Applied Databases

Lecture 12
Online Pattern Matching on Strings

Sebastian Maneth

University of Edinburgh - March 2nd, 2017

Outline

First → some comments wrt Assignment 1

---

1. Naive Method
2. Automaton Method
3. Knuth-Morris-Pratt Algorithm
4. Boyer-Moore Algorithm

String Matching

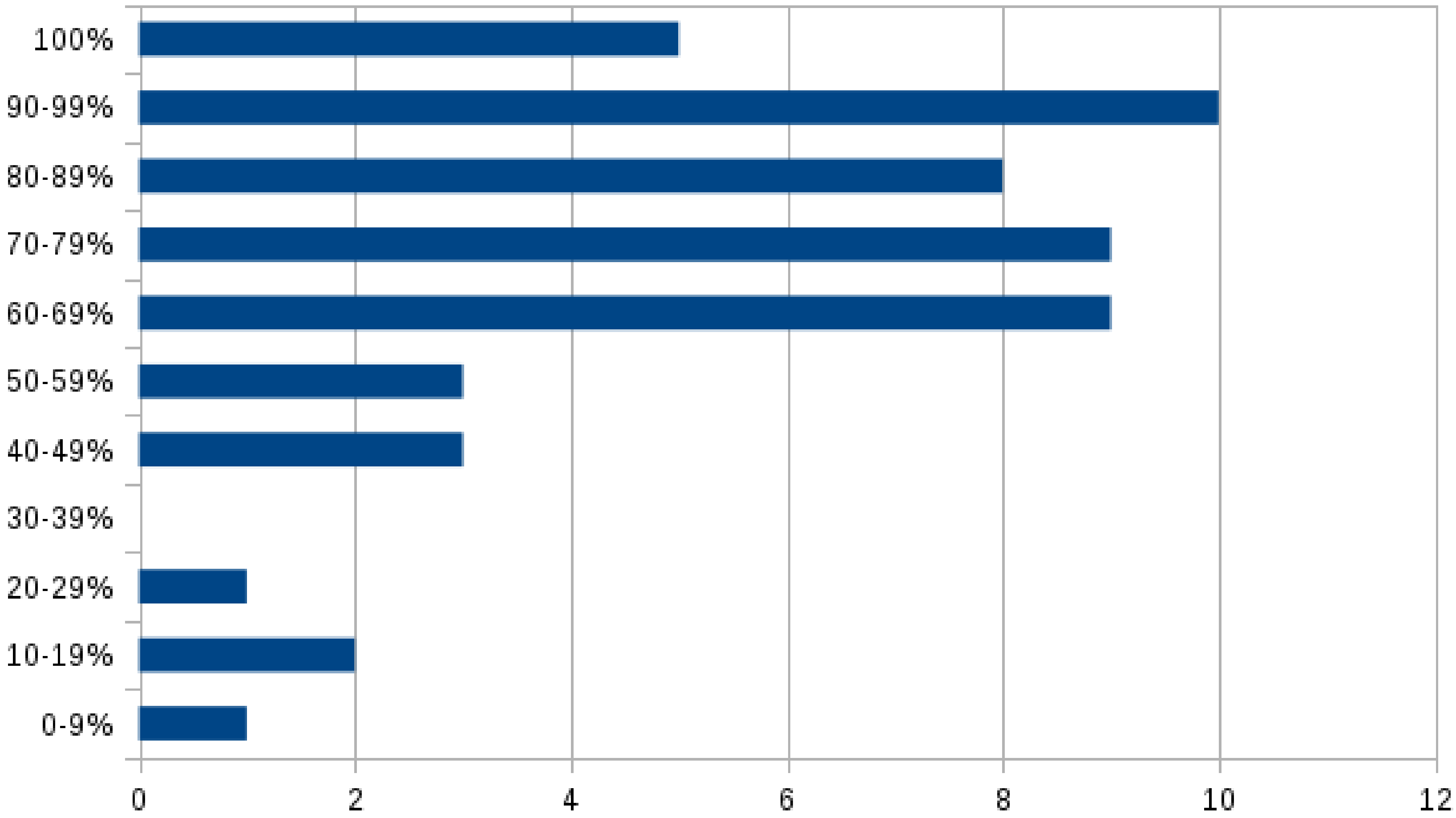
Assignment 1

Automation works correctly and independently of VM:	1 Point
Program compiles and produces some non-empty csv-files:	4 Points
Program successfully loads some data into the database:	1 Point
Data loaded into database is correct, given the DB design:	2 Point
drop.sql: Works correctly without error:	0.5 Points
SQL-scripts have no (or only minor) syntax errors:	0.5 Points
Database does not use any NULL-Values:	2 Points
Long descriptions are correctly truncated:	1 Point
Duplicate entries are correctly removed in the csv-files:	1 Point
All Queries correct:	3.5 Points
	16.5 Points

Theoretical Part (schema design & normal forms): **3.5 Points**

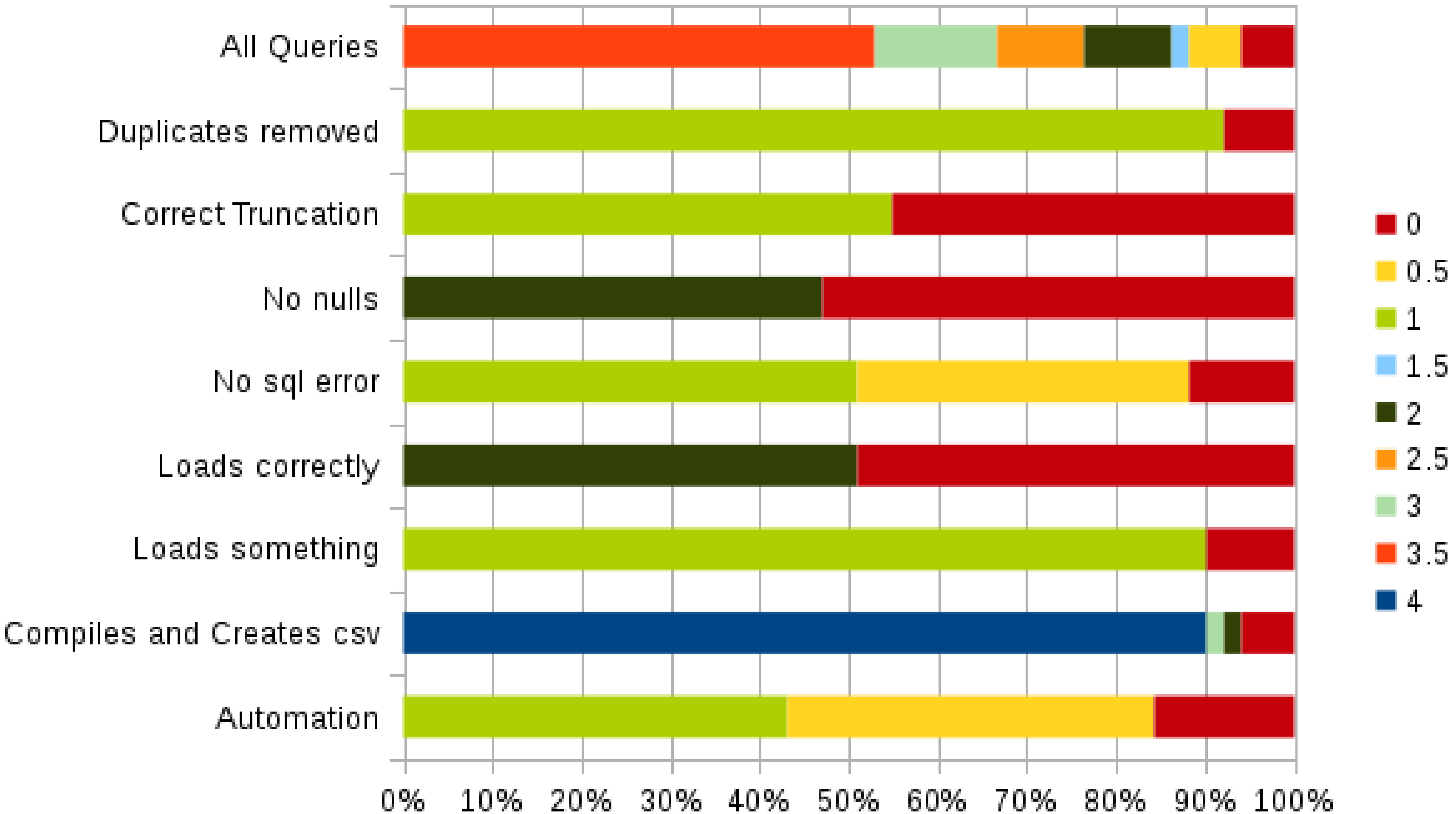
We are still marking these.
Marks will be finalized by tomorrow (Friday) evening.

Assignment 1



Marks so far (out of 16.5 Points) – #submissions = 51

Assignment 1



Marking of Assignment 1

→ relational schema design (3.5 points)

- NULL or pseudo-NULLs (0.5 points)
- optionals of DTD correctly implemented (0.5 points)
- correct Primary Key for each table (0.25 = one error, 0.5 = two errors)

→ item-category: *many-to-many* relationship

table has_category(item_id, category)

→ primary key (item_id, category)
→ consequence: there cannot be duplicates!
→ original XML has such **duplicates**!

must be detected and
eliminated by your program
(not through mySQL)

Marking of Assignment 1

→ relational schema design (3.5 points)

- NULL or pseudo-NULLs (0.5 points)
- optionals of DTD correctly implemented (0.5 points)
- correct Primary Key for each table (0.25 = one error, 0.5 = two errors)

→ item-category: *many-to-many* relationship

table has_category(item_id, category)

→ primary key (item_id, category)
→ consequence: there cannot be duplicates!
→ original XML has such **duplicates**!

```
<Item ItemID="1310018094">
    <Name>2 lanzar 10" DC subs 1000 watt subwoofers</Name>
    <Category>Consumer Electronics</Category>
    <Category>Car Audio &amp; Electronics</Category>
    <Category>Subwoofers</Category>
    <Category>Subwoofers</Category>
    <Category>10 Inch</Category>
    <Currently>$175.00</Currently>
```

has exactly **four** categories, not five!

