This file contains the exercises, hints, and solutions for Chapter 12 of the book "Introduction to the Design and Analysis of Algorithms," 2nd edition, by A. Levitin. The problems that might be challenging for at least some students are marked by ▷; those that might be difficult for a majority of students are marked by ►.

Exercises 12.1

1. a. Continue the backtracking search for a solution to the four-queens problem, which was started in this section, to find the second solution to the problem.

   b. Explain how the board's symmetry can be used to find the second solution to the four-queens problem.

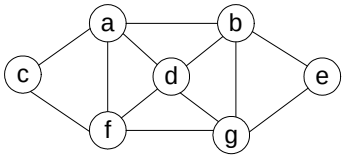
2. a. Which is the *last* solution to the five-queens problem found by the backtracking algorithm?

   b. Use the board's symmetry to find at least four other solutions to the problem.

3. a. Implement the backtracking algorithm for the n-queens problem in the language of your choice. Run your program for a sample of n values to get the numbers of nodes in the algorithm's state-space trees. Compare these numbers with the numbers of candidate solutions generated by the exhaustive-search algorithm for this problem.

   b. For each value of n for which you run your program in part a, estimate the size of the state-space tree by the method described in Section 12.1 and compare the estimate with the actual number of nodes you obtained.

4. Apply backtracking to the problem of finding a Hamiltonian circuit in the following graph.



5. Apply backtracking to solve the 3-coloring problem for the graph in Figure 12.3a.

6. Generate all permutations of {1, 2, 3, 4} by backtracking.

7. a. Apply backtracking to solve the following instance of the subset-sum problem: $S = \{1, 3, 4, 5\}$ and $d = 11$.

   b. Will the backtracking algorithm work correctly if we use just one of the two inequalities to terminate a node as nonpromising?

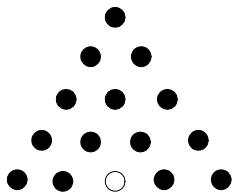
8. The general template for backtracking algorithms, which was given in Section 12.1, works correctly only if no solution is a prefix to another solution to the problem. Change the pseudocode to work correctly for such problems as well.

9. Write a program implementing a backtracking algorithm for

   a. the Hamiltonian circuit problem.

   b. the m-coloring problem.

10. *Puzzle pegs* This puzzle-like game is played on a board with 15 small holes arranged in an equilateral triangle. In an initial position, all but one of the holes are occupied by pegs, as in the example shown below. A legal move is a jump of a peg over its immediate neighbor into an empty square opposite; the jump removes the jumped-over neighbor from the board.



   Design and implement a backtracking algorithm for solving the following versions of this puzzle.

   a. Starting with a given location of the empty hole, find a shortest sequence of moves that eliminates 14 pegs with no limitations on the final position of the remaining peg.

   b. Starting with a given location of the empty hole, find a shortest sequence of moves that eliminates 14 pegs with the remaining peg at the empty hole of the initial board.

Hints to Exercises 12.1

1. a. Resume the algorithm by backtracking from the first solution's leaf.

   b. How can you get the second solution from the first one by exploiting a symmetry of the board?

2. Think backtracking applied backward.

3. a. Take advantage of the general template for backtracking algorithms. You will have to figure out how to check whether no two queens attack each other in a given placement of the queens.

   To make your comparison with an exhaustive-search algorithm easier, you may consider the version that finds all the solutions to the problem without taking advantage of the symmetries of the board. Also note that an exhaustive-search algorithm can try either all placements of n queens on n distinct squares of the n-by-n board, or only placements of the queens in different rows, or only placements in different rows and different columns.

   b. Although it is interesting to see how accurate such an estimate is for a single random path, you would want to compute the average of several of them to get a reasonably accurate estimate of the tree size.

4. Another instance of this problem is solved in the section.

5. Note that without loss of generality, you can assume that vertex a is colored with color 1 and hence associate this information with the root of the state-space tree.

6. This application of backtracking is quite straightforward.

7. a. Another instance of this problem is solved in the section.

   b. Some of the nodes will be deemed promising when, in fact, they are not.

8. A minor change in the template given does the job.

9. n/a

10. Make sure that your program does not duplicates tree nodes for the same board position. And, of course, if a given instance of the puzzle does not have a solution, your program should issue a message to that effect.

Solutions to Exercises 12.1

1. a. The second solution reached by the backtracking algorithm is

	1	2	3	4
1			Q	
2	Q			
3				Q
4		Q		

b. The second solution can be found by reflection of the first solution with respect to the vertical line through the middle of the board:

	1	2	3	4
1		Q		
2				Q
3	Q			
4			Q	

$\Longrightarrow$

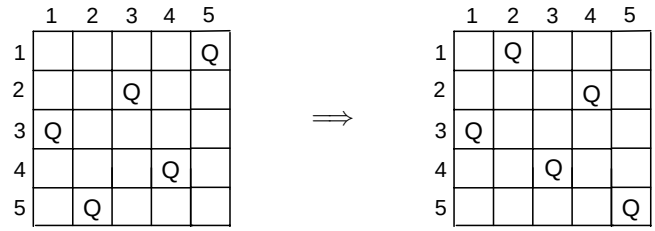
	1	2	3	4
1			Q	
2	Q			
3				Q
4		Q		

2. a. The last solution to the 5-queens puzzle found by backtracking last is

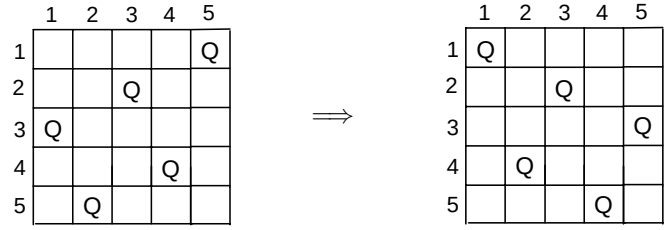
	1	2	3	4	5
1					Q
2			Q		
3	Q				
4				Q	
5		Q			

It is obtained by placing each queen in the last possible column of the queen's row, staring with the first queen and ending with the fifth one. (Any placement with the first queen in column $j < 5$ is found by the backtracking algorithm before a placement with the first queen in column 5, and so on.)

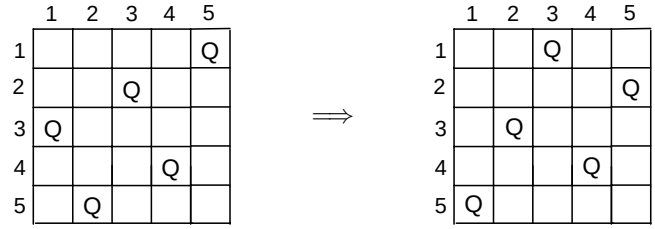
b. The solution obtained using the board's symmetry with respect to its middle row is



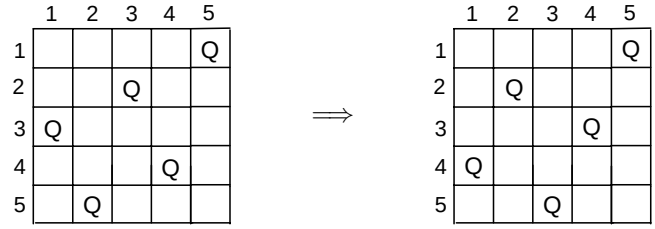
The solution obtained using the board's symmetry with respect to its middle column is



The solution obtained using the board's symmetry with respect to its main north-west–south-east diagonal is

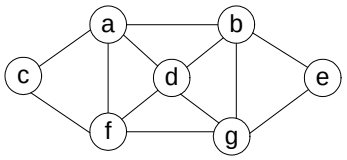


The solution obtained using the board's symmetry with respect to its main south-west–north-east diagonal is

