

COMP261 Lecture 17

Lindsay Groves

String Search 1 of 2



Victoria
UNIVERSITY OF WELLINGTON

*Te Whare Wānanga
o te Ūpoko o te Ika a Māui*



CAPITAL CITY UNIVERSITY

String search

- Given a string **S** and a text **T**,
look for an occurrence of **S** as a substring of **T**.
- Ummm ... which one? What do I do when I find it?
- If found, return index of first character of **S** in **T**;
otherwise return -1 (or some other index outside of **T**).
- What would you expect the cost to be?

String search

- Find the string "vtewfvtxqwfcsrdzcaj" in the text:

qwerxcvvtewfzxcfasfedrsadfsdacfasdrtvtewqwertcsvte
wfvtxqwfcsrdzfeceeaeszxcvtsafsersdxzcvtedfaevsadv
tewfvtxqwfcsvzxgvtasfvtcasrfvtewqtrwtravtewfxtrac
wrtrdtgfdvxvvsbdgfstqtretydfxvzccadawqeewtertgvb
vczfafsvtewfvtxqwfcszsgfsdfdxvzvzvtvsdgfsgtftw6fqwt
qwrcfxtvtewfwtqwfzvwqgtfvtqfwcxetwfazreqresdqxrdqc
fwqdxvgfewcvtwefxvtrfczrqesxqecaqrzfvtqwxvbwyegcbe
bcwtfexvtfwxcrqxeqdcqzrwdfvtwxefvctyvtewfwefxqtfxc
qcdzrqxesrzqxrqcwqtfxtewfcwverygcviewytxvqewtcxzdcd
qwfxtewfvtxqwfcsrdzcajwfcsxtqwefdvewqfvtqfvtqfvtqfvtq

String search - some variations

- Just check whether there, returning Boolean.
- Find first/last/any occurrence of **S** in **T**.
- Find all occurrences of **S** in **T**.
 - What if occurrences overlap?
- Find occurrence(s) as a whole word/anywhere?
- Find occurrences within lines/allow occurrences to extend across line breaks?
- Assume random data? English text? Other data?

String search

- In Java, we can do this using:
 - `T.indexOf(S);`
 - `T.lastIndexOf(S);`
 - `T.contains(S);`
- But we'd like to know if these are good choices – or if we can do better.
- Let's start with a simple algorithm, and see how we can improve upon it.

Brute force approach

- string: $S = \text{ananaba}$
- text: $T = \text{bannabanabananaban}$
- Look for S , starting at $T[0]$:
ananaba
bannabanabananaban
- Look for S , starting at $T[1]$:
ananaba
bannabanabananaban
- Look for S , starting at $T[2]$:
ananaba
bannabanabananaban
- Etc. till found, or none left.

Brute force algorithm

- Basic idea:
Look for S in T,
starting at positions T[0], T[1],
- What is last position in T we need to consider?
- **for** $k \leftarrow 0$ to T.length() - S.length()
 if T.substring(k, S.length()).equals(S) **then return** k
 return -1
- What is cost?

Brute force algorithm

- How can we improve this?
- First, some very simple “improvements”:
 - Don’t call `S.length()` and `T.length()` in the loop.
Avoid cost of method call (compiler may inline it).
 - Don’t call `substring` in the loop.
Don’t need to copy the substring to a new string to compare with `S`.

Brute force algorithm

- Expanding substring and equals,
and assuming $S.length() == m$ and $T.length() == n$.
- **for** $k \leftarrow 0$ to $n-m$
 $found \leftarrow true$
 for $i \leftarrow 0$ to $m-1$
 if $S[i] \neq T[k+i]$ **then** $found \leftarrow false$, **break**
 if $found$ **then return** k
 return -1
- What is cost?

Brute force algorithm: cost

- $S = s_0 \ s_1 \ s_2 \ s_3 \ s_4 \ \dots$
 $T = t_0 \ t_1 \ t_2 \ t_3 \ t_4 \ t_5 \ t_6 \ t_7 \ \dots$
- What is best/worst/expected cost?
- What if text is random? English?
- What case gives best/worst cost (for any m and n)?
 - How many positions in T need to be considered?
 - How many characters need to be considered at each position?

Brute force algorithm: best cost


- $S = s_0 s_1 s_2 s_3 s_4$
 $T = t_0 t_1 t_2 t_3 t_4 t_5 t_6 t_7 \dots$
- Suppose s_0 doesn't occur in T .
 - S_0 will be compared to t_0, t_1, \dots
 - So cost will be?
- Suppose S is a prefix of T .
 - Will compare s_0 with t_0 , s_1 with t_1 , ...
 - So cost is?