Bid Table

→ relational schema design (3.5 points)

- NULL or pseudo-NULLs (0.5 points)
- optionals of DTD correctly implemented (0.5 points)
- correct Primary Key for each table (0.25 = one error, 0.5 = two errors)

→ bid table

- **item_id**
- **bidder_id**
- **time**
- **amount**

→ keys of this table?

Bid Table

→ relational schema design (3.5 points)

- NULL or pseudo-NULLs (0.5 points)
- optionals of DTD correctly implemented (0.5 points)
- correct Primary Key for each table (0.25 = one error, 0.5 = two errors)

→ bid table

- **item_id**
- **bidder_id**
- **time**
- **amount**

not keys:

1) (**item_id, bidder_id**) – bidder can bid multiple times for same item!

Bid Table

→ relational schema design (3.5 points)

- NULL or pseudo-NULLs (0.5 points)
- optionals of DTD correctly implemented (0.5 points)
- correct Primary Key for each table (0.25 = one error, 0.5 = two errors)

→ bid table
- **item_id**
- **bidder_id**
- **time**
- **amount**

not keys:

1) (**item_id, bidder_id**) – bidder can bid multiple times for same item!
2) (**bidder_id, time**) – bidder can make multiple bids at same time!
(e.g., multiple times logged in,
bidding per software, etc.)

Bid Table

→ relational schema design (3.5 points)

- NULL or pseudo-NULLs (0.5 points)
- optionals of DTD correctly implemented (0.5 points)
- correct Primary Key for each table (0.25 = one error, 0.5 = two errors)

→ bid table

- item_id
- bidder_id
- time
- amount

→ is this a key?

(**item_id, bidder_id, time**)

Bid Table

→ relational schema design (3.5 points)

- NULL or pseudo-NULLs (0.5 points)
- optionals of DTD correctly implemented (0.5 points)
- correct Primary Key for each table (0.25 = one error, 0.5 = two errors)

→ bid table

- item_id
- bidder_id
- time
- amount

→ is this a key?

(**item_id, bidder_id, time**)

NO! → It is not minimal

Bid Table

→ relational schema design (3.5 points)

- NULL or pseudo-NULLs (0.5 points)
- optionals of DTD correctly implemented (0.5 points)
- correct Primary Key for each table (0.25 = one error, 0.5 = two errors)

→ bid table
- item_id
- bidder_id
- time
- amount

→ is this a key?

(**item_id, bidder_id, time**)

NO! → It is not minimal

Correct keys:

→ (**item_id, time**)

Bid Table

→ relational schema design (3.5 points)

- NULL or pseudo-NULLs (0.5 points)
- optionals of DTD correctly implemented (0.5 points)
- correct Primary Key for each table (0.25 = one error, 0.5 = two errors)

→ bid table

- item_id
- bidder_id
- time
- amount

→ is this a key?

(**item_id, ~~bidder_id~~, time**)

NO! → It is not minimal

Correct keys:

→ (**item_id, time**)

Any other keys?

Bid Table

→ relational schema design (3.5 points)

- NULL or pseudo-NULLs (0.5 points)
- optionals of DTD correctly implemented (0.5 points)
- correct Primary Key for each table (0.25 = one error, 0.5 = two errors)

→ bid table
- item_id
- bidder_id
- time
- amount

→ is this a key?

(**item_id, bidder_id, time**)

NO! → It is not minimal

Correct keys:

→ (**item_id, time**)

→ (**item_id, amount**)

Marking of Assignment 1

→ relational schema design (3.5 points)

- NULL or pseudo-NULLs (0.5 points)
- optionals of DTD correctly implemented (0.5 points)
- correct Primary Key for each table (0.25 = one error, 0.5 = two errors)
- correct Functional Dependencies (0.5 points)
- 4NF (0.5 points)
  [either you claim 4NF/BCNF but it isn't, or vice versa]

<= 2.5 penality points

If you wrote **something** for this part,
you obtain 1 Point by default! :-)

Marking of Assignment 1

Queries

E.g., Number 3:

```
SELECT COUNT(y.item_id) FROM
  (SELECT item_id, COUNT(item_id) as count
   FROM has_category GROUP BY item_id) y
WHERE y.count=4;
```

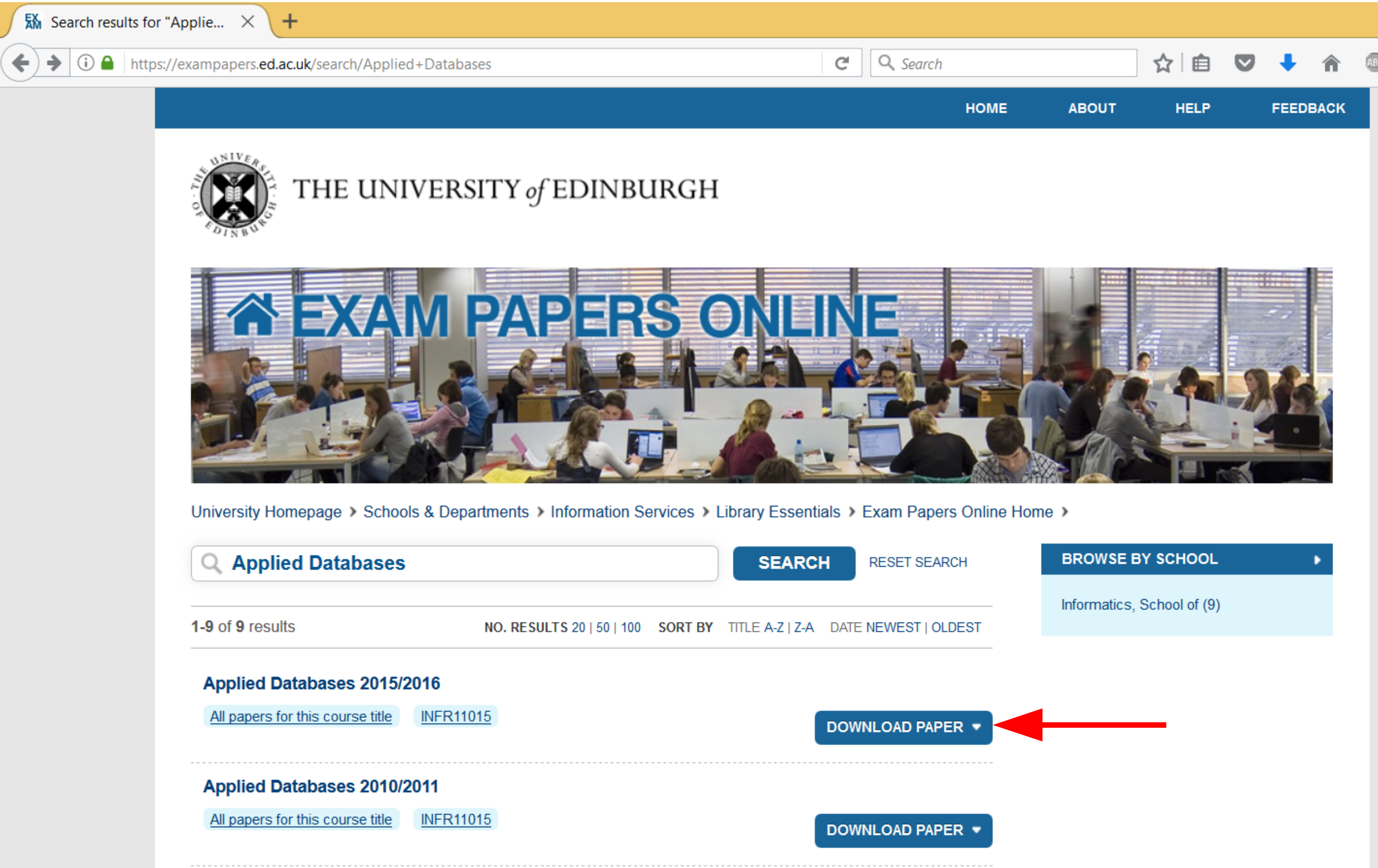
→ assumes duplicate-free has_category table

→ if has_category **has** duplicates, how to write the query?

Answers to Queries

Queries

1) Find the number of users in the database.
13422
2) Find the number of items in "New York",
103
3) Find the number of auctions belonging to exactly four categories.
8365
4) Find the ID(s) of current (unsold) auction(s) with the highest bid.
1046740686
5) Find the number of sellers whose rating is higher than 1000.
3130
6) Find the number of users who are both sellers and bidders.
6717
7) Find the number of categories that include at least one item with a bid of more than $100.
150


Search results for "Applie...
https://exampapers.ed.ac.uk/search/Applied+Databases
Search
HOME
ABOUT
HELP
FEEDBACK
THE UNIVERSITY of EDINBURGH
EXAM PAPERS ONLINE
University Homepage › Schools & Departments › Information Services › Library Essentials › Exam Papers Online Home ›
Applied Databases
SEARCH
RESET SEARCH
BROWSE BY SCHOOL
Informatics, School of (9)
1-9 of 9 results
NO. RESULTS 20 | 50 | 100
SORT BY TITLE A-Z | Z-A DATE NEWEST | OLDEST
Applied Databases 2015/2016
All papers for this course title
INFR11015
DOWNLOAD PAPER
Applied Databases 2010/2011
All papers for this course title
INFR11015
DOWNLOAD PAPER

2. (A) Schemas. [*Bookwork: 4 * 1.5 = 6 marks*]

(1) A functional dependency means that some columns determine the values of other columns. Consider a table with columns Street, City, and ZipCode. List all the functional dependencies for this table. (this is for the US, where a ZipCode does not determine a street).

(2) Explain what is a superkey, what is a key, and what is a primary key.