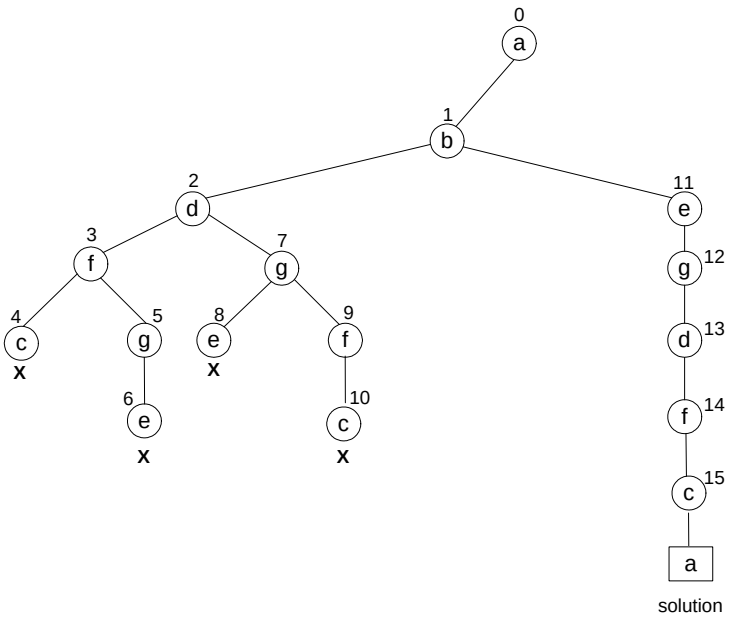


3. For a discussion of efficient implementations of the backtracking algorithm for the n-queens problem, see Timothy Rolfe, "Optimal Queens," *Dr. Dobb's Journal,* May 2005, pp. 32–37.

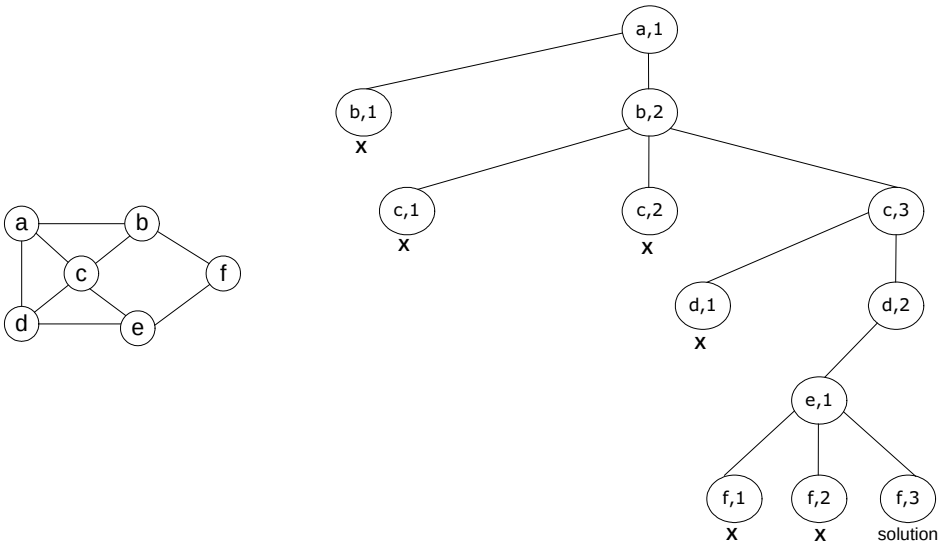
4. Finding a Hamiltonian circuit in the graph



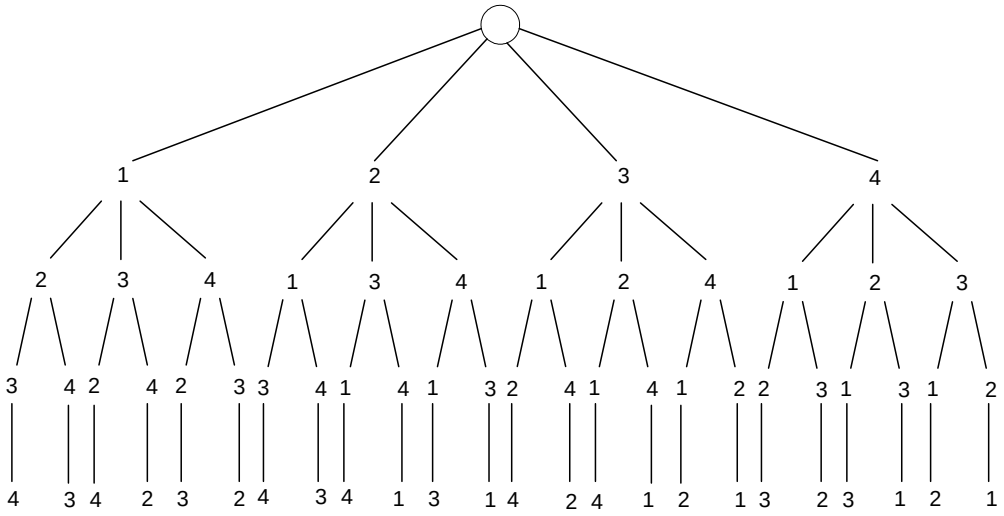
by backtracking yields the following state-space tree:



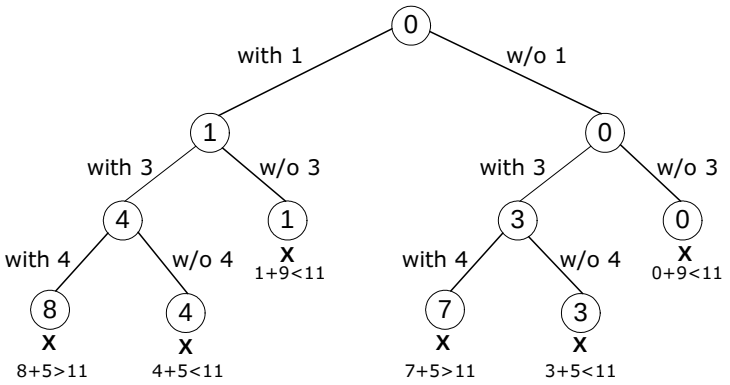
5. Here are the graph and a state-space tree for solving the 3-coloring problem for it using backtracking:



6. Here is a state-space tree for generating all permutations of {1, 2, 3, 4} by backtracking:



7. a. Here is a state-space tree for the given instance of the subset-sum problem: $S = \{1, 3, 4, 5\}$ and $d = 11$:



There is no solution to this instance of the problem.

b. The algorithm will still work correctly but the state-space tree will contain more nodes than necessary.

8. Eliminate **else** from the pseudocode.

9. n/a

10. n/a

Exercises 12.2

1. What data structure would you use to keep track of live nodes in a best-first branch-and-bound algorithm?

2. Solve the same instance of the assignment problem as the one solved in the section by the best-first branch-and-bound algorithm with the bounding function based on matrix columns rather than rows.

3. a. Give an example of the best-case input for the branch-and-bound algorithm for the assignment problem.

   b. In the best case, how many nodes will be in the state-space tree of the branch-and-bound algorithm for the assignment problem?

4. Write a program for solving the assignment problem by the branch-and-bound algorithm. Experiment with your program to determine the average size of the cost matrices for which the problem is solved in under one minute on your computer.

5. Solve the following instance of the knapsack problem by the branch-and-bound algorithm

item	weight	value
1	10	\$100
2	7	\$63
3	8	\$56
4	4	\$12

   $W = 16$

6. a. Suggest a more sophisticated bounding function for solving the knapsack problem than the one used in the section.

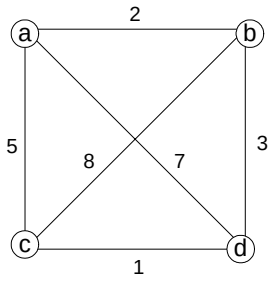
   b. Use your bounding function in the branch-and-bound algorithm applied to the instance of Problem 5.

