Artificial Intelligence Exercises

Marcello Restelli and Nicola Gatti

November 20, 2020

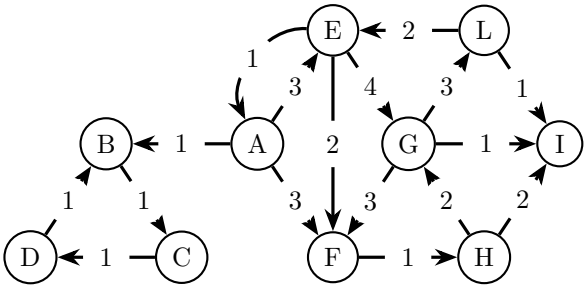
Abstract

Chapter 1

Uninformed Search Strategies

Exercise 1.1

Given the search problem defined by the following graph, where A is the initial state and L is the goal state and the cost of each action is indicated on the corresponding edge

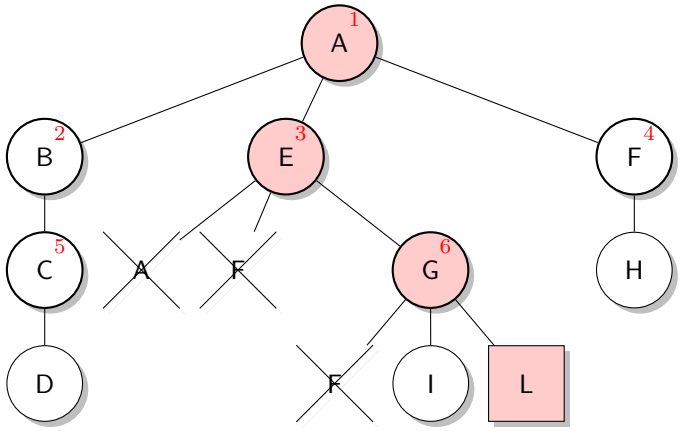


Solve the search problem using the following search strategies:

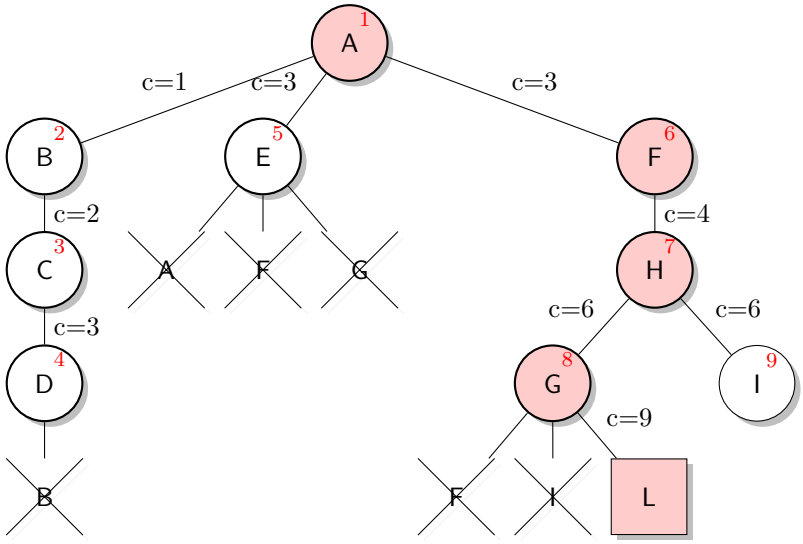
1. breadth-first with elimination of repeated states
2. uniform-cost with elimination of repeated states
3. depth-first with elimination of repeated states

Answer of exercise 1.1

Breadth-first search with elimination of repeated states.

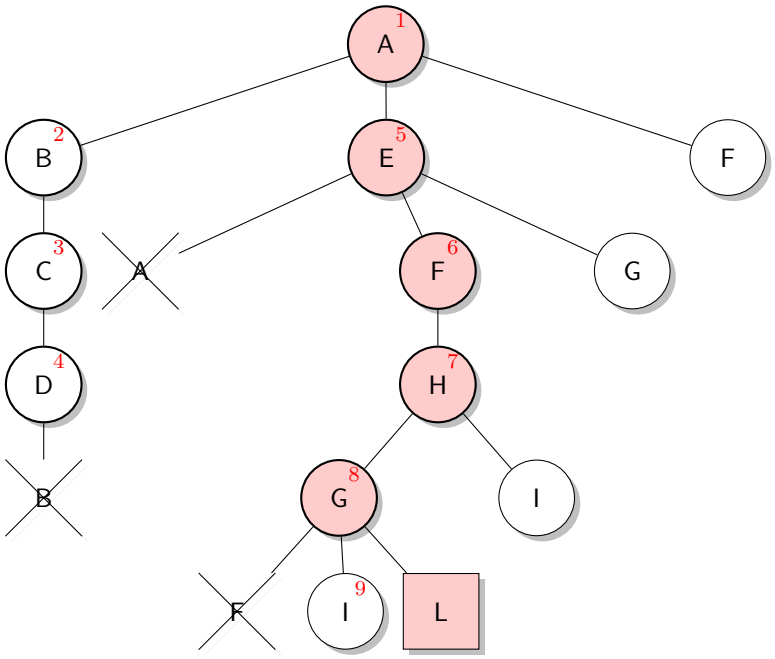


Uniform-cost search with elimination of repeated states.



In case of ties, nodes have been considered in lexicographic order.

Depth-first search with elimination of repeated states.



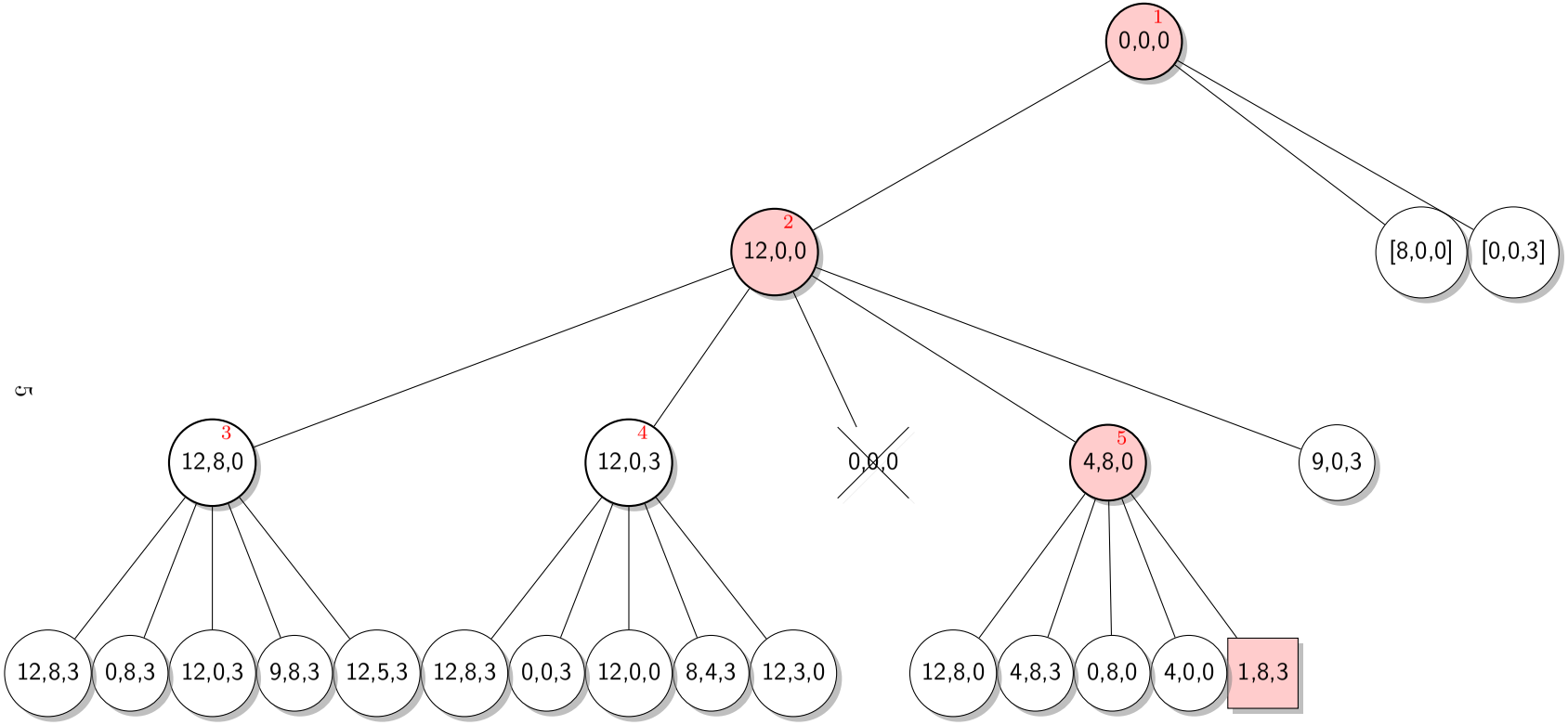
Exercise 1.2

You have three containers that may hold 12 liters, 8 liters and 3 liters of water, respectively, as well as access to a water faucet. You can fill a container from the faucet, pour it into another container, or empty it onto the ground. The goal is to measure exactly one liter of water.

1. Give a precise specification of the task as a search problem.
2. Draw the search tree produced by depth-limited search with maximum depth equal to three and elimination of repeated states.

Answer of exercise 1.2

- **State**: $[x, y, z]$ – contents of 12, 8 and 3 liters
- **Initial state**: $[0, 0, 0]$
- **Fill actions**
  - Fill12: $[x, y, z] \Rightarrow [12, y, z]$
  - Fill8: $[x, y, z] \Rightarrow [x, 8, z]$
  - Fill3: $[x, y, z] \Rightarrow [x, y, 3]$
- **Empty actions**
  - Empty12: $[x, y, z] \Rightarrow [0, y, z]$
  - Empty8: $[x, y, z] \Rightarrow [x, 0, z]$
  - Empty3: $[x, y, z] \Rightarrow [x, y, 0]$
- **Pour actions**
  - Pour12-8: $[x, y, z] \Rightarrow [x - \min(x, 8 - y), y + \min(x, 8 - y), z]$
  - Pour12-3: $[x, y, z] \Rightarrow [x - \min(x, 3 - z), y, z + - \min(x, 3 - z)]$
  - Pour8-12: $[x, y, z] \Rightarrow [x + \min(y, 12 - x), y - \min(y, 12 - x), z]$
  - Pour8-3: $[x, y, z] \Rightarrow [x, y - \min(y, 3 - z), z + \min(y, 3 - z)]$
  - Pour3-12: $[x, y, z] \Rightarrow [x + \min(z, 12 - x), y, z - \min(z, 12 - x)]$
  - Pour3-8: $[x, y, z] \Rightarrow [x, y + \min(z, 8 - y), z - \min(z, 8 - y)]$
- **Goal test**: $[1, y, z] \vee [x, 1, z] \vee [x, y, 1]$
- **Path cost**: number of actions in path from intial state to goal state

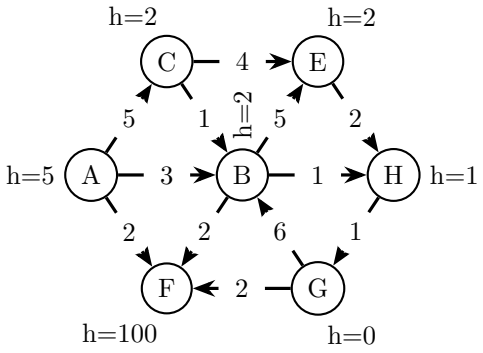

1
0,0,0
2
12,0,0
[8,0,0]
[0,0,3]
3
12,8,0
4
12,0,3
0,0,0
5
4,8,0
9,0,3
12,8,3
0,8,3
12,0,3
9,8,3
12,5,3
12,8,3
0,0,3
12,0,0
8,4,3
12,3,0
12,8,0
4,8,3
0,8,0
4,0,0
1,8,3

Chapter 2

Informed Search Strategies

Exercise 2.1

Given the search problem defined by the following graph, where A is the initial state and G is the goal state and the cost of each action is indicated on the corresponding edge

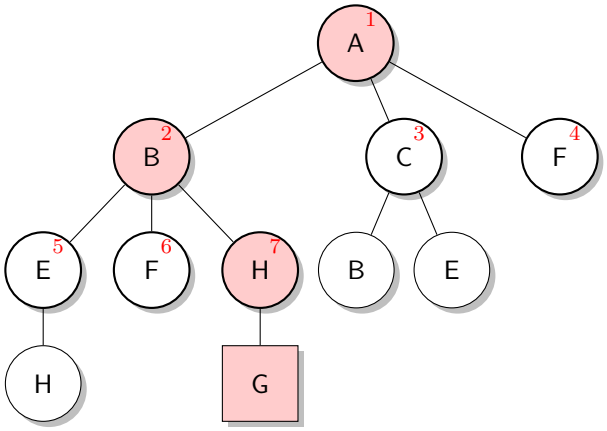


Solve the search problem using the following search strategies:

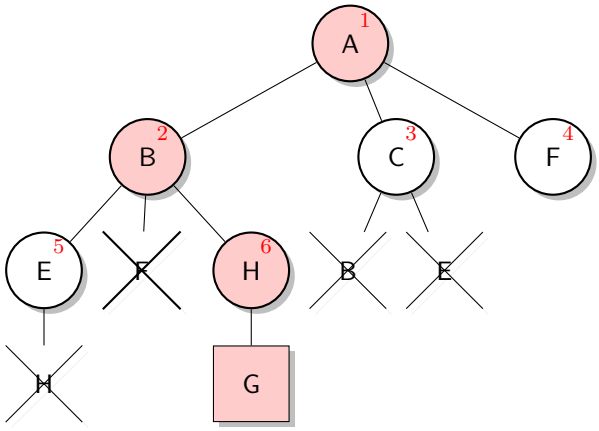
1. breadth-first without elimination of repeated states
2. breadth-first with elimination of repeated states
3. depth-first without elimination of repeated states
4. depth-first with elimination of repeated states
5. uniform-cost without elimination of repeated states
6. uniform-cost with elimination of repeated states
7. greedy best-first without elimination of repeated states
8. greedy best-first with elimination of repeated states
9. A^* without elimination of repeated states
10. A^* with elimination of repeated states
11. Is the heuristic h admissible? Is it consistent?

