k200183-lab-07

April 2, 2023

1 Task 1

[1]:
```python
class Node:
    def __init__(self, value=None):
        self.value = value
        self.children = []


def minimax(node, depth, maximizing_player=True):
    if depth == 0 or not node.children:
        return node.value

    if maximizing_player:
        value = -float('inf')
        for child in node.children:
            value = max(value, minimax(child, depth-1, False))
        return value
    else:
        value = float('inf')
        for child in node.children:
            value = min(value, minimax(child, depth-1, True))
        return value

# Sample tree
root = Node()
root.value = 'A'

n1 = Node('B')
n2 = Node('C')
root.children = [n1, n2]

n3 = Node('D')
n4 = Node('E')
n5 = Node('F')
n6 = Node('G')
n1.children = [n3, n4]
n2.children = [n5, n6]
```

```
n7 = Node(-1)
n8 = Node(4)
n9 = Node(2)
n10 = Node(6)
n3.children = [n7, n8]
n4.children = [n9, n10]

n11 = Node(-3)
n12 = Node(-5)
n13 = Node(0)
n14 = Node(7)
n5.children = [n11, n12]
n6.children = [n13, n14]

# Example usage
value = minimax(root, 3, True)
print(value)
```

4

2 Task 01 Graph 02

[2]:
```
class Node:
  def __init__(self, value=None):
    self.value = value
    self.children = []

def minimax_trees(node, depth, maximizing_player=True):
    if depth == 0 or not node.children:
        return node.value

    if maximizing_player:
        value = -float('inf')
        for child in node.children:
            value = max(value, minimax(child, depth-1, False))
        return value
    else:
        value = float('inf')
        for child in node.children:
            value = min(value, minimax(child, depth-1, True))
        return value

root1 = Node('A')
B = Node('B')
C = Node('C')
```

```
D = Node('D')
E = Node('E')
F = Node('F')
G = Node('G')
H = Node('H')
I = Node('I')
J = Node('J')
K = Node('K')
n1 = Node(4)
n2 = Node(3)
n3 = Node(6)
n4 = Node(2)
n5 = Node(2)
n6 = Node(1)
n7 = Node(9)
n8 = Node(5)
n9 = Node(3)
n10 = Node(1)
n11 = Node(5)
n12 = Node(4)
n13 = Node(7)
n14 = Node(8)

root1.children = [B, C, D]
B.children = [E, F]
C.children = [G, H, I]
D.children = [J, K]
E.children = [n1, n2]
F.children = [n3, n4]
G.children = [n5, n6]
H.children = [n7, n8]
I.children = [n9, n10]
J.children = [n11, n12]
K.children = [n13, n14]

value = minimax_trees(root1, 3, True)
print(value)
```

```
5
```

3 Task 02

[9]:
```
import math

class Node:
    def __init__(self, value=None):
        self.value = value
```

```
        self.children = []


def alpha_beta(node, depth, alpha, beta, maximizing_player=True):
    if depth == 0 or not node.children:
        return node.value


    if maximizing_player:
        value = -math.inf
        for child in node.children:
            value = max(value, alpha_beta(child, depth-1, alpha, beta, False))
            alpha = max(alpha, value)
            if beta <= alpha:
                break
        return value
    else:
        value = math.inf
        for child in node.children:
            value = min(value, alpha_beta(child, depth-1, alpha, beta, True))
            beta = min(beta, value)
            if beta <= alpha:
                break
        return value


# Sample tree
root = Node()
root.value = '6'


n1 = Node(3)
n2 = Node(6)
n3 = Node(5)
root.children = [n1, n2, n3]


n4 = Node(5)
n5 = Node(3)
n6 = Node(6)
n7 = Node(7)
n8 = Node(5)
n9 = Node(8)
n1.children = [n4, n5]
n2.children = [n6, n7]
n3.children = [n8, n9]
```

```
n10 = Node(5)
n11 = Node(4)
n12 = Node(3)
n13 = Node(6)
n14 = Node(6)
n15 = Node(7)
n16 = Node(5)
n17 = Node(8)
n18 = Node(6)
n4.children = [n10, n11]
n5.children = [n12]
n6.children = [n13, n14]
n7.children = [n15]
n8.children = [n16]
n9.children = [n17, n18]

n19 = Node(5)
n20 = Node(6)
n21 = Node(7)
n22 = Node(4)
n23 = Node(5)
n24 = Node(3)
n25 = Node(6)
n26 = Node(6)
n27 = Node(9)
n28 = Node(7)
n29 = Node(5)
n30 = Node(9)
n31 = Node(8)
n32 = Node(6)
n10.children = [n19, n20]
n11.children = [n21, n22, n23]
n12.children = [n24]
n13.children = [n25]
n14.children = [n26, n27]
n15.children = [n28]
n16.children = [n29]
n17.children = [n30, n31]
n18.children = [n32]


# Example usage
value = alpha_beta(root, 4, -math.inf, math.inf)
print(value)
```

