provided we satisfy two constraints. First, we must never place a smaller disk on top of a larger disk. Second—and this is the non-standard part—***we must never move a disk directly from peg 0 to peg 2.***

Describe and analyze an algorithm to compute the exact number of moves required to move all n disks from peg 0 to peg 2, subject to the stated restrictions. For full credit, your algorithm should use only $O(\log n)$ arithmetic operations in the worst case. For the sake of analysis, assume that adding or multiplying two k-digit numbers requires $O(k)$ time. *[Hint: Matrices!]*

Splitting Sequences/Arrays

34. A ***basic arithmetic expression*** is composed of characters from the set $\{1, +, \times\}$ and parentheses. Almost every integer can be represented by more than one basic arithmetic expression. For example, all of the following basic arithmetic expression represent the integer 14:

$$1+1+1+1+1+1+1+1+1+1+1+1+1+1+1$$
$$((1+1)\times(1+1+1+1+1))+((1+1)\times(1+1))$$
$$(1+1)\times(1+1+1+1+1+1+1)$$
$$(1+1)\times(((1+1+1)\times(1+1))+1)$$

Describe and analyze an algorithm to compute, given an integer n as input, the minimum number of 1s in a basic arithmetic expression whose value is equal to n. The number of parentheses doesn't matter, just the number of 1s. For example, when $n = 14$, your algorithm should return 8, for the final expression above. The running time of your algorithm should be bounded by a small polynomial function of n.

35. Suppose you are given a sequence of integers separated by + and − signs; for example:

$$1+3-2-5+1-6+7$$

You can change the value of this expression by adding parentheses in different places. For example:

$$1+3-2-5+1-6+7=-1$$
$$(1+3-(2-5))+(1-6)+7=9$$
$$(1+(3-2))-(5+1)-(6+7)=-17$$

Describe and analyze an algorithm to compute, given a list of integers separated by + and − signs, the maximum possible value the expression

can take by adding parentheses. Parentheses must be used only to group additions and subtractions; in particular, do not use them to create implicit multiplication as in $1+3(-2)(-5)+1-6+7=33$.

36. Suppose you are given a sequence of integers separated by $+$ and $\times$ signs; for example:

$$1+3\times 2\times 0+1\times 6+7$$

You can change the value of this expression by adding parentheses in different places. For example:

$$(1+(3\times 2))\times 0+(1\times 6)+7=13$$
$$((1+(3\times 2\times 0)+1)\times 6)+7=19$$
$$(1+3)\times 2\times(0+1)\times(6+7)=208$$

(a) Describe and analyze an algorithm to compute the maximum possible value the given expression can take by adding parentheses, assuming all integers in the input are *positive*. *[Hint: This is easy.]*

(b) Describe and analyze an algorithm to compute the maximum possible value the given expression can take by adding parentheses, assuming all integers in the input are *non-negative*.

(c) Describe and analyze an algorithm to compute the maximum possible value the given expression can take by adding parentheses, with no restrictions on the input numbers.

Assume any arithmetic operation takes $O(1)$ time.

37. After graduating from Shampoo-Banana University, you decide to interview for a position at the Wall Street bank ***Long Live Boole***. The managing director of the bank, Eloob Egroeg, poses a 'solve-or-die' problems to each new employee, which they must solve within 24 hours. Those who fail to solve the problem are fired immediately!

Entering the bank for the first time, you notice that the employee offices are organized in a straight row, with a large T or F printed on the door of each office. Furthermore, between each adjacent pair of offices, there is a board marked by one of the symbols $\wedge, \vee$, or $\oplus$. When you ask about these arcane symbols, Eloob confirms that T and F represent the boolean values TRUE and FALSE, and the symbols on the boards represent the standard boolean operators AND, OR, and XOR. He also explains that these letters and symbols describe whether certain combinations of employees can work together successfully. At the start of any new project, Eloob hierarchically clusters his employees by adding parentheses to the sequence of symbols, to

obtain an unambiguous boolean expression. The project is successful if this parenthesized boolean expression evaluates to T.

For example, if the bank has three employees, and the sequence of symbols on and between their doors is $T \wedge F \oplus T$, there is exactly one successful parenthesization scheme: $(T \wedge (F \oplus T))$. However, if the list of door symbols is $F \wedge T \oplus F$, there is no way to add parentheses to make the project successful.

Eloob finally poses your solve-or-die interview question: Describe an algorithm to decide whether a given sequence of symbols can be parenthesized so that the resulting boolean expression evaluates to T. Your input is an array $S[0..2n]$, where $S[i] \in \{T, F\}$ when i is even, and $S[i] \in \{\vee, \wedge, \oplus\}$ when i is odd.

