Data Structures

String pattern matching

The notes for the lesson “String pattern matching”: chapter 6.9, with their figures, worked examples, exam traps, quizzes and exam question bank, in one file.

1 chapter | Chapter 6.9

About these notes

These are the notes for the University free n easy video lesson **String pattern matching** on Data Structures: chapter 6.9, with every figure, worked example, exam trap and quiz, and the long-answer exam questions on these chapters.

6.9 String pattern matching: KMP, Boyer-Moore and tries

Pattern matching finds every place where a pattern P of length m occurs in a text T of length n: Ctrl+F, grep, a virus scanner, a DNA search. The naive method slides P along T and compares at every shift, O(nm) in the worst case. Knuth-Morris-Pratt (KMP) never re-reads a text character, O(n + m). Boyer-Moore compares from the right end of the pattern and often skips whole chunks of text, sublinear in practice. A trie stores a whole dictionary of words as a tree of characters so that any word is found in time proportional to its length. Several syllabi list this chapter; others leave it to a later algorithms course.

You need: arrays and strings (an earlier lesson); trees (an earlier lesson) for the trie.

Knuth-Morris-Pratt (1977)

When the naive method mismatches after matching j characters, it shifts by one and forgets everything. KMP keeps what it learned: the j matched characters are P[0..j−1], and the next useful shift is determined by P alone. Precompute the **failure function** (also called the LPS or prefix table): lps[i] = length of the longest proper prefix of P[0..i] that is also a suffix of P[0..i]. On a mismatch with j characters matched, continue matching with j = lps[j−1]; the text index never moves back.

1. **Build lps**: lps[0] = 0, len = 0. For i = 1 to m − 1: while len > 0 and P[i] ≠ P[len], set len = lps[len − 1]; if P[i] = P[len], len = len + 1; lps[i] = len. O(m).
2. **Search**: i = 0 (text), j = 0 (pattern). While i < n: if T[i] = P[j], advance both; if j reaches m, report a match at i − m and set j = lps[j − 1]. On a mismatch, if j > 0 set j = lps[j − 1], else advance i. O(n).

Each step either advances i or decreases j, and j can only decrease as many times as it increased, so the search makes at most 2n comparisons: Θ(n + m) in total, with O(m) extra space.

Boyer-Moore

Align P with T, but compare from the **right end** of P leftwards. On a mismatch at pattern index j with text character c, the **bad character rule** slides P right so that the rightmost occurrence of c in P lines up under the text position: shift = max(1, j − last[c]), where last[c] is the index of the last occurrence of c in P (−1 if absent). If c is not in P at all, the whole pattern jumps past it. The full algorithm adds the **good suffix rule** (reuse the matched suffix) and takes the larger of the two shifts. With both rules the worst case is O(n + m) for a pattern that does not occur; with the bad character rule alone it can degrade to O(nm) (text "aaaa…", pattern "baaa"). The best case is O(n/m): on English text with a 10-letter pattern most alignments cost one comparison and skip ten characters, which is why grep and text editors use Boyer-Moore variants.

Tries

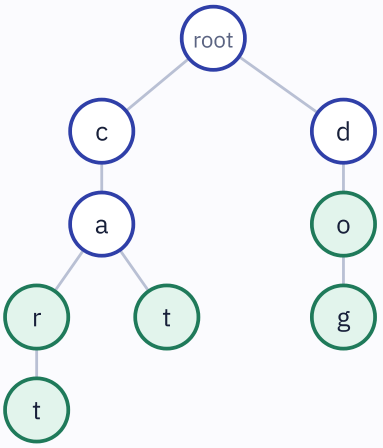
A trie (from re*trie*val) is a tree in which each edge carries one character and each node represents the prefix spelled by the path from the root. Words end at nodes marked as terminal. Insert and search walk one level per character: O(L) for a word of length L, independent of how many words are stored. All words with a given prefix sit in one subtree, which gives autocomplete for free. The cost is space: a node with an array of 26 child pointers wastes most of them, so compact variants (ternary search tries, radix or Patricia trees with compressed edges) are used for large dictionaries, IP routing tables and T9-style keyboards.

KMP failure table, P = ABABAC

i	0	1	2	3	4	5
P[i]	A	B	A	B	A	C
lps[i]	0	0	1	2	3	0

lps[4] = 3: "ABA" is both a prefix and a suffix of ABABA, so after a mismatch at C the search resumes at j = 3.

Trie: car, cat, cart, do, dog



green = end of a word; "ca" is shared by car, cat, cart

Figure 6.9.1 Left: the KMP failure table of ABABAC; each entry is the length of the longest border (prefix that is also a suffix) of the prefix ending there. Right: a trie of five words; a search for "cart" walks root → c → a → r → t in four steps, and the subtree under "ca" lists every completion of that prefix.

WORKED EXAMPLE

Question (a). Build the KMP failure table for P = ABABAC and search for it in T = ABABABAC. Count the character comparisons.

