String Algorithms and Data Structures

Introduction and Pattern Matching

CS 199-225

Brad Solomon

September 9, 2024

A B C D
B B B

UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN

Department of Computer Science

Who am I?

Brad Solomon

Teaching Assistant Professor, Computer Science

2233 Siebel Center for Computer Science

Email: bradsol@illinois.edu

Office Hours:

Thursdays, 11:00 AM - 12:00 PM

… or by appointment

https://courses.engr.illinois.edu/cs225/info/office-hours/

CS 225 Honors Staff Introductions

Who are you?

Take a moment to introduce yourself to your neighbor!

Both left & right!

(Your name, a hobby you enjoy, and one thing you hope to get out of this class)

Piazza Sign up Link: https://piazza.com/illinois/fall2024/cs199225

What is this class about?

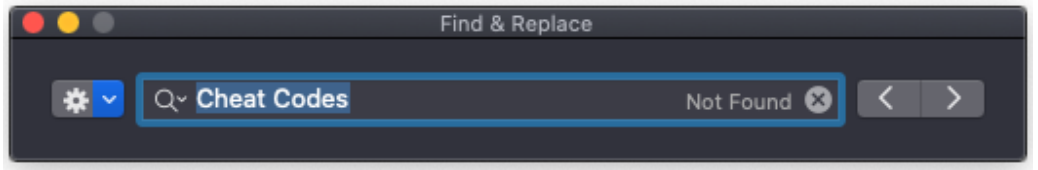
Foundational

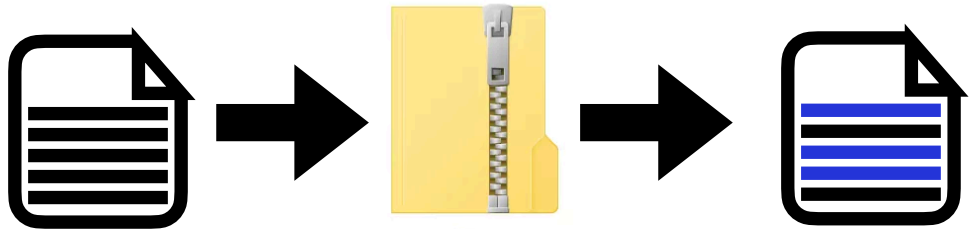
String Algorithms and Data Structures

Exact string matching

Compressed self-indexes

Inexact pattern matching





```
Query: 161 atatcaccacgtcaaaggtgactccaactcca---ccactccattttgtt
           ||||||||||||||||||||||||||||||  |     | |   || ||||
Sbjct: 481 atatcaccacgtcaaaggtgactccaact-tattgatagtgttttatgtt
```

What is this class about?

Foundational

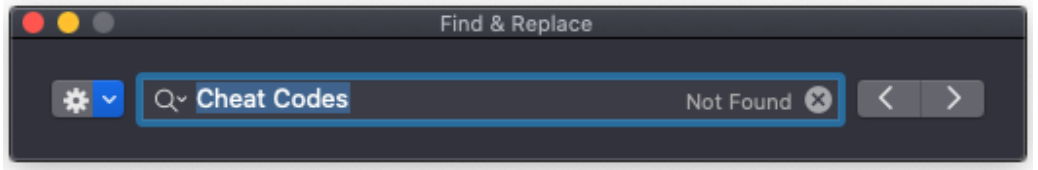
String Algorithms and Data Structures

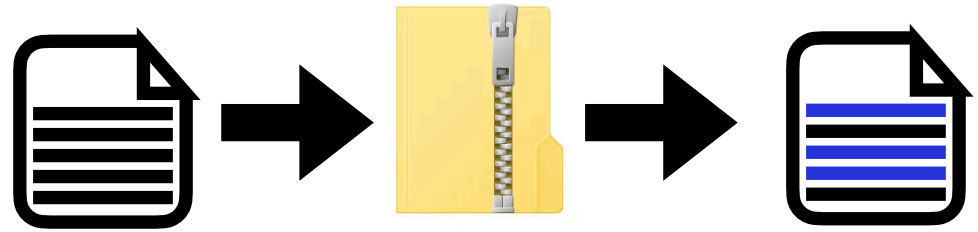
plus cool stuff chosen by you!