38. Every year, as part of its annual meeting, the Antarctican Snail Lovers of Upper Glacierville hold a Round Table Mating Race. Several high-quality breeding snails are placed at the edge of a round table. The snails are numbered in order around the table from 1 to n. During the race, each snail wanders around the table, leaving a trail of slime behind it. The snails have been specially trained never to fall off the edge of the table or to cross a slime trail, even their own. If two snails meet, they are declared a breeding pair, removed from the table, and whisked away to a romantic hole in the ground to make little baby snails. Note that some snails may never find a mate, even if the race goes on forever.

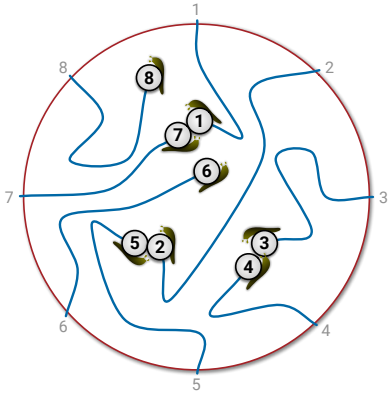


Figure 3.6. The end of a typical Antarctican SLUG race. Snails 6 and 8 never find mates. The organizers must pay $M[3,4] + M[2,5] + M[1,7]$.

For every pair of snails, the Antarctican SLUG race organizers have posted a monetary reward, to be paid to the owners if that pair of snails meets during the Mating Race. Specifically, there is a two-dimensional

array $M[1..n, 1..n]$ posted on the wall behind the Round Table, where $M[i, j] = M[j, i]$ is the reward to be paid if snails i and j meet.

Describe and analyze an algorithm to compute the maximum total reward that the organizers could be forced to pay, given the array M as input.

39. You have mined a large slab of marble from a quarry. For simplicity, suppose the marble slab is a rectangle measuring n inches in height and m inches in width. You want to cut the slab into smaller rectangles of various sizes—some for kitchen counter tops, some for large sculpture projects, others for memorial headstones. You have a marble saw that can make either horizontal or vertical cuts across any rectangular slab. At any time, you can query the spot price $P[x, y]$ of an x-inch by y-inch marble rectangle, for any positive integers x and y. These prices depend on customer demand, and people who buy marble counter tops are weird, so don't make any assumptions about them; in particular, larger rectangles may have significantly smaller spot prices. Given the array of spot prices and the integers m and n as input, describe an algorithm to compute how to subdivide an $n \times m$ marble slab to maximize your profit.

40. This problem asks you to design efficient algorithms to construct optimal binary search trees that satisfy additional balance constraints. Your input consists of a sorted array $A[1..n]$ of search keys and an array $f[1..n]$ of frequency counts, where $f[i]$ is the number of searches for $A[i]$. This is exactly the same cost function as described in Section 3.9. But now your task is to compute an optimal tree that satisfies some additional constraints.

    (a) ***AVL trees*** were the earliest self-balancing balanced binary search trees, first described in 1962 by Georgy Adelson-Velsky and Evgenii Landis. An AVL tree is a binary search tree where for every node v, the height of the left subtree of v and the height of the right subtree of v differ by at most one.

    Describe and analyze an algorithm to construct an optimal AVL tree for a given set of search keys and frequencies.

    (b) ***Symmetric binary B-trees*** are another self-balancing binary trees, first described by Rudolf Bayer; these are better known by the name ***red-black trees***, after a somewhat simpler reformulation by Leo Guibas and Bob Sedgwick in 1978. A red-black tree is a binary search tree with the following additional constraints:

    - Every node is either red or black.
    - Every red node has a black parent.
    - Every root-to-leaf path contains the same number of black nodes.

Describe and analyze an algorithm to construct an optimal red-black tree for a given set of search keys and frequencies.

(c) ***AA trees*** were proposed by proposed by Arne Andersson in 1993 and slightly simplified (and named) by Mark Allen Weiss in 2000. AA trees are also known as *left-leaning red-black trees*, after a symmetric reformulation (with different rebalancing algorithms) by Bob Sedgewick in 2006. An AA-tree is a red-black tree with one additional constraint:

- No left child is red.[24]

Describe and analyze an algorithm to construct an optimal AA-tree for a given set of search keys and frequencies.

41. Suppose you are given an $m \times n$ bitmap as an array $M[1..m, 1..n]$ of 0s and 1s. A *solid block* in M is a subarray of the form $M[i..i', j..i']$ in which all bits are equal. Suppose you want to decompose M into as few disjoint blocks as possible.