(3) List all keys for a table of biddings of an online bidding system (where the same bidder can make multiple bids at a time) with column names ItemID, BidderID, Time, and Amount.

(4) Explain what a multi-valued dependency is, and give an example of one (that is not a functional dependency).

Full-Text Search

→ tokenize natural language documents
→ build inverted files

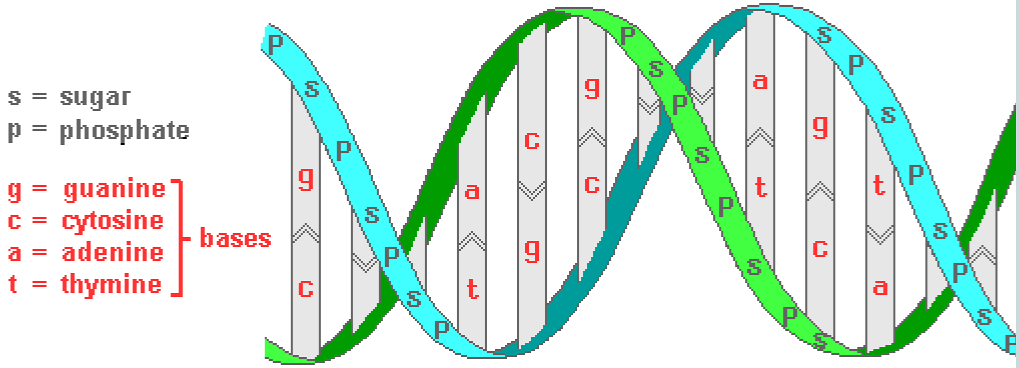
→ execute keyword-queries over inverted files
→ rank results according to TF - IDF-based scoring

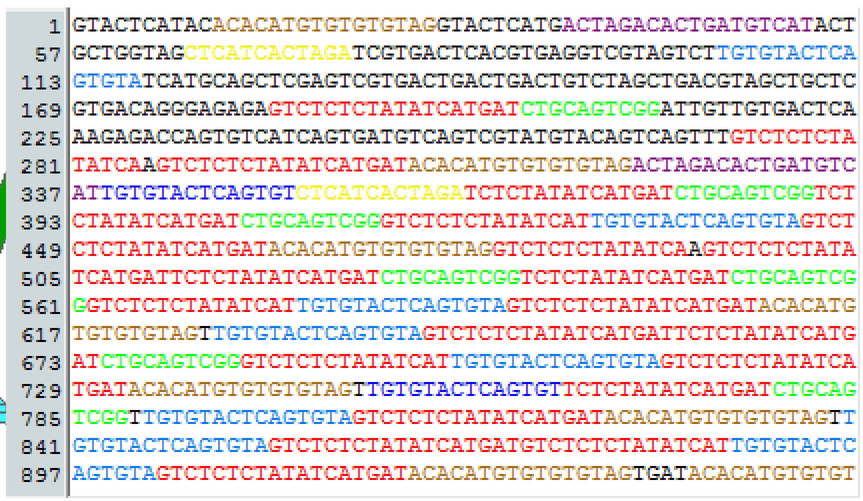
Full-Text Search

→ tokenize natural language documents
→ build inverted files

→ execute keyword-queries over inverted files
→ rank results according to TF - IDF-based scoring

Limits of this approach: → search over DNA sequences
→ huge sequences over C, T, G A (ca. 3.2 billion)
→ no spaces, no tokens....





1	GTACTCATACACACATGTGTGTGTAGGTACTCATGACTAGACACTGATGTCATACT
57	GCTGGTAGCTCATCACTAGATCGTGACTCACGTGAGGTCGTAGTCTTGTGTACTCA
113	GTGTATCATGCAGCTCGAGTCGTGACTGACTGACTGTCTAGCTGACGTAGCTGCTC
169	GTGACAGGGAGAGAGTCTCTCTATATCATGATCTGCAGTCGGATTGTTGTGACTCA
225	AAGAGACCAGTGTCATCAGTGATGTCAGTCGTATGTACAGTCAGTTTGTCTCTCTA
281	TATCAAGTCTCTCTATATCATGATACACATGTGTGTGTAGACTAGACACTGATGTC
337	ATTGTGTACTCAGTGTCTCATCACTAGATCTCTATATCATGATCTGCAGTCGGTCT
393	CTATATCATGATCTGCAGTCGGGTCTCTCTATATCATTGTGTACTCAGTGTAGTCT
449	CTCTATATCATGATACACATGTGTGTGTAGGTCTCTCTATATCAAGTCTCTCTATA
505	TCATGATTCTCTATATCATGATCTGCAGTCGGTCTCTATATCATGATCTGCAGTCG
561	GGTCTCTCTATATCATTGTGTACTCAGTGTAGTCTCTCTATATCATGATACACATG
617	TGTGTGTAGTTGTGTACTCAGTGTAGTCTCTCTATATCATGATTCTCTATATCATG
673	ATCTGCAGTCGGGTCTCTCTATATCATTGTGTACTCAGTGTAGTCTCTCTATATCA
729	TGATACACATGTGTGTGTAGTTGTGTACTCAGTGTTCTCTATATCATGATCTGCAG
785	TCGGTTGTGTACTCAGTGTAGTCTCTCTATATCATGATACACATGTGTGTGTAGTT
841	GTGTACTCAGTGTAGTCTCTCTATATCATGATGTCTCTCTATATCATTGTGTACTC
897	AGTGTAGTCTCTCTATATCATGATACACATGTGTGTGTAGTGATACACATGTGTGT

Pattern Matching on Strings

→ search over DNA sequences
→ huge sequence over C, T, G A (ca. 3.2 billion)
→ no spaces, no tokens....

Given
– a long string T (text) [often: over a fixed alphabet]
– a short string P (pattern)

Problem 1: find all occurrences of P in T
Problem 2: count #occurrence of P in T

Pattern Matching on Strings

→ search over DNA sequences
→ huge sequence over C, T, G A (ca. 3.2 billion)
→ no spaces, no tokens....

Given
– a long string T (text) [often: over a fixed alphabet]
– a short string P (pattern)

Problem 1: find all occurrences of P in T
Problem 2: count #occurrence of P in T

Two versions:

→ **offline** = we may index T, before running the search
→ **online** = directly run search (e.g., T not stored, comes in a stream)
[we may "index" P, this is called "preprocessing"]

Pattern Matching on Strings

Highlights
Online Search: O(|T|) time with O(|P|) preprocessing
Offline Search: O(|P| + #occ) time with O(|T|) preprocessing

Given
– a long string T (text)
– a short string P (pattern)

Problem 1: find all occurrences of P in T
Problem 2: count #occurrence of P in T

Two versions:

→ **offline** = we may index T, before running the search
→ **online** = directly run search (e.g., T not stored, comes in a stream)
[we may “index” P, this is called “preprocessing”]

Online Pattern Matching on Strings

Given
– short string P (pattern)
– long string T (text)

→ may preprocess P!

Problem 1: find all occurrences of P in T
Problem 2: count #occurrence of P in T

---

1) Automaton Method
→ build "match automaton A" for P and run A over T

2) Knuth-Morris-Pratt Algorithm
→ build jump-table for P and use it when traversing T

3) Boyer-Moore Algorithm
→ similar to KMP, but match *backwards* in P

1. Naive Method

Given Pattern P, Text T, find all occurrences of P in T.

P =

a	b	a	b	c

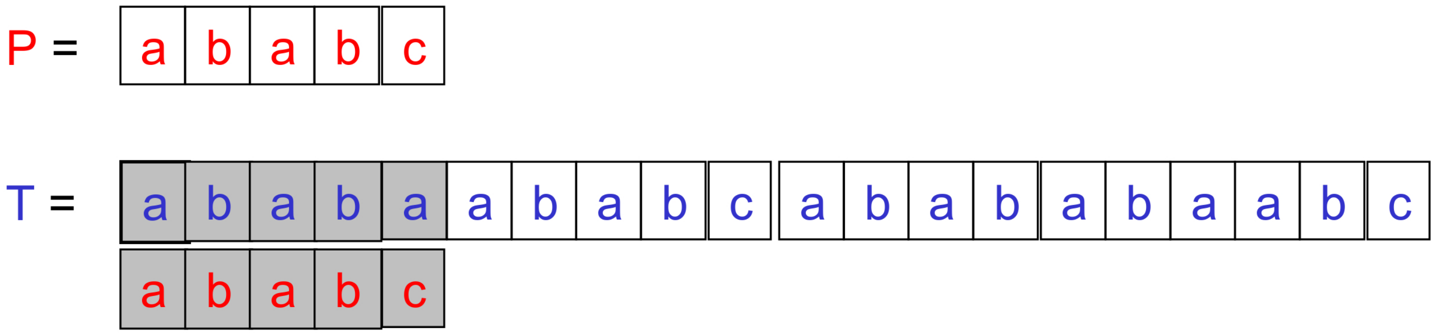
T =

a	b	a	b	a	a	b	a	b	c	a	b	a	b	a	b	a	a	b	c

→ at *each position* in T, try to match P

1. Naive Method

Given Pattern P, Text T, find all occurrences of P in T.



→ at *each position* in T, try to match P

Pos=1: mismatch (|P|=5 comparisons needed)

1. Naive Method

Given Pattern P, Text T, find all occurrences of P in T.

P = a b a b c

T = a b a b a a b a b c a b a b a b a a b c

a b a b c

→ at *each position* in T, try to match P

Pos=1: mismatch (|P|=5 comparisons needed)
Pos=2: mismatch (1)

1. Naive Method

Given Pattern P, Text T, find all occurrences of P in T.

