

Started on	Monday, 12 October 2020, 12:52 PM
State	Finished
Completed on	Friday, 16 October 2020, 5:06 PM
Time taken	4 days 4 hours
Marks	5.00/5.00
Grade	100.00 out of 100.00

Information

Introduction

This quiz starts with a number questions that use *explicit* game trees. Towards the end of the quiz you will see an example that requires *implicit* game trees. Explicit trees are useful for education purposes. However, in practice, game trees are usually very big and would take a lot of memory (and time) if they were to be stored in the memory. It is also inefficient to first generate a tree and then search it. The search and generation can be done at the same time. The search and generation are done according to the rules of the game without using much memory (linear in branching factor and the average number of moves to the end of the game). The time however still grows exponentially with respect to the depth of the tree.

Information

Representation of explicit game trees

An explicit game tree is a tree that is already constructed and resides in the memory (as opposed to being constructed on the fly). We use the following recursive representation for explicit game trees. A game tree is either

- a number which represents the utility (payoff) of a terminal (end-game) state; or
- a list of one or more game trees.

Examples

The root of the following game tree has three children. The first child is a leaf node with a utility of 1. The second child has two children (leaf nodes with utilities 2 and 3). The third child of the root has a single child which has a single child (a leaf node with a utility of 4).

```
game_tree = [1, [2, 3], [[4]]]
```

The following trees are all different. The first one is game tree that is a single leaf node. The second one has a root with one child which is terminal. The third one has a root which has one child which has one terminal child.

```
game_tree1 = 7
game_tree2 = [7]
game_tree3 =[[7]]
```

Question **1**
Correct
Mark 1.00 out of 1.00

Write **two** functions `max_value(tree)` and `min_value(tree)` that given a game tree, return the utility of the root of the tree when the root is a max node or min node correspondingly. Process the children of a node from left (lower index) to right (higher index).

For example:

Test	Result
<pre>from student_answer import min_value, max_value game_tree = 3 print("Root utility for minimiser:", min_value(game_tree)) print("Root utility for maximiser:", max_value(game_tree))</pre>	Root utility for minimiser: 3 Root utility for maximiser: 3
<pre>from student_answer import min_value, max_value game_tree = [1, 2, 3] print("Root utility for minimiser:", min_value(game_tree)) print("Root utility for maximiser:", max_value(game_tree))</pre>	Root utility for minimiser: 1 Root utility for maximiser: 3
<pre>from student_answer import min_value, max_value game_tree = [1, 2, [3]] print(min_value(game_tree)) print(max_value(game_tree))</pre>	1 3
<pre>from student_answer import min_value, max_value game_tree = [[1, 2], [3]] print(min_value(game_tree)) print(max_value(game_tree))</pre>	2 3

Answer: (penalty regime: 0, 15, ... %)

```
1 def min_value(tree):
2     if isinstance(tree, int):
3         return tree
4     else:
5
6         result = float("inf")
7         for node in tree:
8             current = max_value(node)
9             if current < result:
10                 result = current
11
12
13     return result
14
15
16 def max_value(tree):
17     if isinstance(tree, int):
18         return tree
19
20     else:
21
22         result = float("-inf")
23         for node in tree:
24             current = min_value(node)
25             if current > result:
26                 result = current
27
28
29     return result
```

	Test	Expected	Got	
✓	<pre>from student_answer import min_value, max_value game_tree = 3 print("Root utility for minimiser:", min_value(game_tree)) print("Root utility for maximiser:", max_value(game_tree))</pre>	Root utility for minimiser: 3 Root utility for maximiser: 3	Root utility for minimiser: 3 Root utility for maximiser: 3	✓
✓	<pre>from student_answer import min_value, max_value game_tree = [1, 2, 3] print("Root utility for minimiser:", min_value(game_tree)) print("Root utility for maximiser:", max_value(game_tree))</pre>	Root utility for minimiser: 1 Root utility for maximiser: 3	Root utility for minimiser: 1 Root utility for maximiser: 3	✓
✓	<pre>from student_answer import min_value, max_value game_tree = [1, 2, [3]] print(min_value(game_tree)) print(max_value(game_tree))</pre>	1 3	1 3	✓
✓	<pre>from student_answer import min_value, max_value game_tree = [[1, 2], [3]] print(min_value(game_tree)) print(max_value(game_tree))</pre>	2 3	2 3	✓

Passed all tests! ✓

Correct

Marks for this submission: 1.00/1.00.

Question **2**

Correct

Mark 1.00 out of 1.00

Write **two** functions `max_action_value(game_tree)` and `min_action_value(game_tree)` that given a game tree, return a pair where first element is the best action and the second element is the utility of the root of the tree when the root is a max node or min node correspondingly. For a leaf node the action is `None`; for an internal node, the action is the index of the subtree corresponding to the best action. Process the children of a node from left (lower index) to right (higher index). If there is a tie, return the left-most optimal action.

