Session 14 String Algorithms

Naive search bad

Boyer-Moore Fast single string search

Finite State Machine Optimal multiple target search

LCS Longest common subsequence

Editdistance measuring how different two strings are

Naive algorithm

target string = “hello” len(target string) = k, length(search string) = n

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| T | h | i | s |  | i | s |  | m | y |  | s | t | r | i | n | g | . | . | . |
| h | h | e |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  | h | h | h | h | h | h | h | h | h | h | h | h | h | h | h | h | h | h |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| h | e | l | l | h | e | l | l | h | e | l | l | h | e | l | l | h | e | l | l |
| h | e | l | l | o |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  | h | h | h | h | e | l | l | o |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  | h | h | h | h | e |  |  |  |  |  |  |  |  |  |  |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| h | e | h | e | h | e | h | e | h | e |  |  |  |  |  |  |  |  |  |  |
| h | e | l | l | o |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  | h | h | e | l |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  | h | h | e | l |  |  |  |  |  |  |  |  |  |  |  |  |  |

“xxxxxx”

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| x | x | x | x | x | a | x | o | x | o | x | a |  |  |  |  |  |  |  |  |
| x | x | x | x | x | x | x | o | x | o | x | o |  |  |  |  |  |  |  |  |
|  | x | x | x | x |  | x | o | x | x | o | x |  |  |  |  |  |  |  |  |
|  |  |  | h | h | e | l |  |  |  |  |  |  |  |  |  |  |  |  |  |

O(kn) Ω(n)

naive(search, target)

outer:

for (i = 0; i < search.length; ++i)

if (search[i] = target[0]) {  
 for (int j = 1; j < target.length; ++j)

if(search[i+j] != target[j])  
 continue outer;  
 }

target string = “hello“

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| h | e | l | l |  | h | e | l | l |  | h | e | l | l |  | n | g | . | . | . |
| h | e | l | l | o |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  | h | h | h | h | h | e | l | l | o |  |  |  |  |  |  |  |  |  |  |

target string = “xoxoxoxoxoxo”

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| x | o | x | o | x | o | x | o | x | o | x | x | o | x | o | x | o | x | o | x |
| x | o | x | o | x | o | x | o | x | o | x | o |  |  |  |  |  |  |  |  |
|  | x | x | o | x | o | x | o |  |  |  |  |  |  |  |  |  |  |  |  |

O(kn)

# Boyer-Moore

target=”avalanche”

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| T | h | i | s |  | i | s |  | q | u | e | s | t | i | o | n | a | b | l | e |
|  |  |  |  |  |  |  |  | e |  |  |  |  |  |  |  |  | e |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

target=”avalanche”

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| T | h | i | s |  | i | s |  | a |  | t | e | s | t | . |  | t | e | s | t |
|  |  |  |  |  |  |  |  | e |  |  |  | e |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

target=”avalanche”

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| T | h | i | s |  | i | s |  | e | v | e | r | y | o | n | e | . | . | . |  |
|  |  |  |  |  |  |  | h | e |  |  |  | e |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

create a table of 256 offset avalanche

initialize every element of the table to k

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w | x | y | z |
| 4 | 9 | 2 | 9 | 0 | 9 | 9 | 1 | 9 | 9 | 9 | 5 | 9 | 3 | 9 | 9 | 9 | 9 | 9 | 9 | 9 | 7 | 9 | 9 | 9 | 9 |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| a | m | a | z | i | n | g |  | s | t | r | i | n | g |  | s | e | a | r | c |
|  |  |  |  |  |  |  |  | e |  |  |  |  |  |  |  |  | e |  |  |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| a | a | v | a | l | a | n | c | h | e | r | i | n | g |  | s | e | a | r | c |
|  |  |  |  |  |  |  |  | e | e |  |  |  |  |  |  |  | e |  |  |

O(n/k) average case

target = “aaaaaa”

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| b | a | a | a | a | a | b | a | a | a | a | a | b |  |  |  |  |  |  |  |
| a |  |  |  |  | a | a |  |  |  |  |  | a |  |  |  |  | e |  |  |

target “ababab”