P = | a | b | a | b | c |

T = | a | b | a | b | a | a | b | a | b | c | a | b | a | b | a | b | a | a | b | c |

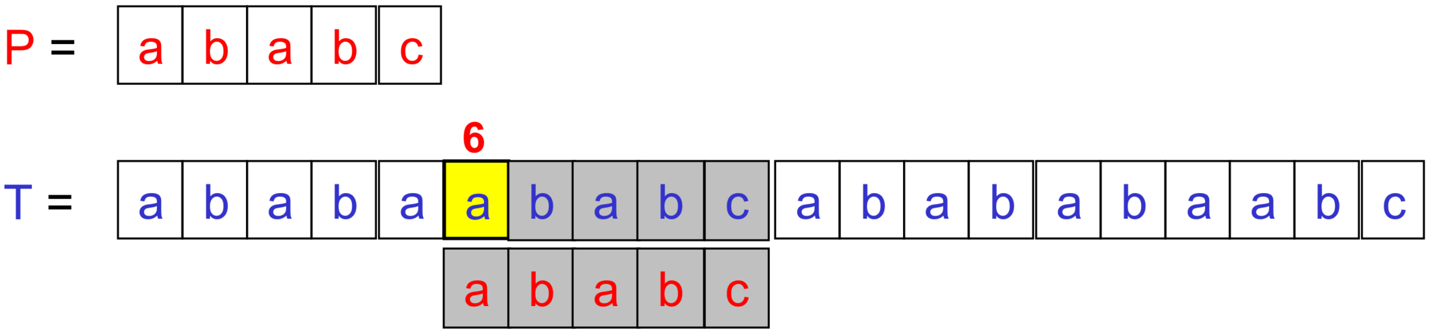
    | a | b | a | b | c |

→ at *each position* in T, try to match P

Pos=1: mismatch (|P|=5 comparisons needed)
Pos=2: mismatch (1)
Pos=3: mismatch (4)

1. Naive Method

Given Pattern P, Text T, find all occurrences of P in T.



→ at *each position* in T, try to match P

Pos=1: mismatch
Pos=2: mismatch
Pos=3: mismatch
….
Pos=6: match!

Result List: [6]

1. Naive Method

Given Pattern P, Text T, find all occurrences of P in T.

P = a b a b c

T = a b a b a a b a b c a b a b a b a a b c

a b a b c

Worst-Case Time Complexity

m := |P|
n := |T|

Naive takes $m(n - m + 1)$ comparisons.

Thus, $O(mn)$ time.

1. Naive Method

Given Pattern P, Text T, find all occurrences of P in T.

P =

a	b	a	b	c

T =

a	b	a	b	a	a	b	a	b	c	a	b	a	b	a	b	a	a	b	c
					a	b	a	b	c										

Worst-Case Time Complexity

m := |P|
n := |T|

Naive takes m(n – m + 1) comparisons.

Thus, O(mn) time.

Questions

Best-Case Complexity?

Average-Case Complexity?
(on random strings)

1. Naive Method

Given Pattern P, Text T, find all occurrences of P in T.

P =

a	b	a	b	c

T =

a	b	a	b	a	a	b	a	b	c	a	b	a	b	a	b	a	a	b	c
					a	b	a	b	c										

Worst-Case Time Complexity

m := |P|
n := |T|

Naive takes m(n – m + 1) comparisons.

Thus, O(mn) time.

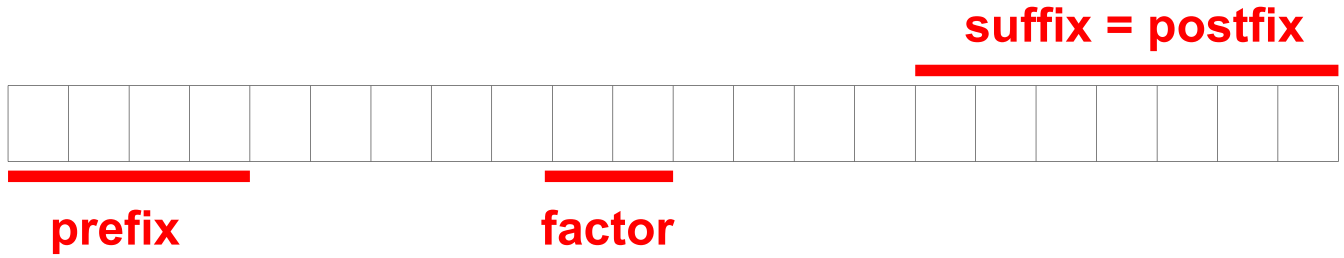
Note
Lower Bound on average:
O(n (log m)/m)
[A. C. Yao 1979]

Some Definitions

Word **v** is a **suffix** of word w, if w = u**v** for some u. (or: **postfix)**
("proper suffix", if u is non-empty)

Word **u** is a **prefix** of w, if w = **u**v for some v.
("proper prefix", if v is non-empty)

Word **u** is a **factor** of w, if there are v and v' such that w = v**u**v'



2. Automaton Method

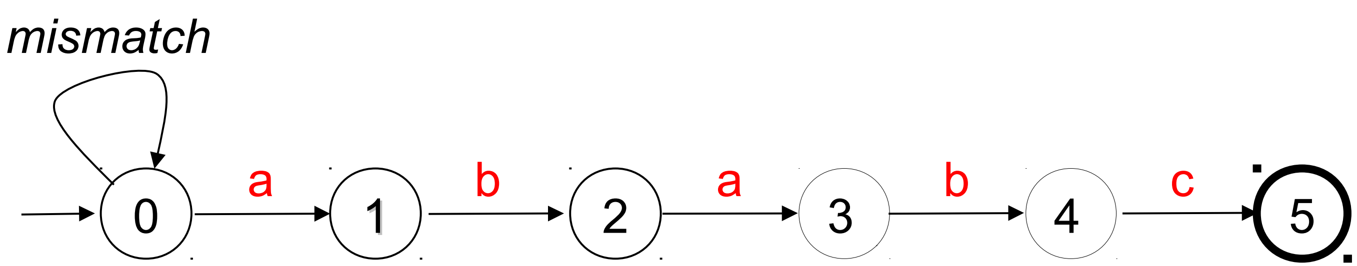
Given Pattern P, Text T, find all occurrences of P in T.

P =

a	b	a	b	c

T =

a	b	a	b	a	a	b	a	b	c	a	b	a	b	a	b	a	a	b	c



→ "*mismatch*" means "not a"

→ if character set C is known,
then for every **c** in C – {a}, we have one *transition* *d*(0, **c**) = 0

2. Automaton Method

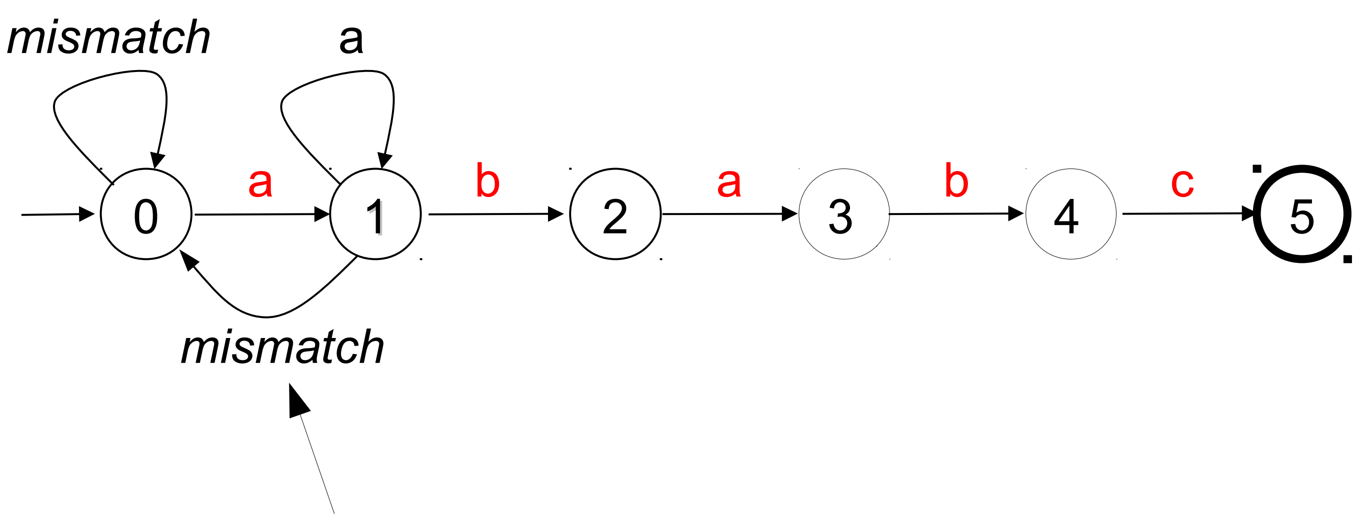
Given Pattern P, Text T, find all occurrences of P in T.

P =

a	b	a	b	c

T =

a	b	a	b	a	a	b	a	b	c	a	b	a	b	a	b	a	a	b	c



means “not a and not b”

2. Automaton Method

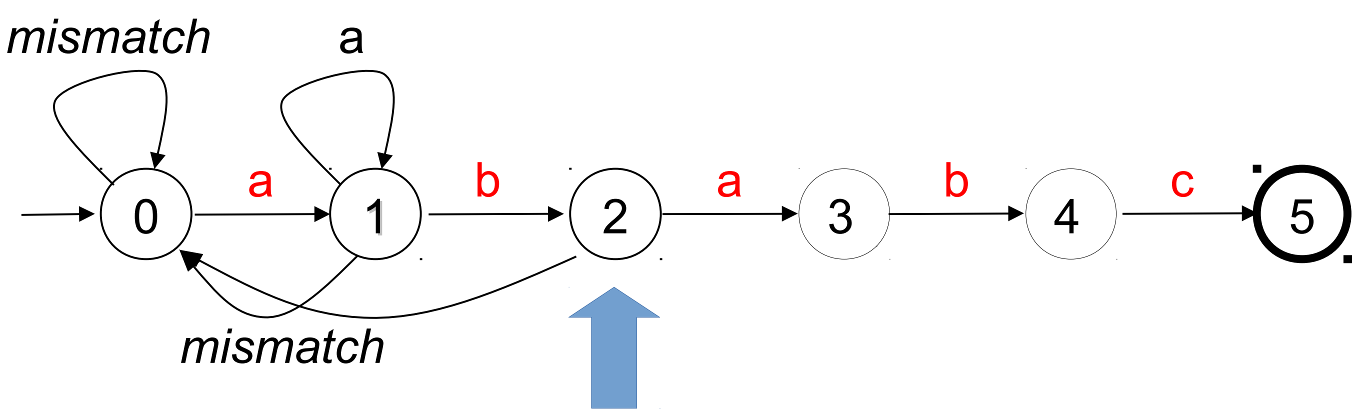
Given Pattern P, Text T, find all occurrences of P in T.

