comparisons)

1: match, 0: mismatch

```
T:   xabxyabxyabxz
P:   abxyabxz
     0
      abxyabxz
      11111110
```



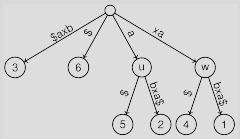
Some Intuition for a Speedup

Example 6 (skip shifts and comparisons)

1: match, 0: mismatch

```
T:   xabxyabxyabxz
P:   abxyabxz
     0
      abxyabxz
      11111110
          abxyabxz
             1
```

Property of P: Whenever the first mismatch is at the 8th position in P, then we can skip 3 shifts and start comparing at position 4 in P.



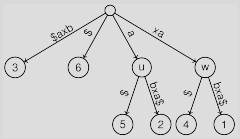
Some Intuition for a Speedup

Example 6 (skip shifts and comparisons)

1: match, 0: mismatch

```
T:   xabxyabxyabxz
P:   abxyabxz
     0
      abxyabxz
      11111110
          abxyabxz
             11
```

Property of P: Whenever the first mismatch is at the 8th position in P, then we can skip 3 shifts and start comparing at position 4 in P.



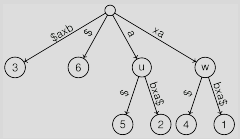
Some Intuition for a Speedup

Example 6 (skip shifts and comparisons)

1: match, 0: mismatch

```
T:   xabxyabxyabxz
P:   abxyabxz
     0
      abxyabxz
      11111110
          abxyabxz
             111
```

Property of P: Whenever the first mismatch is at the 8th position in P, then we can skip 3 shifts and start comparing at position 4 in P.



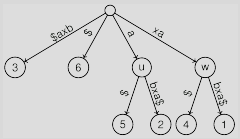
Some Intuition for a Speedup

Example 6 (skip shifts and comparisons)

1: match, 0: mismatch

```
T:   xabxyabxyabxz
P:   abxyabxz
     0
      abxyabxz
      11111110
          abxyabxz
             1111
```

Property of P: Whenever the first mismatch is at the 8th position in P, then we can skip 3 shifts and start comparing at position 4 in P.



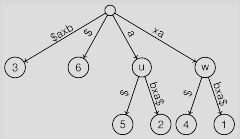
Some Intuition for a Speedup

Example 6 (skip shifts and comparisons)

1: match, 0: mismatch

```
T:   xabxyabxyabxz
P:   abxyabxz
     0
      abxyabxz
      11111110
          abxyabxz
             11111
```

Property of P: Whenever the first mismatch is at the 8th position in P, then we can skip 3 shifts and start comparing at position 4 in P.



Some Intuition for a Speedup

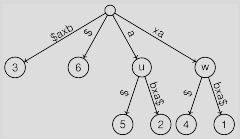
Example 6 (skip shifts and comparisons)

1: match, 0: mismatch

```
T:   xabxyabxyabxz
P:   abxyabxz
     0
      abxyabxz
      11111110
          abxyabxz
             11111
```

Need 14 comparisons.

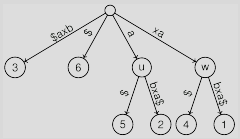
Property of P: Whenever the first mismatch is at the 8th position in P, then we can skip 3 shifts and start comparing at position 4 in P.



Preprocessing of Pattern

Preprocessing

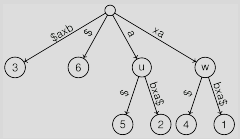
- before looking at text T examine internal structure of P
- preprocessing should be in $O(n)$
- several different algorithms use same fundamental preprocessing: Z algorithm
- will later use preprocessing results to search P in T in $O(m)$.



Preprocessing of Pattern

Preprocessing

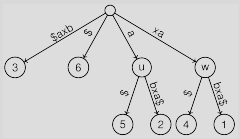
- before looking at text T examine internal structure of P
- preprocessing should be in $O(n)$
- several different algorithms use same fundamental preprocessing: Z algorithm
- will later use preprocessing results to search P in T in $O(m)$.



Preprocessing of Pattern

Preprocessing

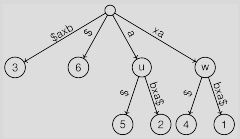
- before looking at text T examine internal structure of P
- preprocessing should be in $O(n)$
- several different algorithms use same fundamental preprocessing: Z algorithm
- will later use preprocessing results to search P in T in $O(m)$.



