Applied Databases

Lecture 20
Recap II

Sebastian Maneth

University of Edinburgh - March 30th, 2017

Recap II

1. Schemas, Normal Forms, SQL
2. TFIDF-ranking, string matching (KMP, automata, Boyer-Moore)

2. Relational DBs

1) explain, using examples, what a functional dependency (fd) is, and what a fd-redundancy is.

2) explain BCNF and how it removes fd-redundancies.

3) are there any “harmful” side-effects when transforming a table to BCNF?

1) explain, using examples, what a functional dependency (fd) is, and what a fd-redundancy is.

Let S and T be non-empty sets of attributes (column names).

A table R has a functional dependency from S to T, if R's projection to S union T gives a function from S to T.

Such a function implies that for every S-tuple, there is at most one T-tuple in R.

1) explain, using examples, what a functional dependency (fd) is, and what a fd-redundancy is.

Let S and T be non-empty sets of attributes (column names).

A table R has a functional dependency from S to T, if R's projection to S union T gives a function from S to T.

Such a function implies that for every S-tuple, there is at most one T-tuple in R.

X	A
1	2
2	2

Functional dependencies? (“closed world assumption”)

1) explain, using examples, what a functional dependency (fd) is,
and what a fd-redundancy is.

Let S and T be non-empty sets of attributes (column names).

A table R has a functional dependency from S to T,
if R's projection to S union T gives a function from S to T.

Such a function implies that for every S-tuple, there is at most one T-tuple in R.

X	A
1	2
2	2

Functional dependencies? ("closed world assumption")

$X \rightarrow X$
$X \rightarrow A$
$X \rightarrow XA$
$A \rightarrow A$
$\cancel{A \rightarrow X}$
$\cancel{A \rightarrow XA}$

$XA \rightarrow X$
$XA \rightarrow A$
$XA \rightarrow XA$

1) explain, using examples, what a functional dependency (fd) is, and what a fd-redundancy is.

Let S and T be non-empty sets of attributes (column names).

A table R has a functional dependency from S to T, if R's projection to S union T gives a function from S to T.

Such a function implies that for every S-tuple, there is at most one T-tuple in R.

X	A
1	2
2	2

Functional dependencies? ("closed world assumption")

$X \to X$
$X \to A$
$X \to XA$
$A \to A$
$\cancel{A \to X}$
$\cancel{A \to XA}$

$XA \to X$
$XA \to A$
$XA \to XA$

$\boxed{X \to A}$

FD's with disjoint S, T

1) explain, using examples, what a functional dependency (fd) is, and what a fd-redundancy is.

Let S and T be non-empty sets of attributes (column names).
Functional dependency from S to T:
for every S-tuple, there is at most one T-tuple in R.

X	A	Z
1	2	5
1	2	6
1	2	7

← how many functional dependencies?

1) explain, using examples, what a functional dependency (fd) is, and what a fd-redundancy is.

Let S and T be non-empty sets of attributes (column names).
Functional dependency from S to T:
for every S-tuple, there is at most one T-tuple in R.

X	A	Z
1	2	5
1	2	6
1	2	7

← how many functional dependencies?

at most:
→ (2^3 – 1) * (2^3 – 1) = 7*7 = 49

Which ones are excluded?

A → Z, A → XZ, A → XA, A → XAZ
X → Z, X → AZ, X → XA, X → XAZ
XA → Z, XA → AZ, XA → xZ, XA → XAZ

1) explain, using examples, what a functional dependency (fd) is, and what a fd-redundancy is.

Let S and T be non-empty sets of attributes (column names).
Functional dependency from S to T:
for every S-tuple, there is at most one T-tuple in R.

A table R has fd-redundancy w.r.t. $S \rightarrow T$,
if R contains two distinct tuples with equal (S,T)-values.

1) explain, using examples, what a functional dependency (fd) is, and what a fd-redundancy is.

Let S and T be non-empty sets of attributes (column names).
Functional dependency from S to T:
for every S-tuple, there is at most one T-tuple in R.

A table R has fd-redundancy w.r.t. $S \rightarrow T$,
if R contains two distinct tuples with equal (S,T)-values.

X	A	Z
1	2	5
1	2	6

← are there fd-redundancies?

1) explain, using examples, what a functional dependency (fd) is, and what a fd-redundancy is.

