Algorithms for matching

Exercise session

Pierre Lison

(based on slides from Geert-Jan M. Kruijff)

⟨plison,gj@dfki.de⟩

The exercises for today

a. Show how to compute Z_i stepwise for i > 1 (using the notion of Z-boxes) for the following strings:

   i. AABCAABXAAZ

   ii. ABCDXABCYABDXY

b. Apply the Boyer-Moore algorithm to find occurrences of ABXYABXZ in XABXYABXYABXZABXZABXYABXZA

a.i.) Z_i for AABCAABXAAZ

S = AABCAABXAAZ

Step 0)

Compute $Z_2(S)$ by comparing left-to-right S[2..|S|] and S[1..|S|] until a mismatch is found; $Z_2(S)$ is the length of that string. If $Z_2(S) > 0$ then r=r2=$Z_2(S)$+1 and l=2, else l=r=0

S	A	A	B	C	A	A	B	X	A	A	Z
	1	2	3	4	5	6	7	8	9	10	11
$Z_i(S)$	--	1									

$Z_2(S)$=1: { *A* **A** B ... } so l=2, r=$Z_2(S)$+1=1+1=2

a.i.) Z_i for AABCAABXAAZ

Step 1)

k > r: 3 > (r=2) so find $Z_3(S)$ by comparing S[3...|S|] to S[1..|S|] until a mismatch is found; if $Z_3(S) > 0$ then l=3, $r=3+Z_3(S)-1$

S(3)='B' ≠ S(1)='A', hence $Z_3(S)=0$, l and r remain as they are: l=r=2

S	A	A	B	C	A	A	B	X	A	A	Z
	1	2	3	4	5	6	7	8	9	10	11
$Z_i(S)$	--	1	0								

$Z_3(S)=0$ so l=2, r=2

a.i.) Z_i for AABCAABXAAZ

Step 1)

k > r: 4 > (r=2) so find $Z_4(S)$ by comparing S[4...|S|] to S[1..|S|] until a mismatch is found; if $Z_4(S) > 0$ then l=4, $r=4+Z_4(S)-1$

S(4)='C' ≠ S(1)='A', hence $Z_4(S)=0$, l and r remain as they are: l=r=2

S	A	A	B	C	A	A	B	X	A	A	Z
	1	2	3	4	5	6	7	8	9	10	11
$Z_i(S)$	--	1	0	0							

$Z_4(S)=0$ so l=2, r=2

a.i.) Z_i for AABCAABXAAZ

Step 1)

k > r: 5 > (r=2) so find $Z_5(S)$ by comparing S[5...|S|] to S[1..|S|] until a mismatch is found; if $Z_5(S) > 0$ then l=5, r=5+$Z_5(S)$-1

S[5..7]="A A B" matches S[1..3]="A A B", hence $Z_5(S)$=3, and l and r are set as follows: l=5, r=5+$Z_5(S)$-1=5+3-1=7

S	A	A	B	C	A	A	B	X	A	A	Z
	1	2	3	4	5	6	7	8	9	10	11
$Z_i(S)$	--	1	0	0	3						

$Z_5(S)$=3 so l=5, r=7

a.i.) Z_i for AABCAABXAAZ

Step 2)

$6 \leq (r=7)$: position k=6 is contained in a Z-box (namely, ”AAB”=S[5..7], with S(6)=’A’).

Hence S(6) also appears in k’=k-l=6-5+1=2: S(6)=S(2)=’A’

Therefore, S[6..7] must match S[2..3], which it does

Furthermore, there must be a match to a prefix of S of length minimum [Z_2(S), |S[2..3]|], i.e. minimum [1,r-k+1=2] = 2

Step 2a)

Z_6(S)=Z_2(S)=1 which is smaller than the length of S[2..3], hence l and r stay the same

S	A	A	B	C	A	A	B	X	A	A	Z
	1	2	3	4	5	6	7	8	9	10	11
Z_i(S)	--	1	0	0	3	1					

Z_6(S)=Z_2(S)=1 so l and r remain the same: l=5, r=7

a.i.) Z_i for AABCAABXAAZ

Step 2)

7 ≤ (r=7): position k=7 is contained in S[5..7], with S(7)='B'.

