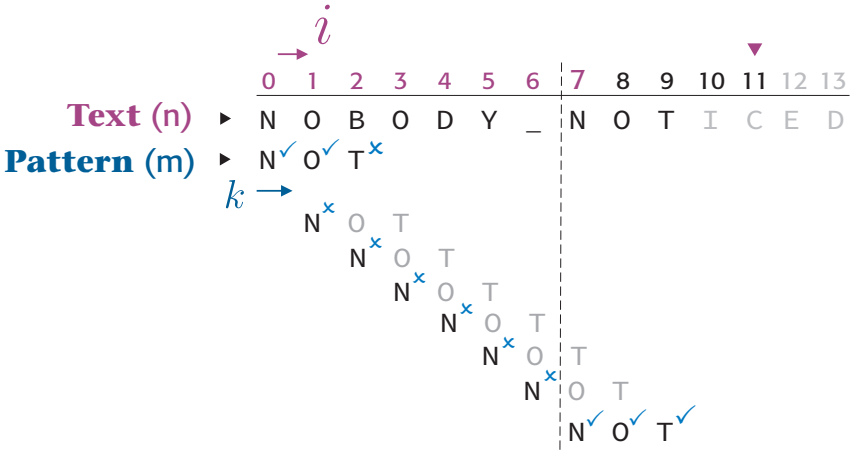
Revisit String Matching



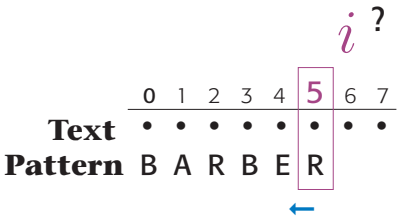
Exercise
Determine #char-comps for the instance. What is the worst-case for this family of instances? *Ans. next slide.*

Quiz
When would brute force string matching result in linear efficiency?

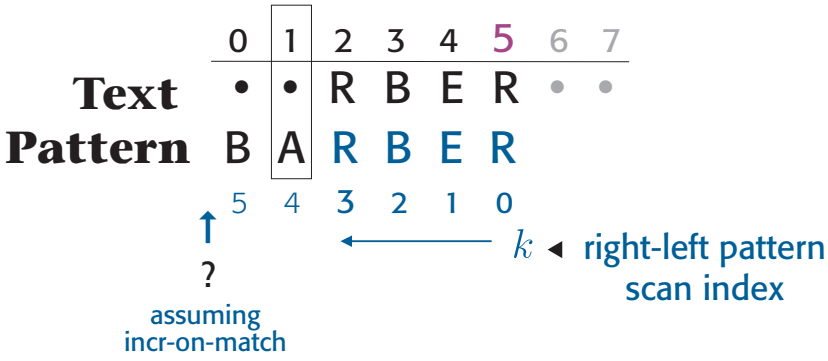
Brute force performance

- How many match positions (tries)?
- How many comparisons/try?
- At worst? Linear? When? A limitation

Horspool's String Matching
Basic Match Procedure



Quiz
Determine: pattern size, initial match position, what's the pattern match success condition?



⇨ Match success condition

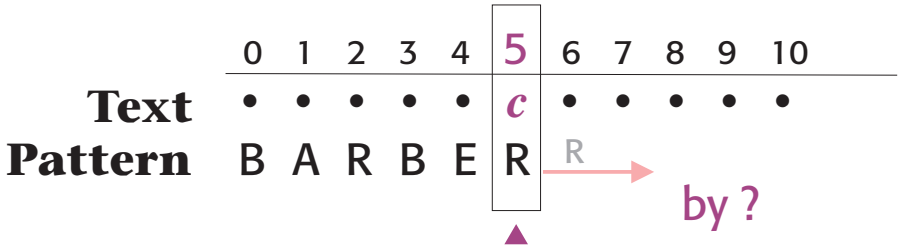
⇨ Where next on mismatch?

For instances n=14, m=3, match at all positions up to 11 (no sense past since less chars than pattern), 7 this instance, comp-chars in pattern up to 3 each time, 12 this instance. Family of instances worst-case: 12 pos × 3 comps = 36, matches formula $m(n-m+1)$.

Horspool's String Matching
Pattern Shifts

No more always shift by one

Shifts depend on *c*; if not in pattern, a shift by one guarantees a mismatch, why? Hint: assume *c* = S (or any char not in pattern).