Exact string matching

Compressed self-indexes

Inexact pattern matching





```
Query: 161 atatcaccacgtcaaaggtgactccaactcca---ccactccattttgtt
           ||||||||||||||||||||||||||||||  |     | |   || ||||
Sbjct: 481 atatcaccacgtcaaaggtgactccaact-tattgatagtgttttatgtt
```

What will you get out of this class?

Understand fundamental string algorithms

Experience applying data structures, algorithms, and algorithm design principles to real world problems

Justify implementation choices based on theoretical or practical considerations

Build a foundation for future data science projects

Course Webpage

https://courses.grainger.illinois.edu/cs225/fa2024/pages/honors.html

All course information and links can be found here!

Mediaspace recordings

Piazza

Syllabus

Syllabus

Please read — many important topics:

- Course Goals & Topics
- Course Expectations
- Grading
- Commitments to Diversity, Equity, Inclusion
- Commitments to Mental Health
- Ethics and Academic Integrity Policies

Course Expectations

Weekly assignments (11 total):

- Small assignments (~ 1-3 hours / week)
- Must pass at least 10 of them (80% is passing)
- Must submit your own work
- One week extensions for 80% credit

Course Expectations

Class participation:

No attendance grades

Ask questions (synchronously or asynchronously)

Participate in breakout rooms and polls

Mental Health

This class should be low-stress, light work-load.

UIUC offers a variety of confidential services:

Counseling Center: 217-333-3704

610 East John Street Champaign, IL 61820

McKinley Health Center: 217-333-2700

1109 South Lincoln Avenue, Urbana, Illinois 61801

Diversity, Equity, and Inclusion

"If you witness or experience racism, discrimination, micro-aggressions, or other offensive behavior, you are encouraged to bring this to the attention of…"

Staff (CAs and TAs for CS 225)

Faculty (Brad Solomon)

Campus Belonging Office (Link)

The Office of Student Conflict Resolution (Link)

CS CARES (Link)

Learning Objectives

Review fundamentals of strings

Introduce exact pattern matching problem

What is a string?

String S is a finite sequence of characters

Characters are drawn from alphabet Σ, usually assumed finite

Nucleic acid alphabet: { A, C, G, T } ← 4 letters

English: { 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 } ← 26 letters

What are some other alphabets we could use?

Digits (0-9)

Binary (0,1)

↳ other symbols / symbol based alphabets

↳ ASCII 256

↳ Unicode

What is a string… in C++?

***char*:** 1-byte (8-bit) character encoding [ASCII 256]

***std::string*:** uses *char* alphabet (by default), has significant operation support

string_main.cpp

```
1  #include <string>
2  #include <iostream>
3
4  int main() {
5
6    char c[] = "Hello World";
7
8    std::string str = "Hello World";
9
10   return 0;
11 }
```

Fundamental operations

(ADT)

Math

+

-

%

/

*

Strings

concatenation

substring / remove string

length

getData / index

deleting / make new string

Fundamental string operations

Big O

“How efficient is my algorithm at searching for a given pattern P ?”

Length | Size

“How much memory do I need to allocate for this text file?”

Fundamental string operations

Size of *S*, |*S*|: The number of characters in *S*.

S = “How big?”

|S| = 8

9 b/c null ptr

10

Fundamental string operations

Size of *S*, |*S*|: The number of characters in *S*.

S = “How big?”

|S| = **8**

0	1	2	3	4	5	6	7
H	o	w	_	b	i	g	?

Fundamental string operations

Size of *S*, |*S*|: The length of *S* (in terms of bytes).

`S.length()`

size.cpp

```
 1  #include <string>
 2  #include <iostream>
 3
 4  int main() {
 5    std::string S = "Is this a string?";
 6    std::string T = "No, this is Patrick.";
 7
 8    std::cout << S.length() << std::endl;
 9    std::cout << T.length() << std::endl;
10
11    return 0;
12  }
13
14
```

Fundamental string operations

“Is this book about data structures?”

Accessing data

Equality

“Is this student enrolled at UIUC?”