Answer of exercise 2.1

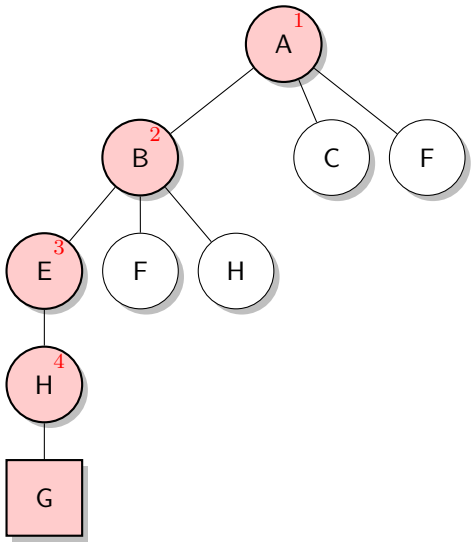
Breadth-first without elimination of repeated states.



Breadth-first with elimination of repeated states.

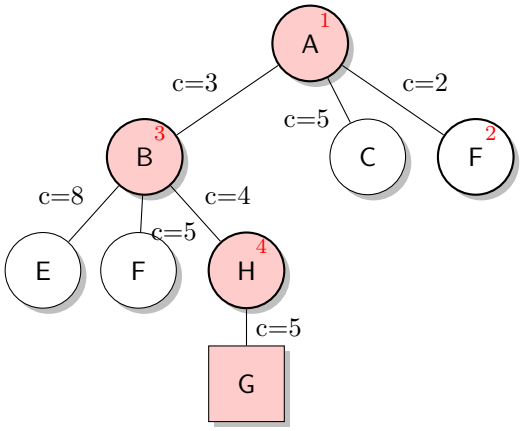


Depth-first without elimination of repeated states.



Depth-first with elimination of repeated states. The same as above.

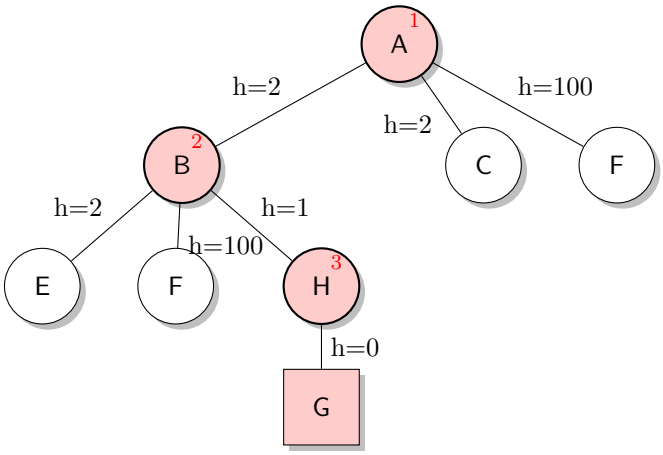
Uniform-cost without elimination of repeated states.



In case of ties, nodes where considered using a LIFO strategy.

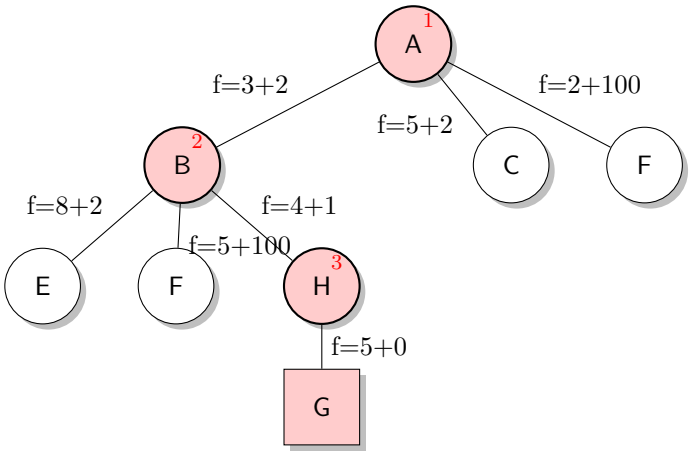
Uniform-cost without elimination of repeated states. The same as above.

Greedy best-first without elimination of repeated states.



Greedy best-first with elimination of repeated states. The same as above.

A^* without elimination of repeated states.



A^* with elimination of repeated states. The same as above.

Is the heuristic h admissible? Is it consistent? For what concerns the consistency of h, we can see that the cost of each edge is never smaller than the difference of the heuristic values of the source node and the one of the sink node. This implies consistency. When goal states have a heuristic value of zero, consistency implies admissibility.

Exercise 2.2

Suppose you are trying to solve the following puzzle. The puzzle involves numbers from 100 to 999.

You are given two numbers called S and G. You are also given a set of numbers called bad. A move consists of transforming one number into another by adding 1 to one of its digits or subtracting 1 from one of its digits; for instance, a move can take you from 678 to 679; or from 234 to 134. Moves are subject to the following constraints:

- You cannot add to the digit 9 or subtract from the digit 0. That is to say, no "carries" are allowed and the digits must remain in the range from 0 to 9.
- You cannot make a move which transforms your current number into one of the numbers in the set bad.
- You cannot change the same digit twice in two successive moves.

Since the numbers have only 3 digits, there are at most 6 possible moves at the start. And since all moves except the first are preceded by another move which uses one of the digits, after the start there are at most 4 possible moves per turn. You solve the puzzle by getting from S to G in the fewest possible moves. Your task is to use A* search to find a solution to the puzzle.

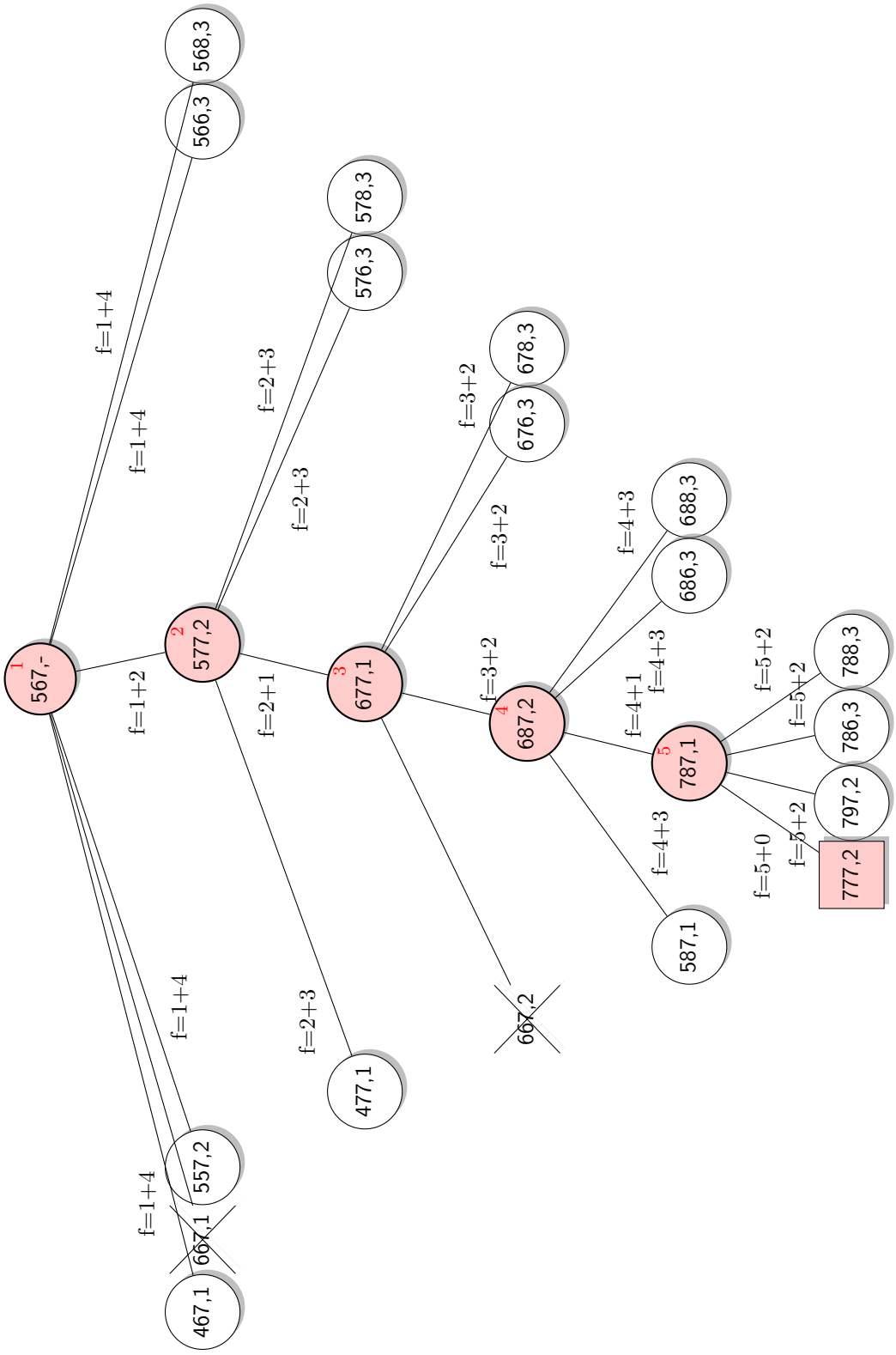
1. Briefly list the information needed in the state description in order to apply A* to this problem.
2. Find a heuristic for use with A* search in this problem which is admissible and which does not require extensive mathematical calculation. Explain clearly why your heuristic is admissible.
3. Use your heuristic to carry out an A* search to find a solution when S = 567, G = 777, and bad = [666; 667]. For nodes that tie for best-node-to-expand, choose the node with higher path cost.

Answer of exercise 2.2

State description. The state can be represented with three variables for storing the values of the three digits and a fourth variable to keep track of the last-changed digit.

Heuristic. An admissible heuristic for the given problem can be computed as the sum of the absolute differences between the digits in the current state and the digits in the goal state.

Search tree.

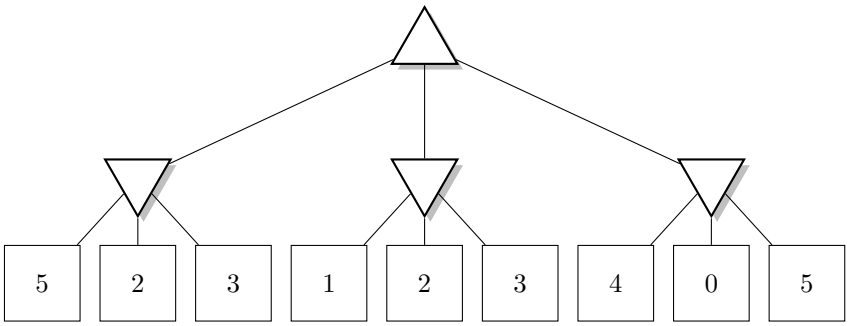

1
567,-
2
577,2
3
677,1
4
687,2
5
787,1
568,3
566,3
578,3
576,3
678,3
676,3
688,3
686,3
788,3
786,3
797,2
777,2
587,1
667,2
477,1
557,2
667,1
467,1
f=1+4
f=1+2
f=2+1
f=2+3
f=3+2
f=4+1
f=4+3
f=5+2
f=5+0

Chapter 3

Adversarial Search Strategies

Exercise 3.1

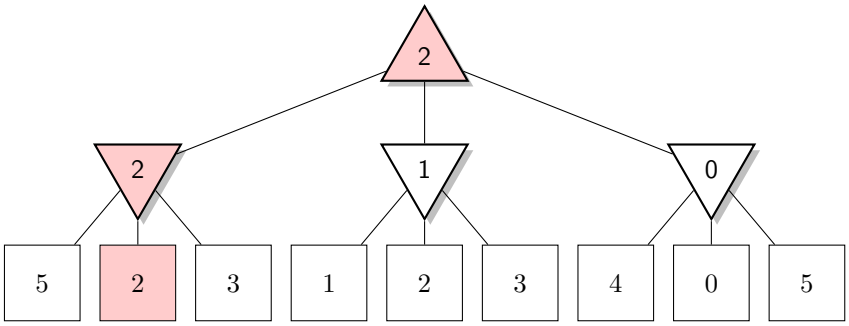
Consider the following tree representing a zero-sum game, where triangles pointing up are max nodes, triangles pointing down are min nodes, and squares are terminal nodes with the corresponding value of utility function for max player.



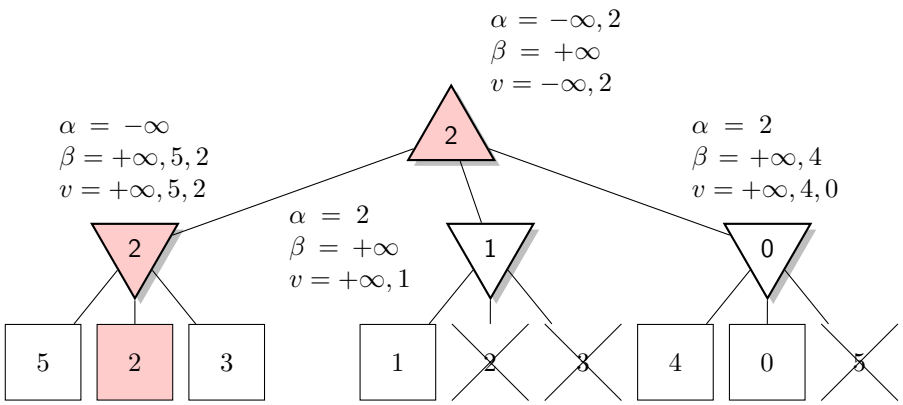
1. Apply the minimax algorithm for finding the best action for the max player at the root.
2. Apply the minimax algorithm with alpha-beta pruning for finding the best action for the max player at the root.

