Nilsson, N. J. *Principles of Artificial Intelligence.* Palo Alto, CA: Tioga, 1980. A detailed account of the A*-search, which was originally developed by Hart, Nilsson, and Raphael.

Pearl, J. *Heuristics: Intelligent Search Strategies for Computer Problem Solving.* Reading, MA: Addison-Wesley, 1984. A text devoted to heuristic searching.

Two books on artificial intelligence that contain extensive discussions of search strategies and game playing:

Rich, E., and K. Knight. Artificial Intelligence. 2nd ed. New York: McGraw-Hill, 1991.

Russell, S. J., and P. Norvig. Artificial Intelligence: A Modern Approach. Englewood Cliffs, NJ: Prentice Hall, 1995.

Patashnik, O. "Qubic: 4 3 4 3 4 Tic-Tac-Toe," Mathematics Magazine 53 (1980): 202–223. Survey discussion of n-dimensional tic-tac-toe, as well as the proof that the 4 3 4 3 4 is a first-player win.

Two papers containing detailed analyses of alpha-beta and deep alpha-beta pruning:

Baudet, G. "An Analysis of the Full Alpha-Beta Pruning Algorithm," Proceedings of the 10th Annual ACM Symposium on Theories of Computing, San Diego, CA: Association for Computing Machinery, 1978, pp. 296–313.

Knuth, D. "An Analysis of Alpha-Beta Cutoffs," Artificial Intelligence 6 (1975): 293–323.

Berlekamp, E. R., J. H. Conway, and R. K. Guy. Winning Ways, for Your Mathematical Plays. Vol. I, II. New York: Academic Press, 1982. Covers strategies for a host of games.

EXERCISES

Section 23.2 8-Puzzle Game

23.1 Draw the first three levels of the state-space tree generated by a breadth-first search for the 8-puzzle game with the following initial and goal states:

goal

1	2	3
4	5	6
7	8	

initial state

4	2	7
1		6
3	5	8

23.2 The $(n^2 - 1)$-puzzle is a generalization of the 8-puzzle to the $n \times n$ board. The goal position is where the tiles are in row-major order (with the empty space in the lower-right corner). For $k \in \{1, \ldots, n^2\}$, let $L(k)$ denote the number of tiles t, $t < k$, such that the position of t comes after k in the row-major order in the initial arrangement (the empty space is considered as tile n^2). Show that a necessary and sufficient condition that the goal can be reached is that

$$\sum_{k=1}^{n^2} L(k) \equiv i + j (\text{mod } 2), \qquad \textbf{(23.3.4)}$$

where (i, j) is the position of the empty space in the initial arrangement.

Section 23.3 A*-Search

23.3 Show that if h is a monotone heuristic, then $h(v)$ is a lower bound of the cost of the shortest path from v to a (nearest) goal.

23.4 Prove Proposition 23.3.3.

23.5 Design an algorithm for an A*-search using a heuristic that is not necessarily monotone.

23.6 For the 8-puzzle game, consider the following heuristic:

$h(v)$ = the number of tiles not in correct cell in the state v.

Show that $h(v)$ satisfies the monotone restriction.

23.7 For the 8-puzzle game, consider the following heuristic:

$h(v)$ = the sum of the Manhattan distances (vertical steps plus horizontal steps) from the tiles to their proper positions in the goal state.

Show that h satisfies the monotone restriction.

23.8 For the 8-puzzle game, let h be the monotone heuristic defined in Exercise 23.6. For the following initial and goal states, draw the states generated by making the first three moves in the game using an A*-search with the priority function $f(v) = g(v) + h(v)$ [$g(v)$ is the number of moves made from the initial state to v]. When enqueuing states, assume that the (possible) moves of the empty tile are ordered as follows: move left, move right, move up, move down. Label each state v with its f-value.

goal

1	2	3
4	5	6
7	8	

initial state

4	2	7
1		6
3	5	8

23.9 Repeat Exercise 23.8 for the heuristic h defined in Exercise 23.7.

23.10 Write a program for the n-puzzle game using the Manhattan distance heuristic. Test your program for $n = 8$ and $n = 15$.