Hence S(7) also appears in k'=k-l=7-5+1=3: S(7)=S(3)='B'

Therefore, S[7..7] must match S[3..3], i.e. S(7)=S(3), which it does

Furthermore, there must be a match to a prefix of S of length minimum [Z_3(S), |S[3..3]|], i.e. minimum [0,r-k+1=1] = 1

Step 2a)

Z_7(S)=Z_3(S)=0 which is smaller than the length of S[3..3], hence l and r stay the same

S	A	A	B	C	A	A	B	X	A	A	Z
	1	2	3	4	5	6	7	8	9	10	11
Z_i(S)	--	1	0	0	3	1	0				

Z_7(S)=Z_3(S)=0 so l and r remain the same: l=5, r=7

a.i.) Z_i for AABCAABXAAZ

k=8 > (r=7) so step 1:

match S[8..|S|] to S[1..|S|]: mismatch, so Z_8(S)=0, l and r remain the same

S	A	A	B	C	A	A	B	X	A	A	Z
	1	2	3	4	5	6	7	8	9	10	11
Z_i(S)	--	1	0	0	3	1	0	0			

Z_8(S)=0 so l=5, r=7

k=9 > (r=7) so step 1:

match S[9..|S|] to S[1..|S|]: match S[9..10]=S[1..2], so Z_9(S)=2, l=9 and r=10

S	A	A	B	C	A	A	B	X	A	A	Z
	1	2	3	4	5	6	7	8	9	10	11
Z_i(S)	--	1	0	0	3	1	0	0	2		

Z_9(S)=2 so l=9, r=10

a.i.) Z_i for AABCAABXAAZ

k=10 ≤ (r=10) so step 2:

S(10) contained in S[9..10]; S(10) matches S(10-9+1)=S(2)='A';

Z_2(S)=1 ≥ |S[10..10]|=1, hence **Step 2b)** but mismatch

S	A	A	B	C	A	A	B	X	A	A	Z
	1	2	3	4	5	6	7	8	9	10	11
Z_i(S)	--	1	0	0	3	1	0	0	2	1	

Z_{10}(S)=1

k=11 > (r=10) so step 1:

match S[11..|S|] to S[1..|S|]: mismatch so Z_{11}(S)=0

S	A	A	B	C	A	A	B	X	A	A	Z
	1	2	3	4	5	6	7	8	9	10	11
Z_i(S)	--	1	0	0	3	1	0	0	2	1	0

Z_{11}(S)=0

a.ii) Z_i for ABCDXABCYABDXY

$Z_2(S)$: $S(2) \neq S(1)$ so $Z_2(S)=0$, r=l=0

S	A	B	C	D	X	A	B	C	Y	A	B	D	X	Y
	1	2	3	4	5	6	7	8	9	10	11	12	13	14
$Z_i(S)$	--	0												

$Z_2(S)=0$

i=3..5: $Z_i(S)$: $S(i) \neq S(1)$ so $Z_i(S)=0$, r=l=0

S	A	B	C	D	X	A	B	C	Y	A	B	D	X	Y
	1	2	3	4	5	6	7	8	9	10	11	12	13	14
$Z_i(S)$	--	0	0	0	0									

$Z_{i=3..5}(S)=0$

$Z_6(S)$: $S(6) = S(1)$: S[6..8] matches S[1..3], so $Z_6(S)=3$, l=6 and r=8

S	A	B	C	D	X	A	B	C	Y	A	B	D	X	Y
	1	2	3	4	5	6	7	8	9	10	11	12	13	14
$Z_i(S)$	--	0	0	0	0	3								

$Z_6(S)=3$

a.ii) Z_i for ABCDXABCYABDXY

$Z_7(S)$: 7 ≤ (r=8) hence S(7)=S(7-6+1)=S(2)='B', $Z_2(S)$=0 whereas |S[7..8]|=2, hence $Z_7(S)=Z_2(S)$=0 and l and remain as they are: l=6 and r=8

