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

Instructor's Solutions Manual

WILEY

Chapter 13 Text Processing

Hints and Solutions

Reinforcement

R-13.1) Hint The empty string is one of them.

R-13.1) Solution There are four: `a, aa, aaa,` and the empty string.

R-13.2) Hint Recall the definitions of prefix and suffix.

R-13.2) Solution The longest prefix that is also a suffix of this string is `"cgtacg"`.

R-13.3) Hint Mimic the style of the text-matching figures in the book.

R-13.3) Solution

a	a	a	b	a	a	d	a	a	b	a	a	a
a	a	b	a	a	a							
	a	a	b	a	a	a						
		a	a	b	a	a	a					
			a	a	b	a	a	a				
				a	a	b	a	a	a			
					a	a	b	a	a	a		
						a	a	b	a	a	a	
							a	a	b	a	a	a

R-13.4) Hint Mimic the style of the text-matching figures in the book.

R-13.4) Solution

a	a	a	b	a	a	d	a	a	b	a	a	a
a	a	b	a	a	a							
	a	a	b	a	a	a						
				a	a	b	a	a	a			
					a	a	b	a	a	a		
						a	a	b	a	a	a	
							a	a	b	a	a	a

R-13.5) Hint Mimic the style of the text-matching figures in the book.

R-13.5) Solution

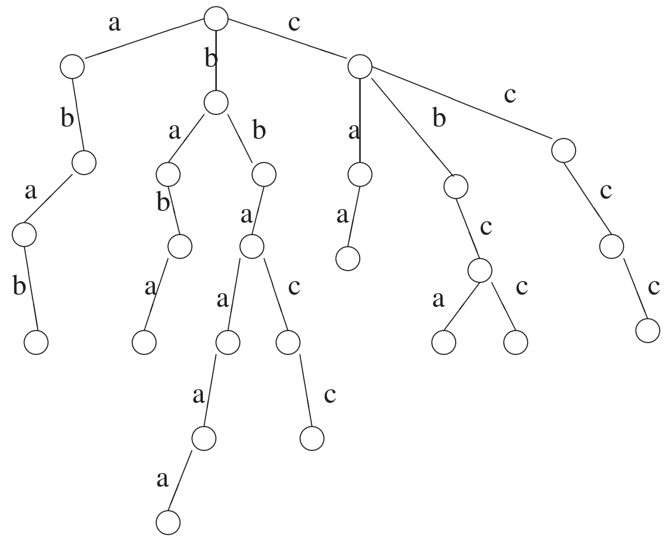
a	a	a	b	a	a	d	a	a	b	a	a	a
a	a	b	a	a	a							
			a	a	b	a	a	a				
						a	a	b	a	a	a	
							a	a	b	a	a	a

R-13.6) Hint Use the algorithm presented in the book.

R-13.7) Hint Use the version of the algorithm presented in the book.

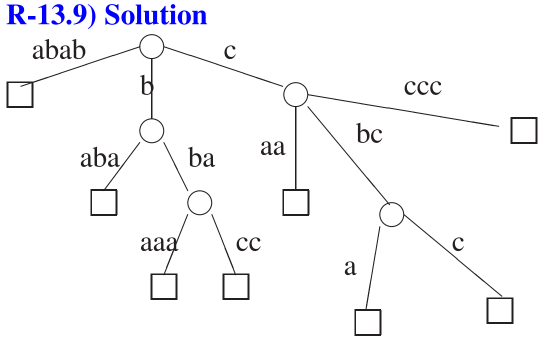
R-13.8) Hint Mimic the drawing style used in the book.

R-13.8) Solution



R-13.9) Hint Mimic the drawing style used in the book.

R-13.9) Solution



R-13.10) Hint Mimic the drawing style used in the book.

R-13.11) Hint Don't forget to include the space character.

R-13.12) Hint Draw the entire table for the dynamic programming algorithm.

R-13.13) Hint All answers are encoded in the table.

R-13.13) Solution Two additional common substrings are `CTAATA` and `GTAATA`.

R-13.14) Hint Simulate a running of the algorithm presented in the book.

Creativity

C-13.15) Hint Make the text and the pattern very periodic.

C-13.15) Solution $T =$ "`aaaaaaaaaaaaaaaaaa`", $P =$ "`aaaab`".

C-13.16) Hint Use symmetry to redesign the search from right to left, yet still returning the index at which the pattern *starts*.

C-13.17) Hint Use symmetry to redesign the search from right to left, including the definition of the "last" map.

C-13.18) Hint Use symmetry to redesign the search from right to left, including the definition of the failure function.

C-13.19) Hint The justification is similar to the argument that the number of iterations in findKMP is $O(n)$.

C-13.19) Solution The justification is similar to the argument that the number of iterations in findKMP is $O(n)$.