P =

a	b	a	b	c

T =

a	b	a	b	a	a	b	a	b	c	a	b	a	b	a	b	a	a	b	c



2. Automaton Method

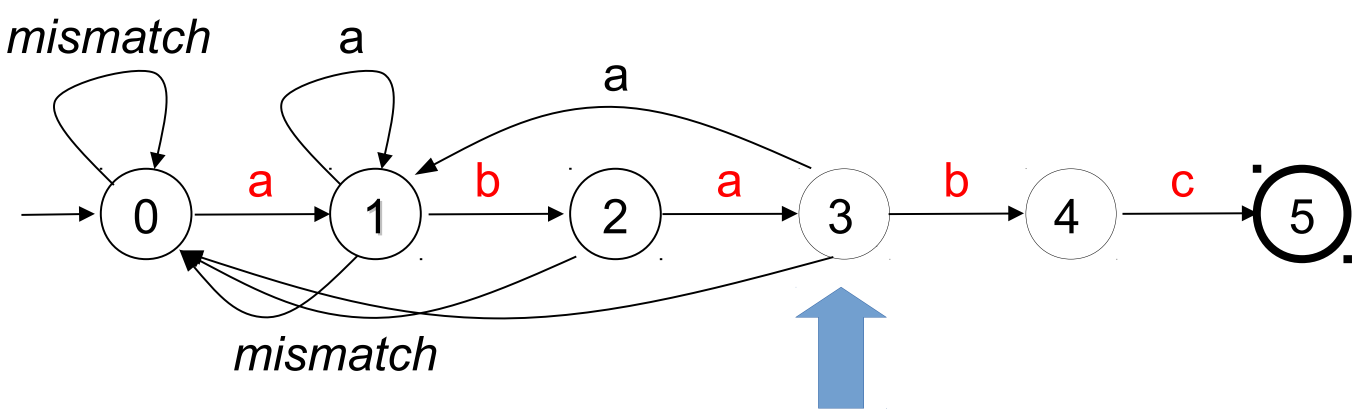
Given Pattern P, Text T, find all occurrences of P in T.

P =

a	b	a	b	c

T =

a	b	a	b	a	a	b	a	b	c	a	b	a	b	a	b	a	a	b	c



2. Automaton Method

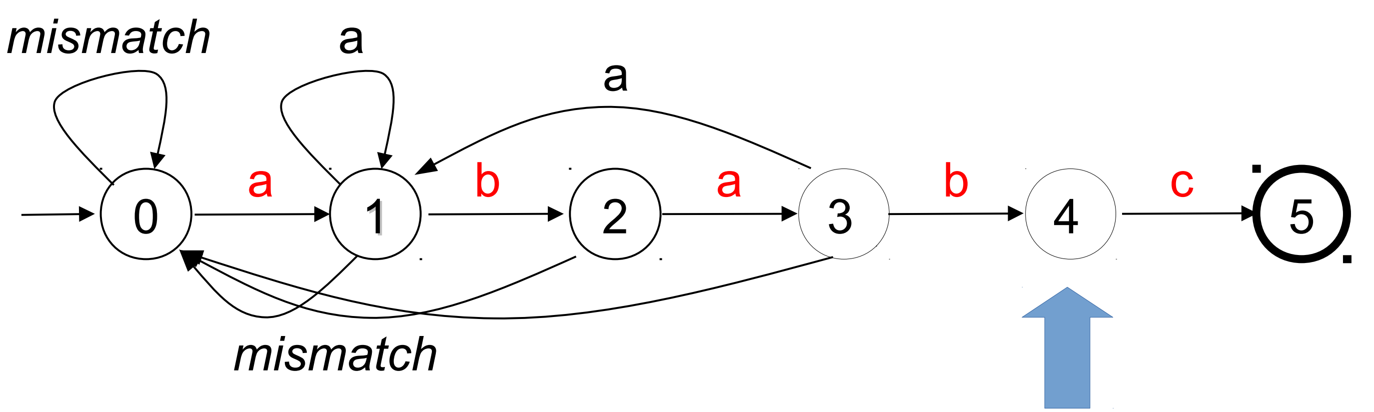
Given Pattern P, Text T, find all occurrences of P in T.

P =

a	b	a	b	c

T =

a	b	a	b	a	a	b	a	b	c	a	b	a	b	a	b	a	a	b	c



a-transition
→ where should it go???

2. Automaton Method

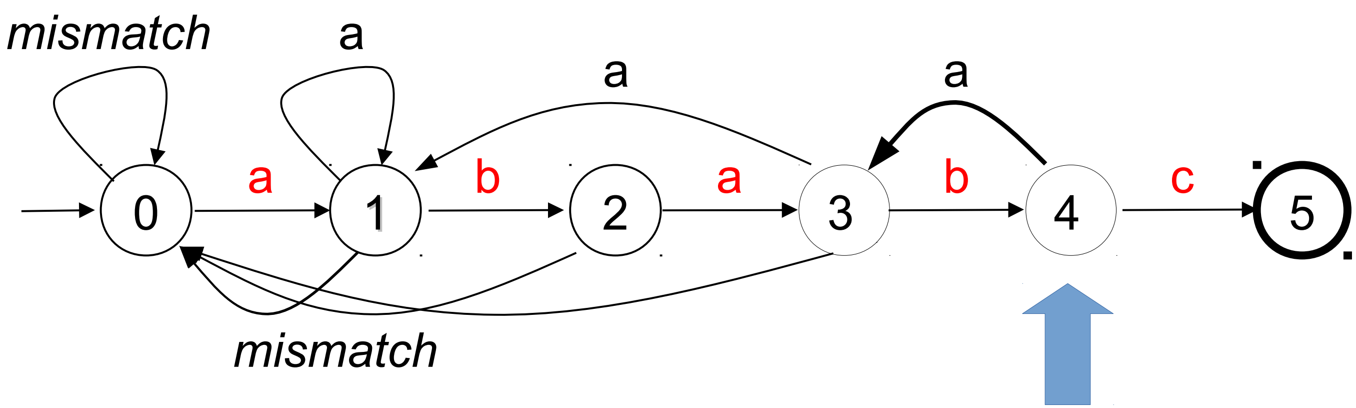
Given Pattern P, Text T, find all occurrences of P in T.

P =

a	b	a	b	c

T =

a	b	a	b	a	a	b	a	b	c	a	b	a	b	a	b	a	a	b	c



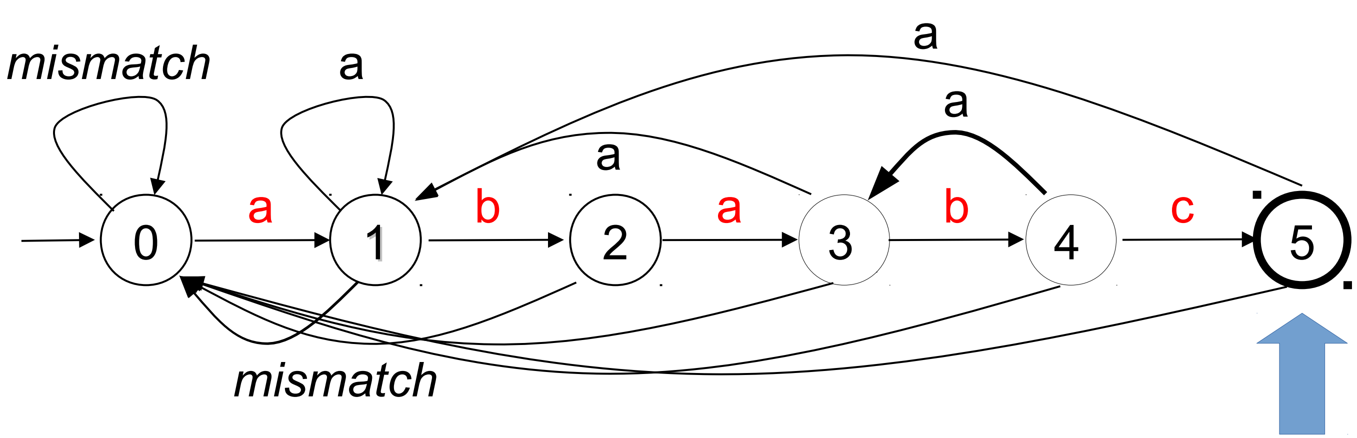
→ why?
→ because aba is the **longest suffix** of ababa, that is a **prefix** of ababa

2. Automaton Method

Given Pattern P, Text T, find all occurrences of P in T.