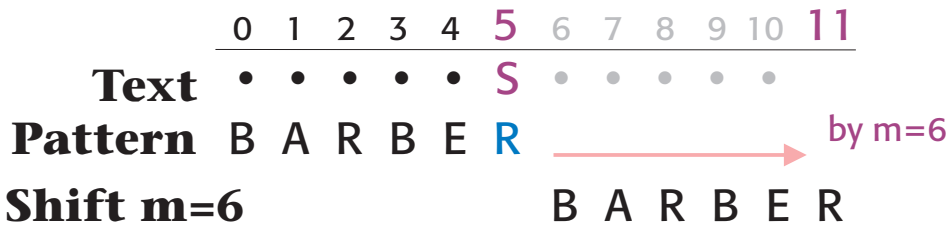
landing text char *c* aligned
against rightmost pattern char

Horspool's String Matching
Full Shifts

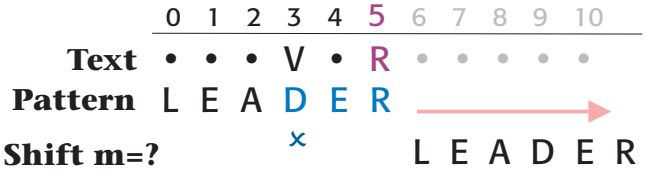
Where to try next on mismatch?

Cases are easier to see when the first match try is considered for a basic case.

c = S not in pattern; no sense trying to match anywhere left to S.



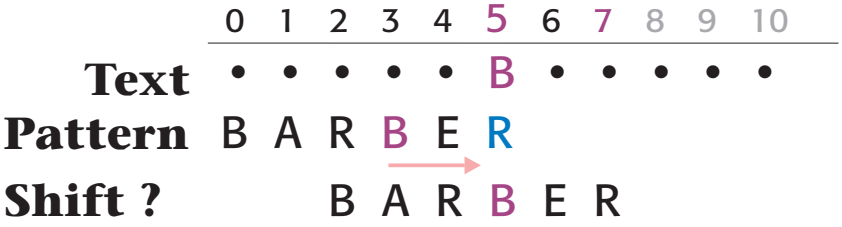
c = R not repeated in pattern, no sense trying to match anywhere left of V.



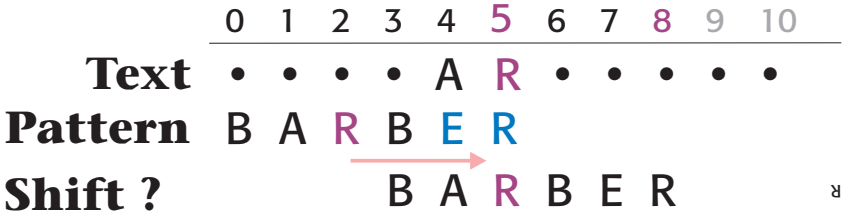
Exercise
Try: LERDER, TRADER, READER. *Ans. next slide.*

Horspool's String Matching
Partial Shifts

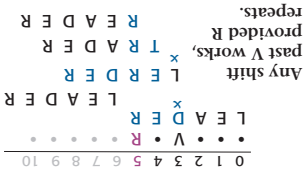
Align against right-most B since c = B is in the pattern.



Align against right-most R since c = R repeats in the pattern (note answer to previous exercise).



Either way, landing char (B top, R bottom) occurs or repeats in pattern.



Horspool's String Matching
Trading Space for Time

Shifts the same, depend only on *c*.

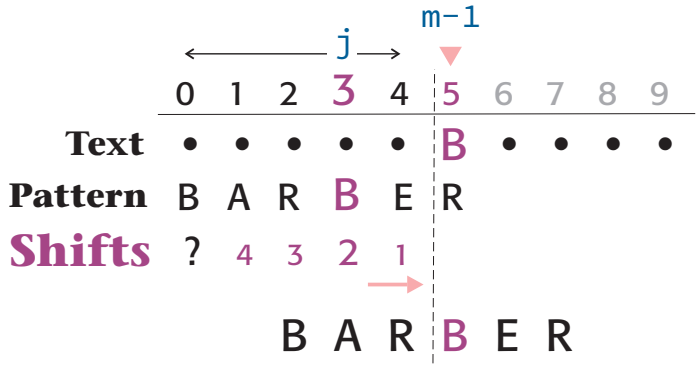
No need to check for cases on each match attempt, a constant time memory lookup will be faster, if the cost of generating lookup info is low enough.

Quiz
What's the down side to this approach?

Pre-calculate and store shifts for each possible char in text based on pattern, accounting for all cases

Partial Shifts ⭐ Amounts

Exercise
Why shift based on the closest to last occurrence if a character is repeated in the pattern? (i.e., B in *P*[3])
Hint: check textbook and "Partial Shifts" slide.



Quiz
Clearly the shift for B is 2, but what is the general formula for position `j`? What's the range of `j`?

Horspool's String Matching
Pattern Shift Table

➪ **Text alphabet**

```
Table[c]
  A:4
  B:2
  C:6
  D:6
  E:1
  ...
  R:3
  ...
  Z:6
  :6
```

Example alphabets

- ✎ Upper case + space
- ✎ Lower/upper case + space + digits

Algorithm $ShiftTable$
Input $P[0..m-1]$
Output Pattern shift $Table[0..size-1]$ indexed by text alphabet

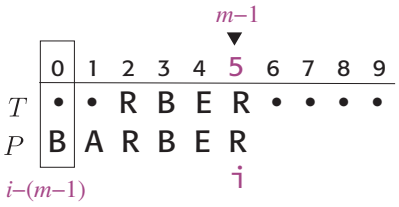
1: $\forall j\ (0 \le j < size)\ Table[j] \leftarrow m$
2: **for** $j \leftarrow 0$ **to** $m-2$ **do**
3: $\quad Table[P[j]] \leftarrow m-1-j$
4: **return** $Table$

Quiz
Determine an alphabet for searching in a natural non-technical text in Arabic?
Hint: how many shapes do you need to match?