Answer of exercise 3.1

Minimax solution

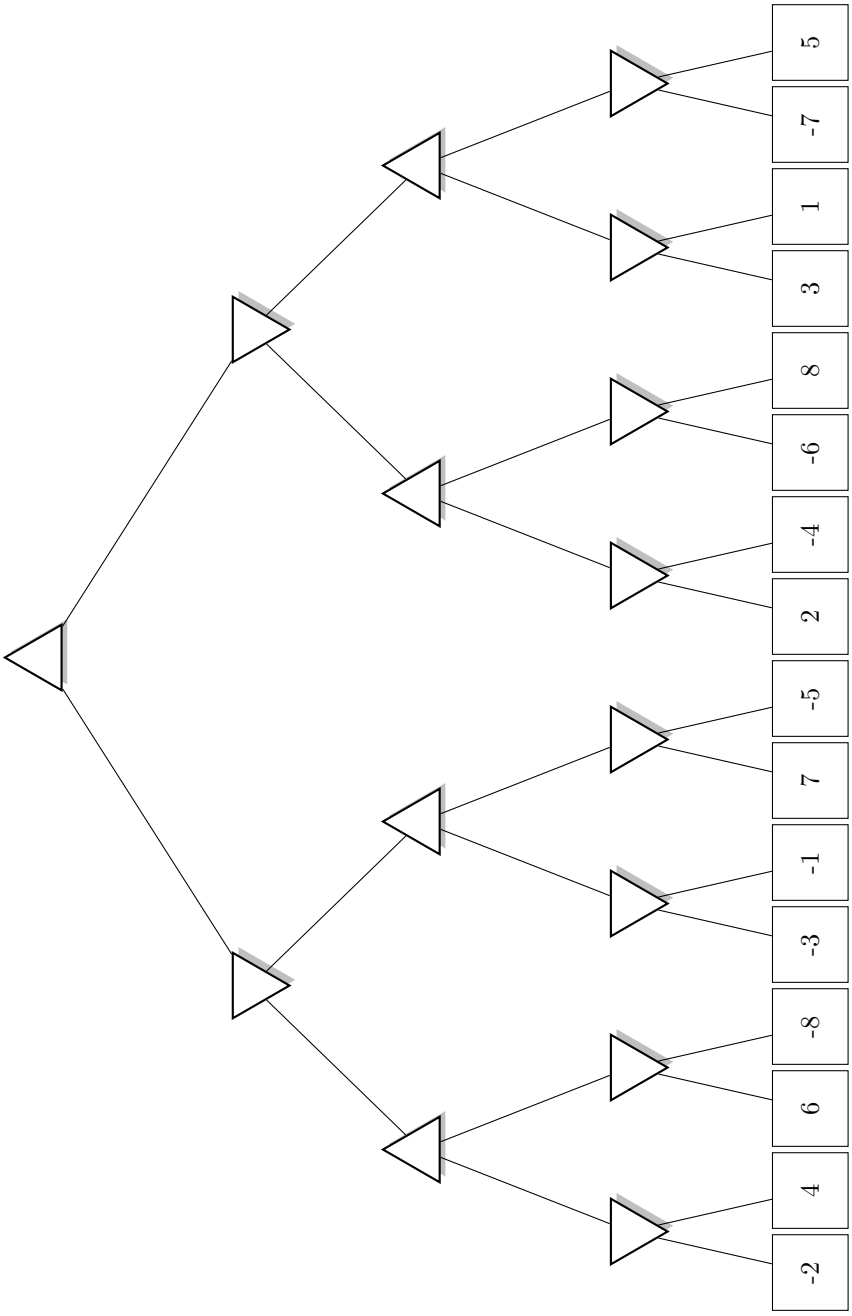


Minimax solution with alpha-beta pruning



Exercise 3.2

Consider the following tree representing a zero-sum game, where triangles pointing up are max nodes, triangles pointing down are min nodes, and squares are terminal nodes with the corresponding value of utility function for max player.

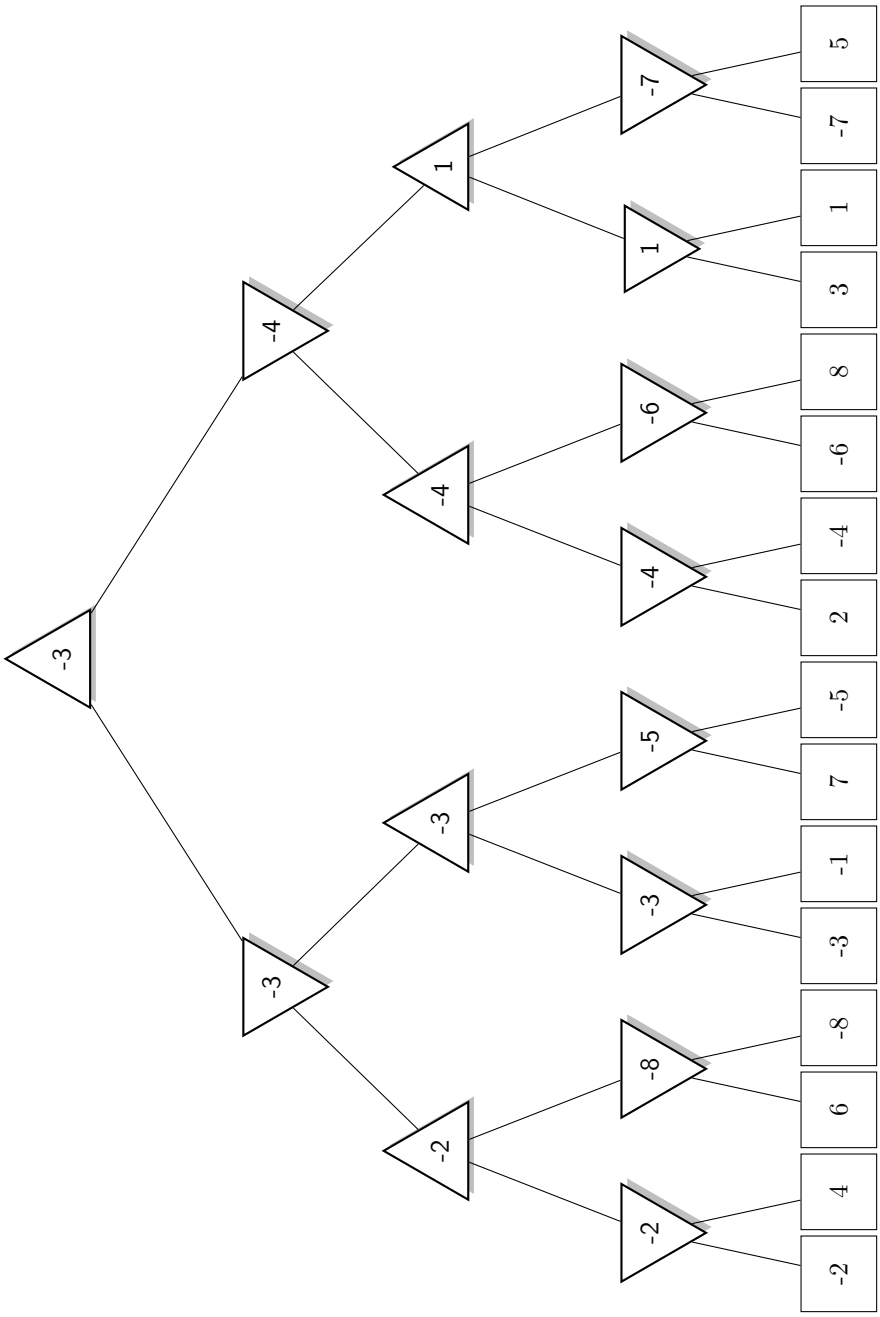


1. Apply the minimax algorithm for finding the best action for the max player at the root.

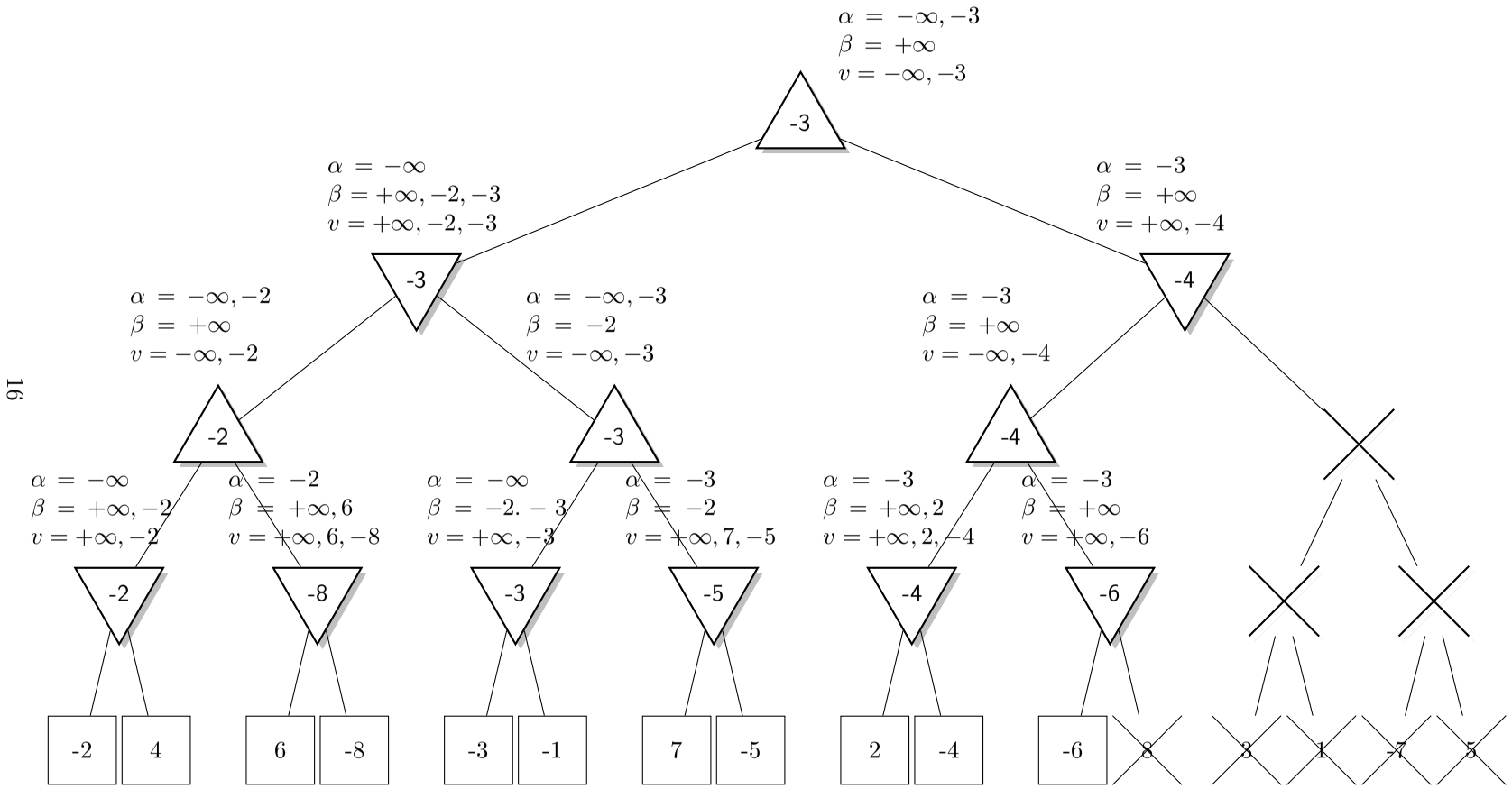
2. Apply the minimax algorithm with alpha-beta pruning for finding the best action for the max player at the root.

Answer of exercise 3.2

Minimax solution.



Minimax solution with alpha-beta pruning.