P =	a	b	a	b	c

T =	a	b	a	b	a	a	b	a	b	c	a	b	a	b	a	b	a	a	b	c



2. Automaton Method

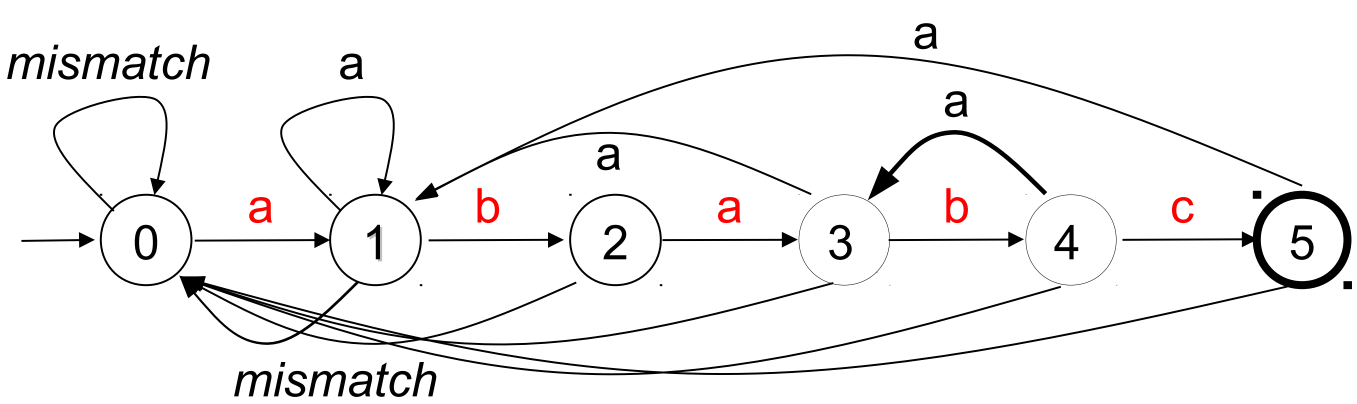
Given Pattern P, Text T, find all occurrences of P in T.

P =

a	b	a	b	c

T =

a	b	a	b	a	a	b	a	b	c	a	b	a	b	a	b	a	a	b	c



→ Deterministic Finite Automaton

→ O(|P||**S**|) size, where **S** = alphabet

2. Automaton Method

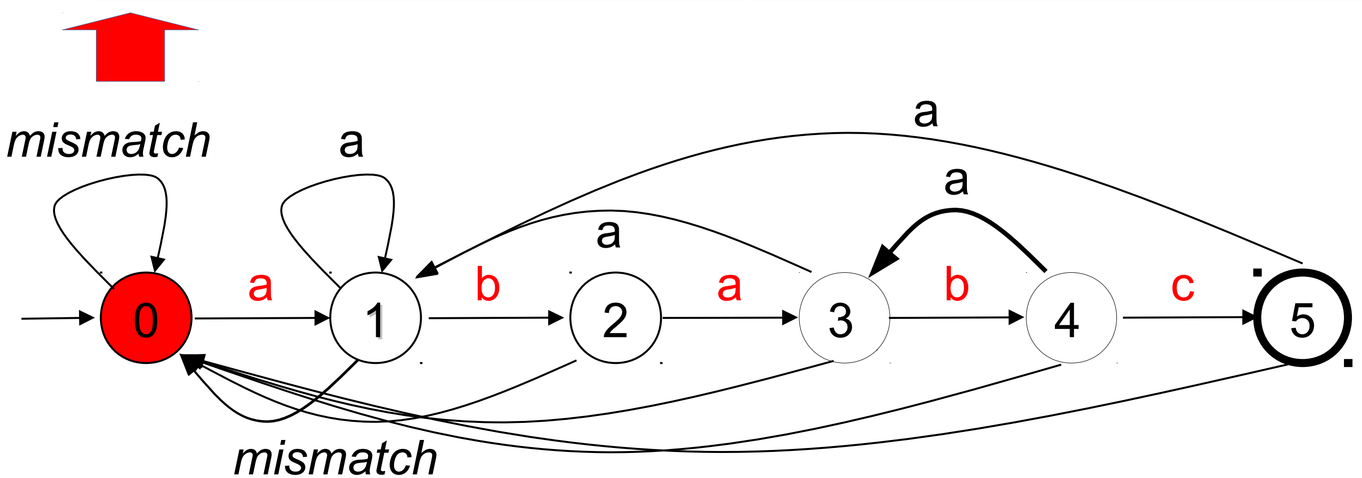
Given Pattern P, Text T, find all occurrences of P in T.

P =

a	b	a	b	c

T =

a	b	a	b	a	a	b	a	b	c	a	b	a	b	a	b	a	a	b	c



→ Deterministic Finite Automaton

→ O(|P||**S**|) size, where **S** = alphabet

→ simply run it in O(|T|) time to determine all occurrences of P in T

2. Automaton Method

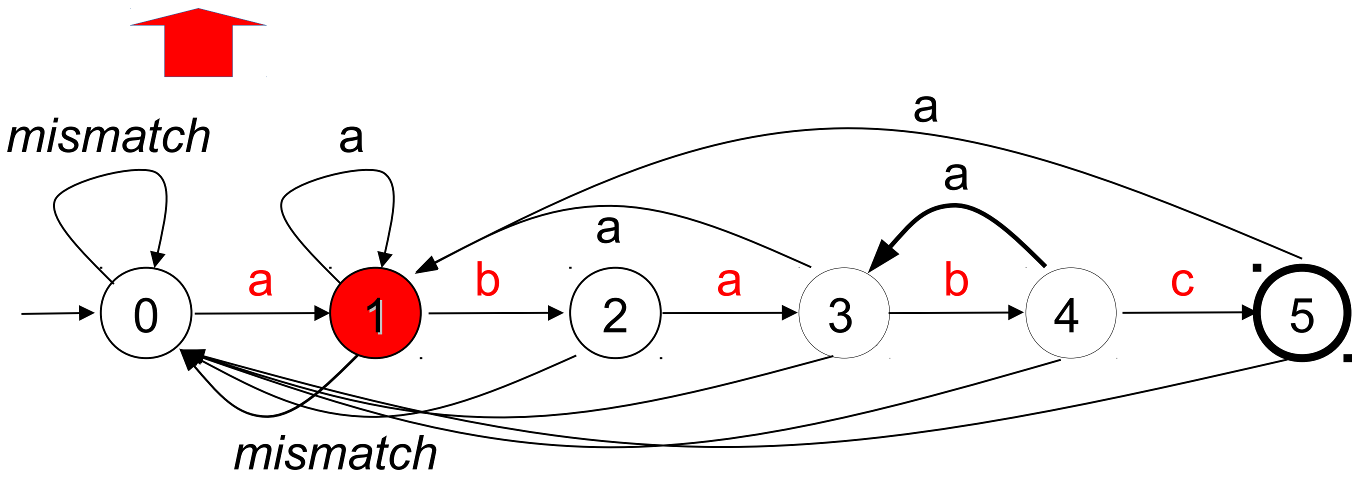
Given Pattern P, Text T, find all occurrences of P in T.

P =

a	b	a	b	c

T =

a	b	a	b	a	a	b	a	b	c	a	b	a	b	a	b	a	a	b	c



2. Automaton Method

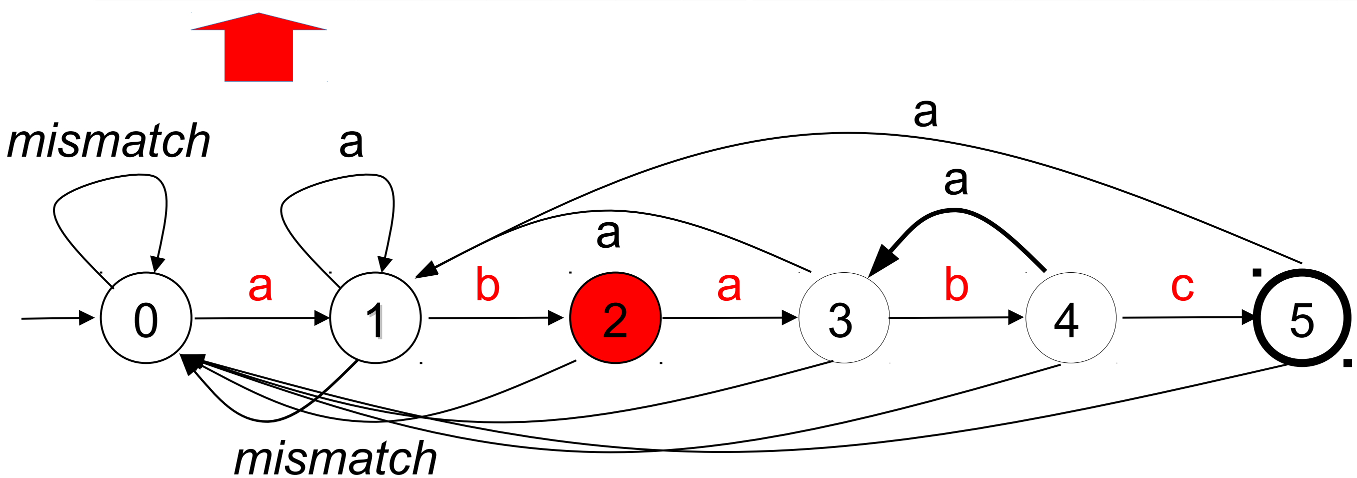
Given Pattern P, Text T, find all occurrences of P in T.

P =

a	b	a	b	c

T =

a	b	a	b	a	a	b	a	b	c	a	b	a	b	a	b	a	a	b	c



2. Automaton Method

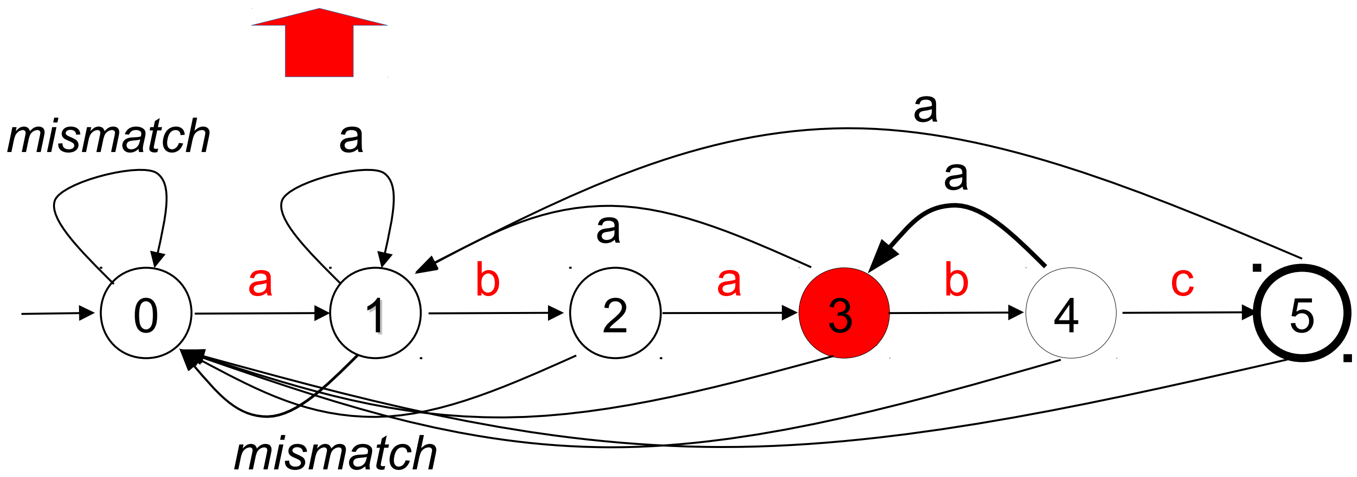
Given Pattern P, Text T, find all occurrences of P in T.