Preprocessing of Pattern

Preprocessing

- before looking at text T examine internal structure of P
- preprocessing should be in $O(n)$
- several different algorithms use same fundamental preprocessing: Z algorithm
- will later use preprocessing results to search P in T in $O(m)$.



Preprocessing

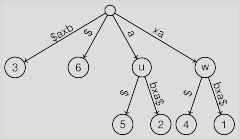
We will preprocess a string S. S will later often play the role of P.

Definition 7 (Z values)

Let $i > 1$ be a position in string S.
$Z_i = Z_i(S)$ is the length of the longest substring of S that starts at i and matches a prefix of S.

Example 8

$S = \texttt{eiderdeiderlei}$
$Z_7 = 5$
$Z_2 = 0$
$Z_{13} = 2$

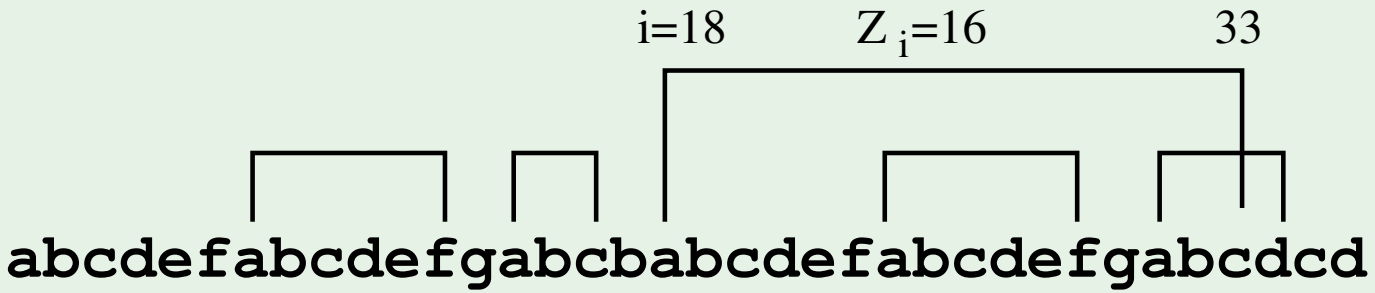


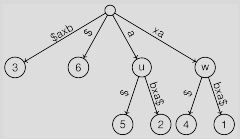
Z-Box

Definition 9 (Z-Box)

Let $i > 1$ such that $Z_i > 0$. Then the Z-box at i is the interval starting at i and ending at $i + Z_i - 1$.

Example 10



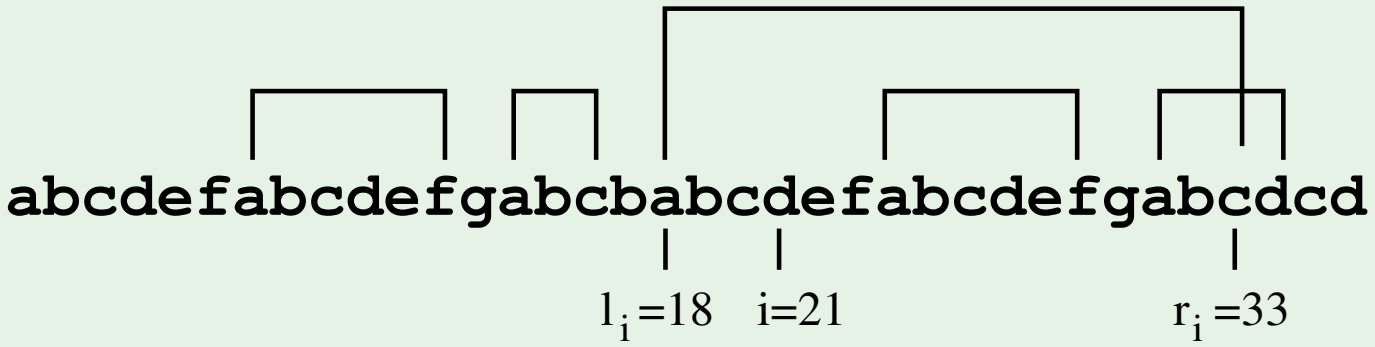


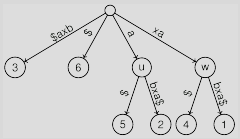
Z-Box

r_i, ℓ_i

Let $i > 1$. In the following $[\ell_i, r_i]$ will denote a Z-box that contains position i. If no Z-box contains position i, then $r_i < i$ (e.g. $r_i = 0$).