S	A	B	C	D	X	A	B	C	Y	A	B	D	X	Y
	1	2	3	4	5	6	7	8	9	10	11	12	13	14
$Z_i(S)$	--	0	0	0	0	3	0							

$Z_7(S)=0$

$Z_8(S)$: 8 ≤ (r=8) hence S(8)=S(8-6+1)=S(3)='C', $Z_3(S)$=0 whereas |S[8..8]|=1, hence $Z_8(S)=Z_3(S)$=0 and l and remain as they are: l=6 and r=8

S	A	B	C	D	X	A	B	C	Y	A	B	D	X	Y
	1	2	3	4	5	6	7	8	9	10	11	12	13	14
$Z_i(S)$	--	0	0	0	0	3	0	0						

$Z_8(S)=0$

$Z_9(S)$: 9 > (r=8) but S(9) ≠ S(1) hence $Z_9(S)$=0 and l and remain as they are: l=6 and r=8

S	A	B	C	D	X	A	B	C	Y	A	B	D	X	Y
	1	2	3	4	5	6	7	8	9	10	11	12	13	14
$Z_i(S)$	--	0	0	0	0	3	0	0	0					

$Z_9(S)=0$

a.ii) Z_i for ABCDXABCYABDXY

$Z_{10}(S)$: 10 > (r=8), S(10)=S(1), match S[10..1] with S[1..2], hence $Z_{10}(S)$=2 and l=10 and r=11

S	A	B	C	D	X	A	B	C	Y	A	B	D	X	Y
	1	2	3	4	5	6	7	8	9	10	11	12	13	14
$Z_i(S)$	--	0	0	0	0	3	0	0	0	2				

$Z_{10}(S)$=0

$Z_{11}(S)$: 11 ≤ (r=11) hence S(11)=S(11-10+1)=S(2)='B', $Z_2(S)$=0 whereas lS[11..11]l=1, hence $Z_{11}(S)$=$Z_2(S)$=0 and l and remain as they are: l=10 and r=11

S	A	B	C	D	X	A	B	C	Y	A	B	D	X	Y
	1	2	3	4	5	6	7	8	9	10	11	12	13	14
$Z_i(S)$	--	0	0	0	0	3	0	0	0	2	0			

$Z_{11}(S)$=0

i=12..14: $Z_i(S)$=0

S	A	B	C	D	X	A	B	C	Y	A	B	D	X	Y
	1	2	3	4	5	6	7	8	9	10	11	12	13	14
$Z_i(S)$	--	0	0	0	0	3	0	0	0	2	0	0	0	0

i=12..14: $Z_i(S)$=0

b) Boyer-Moore

- “Apply the Boyer-Moore algorithm to find occurrences of P = ABXYABXZ in T = XABXYABXYABXZABXZABXYABXZA”
- The intuition behind Boyer-Moore:
  - Align P with T, check whether characters in P and T match, **from right to left**
  - Apply two heuristic rules: the bad character rule and the good suffix rule -- and apply the rule which yields the **maximum shift**

We start with the necessary preprocessing:

- *Compute L’(i) and l’(i) for each position i of P*
- *and compute R(x) for each character* $x \in \Sigma$

b) Boyer-Moore: preprocessing

- **First preprocessing step**: computing L'(i) for each position in P
  - For each i, L'(i) is the largest position less than n such that string P[i..n] matches a suffix of P[1..L'(i)] *and* such that the character preceding the suffix is not equal to P(i-1).
  - and L'(i) = 0 if there is no position satisfying the conditions.
  - Example:

P =	C	A	B	D	A	B	D	A	B	L(8)=6 L'(8)=3
	1	2	3	4	5	6	7	8	9	

  - For our pattern **P = ABXYABXZ** , we can notice right away that L'(i) = 0 for all i, but here we'll show the computation in detail
    - Why? Since the character "Z" only appears once at the end of the string, there can be no substring of P[1...(n-1)] able to match a suffix of P

b) Boyer-Moore: preprocessing

- We can compute L'(i) based on the $N_j(P)$ values
  - $N_j(P)$ is the length of the longest suffix of the substring P[1...j] that is also a suffix of the full string P.
  - $N_j(P)$ is the *reverse* operation of $Z_j(P)$
  - For our pattern **P = ABXYABXZ**, we can immediately notice that $\mathbf{N_j(P) = 0}$ for all $1 \leq j \leq |P|$
  - As a consequence, L'(i) is also = 0 for all $1 \leq i \leq |P|$

b) Boyer-Moore: preprocessing

- **Second preprocessing step**: Computing the l'(i) values:
  - l'(i) denotes the longest suffix of P[i..n] that is also a prefix of P, if one exists. If none exists, let l'(i) be zero.
  - For our pattern **P = ABXYABXZ**, there is no suffix of P[i..n] that is also a prefix of P
    - Why? Same reason as for L'(i): the character Z does not appear anywhere else in the string

- **Third (and final) preprocessing step**: computing R(x) for each character x ∈ Σ
  - For our pattern **P = ABXYABXZ**, we therefore have R(A) = 5, R(B) = 6, R(X) = 7, R(Y) = 4, and R(Z) = 8

b) Boyer-Moore: search

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
X	A	B	X	Y	A	B	X	Y	A	B	X	Z	A	B	X	Z	A	B	X	Y	A	B	X	Z	A

A	B	X	Y	A	B	X	Z
1	2	3	4	5	6	7	8
							✘

mismatch at T(8) = X

... and R(X) = 7

The bad character rule tells us that we can shift P to the right by max[1, i-R(T(k))] = places

⇨ In this case, max[1, i-R(T(k))] = 1

b) Boyer-Moore: search

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
X	A	B	X	Y	A	B	X	Y	A	B	X	Z	A	B	X	Z	A	B	X	Y	A	B	X	Z	A

A	B	X	Y	A	B	X	Z
1	2	3	4	5	6	7	8

✘ mismatch at T(9) = Y

... and R(Y) = 4

The bad character rule tells us that we can shift P to the right by max[1, i-R(T(k))] = places

➡ In this case, max[1, i-R(T(k))] = 4

b) Boyer-Moore: search

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
X	A	B	X	Y	A	B	X	Y	A	B	X	Z	A	B	X	Z	A	B	X	Y	A	B	X	Z	A
					A	B	X	Y	A	B	X	Z													
					1	2	3	4	5	6	7	8													
					✔	✔	✔	✔	✔	✔	✔	✔													

We found a full match at T[6] !

We can now shift the pattern by (n-l'(2)) places

➡ In this case, (n-l'(2)) = 8

b) Boyer-Moore: search

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26

X A B X Y A B X Y A B X Z A B X Z A B X Y A B X Z A

A B X Y A B X Z

1 2 3 4 5 6 7 8

mismatch at T(21) = Y

... and R(Y) = 4

The bad character rule tells us that we can shift P to the right by max[1, i-R(T(k))] = places

In this case, max[1, i-R(T(k))] = 4

b) Boyer-Moore: search

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
X	A	B	X	Y	A	B	X	Y	A	B	X	Z	A	B	X	Z	A	B	X	Y	A	B	X	Z	A
																	A	B	X	Y	A	B	X	Z	
																	1	2	3	4	5	6	7	8	
																	✔	✔	✔	✔	✔	✔	✔	✔	

We found another full match at T[18] !