One natural recursive partitioning strategy is called a *guillotine subdivision*. If the entire bitmap M is a solid block, there is nothing to do. Otherwise, we cut M into two smaller bitmaps along a horizontal or vertical line, and then recursively decompose the two smaller bitmaps into solid blocks.

Any guillotine subdivision can be represented as a binary tree, where each internal node stores the position and orientation of a cut, and each leaf stores a single but 0 or 1 indicting the contents of the corresponding block. The *size* of a guillotine subdivision is the number of leaves in the corresponding binary tree (that is, the final number of solid blocks), and the *depth* of a guillotine subdivision is the depth of the corresponding binary tree.

(a) Describe and analyze an algorithm to compute a guillotine subdivision of M of minimum possible size.

(b) Show that a guillotine subdivision does *not* always yield a partition into the smallest number of solid blocks.

(c) Describe and analyze an algorithm to compute a guillotine subdivision for M with the smallest possible depth.

(d) Describe and analyze an algorithm to determine $M[i, j]$, given the tree representing a guillotine decomposition for M and two indices i and j.

[24] Sedgwick's reformulation requires that no *right* child is red. Whatever. Andersson and Sedgewick are all strangely silent about whether to measure angles clockwise or counterclockwise, whether Pluto is a planet, whether "lower rank" means "better" or "worse", or whether to fight a hundred duck-sized horses or a single horse-sized duck.

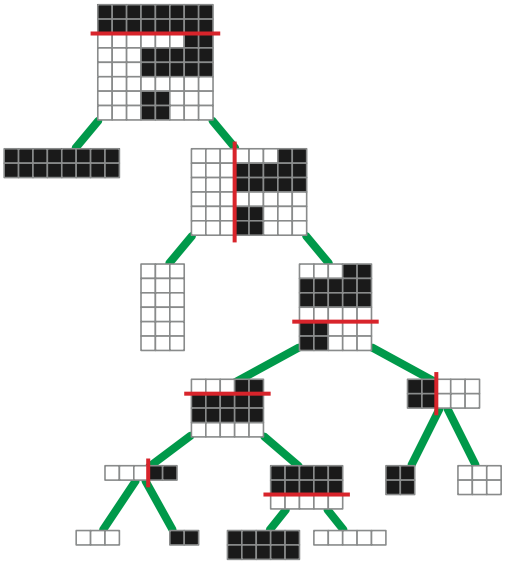


Figure 3.7. A guillotine subdivision with size 8 and depth 5.

(e) Define the *depth* of a pixel $M[i, j]$ in a guillotine subdivision to be the depth of the leaf that contains that pixel. Describe and analyze an algorithm to compute a guillotine subdivision for M such that the sum of the depths of the pixels as small as possible.

(f) Describe and analyze an algorithm to compute a guillotine subdivision for M such that the sum of the depths of the *black* pixels as small as possible.

♠42. Congratulations! You've been hired by the giant online bookstore DeNile ("Not just a river in Egypt!") to optimize their warehouse robots. Each book that DeNile sells has a unique ISBN (International Standard Book Number), which is just a numerical value. Each of DeNile's warehouses contains a long row of bins, each containing multiple copies of a single book. These bins are arranged in sorted order by ISBN; each bin's ISBN is printed on the front of the bin in machine-readable form. Books are retrieved from these bins by robots, which run along rails parallel to the row of bins.

DeNile does not maintain a list of which bins contain which ISBN numbers; that would be too simple! Instead, to retrieve a desired book, the robot must first find that book's bin using a binary search. Because the search requires physical motion by the robot, we can no longer assume that each step of the binary search requires $O(1)$ time. Specifically:

- The robot always starts at the "0th bin" (where the books are loaded into boxes to ship to customers).

- Moving the robot from the ith bin to the jth bin requires $\alpha|i-j|$ seconds for some constant α.
- The robot must be directly in front of a bin in order to read that bin's ISBN. Reading an ISBN requires β seconds, for some constant β.
- Reversing the robot's direction of motion (from increasing to decreasing or vice versa) requires γ additional seconds, for some constant γ.
- When the robot finds the target bin, it extracts one book from that bin and returns to "the 0th bin".

Design and analyze an algorithm to compute a binary search tree over the bins that minimizes the total time the robot spends searching for books. Your input is an array $f[1..n]$ of integers, where $f[i]$ is the number of times that the robot will be asked to retrieve a book from the ith bin, along with the time parameters α, β, and γ.