Fundamental string operations

S ***equals*** T if each character, in order, is the same

S == T

equals.cpp

```
1  #include <string>
2  #include <iostream>
3
4  int main() {
5    std::string S = "Thing 1";
6    std::string T = "Thing 1";
7
8    if (S == T){
9      std::cout << "S == T" << std::endl;
10   } else {
11     std::cout << "S != T" << std::endl;
12   }
13  return 0;
14 }
```

Fundamental string operations

S ***equals*** T if each character, in order, is the same

S == T

char_equals.cpp

```
1  #include <string>
2  #include <iostream>
3
4  int main() {
5    char S[] = "Thing 1";
6    char T[] = "Thing 1";
7
8    if (S == T){
9      std::cout << "S == T" << std::endl;
10   } else {
11     std::cout << "S != T" << std::endl;
12   }
13  return 0;
14 }
```

?

Compares objects not values

Compares pointer addresses

Fundamental string operations

S ***equals*** T if each character, in order, is the same

S == T

char_equals.cpp

```
substring.cpp:8:9: warning: array comparison always evaluates to false [-Wtautological-compare]
  if (S == T){
      ^
```

```
1  int main() {
2    char S[] = "Thing 1";
3    char T[] = "Thing 1";
4    if (S == T){
5      std::cout << "S == T" << std::endl;
6    } else {
7      std::cout << "S != T" << std::endl;
8    }
9   return 0;
10 }
```

S.equal(T)

!!!

!=

Fundamental string operations

Reads

GTATGCACGCGATAG	TATGTCGCAGTATCT	CACCCTATGTCGCAG	GAGACGCTGGAGCCG
TAGCATTGCGAGACG	GGTATGCACGCGATA	TGGAGCCGGAGCACC	CGCTGGAGCCGGAGC
TGTCTTTGATTCCTG	CGCGATAGCATTGCG	GCATTGCGAGACGCT	CCTATGTCGCAGTAT
GACGCTGGAGCCGGA	GCACCCTATGTCGCA	GTATCTGTCTTTGAT	CCTCATCCTATTATT
TATCGCACCTACGTT	CAATATTCGATCATG	GATCACAGGTCTATC	ACCCTATTAACCACT
CACGGGAGCTCTCCA	TGCATTTGGTATTTT	CGTCTGGGGGGTATG	CACGCGATAGCATTG
GTATGCACGCGATAG	ACCTACGTTCAATAT	TATTTATCGCACCTA	CCACTCACGGGAGCT
GCGAGACGCTGGAGC	CTATCACCCTATTAA	CTGTCTTTGATTCCT	ACTCACGGGAGCTCT
CCTACGTTCAATATT	GCACCTACGTTCAAT	GTCTGGGGGGTATGC	AGCCGGAGCACCCTA
GACGCTGGAGCCGGA	GCACCCTATGTCGCA	GTATCTGTCTTTGAT	CCTCATCCTATTATT
TATCGCACCTACGTT	CAATATTCGATCATG	GATCACAGGTCTATC	ACCCTATTAACCACT
CACGGGAGCTCTCCA	TGCATTTGGTATTTT	CGTCTGGGGGGTATG	CACGCGATAGCATTG

Genome

identifying overlap?

CGTCTGGGGGGTATGCACGCGATAGCATTGCGAGACGCTGGAGCCGGAGCACCCTATGTCGCAGTATCTGTCTTTGATTCCTG

substring + equality

Fundamental string operations

Concatenation of *S* and *T*: characters of *S* followed by characters of *T*

S = “Beep” T = “Boop”

What is the string *ST* ? "BeepBoop"

What is the string *T$S* ? "Boop$Beep"

Fundamental string operations

Concatenation of *S* and *T*: characters of *S* followed by characters of *T*

S + T

concat.cpp

```
#include <string>
#include <iostream>

int main() {
  std::string S = "Beep";
  std::string T = "Boop";

  std::cout << S + T << std::endl;
  std::cout << T + S << std::endl;

  std::cout << S + '$' + T << std::endl;
  std::cout << T + '$' + S << std::endl;
}

```

Fundamental string operations

1) Equality
2) Substring

"Is this book about data structures?"