6

4 Task 03

[11]:
```
from constraint import *

def n_queens(n):
    problem = Problem()

    columns = range(n)
    rows = range(n)

    # Define variables and domains
    problem.addVariables(columns, rows)

    # Define constraints
    for i in columns:
        for j in columns:
            if i < j:
                problem.addConstraint(lambda x, y, i=i, j=j: x != y and abs(x -␣
 ↪y) != abs(i - j), (i, j))

    # Get solutions
    solutions = problem.getSolutions()

    # Return solutions
    return solutions

# Example usage
solutions = n_queens(4)
print(solutions)
```

```
[{0: 2, 1: 0, 2: 3, 3: 1}, {0: 1, 1: 3, 2: 0, 3: 2}]
```

5 Task 04

[14]:
```
#Task 4

from constraint import *

problem = Problem()

# variables and possible values
departments = ['Department1', 'Department2', 'Department3', 'Department4']
offices = ['office1', 'office2', 'office3', 'office4']
problem.addVariables(departments, offices)

#constraints
def different_offices(dept1, dept2):
```

```
    return dept1 != dept2
for dept1 in departments:
    for dept2 in departments:
        if dept1 != dept2:
            problem.addConstraint(different_offices, [dept1, dept2])

def max_occupancy(office, max_capacity):
    def occupancy(*departments):
        return sum(dept == office for dept in departments) <= max_capacity
    return occupancy
problem.addConstraint(max_occupancy("office1", 1), departments)
problem.addConstraint(max_occupancy("office2", 2), departments)
problem.addConstraint(max_occupancy("office3", 1), departments)
problem.addConstraint(max_occupancy("office4", 2), departments)

solutions = problem.getSolutions()

for solution in solutions:
    print(solution)
```

```
{'Department1': 'office4', 'Department2': 'office3', 'Department3': 'office2',
'Department4': 'office1'}
{'Department1': 'office4', 'Department2': 'office3', 'Department3': 'office1',
'Department4': 'office2'}
{'Department1': 'office4', 'Department2': 'office2', 'Department3': 'office3',
'Department4': 'office1'}
{'Department1': 'office4', 'Department2': 'office2', 'Department3': 'office1',
'Department4': 'office3'}
{'Department1': 'office4', 'Department2': 'office1', 'Department3': 'office2',
'Department4': 'office3'}
{'Department1': 'office4', 'Department2': 'office1', 'Department3': 'office3',
'Department4': 'office2'}
{'Department1': 'office3', 'Department2': 'office4', 'Department3': 'office1',
'Department4': 'office2'}
{'Department1': 'office3', 'Department2': 'office4', 'Department3': 'office2',
'Department4': 'office1'}
{'Department1': 'office3', 'Department2': 'office2', 'Department3': 'office4',
'Department4': 'office1'}
{'Department1': 'office3', 'Department2': 'office2', 'Department3': 'office1',
'Department4': 'office4'}
{'Department1': 'office3', 'Department2': 'office1', 'Department3': 'office2',
'Department4': 'office4'}
{'Department1': 'office3', 'Department2': 'office1', 'Department3': 'office4',
'Department4': 'office2'}
{'Department1': 'office2', 'Department2': 'office3', 'Department3': 'office1',
'Department4': 'office4'}
{'Department1': 'office2', 'Department2': 'office3', 'Department3': 'office4',
```

```
'Department4': 'office1'}
{'Department1': 'office2', 'Department2': 'office4', 'Department3': 'office3',
'Department4': 'office1'}
{'Department1': 'office2', 'Department2': 'office4', 'Department3': 'office1',
'Department4': 'office3'}
{'Department1': 'office2', 'Department2': 'office1', 'Department3': 'office4',
'Department4': 'office3'}
{'Department1': 'office2', 'Department2': 'office1', 'Department3': 'office3',
'Department4': 'office4'}
{'Department1': 'office1', 'Department2': 'office2', 'Department3': 'office4',
'Department4': 'office3'}
{'Department1': 'office1', 'Department2': 'office2', 'Department3': 'office3',
'Department4': 'office4'}
{'Department1': 'office1', 'Department2': 'office3', 'Department3': 'office2',
'Department4': 'office4'}
{'Department1': 'office1', 'Department2': 'office3', 'Department3': 'office4',
'Department4': 'office2'}
{'Department1': 'office1', 'Department2': 'office4', 'Department3': 'office3',
'Department4': 'office2'}
{'Department1': 'office1', 'Department2': 'office4', 'Department3': 'office2',
'Department4': 'office3'}
```