Efficiency?

String Matching
Horspool's Algorithm

Exercise
Discuss choice of input size parameter.



Exercise
Trace the first iteration (i=5). What is the shift amount? Hint: check Table['R'].

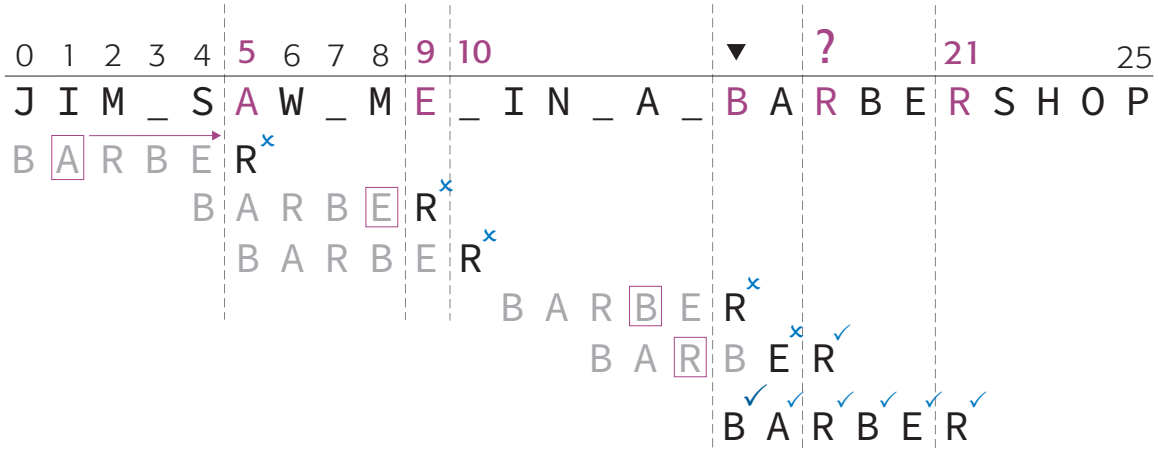
Exercise
Use on instance from Slide 1 and compare to brute force results.

Algorithm *HorspoolMatching*
Input Pattern $P[0..m-1]$, text $T[0..n-1]$
Output Index of first match of pattern in text otherwise -1

```
1: ShiftTable(P[0..m-1])
2: i ← m-1                                  ▷ first match position
3: while i ≤ n-1 do                         ▷ try to match at pos i
4:     k ← 0                                ▷ start rigth-to-left pattern scan
5:     while k ≤ m-1 and p[m-1-k] = T[i-k] do
6:         k ← k+1
7:     if k = m then                        ▷ match success condition
8:         return i-m+1                     ▷ successful match index
9:     else i ← i+Table[T[i]]               ▷ or try next match position
10: return -1
```

Horspool's Algorithm Example

```
Table[c]
  A:4
  B:2
  C:6
  D:6
  E:1
  ...
  R:3
  ...
  Z:6
  :6
```



Quiz
How many match positions (i.e., match tries) before a match? Compare to brute force. (Repeat other.)

Exercise
Label each line with the applicable case from your textbook. For each case what is *c*?

```
0123456789012345678 9
JVFRYEHULJTVTRCRORZQ
CROR
```

```
0123456789012345678 9
JVFRYEHULJTVTRCRORZQ
TROR
```

String Matching
Results Summary

➪ **Linear guarantee***

Exercise
What is the efficiency in each case? Under what conditions should we expect these results? (Answers in textbook).

➪ Worst-case performance

- ✎ Brute-force: quadratic in m · n
- ✎ Horspool's: ?
- 🕮 * ✎ ?Boyer-Moore (Knuth-Morris-Pratt)

Exercise
Compare Horspool's algorithm to brute-force string matching. Which one would you use to search strings encoding DNA sequences, why?

➪ Average performance: when?

- ✎ Brute-force: linear in n+m, ?
- ✎ … ?

String Matching Applications

Exercise
Suggest a suitable matching algorithm for each application, give reasons.

➪ Natural text search

- ✎ Digital documents, libraries
- ✎ Web search (engines)

A big business in addition to being an important application.

Mining textual data such as security logs, and plagiarism detection are major text pattern recognition applications.

➪ Natural text processing

Parsing/matching strings is important to compilers and programming.

➪ Computer science

➪ Bioinformatics

Applications in **computational molecular biology** include searching a genome, DNA and protein sequences.