Let S and T be non-empty sets of attributes (column names).
Functional dependency from S to T:
for every S-tuple, there is at most one T-tuple in R.

A table R has fd-redundancy w.r.t. $S \rightarrow T$,
if R contains two distinct tuples with equal (S,T)-values.

X	A	Z
1	2	5
1	2	6

← are there fd-redundancies?

Yes: 1) fd-redundancy wrt **X** → **A**
2) fd-redundancy wrt **A** → **X**

2) explain BCNF and how it removes fd-redundancies.

BCNF = if $S \rightarrow T$ is a functional dependency of R, then S is a superkey.

(assuming S disjoint T)

2) explain BCNF and how it removes fd-redundancies.

BCNF = if $S \rightarrow T$ is a functional dependency of R, then S is a superkey.

(assuming S disjoint T)

X	A
1	2
2	2

← in BCNF?

2) explain BCNF and how it removes fd-redundancies.

BCNF = if $S \rightarrow T$ is a functional dependency of R, then S is a superkey.

(assuming S disjoint T)

X	A
1	2
2	2

← in BCNF?

Yes: **X** is superkey, and

X → **A** is the only functional dependency.

2) explain BCNF and how it removes fd-redundancies.

BCNF = if $S \rightarrow T$ is a functional dependency of R, then S is a superkey.

(assuming S disjoint T)

X	A	Z
1	2	5
1	2	6

← in BCNF?

2) explain BCNF and how it removes fd-redundancies.

BCNF = if $S \rightarrow T$ is a functional dependency of R, then S is a superkey.

(assuming S disjoint T)

X	A	Z
1	2	5
1	2	6

← in BCNF?

No: **X** → **A** is fd, but **X** is not a superkey

A → **X** is fd, but **A** is not a superkey

2) explain BCNF and how it removes fd-redundancies.

BCNF = if $S \rightarrow T$ is a functional dependency of R, then S is a superkey.

(assuming S disjoint T)

X	A	Z
1	2	5
1	2	6

X	A
1	2
1	2

X	Z
1	5
1	6

2) explain BCNF and how it removes fd-redundancies.

BCNF = if $S \rightarrow T$ is a functional dependency of R, then S is a superkey.

(assuming S disjoint T)

X	A	Z
1	2	5
1	2	6

$\Rightarrow$

X	A
1	2
1	2

X	Z
1	5
1	6

In BCNF, there can be **no** fd-redundancies.

Why?

2) explain BCNF and how it removes fd-redundancies.

BCNF = if $S \rightarrow T$ is a functional dependency of R, then S is a superkey.

(assuming S disjoint T)

X	A	Z
1	2	5
1	2	6

X	A
1	2
1	2

X	Z
1	5
1	6

In BCNF, there can be **no** fd-redundancies.

Why?

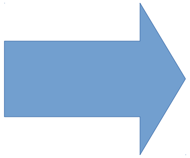
Would imply that a tuple exists twice in R with same superkey-values

3) are there any “harmful” side-effects when transforming a table to BCNF?

X	A	Z
1	2	5
1	2	6
2	2	6

3) are there any "harmful" side-effects when transforming a table to BCNF?

X	A	Z
1	2	5
1	2	6
2	2	6



X	A
1	2
2	2

X	Z
1	5
1	6
2	6

3) are there any “harmful” side-effects when transforming a table to BCNF?

X	A	Z
1	2	5
1	2	6
2	2	6

$\Rightarrow$

X	A
1	2
2	2

X	Z
1	5
1	6
2	6

dependency **XZ** → **A** is lost

SQL

a	b	c
1	2	5
1	3	6
2	2	6

List the result tuples for each of these SQL queries:

1) `SELECT * FROM R where b>a;`
2) `SELECT COUNT(*) FROM R r1, R r2 where r1.b>r2.a;`
3) `SELECT COUNT(*) FROM R r1, R r2 where r1.b>r1.a;`
4) `SELECT a FROM R UNION SELECT a FROM R;`
5) `SELECT * FROM R r1, R r2 where r1.b=r2.a;`

SQL

a	b	c
1	2	5
1	3	6
2	2	6

List the result tuples for each of these SQL queries:

1) `SELECT * FROM R where b>a;`
2) `SELECT COUNT(*) FROM R r1, R r2 where r1.b>r2.a;`
3) `SELECT COUNT(*) FROM R r1, R r2 where r1.b>r1.a;`
4) `SELECT a FROM R UNION SELECT a FROM R;`
5) `SELECT * FROM R r1, R r2 where r1.b=r2.a;`

---

1)

SQL

a	b	c
1	2	5
1	3	6
2	2	6

List the result tuples for each of these SQL queries:

1) `SELECT * FROM R where b>a;`
2) `SELECT COUNT(*) FROM R r1, R r2 where r1.b>r2.a;`
3) `SELECT COUNT(*) FROM R r1, R r2 where r1.b>r1.a;`
4) `SELECT a FROM R UNION SELECT a FROM R;`
5) `SELECT * FROM R r1, R r2 where r1.b=r2.a;`

---

1) (1,2,5) and (1,3,5)

SQL

a	b	c
1	2	5
1	3	6
2	2	6

List the result tuples for each of these SQL queries:

1) `SELECT * FROM R where b>a;`
2) `SELECT COUNT(*) FROM R r1, R r2 where r1.b>r2.a;`
3) `SELECT COUNT(*) FROM R r1, R r2 where r1.b>r1.a;`
4) `SELECT a FROM R UNION SELECT a FROM R;`
5) `SELECT * FROM R r1, R r2 where r1.b=r2.a;`

---

1) (1,2,5) and (1,3,5)

2)

SQL

a	b	c
1	2	5
1	3	6
2	2	6

List the result tuples for each of these SQL queries:

1) `SELECT * FROM R where b>a;`
2) `SELECT COUNT(*) FROM R r1, R r2 where r1.b>r2.a;`
3) `SELECT COUNT(*) FROM R r1, R r2 where r1.b>r1.a;`
4) `SELECT a FROM R UNION SELECT a FROM R;`
5) `SELECT * FROM R r1, R r2 where r1.b=r2.a;`

---

1) (1,2,5) and (1,3,5)

2) (7)

SQL

a	b	c
1	2	5
1	3	6
2	2	6

List the result tuples for each of these SQL queries:

```
1) SELECT * FROM R where b>a;
2) SELECT COUNT(*) FROM R r1, R r2 where r1.b>r2.a;
3) SELECT COUNT(*) FROM R r1, R r2 where r1.b>r1.a;
4) SELECT a FROM R UNION SELECT a FROM R;
5) SELECT * FROM R r1, R r2 where r1.b=r2.a;
```

---

1) (1,2,5) and (1,3,5)

2) (7)

3)

SQL

a	b	c
1	2	5
1	3	6
2	2	6

List the result tuples for each of these SQL queries:

```
1) SELECT * FROM R where b>a;
2) SELECT COUNT(*) FROM R r1, R r2 where r1.b>r2.a;
3) SELECT COUNT(*) FROM R r1, R r2 where r1.b>r1.a;
4) SELECT a FROM R UNION SELECT a FROM R;
5) SELECT * FROM R r1, R r2 where r1.b=r2.a;
```

---

1) (1,2,5) and (1,3,5)

2) (7)

3) (6)

SQL

a	b	c
1	2	5
1	3	6
2	2	6

List the result tuples for each of these SQL queries:

```
1) SELECT * FROM R where b>a;
2) SELECT COUNT(*) FROM R r1, R r2 where r1.b>r2.a;
3) SELECT COUNT(*) FROM R r1, R r2 where r1.b>r1.a;
4) SELECT a FROM R UNION SELECT a FROM R;
5) SELECT * FROM R r1, R r2 where r1.b=r2.a;
```

---

1) (1,2,5) and (1,3,5)

2) (7)

3) (6)

4)

SQL

a	b	c
1	2	5
1	3	6
2	2	6

List the result tuples for each of these SQL queries:

```
1) SELECT * FROM R where b>a;
2) SELECT COUNT(*) FROM R r1, R r2 where r1.b>r2.a;
3) SELECT COUNT(*) FROM R r1, R r2 where r1.b>r1.a;
4) SELECT a FROM R UNION SELECT a FROM R;
5) SELECT * FROM R r1, R r2 where r1.b=r2.a;
```

---

1) (1,2,5) and (1,3,5)

2) (7)

3) (6)

4) (1) and (2)