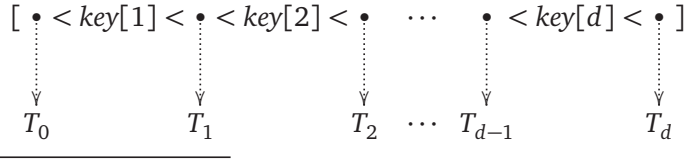
♠43. A standard method to improve the cache performance of search trees is to pack more search keys and subtrees into each node. A ***B-tree*** is a rooted tree in which each internal node stores up to B keys and pointers to up to $B+1$ children, each the root of a smaller B-tree. Specifically, each node v stores three fields:

- a positive integer $v.d \le B$,
- a *sorted* array $v.key[1..v.d]$, and
- an array $v.child[0..v.d]$ of child pointers.

In particular, the number of child pointers is always exactly one more than the number of keys.[25]

Each pointer $v.child[i]$ is either NULL or a pointer to the root of a B-tree whose keys are all larger than $v.key[i]$ and smaller than $v.key[i+1]$. In particular, all keys in the leftmost subtree $v.child[0]$ are smaller than $v.key[1]$, and all keys in the rightmost subtree $v.child[v.d]$ are larger than $v.key[v.d]$.

Intuitively, you should have the following picture in mind:



[25]Normally, B-trees are required to satisfy two additional constraints, which guarantee a worst-case search cost of $O(\log_B n)$: Every leaf must have exactly the same depth, and every node except possibly the root must contain at least $B/2$ keys. However, in this problem, we are not interested in optimizing the *worst-case* search cost, but rather the *total* cost of a sequence of searches, so we will not impose these additional constraints.

Here T_i is the subtree pointed to by $child[i]$.

The ***cost*** of searching for a key x in a B-tree is the number of nodes in the path from the root to the node containing x as one of its keys. A 1-tree is just a standard binary search tree.

Fix an arbitrary positive integer $B > 0$. (I suggest $B = 8$.) Suppose your are given a sorted array $A[1, \ldots, n]$ of search keys and a corresponding array $F[1, \ldots, n]$ of frequency counts, where $F[i]$ is the number of times that we will search for $A[i]$. Your task is to describe and analyze an efficient algorithm to find a B-tree that minimizes the total cost of searching for the given keys with the given frequencies.

(a) Describe a polynomial-time algorithm for the special case $B = 2$.

(b) Describe an algorithm for arbitrary B that runs in $O(n^{B+c})$ time for some fixed integer c.

♥(c) Describe an algorithm for arbitrary B that runs in $O(n^c)$ time for some fixed integer c that does *not* depend on B.

44. A string w of parentheses `(` and `)` and brackets `[` and `]` is ***balanced*** if it satisfies one of the following conditions:

- w is the empty string.
- $w =$ `(`x`)` for some balanced string x
- $w =$ `[`x`]` for some balanced string x
- $w = xy$ for some balanced strings x and y

For example, the string

$$w = \texttt{([()][]())[()()]()}$$

is balanced, because $w = xy$, where

$$x = \texttt{([()] [] ())} \qquad \text{and} \qquad y = \texttt{[() ()] ()}.$$

(a) Describe and analyze an algorithm to determine whether a given string of parentheses and brackets is balanced.

(b) Describe and analyze an algorithm to compute the length of a longest balanced subsequence of a given string of parentheses and brackets.

(c) Describe and analyze an algorithm to compute the length of a shortest balanced supersequence of a given string of parentheses and brackets.

(d) Describe and analyze an algorithm to compute the minimum edit distance from a given string of parentheses and brackets to a balanced string of parentheses and brackets.

♥(e) Describe and analyze an algorithm to compute the longest common balanced subsequence of two given strings of parentheses and brackets.

♥(f) Describe and analyze an algorithm to compute the longest palindromic balanced subsequence of a given string of parentheses and brackets.

♥(g) Describe and analyze an algorithm to compute the longest common palindromic balanced subsequence (whew!) of two given strings of parentheses and brackets.

For each problem, your input is an array $w[1..n]$, where $w[i] \in \{(,), [,]\}$ for every index i. (You may prefer to use different symbols instead of parentheses and brackets—for example, L, R, l, r or ◁, ▷, ◀, ▶—but please tell your grader what symbols you're using!)

♥45. Congratulations! Your research team has just been awarded a $50M multi-year project, jointly funded by DARPA, Google, and McDonald's, to produce DWIM: The first compiler to read programmers' minds! Your proposal and your numerous press releases all promise that DWIM will automatically correct errors in any given piece of code, while modifying that code as little as possible. Unfortunately, now it's time to start actually making the damn thing work.

