

---

# **Data Structures and Algorithms in Java™**

**Sixth Edition**

**Michael T. Goodrich**

Department of Computer Science  
University of California, Irvine

**Roberto Tamassia**

Department of Computer Science  
Brown University

**Michael H. Goldwasser**

Department of Mathematics and Computer Science  
Saint Louis University

---

## **Study Guide: Hints to Exercises**

**WILEY**

# Chapter 13

---

## Text Processing

---

### Hints

---

#### Reinforcement

- R-13.1)** The empty string is one of them.
- R-13.2)** Recall the definitions of prefix and suffix.
- R-13.3)** Mimic the style of the text-matching figures in the book.
- R-13.4)** Mimic the style of the text-matching figures in the book.
- R-13.5)** Mimic the style of the text-matching figures in the book.
- R-13.6)** Use the algorithm presented in the book.
- R-13.7)** Use the version of the algorithm presented in the book.
- R-13.8)** Mimic the drawing style used in the book.
- R-13.9)** Mimic the drawing style used in the book.
- R-13.10)** Mimic the drawing style used in the book.
- R-13.11)** Don't forget to include the space character.
- R-13.12)** Draw the entire table for the dynamic programming algorithm.
- R-13.13)** All answers are encoded in the table.
- R-13.14)** Simulate a running of the algorithm presented in the book.

---

#### Creativity

- C-13.15)** Make the text and the pattern very periodic.
- C-13.16)** Use symmetry to redesign the search from right to left, yet still returning the index at which the pattern *starts*.
- C-13.17)** Use symmetry to redesign the search from right to left, including the definition of the “last” map.
- C-13.18)** Use symmetry to redesign the search from right to left, including the definition of the failure function.
- C-13.19)** The justification is similar to the argument that the number of iterations in findKMP is  $O(n)$ .

- C-13.20)** Consider modifying the KMP matching algorithm.
- C-13.21)** Convert this problem to a noncircular pattern-matching problem.
- C-13.22)** The failure function can now take advantage of the fact that it knows what does match in the mismatched location.
- C-13.23)** You need to incorporate a failure function with the Boyer-Moore heuristics.
- C-13.24)** Consider using a prefix trie.
- C-13.25)** Start by building a suffix trie.
- C-13.26)** Start by locating the leaf that corresponds to the end of the string.
- C-13.27)** Start by locating the leaf that corresponds to the end of the string.
- C-13.28)** Recall how you identify the branches of the suffix trie that can be compressed.
- C-13.29)** Create some way of visualizing your standard trie so that you can verify that it is being constructed correctly.
- C-13.30)** Create some way of visualizing your compressed trie so that you can verify that it is being constructed correctly.
- C-13.31)** Create some way of visualizing your prefix trie so that you can verify that it is being constructed correctly.
- C-13.32)** Build a prefix tree for  $X$  and a suffix tree for  $Y$ .
- C-13.33)** First give as many quarters as possible.
- C-13.34)** Don't use normal denominations like you would find in a country on earth.
- C-13.35)** We can use a greedy algorithm.
- C-13.36)** Consider using a greedy algorithm.
- C-13.37)** You can rely on our implementation of trees and priority queues.
- C-13.38)** Keep around extra information in the table for the dynamic programming algorithm.
- C-13.39)** Anatjari should use a greedy algorithm.
- C-13.40)** Review the LCS algorithm.
- C-13.41)** There is a surprising similarity between this problem and the matrix chain-product problem.
- C-13.42)** Use a greedy algorithm.
- C-13.43)** Review the LCS algorithm.
- C-13.44)** The edit distance algorithm is a dynamic program based on the LCS problem.

**C-13.45)** Use dynamic programming.

**C-13.46)** Use brute force, first to enumerate all pairs  $(a, b)$  such that  $a$  is in  $A$  and  $b$  is in  $B$ .

**C-13.47)** Use dynamic programming.

---

## Projects

**P-13.48)** You can find large documents on the Internet.

**P-13.49)** You can find large documents on the Internet.

**P-13.50)** You can find large documents on the Internet.

**P-13.51)** Try using inputs that are likely to cause both best-case and worst-case running times for various algorithms.

**P-13.52)** Make sure to avoid integer overflow in your intermediate calculations when evaluating the hash function.

**P-13.53)** Use an inverted file data structure.

**P-13.54)** Use an inverted file data structure and store page ranks.

**P-13.55)** Stick to the smaller strings, since LCS is a quadratic algorithm.

**P-13.56)** On Unix/Linux systems, there is usually a list of words located at `/usr/dict/words` or `/usr/share/dict/words`.