Solution. Table: i = 0 → 0; i = 1: B ≠ A → 0; i = 2: A = P[0] → 1; i = 3: B = P[1] → 2; i = 4: A = P[2] → 3; i = 5: C ≠ P[3] = B, fall back to lps[2] = 1, C ≠ P[1] = B, fall back to lps[0] = 0, C ≠ P[0] = A → 0. lps = **[0, 0, 1, 2, 3, 0]**.

I (TEXT)	T[I]	J (PATTERN)	P[J]	RESULT
0–4	A B A B A	0–4	A B A B A	5 matches, j = 5
5	B	5	C	mismatch → j = lps[4] = 3
5	B	3	B	match, j = 4
6	A	4	A	match, j = 5
7	C	5	C	match, j = 6 = m: found at 7 − 6 + 1 = **index 2**

9 comparisons; the text index never went backwards. The naive method would make 6 + 1 + 6 = 13.

Question (b). Use Boyer-Moore with the bad character rule to find P = NEEDLE in T = FINDINAHAYSTACKNEEDLEINA (n = 24).

Solution. last[ ] table: N → 0, E → 5, D → 3, L → 4; every other letter −1.

SHIFT S	TEXT WINDOW	COMPARE FROM RIGHT	BAD CHARACTER	SHIFT = MAX(1, J − LAST[C])
0	FINDIN	T[5] = N vs E: mismatch at j = 5	N, last = 0	5 − 0 = 5
5	NAHAYS	T[10] = S vs E: mismatch at j = 5	S, last = −1	5 + 1 = 6
11	TACKNE	E = E; T[15] = N vs L: mismatch at j = 4	N, last = 0	4 − 0 = 4
15	NEEDLE	E, L, D, E, E, N all match	—	found at **index 15**

Only **10 comparisons** for a 24-character text, and 16 of the 24 characters of T were never looked at.

Question (c). Insert car, cat, cart, do, dog into an empty trie and state the number of nodes.

Solution. car creates root → c → a → r (3 new nodes, r terminal). cat reuses c, a and adds t (1 node). cart reuses c, a, r and adds t under r (1 node). do adds d, o (2 nodes). dog adds g (1 node). Total **8 nodes plus the root**; terminal nodes: r, t, t, o, g. A search for "ca" reaches node a, which is not terminal, so "ca" is a prefix but not a stored word.

EXAM TRAPS

- lps[i] is the longest *proper* prefix that is also a suffix: never the whole string, so lps[0] = 0 always.
- On a KMP mismatch the new j is lps[j − 1], not lps[j], and the text pointer i does not move.
- Boyer-Moore compares right to left but shifts the pattern to the right; a shift computed as negative or zero must become 1.
- KMP is O(n + m) in the worst case; Boyer-Moore's famous O(n/m) is the best case. Do not swap them.
- Trie search is O(L) in the word's length, not O(log N) in the number of words; the price is space, up to 26 pointers per node.

CHECK YOURSELF

– REMEMBER **Give the time complexity of naive, KMP and Boyer-Moore matching.** 3 marks

Naive O(nm) worst case; KMP Θ(n + m); Boyer-Moore O(n/m) best, O(n + m) worst with both rules, O(nm) with the bad-character rule alone.

– REMEMBER **What does lps[i] mean?** 2 marks

The length of the longest proper prefix of P[0..i] that is also a suffix of P[0..i].

– UNDERSTAND **Why does Boyer-Moore compare from the right end of the pattern?** 3 marks

A mismatch at the right end reveals a text character that, if absent from the pattern, lets the whole pattern skip past it: m characters of text for one comparison. Comparing from the left could skip at most one position per mismatch.

– APPLY **Compute the failure table for P = AABAAAB.** 4 marks

i=1: A=A → 1; i=2: B≠A, back to lps[0]=0, B≠A → 0; i=3: A → 1; i=4: A → 2; i=5: A≠B, back to lps[1]=1, A=A → 2; i=6: B=P[2] → 3. lps = [0, 1, 0, 1, 2, 2, 3].

– APPLY **P = TRAIN, T = IRCTCTRAINS. First Boyer-Moore shift using the bad character rule?** 3 marks

Window IRCTC, compare T[4] = C with N: mismatch at j = 4; C is not in TRAIN, so shift = 4 − (−1) = 5. The window becomes TRAIN at index 5 and matches on the next alignment.

– ANALYZE **An autocomplete for 5 lakh station names must list completions of a typed prefix in under a millisecond. Compare a sorted array with binary search, a hash table and a trie.** 4 marks

Hash table: exact lookups only, no prefix queries. Sorted array: binary search finds the first match in O(log n) ≈ 19 steps, then completions are contiguous; memory-light, but each step is a string comparison. Trie: walk L characters to the prefix node and list its subtree; fastest per query and independent of n, but memory-heavy unless compressed (radix tree). Either of the last two works; the trie wins when queries dominate, the sorted array when memory does.