S: `Data Structures`

1.1 Why Compact Data Structures?

T:

Google's stated mission, "to organize the world's information and make it universally accessible and useful," could not better capture the immense ambition of modern society for gathering all kinds of data and putting them to use to improve our lives. We are collecting not only huge amounts of data from the physical world (astronomical, climatological, geographical, biological), but also human-generated data (voice, pictures, music, video, books, news, Web contents, emails, blogs, tweets) and society-based behavioral data (markets, shopping, traffic, clicks, Web navigation, likes, friendship

Fundamental string operations

S is a ***substring*** of *T* if there exists (possibly empty) strings *u* and *v* such that T = *uSv*

A ***substring*** is a sequence of characters (a string) contained within another string

S: `p e p p e r`

T: `I _ l i k e _ p e p p e r o n i _ p i z z a`

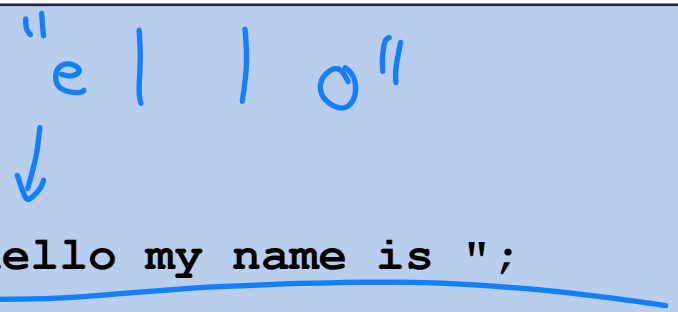
Fundamental string operations

A ***substring*** of *S* is a string occurring inside *S*

`S.substr(size_t pos, size_t len)`

substring.cpp

```
1  #include <string>
2  #include <iostream>
3
4  int main() {
5    std::string T = "Hello my name is ";
6
7    std::cout << T.substr(1,4) << std::endl;
8
9    return 0;
10 }
```



Fundamental string operations

A ***substring*** of *S* is a string occurring inside *S*

```
S.substr(size_t pos, size_t len)
```

substring.cpp

```
1  #include <string>
2  #include <iostream>
3
4  int main() {
5    std::string T = "Hello my name is ";
6
7    std::cout << T.substr(1,4) << std::endl;
8
9    return 0;
10 }
```

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
H	e	l	l	o	_	m	y	_	n	a	m	e	_	i	s	_

Fundamental string operations

S is a ***prefix*** of *T* if there exists a string *v* such that *T* = *Sv*

A ***prefix*** `is a substring` *T=uSv* `where` *u=“”*

T: **G T T A T A G C T G A T**

G T T A T A G C T G A T

S *v*

substring starts at index 0

Fundamental string operations

S is a ***prefix*** of *T* if there exists a string *v* such that *T* = *Sv*

T: **G T T A T A G C T G A T**

G T T A T A G C T G A T
G T T A T A G C T G A
G T T A T A G C T G
G T T A T A G C T
G T T A T A G C
G T T A T A G
G T T A T A
G T T A T
G T T A
G T T
G T
G

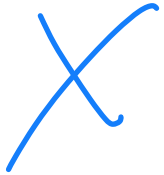
Fundamental string operations

S is a ***prefix*** of *T* if there exists a string *v* such that *T* = *Sv*

T: `P a t t e r n  m a t c h i n g`

`P a t t e r`

`m a t c h i n g`



`P a t r i c k`

Fundamental string operations

S is a ***prefix*** of *T* if there exists a string *v* such that *T* = *Sv*

T: "`Pattern matching`"

`Patter` ✓

`matching` ✗

`Patrick` ✗

Fundamental string operations

S is a ***suffix*** of *T* if there exists a string *u* such that $T = uS$

*A **suffix** is a substring* $T=uSv$ *where* $v=$“”