As a warmup exercise, you decide to tackle the following necessary subproblem. Recall that the *edit distance* between two strings is the minimum number of single-character insertions, deletions, and replacements required to transform one string into the other. An *arithmetic expression* is a string w such that

- w is a string of one or more decimal digits,
- $w = (x)$ for some arithmetic expression x, or
- $w = x \diamond y$ for some arithmetic expressions x and y and some binary operator $\diamond$.

Suppose you are given a string of tokens from the alphabet $\{\#, \diamond, (,)\}$, where # represents a decimal digit and $\diamond$ represents a binary operator. Describe and analyze an algorithm to compute the minimum edit distance from the given string to an arithmetic expression.

46. Ribonucleic acid (RNA) molecules are long chains of millions of nucleotides or *bases* of four different types: adenine (A), cytosine (C), guanine (G), and uracil (U). The *sequence* of an RNA molecule is a string $b[1..n]$, where each character $b[i] \in \{A, C, G, U\}$ corresponds to a base. In addition to the chemical bonds between adjacent bases in the sequence, hydrogen bonds can form between certain pairs of bases. The set of bonded base pairs is called the *secondary structure* of the RNA molecule.

We say that two base pairs (i, j) and (i', j') with $i < j$ and $i' < j'$ ***overlap*** if $i < i' < j < j'$ or $i' < i < j' < j$. In practice, most base pairs are non-overlapping. Overlapping base pairs create so-called *pseudoknots* in the secondary structure, which are essential for some RNA functions, but are more difficult to predict.

Suppose we want to predict the best possible secondary structure for a given RNA sequence. We will adopt a drastically simplified model of secondary structure:

- Each base can bond with at most one other base.
- Only A–U pairs and C–G pairs can bond.
- Pairs of the form $(i, i+1)$ and $(i, i+2)$ cannot bond.
- Bonded base pairs cannot overlap.

The last (and least realistic) restriction allows us to visualize RNA secondary structure as a sort of fat tree, as shown below.

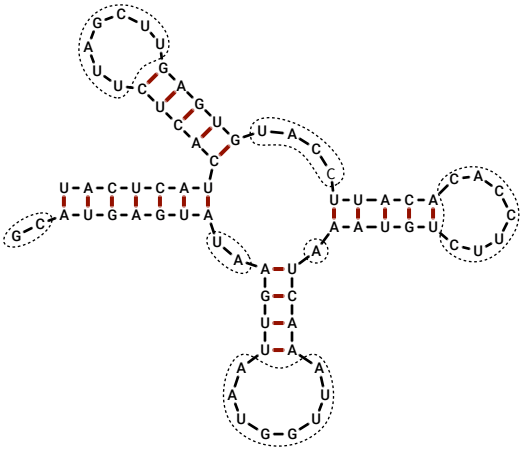


Figure 3.8. Example RNA secondary structure with 21 bonded base pairs, indicated by heavy red lines. Gaps are indicated by dotted curves. This structure has score $2^2 + 2^2 + 8^2 + 1^2 + 7^2 + 4^2 + 7^2 = 187$.

(a) Describe and analyze an algorithm that computes the maximum possible *number* of bonded base pairs in a secondary structure for a given RNA sequence.

(b) A *gap* in a secondary structure is a maximal substring of unpaired bases. Large gaps lead to chemical instabilities, so secondary structures with smaller gaps are more likely. To account for this preference, let's define the *score* of a secondary structure to be the sum of the *squares* of the gap lengths; see Figure 3.8. (This score function is utterly fictional; real RNA structure prediction requires *much* more complicated scoring functions.)

Describe and analyze an algorithm that computes the minimum possible score of a secondary structure for a given RNA sequence.

♣47. (a) Describe and analyze an efficient algorithm to determine, given a string w and a regular expression R, whether $w \in L(R)$.

(b) *Generalized* regular expressions allow the binary operator $\cap$ (intersection) and the unary operator $\neg$ (complement), in addition to the usual $\bullet$ (concatenation), $+$ (or), and * (Kleene closure) operators. NFA constructions and Kleene's theorem imply that any generalized regular expression E represents a regular language $L(E)$.

Describe and analyze an efficient algorithm to determine, given a string w and a generalized regular expression E, whether $w \in L(E)$.

In both problems, assume that you are actually given a parse tree for the (generalized) regular expression, not just a string.