Example 11



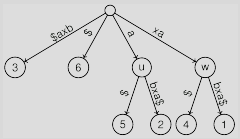


Z Algorithm

Z Algorithm

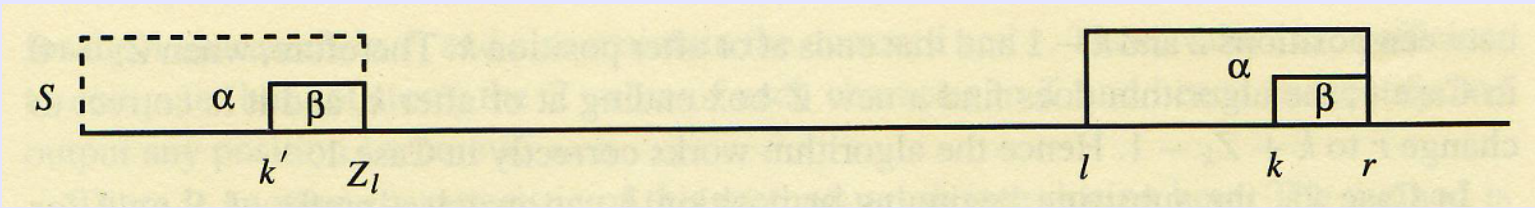
- preprocessing of S: compute all Z_i values
- direct approach takes time $O(|S|^2)$
- want to do it in $O(|S|)$
- will compute Z_k for increasing k
- Idea: When computing Z_k, r_k, ℓ_k we will reuse
  - $r = r_{k-1}, \ell = \ell_{k-1}$ and
  - previously computed values of $Z_2, \ldots, Z_{k-1}$.

  in order to save comparisons.

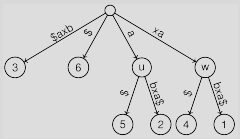


Z Algorithm: Idea

Reuse previously computed $Z_{k'}$

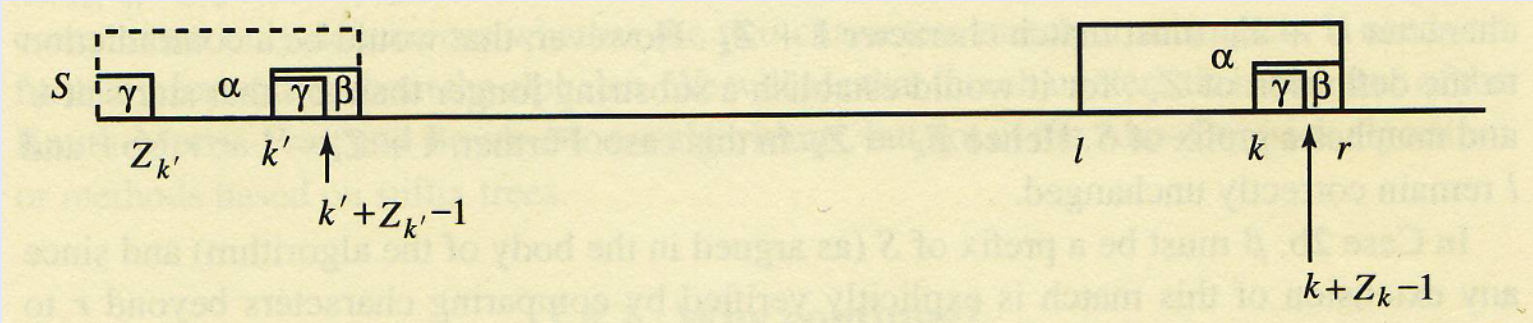


If $r \geq k$ then the string $\beta = S[k..r]$ occurs also at the end of the prefix of S of length Z_ℓ, starting at $k' = k - \ell + 1$ in S. We can use the value of $Z_{k'}$ to save comparison operations when determining Z_k.