T: **G T T A T A G C T G A T**

G T T A T A G C T G A T

u *S*

substring which ends at last char

Fundamental string operations

S is a ***suffix*** of *T* if there exists a string *u* such that $T = uS$

Fundamental string operations

S is a ***suffix*** of *T* if there exists a string *u* such that $T = uS$

T: `P a t t e r n  m a t c h i n g`

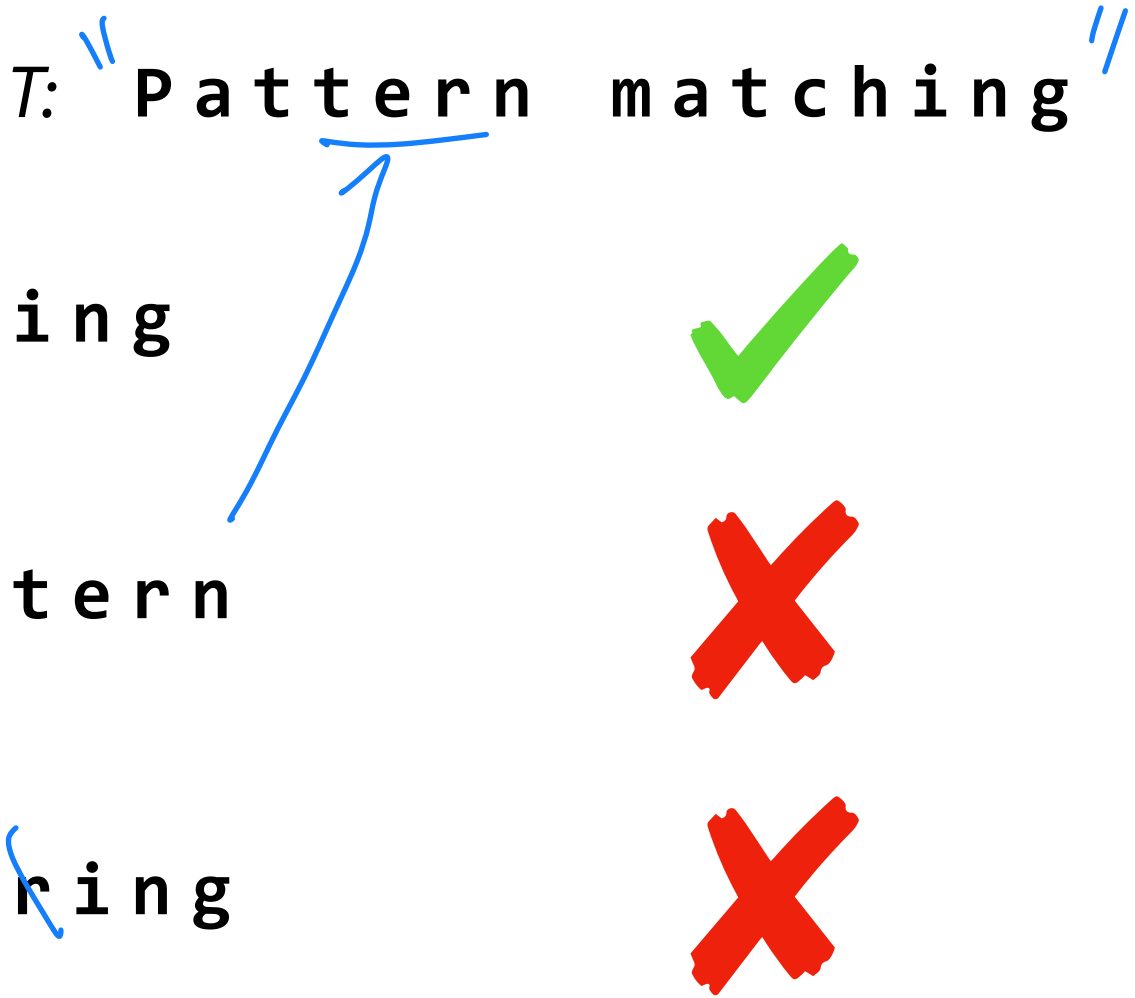
`i n g`

`t e r n`

`r i n g`

Fundamental string operations

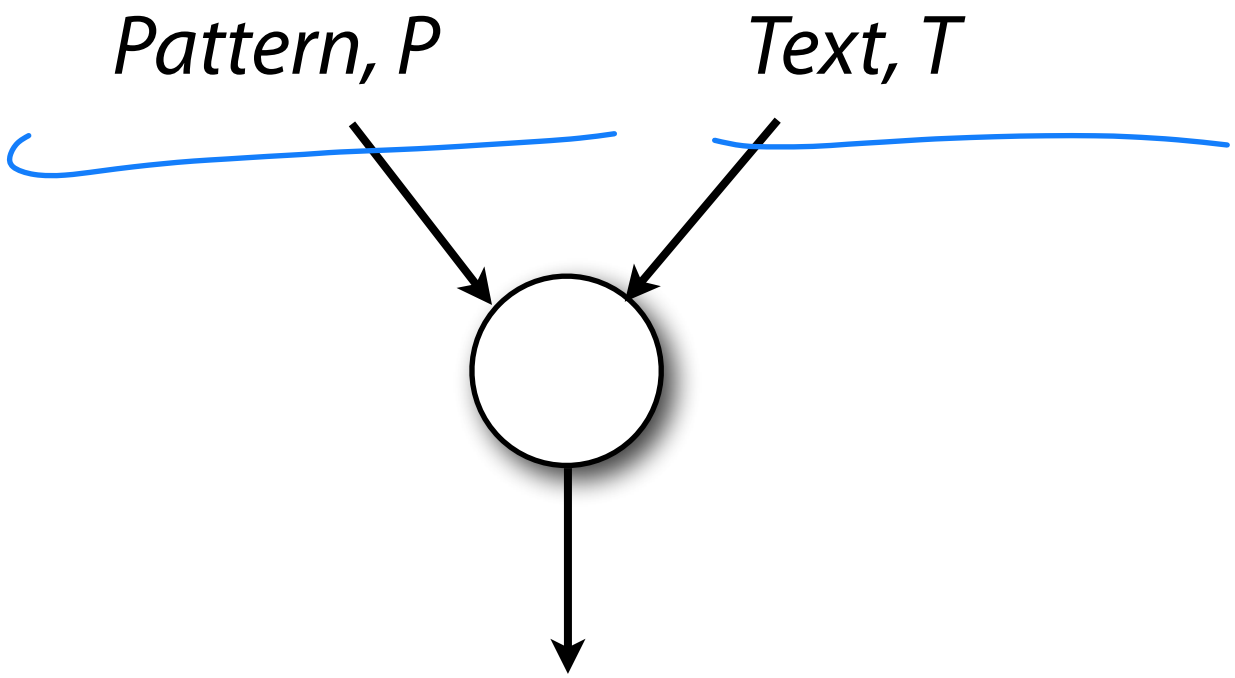