BEYOND THE SYLLABUS

Not for the exam. Aho-Corasick extends KMP's failure links to a trie so that thousands of patterns (virus signatures, banned words) are matched in one pass of the text. Suffix trees and suffix arrays index the *text* instead of the pattern and power genome search. Rabin-Karp hashes windows of the text (rolling hash) and is the base of plagiarism checkers. Modern grep tools use Boyer-Moore-Horspool, SIMD memchr scans and the Two-Way algorithm. Search for "Aho-Corasick", "suffix array construction" and "ripgrep Teddy algorithm".

Key terms of this lesson

TERM	MEANING OR FORMULA	CHAPTER
KMP failure table	lps[i] = longest proper prefix of P[0..i] that is a suffix; search Θ(n + m)	6.9
Bad character rule	shift = max(1, j − last[c]); Boyer-Moore best case O(n/m)	6.9
Trie	One character per edge; search O(L) in word length	6.9

Exam question bank

Long-answer exam questions on these chapters, each with an outline of the points an examiner looks for. Answer one in full on paper before you read its outline.

Remember and understand

- **6. Describe the KMP algorithm. Explain the failure function and why the text pointer never moves backwards. (8 marks)**

  Outline: definition of lps[i] as the longest proper prefix-suffix (border); construction in O(m); search loop with j = lps[j−1] on mismatch; amortized argument that j decreases at most as often as it increases → ≤ 2n comparisons; example table for a short pattern.

Apply

- **12. Build the KMP failure table for P = AABAAB and find all occurrences in T = AABAAAABAABAAB, listing every fallback. (10 marks)**

  Outline: lps = [0, 1, 0, 1, 2, 3]; the first five text characters match (j = 5), T[5] = A ≠ B → j = lps[4] = 2; A ≠ B again → j = lps[1] = 1; A = A → j = 2; T[6] = A ≠ B → j = 1; then matching proceeds to a full match ending at index 10 (start index 5), j = lps[5] = 3, and a second match at start index 8; 17 comparisons for n = 14, within the 2n bound; the text pointer never moved back.

Analyze and evaluate

- **16. A search feature must support exact match, prefix autocomplete and "find this substring in a document". Match each need to binary search, hashing, a trie, KMP or Boyer-Moore and justify the costs. (10 marks, spans chapters 6.1, 6.7 and 6.9)**

  Outline: exact match → hash table O(1) (or binary search O(log n) when memory matters); prefix → trie O(L) or binary search on a sorted array for the first match; substring in a document → KMP Θ(n + m) guaranteed, Boyer-Moore faster on natural text; explain why a hash cannot do prefixes and why a trie cannot do substrings without a suffix structure.

Glossary (terms used in this lesson)

Bad character rule	Boyer-Moore's shift: align the mismatched text character with its last occurrence in the pattern.
Binary search	Search on a sorted array that halves the range with every comparison; O(log n).
Border	A proper prefix of a string that is also its suffix; the KMP failure table stores the longest border of each prefix.
Trie	A tree with one character per edge; stores a set of strings with O(L) lookup.

Sources and further reading

- Mark Allen Weiss, *Data Structures and Algorithm Analysis in C*, 2nd edition: the chapters on hashing and sorting are prescribed reading in many syllabi.
- Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest and Clifford Stein, *Introduction to Algorithms*: heapsort, quicksort, lower bounds and linear-time sorts, hash tables, and string matching (KMP).
- Ellis Horowitz, Sartaj Sahni and Susan Anderson-Freed, *Fundamentals of Data Structures in C*: sorting (including external sorting) and hashing.
- Robert Sedgewick and Kevin Wayne, *Algorithms*, 4th edition: shellsort, hash tables, tries and substring search, with the free companion site.
- Donald E. Knuth, *The Art of Computer Programming, Volume 3: Sorting and Searching*: the source of the probe-count formulas for linear probing and of the multiplication-method constant.
- Wikipedia, "Linear probing": en.wikipedia.org/wiki/Linear_probing, for the expected-cost bounds in terms of the load factor (opened Nov 2026).
- Wikipedia, "Knuth–Morris–Pratt algorithm": en.wikipedia.org/wiki/Knuth–Morris–Pratt_algorithm, for the 1977 date and the Θ(m) + Θ(n) bounds (opened Nov 2026).
- Wikipedia, "Shellsort": en.wikipedia.org/wiki/Shellsort, for Shell's 1959 paper, the gap sequences and their worst-case bounds (opened Nov 2026).
- Syllabi: the universal syllabus and a range of university course documents.

Built for the University free n easy YouTube channel from the universal syllabus of Data Structures (Oct 2026). Written with top universities' syllabi and standard textbooks; check a formula against your prescribed book before an exam, and report mistakes on the channel.