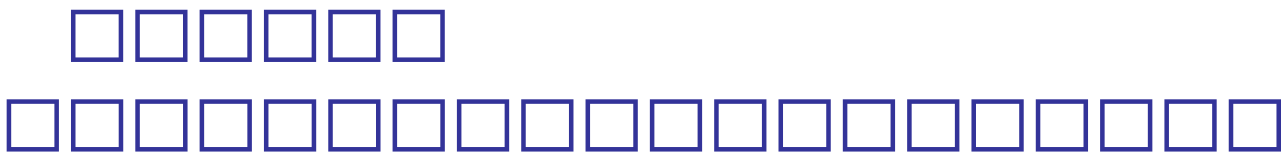
Brute force algorithm: worst cost

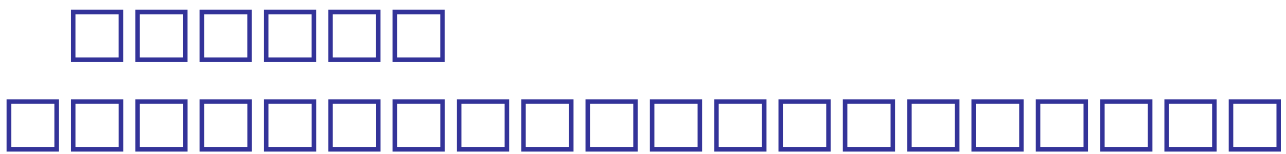
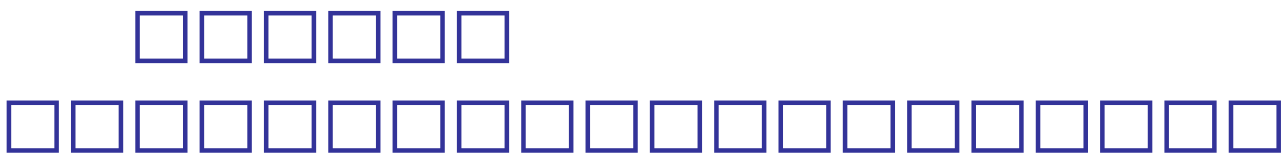
- $S = s_0 s_1 s_2 s_3 s_4$
 $T = t_0 t_1 t_2 t_3 t_4 t_5 t_6 t_7 \dots$
- What case will force the algorithm to do the most comparisons?
- Hint 1: Want S not in T , so it tries the maximum number of positions.
- Hint 2: At each position, want algorithm to do the most possible comparisons before failing.
 \Rightarrow Fail on the last character in S !
- What inputs would do this?

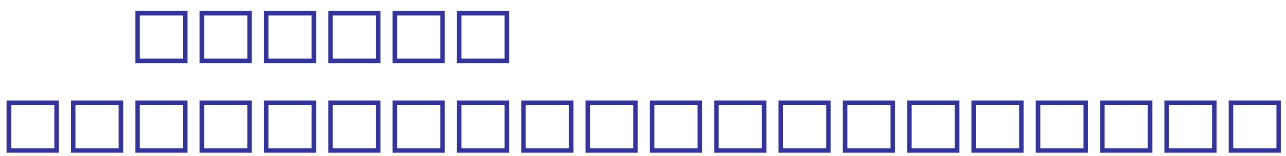
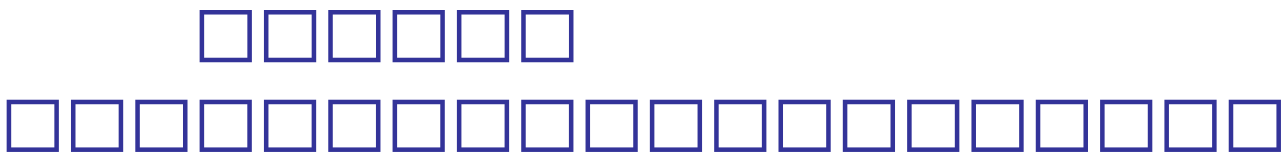
Brute force algorithm: worst cost

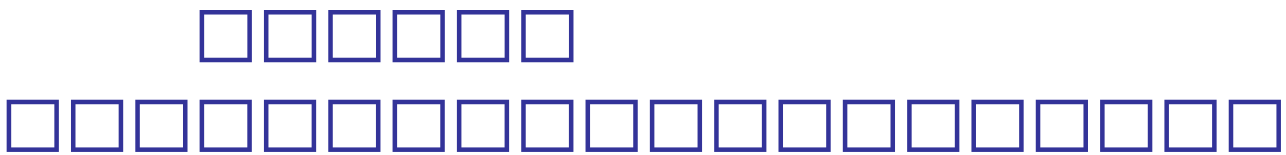
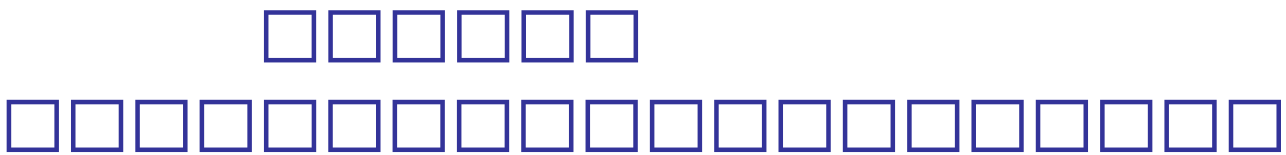
- What inputs would do this?