S is a ***suffix*** of *T* if there exists a string *u* such that $T = uS$



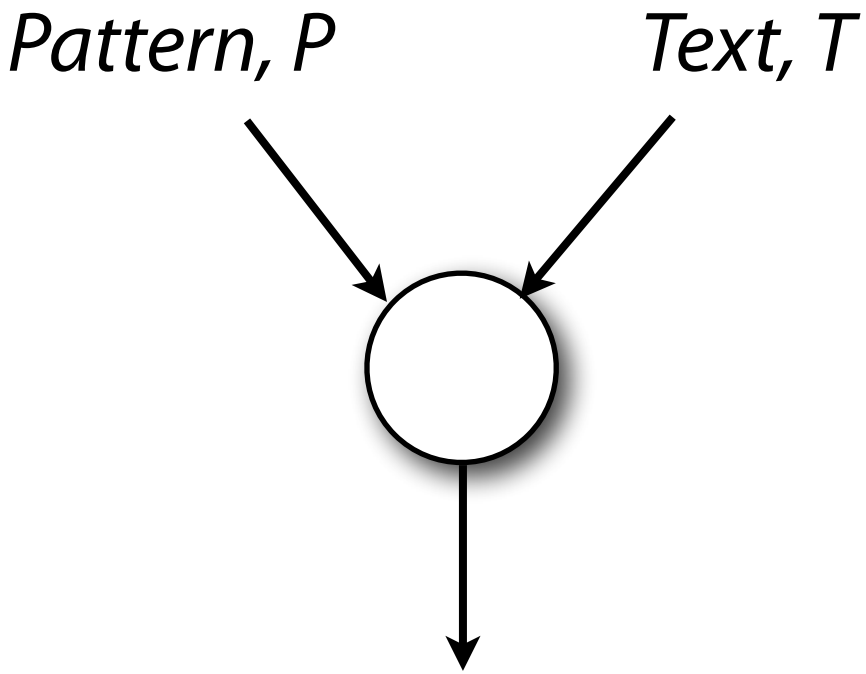
Fundamental string operations

Size, $\|S\|$	`S.length()`
Equals, $S == T$	`S == T`
Concatenation, ST	`S + T`
Substring, uSv	`S.substr(pos, len)`

Exact Pattern Matching



Exact Pattern Matching



Find instances of *P* in *T*

'instances': An exact, full length copy

Exact Pattern Matching

Find places where *pattern P* occurs as a substring of *text T*. Each such place is an *occurrence* or *match*.

