String Algorithms and Data Structures

Introduction and Pattern Matching

CS 199-225

Brad Solomon

January 24, 2022

UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN

Department of Computer Science

Who am I?

Brad Solomon (PhD, CMU)

Teaching Assistant Professor, Computer Science

2233 Siebel Center for Computer Science

Email: bradsol@illinois.edu

Office Hours:

Thursdays, 10:30-12:00 PM (Zoom)

… or by appointment

https://courses.engr.illinois.edu/cs225/sp2022/info/office-hours/

Who are you?

Take a moment to introduce yourself in chat!

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

https://campuswire.com/c/G5A2320D7

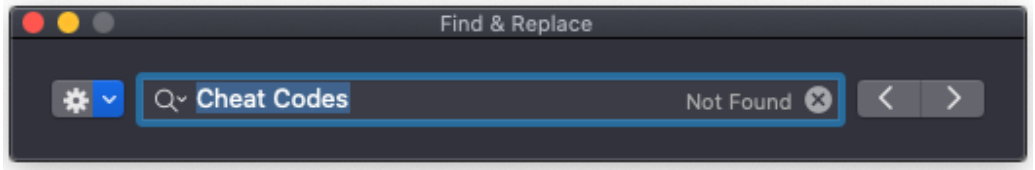
What is this class about?

Foundational

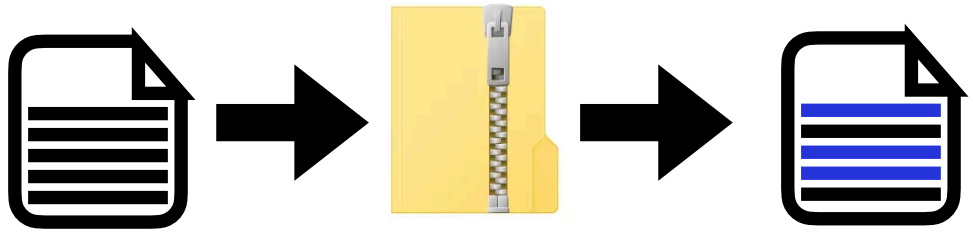
String Algorithms and Data Structures

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.engr.illinois.edu/cs225/sp2022/pages/honors.html

All course information and links can be found here!

- Zoom
- Mediaspace recordings
- Campuswire
- Moodle
- 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 (1000 points):

- Small assignments (~ 2-3 hours / week)
- No lab sections
- Must submit your own work
- Two 24-hour extensions by request
- The lowest homework grade will be dropped

Course Expectations

Class participation (0 points):

No attendance grades

Ask questions (synchronously or asynchronously)

Participate in breakout rooms and polls

Grading

Total points in class: 1000

A: [900, 1000]

B: [800, 900)

C: [700, 800)

D: [600, 700)

F: [0, 600)

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 (Myself or Carl)

BART (https://bart.illinois.edu/)

For more details about the BART response policy, refer to https://bart.illinois.edu/procedures/

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 }

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 }

What are some other alphabets we could use?

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

Math

Strings

Fundamental string operations

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

“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| = ?

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?”

“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

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

int main() {
  char S[] = "Thing 1";
  char T[] = "Thing 1";

  if (S == T){
    std::cout << "S == T" << std::endl;
  } else {
    std::cout << "S != T" << std::endl;
  }
 return 0;
}
```

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

CGTCTGGGGGGTATGCACGCGATAGCATTGCGAGACGCTGGAGCCGGAGCACCCTATGTCGCAGTATCTGTCTTTGATTCCTG

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* ?

What is the string *T$S* ?

Fundamental string operations

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

S + T

concat.cpp

```
 1  #include <string>
 2  #include <iostream>
 3
 4  int main() {
 5    std::string S = "Beep";
 6    std::string T = "Boop";
 7
 8    std::cout << S + T << std::endl;
 9    std::cout << T + S << std::endl;
10
11    std::cout << S + '$' + T << std::endl;
12    std::cout << T + '$' + S << std::endl;
13  }
14
```

Fundamental string operations

“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 }
```