SQL

a	b	c
1	2	5
1	3	6
2	2	6

List the result tuples for each of these SQL queries:

```
1) SELECT * FROM R where b>a;
2) SELECT COUNT(*) FROM R r1, R r2 where r1.b>r2.a;
3) SELECT COUNT(*) FROM R r1, R r2 where r1.b>r1.a;
4) SELECT a FROM R UNION SELECT a FROM R;
5) SELECT * FROM R r1, R r2 where r1.b=r2.a;
```

---

1) (1,2,5) and (1,3,5)

2) (7)

3) (6)

4) (1) and (2)

5)

SQL

a	b	c
1	2	5
1	3	6
2	2	6

List the result tuples for each of these SQL queries:

1) `SELECT * FROM R where b>a;`
2) `SELECT COUNT(*) FROM R r1, R r2 where r1.b>r2.a;`
3) `SELECT COUNT(*) FROM R r1, R r2 where r1.b>r1.a;`
4) `SELECT a FROM R UNION SELECT a FROM R;`
5) `SELECT * FROM R r1, R r2 where r1.b=r2.a;`

---

1) (1,2,5) and (1,3,5)

2) (7)

3) (6)

4) (1) and (2)

5) (1,2,5,2,2,6) and (2,2,6,2,2,6)

SQL

a	b	c
1	2	5
1	3	6
2	2	6

List the result tuples for each of these SQL queries:

1) `SELECT * FROM R where b>a;`
2) `SELECT COUNT(*) FROM R r1, R r2 where r1.b>r2.a;`
3) `SELECT COUNT(*) FROM R r1, R r2 where r1.b>r1.a;`
4) `SELECT a FROM R UNION SELECT a FROM R;`
5) `SELECT * FROM R r1, R r2 where r1.b=r2.a;`

Give SQL queries for

(a) all values (with duplicates) in the entire table R

(b) all distinct values in the entire table R, with their frequencies

(c) all distinct b-values in R, that are smaller than the average
over all values (with duplicates) in the entire R.

SQL

a	b	c
1	2	5
1	3	6
2	2	6

List the result tuples for each of these SQL queries:

1) `SELECT * FROM R where b>a;`
2) `SELECT COUNT(*) FROM R r1, R r2 where r1.b>r2.a;`
3) `SELECT COUNT(*) FROM R r1, R r2 where r1.b>r1.a;`
4) `SELECT a FROM R UNION SELECT a FROM R;`
5) `SELECT * FROM R r1, R r2 where r1.b=r2.a;`

Give SQL query for

(a) all values (with duplicates) in the entire table R

SQL

a	b	c
1	2	5
1	3	6
2	2	6

List the result tuples for each of these SQL queries:

1) `SELECT * FROM R where b>a;`
2) `SELECT COUNT(*) FROM R r1, R r2 where r1.b>r2.a;`
3) `SELECT COUNT(*) FROM R r1, R r2 where r1.b>r1.a;`
4) `SELECT a FROM R UNION SELECT a FROM R;`
5) `SELECT * FROM R r1, R r2 where r1.b=r2.a;`

Give SQL query for

(a) all values (with duplicates) in the entire table R

→
```
SELECT a FROM R UNION ALL
SELECT b FROM R UNION ALL
SELECT c FROM R;
```

SQL

a	b	c
1	2	5
1	3	6
2	2	6

List the result tuples for each of these SQL queries:

```
1) SELECT * FROM R where b>a;
2) SELECT COUNT(*) FROM R r1, R r2 where r1.b>r2.a;
3) SELECT COUNT(*) FROM R r1, R r2 where r1.b>r1.a;
4) SELECT a FROM R UNION SELECT a FROM R;
5) SELECT * FROM R r1, R r2 where r1.b=r2.a;
```

Give SQL query for

(b) all distinct values in the entire table R, with their frequencies

SQL

a	b	c
1	2	5
1	3	6
2	2	6

List the result tuples for each of these SQL queries:

1) `SELECT * FROM R where b>a;`
2) `SELECT COUNT(*) FROM R r1, R r2 where r1.b>r2.a;`
3) `SELECT COUNT(*) FROM R r1, R r2 where r1.b>r1.a;`
4) `SELECT a FROM R UNION SELECT a FROM R;`
5) `SELECT * FROM R r1, R r2 where r1.b=r2.a;`