23.11 Can you find better heuristics (not necessarily lower bounds) for the n-puzzle game than the Manhattan distance heuristic? Test your heuristic empirically for $n = 8$ and $n = 15$.

23.12 Prove Theorem 23.3.1

Section 23.4 Least-Cost Branch-and-Bound

23.13 Show that the heuristic $h(v)$ given in Section 23.4 for the coin-changing problem is a lower bound for the minimum number of additional coins required to make correct change from the given problem state v.

23.14 Write a program implementing a least-cost branch-and-bound solution to the coin-changing problem.

23.15 Design a heuristic and a least-cost branch-and-bound algorithm for the variation of the coin-changing problem in which we have a limited number of coins of each denomination. Assume the number of coins of each denomination is input along with the denominations.

23.16 Draw the portion of the variable-tuple state-space tree generated by least-cost branch-and-bound for the instance of the 0/1 knapsack problem given in Figure 10.13 in Chapter 10, using the heuristic given in Section 23.4. Label each node with the value of $c(v)$ and the current value of UB.

23.17 Repeat Exercise 23.16 for the fixed-tuple state-space tree.

23.18 Write a program implementing a least-cost branch-and-bound solution to the 0/1 knapsack problem.

23.19 Given the complete digraph $\hat{K}_n$ with vertices $0, 1, \ldots, n-1$ and a nonnegative cost matrix $C = (c_{ij})$ for its edges (we set $c_{ij} = \infty$ if $i = j$ or if the edge ij does not exist), a traveling salesman tour starting at vertex 0 corresponds to a sequence of vertices $0, i_1, i_2, \ldots, i_{n-1}, 0$, where $i_1, i_2, \ldots, i_{n-1}$ is a permutation of $1, \ldots, n-1$. Consider a state-space tree T for the traveling salesman problem (finding a minimum-cost tour) where a node at level k in T corresponds to a simple path containing $k+1$ vertices, starting with vertex 0. Thus, T has depth n, and leaf nodes correspond to a sequence of choices $i_1, i_2, \ldots, i_{n-1}$, determining the tour $0, i_1, i_2, \ldots, i_{n-1}, 0$.

We now describe a cost function $c(v)$ for a least-cost branch-and-bound algorithm for the traveling salesman problem. The definition of $c(v)$ is based on the notion of a reduced-cost matrix. A row (or column) of a nonnegative cost matrix is said to be *reduced* if it contains at least one zero. A nonnegative cost matrix is *reduced* if each row and column of the matrix is reduced (except for rows and columns whose elements are all equal to ∞). Given the cost matrix C, an associated reduced-cost matrix C_r is constructed as follows. First, reduce each row by subtracting the minimum entry in the row from each element in the row. In the resulting matrix, repeat this process for each column. We define $c(r)$ to be the total amount subtracted. The following example illustrates C_r for a sample C.

$$C = \begin{pmatrix} \infty & 23 & 9 & 32 & 12 \\ 21 & \infty & 2 & 16 & 4 \\ 4 & 8 & \infty & 20 & 6 \\ 15 & 10 & 4 & \infty & 2 \\ 9 & 5 & 8 & 10 & \infty \end{pmatrix} \qquad C_r = \begin{pmatrix} \infty & 14 & 0 & 18 & 3 \\ 19 & \infty & 0 & 9 & 2 \\ 0 & 4 & \infty & 11 & 2 \\ 13 & 8 & 2 & \infty & 0 \\ 4 & 0 & 3 & 0 & \infty \end{pmatrix} \qquad c(r) = 27$$

More generally, we define (inductively on the levels of T) a reduced-cost matrix for each nonleaf node by suitably reducing the cost matrix C_u associated with the parent node u of v. Suppose u corresponds to a path ending at vertex i, and v corresponds to adding the edge ij to this path. We then change all the entries in row i and column j of C_u to ∞, as well as the entry in the j^{th} row and first column. We then perform the same subtracting operation on the resulting matrix as we did when computing C_r. Let s_v denote the total amount subtracted, and define $c(v) = c(u) + C_u(i, j) + s_v$.