$\alpha = -\infty, -3$
$\beta = +\infty$
$v = -\infty, -3$
-3
$\alpha = -\infty$
$\beta = +\infty, -2, -3$
$v = +\infty, -2, -3$
$\alpha = -3$
$\beta = +\infty$
$v = +\infty, -4$
-3
-4
$\alpha = -\infty, -2$
$\beta = +\infty$
$v = -\infty, -2$
$\alpha = -\infty, -3$
$\beta = -2$
$v = -\infty, -3$
$\alpha = -3$
$\beta = +\infty$
$v = -\infty, -4$
-2
-3
-4
$\alpha = -\infty$
$\beta = +\infty, -2$
$v = +\infty, -2$
$\alpha = -2$
$\beta = +\infty, 6$
$v = +\infty, 6, -8$
$\alpha = -\infty$
$\beta = -2. - 3$
$v = +\infty, -3$
$\alpha = -3$
$\beta = -2$
$v = +\infty, 7, -5$
$\alpha = -3$
$\beta = +\infty, 2$
$v = +\infty, 2, -4$
$\alpha = -3$
$\beta = +\infty$
$v = +\infty, -6$
-2
-8
-3
-5
-4
-6
-2
4
6
-8
-3
-1
7
-5
2
-4
-6
8
3
1
-7
5

Exercise 3.3

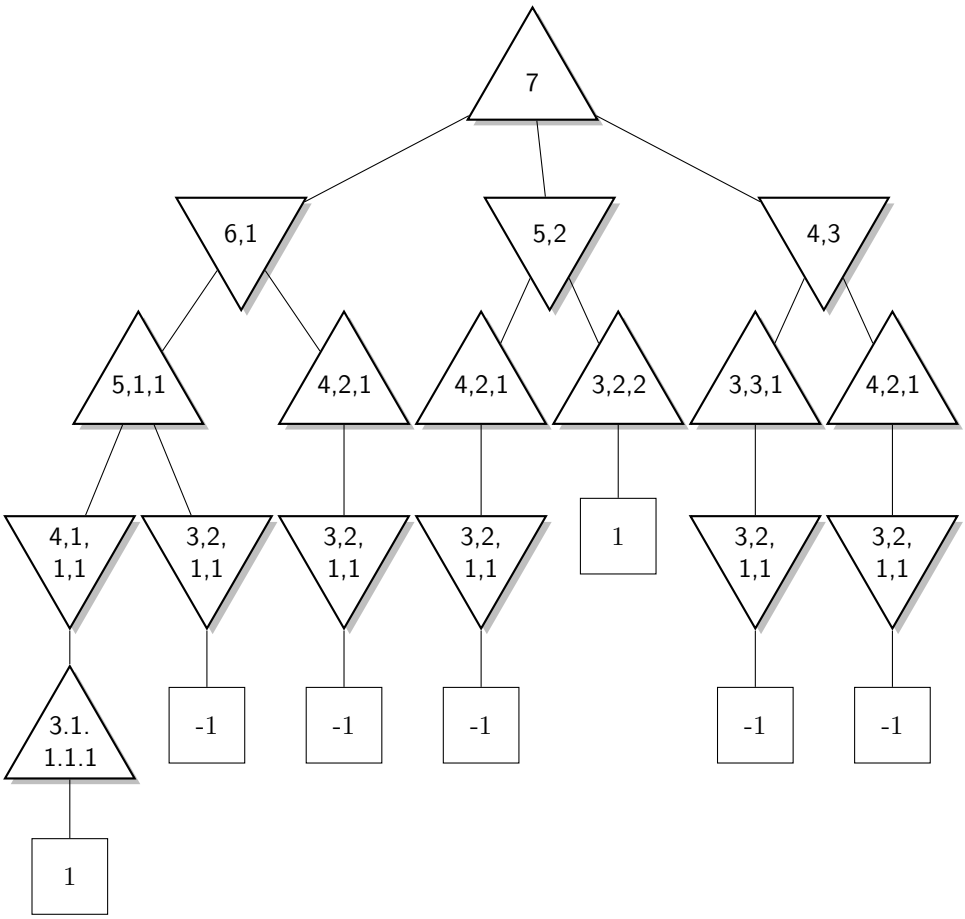
Consider the following two-player zero-sum game. The game begins with a pile of seven bricks. On your move, you must split one pile of bricks into two piles. You may not split a pile of bricks into two equal piles. If it is your turn and all the piles of bricks have either one or two bricks, you have lost the game.

1. Apply the minimax algorithm for finding the best action for the max player at the root.

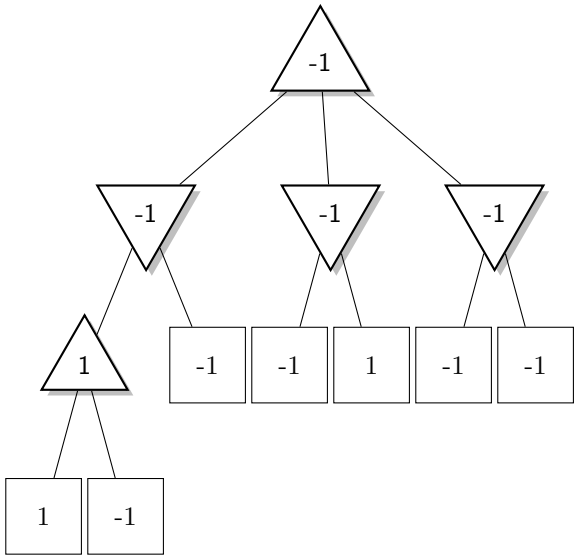
2. Apply the minimax algorithm with alpha-beta pruning for finding the best action for the max player at the root.

Answer of exercise 3.3

Here, you can find the minmax tree of the game:

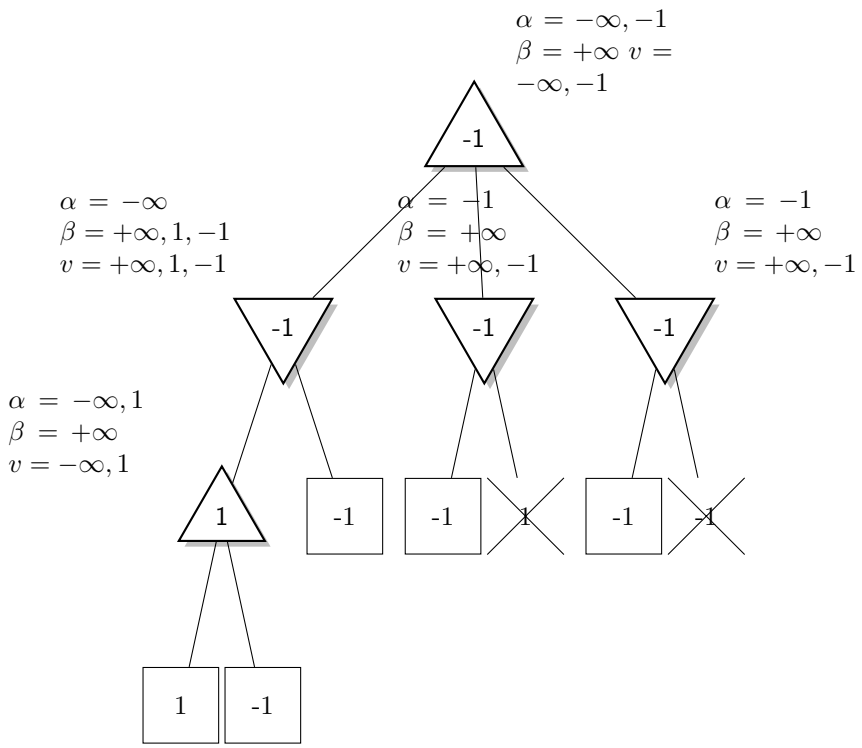


Minimax solution.



Player max will lose anyway.

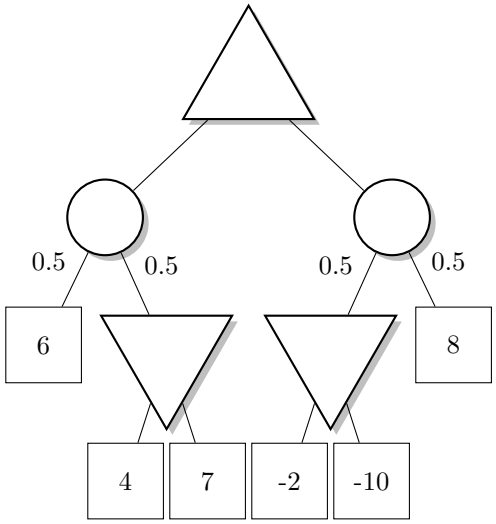
Minimax solution with alpha-beta pruning.



Exercise 3.4

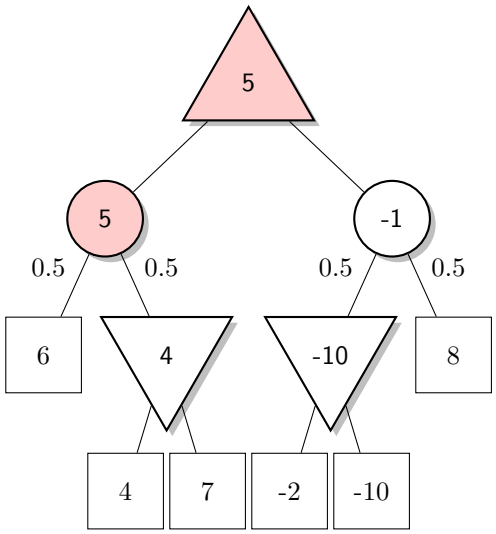
Let's consider the two-player zero-sum game with elements of chance represented by the following tree. Triangles pointing up are max nodes, triangles pointing down are min nodes, circles are chance nodes with the probabilities of reaching the next node reported on the outgoing edges, and squares are terminal nodes with the corresponding value of utility function for max player.

1. Compute the expectiminimax value of the root node and the action chosen by the max player.
2. Would max player change action if the largest payoff (8) were changed in 80?

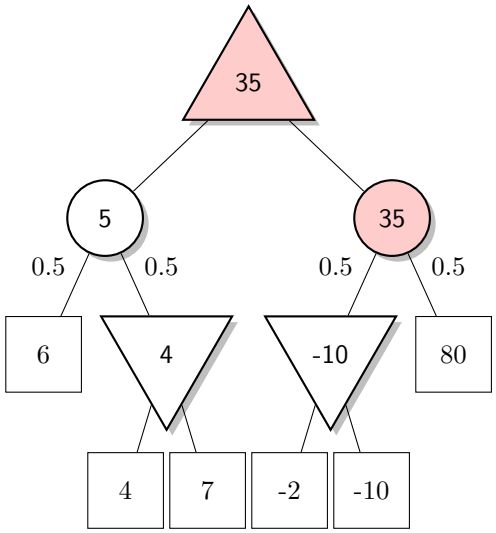


Answer of exercise 3.4

Expectiminimax.



Changing payoff 8 into 80.



As it can be noticed by the above tree, the action chosen by max player is changed. This represents a difference with respect to the minimax algorithm. In fact, in deterministic games, the strategy of a player is not affected by transformations of the payoffs that do not change their relative ordering.

Chapter 4

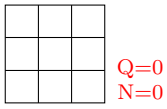
Monte Carlo Tree Search

Exercise 4.1

Show an example of the first seven iterations of the Monte Carlo Tree Search algorithm applied to the tic-tac-toe game. As initial state, consider the empty board. Using the *robust child* criterion (that is the most visited root child), which action is selected by the first player?

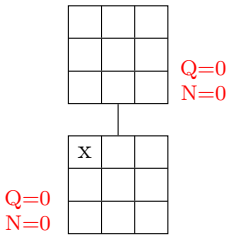
Answer of exercise 4.1

Below we show the initial state. Variable Q contains the sum of the payoffs and variable N counts the number of times that we have traversed the node.

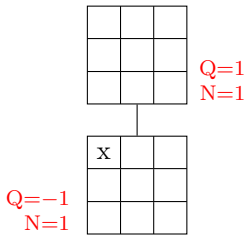


First iteration.

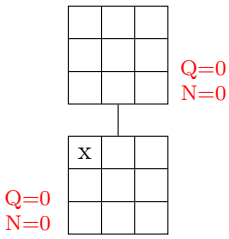
Selection: since there is only the root node, we select it for expansion.
Expansion: since the root node is not fully expanded, we generate one child.



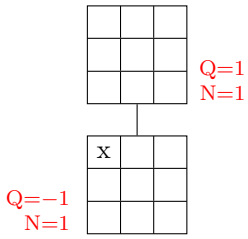
Simulation: starting from the newly expanded node, we generate a random match that terminates with a defeat for player x.
Update: we update the values of Q and N for the expanded node and all the nodes selected in this iteration.



Selection: since there is only the root node, we select it for expansion.
Expansion: since the root node is not fully expanded, we generate one child.

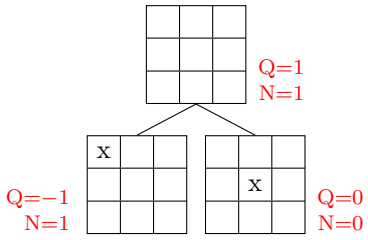


Simulation: starting from the newly expanded node, we generate a random match that terminates with a defeat for player x.
Update: we update the values of Q and N for the expanded node and all the nodes selected in this iteration.

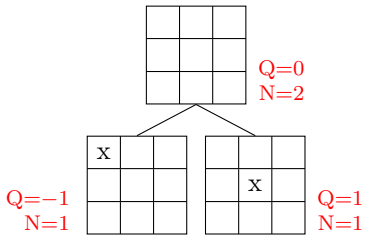


Second iteration.

Selection: since the root node is not fully expanded, we select it for expansion.
Expansion: we generate the second child of the root node.

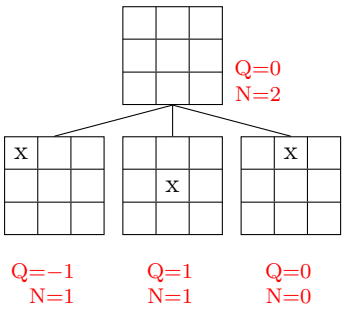


Simulation: starting from the newly expanded node, we generate a random match that terminates with a win for player x.
Update: we update the values of Q and N for the expanded node and all the nodes selected in this iteration.

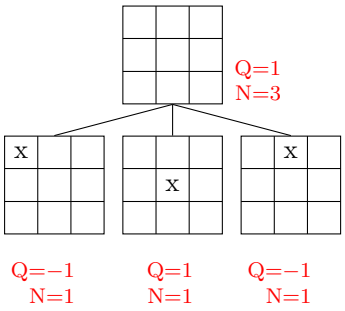


Third iteration.

Selection: since the root node is not fully expanded, we select it for expansion.
Expansion: since the root node is not fully expanded, we generate one child.



Expansion: starting from the newly expanded node, we generate a random match that terminates with a defeat for player x.
Update: we update the values of Q and N for the expanded node and all the nodes selected in this iteration.

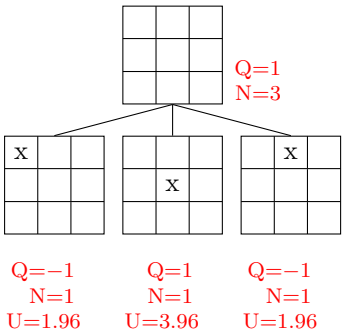


Fourth iteration.