Give SQL query for

(b) all distinct values in the entire table R, with their frequencies

→
```
SELECT a,COUNT(a) FROM
(SELECT a FROM R UNION ALL
 SELECT b FROM R UNION ALL
 SELECT c FROM R) z
GROUP BY a;
```

SQL

a	b	c
1	2	5
1	3	6
2	2	6

List the result tuples for each of these SQL SQL queries:

1) `SELECT * FROM R where b>a;`
2) `SELECT COUNT(*) FROM R r1, R r2 where r1.b>r2.a;`
3) `SELECT COUNT(*) FROM R r1, R r2 where r1.b>r1.a;`
4) `SELECT a FROM R UNION SELECT a FROM R;`
5) `SELECT * FROM R r1, R r2 where r1.b=r2.a;`

Give SQL query for

(c) all distinct b-values in R, that are smaller than the average over all values (with duplicates) in the entire R.

SQL

a	b	c
1	2	5
1	3	6
2	2	6

List the result tuples for each of these SQL SQL queries:

1) `SELECT * FROM R where b>a;`
2) `SELECT COUNT(*) FROM R r1, R r2 where r1.b>r2.a;`
3) `SELECT COUNT(*) FROM R r1, R r2 where r1.b>r1.a;`
4) `SELECT a FROM R UNION SELECT a FROM R;`
5) `SELECT * FROM R r1, R r2 where r1.b=r2.a;`

Give SQL query for

(c) all distinct b-values in R, that are smaller than the average over all values (with duplicates) in the entire R.

→
```
SELECT DISTINCT b FROM R WHERE b< (SELECT AVG(a) FROM
                  (SELECT a FROM R UNION ALL
                   SELECT b FROM R UNION ALL
                   SELECT c FROM R) z);
```

TFIDF Ranking

TFIDF Ranking

1	The old night keeper keeps the keep in the town
2	In the big old house in the big old gown.
3	The house in the town had the big old keep
4	Where the old night keeper never did sleep.
5	The night keeper keeps the keep in the night
6	And keeps in the dark and sleeps in the light.

Assume casefolding and stemming. We only care about these words:

big, house, keep, night, old

1) make a table of term frequencies of these words (rows=words, columns=docs)
2) normalize by dividing column-wise by maximum
3) compute IDF of each word w as log_10(N/df_w)
4) multiply normalized term frequencies by IDF, to obtain TFIDF table.
5) compute cosine similarity between doc-2 and “big old house”

1	The old night keeper keeps the keep in the town
2	In the big old house in the big old gown.
3	The house in the town had the big old keep
4	Where the old night keeper never did sleep.
5	The night keeper keeps the keep in the night
6	And keeps in the dark and sleeps in the light.

Assume casefolding and stemming. We only care about these words:

big, house, keep, night, old

1) make a table of term frequencies of these words (rows=words, columns=docs)

	1	2	3	4	5	6
big		2	1			
house		1	1			
keep	3		1	1	3	1
night	1			1	2	
old	1	2	1	1		

1 The old night keeper keeps the keep in the town
2 In the big old house in the big old gown.
3 The house in the town had the big old keep
4 Where the old night keeper never did sleep.
5 The night keeper keeps the keep in the night
6 And keeps in the dark and sleeps in the light.

2) normalize by dividing column-wise by maximum

	1	2	3	4	5	6
big		1	1			
house		1/2	1			
keep	1		1	1	1	1
night	1/3			1	2/3	
old	1/3	1	1	1		

1	The old night keeper keeps the keep in the town
2	In the big old house in the big old gown.
3	The house in the town had the big old keep
4	Where the old night keeper never did sleep.
5	The night keeper keeps the keep in the night
6	And keeps in the dark and sleeps in the light.

3) compute IDF of each word w as log_10(N/df_w)

	1	2	3	4	5	6	IDF
big		1	1				log(6/2)=.477
house		1/2	1				.477
keep	1		1	1	1	1	log(6/5)=.079
night	1/3			1	2/3		log(6/3)=.301
old	1/3	1	1	1			log(6/4)=.176

1	The old night keeper keeps the keep in the town
2	In the big old house in the big old gown.
3	The house in the town had the big old keep
4	Where the old night keeper never did sleep.
5	The night keeper keeps the keep in the night
6	And keeps in the dark and sleeps in the light.