For leaf nodes v, $c(v)$ is defined as the cost of the tour determined by v.

a. Show that $c(r)$ is a lower bound for the minimum cost of a tour.

b. More generally, show that $c(v) \leq \phi^*(v) =$ the minimum cost over all tours determined by the leaf nodes of the subtree of T rooted at v.

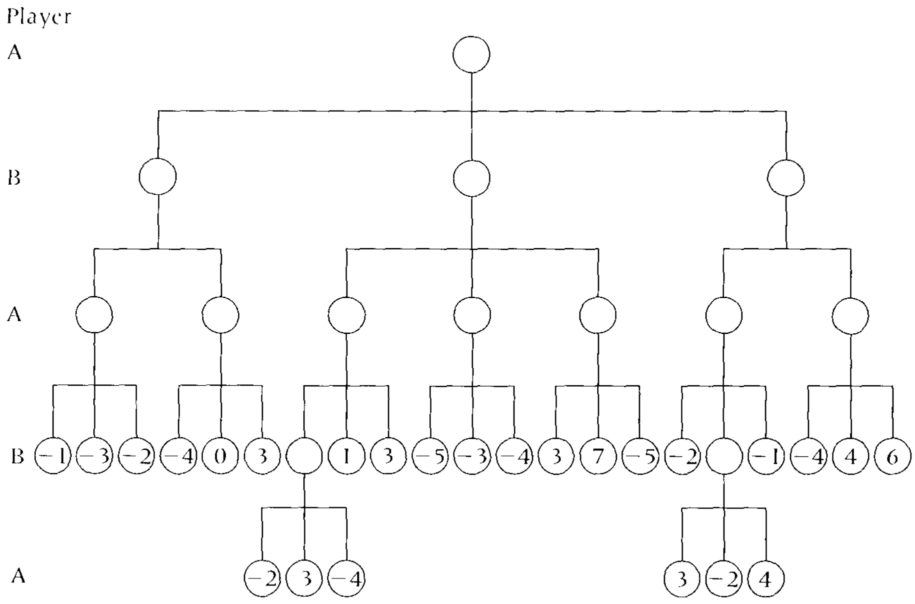
c. Part (b) shows that $c(v)$ is suitable for *LeastCostBranchAndBound*. Design and give pseudocode for *LeastCost BranchAndBound* implementing $c(v)$.

23.20 Draw the portion of the state-space tree T generated by the least-cost branch-and-bound discussed in the Exercise 23.19 for the cost matrix illustrated in that exercise. Label each node v with its cost value $c(v)$. Also, write out the reduced matrix associated with each node generated.

23.21 Discuss other state-space trees and associated cost functions $c(v)$ for the traveling salesman problem.

Section 23.5 Game Trees

23.22 Consider the two-person zero-sum game shown in the figure below. The values in the leaf nodes are values to player A. Use the minimax strategy (postorder traversal) to determine the value of the game to player A. Show clearly where alpha-cutoff and beta-cutoff occur, as well as (final) actual values, alpha values, and beta values of all nodes reached in the traversal.



23.23 Rewrite the recursive function *NodeValue* as a recursive procedure that has the same input parameters, *Y*, *NumLevels*, *ParentValue*, but now returns in output parameters the value *V* of *Y* and the child C_i, whose value is $-V$.

23.24 Find a sequence of admissible moves for the two-level heuristic search illustrated in Figure 23.16 that leads to a loss for player A in 3×3 tic-tac-toe.

23.25 Show that by assigning the values 9, 0, and -9 to terminal nodes that are wins, ties, or losses, respectively, for player A, the two-level search illustrated in Figure 23.16 never leads to a loss for player A.

23.26 Because there are no tie positions in the $3 \times 3 \times 3$ tic-tac-toe game, the first player has a winning strategy. Find a winning strategy for the first player.

23.27 Design a recursive function *DABNodeValue*(*X*, *NumLevels*, *ParentValue*, *NodeValueLowBnd*) for deep alpha-beta pruning. The initial invocation of *DABNodeValue* should have *ParentValue* $= \infty$ and *NodeValueLowBnd* $= -\infty$.

23.28 Redo Exercise 23.22 for deep alpha-beta pruning. Indicate any pruned nodes that were not pruned by alpha-beta pruning.