7. Write a program to solve the knapsack problem with the branch-and-bound algorithm.

8. a. Prove the validity of the lower bound given by formula (12.2) for instances of the traveling salesman problem with integer symmetric matrices of intercity distances.

   b. How would you modify lower bound (12.2) for nonsymmetric distance matrices?

9. Apply the branch-and-bound algorithm to solve the traveling salesman problem for the following graph.



(We solved this problem by exhaustive search in Section 3.4.)

10. As a research project, write a report on how state-space trees are used for programming such games as chess, checkers, and tic-tac-toe. The two principal algorithms you should read about are the minimax algorithm and alpha-beta pruning.

Hints to Exercises 12.2

1. What operations does a best-first branch-and-bound algorithm perform on the live nodes of its state-space tree?

2. Use the smallest numbers selected from the columns of the cost matrix to compute the lower bounds. With this bounding function, it's more logical to consider four ways to assign job 1 for the nodes on the first level of the tree.

3. a. Your answer should be an n-by-n matrix with a simple structure making the algorithm work the fastest.

   b. Sketch the structure of the state-space tree for your answer to part (a).

4. n/a

5. A similar problem is solved in the section.

6. Take into account more than a single item from those not included in the subset under consideration.

7. n/a

8. A Hamiltonian circuit must have exactly two edges incident with each vertex of the graph.

9. A similar problem is solved in the section.

10. n/a

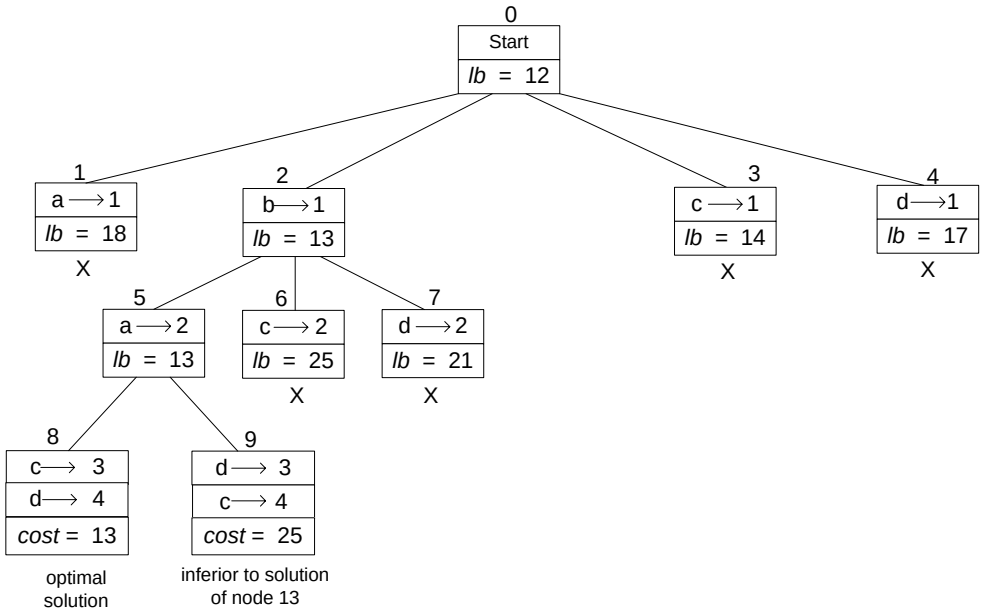
Solutions to Exercises 12.2

1. The heap and min-heap for maximization and minimization problems, respectively.

2. The instance discussed in the section is specified by the matrix

$$C = \begin{array}{cccc} \text{job 1} & \text{job 2} & \text{job 3} & \text{job 4} \\ 9 & 2 & 7 & 8 \\ 6 & 4 & 3 & 7 \\ 5 & 8 & 1 & 8 \\ 7 & 6 & 9 & 4 \end{array} \begin{array}{l} \\ \text{person } a \\ \text{person } b \\ \text{person } c \\ \text{person } d \end{array}$$

Here is the state-space tree in question:



The optimal assignment is $b \to 1$, $a \to 2$, $c \to 3$, $d \to 4$.

3. a. An n-by-n matrix whose elements are all the same is one such example.

   b. In the best case, the state-space tree will have just one node developed on each of its levels. Accordingly, the total number of nodes will be

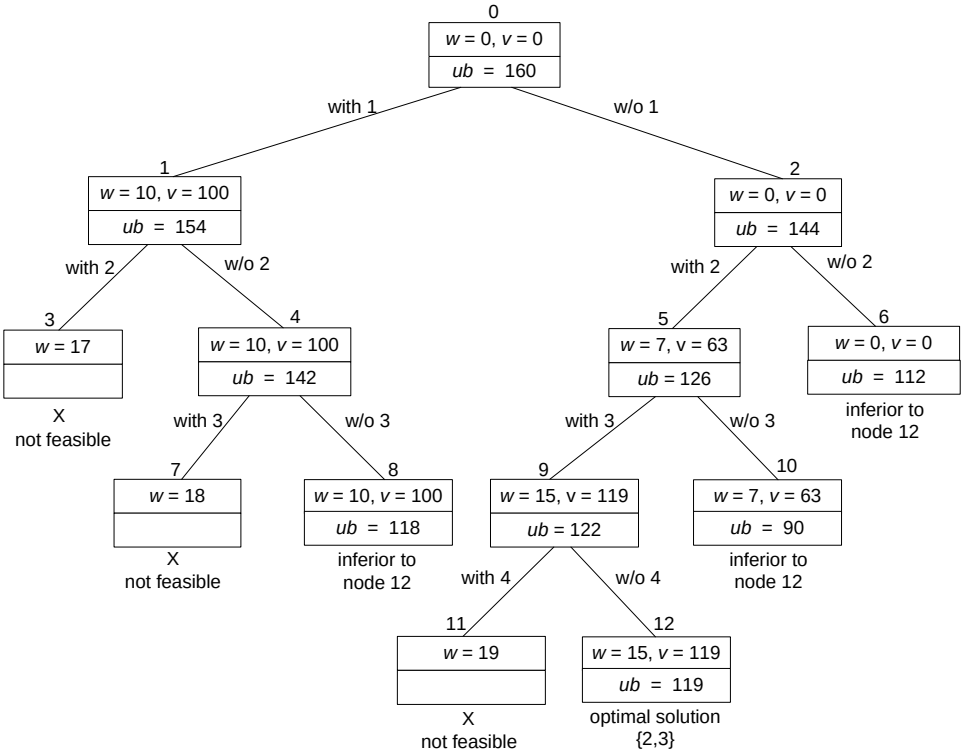
$$1 + n + (n-1) + \ldots + 2 = \frac{n(n+1)}{2}.$$

4. n/a

5. The instance in question is specified by the following table, in which the items are listed in nonincreasing order of their value-to-weight ratios:

item	weight	value	
1	10	\$100	
2	7	\$63	$W = 16$
3	8	\$56	
4	4	\$12	

Here is the state-space tree of the best-first branch-and-bound algorithm with the simple bounding function indicated in the section:

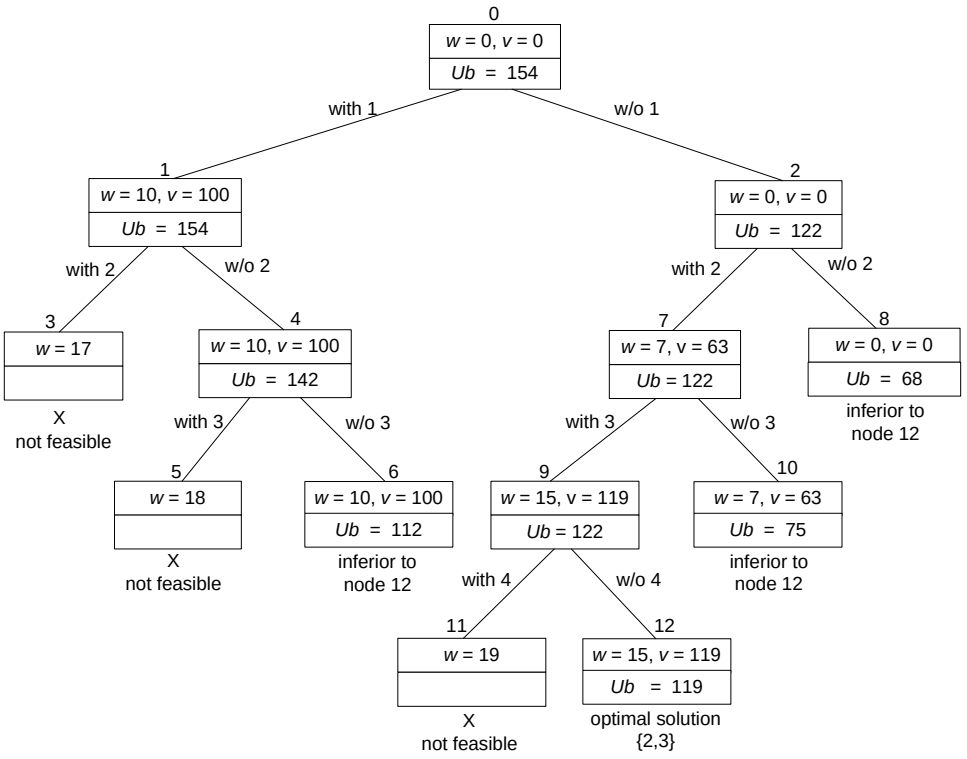


The found optimal solution is {item 2, item 3} of value \$119.

6. a. We assume that the items are sorted in nonincreasing order of their efficiencies: $v_1/w_1 \geq ... \geq v_n/w_n$ and, hence, new items are added in this order. Consider a subset $S = \{\text{item } i_1, ..., \text{item } i_k\}$ represented by a node in a branch-and-bound tree; the total weight and value of the items in S are $w(S) = w_{i_1} + ... + w_{i_k}$ and $v(S) = v_{i_1} + ... + v_{i_k}$, respectively. We can compute the upper bound $Ub(S)$ on the value of any subset that can be obtained by adding items to S as follows. We'll add to $v(S)$ the consecutive values of the items not included in S as long as the total weight doesn't exceed the knapsack's capacity. If we do encounter an item that violates this requirement, we'll determine its fraction needed to fill the knapsack to full capacity and use the product of this fraction and that item's efficiency as the last addend in computing $Ub(S)$. For example, for the instance of Problem 5, the upper bound for the root of the tree will be computed as follows: $Ub = 100 + (6/7)63 = 154$.

   b. The instance is specified by the following data, in which the items are listed in nonincreasing order of their value to weight ratios:

item	weight	value	
1	10	\$100	
2	7	\$63	$W = 16$
3	8	\$56	
4	4	\$12	



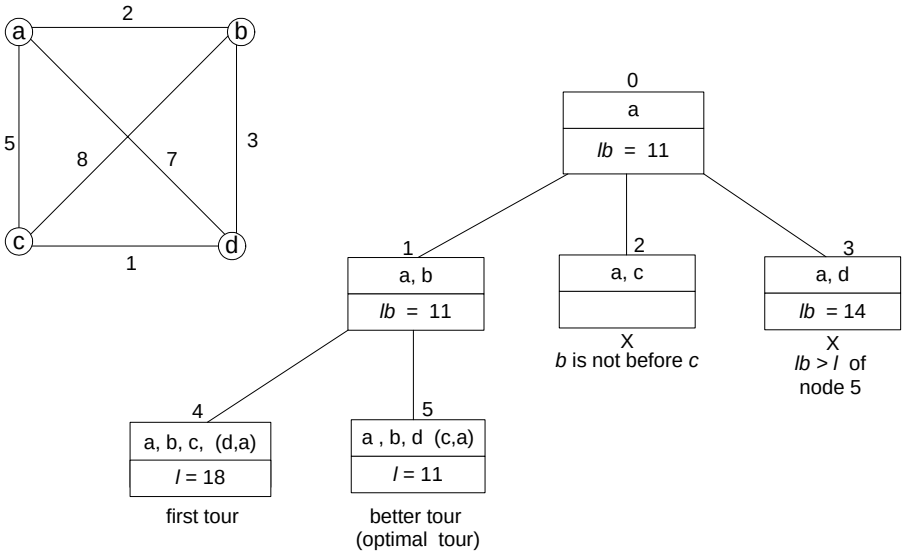
The found optimal solution is {item 2, item 3} of value \$119.

7. n/a

8. a. Any Hamiltonian circuit must have exactly two edges incident with each vertex of the graph: one for leaving the vertex and the other for entering it. The length of the edge leaving the ith vertex, $i = 1, 2, ...n$, is given by some nondiagonal element $D[i, j]$ in the ith row of the distance matrix. The length of the entering edge is given by some nondiagonal element $D[j', i]$ in the ith column of the distance matrix, which is not symmetric to the element in the ith row ($j' \neq j$) because the two edges must be distinct; this element is equal to some other element $D[i, j']$ in the ith row of the symmetric distance matrix. Hence, for any Hamiltonian circuit, the sum of the lengths of two edges incident with the ith vertex must be greater than or equal to s_i, the sum of the two smallest elements (not on the main diagonal of the matrix) in the ith row . Summing up these inequalities for all n vertices yields $2l \geq s$, where l is the length of any tour. Since l must be an integer if all the distances are integers, $l \geq \lceil s/2 \rceil$, which proves the assertion.

b. Redefine s_i as the sum of the smallest element in the ith row and the smallest element in the ith column (neither on the main diagonal of the matrix).

9. Without loss of generality, we'll consider a as the starting vertex and ignore the tours in which c is visited before b. Here is the state-space tree in question:



The found optimal tour is a, b, d, c, a of length 11.

10. n/a

Exercises 12.3

1. a. Apply the nearest-neighbor algorithm to the instance defined by the distance matrix below. Start the algorithm at the first city, assuming that the cities are numbered from 1 to 5.

$$\begin{bmatrix} 0 & 14 & 4 & 10 & \infty \\ 14 & 0 & 5 & 8 & 7 \\ 4 & 5 & 0 & 9 & 16 \\ 10 & 8 & 9 & 0 & 32 \\ \infty & 7 & 16 & 32 & 0 \end{bmatrix}$$

   b. Compute the accuracy ratio of this approximate solution.

2. a. Write a pseudocode for the nearest-neighbor algorithm. Assume that its input is given by an n-by-n distance matrix.

   b. What is the time efficiency of the nearest-neighbor algorithm?

3. Apply the twice-around-the-tree algorithm to the graph in Figure 12.11a with a walk around the minimum spanning tree that starts at the same vertex a but differs from the walk in Figure 12.11b. Is the length of the obtained tour the same as the length of the tour in Figure 12.11b?

4. ▷Prove that making a shortcut of the kind used by the twice-around-the-tree algorithm cannot increase the tour's length in an Euclidean graph.

5. What is the time efficiency class of the greedy algorithm for the knapsack problem?

6. ►Prove that the performance ratio R_A of the enhanced greedy algorithm for the knapsack problem is equal to 2.

7. Consider the greedy algorithm for the bin-packing problem, which is called the ***first-fit*** (FF) ***algorithm***: place each of the items in the order given into the first bin the item fits in; when there are no such bins, place the item in a new bin and add this bin to the end of the bin list.

   a. Apply FF to the instance

$$s_1 = 0.4, \quad s_2 = 0.7, \quad s_3 = 0.2, \quad s_4 = 0.1, \quad s_5 = 0.5$$

   and determine whether the solution obtained is optimal.

   b. Determine the worst-case time efficiency of FF.

   c.► Prove that FF is a 2-approximation algorithm.

8. The ***first-fit decreasing*** (FFD) approximation algorithm for the bin packing problem starts by sorting the items in nonincreasing order of their

sizes and then acts as the first-fit algorithm.

a. Apply *FFD* to the instance

$$s_1 = 0.4, \quad s_2 = 0.7, \quad s_3 = 0.2, \quad s_4 = 0.1, \quad s_5 = 0.5$$

and determine whether the solution obtained is optimal.

b. Does *FFD* always yield an optimal solution? Justify your answer.

c.▶ Prove that *FFD* is a 1.5-approximation algorithm.

d. Run an experiment to determine which of the two algorithms—*FF* or *FFD*—yields more accurate approximations on a random sample of the problem's instances.

9. a.▷ Design a simple 2-approximation algorithm for finding the ***minimum vertex cover*** (the vertex cover with the smallest number of vertices) in a given graph.

   b.▶ Consider the following approximation algorithm for finding the ***maximum independent set*** (the independent set with the largest number of vertices) in a given graph. Apply the 2-approximation algorithm of part a and output all the vertices that are not in the obtained vertex cover. Can we claim that this algorithm is a 2-approximation algorithm, too?

10. a. Design a polynomial-time greedy algorithm for the graph-coloring problem.

    b. Show that the performance ratio of your approximation algorithm is infinitely large.

Hints to Exercises 12.3

1. a. Start by marking the first column of the matrix and finding the smallest element in the first row and an unmarked column.

   b. You will have to find an optimal solution by exhaustive search as explained in Section 3.4.

2. a. The simplest approach is to mark matrix columns that correspond to visited cities. Alternatively, you can maintain a linked list of unvisited cities.

   b. Following the standard plan for analyzing algorithm efficiency should pose no difficulty (and yield the same result for either of the two options mentioned in the hint to part a).

3. Do the walk in the clockwise direction.

4. Extend the triangle inequality to the case of $k \geq 1$ intermediate vertices and prove its validity by mathematical induction.

5. First, determine the time efficiency of each of the three steps of the algorithm.

6. You will have to prove two facts:

   i. $f(s^*) \leq 2f(s_a)$ for any instance of the knapsack problem, where $f(s_a)$ is the value of the approximate solution obtained by the enhanced greedy algorithm and $f(s^*)$ is the optimal value of the exact solution for the same instance.

   ii. The smallest constant for which the assertion above is true is 2.

   In order to prove i, use the value of the optimal solution to the continuous version of the problem and its relationship to the value of the approximate solution. In order to prove ii, find a family of three-item instances that prove the point (two of them can be of weight $W/2$ and the third one can be of a weight slightly more than $W/2$).

7. a. Trace the algorithm on the instance given and then answer the question whether you can put the same items in fewer bins.

   b. What is the basic operation of this algorithm? What inputs make the algorithm run the longest?

   c. Prove first the inequality

$$B_{FF} < 2\sum_{i=1}^{n} s_i \quad \text{for any instance with } B_{FF} > 1,$$

where B_{FF} is the number of bins obtained by applying the first-fit (FF) algorithm to an instance with sizes $s_1, s_2, ..., s_n$. To prove it, take advantage of the fact that there can be no more than one bin that is half full or less.

8. a. Trace the algorithm on the instance given and then answer the question whether you can put the same items in fewer bins.

   b. You can answer the question either with a theoretical argument or by providing a counterexample.

   c. Take advantage of the two following properties:

   i. All the items placed by FFD in extra bins, i.e., bins after the first B^* ones, have size at most $1/3$.

   ii. The total number of items placed in extra bins is at most $B^* - 1$. (B^* is the optimal number of bins.)

   d. This task has two versions of dramatically different levels of difficulty. What are they?

9. a. One such algorithm is based on the idea similar to that of the source removal algorithm for the transitive closure except that it starts with an arbitrary edge of the graph.

   b. Recall our warning that polynomial-time equivalence of solving NP-hard problems exactly does not imply the same for their approximate solving.

10. a. Color the vertices without introducing new colors unnecessarily.

    b. Find a sequence of graphs G_n for which the ratio

$$\frac{\varkappa_a(G_n)}{\varkappa^*(G_n)}$$

    (where $\varkappa_a(G_n)$ and $\varkappa^*(G_n)$ are the number of colors obtained by the greedy algorithm and the optimal number of colors, respectively) can be made as large as we wish.

Solutions to Exercises 12.3

1. a. The nearest-neighbor algorithm yields the tour $1-3-2-5-4-1$ of length 58.

   b. To compute the accuracy ratio, we'll have to find the length of the optimal tour for the instance given by the distance matrix

$$\begin{bmatrix} 0 & 14 & 4 & 10 & \infty \\ 14 & 0 & 5 & 8 & 7 \\ 4 & 5 & 0 & 9 & 16 \\ 10 & 8 & 9 & 0 & 32 \\ \infty & 7 & 16 & 32 & 0 \end{bmatrix}.$$

   Generating all the finite-length tours that start and end at city 1 and visit city 2 before city 3 (see Section 3.4) yields the following:

   $1-2-3-5-4-1$ of length 77
   $1-2-4-5-3-1$ of length 74
   $1-2-5-3-4-1$ of length 56
   $1-2-5-4-3-1$ of length 66
   $1-4-2-5-3-1$ of length 45
   $1-4-5-2-3-1$ of length 58,

   with the tour $1-4-2-5-3-1$ of length 45 being optimal. Hence, the accuracy ratio of the approximate solution obtained by the nearest-neighbor algorithm is

$$r(s_a) = \frac{f(s_a)}{f(s^*)} = \frac{58}{45} \approx 1.29.$$

2. a. This is a pseudocode of a straightforward implementation.

   Algorithm *NearestNeighbor*($D[1..n, 1..n]$, s)
   //Implements the nearest-neighbor heuristic for the TSP
   //Input: A matrix $D[1..n, 1..n]$ of intercity distances and
   // an index s of the starting city
   //Output: A list $Tour$ of the vertices composing the tour obtained
   for $i \leftarrow 1$ **to** n **do** $Visited[i] \leftarrow$ **false**
   initialize a list $Tour$ with s
   $Visited[s] \leftarrow$ **true**
   $current \leftarrow s$
   for $i \leftarrow 2$ **to** n **do**
   find column j with smallest element in row *current* & unmarked col.
   $current \leftarrow j$
   $Visited[j] \leftarrow$ **true**
   add j to the end of list $Tour$

add s to the end of list $Tour$
return $Tour$

b. With either implementation (see the hint), the algorithm's time efficiency is in $\Theta(n^2)$.

3. The tour in question is *a, b, d, e, c, a* of length 38, which is not the same as the length of the tour based on the counter-clockwise walk around the minimum spanning tree.

4. The assertion in question follows immediately from the following generalization of the triangle inequality. If distances satisfy the triangle inequality, then they also satisfy its extension to an arbitrary positive number k of intermediate cities:

$$d[i, j] \leq d[i, m_1] + d[m_1, m_2] + ... + d[m_{k-1}, m_k] + d[m_k, j].$$

We will prove this by mathematical induction. The basis case of $k = 1$ is the triangle inequality itself:

$$d[i, j] \leq d[i, m_1] + d[m_1, j].$$

For the general case, we assume that for any two cities i and j and any set of $k \geq 1$ cities $m_1, m_2, ..., m_k$

$$d[i, j] \leq d[i, m_1] + d[m_1, m_2] + ... + d[m_{k-1}, m_k] + d[m_k, j]$$

to prove that for any two cities i and j and any set of $k + 1$ cities $m_1, m_2, ..., m_{k+1}$

$$d[i, j] \leq d[i, m_1] + d[m_1, m_2] + ... + d[m_k, m_{k+1}] + d[m_{k+1}, j].$$

Indeed, by the triangle inequality applied to m_k, j, and one intermediate city m_{k+1}, we obtain

$$d[m_k, j] \leq d[m_k, m_{k+1}] + d[m_{k+1}, j].$$

Replacing $d[m_k, j]$ in the inductive assumption by $d[m_k, m_{k+1}]+d[m_{k+1}, j]$ yields the inequality we wanted to prove.

5. Computing n value-to-weight ratios is in $\Theta(n)$. The time efficiency of sorting depends on the sorting algorithm used: it's in $O(n^2)$ for elementary sorting algorithms and in $O(n \log n)$ for advanced sorting algorithms such as mergesort. The third step is in $O(n)$. Hence, the running time of the entire algorithm will be dominated by the sorting step and will be in $\Theta(n) + O(n \log n) + O(n) = O(n \log n)$, provided an efficient sorting algorithm is used.

6. i. Let us prove that

$$f(s^*) \le 2f(s_a)$$

for any instance of the knapsack problem, where $f(s_a)$ is the value of the approximate solution obtained by the enhanced greedy algorithm and $f(s^*)$ is the optimal value of the exact solution for the same instance. In the trivial case when all the items fit into the knapsack, $f(s_a) = f(s^*)$ and the inequality obviously holds. Let I be a nontrivial instance. We can assume without loss of generality that the items are numbered in nonincreasing order of their efficiency (i.e., value to weight) ratios, that each of them fits into the knapsack, and that the item of the largest value has index m. Let s^* be the exact optimal solution for this discrete instance I and let k be the index of the first item that does not fit into the knapsack. Let c^* be the (exact) solution to the continuous counterpart of instance I. We have the following upper estimate for $f(s^*)$:

$$f(s^*) \le f(c^*) < \sum_{i=1}^{k-1} v_i + v_k \le f(s_a) + v_m \le f(s_a) + f(s_a) = 2f(s_a).$$

ii. Consider, for example, the family of instances defined as follows:

item	*weight*	*value*	$\frac{value}{weight}$
1	$(W+1)/2$	$(W+2)/2$	>1
2	$W/2$	$W/2$	1
3	$W/2$	$W/2$	1

with the knapsack's capacity $W \ge 2$, which will serve as the family's parameter.

The optimal solution s^* to any instance of this family is {item 2, item 3} of value W. The approximate solution s_a obtained by the enhanced greedy algorithm is {item 1} of value $(W+2)/2$. Hence,

$$\frac{f(s^*)}{f(s_a)} = \frac{W}{(W+2)/2} = \frac{2}{1+2/W},$$

which can be made as close to 2 as we wish by making W sufficiently large.

7. a. The first-fit algorithm places items 1, 3, and 4 into the first bin, item 2 into the second bin, and item 5 into the third bin. This solution is not optimal because it's inferior to the solution that places items 1 and 5 into one bin and items 2, 3, and 4 into the other. The latter solution is optimal because the two bins is the smallest number possible since $\sum_{i=1}^{5} s_i > 1$ and hence all the items cannot fit into a single bin.

b. The basis operation is to check whether the current item fits into a

particular bin. (It can be done in constant time if the amount of remaining space is maintained for each started bin.) The worst-case input will force the algorithm to check all $i-1$ started bins when placing the ith item for $i = 2, ..., n$. For example, this will happen for any input with items of the same size that is greater than 0.5. Hence, the worst-case efficiency of the first-fit is quadratic because

$$\sum_{i=1}^{n}(i-1) = \frac{(n-1)n}{2} \in \Theta(n^2).$$

(Note: If we don't assume that the size of each input item doesn't exceed the bin's capacity, the algorithm will have to make i comparisons on its ith iteration in the worst case. This will not change the worst-case efficiency class of the algorithm since $\sum_{i=1}^{n} i = n(n+1)/2 \in \Theta(n^2)$.)

c. Obviously, *FF* is a polynomial time algorithm. We'll prove first that for any instance of the problem that requires more than one bin (i.e., $\sum_{i=1}^{n} s_i > 1$),

$$B_{FF} < 2\sum_{i=1}^{n} s_i,$$

where B_{FF} is the number of bins in the approximate solution obtained by the first-fit (FF) algorithm. In any solution obtained by this algorithm, there are no more than one bin that is half full or less, i.e., $S_k \le 0.5$, where S_k is the sum of the sizes of the items that go into bin k, $k = 1, 2, ..., B_{FF}$. (Indeed, if there were more that one such bin, the algorithm would've placed all the items in the later-filled bin of the two into the earlier-filled one at the latest.) Let $\tilde{k}$ be the index of the bin in the solution with the smallest sum of the item sizes in it and let $\bar{k}$ be the index of any other bin in the solution. (Since $\sum_{i=1}^{n} s_i > 1$, such other bin must exist.) The sum S of the sizes of the items in these two bins must exceed 1. Therefore, we have the following:

$$\sum_{i=1}^{n} s_i = \sum_{k=1}^{B_{FF}} S_k = \sum_{k=1,\ k\ne\tilde{k},\ k\ne\bar{k}}^{B_{FF}} S_k + S > 0.5(B_{FF}-2) + 1 = 0.5B_{FF},$$

which proves the inequality $B_{FF} < 2\sum_{i=1}^{n} s_i$. Combining this with the obvious observation that $\sum_{i=1}^{n} s_i \le B^*$, we obtain

$$B_{FF} < 2\sum_{i=1}^{n} s_i \le 2B^*.$$

For the trivial case of $\sum_{i=1}^{n} s_i \le 1$, *FF* puts all the items in the first bin, and hence $B_{FF} = B^* < 2B^*$ as well.

Note: The best (and tight) bound for the first fit is

$$B_{FF} \leq \lceil 1.7B^* \rceil \text{ for all inputs.}$$

(See the survey by Coffman et al. in "Approximation Algorithms for *NP*-hard Problems," edited by D.S. Hochbaum, PWS Publishing, 1995.)

8. a. The first-fit decreasing algorithm (*FFD*) places items of sizes 0.7, 0.2, and 0.1 in the first bin and items of sizes 0.5 and 0.4 in the second one. Since $B^* \geq \lceil \sum_{i=1}^{5} s_i \rceil = 2$, at least two bins are necessary, making the solution obtained by *FFD* optimal.

   b. The answer is no: if it did, *FFD* would be a polynomial time algorithm that solves this *NP*-hard problem. Here is one counterexample:

$$s_1 = 0.5, \quad s_2 = 0.4, \quad s_3 = 0.3, \quad s_4 = 0.3, \quad s_5 = 0.25, \quad s_6 = 0.25.$$

   (*FFD* yields a solution with 3 bins while the optimal number of bins is 2.)

   c. Obviously, *FFD* is a polynomial time algorithm. If *FFD* yields B_{FFD} bins while the optimal number of bins is B^*, we know from the properties quoted in the hint that the number of items in the extra bins is at most $B^* - 1$, with each of the items be of size at most 1/3. Therefore the total number of extra bins is at most $\lceil (B^* - 1)/3 \rceil$, and we have the following upper bound on the approximation's accuracy ratio:

$$B_{FFD} \leq B^* + \lceil (B^* - 1)/3 \rceil \leq B^* + \frac{B^* + 1}{3}.$$

   (You can check the validity of the last replacement of $\lceil (B^* - 1)/3 \rceil$ by $(B^* + 1)/3$ by considering separately three cases: $B^* = 3i$, $B^* = 3i + 1$, and $B^* = 3i + 2$.). Finally,

$$B^* + \frac{B^* + 1}{3} = (\frac{4}{3} + \frac{1}{3B^*})B^* \leq (\frac{4}{3} + \frac{1}{3 \cdot 2})B^* = 1.5B^*.$$

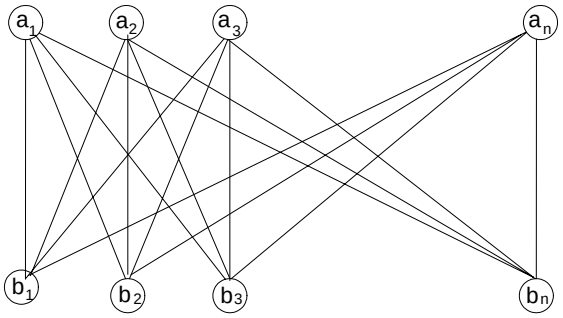
   (We used the observation that when $B^* \geq 2$, $\frac{1}{3B^*}$ is the largest when $B^* = 2$. When $B^* = 1$, *FFD* yields an exact solution, and the inequality $B_{FFD} \leq 1.5B^*$ checks out directly.)

   d. Note the two versions of this task. The easy one would simply compare which of the two greedy algorithms yields a more accurate solution more often. It is easy because, in this form, one doesn't need to know the optimal number of bins in a generated instance. The much more difficult version is to compare the average accuracy of the two approximation algorithms, which would require information about the optimal number of bins in each of the generated instances.

9. a. Initialize the vertex cover to the empty set. Repeat the following until no edges are left: select an arbitrary edge, add both its endpoints to the vertex cover, and remove from the graph all the edges incident with either of these two endpoint vertices.

   Let $L = e_1, e_2, ..., e_k$ be the list of edges chosen by the algorithm. The number of vertices in the vertex cover the algorithm returns, $|VC_a|$, is $2k$. Since no two edges in L have a common vertex, a minimum vertex cover of the graph must include at least one endpoint of each edge in L; therefore the number of vertices in it, $|VC^*|$, is at least k. Hence $|VC_a| \leq 2|VC^*|$.

   b. No. Consider, for example, the complete bipartite graph $K_{n,n}$ with vertices a_1, b_1, ..., a_n, b_n:



   Selecting edges (a_i, b_i), $i = 1, 2, ..., n$, in the 2-approximation vertex-cover algorithm of part a, yields the vertex cover with $2n$ vertices and hence 0 independent vertices. The maximum independent set has, in fact, n vertices (all a vertices or all b vertices).

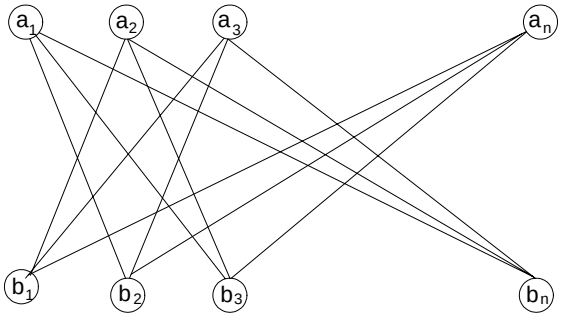
10. a. The simplest greedy heuristic, called *sequential coloring*, is to color a vertex in the first available color, i.e., the first color not used for coloring any vertex adjacent to it. (Vertices are colored in the order given by the graph's data structure.)

```
Algorithm SC(G)
//Implements sequential coloring of a given graph
//Input: A graph G = ⟨V, E⟩
//Output: An array Color of numeric colors assigned to the vertices
for i ← 1 to |V| do
    Color[i] = 0  //0 signifies no color
for i ← 1 to |V| do
    c ← 1
    while Color[i] = 0 do
        if no vertex adjacent to v_i has color c
          Color[i] ← c
```

else $c \leftarrow c + 1$
return *Color*

The algorithm's time efficiency is clearly in $O(|V|^3)$ because for each of the $|V|$ vertices, the algorithm checks no more than $|V|$ colors for up to $O(|V|)$ vertices adjacent to it.

b. Consider the following sequence of graphs G_n with $2n$ vertices specified in the order a_1, b_1, ..., a_n, b_n:



The smallest number of colors is 2 (color all the a_i's with color 1 and all the b_i's with color 2). The sequential coloring yields n colors: one for each pair of vertices a_i and b_i. Hence, for this sequence of graphs,

$$\frac{\varkappa_a(G_n)}{\varkappa^*(G_n)} = \frac{n}{2},$$

which is not bounded above.

Exercises 12.4

1. a. Find on the Internet or in your library a procedure for finding a real root of the general cubic equation $ax^3 + bx^2 + cx + d = 0$ with real coefficients.

   b. What general algorithm design technique is it based on?

2. Indicate how many roots each of the following equations has.

   a. $xe^x - 1 = 0$ b. $x - \ln x = 0$ c. $x \sin x - 1 = 0$

3. a. Prove that if $p(x)$ is a polynomial of an odd degree, then it must have at least one real root.

   b. Prove that if x_0 is a root of an n-degree polynomial $p(x)$, the polynomial can be factored into

   $$p(x) = (x - x_0)q(x),$$

   where $q(x)$ is a polynomial of degree $n-1$. Explain what significance this theorem has for finding roots of a polynomial.

   c. Prove that if x_0 is a root of an n-degree polynomial $p(x)$, then

   $$p'(x_0) = q(x_0),$$

   where $q(x)$ is the quotient of the division of $p(x)$ by $x - x_0$.

4. Prove inequality (12.7).

5. Apply the bisection method to find the root of the equation

   $$x^3 + x - 1 = 0$$

   with an absolute error smaller than 10^{-2}.

6. Derive formula (12.10) underlying the method of false position.

7. Apply the method of false position to find the root of the equation

   $$x^3 + x - 1 = 0$$

   with an absolute error smaller than 10^{-2}.

8. Derive formula (12.11) underlying Newton's method.

9. Apply Newton's method to find the root of the equation

   $$x^3 + x - 1 = 0$$

   with an absolute error smaller than 10^{-2}

10. Give an example that shows that the approximation sequence of Newton's method may diverge.

11. *Gobbling goat* A grassy field is in the shape of a circle of radius 100ft. A goat is attached by a rope to a hook at a fixed point of the field's border. How long should the rope be to let the goat reach only half of the grass in the field?

Hints to Exercises 12.4

1. It might help your search to know that the solution was first published by Italian Renaissance mathematician Girolamo Cardano.

2. You can answer these questions without using calculus or a sophisticated calculator by representing equations in the form $f_1(x) = f_2(x)$ and graphing functions $f_1(x)$ and $f_2(x)$.

3. a. Use the property underlying the bisection method.

   b. Use the definition of division of polynomial $p(x)$ by $x - x_0$, i.e., the equality
   $$p(x) = q(x)(x - x_0) + r$$
   where x_0 is a root of $p(x)$, $q(x)$ and r are the quotient and remainder of this division, respectively.

   c. Differentiate both sides of the equality given in part (b) and substitute x_0 in the result.

4. Use the fact that $|x_n - x^*|$ is the distance between x_n, the middle of interval $[a_n, b_n]$, and root x^*.

5. Sketch the graph to determine a general location of the root and choose an initial interval bracketing it. Use an appropriate inequality given in Section 12.4 to determine the smallest number of iterations required. Perform the iterations of the algorithm as it is done for the example in the section.

6. Write an equation of the line through the points $(a_n, f(a_n))$ and $(b_n, f(b_n))$ and find its x-intercept.

7. See the example given in the section. As a stopping criterion, you may use either the length of segment (a_n, b_n) or inequality (12.12).

8. Write an equation of the tangent line to the graph of the function at $(x_n, f(x_n))$ and find its x-intercept.

9. See the example given in the section. Of course, you may start with a different x_0 than the one used in that example.

10. Consider, for example, $f(x) = \sqrt[3]{x}$.

11. Derive an equation for the area in question and then solve it by using one of the methods discussed in the section.

Solutions to Exercises 12.4

1. a. Here is a solution as described at http://www.sosmath.com/algebra/factor/fac11/fac11.html:

   First, substitute $x = y - b/3a$ to reduce the general cubic equation

   $$ax^3 + bx^2 + cx + d = 0$$

   to the "depressed" cubic equation

   $$y^3 + Ay = B.$$

   Then solve the system

   $$\begin{aligned} 3st &= A \\ s^3 - t^3 &= B \end{aligned}$$

   by substituting $s = A/3t$ into the second equation to get the "tri-quadratic" equation for t

   $$t^6 + Bt^3 - \frac{A^3}{27} = 0.$$

   (The last equation can be solved as a quadratic equation after substitution $u = t^3$.) Finally,

   $$y = s - t$$

   yields the y's value, from which we get the root as

   $$x = y - b/3a.$$

   b. Transform-and-conquer.