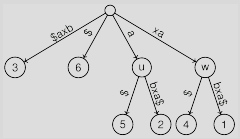


Z Algorithm: Idea

Case A

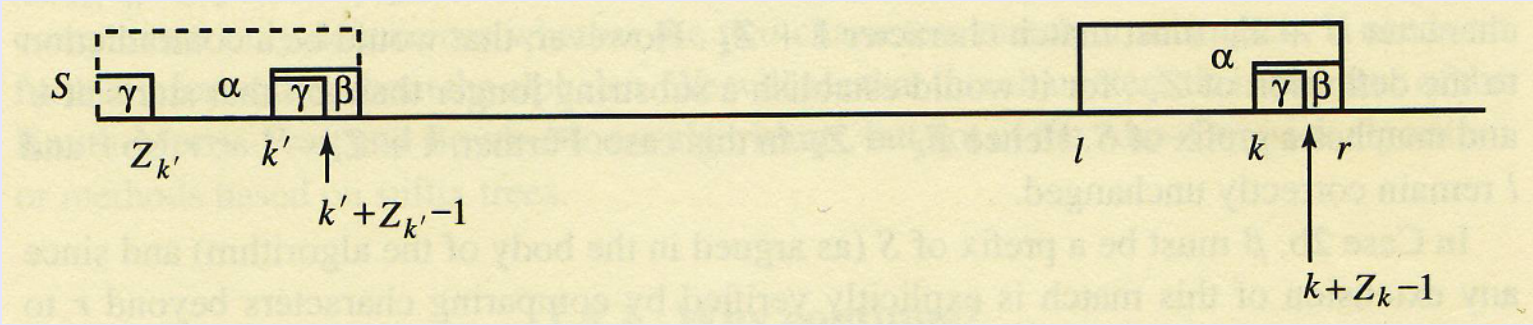


If $Z_{k'} < |\beta| = r - k + 1$, then the string $\gamma = S[1..Z_{k'}]$ starts also at k' and at k.
We get $Z_k = Z_{k'}$ without any further comparisons.



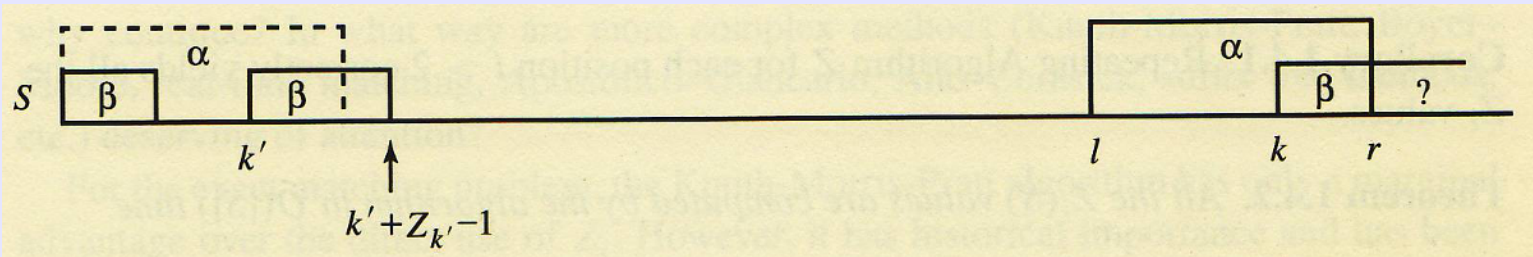
Z Algorithm: Idea

Case A



If $Z_{k'} < |\beta| = r - k + 1$, then the string $\gamma = S[1..Z_{k'}]$ starts also at k' and at k.
We get $Z_k = Z_{k'}$ without any further comparisons.

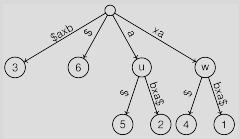
Case B



If $Z_{k'} \geq |\beta|$, then Z_k is at least $|\beta|$, too. We make further comparisons starting at $r + 1$, but don't have to compare the characters in $[k..r]$.