For example:

Test	Result
<pre>from student_answer import min_action_value, max_action_value game_tree = [2, [-3, 1], 4, 1] action, value = min_action_value(game_tree) print("Best action if playing min:", action) print("Best guaranteed utility:", value) print() action, value = max_action_value(game_tree) print("Best action if playing max:", action) print("Best guaranteed utility:", value)</pre>	<pre>Best action if playing min: 1 Best guaranteed utility: 1 Best action if playing max: 2 Best guaranteed utility: 4</pre>
<pre>from student_answer import min_action_value, max_action_value game_tree = 3 action, value = min_action_value(game_tree) print("Best action if playing min:", action) print("Best guaranteed utility:", value) print() action, value = max_action_value(game_tree) print("Best action if playing max:", action) print("Best guaranteed utility:", value)</pre>	<pre>Best action if playing min: None Best guaranteed utility: 3 Best action if playing max: None Best guaranteed utility: 3</pre>
<pre>from student_answer import min_action_value, max_action_value game_tree = [1, 2, [3]] action, value = min_action_value(game_tree) print("Best action if playing min:", action) print("Best guaranteed utility:", value) print() action, value = max_action_value(game_tree) print("Best action if playing max:", action) print("Best guaranteed utility:", value)</pre>	<pre>Best action if playing min: 0 Best guaranteed utility: 1 Best action if playing max: 2 Best guaranteed utility: 3</pre>

Answer: (penalty regime: 0, 15, ... %)

```
1 def min_value(tree):
2     if isinstance(tree, int):
3         return tree
4     else:
5         result = float("inf")
6         for node in tree:
7             current = max_value(node)
8             if current < result:
9                 result = current
10        return result
11
12
13
14
15
16 def max_value(tree):
17     if isinstance(tree, int):
18         return tree
19     else:
20         result = float("-inf")
21         for node in tree:
22             current = min_value(node)
23             if current > result:
24                 result = current
25        return result
26
27
28 def max_action_value(game_tree):
29     if isinstance(game_tree, int):
30         return None, game_tree
31
32     result = float("-inf")
33     for node in game tree:
```

```

34     previous_v = result
35     result = max(result, min_value(node))
36
37     if result > previous_v:
38         best_action = game_tree.index(node)
39     return best_action, result
40
41
42
43 def min_action_value(game_tree):
44     if isinstance(game_tree, int):
45         return None, game_tree
46     result = float("inf")
47     for node in game_tree:
48         previous_v = result
49         result = min(result, max_value(node))
50
51     if result < previous_v:
52         best_action = game_tree.index(node)
53
54
55     return best_action, result

```

	Test	Expected	Got	
✓	<pre> from student_answer import min_action_value, max_action_value game_tree = [2, [-3, 1], 4, 1] action, value = min_action_value(game_tree) print("Best action if playing min:", action) print("Best guaranteed utility:", value) print() action, value = max_action_value(game_tree) print("Best action if playing max:", action) print("Best guaranteed utility:", value) </pre>	<p>Best action if playing min: 1 Best guaranteed utility: 1</p> <p>Best action if playing max: 2 Best guaranteed utility: 4</p>	<p>Best action if playing min: 1 Best guaranteed utility: 1</p> <p>Best action if playing max: 2 Best guaranteed utility: 4</p>	✓
✓	<pre> from student_answer import min_action_value, max_action_value game_tree = 3 action, value = min_action_value(game_tree) print("Best action if playing min:", action) print("Best guaranteed utility:", value) print() action, value = max_action_value(game_tree) print("Best action if playing max:", action) print("Best guaranteed utility:", value) </pre>	<p>Best action if playing min: None Best guaranteed utility: 3</p> <p>Best action if playing max: None Best guaranteed utility: 3</p>	<p>Best action if playing min: None Best guaranteed utility: 3</p> <p>Best action if playing max: None Best guaranteed utility: 3</p>	✓

	Test	Expected	Got	
✓	<pre>from student_answer import min_action_value, max_action_value game_tree = [1, 2, [3]] action, value = min_action_value(game_tree) print("Best action if playing min:", action) print("Best guaranteed utility:", value) print() action, value = max_action_value(game_tree) print("Best action if playing max:", action) print("Best guaranteed utility:", value)</pre>	<pre>Best action if playing min: 0 Best guaranteed utility: 1 Best action if playing max: 2 Best guaranteed utility: 3</pre>	<pre>Best action if playing min: 0 Best guaranteed utility: 1 Best action if playing max: 2 Best guaranteed utility: 3</pre>	✓

Passed all tests! ✓

Correct
Marks for this submission: 1.00/1.00.

Information

α-β pruning

In an alpha-beta pruning question you are given an explicit game tree (where either max is playing at root or min) and then asked to prune the tree. You need to provide two variables: `pruned_tree` which is the pruned game tree and `pruning_events` which is a list of pairs of alpha and beta when a pruning event was triggered. Please keep the following points in mind:

- It might be easier to answer this question by drawing the tree.
- Process the children of a node from left to right.
- A pruning event is triggered when alpha becomes greater than or equal to beta.
- Children of a node are processed from left to right.
- A pruning event might be triggered without any branches getting pruned. This happens when the event is triggered after seeing the last child. Such events must also be included in the list.
- For some problems the pruned tree is the same as the original tree (i.e. no pruning) even if there have been some pruning events.
- You do not need to provide any function in your answer, however, if you wish, you can write a program to compute the requested variables automatically.