4) multiply normalized term frequencies by IDF, to obtain TFIDF table.

	1	2	3	4	5	6	IDF
big		.477	.477				.477
house		.239	.477				.477
keep	.079		.079	.079	.079	.079	.079
night	.100			.301	.201		.301
old	.059	.176	.176	.176			.176

1	The old night keeper keeps the keep in the town
2	In the big old house in the big old gown.
3	The house in the town had the big old keep
4	Where the old night keeper never did sleep.
5	The night keeper keeps the keep in the night
6	And keeps in the dark and sleeps in the light.

5) compute cosine similarity between doc-2 and "big old house"

	1	2	3	4	5	6	IDF	
big		.477	.477				.477	.477
house		.239	.477				.477	.477
keep	.079		.079	.079	.079	.079	.079	
night	.100			.301	.201		.301	
old	.059	.176	.176	.176			.176	.176

```
1    The old night keeper keeps the keep in the town
2    In the big old house in the big old gown.
3    The house in the town had the big old keep
4    Where the old night keeper never did sleep.
5    The night keeper keeps the keep in the night
6    And keeps in the dark and sleeps in the light.
```

5) compute cosine similarity between doc-2 and "big old house"

	1	2	3	4	5	6	IDF	
big		.477	.477				.477	.477
house		.239	.477				.477	.477
keep	.079		.079	.079	.079	.079	.079	
night	.100			.301	.201		.301	
old	.059	.176	.176	.176			.176	.176

cos-sim(Q,d2) = (.477*.477 + .239*.477 + .176*.176) /
(sqrt(.477^2 +.239^2 + .176^2)*sqrt(.477^2 + .477^2 + .176^2))
= .3725 / (0.5618 * 0.6972) = **0.9510**

String Matching

1) explain the difference between the Matching Automaton and KMP.
2) draw the Matching Automaton for the string abaaba
3) give the KMP table for abaaba
4) how many comparisons does Horspool need for this pattern
   on the string aababaaba?

String Matching

1) explain the difference between the Matching Automaton and KMP.
2) draw the Matching Automaton for the string abaaba
3) give the KMP table for abaaba
4) how many comparisons does Horspool need for this pattern on the string aababaaba?

1) → size of matching automaton is |P||S| which can be |P|^2 (S = alphabet)

→ KMP table has only |P|-many entries.

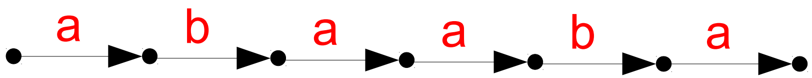
→ automaton uses one look-up per text-symbol, i.e., O(|T|) matching time

→ KMP may require several look-ups per text-symbol (at most (log |P|)–many)

String Matching

1) explain the difference between the Matching Automaton and KMP.
2) draw the Matching Automaton for the string abaaba
3) give the KMP table for abaaba
4) how many comparisons does Horspool need for this pattern on the string aababaaba?

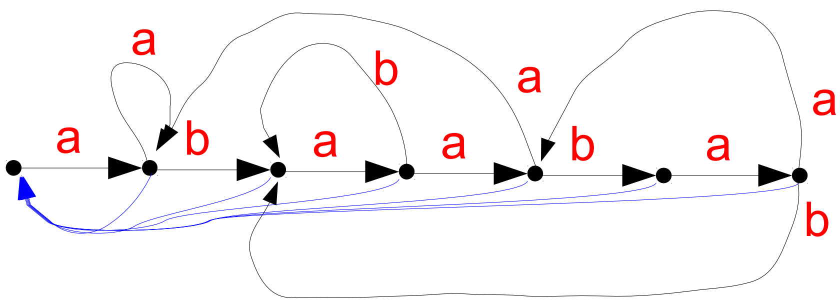
2)



String Matching

1) explain the difference between the Matching Automaton and KMP.
2) draw the Matching Automaton for the string abaaba
3) give the KMP table for abaaba
4) how many comparisons does Horspool need for this pattern on the string aababaaba?

2)



→ blue edges w.o. label means “else” = “any other letter”

String Matching

1) explain the difference between the Matching Automaton and KMP.
2) draw the Maatching automaton for the string abaaba
3) give the KMP table for abaaba
4) how many comparisons does Horspool need for this pattern on the string aababaaba?