Z Algorithm

1: $r \leftarrow \ell \leftarrow 0$
2: **for** $k = 2$ to $|S|$ **do**
3: // Case 1:
4: **if** $r < k$ **then**
5: Compute Z_k explicitly by comparing the characters starting at k to the characters starting at 1 until a mismatch is found.
6: Set Z_k to the length of the match.
7: **if** $Z_k > 0$ **then**
8: Set $r \leftarrow k + Z_k - 1$, $\ell \leftarrow k$.
9: // Case 2:
10: **else**
11: $k' \leftarrow k - \ell + 1$
12: // Case 2a:
13: **if** $Z_{k'} < r - k + 1$ **then**
14: $Z_k \leftarrow Z_{k'}$
15: // Case 2b:
16: **else**
17: $\ell \leftarrow k$
18: compare the characters starting at $r + 1$ with the characters starting at $r - k + 2$ of S until a mismatch occurs
19: let $q > r$ be the position of the first mismatch or $q = |S| + 1$ if no mismatch occurs
20: $Z_k \leftarrow q - k$, $r \leftarrow q - 1$



Proof of Correctness

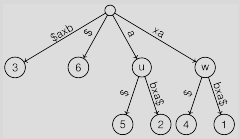
Theorem 12 (Correctness of Z Algorithm)

The Z algorithm computes all values $Z_2, \ldots, Z_{|S|}$ correctly.

Proof.

(chalk board)

□



Running Time of Z Algorithm

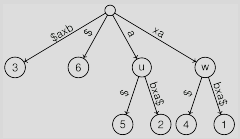
Theorem 13 (Running Time of Z Algorithm)

The Z algorithm computes all the Z_i values in $O(|S|)$ time.

Proof.

(chalk board)

□

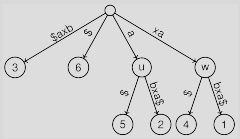


A Linear-Time Exact Matching Algorithm

Simple Linear-Time Exact Matching Algorithm

Require: character \$ not occuring in P or T

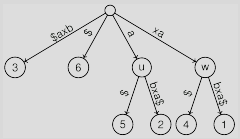
1: $n \leftarrow |P|$, $m \leftarrow |T|$
2: $S \leftarrow P\$T$
3: apply Z algorithm to S
4: **for** $i = n + 2$ to $m + 2$ **do**
5: **if** $Z_i = n$ **then**
6: output occurence of P starting at position $i - n - 1$ in T



A Linear-Time Exact Matching Algorithm

Theorem 14 (Correctness of Simple Exact Matching Algorithm)

Above algorithm reports all exact occurences of P in T.



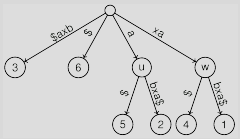
A Linear-Time Exact Matching Algorithm

Theorem 14 (Correctness of Simple Exact Matching Algorithm)

Above algorithm reports all exact occurences of P in T.

Proof.

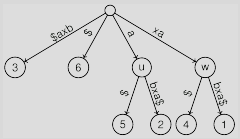
If $Z_i = n$ for any i in the range $n+2 \leq i \leq m+2$ then, by definition of Z_i, we have $S[1..n] = S[i..i+n-1]$, and therefore, by definition of S, $P = T[i-n-1..i-2]$. Therefore, P occurs as substring in T at position $i-n-1$.
Conversely, if P occurs starting at position j in T, then P occurs starting at position $j+n+1$ in S and $i := j+n+1$ is in the range $i = n+2$ to $m+2$. Therefore, $Z_i \geq n$. As \$ does not occur in T, we have $Z_i = n$ and position j is reported as occurence of P in T. □



A Linear-Time Exact Matching Algorithm

Theorem 15

Above algorithm runs in time $O(m)$ with $|T| = m$.



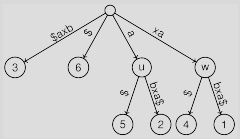
A Linear-Time Exact Matching Algorithm

Theorem 15

Above algorithm runs in time $O(m)$ with $|T| = m$.

Proof.

As shown previously, the Z algorithm takes time $O(|S|) = O(m+n) = O(m)$, since $n \leq m$. □



A Linear-Time Exact Matching Algorithm

Theorem 15

Above algorithm runs in time $O(m)$ with $|T| = m$.

Proof.

As shown previously, the Z algorithm takes time $O(|S|) = O(m+n) = O(m)$, since $n \leq m$. □

Space

Above algorithm can be implemented requiring $O(n)$ space in addition to storing P and T: We simply store the Z_i values only for $i \leq n$. Since \$ is not in T, we always have $Z_i \leq n$ and therefore k' from the Z algorithm is always less than or equal to n. We never need to recurse to a Z_i value for $i > n$.