Trees and Subtrees

48. You've just been appointed as the new organizer of Giggle, Inc.'s annual mandatory holiday party. The employees at Giggle are organized into a strict hierarchy, that is, a tree with the company president at the root. The all-knowing oracles in Human Resources have assigned a real number to each employee measuring how "fun" the employee is. In order to keep things social, there is one restriction on the guest list: an employee cannot attend the party if their immediate supervisor is also present. On the other hand, the president of the company *must* attend the party, even though she has a negative fun rating; it's her company, after all. Give an algorithm that makes a guest list for the party that maximizes the sum of the "fun" ratings of the guests.

49. Since so few people came to last year's holiday party, the president of Giggle, Inc. decides to give each employee a present instead this year. Specifically, each employee must receive on the three gifts: (1) an all-expenses-paid six-week vacation anywhere in the world, (2) an all-the-pancakes-you-can-eat breakfast for two at Jumping Jack Flash's Flapjack Stack Shack, or (3) a burning paper bag full of dog poop. Corporate regulations prohibit any employee from receiving exactly the same gift as his/her direct supervisor. Any employee who receives a better gift than his/her direct supervisor will almost certainly be fired in a fit of jealousy.

As Giggle, Inc.'s official party czar, it's *your* job to decide which gift each employee receives. Describe an algorithm to distribute gifts so that the minimum number of people are fired. Yes, you may send the president a flaming bag of dog poop.

More formally, you are given a rooted tree T, representing the company hierarchy, and you want to label the nodes of T with integers 1, 2, or 3, so

that every node has a different label from its parent. The *cost* of an labeling is the number of nodes with smaller labels than their parents. See Figure 3.9 for an example. Describe and analyze an algorithm to compute the minimum-cost labeling of T.

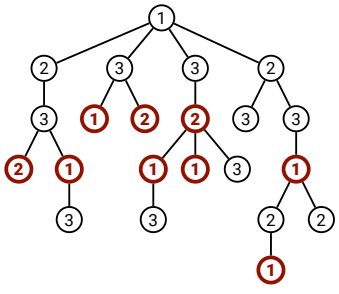


Figure 3.9. A tree labeling with cost 9. The nine bold nodes have smaller labels than their parents. This is *not* the optimal labeling for this tree.

50. After the Flaming Dog Poop Holiday Debacle, you were strongly encouraged to seek other employment, and so you left Giggle for rival company Twitbook. Unfortunately, the new president of Twitbook just decided to imitate Giggle by throwing her own holiday party, and in light of your past experience, appointed you as the official party organizer. The president demands that you invite exactly k employees, including the president herself, and everyone who is invited is required to attend. Yeah, that'll be fun.

    Just like at Giggle, employees at Twitbook are organized into a strict hierarchy: a tree with the company president at the root. The all-knowing oracles in Human Resources have assigned a real number to each employee indicating the *awkwardness* of inviting both that employee and their immediate supervisor; a negative value indicates that the employee and their supervisor actually like each other. Your goal is to choose a subset of exactly k employees to invite, so that the total awkwardness of the resulting party is as small as possible. For example, if the guest list does not include both an employee and their immediate supervisor, the total awkwardness is zero. The input to your algorithm is the tree T, the integer k, and the awkwardness of each node in T.

    (a) Describe an algorithm that computes the total awkwardness of the least awkward subset of k employees, assuming the company hierarchy is described by a *binary* tree. That is, assume that each employee directly supervises at most two others.

    ♥(b) Describe an algorithm that computes the total awkwardness of the least awkward subset of k employees, with no restrictions on the company hierarchy.

51. Suppose we need to broadcast a message to all the nodes in a rooted tree. Initially, only the root node knows the message. In a single round, any node that knows the message can forward it to at most one of its children. See Figure 3.10 for an example.

    (a) Design an algorithm to compute the minimum number of rounds required to broadcast the message to all nodes in a *binary* tree.

    ♦(b) Design an algorithm to compute the minimum number of rounds required to broadcast the message to all nodes in an *arbitrary* rooted tree. *[Hint: You may find techniques in the next chapter useful to prove your algorithm is correct, even though it's not a greedy algorithm.]*

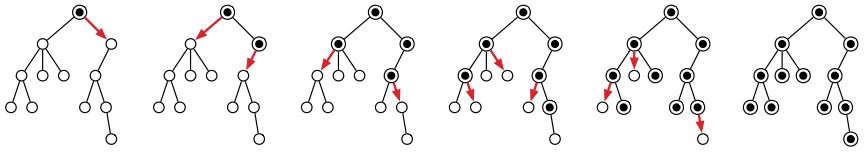


Figure 3.10. A message being distributed through a tree in five rounds.

52. One day, Alex got tired of climbing in a gym and decided to take a very large group of climber friends outside to climb. The climbing area where they went, had a huge wide boulder, not very tall, with various marked hand and foot holds. Alex quickly determined an "allowed" set of moves that her group of friends can perform to get from one hold to another.