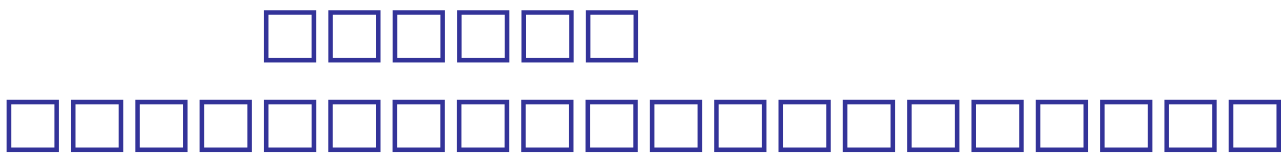
- $k = 0$: 


- $k = 1$: 


- $k = 2$: 


- $k = 3$: 


- $k = 4$: 



Brute force algorithm: worst cost

- What inputs would do this?
- $s_0=t_0, s_1=t_1, \dots, s_{(m-1)} \neq t_{(m-1)}$
- $s_0=t_1, s_1=t_2, \dots, s_{(m-1)} \neq t_{(m)}$
- ...
- So, $s_0=s_1=\dots=s_{(m-2)}=t_0=t_1=\dots=t_{(n-1)}$
 $s_{(m-1)} \neq s_0$
- E.g.:
 - $S = \text{aaaaab}$
 - $T = \text{aaaaaaaaaaaaaaaaaaaaa}$
- What is cost?
- Would this ever happen with English text?

String search

- Can we do better? Can we avoid rechecking?
- `abcm`ndsjhhsjgrjgslagfiigirnvkfir
`abc``e`fg
 - After checking `abc`, where should we check next?
- `anan`dfjoijtoiinkjjkjgfgkjkkhgklhg
`anan``a`ba
 - After checking `anan`, where should we check next?
- Key idea: Use characters in partial match to determine where to start next match attempt.

String search: Example

- T = abc abcdab abcdababdabde
S = abcdabd
- T = abc abcdab abcdababdabde
S = abcdabd
- T = abc abcdab abcdababdabde
S = abcdabd

String search: Example

- T = abc abcd**ab** abcdab**cd**abde
 S = **ab****c**dabd
- T = abc abcdab abcdab**cd**abde
 S = **a**bcdabd
- T = abc abcdab **abcdab****c**dabde
 S = **abcdab****d**
- T = abc abcdab abcd**abcdab**de
 S = **abcdab**d

Knuth-Morris-Pratt (KMP) algorithm

- Fast string search – never rechecks characters.
- After a mismatch, advance to the earliest place where search string could possibly match.
- How do we determine how far to advance?
- Use a table based on the search string.
- Let $M[0..m-1]$ be a table showing how far to back up the search if a prefix of S has been matched.

Knuth Morris Pratt

input: string $S[0 \dots m-1]$, text $T[0 \dots n-1]$, partial match table $M[0 \dots m-1]$

output: the position in T at which S is found, or -1 if not present

variables: $k \leftarrow 0$ *start of current match in T*

$i \leftarrow 0$ *position of current character in S*

while $k + i < n$

if $S[k] = T[k + i]$ **then** *// match*

$k \leftarrow k + 1$

if $i = m$ **then return** k *// found S*

else if $M[i] = -1$ **then** *// mismatch, no self overlap*

$i \leftarrow 0$, $k \leftarrow k + i + 1$,

else *// mismatch, with self overlap*

$k \leftarrow k + i - M[i]$ *// match position jumps forward*

$i \leftarrow M[i]$

return -1 *// failed to find S*

KMP how far to move along?

- string: ananaba
- text: ...ananx???...
- If mismatch at string position s (and text position $t+s$)
 - find longest **suffix (substring ending at $s-1$)** that matches a **prefix** of string
 - move k forward by $(i - \text{length of substring})$
 - keep matching from $i \leftarrow \text{length of substring}$
- special case:
 - if $i = 0$, then move k to $k + 1$ and match from $i \leftarrow 0$

Knuth Morris Pratt

- Summary
 - searches forward,
 - never matches a text character twice (and never skips a text character)
 - jumps string forward based on self match within the string:
 - prefix of string matching a later substring.
 - doesn't use the character in the text to determine the jump.
- Cost?