Question **3**

Correct

Mark 1.00 out of 1.00

Consider the following explicit game tree.

[2, [-1, 5], [1, 3], 4]

Assuming that the player at the root of the tree is Max, prune the tree (if necessary). Provide two variables: `pruned_tree` which is the pruned game tree and `pruning_events` which is a list of pairs of alpha and beta, one for each time a pruning event was triggered.

.
For example:

Test	Result
<code>import student_answer</code>	OK
<code>check_it_is_a_gametree(student_answer.pruned_tree)</code> <code>check_it_is_a_pruning_events_list(student_answer.pruning_events)</code>	OK

Answer: (penalty regime: 0, 15, ... %)

Reset answer

```
1 from math import inf
2 pruned_tree = [2, [-1], [1], 4]
3
4
5 pruning_events = [
6     # (alpha, beta),
7     (2, -1),
8     (2, 1),
9 ]
```

	Test	Expected	Got	
✓	<code>import student_answer</code> <code>check_it_is_a_gametree(student_answer.pruned_tree)</code> <code>check_it_is_a_pruning_events_list(student_answer.pruning_events)</code>	OK OK	OK OK	✓
✓	<code>import student_answer</code> <code>check_pruned_tree(student_answer.pruned_tree)</code>	OK	OK	✓
✓	<code>import student_answer</code> <code>check_pruning_events(student_answer.pruning_events)</code>	OK	OK	✓

Passed all tests! ✓

Correct

Marks for this submission: 1.00/1.00.

Question **4**

Correct

Mark 1.00 out of 1.00

Consider the following explicit game tree.

[0, [-2, 1], 5]

Assuming that the player at the root of the tree is Min, prune the tree (if necessary). Provide two variables: `pruned_tree` which is the pruned game tree and `pruning_events` which is a list of pairs of alpha and beta, one for each time a pruning event was triggered.

.
For example:

Test	Result
<code>import student_answer</code>	OK
<code>check_it_is_a_gametree(student_answer.pruned_tree)</code> <code>check_it_is_a_pruning_events_list(student_answer.pruning_events)</code>	OK

Answer: (penalty regime: 0, 15, ... %)

Reset answer

```
1 | from math import inf
2 |
3 | pruned_tree = [0, [-2, 1], 5]
4 |
5 |
6 | pruning_events = [
7 |     # (alpha, beta),
8 |     (1, 0),
9 | ]
```

	Test	Expected	Got	
✓	<code>import student_answer</code> <code>check_it_is_a_gametree(student_answer.pruned_tree)</code> <code>check_it_is_a_pruning_events_list(student_answer.pruning_events)</code>	OK OK	OK OK	✓
✓	<code>import student_answer</code> <code>check_pruned_tree(student_answer.pruned_tree)</code>	OK	OK	✓
✓	<code>import student_answer</code> <code>check_pruning_events(student_answer.pruning_events)</code>	OK	OK	✓

Passed all tests! ✓

Correct
Marks for this submission: 1.00/1.00.

Question **5**

Correct

Mark 1.00 out of 1.00

Consider the following explicit game tree.

[3, [[2, 1], [4, [7, -2]]], 0]

Assuming that the player at the root of the tree is Max, prune the tree (if necessary). Provide two variables: `pruned_tree` which is the pruned game tree and `pruning_events` which is a list of pairs of alpha and beta, one for each time a pruning event was triggered.

.

For example:

Test	Result
<code>import student_answer</code>	OK
<code>check_it_is_a_gametree(student_answer.pruned_tree)</code> <code>check_it_is_a_pruning_events_list(student_answer.pruning_events)</code>	OK

Answer: (penalty regime: 0, 15, ... %)

Reset answer

```
1 | from math import inf
2 |
3 | pruned_tree = [3, [[2, 1]], 0]
4 |
5 |
6 | pruning_events = [
7 |     # (alpha, beta),
8 |     (3, 2),
9 | ]
```

	Test	Expected	Got	
✓	<code>import student_answer</code> <code>check_it_is_a_gametree(student_answer.pruned_tree)</code> <code>check_it_is_a_pruning_events_list(student_answer.pruning_events)</code>	OK OK	OK OK	✓
✓	<code>import student_answer</code> <code>check_pruned_tree(student_answer.pruned_tree)</code>	OK	OK	✓
✓	<code>import student_answer</code> <code>check_pruning_events(student_answer.pruning_events)</code>	OK	OK	✓

Passed all tests! ✓

Correct
Marks for this submission: 1.00/1.00.

Information

Extra challenge

If you are interested in implementing a complete game-playing agent for a small game (noughts and crosses) see the additional material section. Take this additional challenge only if you are interested and have the time. It is not an official part of the course and does not carry any marks.

◀ 10. Machine learning with kNN and basic neural networks

Jump to...

A1. A* Routing ▶