    The overall system of holds can be described by a rooted tree T with n vertices, where each vertex corresponds to a hold and each edge corresponds to an allowed move between holds. The climbing paths converge as they go up the boulder, leading to a unique hold at the summit, represented by the root of T.[26]

    Alex and her friends (who are all excellent climbers) decided to play a game, where as many climbers as possible are simultaneously on the boulder and each climber needs to perform a sequence of *exactly* k moves. Each climber can choose an arbitrary hold to start from, and all moves must move away from the ground. Thus, each climber traces out a path of k edges in the tree T, all directed toward the root. However, no two climbers are allowed to touch the same hold; the paths followed by different climbers cannot intersect at all.

    Describe and analyze an efficient algorithm to compute the maximum number of climbers that can play this game. More formally, you are given a rooted tree T and an integer k, and you want to find the largest possible

[26] Q: Why do computer science professors think trees have their roots at the top?
A: Because they've never been outside!

number of disjoint paths in T, where each path has length k. Do ***not*** assume that T is a binary tree. For example, given the tree T below and $k = 3$ as input, your algorithm should return the integer 8.

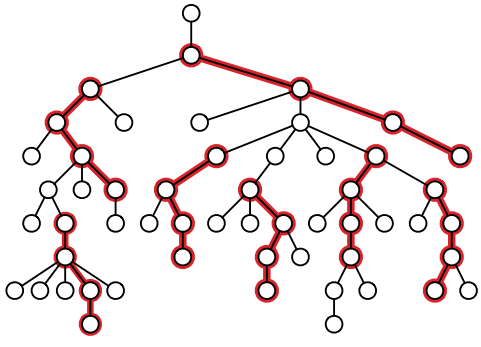


Figure 3.11. Seven disjoint paths of length $k = 3$. This is *not* the largest such set of paths in this tree.

53. Let T be a rooted binary tree with n vertices, and let $k \le n$ be a positive integer. We would like to mark k vertices in T so that every vertex has a nearby marked ancestor. More formally, we define the *clustering cost* of any subset K of vertices as

$$cost(K) = \max_v cost(v, K),$$

where the maximum is taken over all vertices v in the tree, and $cost(v, K)$ is the distance from v to its nearest ancestor in K:

$$cost(v, K) = \begin{cases} 0 & \text{if } v \in K \\ \infty & \text{if } v \text{ is the root of } T \text{ and } v \notin K \\ 1 + cost(parent(v)) & \text{otherwise} \end{cases}$$

In particular, $cost(K) = \infty$ if K excludes the root of T.

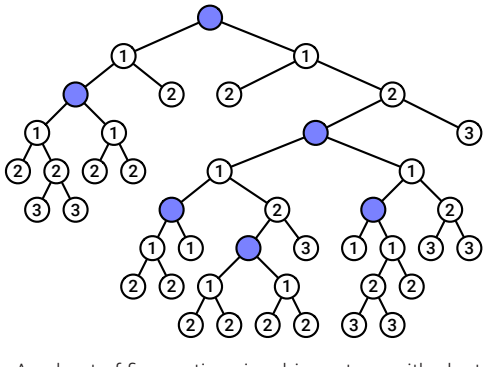


Figure 3.12. A subset of five vertices in a binary tree, with clustering cost 3.

♥(a) Describe a dynamic programming algorithm to compute, given the tree T and an integer k, the minimum clustering cost of any subset of k vertices in T. For full credit, your algorithm should run in $O(n^2k^2)$ time.

(b) Describe a dynamic programming algorithm to compute, given the tree T and an integer r, the size of the smallest subset of vertices whose clustering cost is at most r. For full credit, your algorithm should run in $O(nr)$ time.

(c) Show that your solution for part (b) implies an algorithm for part (a) that runs in $O(n^2 \log n)$ time.

54. This question asks you to find efficient algorithms to compute the ***largest common rooted subtree*** of two given rooted trees. Recall that a *rooted tree* is a connected acyclic graph with a designated node called the root. A rooted subtree of a rooted tree consists of an arbitrary node and all its descendants. The precise definition of "common" depends on which pairs of rooted trees we consider isomorphic.

(a) Recall that a *binary* tree is a rooted tree in which every node has a (possibly empty) *left* subtree and a (possibly empty) *right* subtree. Two binary trees are isomorphic if and only if they are both empty, or their left subtrees are isomorphic and their right subtrees are isomorphic. Describe an algorithm to find the largest common *binary* subtree of two given *binary* trees.