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*

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**

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*

Fundamental string operations

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

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

Fundamental string operations

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

T: `Pattern matching`

`ing` 

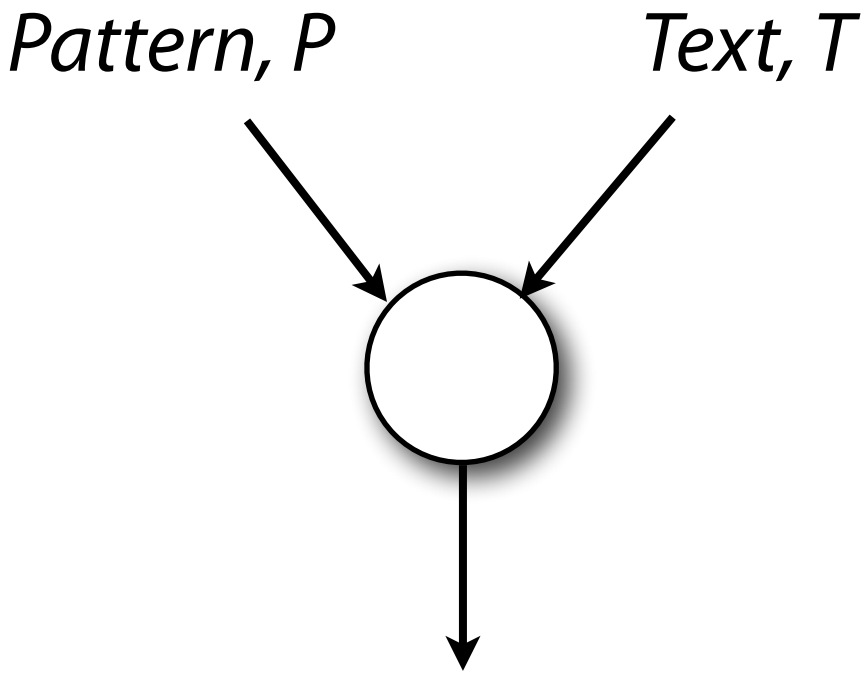
`tern` 

`ring` 

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

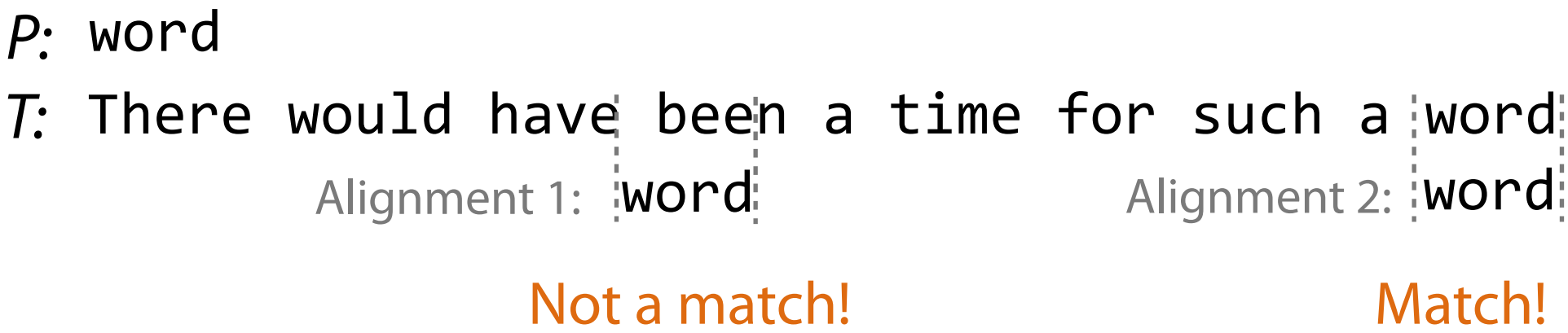


Find instances of *P* in *T*

'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*.



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**

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

Due: January 31st 11:59 PM

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

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?