P: `word`

T: `There would have been a time for such a word`

Exact Pattern Matching

Find places where *pattern P* occurs as a substring of *text T*. Each such place is an *occurrence* or *match*.

P: `word`

T: `There would have been a time for such a word`
`word`
`word`

Exact Pattern Matching

Find places where *pattern P* occurs as a substring of *text T*. Each such place is an *occurrence* or *match*.

P: `word`

T: `There would have been a time for such a word`

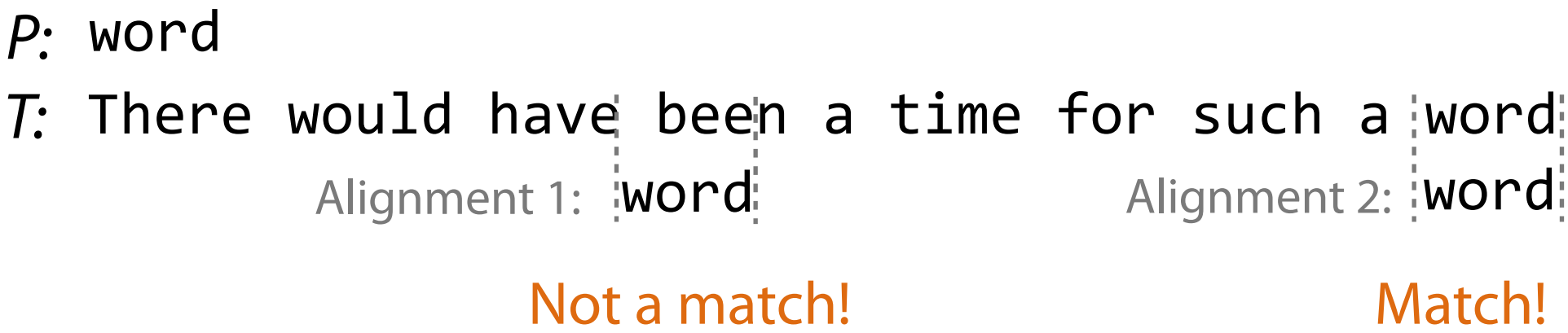
Alignment 1: `word`

Alignment 2: `word`

Alignment: a way of putting *P's* characters opposite *T's*. May or may not correspond to a match.

Exact Pattern Matching

Find places where *pattern P* occurs as a substring of *text T*. Each such place is an *occurrence* or *match*.



Alignment: a way of putting *P's* characters opposite *T's*. May or may not correspond to a match.

Exact Pattern Matching

What's a simple algorithm for exact matching?

P: **word**

T: **There would have been a time for such a word**

Loop through every character

↳ substring |P| length

↳ compare for equality

Exact Pattern Matching

count of matches

What's a simple algorithm for exact matching?

index

P: **word**

T: **There would have been a time for such a word**
word word word word word word word word **word**
word word word word word word word word
word word word word word word word word
word word word word word word word word
word word word word word word word word

One occurrence

Try all possible alignments. For each, check if it matches. This is the *naïve algorithm*.

Assignment 1: a_naive

Learning Objective:

Conceptualize exact pattern matching w/ naïve search

Demonstrate understanding of fundamental operations

Think about as you code: is naïve search a good solution?

End-of-class brainstorm

How can we improve the naïve algorithm?

End-of-class brainstorm

How can we improve the naïve algorithm?

… if you have infinite space?