P =

a	b	a	b	c

T =

a	b	a	b	a	a	b	a	b	c	a	b	a	b	a	b	a	a	b	c



2. Automaton Method

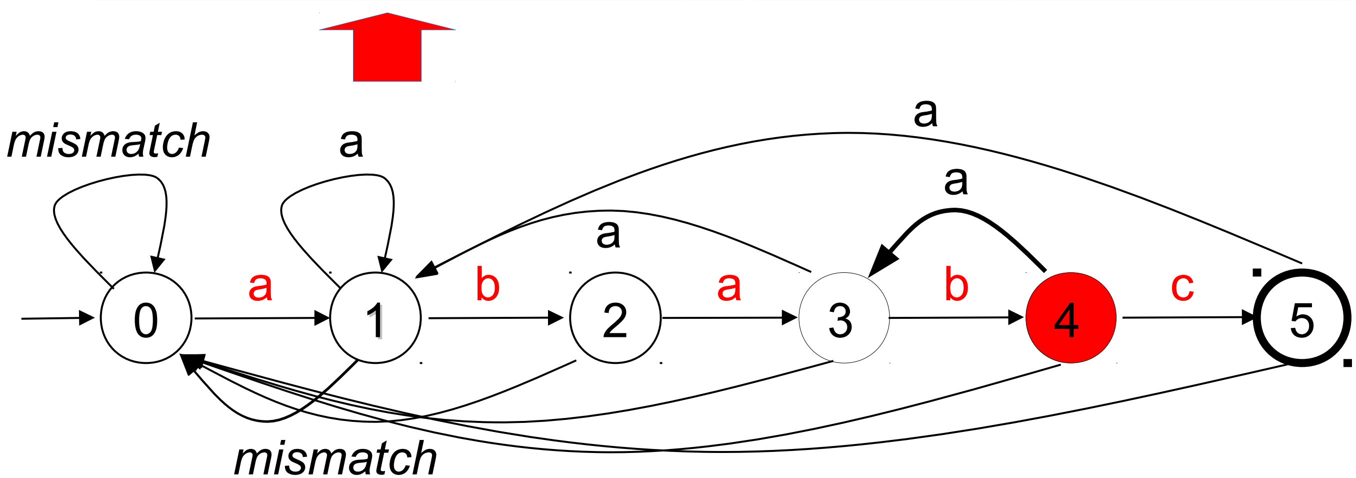
Given Pattern P, Text T, find all occurrences of P in T.

P =

a	b	a	b	c

T =

a	b	a	b	a	a	b	a	b	c	a	b	a	b	a	b	a	a	b	c

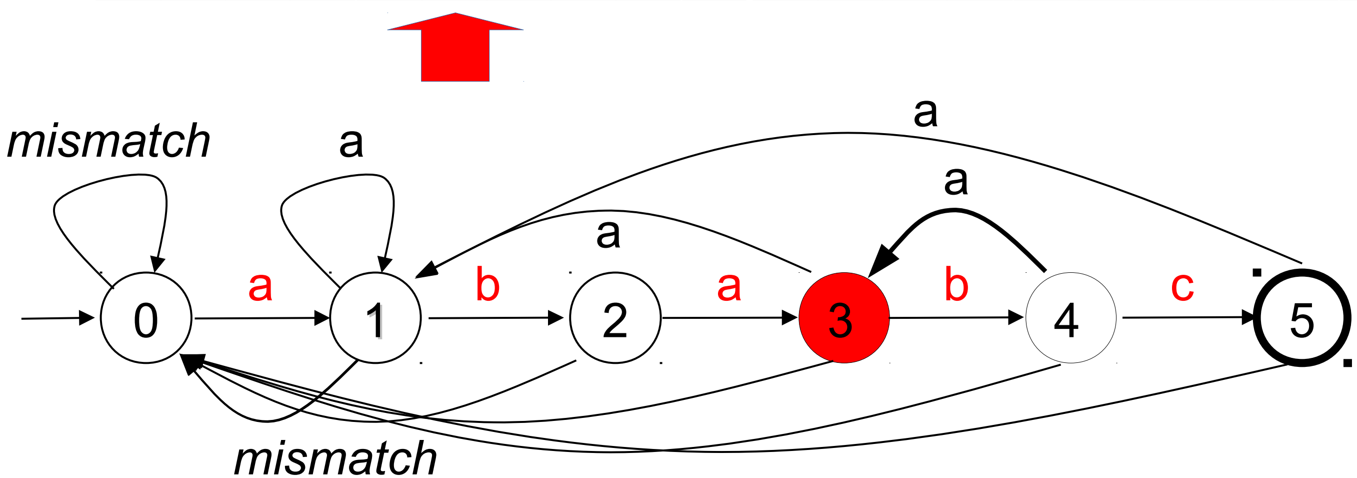


2. Automaton Method

Given Pattern P, Text T, find all occurrences of P in T.

P = | a | b | a | b | c |

T = | a | b | a | b | a | a | b | a | b | c | a | b | a | b | a | b | a | a | b | c |



2. Automaton Method

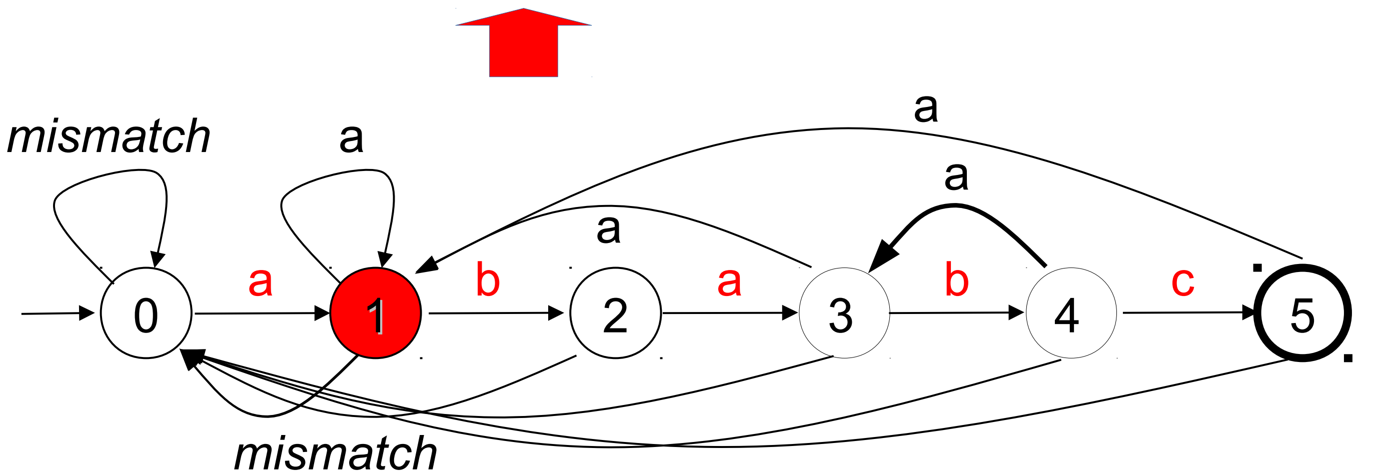
Given Pattern P, Text T, find all occurrences of P in T.

P =

a	b	a	b	c

T =

a	b	a	b	a	a	b	a	b	c	a	b	a	b	a	b	a	a	b	c



2. Automaton Method

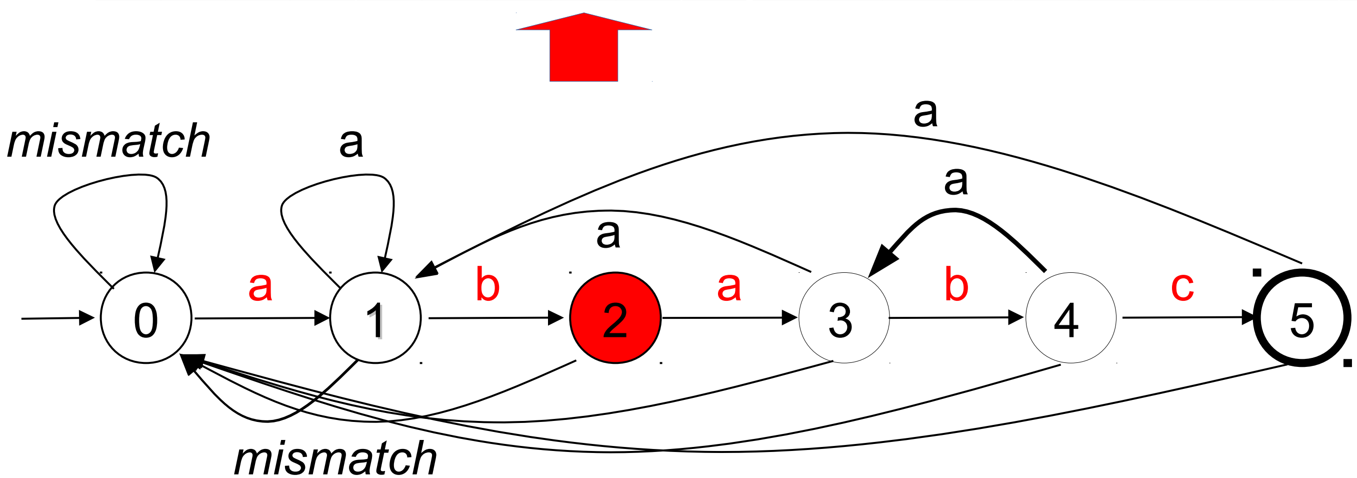
Given Pattern P, Text T, find all occurrences of P in T.