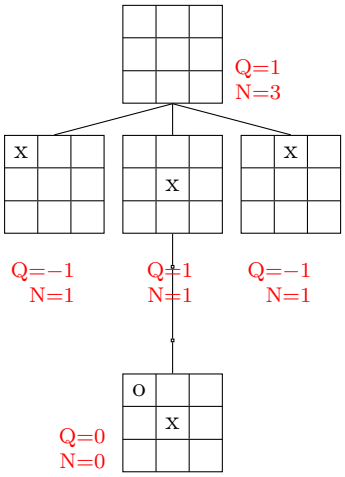
Selection: since the root node is now fully expanded (placing the x in any other cell can be reduced to one of these three cases), we have to select one of its children. To do this we compute for each child an upper confidence bound and we will select the node with the maximum value. The equation for computing the upper confidence bound U of a node is:

$$U = \frac{Q}{N} + 2\sqrt{\frac{2\ln(N_{parent})}{N}},$$

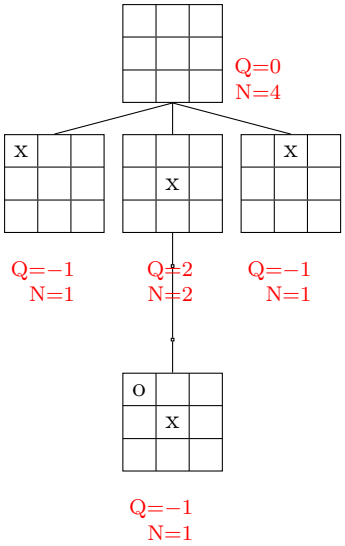
where N_{parent} is the number of visits to the parent node. Applying the above formula to the three nodes we get:



Expansion: we expand the node with the largest upper confidence bound.

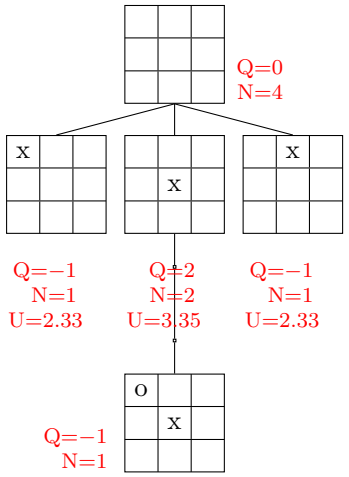


Simulation: starting from the newly expanded node, we generate a random match that terminates with a win for player x.
Update: we update the values of Q and N for the expanded node and all the nodes selected in this iteration.

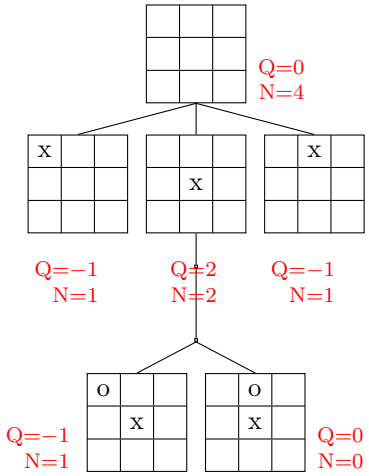


Fifth iteration.

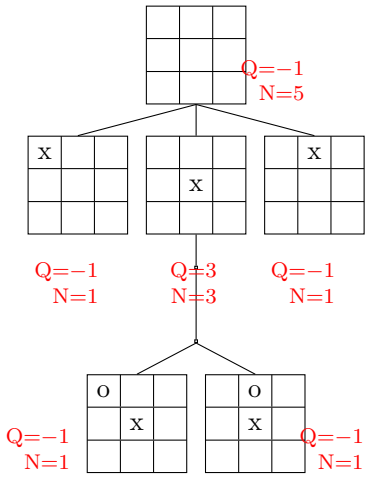
Selection: since the root node is now fully expanded, we have to select the child with the largest upper confidence bound:



Since the node with the largest upper confidence bound is not fully expanded, we select it for expansion.
Expansion: we expand the node selected in the previous step.

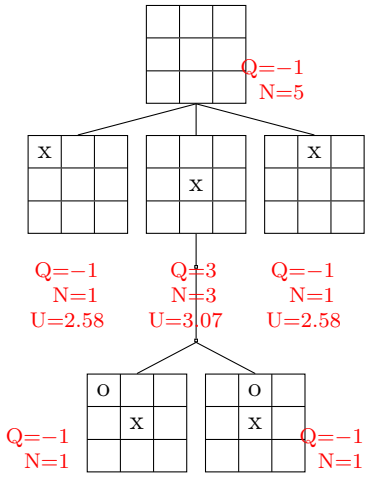


Simulation: starting from the newly expanded node, we generate a random match that terminates with a win for player x.
Update: we update the values of Q and N for the expanded node and all the nodes selected in this iteration.

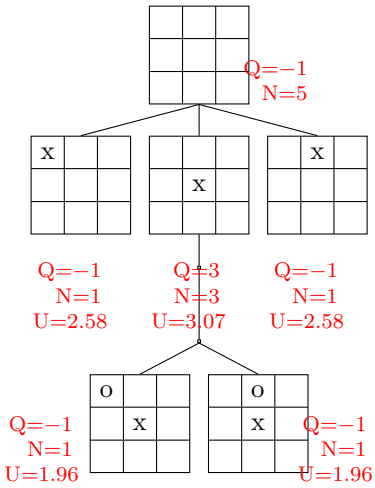


Sixth iteration.

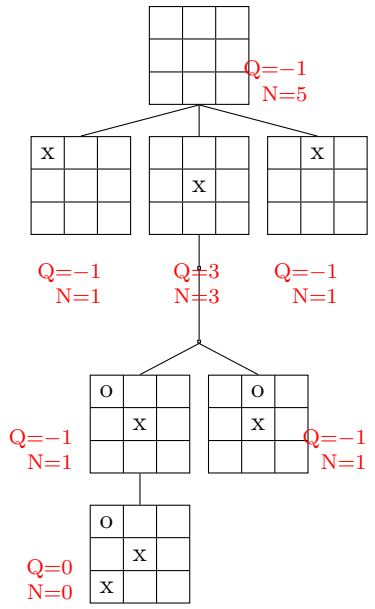
Selection: since the root node is now fully expanded, we have to select the child with the largest upper confidence bound:



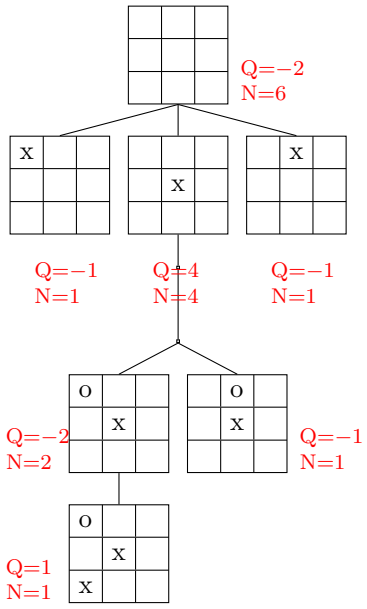
Since the node with the largest upper confidence bound is fully expanded, we need to select one of its children:



Since the two nodes have the same upper confidence bound, we can choose arbitrarily. We select the node on the left.
Expansion: we expand the node selected in the previous step.

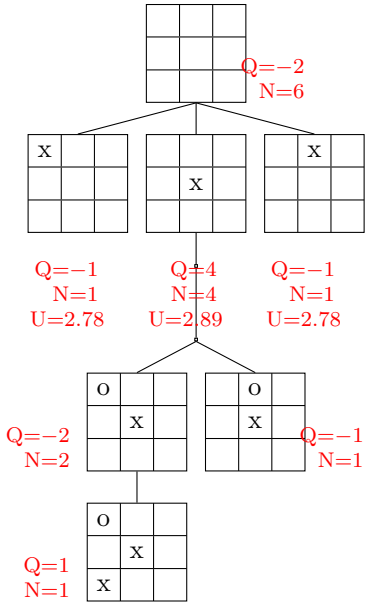


Simulation: starting from the newly expanded node, we generate a random match that terminates with a win for player x.
Update: we update the values of Q and N for the expanded node and all the nodes selected in this iteration.

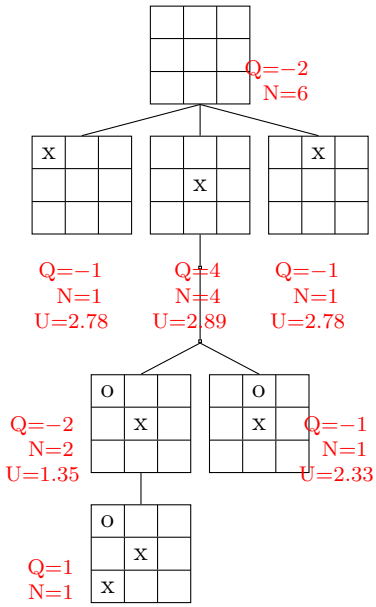


Seventh iteration.

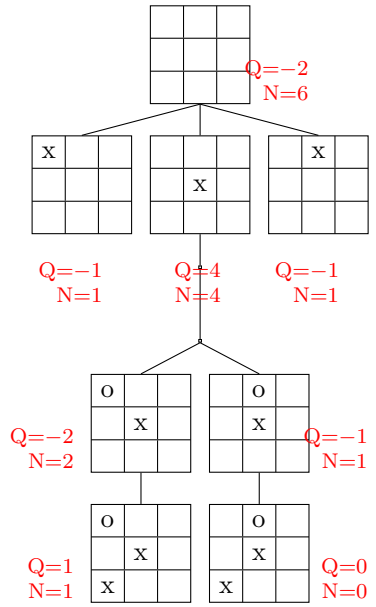
Selection: since the root node is now fully expanded, we have to select the child with the largest upper confidence bound:



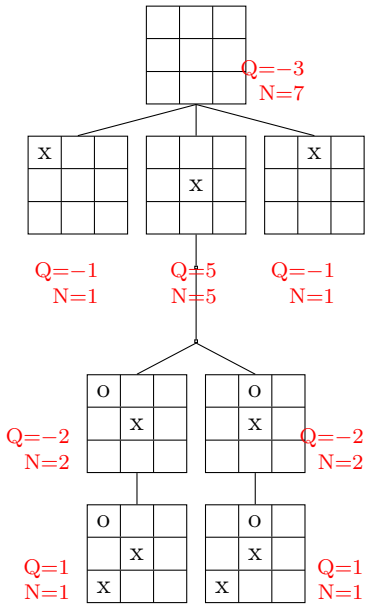
Since the node with the largest upper confidence bound is fully expanded, we need to select one of its children:



We select the child on the right that has the largest upper confidence bound.
Expansion: we expand the node selected in the previous step.



Simulation: starting from the newly expanded node, we generate a random match that terminates with a win for player x.
Update: we update the values of Q and N for the expanded node and all the nodes selected in this iteration.



Action selection.

Since the most visited root child is the second one (five visits), the first player will place her symbol in the middle of the grid.

Chapter 5

Constraint Satisfaction Problems

Exercise 5.1

Consider the following constraint satisfaction problem (CSP):

Variables: X_1, X_2, X_3

Domains: $D_1 = \{1, 2, 3, 4\}$, $D_2 = \{\alpha, \beta, \gamma\}$, $D_3 = \{a, b\}$

Constraints:

$C(X_1, X_2) = \{\langle 1, \alpha\rangle, \langle 2, \gamma\rangle, \langle 3, \alpha\rangle, \langle 4, \gamma\rangle\}$
$C(X_1, X_3) = \{\langle 1, a\rangle, \langle 3, a\rangle, \langle 4, b\rangle\}$
$C(X_2, X_3) = \{\langle \alpha, b\rangle, \langle \beta, a\rangle, \langle \gamma, b\rangle\}$

Apply the arc consistency algorithm AC3 to the problem and report the resulting domains.

Answer of exercise 5.1

The initial queue of arcs is:
$Q = \{X_1 \to X_2, X_2 \to X_1, X_1 \to X_3, X_3 \to X_1, X_2 \to X_3, X_3 \to X_2\}$
$X_1 \to X_2$: nothing
$Q = \{X_2 \to X_1, X_1 \to X_3, X_3 \to X_1, X_2 \to X_3, X_3 \to X_2\}$
$X_2 \to X_1$: $D_2 = \{\alpha, \gamma\}$
$Q = \{X_1 \to X_3, X_3 \to X_1, X_2 \to X_3, X_3 \to X_2, X_3 \to X_2\}$
$X_1 \to X_3$: $D_1 = \{1, 3, 4\}$
$Q = \{X_3 \to X_1, X_2 \to X_3, X_3 \to X_2, X_3 \to X_2, X_2 \to X_1\}$
$X_3 \to X_1$: nothing
$Q = \{X_2 \to X_3, X_3 \to X_2, X_3 \to X_2, X_2 \to X_1\}$
$X_2 \to X_3$: nothing
$Q = \{X_3 \to X_2, X_3 \to X_2, X_2 \to X_1\}$
$X_3 \to X_2$: $D_3 = \{b\}$
$Q = \{X_3 \to X_2, X_2 \to X_1, X_1 \to X_3\}$
$X_3 \to X_2$: nothing
$Q = \{X_2 \to X_1, X_1 \to X_3\}$
$X_2 \to X_1$: nothing

$Q = \{X_1 \to X_3\}$
$X_1 \to X_3$: $D_1 = \{4\}$
$Q = \{X_2 \to X_1\}$
$X_2 \to X_1$: $D_2 = \{\gamma\}$
$Q = \{\}$

The domain of the three variables are $D_1 = \{4\}$, $D_2 = \{\gamma\}$, $D_3 = \{b\}$.