2. a. Equation $xe^x - 1 = 0$ is equivalent to $e^x = 1/x$. The graphs of $f_1(x) = e^x$ and $f_2(x) = 1/x$ clearly have a single common point between 0 and 1.

   b. Equation $x - \ln x = 0$ is equivalent to $x = \ln x$, and the graphs of $f_1(x) = x$ and $f_2(x) = \ln x$ clearly don't intersect.

   c. Equation $x \sin x - 1 = 0$ is equivalent to $\sin x = 1/x$, and the graphs of $f_1(x) = \sin x$ and $f_2(x) = 1/x$ clearly intersect at infinitely many points.

3. a. For a polynomial $p(x) = a_n x^n + a_{n-1}x^{n-1} + ... + a_0$ of an odd degree, where $a_n > 0$,

   $$\lim_{x \to -\infty} p(x) = -\infty \quad \text{and} \quad \lim_{x \to +\infty} p(x) = +\infty.$$

Therefore there exist real numbers a and b, $a < b$, such that $p(a) < 0$ and $p(b) > 0$. In addition, any polynomial is a continuous function everywhere. Hence, by the theorem mentioned in conjunction with the bisection method, $p(x)$ must have a root between a and b.
The case of $a_n < 0$ is reduced to the one with the positive coefficient by considering $-p(x)$.

b. By definition of division of $p(x)$ by $x - x_0$, where x_0 is a root of $p(x)$,

$$p(x) = q(x)(x - x_0) + r,$$

where $q(x)$ and r are the quotient and remainder of this division, respectively. Substituting x_0 for x into the above equation and taking into account that $p(x_0) = 0$ proves that remainder r is equal to 0:

$$p(x_0) = r = 0.$$

Hence,

$$p(x) = q(x)(x - x_0).$$

This implies that if one root of an n-degree polynomial $p(x)$ is known, the other roots can be found by solving

$$q(x) = 0,$$

where $q(x)$—the quotient of the division of $p(x)$ by $x - x_0$ (x_0 is a known root)—is a polynomial of degree $n - 1$.

c. Differentiating both hand sides of equality

$$p(x) = q(x)(x - x_0)$$

yields

$$p'(x) = q'(x)(x - x_0) + q(x).$$

Substituting x_0 for x results in

$$p'(x_0) = q'(x_0)(x_0 - x_0) + q(x_0) = q(x_0).$$

4. Since x_n is the middle point of interval $[a_n, b_n]$, its distance to any point within that interval, including root x^*, cannot exceed the interval's half length, which is $(b_n - a_n)/2$. That is

$$|x_n - x^*| \leq \frac{b_n - a_n}{2} \text{ for } n = 1, 2, ...$$

But the length of the intervals $[a_n, b_n]$ is halved on each iteration. Hence,

$$b_n - a_n = \frac{b_{n-1} - a_{n-1}}{2} = \frac{b_{n-2} - a_{n-2}}{2^2} = ... = \frac{b_1 - a_1}{2^{n-1}}.$$

(Use mathematical induction, if you prefer a more formal proof.) Thus,

$$\frac{b_n - a_n}{2} = \frac{b_1 - a_1}{2^n}.$$

Substituting this in the inequality above yields

$$|x_n - x^*| \leq \frac{b_1 - a_1}{2^n} \quad \text{for } n = 1, 2, ...$$

5. The graph of $f(x) = x^3 + x - 1$ makes it obvious that this polynomial has a single real root that lies in the interval $0 < x < 1$. (It also follows from the fact that this polynomial has an odd degree and its derivative is positive for every x.) Solving inequality (12.8) with $a = 0$, $b = 1$, and $\epsilon = 10^{-2}$, i.e.,

$$n > \log_2 \frac{1 - 0}{10^{-2}},$$

yields $n \geq 7$. The following table contains the results of the first seven iterations of the bisection method:

n	a_n	b_n	x_n	$f(x_n)$
1	0.0-	1.0+	0.5	-0.375
2	0.5-	1.0+	0.75	0.171875
3	0.5-	0.75+	0.625	-0.130859
4	0.625-	0.75+	0.6875	0.012451
5	0.625-	0.6875+	0.65625	-0.061127
6	0.65625-	0.6875+	0.671875	-0.024830
7	0.671875-	0.6875+	0.6796875	-0.006314

Thus, the obtained approximation is $x_7 = 0.6796875$.

6. Substituting $(a_n, f(a_n))$ and $(b_n, f(b_n))$, the two given points, into the standard straight-line equation

$$y - y_1 = \frac{y_2 - y_1}{x_2 - x_1}(x - x_1),$$

we obtain

$$y - f(a_n) = \frac{f(b_n) - f(a_n)}{b_n - a_n}(x - a_n).$$

Setting y to 0 to find the line's x-intercept, we obtain the following equation for x_n

$$-f(a_n) = \frac{f(b_n) - f(a_n)}{b_n - a_n}(x_n - a_n).$$

Solving for x_n yields

$$x_n = a_n - \frac{f(a_n)(b_n - a_n)}{f(b_n) - f(a_n)},$$

or, after standard algebraic simplifications,

$$x_n = \frac{a_n f(b_n) - f(a_n) b_n}{f(b_n) - f(a_n)},$$

which is the formula for the approximation sequence of the method of false position.

7. For $f(x) = x^3 + x - 1$,

$$f'(x) = 3x^2 + 1 \geq 1 \text{ for every } x.$$

Hence, according to inequality (12.12), we can stop the iterations as soon as

$$|x_n - x^*| \leq |f(x_n)| < 10^{-2}.$$

The following table contains the results of the first four iterations of the method of false position:

n	a_n	b_n	x_n	$f(x_n)$
1	0.0-	1.0+	0.5	-0.375
2	0.5-	1.0+	0.636364	-0.105935
3	0.636364-	1.0+	0.671196	-0.026428
4	0.671196-	1.0+	0.679662	-0.006375

Thus, the obtained approximation is $x_4 = 0.679662$.

8. Using the standard equation for the tangent line to the graph of the function $f(x)$ at $(x_n, f(x_n))$, we obtain

$$y - f(x_n) = f'(x_n)(x - x_n).$$

Setting y to 0 to find its x-intercept, which is x_{n+1} of Newton's method, yields

$$-f(x_n) = f'(x_n)(x_{n+1} - x_n).$$

Solving for x_{n+1} yields, if $f'(x_n) \neq 0$,

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)},$$

which is the formula for the approximation sequence of Newton's method.

9. For $f(x) = x^3 + x - 1$,

$$f'(x) = 3x^2 + 1 \geq 1 \text{ for every } x.$$

Hence, according to inequality (12.12), we can stop the iterations as soon as

$$|x_n - x^*| \leq |f(x_n)| < 10^{-2}.$$

The following table contains the results of the first two iterations of Newton's method, with $x_0 = 1$:

n	x_n	x_{n+1}	$f(x_n)$
0	1.0	0.75	0.171875
1	0.75	0.686047	0.008941

Thus, the obtained approximation is $x_2 = 0.686047$.

10. Equation $\sqrt[3]{x} = 0$ has $x = 0$ as its only root. Using the geometric interpretation of Newton's method, it is easy to see that the approximation sequence (the x-intercepts of the tangent lines) diverges for any initial approximation $x_0 \neq 0$. Here is a formal proof of this fact. The approximation sequence of Newton's method is given by the formula

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} = x_n - \frac{x_n^{1/3}}{\frac{1}{3}x_n^{-2/3}} = x_n - 3x_n = -2x_n.$$

This equality means that if $x_0 \neq 0$, each next approximation x_{n+1} is twice as far from 0, the equation's root, as its predecessor x_n. Hence, sequence $\{x_n\}$ diverges for any initial approximation $x_0 \neq 0$.

11. You can find a solution to this classic puzzle at http://plus.maths.org/issue9/puzzle/solution.html.