Define $s = j - k$ for the sake of analysis. One of the following conditions occurs at each iteration of the loop:

- If $P[j] = P[k]$, then j increases by 1, and s does not change, since k also increases by 1.
- If $P[j] \neq P[k]$ and $k > 0$, then j does not change and s increases by at least 1, since in this case s changes from $j-k$ to $j - f(k-1)$, which is an addition of $k - f(k-1)$, which is positive because $f(k-1) < k$.
- If $P[j] \neq P[k]$ and $k = 0$, then j increases by 1 and s increases by 1, since k does not change.

Since either j or s (or both) increase by at least one during each iteration, and yet both are bounded by m, the number of iterations is at most $2m$. Therefore, KMPFailureFunction runs in $O(m)$ time.

C-13.20) Hint Consider modifying the KMP matching algorithm.

C-13.20) Solution Modify the findKMP algorithm to maintain a variable *maxIndex* which is the index of the longest prefix found, *maxLen* which is the length of the longest prefix found, and *currentLen* which is the length of the current prefix. Initialize all three variables to zero and modify the loop in findKMP as follows:

- If $T[j] = P[k]$, increment *currentLen*.
- If $T[j] \neq P[k]$ and $k > 0$, if $currentLen > maxLen$, then set $maxLen = currentLen$ and $maxIndex = j - k$. In any case, reset $currentLen = 0$.

When the algorithm terminates, *maxIndex* and *maxLen* will hold the location and length of the longest prefix.

C-13.21) Hint Convert this problem to a noncircular pattern-matching problem.

C-13.21) Solution Generate a new text $T' = T[n-m \ldots n] + T[0 \ldots m]$. Run findKMP on T' and P.

C-13.22) Hint The failure function can now take advantage of the fact that it knows what does match in the mismatched location.

C-13.23) Hint You need to incorporate a failure function with the Boyer-Moore heuristics.

C-13.24) Hint Consider using a prefix trie.

C-13.25) Hint Start by building a suffix trie.

C-13.26) Hint Start by locating the leaf that corresponds to the end of the string.

C-13.26) Solution Locate the leaf that corresponds to the end of the string. While walking back to the root of the trie, delete every leaf encountered. The running time of this algorithm is $O(s)$ where s is the length of the string to be deleted.

C-13.27) Hint Start by locating the leaf that corresponds to the end of the string.

C-13.28) Hint Recall how you identify the branches of the suffix trie that can be compressed.

C-13.29) Hint Create some way of visualizing your standard trie so that you can verify that it is being constructed correctly.

C-13.30) Hint Create some way of visualizing your compressed trie so that you can verify that it is being constructed correctly.

C-13.31) Hint Create some way of visualizing your prefix trie so that you can verify that it is being constructed correctly.

C-13.32) Hint Build a prefix tree for X and a suffix tree for Y.

C-13.33) Hint First give as many quarters as possible.

C-13.33) Solution First give as many quarters as possible, then dimes, then nickels and finally pennies.

C-13.34) Hint Don't use normal denominations like you would find in a country on earth.

C-13.34) Solution If the denominations are \$0.25, \$0.24, \$0.01, then a greedy algorithm for making change for 48 cents would give 1 quarter and 23 pennies.

C-13.35) Hint We can use a greedy algorithm.

C-13.35) Solution We can use a greedy algorithm, which seeks to cover all the designated points on L with the fewest number of length-2 intervals (for such an interval is the distance one guard can protect). This greedy algorithm starts with x_0 and covers all the points that are within distance 2 of x_0. If x_i is the next uncovered point, then we repeat this same covering step starting from x_i. We then repeat this process until we have covered all the points in X.

C-13.36) Hint Consider using a greedy algorithm.

C-13.37) Hint You can rely on our implementation of trees and priority queues.

C-13.38) Hint Keep around extra information in the table for the dynamic programming algorithm.

C-13.39) Hint Anatjari should use a greedy algorithm.

C-13.39) Solution Anatjari should use a greedy algorithm. He should draw a line between every pair of watering holes that are within k miles of one another, for he can get from one to the other with one bottle of water. Anatjari should then use a path that uses the fewest number of lines and leads from his present position to a watering hole that is within with k miles of the other side. There can be no path with fewer stops, for he is including in his minimization all pairs of watering holes that can be reached with one bottle of water.

C-13.40) Hint Review the LCS algorithm.

C-13.41) Hint There is a surprising similarity between this problem and the matrix chain-product problem.

C-13.42) Hint Use a greedy algorithm.

C-13.43) Hint Review the LCS algorithm.

C-13.44) Hint The edit distance algorithm is a dynamic program based on the LCS problem.

C-13.45) Hint Use dynamic programming.

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

C-13.47) Hint Use dynamic programming.

Projects

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

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

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

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

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

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

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

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

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