Exercise 5.2

Consider the following constraint satisfaction problem (CSP):

Variables: X_1, X_2, X_3

Domains: $D_1 = \{1, 2, 3, 4\}$, $D_2 = \{a, b, c\}$, $D_3 = \{\alpha, \beta, \gamma\}$

Constraints:
$C(X_1, X_2) = \{\langle 1, a\rangle, \langle 2, b\rangle, \langle 3, a\rangle, \langle 3, b\rangle, \langle 4, b\rangle\}$
$C(X_1, X_3) = \{\langle 1, \beta\rangle, \langle 3, \beta\rangle, \langle 4, \beta\rangle\}$
$C(X_2, X_3) = \{\langle a, \gamma\rangle, \langle b, \beta\rangle, \langle b, \alpha\rangle, \langle c, \gamma\rangle\}$

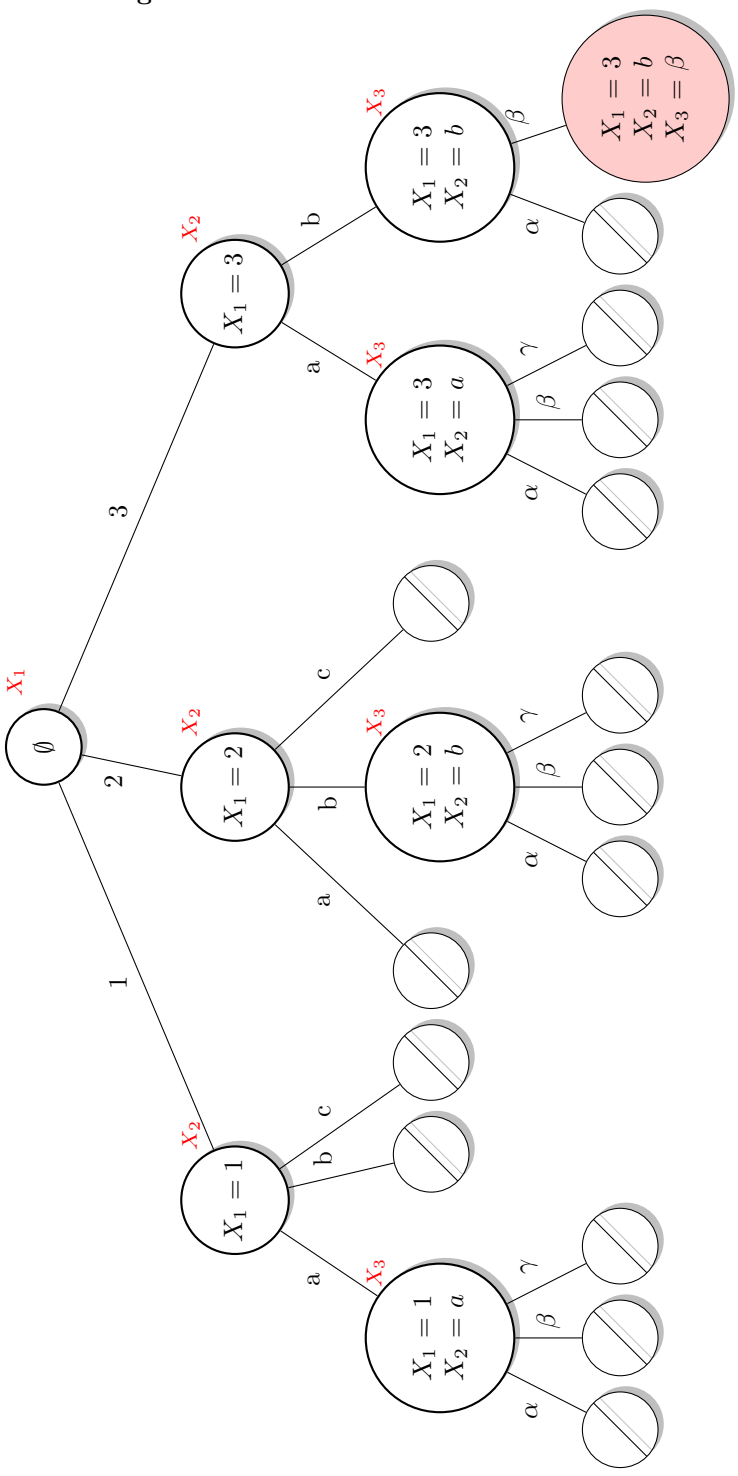
Solve the problem using:

1. backtracking;
2. backtracking and forward checking;
3. backtracking with minimum-remaining-values heuristic and forward checking;
4. backtracking with minimum-remaining-values and least-constraining-value heuristics and forward checking.

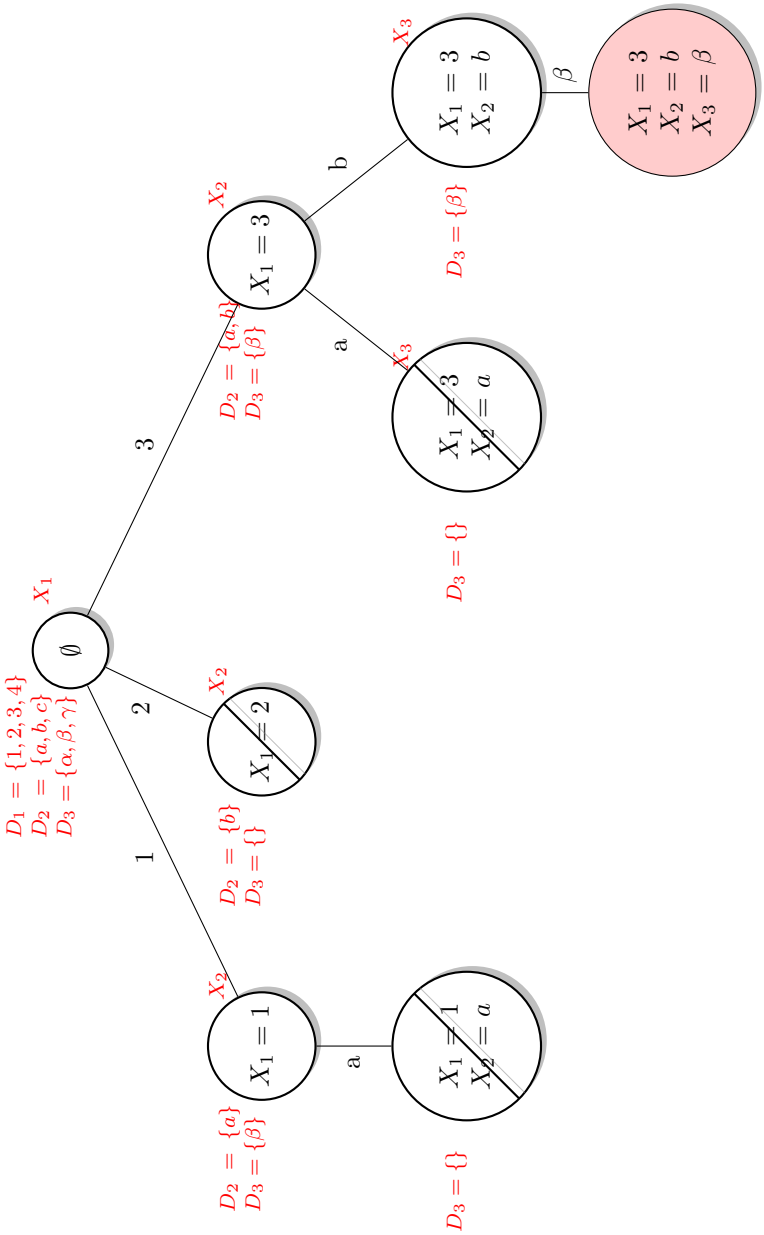
Report the search tree, the domains after each application of forward checking, and the solution found. Choose variables and values at random. Only one solution is required.

Answer of exercise 5.2

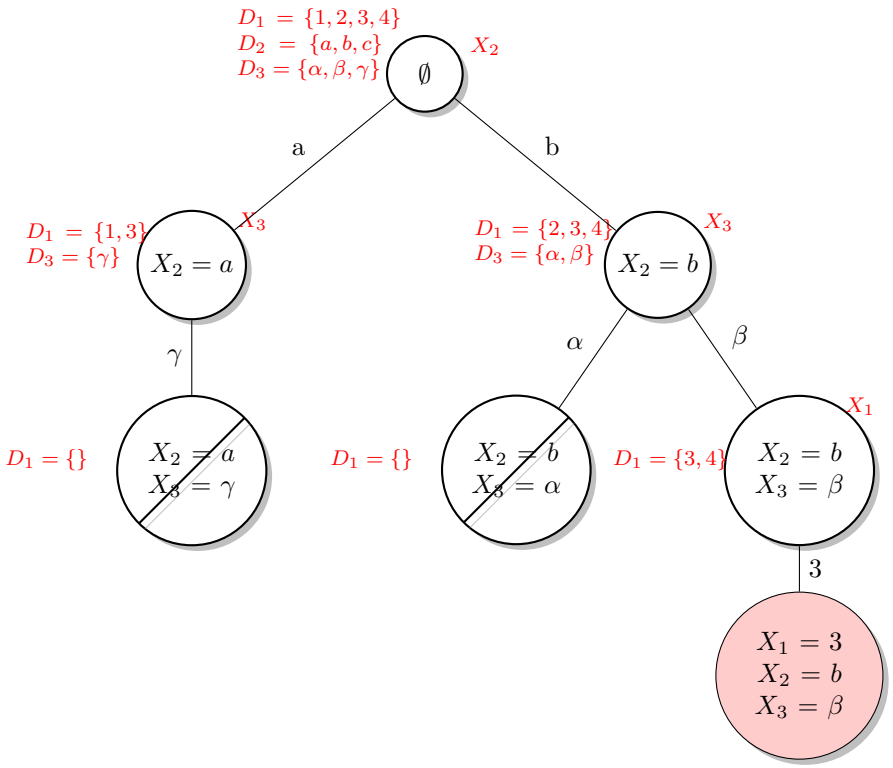
Backtracking.



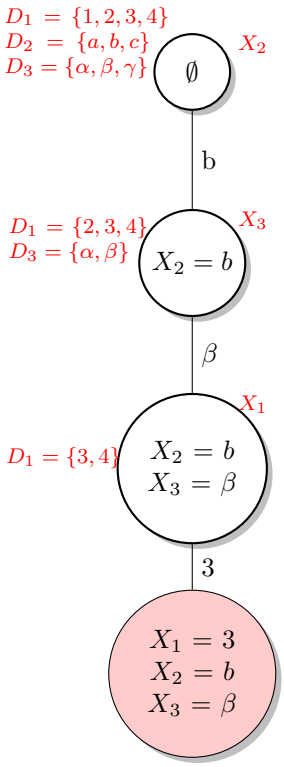
Backtracking and forward checking.



Backtracking with minimum-remaining-values heuristic and forward checking.



Backtracking with minimum-remaining-values and least-constraining-value heuristics and forward checking.



Exercise 5.3

Solve the 4-Queens problem. The problem consists of placing 4 queens on a 4x4 chess board so that no queen can attack any other. Formulate the problem as a constraint satisfaction problem and solve it using backtracking with minimum-remaining-values heuristic and forward checking. Only one solution is required.

Answer of exercise 5.3

Each row and each column will contain one and only one queen. We can use four variables X_1, X_2, X_3, X_4, one for each variable, that represent the row of the corresponding queen. For example, X_1 is the row of the queen in the first column. All the variables have the same domain: $D_1 = D_2 = D_3 = D_4 = \{1,2,3,4\}$. We can specify the constraints among these four variables by listing the set of pairs that are feasible:

$$C(X_1, X_2) = \{\langle 1,3\rangle, \langle 1,4\rangle, \langle 2,4\rangle, \langle 3,1\rangle, \langle 4,1\rangle, \langle 4,2\rangle\}$$

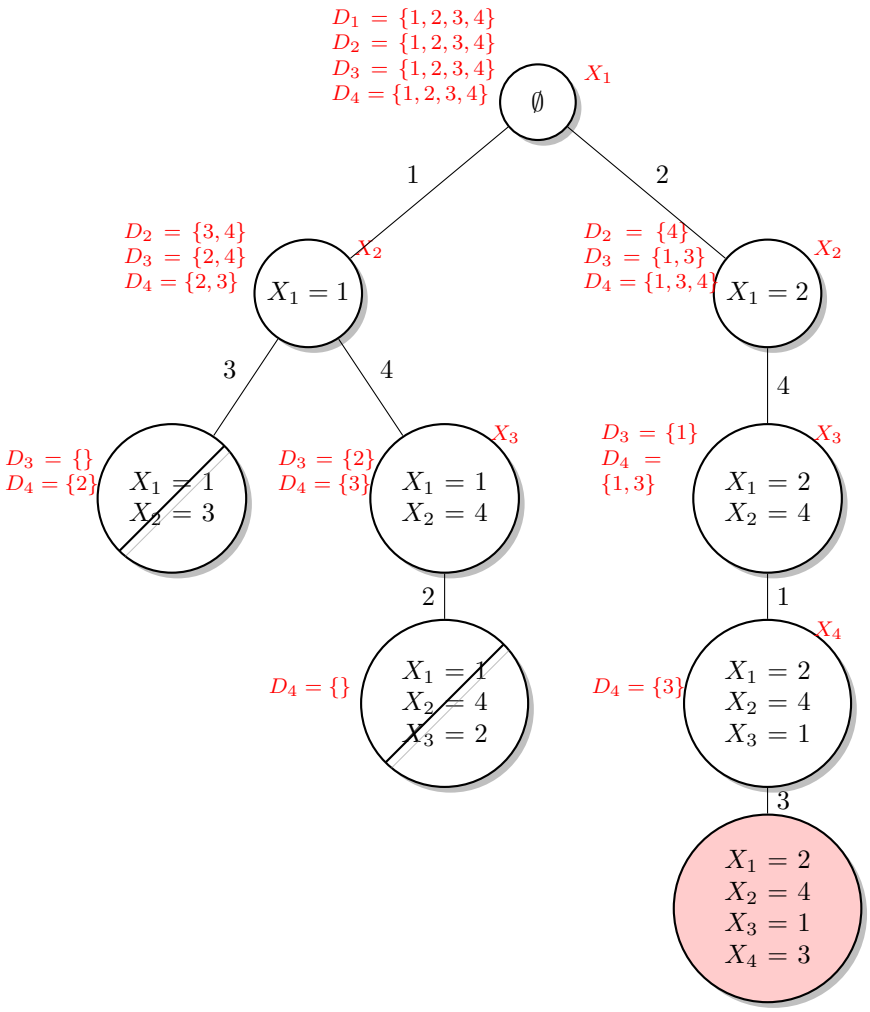
$$C(X_1, X_3) = \{\langle 1,2\rangle, \langle 1,4\rangle, \langle 2,1\rangle, \langle 2,3\rangle, \langle 3,2\rangle, \langle 3,4\rangle, \langle 4,1\rangle, \langle 4,3\rangle\}$$