Store all substrings of the text

↳ of all length

O(1) ***

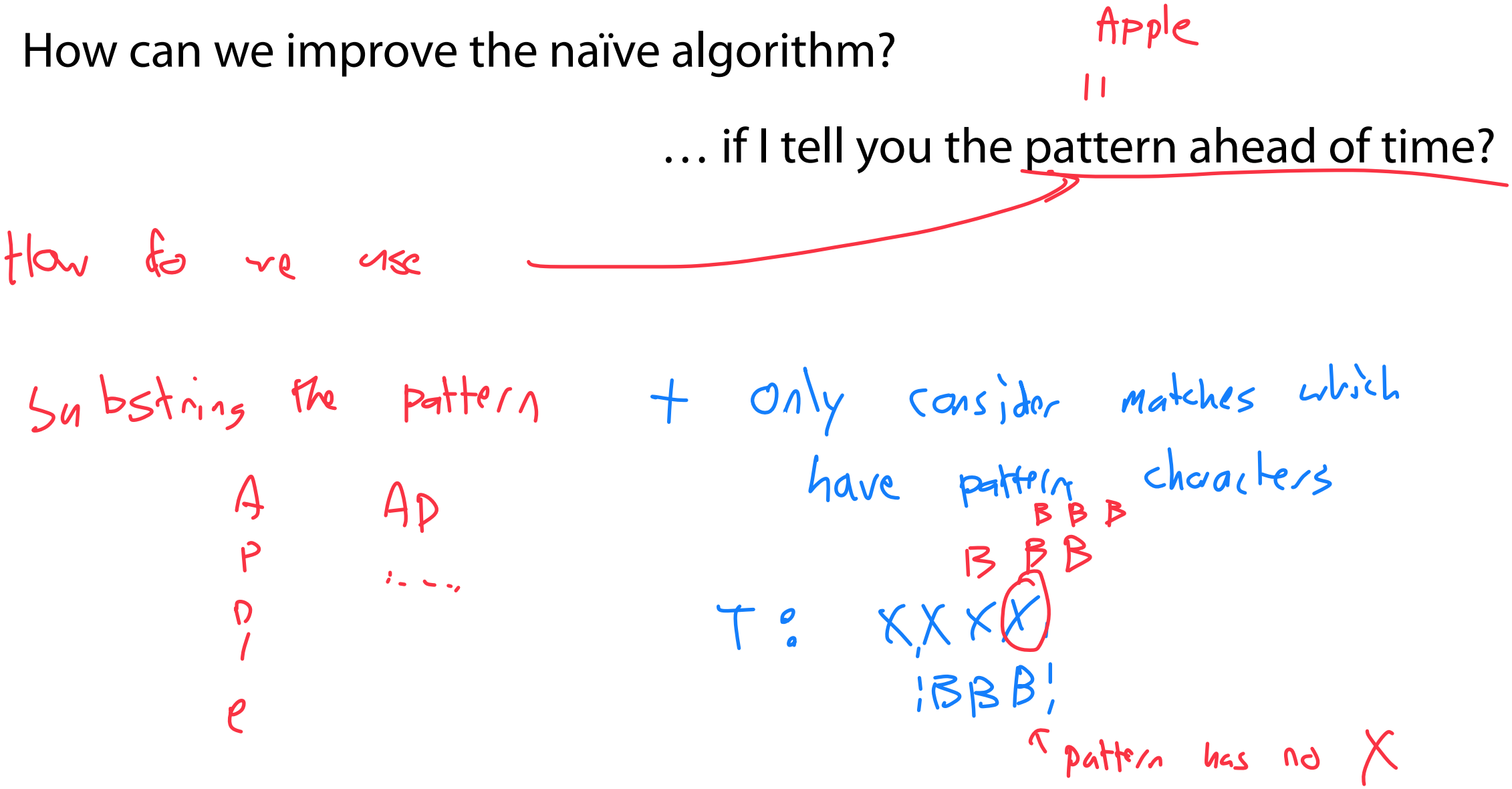
A → 1, 5, 8

At → 1, 8

End-of-class brainstorm

How can we improve the naïve algorithm?

… if I tell you the pattern ahead of time?



End-of-class brainstorm

How can we improve the naïve algorithm?

… if I tell you the text ahead of time?

preprocess for every possibl

The infinite space answer works here

Compare text to pattern size

End-of-class brainstorm

How can we improve the naïve algorithm?

… if you have infinite space?

… if I tell you the pattern ahead of time?

… if I tell you the text ahead of time?