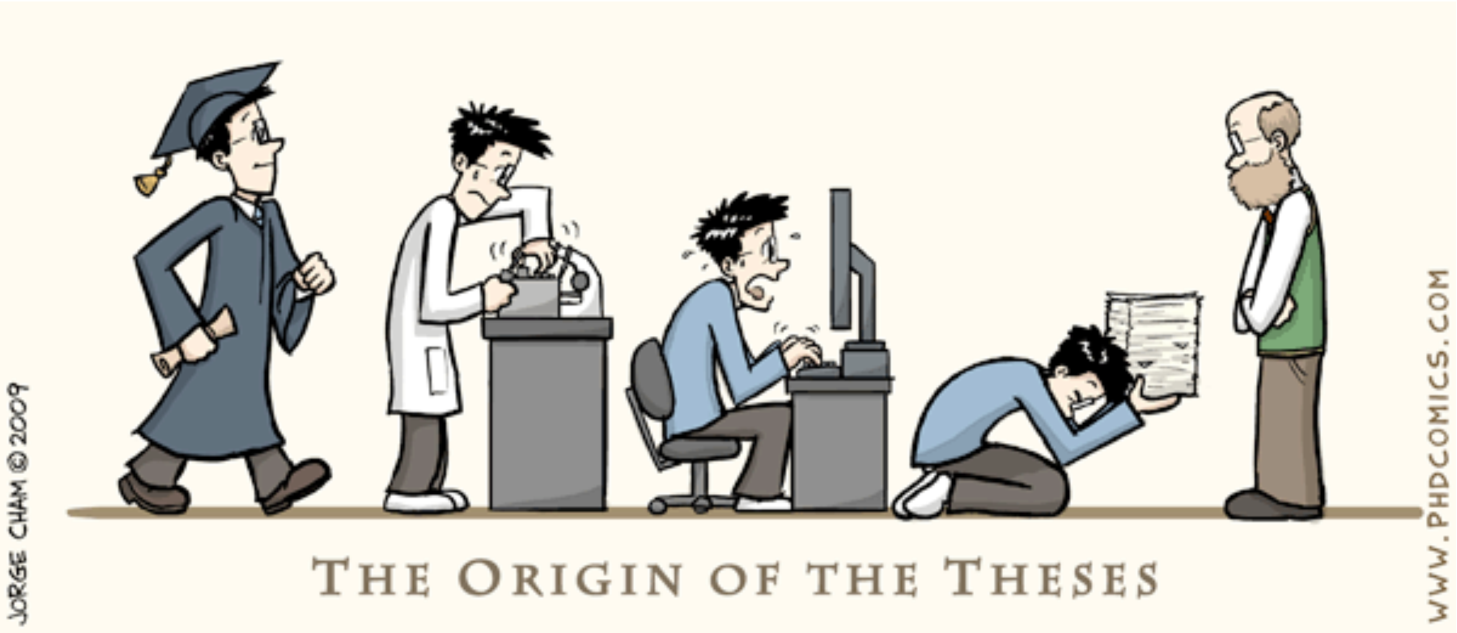
... and we‘re done :-)

Final words

- The exam will contain one question about string matching
- It will consist of a question similar to the ones of this exercise session (no bad surprises)
- What is important is that you describe in detail the *steps* that you follow in the algorithm
  - Provide intermediate results (values for Z_i, N_i, L'(i), etc.)
  - Show me that you understand how the algorithm works!

Shameless plug

- I assume that many of you will start searching soon for a good topic for your M.Sc. thesis



- If you‘re interested, I wrote down a list of topics for which I can provide some guidance
  - Mostly about dialogue systems, but also 2-3 more “linguistically-oriented” topics
- The list is available on my website:
  - http://www.dfki.de/~plison/thesistopics.html
  - Just let me know if you‘re interested!