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| a | b | a | b | a | a | b | a | b | a | a | a | b |  |  |  |  |  |  |  |
|  |  |  |  |  | b | b |  | e | e |  |  |  |  |  |  |  | e |  |  |

# Boyer-Moore Practice

search = “do you know your alphabet?”

target=”alphabet” length = k = 8

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w | x | y | z |
| 3 | 2 | 8 | 8 | 1 | 8 | 8 | 4 | 8 | 8 | 8 | 6 | 8 | 8 | 8 | 5 | 8 | 8 | 8 | 0 | 8 | 8 | 8 | 8 | 8 | 8 |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| d | o |  | y | o | u |  | k | n | o | w |  | y | o | u | r |  | a | l | p |
|  |  |  |  |  |  |  | \* |  |  |  |  |  |  |  | \* |  |  |  |  |

BoyerMoore(search, target)

for i =0 to 255

table[i] = k

for i = 0 to k

table[target[i]] = k - i - 1

i = k-1

while (i < n)

jump ← table[search[i]]

if (jump == 0) { // possible match

}

i ← i + jump

end

# 

# 

# Finite State Machine

Boyer-Moore is the fastest for single string. But what if we look for “cat” OR “dog”?

m=number of words

k = length of targets

n = length of source

O(n/k \* m)

Regular expressions → finite state machines are optimal.

First let’s review regex

/abc/ → a followed by b followed by c

/[ABRT]/ any single letter A, B, R or T

/[A-Z]/ any single letter A through Z (not lowercase)

a\* zero or more occurrences of a

a+ one or more occurrences of a

a? zero or one occurrences of a

/[A-Z][a-z]\*/ → any single capital letter followed by zero or more lowercase

/[A-Z][^A-Z]/ → any capital letter followed by anything that is NOT a capital letter

abc|def either abc or def

(grouping) parentheses group

a{number} repeat factor

a{min,max} from min to max copies of a

\d shorthand for [0-9]

\w shorthand for [a-zA-Z0-9]

\s shorthand for [ \t\n] any space character

ab\*cb+a abcba acba acbba

/cat|(dog)+/

remembering

/public\s+class\s+([A-Za-z][A-Za-z0-9\_]\*)/

public class Test {  
  
}

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| t | h | i | s |  | i | s | a |  | c | a | t |  | d | o | g |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

searching for m strings: Boyer-Moore would be O(n/k \* m)

/cat|dog/

cccctaxcdoqcatdog

c

[^cd]

[^cad]

a

d

d

o

[^ocd]

c

c

t

g

c

d

c

d

[^gcd]

O(n) (length of the search string) regardless of how complicated the state machine is

pat = /cat|dog/;

if (str.contains(pat)) // O(n) where n is length of the string

google pre-indexes search text on many keys: including mis-spelling

spelling spell spel

1012 TB PB = 1015 ExaByte = 1018

Map-Reduce

Pre-indexed on all keys

parallel string search

apple

bear

google

cat

dog

# LCS Longest Common Subsequence

// xStudent A

public class A {   
 int x = 24\*60\*60;

print(x);   
}

// Student A

public class A {   
 int x = 24\*60\*60;

print(x);   
}

int lcs\_length(const char \* A, const char \* B)  
 {  
 if (\*A == '\0' || \*B == '\0') return 0;  
 else if (\*A == \*B) return 1 + lcs\_length(A+1, B+1);  
 else return max(lcs\_length(A+1,B), lcs\_length(A,B+1));  
 }

Merkle, Blockchain

diff a.txt b.txt

// xStudent A → hashX

public class A { → hash  
 int x = 24\*60\*60; → hash

print(x); → hash  
}

// Student A → hashY

public class A { → hash  
 int x = 24\*60\*60; → hash

print(x); → hash  
}

diff does LCS using memoization on the hashed lines

NOT on the test, just for you!

A

B  
C  
D

A  
B  
X  
C  
D

LCS with memoization: https://www.ics.uci.edu/~eppstein/161/960229.html

TO make diff faster:

hash each line and compare hashes!

this is my line

this is my EDITED line

line1 → hash

line2 → hash

line3

line1 → hash

line2

line3

line4

int main() {  
 int a = 5;

cout << a + 1;

}

int main() {  
 int b = 5;

cout << b + 1;

}