3) KMP table = longest prefix that is proper suffix (up to current character)
and such that the next letter (if exists) is *different*
("–1" if such a prefix not exist)

a	b	a	a	b	a

String Matching

1) explain the difference between the Matching Automaton and KMP.
2) draw the Maatching automaton for the string abaaba
3) give the KMP table for abaaba
4) how many comparisons does Horspool need for this pattern on the string aababaaba?

3) KMP table = longest prefix that is proper suffix (up to current character)
and such that the next letter (if exists) is *different*
("–1" if such a prefix not exist)

a	b	a	a	b	a
0					

String Matching

1) explain the difference between the Matching Automaton and KMP.
2) draw the Maatching automaton for the string abaaba
3) give the KMP table for abaaba
4) how many comparisons does Horspool need for this pattern on the string aabababaaba?

3) KMP table = longest prefix that is proper suffix (up to current character)
   and such that the next letter (if exists) is *different*
   ("–1" if such a prefix not exist)

a	b	a	a	b	a
0	-1				

String Matching

1) explain the difference between the Matching Automaton and KMP.
2) draw the Maatching automaton for the string abaaba
3) give the KMP table for abaaba
4) how many comparisons does Horspool need for this pattern on the string aababaaba?

3) KMP table = longest prefix that is proper suffix (up to current character)
and such that the next letter (if exists) is *different*
("–1" if such a prefix not exist)

a	b	a	a	b	a
0	-1	1			

String Matching

1) explain the difference between the Matching Automaton and KMP.
2) draw the Maatching automaton for the string abaaba
3) give the KMP table for abaaba
4) how many comparisons does Horspool need for this pattern on the string aabababaaba?

3) KMP table = longest prefix that is proper suffix (up to current character)
   and such that the next letter (if exists) is *different*
   ("–1" if such a prefix not exist)

a	b	a	a	b	a
0	-1	1	0	-1	3

String Matching

1) explain the difference between the Matching Automaton and KMP.
2) draw the Matching automaton for the string abababa
3) give the KMP table for abaaba
4) how many comparisons does Horspool need for this pattern on the string aabababaaba?

4) If mismatch with P[m] aligned to z in T, shift RIGHT by R(z).

R(z) = distance from the right-most (non-last) "z" in P to the end of P (and |P| if there is no occurrence)

String Matching

1) explain the difference between the Matching Automaton and KMP.
2) draw the Matching automaton for the string abaaba
3) give the KMP table for abaaba
4) how many comparisons does Horspool need for this pattern on the string aababaaba?

4) If mismatch with P[m] aligned to z in T, shift RIGHT by R(z).

R(z) = distance from the right-most (non-last) "z" in P to the end of P (and |P| if there is no occurrence)

a	a	b	a	b	a	a	b	a
a	b	a	a	b	a			

#comparisons = 4

String Matching

1) explain the difference between the Matching Automaton and KMP.
2) draw the Matching automaton for the string abaaba
3) give the KMP table for abaaba
4) how many comparisons does Horspool need for this pattern on the string aababaaba?

4) If mismatch with P[m] aligned to z in T, shift RIGHT by R(z).

R(z) = distance from the right-most (non-last) “z” in P to the end of P
(and |P| if there is no occurrence)

a	a	b	a	b	a	a	b	a
a	b	a	a	b	a			
		a	b	a	a	b	a	

→ R(a) = 2
→ shift RIGHT by 2

#comparisons = 4 + 1

String Matching

1) explain the difference between the Matching Automaton and KMP.
2) draw the Matching automaton for the string abaaba
3) give the KMP table for abaaba
4) how many comparisons does Horspool need for this pattern on the string aababaaba?

4) If mismatch with P[m] aligned to z in T, shift RIGHT by R(z).

R(z) = distance from the right-most (non-last) “z” in P to the end of P (and |P| if there is no occurrence)

a	a	b	a	b	a	a	b	a
a	b	a	a	b	a			
		a	b	a	a	b	a	
			a	a	b	a	b	a

SHIFT by 1

Mismatch with “b” aligned to P[m].
→ shift by 1 = R(b)

#comparisons = 4 + 1 + 6 = **11**

END

Lecture 20

All the best with the exam!!
Remember: no lectures next week!