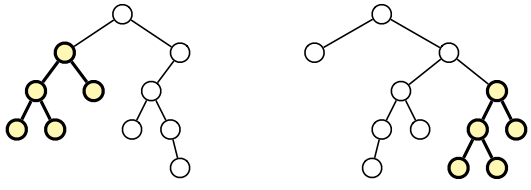


Figure 3.13. Two binary trees, with their largest common (rooted) subtree emphasized.

(b) In an *ordered* rooted tree, each node has a *sequence* of children, which are the roots of ordered rooted subtrees. Two ordered rooted trees are isomorphic if they are both empty, or if their ith subtrees are isomorphic for every index i. Describe an algorithm to find the largest common ordered subtree of two ordered trees T_1 and T_2.

♦♥(c) In an *unordered* rooted tree, each node has an unordered *set* of children, which are the roots of unordered rooted subtrees. Two unordered rooted trees are isomorphic if they are both empty, or the subtrees of each root *can be ordered so that* their ith subtrees are isomorphic for every index i. Describe an algorithm to find the largest common unordered subtree of two unordered trees T_1 and T_2.

55. This question asks you to find efficient algorithms to compute optimal subtrees in *unrooted* trees—connected acyclic undirected graphs. A *subtree* of an unrooted tree is any connected subgraph.

    (a) Suppose you are given an unrooted tree T with weights on its *edges*, which may be positive, negative, or zero. Describe an algorithm to find a *path* in T with maximum total weight.

    (b) Suppose you are given an unrooted tree T with weights on its *vertices*, which may be positive, negative, or zero. Describe an algorithm to find a *subtree* of T with maximum total weight. *[This was a 2016 Google interview question.]*

    (c) Let T_1 and T_2 be arbitrary *ordered* unrooted trees, meaning that the neighbors of every node have a well-defined cyclic order. Describe an algorithm to find the largest common *ordered* subtree of T_1 and T_2.

♦♥(d) Let T_1 and T_2 be arbitrary *unordered* unrooted trees. Describe an algorithm to find the largest common *unordered* subtree of T_1 and T_2.

56. ***Rooted minors*** of rooted trees are a natural generalization of subsequences. A rooted minor of a rooted tree T is any tree obtained by *contracting* one or more edges. When we contract an edge $u \to v$, where u is the parent of v, the children of v become new children of u and then v is deleted. In particular, the root of T is also the root of every rooted minor of T.

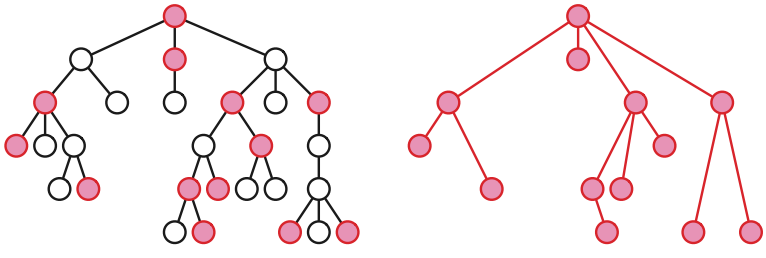


Figure 3.14. A rooted tree and one of its rooted minors.

    (a) Let T be a rooted tree with labeled nodes. We say that T is *boring* if, for each node x, all children of x have the same label; children of different nodes may have different labels. Describe an algorithm to find the largest boring rooted minor of a given labeled rooted tree.

    (b) Suppose we are given a rooted tree T whose nodes are labeled with numbers. Describe an algorithm to find the largest *heap-ordered rooted minor* of T. That is, your algorithm should return the largest rooted minor M such that every node in M has a smaller label than its children in M.

(c) Suppose we are given a *binary* tree T whose nodes are labeled with numbers. Describe an algorithm to find the largest *binary-search-ordered rooted minor* of T. That is, your algorithm should return a rooted minor M such that every node in M has at most two children, and an inorder traversal of M is an increasing subsequence of an inorder traversal of T.

(d) Recall that a rooted tree is *ordered* if the children of each node have a well-defined left-to-right order. Describe an algorithm to find the largest binary-search-ordered minor of an *arbitrary* ordered tree T whose nodes are labeled with numbers. Again, the left-to-right order of nodes in M should be consistent with their order in T.

♥(e) Describe an algorithm to find the largest common *ordered* rooted minor of two *ordered* labeled rooted trees.

♦♥(f) Describe an algorithm to find the largest common *unordered* rooted minor of two *unordered* labeled rooted trees. *[Hint: Combine dynamic programming with maximum flows.]*