P =

a	b	a	b	c

T =

a	b	a	b	a	a	b	a	b	c	a	b	a	b	a	b	a	a	b	c



2. Automaton Method

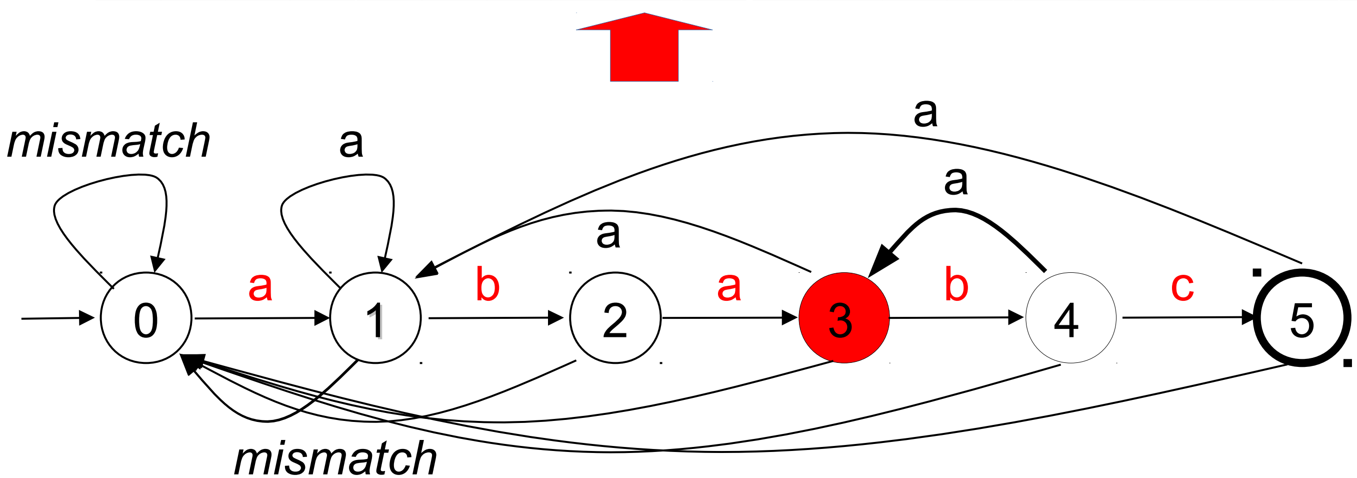
Given Pattern P, Text T, find all occurrences of P in T.

P =

a	b	a	b	c

T =

a	b	a	b	a	a	b	a	b	c	a	b	a	b	a	b	a	a	b	c



2. Automaton Method

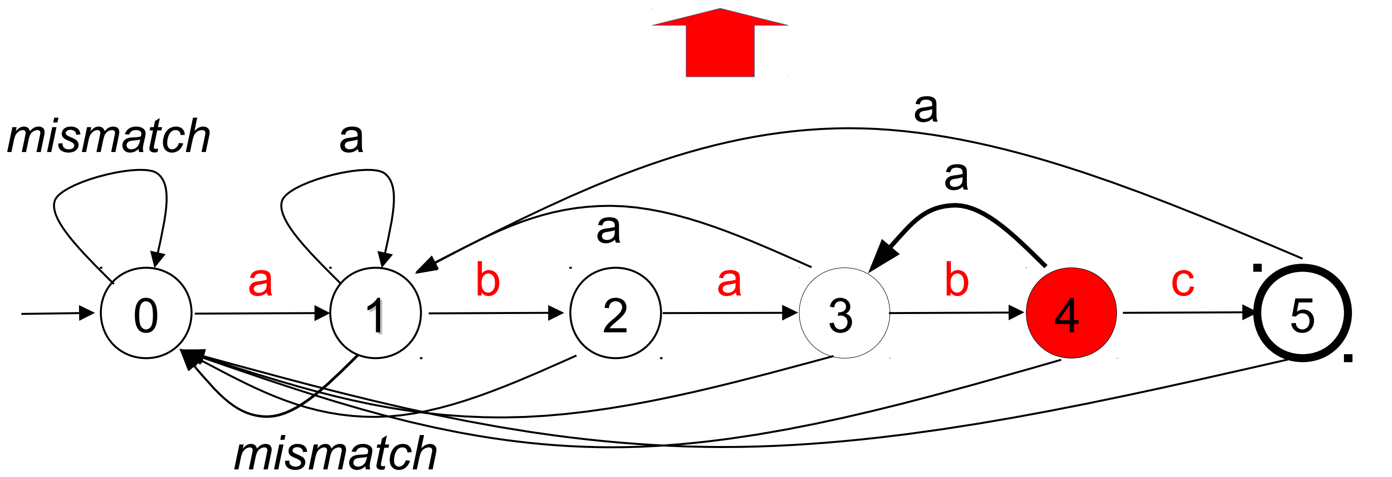
Given Pattern P, Text T, find all occurrences of P in T.

P =

a	b	a	b	c

T =

a	b	a	b	a	a	b	a	b	c	a	b	a	b	a	b	a	a	b	c



2. Automaton Method

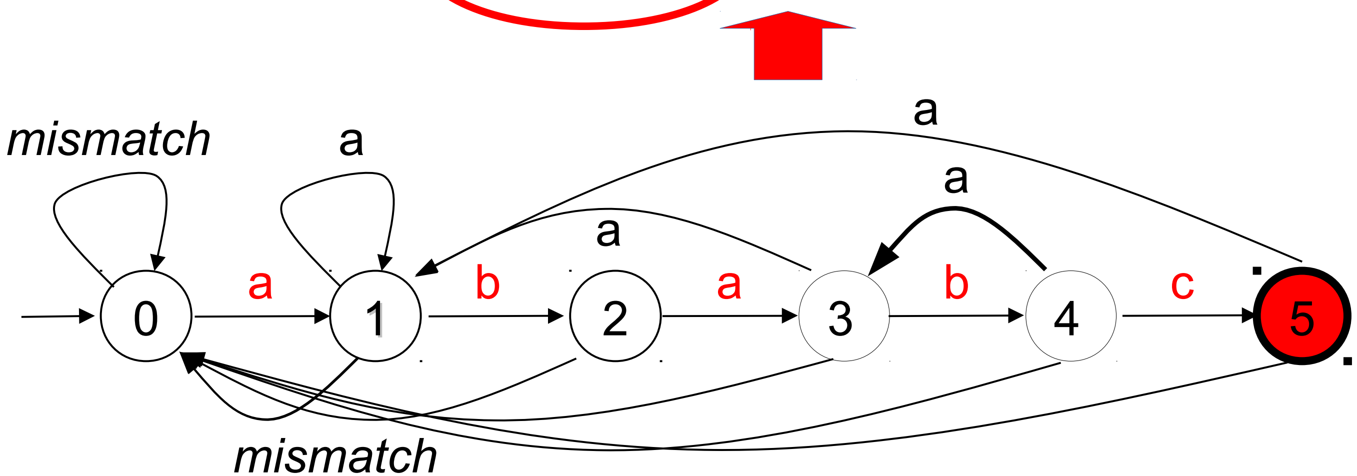
Given Pattern P, Text T, find all occurrences of P in T.

P =

a	b	a	b	c

T =

a	b	a	b	a	a	b	a	b	c	a	b	a	b	a	b	a	a	b	c



→ Match!

2. Automaton Method

Given Pattern P, how to build the automaton?

P =

a	b	a	b	c
1	2	3	4	5

→ for state k and symbol x, how to build transition d(k,x)?

→ length of the **longest proper suffix** of P[1] … P[x]x that is **prefix** of P

E.g. d(4, a) = ?

2. Automaton Method

Given Pattern P, how to build the automaton?

P =

a	b	a	b	c
1	2	3	4	5

→ for state k and symbol x, how to build transition d(k,x)?

→ length of the **longest proper suffix** of P[1] … P[x]x that is **prefix** of P

E.g. d(4, a) = ?

P[1]...P[4]a =

ababa

proper suffix

2. Automaton Method

Given Pattern P, how to build the automaton?

P =

a	b	a	b	c
1	2	3	4	5

→ for state k and symbol x, how to build transition d(k,x)?

→ length of the **longest proper suffix** of P[1] … P[x]x that is **prefix** of P

E.g. d(4, a) = ?

P[1]...P[4]a =

ababa

is also prefix!

proper suffix

2. Automaton Method

Given Pattern P, how to build the automaton?

P =

a	b	a	b	c
1	2	3	4	5

→ for state k and symbol x, how to build transition d(k,x)?

→ length of the **longest proper suffix** of P[1] … P[x]x that is **prefix** of P

E.g. d(4, a) = **3** = length(aba)

P[1]...P[4]a =

ababa

is also prefix!

proper suffix

Lopopre(u, v) =
longest proper suffix of u
that is prefix of v

Drawback of Automaton Method

→ matching time: O(n) $\qquad$ n = |T|
nice! $\qquad$ m = |P|

→ preprocessing time: O(m * |S|)
can be O(m * m)

not so nice... (for large patterns)

→ **Ideally** would like to have

– O(n) matching time or O(n + m)
– O(m) preprocessing time

END
Lecture 12