$$C(X_1, X_4) = \{\langle 1,2\rangle, \langle 1,3\rangle, \langle 2,1\rangle, \langle 2,3\rangle, \langle 2,4\rangle, \langle 3,1\rangle, \langle 3,2\rangle, \langle 3,4\rangle, \langle 4,2\rangle, \langle 4,3\rangle\}$$

$$C(X_2, X_3) = C(X_1, X_2)$$

$$C(X_2, X_4) = C(X_1, X_3)$$

$$C(X_3, X_4) = C(X_1, X_2)$$


$D_1 = \{1,2,3,4\}$
$D_2 = \{1,2,3,4\}$
$D_3 = \{1,2,3,4\}$
$D_4 = \{1,2,3,4\}$
X_1
$\emptyset$
1
2
$D_2 = \{3,4\}$
$D_3 = \{2,4\}$
$D_4 = \{2,3\}$
X_2
$X_1 = 1$
$D_2 = \{4\}$
$D_3 = \{1,3\}$
$D_4 = \{1,3,4\}$
X_2
$X_1 = 2$
3
4
4
$D_3 = \{\}$
$D_4 = \{2\}$
$X_1 = 1$
$X_2 = 3$
$D_3 = \{2\}$
$D_4 = \{3\}$
X_3
$X_1 = 1$
$X_2 = 4$
$D_3 = \{1\}$
$D_4 = \{1,3\}$
X_3
$X_1 = 2$
$X_2 = 4$
2
1
$D_4 = \{\}$
$X_1 = 1$
$X_2 = 4$
$X_3 = 2$
$D_4 = \{3\}$
X_4
$X_1 = 2$
$X_2 = 4$
$X_3 = 1$
3
$X_1 = 2$
$X_2 = 4$
$X_3 = 1$
$X_4 = 3$

Chapter 6

Inference in Propositional Logic

Exercise 6.1

Consider the following formula in propositional logic:

$$U \Rightarrow (\neg T \Rightarrow (\neg S \wedge P))$$

Convert the above formula in conjunctive normal form (CNF), reporting all the steps and intermediate formulas.

Answer of exercise 6.1

1. Eliminate implications.

$$\neg U \vee (\neg T \Rightarrow (\neg S \wedge P))$$

$$\neg U \vee T \vee (\neg S \wedge P)$$

2. Distribute or over and:

$$(\neg U \vee T \vee \neg S) \wedge (\neg U \vee T \vee P)$$

Exercise 6.2

Consider the following formula in propositional logic:

$$(A \Leftrightarrow (B \wedge C))$$

Convert the above formula in conjunctive normal form (CNF), reporting all the steps and intermediate formulas.

Answer of exercise 6.2

1. Eliminate co-implications:

$$(A \Rightarrow (B \wedge C)) \wedge ((B \wedge C) \Rightarrow A)$$

2. Eliminate implications:

$$(\neg A \vee (B \wedge C)) \wedge (\neg(B \wedge C) \vee A)$$

3. Move not inwards:

$$(\neg A \vee (B \wedge C)) \wedge (\neg B \vee \neg C \vee A)$$

4. Distribute or over and:

$$(\neg A \vee B) \wedge (\neg A \vee C) \wedge (\neg B \vee \neg C \vee A)$$

Exercise 6.3

Consider the following formulas in propositional logic:

$$\Phi_1 = A \wedge (B \vee Q)$$

$$\Phi_2 = (A \wedge B) \vee (A \wedge Q)$$

Prove:

1. $\Phi_1 \models \Phi_2$, by using the resolution inference procedure
2. $\Phi_1 \models \Phi_2$, by using the DPLL algorithm
3. $\Phi_2 \models \Phi_1$, by using the resolution inference procedure
4. $\Phi_2 \models \Phi_1$, by using the DPLL algorithm

Answer of exercise 6.3

$\Phi_1 \models \Phi_2$, by using the resolution inference procedure.
First of all we have to write Φ_1 and $\neg\Phi_2$ in CNF.

$$\text{CNF}(\Phi_1) = A \wedge (B \vee Q)$$

$\text{CNF}(\neg\Phi_2) = \neg((A \wedge B) \vee (A \wedge Q)) = \neg(A \wedge B) \wedge \neg(A \wedge Q) = (\neg A \vee \neg B) \wedge (\neg A \vee \neg Q)$

Then we apply resolution inference:

1. A
2. $B \vee Q$
3. $\neg A \vee \neg B$
4. $\neg A \vee \neg Q$

---

5. $\neg B$ (1,3)
6. $\neg Q$ (1,4)
7. $\neg A \vee Q$ (2,3)

8. $\neg A \vee B$ (2,4)

---

9. Q (1,7)
10. B (1,8)
11. $\neg A$ (3,8)

---

12. $\{\}$ (1,11)

Since we found an empty clause the theorem is true.

$\Phi_1 \models \Phi_2$, by using the DPLL algorithm.

$\{\}$	$A \wedge (B \vee Q) \wedge (\neg A \vee \neg B) \wedge (\neg A \vee \neg Q)$	One-Literal on A
$\{A\}$	$(B \vee Q) \wedge (\neg B) \wedge (\neg Q)$	One-Literal on $\neg B$
$\{A, \neg B\}$	$Q \wedge \neg Q$	One-Literal on Q
$\{A, \neg B, Q\}$	$\{\}$	Unsatisfiable

$\Phi_2 \models \Phi_1$, by using the resolution inference procedure.
First of all we have to write $\neg\Phi_1$ and Φ_2 in CNF.

$$\text{CNF}(\Phi_2) = (A \wedge B) \vee (A \wedge Q) = A \wedge (A \vee Q) \wedge (A \vee B) \wedge (B \vee Q)$$

$$\text{CNF}(\neg\Phi_1) = \neg(A \wedge (B \vee Q)) = \neg A \vee (\neg B \wedge \neg Q) = (\neg A \vee \neg B) \wedge (\neg A \vee \neg Q)$$

Then we apply resolution inference:

1. A
2. $A \vee Q$
3. $A \vee B$
4. $B \vee Q$
5. $\neg A \vee \neg B$
6. $\neg A \vee \neg Q$

---

7. $\neg B$ (1,5)
8. $\neg Q$ (1,6)
9. $\neg B \vee Q$ (2,5)
10. $B \vee \neg Q$ (3,6)
11. $\neg A \vee Q$ (4,5)
12. $\neg A \vee B$ (4,6)

---

13. Q (1,11)

14. B (1,12)

15. $\neg A$ (5,12)

---

16. $\{\}$ (1,15)

Since we found an empty clause the theorem is true.

$\Phi_1 \models \Phi_2$, by using the DPLL algorithm.

$\{\}$	$A \wedge (A \vee Q) \wedge (A \vee B) \wedge (B \vee Q) \wedge (\neg A \vee \neg B) \wedge (\neg A \vee \neg Q)$	One-Literal on A
$\{A\}$	$(B \vee Q) \wedge (\neg B) \wedge (\neg Q)$	One-Literal on $\neg B$
$\{A, \neg B\}$	$Q \wedge \neg Q$	One-Literal on Q
$\{A, \neg B, Q\}$	$\{\}$	Unsatisfiable

Exercise 6.4

Given the following propositional formula:

$$(P \vee Q \vee \neg R) \wedge (P \vee \neg Q) \wedge \neg P \wedge R \wedge U$$

use the DPLL algorithm to find an assignment that satisfies the formula or to show that such assignment does not exist.

Answer of exercise 6.4

$\{\}$	$(P \vee Q \vee \neg R) \wedge (P \vee \neg Q) \wedge \neg P \wedge R \wedge U$	Pure-Literal on U
$\{U\}$	$(P \vee Q \vee \neg R) \wedge (P \vee \neg Q) \wedge \neg P \wedge R$	One-Literal on $\neg P$
$\{\neg P, U\}$	$(Q \vee \neg R) \wedge (\neg Q) \wedge R$	One-Literal on $\neg Q$
$\{\neg P, \neg Q, U\}$	$\neg R \wedge R$	One-Literal on $\neg R$
$\{\neg P, \neg Q, \neg R, U\}$	$\{\}$	Unsatisfiable

Exercise 6.5

Given the following propositional formula:

$$(P \vee Q) \wedge \neg Q \wedge (\neg P \vee Q \vee \neg R)$$

use the DPLL algorithm to find an assignment that satisfies the formula or to show that such assignment does not exist.

Answer of exercise 6.5

$\{\}$	$(P \vee Q) \wedge \neg Q \wedge (\neg P \vee Q \vee \neg R)$	Pure-Literal on $\neg R$
$\{\neg R\}$	$(P \vee Q) \wedge \neg Q$	Pure-Literal on P
$\{P, \neg R\}$	$\neg Q$	Pure-Literal on $\neg Q$
$\{P, \neg Q, \neg R\}$	True	Satisfiable

Exercise 6.6

Given the following propositional formula:

$$(P \vee Q) \wedge (P \vee \neg Q) \wedge (\neg P \vee Q) \wedge (\neg P \vee \neg R)$$

use the DPLL algorithm to find an assignment that satisfies the formula or to show that such assignment does not exist.

Answer of exercise 6.6

$\{\}$	$(P \vee Q) \wedge (P \vee \neg Q) \wedge (\neg P \vee Q) \wedge (\neg P \vee \neg R)$	Pure-Literal on $\neg R$
$\{\neg R\}$	$(P \vee Q) \wedge (P \vee \neg Q) \wedge (\neg P \vee Q)$	Split on P
$s' = \{\neg P, \neg R\}$	$Q \wedge \neg Q$	One-Literal on Q
$s' = \{\neg P, Q, \neg R\}$	$\{\}$	Unsatisfiable
$s'' = \{P, \neg R\}$	Q	Pure-Literal on Q
$s'' = \{P, Q, \neg R\}$	True	Satisfiable

Chapter 7

Planning

Exercise 7.1

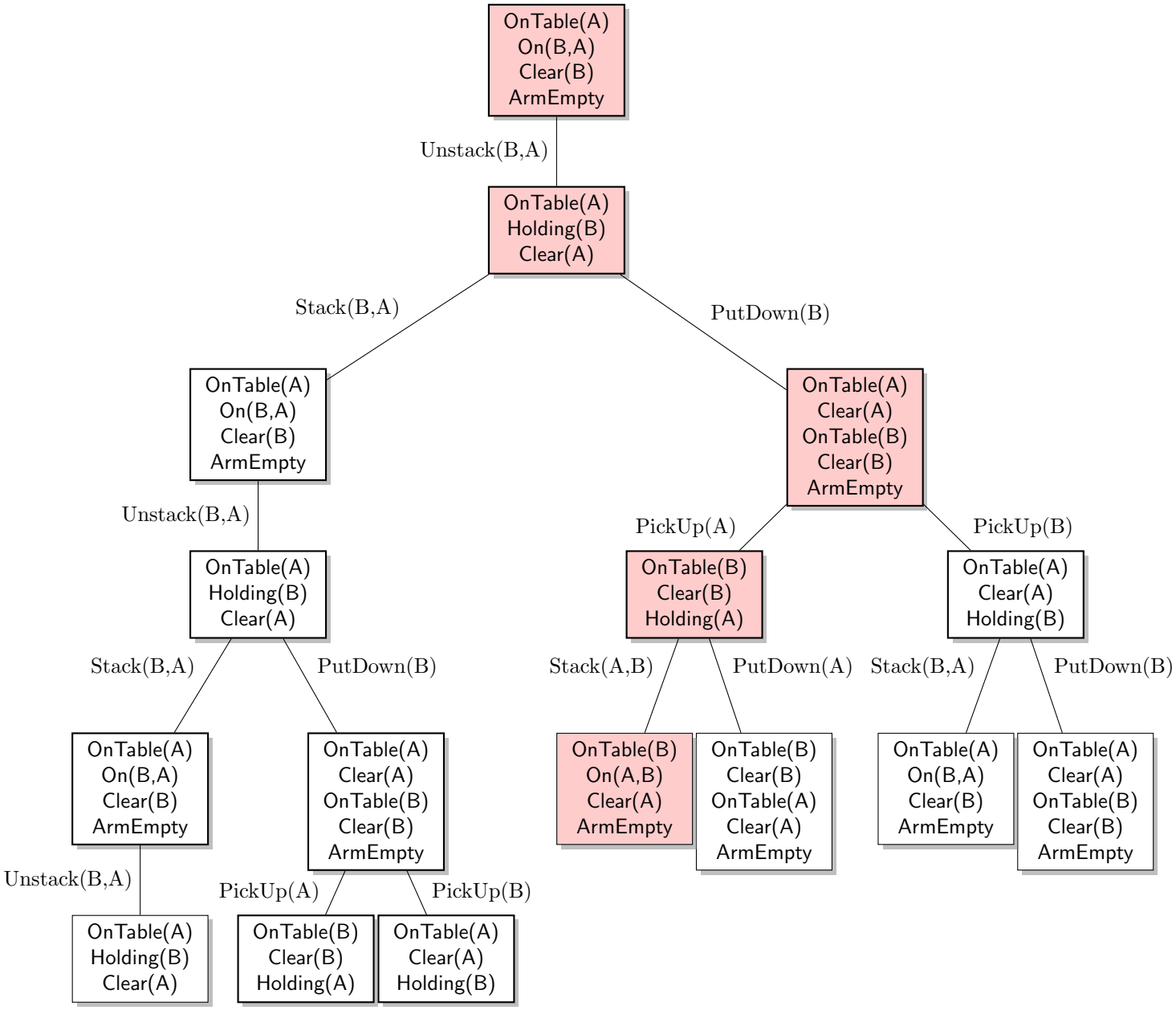
Consider the problem of using a robotic arm with gripper to move two blocks, A and B, from the initial state shown below on the left to the final one shown on the right:

This problem is modeled as the following STRIPS planning problem:

```
Init(OnTable(A) ∧ On(B,A) ∧ Clear(B) ∧ ArmEmpty)
Goal(OnTable(B) ∧ On(A,B) ∧ Clear(A) ∧ ArmEmpty)
Action(Stack(x,y),
 Precond: Holding(x) ∧ Clear(y)
 Effect: ¬Holding(x) ∧ ¬Clear(y) ∧ On(x,y) ∧ Clear(x)
         ∧ ArmEmpty)
Action(UnStack(x,y),
 Precond: On(x,y) ∧ Clear(x) ∧ ArmEmpty
 Effect: ¬On(x,y) ∧ ¬Clear(x) ∧ ¬ArmEmpty ∧ Holding(x)
         ∧ Clear(y))
Action(PutDown(x),
 Precond: Holding(x)
 Effect: ¬Holding(x) ∧ OnTable(x) ∧ Clear(x) ∧ ArmEmpty)
Action(PickUp(x),
 Precond: OnTable(x) ∧ Clear(x) ∧ ArmEmpty
 Effect: ¬OnTable(x) ∧ ¬Clear(x) ∧ ¬ArmEmpty) ∧ Holding(x))
```

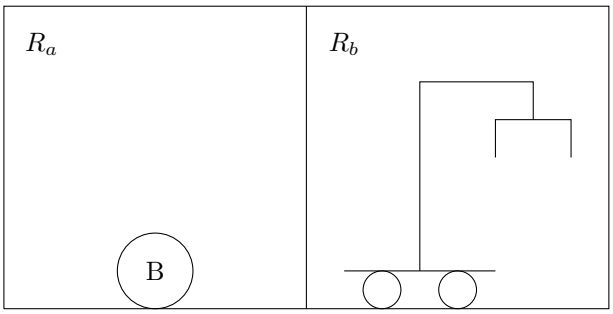
Solve the planning problem using forward search in the state space (using breadth-first search without elimination of repeated states), drawing the search tree and showing the solution.

Answer of exercise 7.1



Exercise 7.2

Robby is a mobile robot with a gripper that moves in a world with two rooms (R_a and R_b) where it is present a ball P. Roby can take the following actions: move from one room to the other one, pick the ball from the floor with the gripper and drop the ball in the gripper to the floor. The initial state is the following:

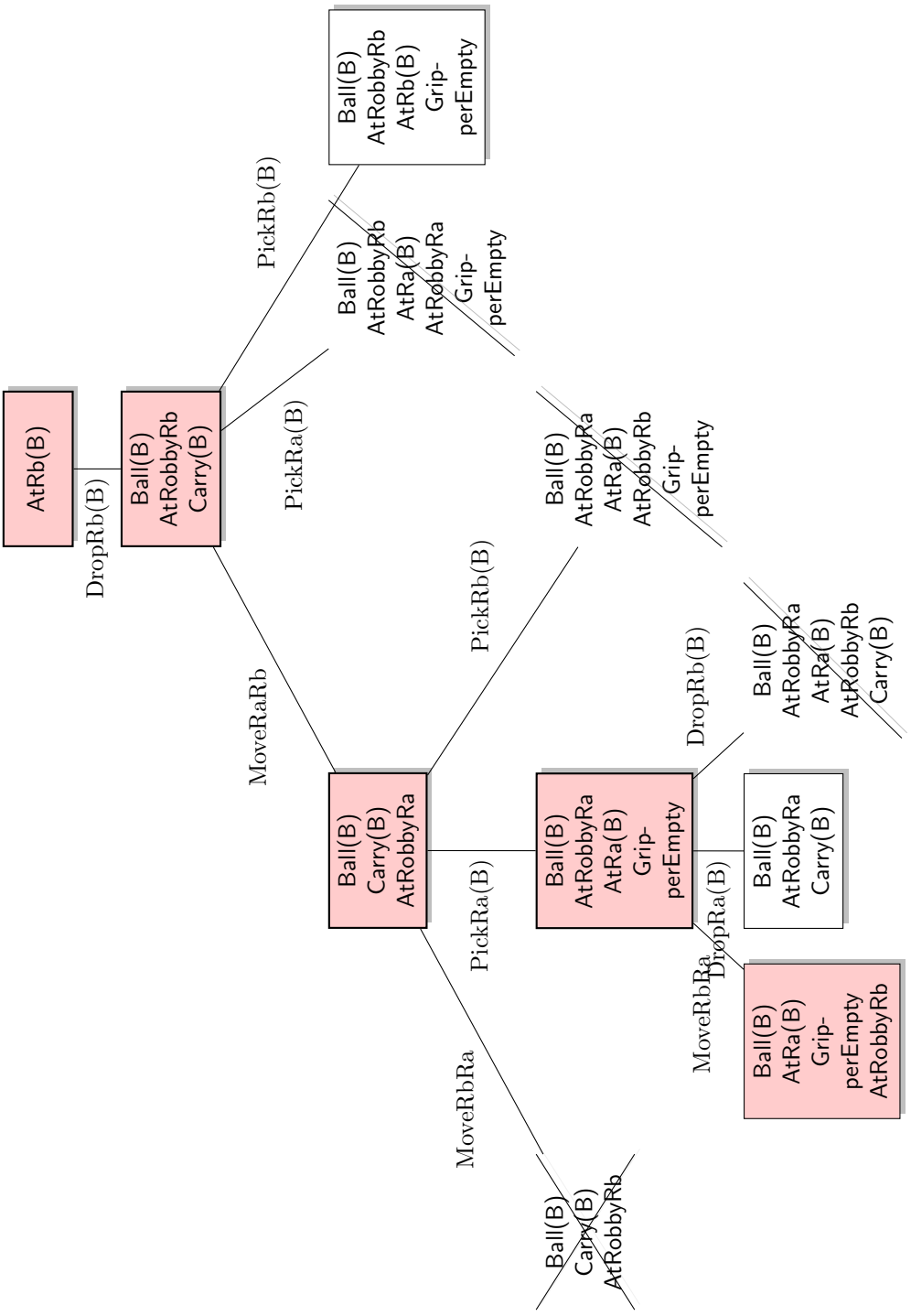


We want Robby to bring the ball into room R_b. This problem is modeled as the following STRIPS planning problem:

```
Init(Ball(B) ∧ AtRa(B) ∧ AtRobbyRb ∧ GripperEmpty)
Goal(AtRb(B))
Action(MoveRaRb,
 Precond: AtRobbyRa
 Effect: ¬AtRobbyRa ∧ AtRobbyRb)
Action(MoveRbRa,
 Precond: AtRobbyRb
 Effect: ¬AtRobbyRb ∧ AtRobbyRa)
Action(PickRa(o),
 Precond: Ball(o) ∧ AtRa(o) ∧ AtRobbyRa ∧ GripperEmpty
 Effect: ¬AtRa(o) ∧ ¬GripperEmpty ∧ Carry(o))
Action(PickRb(o),
 Precond: Ball(o) ∧ AtRb(o) ∧ AtRobbyRb ∧ GripperEmpty
 Effect: ¬AtRb(o) ∧ ¬GripperEmpty ∧ Carry(o))
Action(DropRa(o),
 Precond: Ball(o) ∧ AtRobbyRa ∧ Carry(o)
 Effect: ¬Carry(o) ∧ AtRa(o) ∧ GripperEmpty)
Action(DropRb(o),
 Precond: Ball(o) ∧ AtRobbyRb ∧ Carry(o)
 Effect: ¬Carry(o) ∧ AtRb(o) ∧ GripperEmpty)
```

Solve the planning problem using backward search in the state space (using depth-first search with elimination of repeated states), drawing the search tree and showing the solution.

Answer of exercise 7.2



The goal state with a cross is a repeated state. The goal states with a diagonal line are inconsistent goal states.

Exercise 7.3

Consider the Hanoi tower problem. This puzzle has two discs, D1, D2, with holes in their centers, and three pegs, A, B, C, on which the discs can be placed. Disc D2 is larger than disc D1. Initially, all the discs are on peg A, with D2 on the bottom and D1 on top. We want them on peg C in the same configuration.

The following rules apply:

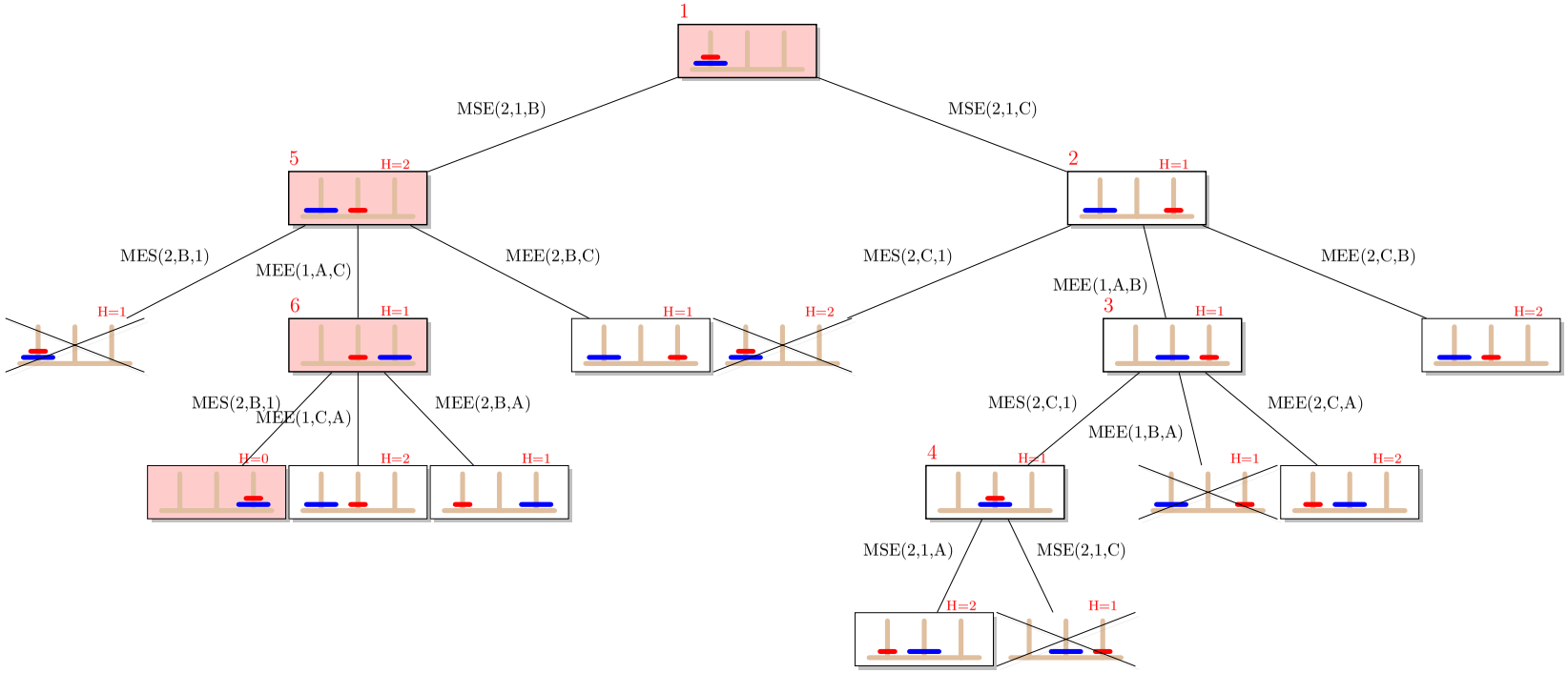
- only the top disc on a peg can be moved;
- a disc cannot be placed on top of a smaller one.

The problem can be modeled using the STRPS notation as follows:

```
Init(Smaller(2,1) ∧ In(1,A) ∧ On(2,1) ∧ Free(2) ∧
     Empty(B) ∧ Empty(C))
Goal(Empty(A) ∧ Empty(B))
Action(MoveStackStack(x,y,z),
 Precond: On(x,y) ∧ Free(x) ∧ Free(z) ∧ Smaller(x,z)
 Effect: ¬On(x,y) ∧ ¬Free(z) ∧ Free(y) ∧ On(x,z))
Action(MoveStackEmpty(x,y,z),
 Precond: On(x,y) ∧ Free(x) ∧ Empty(z)
 Effect: ¬On(x,y) ∧ ¬Empty(z) ∧ Free(y) ∧ In(x,z))
Action(MoveEmptyStack(x,y,z),
 Precond: In(x,y) ∧ Free(x) ∧ Free(z) ∧ Smaller(x,z)
 Effect: ¬In(x,y) ∧ ¬Free(z) ∧ Empty(y) ∧ On(x,z))
Action(MoveEmptyEmpty(x,y,z),
 Precond: In(x,y) ∧ Free(x) ∧ Empty(z)
 Effect: ¬In(x,y) ∧ ¬Empty(z) ∧ Empty(y) ∧ In(x,z))
```

Solve the problem with forward planning in the state space with greedy best-first with elimination of repeated states (as heuristics, use the number of unsatisfied goal literals; break ties using a FIFO strategy). Report the search tree, the repeated nodes that are not expanded, and the solution found.

Answer of exercise 7.3


1
MSE(2,1,B)
MSE(2,1,C)
5
H=2
2
H=1
MES(2,B,1)
MEE(1,A,C)
MEE(2,B,C)
MES(2,C,1)
MEE(1,A,B)
MEE(2,C,B)
H=1
6
H=1
H=1
H=2
3
H=1
H=2
MES(2,B,1)
MEE(1,C,A)
MEE(2,B,A)
MES(2,C,1)
MEE(1,B,A)
MEE(2,C,A)
H=0
H=2
H=1
4
H=1
H=1
H=2
MSE(2,1,A)
MSE(2,1,C)
H=2
H=1