6 Task 05

[15]:
```python
from constraint import *

problem = Problem()

# variables and possible values
departments = ["A", "B", "C", "D"]
offices = ["1", "2", "3", "4"]
problem.addVariables(departments, offices)

#constraints
def different_offices(dept1, dept2):
    return dept1 != dept2
for dept1 in departments:
    for dept2 in departments:
        if dept1 != dept2:
            problem.addConstraint(different_offices, [dept1, dept2])

def max_occupancy(office, max_capacity):
    def occupancy(*departments):
        return sum(dept == office for dept in departments) <= max_capacity
    return occupancy
problem.addConstraint(max_occupancy("1", 1), departments)
problem.addConstraint(max_occupancy("2", 2), departments)
```

```
problem.addConstraint(max_occupancy("3", 1), departments)
problem.addConstraint(max_occupancy("4", 2), departments)

solutions = problem.getSolutions()

for solution in solutions:
    print(solution)
```

```
{'A': '4', 'B': '3', 'C': '2', 'D': '1'}
{'A': '4', 'B': '3', 'C': '1', 'D': '2'}
{'A': '4', 'B': '2', 'C': '3', 'D': '1'}
{'A': '4', 'B': '2', 'C': '1', 'D': '3'}
{'A': '4', 'B': '1', 'C': '2', 'D': '3'}
{'A': '4', 'B': '1', 'C': '3', 'D': '2'}
{'A': '3', 'B': '4', 'C': '1', 'D': '2'}
{'A': '3', 'B': '4', 'C': '2', 'D': '1'}
{'A': '3', 'B': '2', 'C': '4', 'D': '1'}
{'A': '3', 'B': '2', 'C': '1', 'D': '4'}
{'A': '3', 'B': '1', 'C': '2', 'D': '4'}
{'A': '3', 'B': '1', 'C': '4', 'D': '2'}
{'A': '2', 'B': '3', 'C': '1', 'D': '4'}
{'A': '2', 'B': '3', 'C': '4', 'D': '1'}
{'A': '2', 'B': '4', 'C': '3', 'D': '1'}
{'A': '2', 'B': '4', 'C': '1', 'D': '3'}
{'A': '2', 'B': '1', 'C': '4', 'D': '3'}
{'A': '2', 'B': '1', 'C': '3', 'D': '4'}
{'A': '1', 'B': '2', 'C': '4', 'D': '3'}
{'A': '1', 'B': '2', 'C': '3', 'D': '4'}
{'A': '1', 'B': '3', 'C': '2', 'D': '4'}
{'A': '1', 'B': '3', 'C': '4', 'D': '2'}
{'A': '1', 'B': '4', 'C': '3', 'D': '2'}
{'A': '1', 'B': '4', 'C': '2', 'D': '3'}
```

[ ]: