Tower of Hanoi

02.03.2020

Lokesh Nirania 170010009

Sudarshan Das 170010010

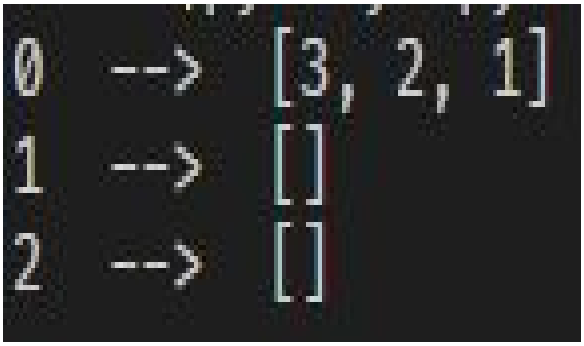
Brief Description about the Domain

Tower of Hanoi is a mathematical puzzle where we have three rods and n disks. The objective of the puzzle is to move the entire stack to another rod, obeying the following simple rules:

1. Only one disk can be moved at a time.
2. Each move consists of taking the upper disk from one of the stacks and placing it on top of another stack i.e. a disk can only be moved if it is the uppermost disk on a stack.
3. No disk may be placed on top of a smaller disk.

START STATE

An animated solution of the **Tower of Hanoi** puzzle for *T*(4, 3)



An animated solution of the **Tower of Hanoi** puzzle for *T*(4, 3)

```
0  -->  []
1  -->  []
2  -->  [3, 2, 1]
```

MoveGen(State)

// Try all combinations to move from disk i to disk j (i,j = 1 to 3)

// Accept only that state which follows the rule i.e., no bigger disk on top

For i = 1 to 3

    For j = 1 to 3

        If i != j // no point in swapping disks from same pole

            Swap from pole i to j

            If state valid then accept

            Else reject

GoalTest(State)

If State == goal

    Return true

Return false

Heuristic Function

I. Heuristic 1

```
def heuristic_1(node):
      value = discs * discs
            for i in range(0,3):
                  for j in node.poles[i]:
                        value = value + ((i + 1) * j)
       return value
```

// This way the goal state will have highest heuristic and start state have lowest heuristic value

II. Heuristic 2

```
def heuristic_2(node):
      value = discs * discs
            for i in range(0,3):
                  for j in range(node.poles[i]):
                        if state formed is correct according to
                        Tower of Hanoi Domain:
                              value = value - ((i + 1) *
                              node.poles[i][j] * (j + 1))
                        else:
                              value = value + ((i + 1) *
                              node.poles[i][j] * (j + 1))
            for discs in 1st pole:
                  if discs are not touched:
                        value = value + ((i + 1) * node.poles[i][j]
                                                 * (j + 1))
       return value
```

// This way the goal state will have zero heuristic and other corrects state will have lowest heuristic value to move forward

III. Heuristic 3

```
def heuristic_3(node):
      value = discs * discs
            for i in range(0,3):
                  for j in range(node.poles[i]):
                        if state formed is correct according to
                        Tower of Hanoi Domain:
                              value = value + (j + 1)
                        else:
                              value = value - (j + 1)
            for discs in 1st pole:
                  if discs are not touched:
                        value = value - (j + 1)
       return value
```

// This way the goal state will have highest heuristic and other corrects state will have higher heuristic value from start state to move forward

A* Algorithm Analysis

General A* Algorithm

```
while(OPEN is not EMPTY):
    n = Head(OPEN)
    If GoalTest(n):
        Reconstruct path
        break
    neighbours = movengen(n)
    For m in neighbours:
        If m in OPEN:
            If n.g + k(n, m) < m.g:
                m.parent = n
                m.g = n.g + k(n, m)
                m.f = m.g + m.h
        If m in CLOSED:
            If n.g + k(n, m) < m.g:
                m.parent = n
                m.g = n.g + k(n, m)
                m.f = m.g + m.h
                Propagation_Improvement(m)
        If m not in OPEN and m not in CLOSED:
            OPEN.append(m)
            m.parent = n
            m.g = n.g + k(n, m)
            m.f = m.g + m.h
```

For h* heuristic which also satisfy monotone property:

state .h = heuristic_1(goal) - heuristic_1(state)

// Here f* = g* + h*. Since h* is the optimal heuristic, f* gives optimal values thus giving optimal solution at the end.

// f* is admissible as g* >= g* and h* <= h*

// Therefore path formed is the optimal path with optimal costs.

For h heuristic which overestimate cost to goal node:

state .h = heuristic_2(goal) - heuristic_2(state) + 2 * (-1 * (heuristic_2(goal) - heuristic_2(start)))

// Here f = g + h.

// g >= g* and h !<= h*. Thus f is not admissible

// Since h overestimate cost to goal node, f will have higher values than the optimal value and will tend to reject the path thus found.

// Therefore the path formed will be longer with higher costs.

For h heuristic which underestimate cost to goal node:

state .h = heuristic_3(goal) - heuristic_3(state)

// Here f = g + h.

// g >= g* and h <= h*. Thus f is not admissible

// Since h underestimate cost to the goal node, f will have lower values than the optimal value and will tend to move towards the path thus found as the cost estimated is less than the optimal.

// Therefore the path formed will be optimal with optimal costs.

Observations:

Discs = 3

Heuristic	Start State Cost	Path Length
Normal/monotone	12	8
Overestimate	30	12
Underestimate	6	8

```
3
Normal Funciton
12 0
goal reached
path length =  8
path     ->  [0, 3, 4, 6, 15, 16, 19, 1]
h-values ->  [12, 10, 8, 9, 3, 4, 2, 0]
g-values ->  [0, 1, 2, 3, 4, 5, 6, 7]
f-values ->  [12, 11, 10, 12, 7, 9, 8, 7]
--------------------------------
Over estimate
30 0
goal reached
path length =  12
path     ->  [0, 2, 4, 6, 10, 12, 13, 14, 17, 21, 24, 1]
h-values ->  [30, 28, 22, 30, 24, 14, 18, 24, 39, 35, 49, 0]
g-values ->  [0, 1, 5, 6, 9, 8, 4, 3, 6, 7, 11, 12]
f-values ->  [30, 29, 27, 36, 33, 22, 22, 27, 45, 42, 60, 12]
--------------------------------
Under estimate
6 0
goal reached
path length =  8
path     ->  [0, 3, 4, 6, 7, 8, 11, 1]
h-values ->  [6, 5, 4, 3, 2, 3, 2, 0]
g-values ->  [0, 1, 2, 3, 4, 5, 6, 7]
f-values ->  [6, 6, 6, 6, 6, 8, 8, 7]
--------------------------------
```

Discs = 4

Heuristic	Start State Cost	Path Length
Normal/monotone	20	16
Overestimate	60	26
Underestimate	10	16

```
4
Normal Funciton
20 0
goal reached
path length =  16
path     ->  [0, 2, 7, 9, 10, 11, 14, 34, 39, 41, 42, 48, 51, 52, 57, 1]
h-values ->  [20, 19, 15, 14, 11, 13, 15, 14, 6, 5, 7, 9, 6, 5, 1, 0]
g-values ->  [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
f-values ->  [20, 20, 17, 17, 15, 18, 21, 21, 14, 14, 17, 20, 18, 18, 15, 15]
---------------------------------
Over estimate
60 0
goal reached
path length =  26
path     ->  [0, 3, 4, 6, 9, 10, 17, 23, 28, 29, 31, 32, 37, 39, 41, 42, 45, 55, 58, 59, 61, 62, 65, 75, 79, 1]
h-values ->  [60, 57, 53, 60, 51, 49, 65, 72, 64, 54, 34, 41, 47, 38, 50, 56, 76, 75, 59, 55, 63, 69, 90, 85, 107, 0]
g-values ->  [0, 2, 4, 3, 9, 8, 10, 12, 16, 14, 12, 13, 7, 8, 6, 4, 8, 9, 13, 12, 10, 11, 17, 16, 18, 20]
f-values ->  [60, 59, 57, 63, 60, 57, 75, 84, 80, 68, 46, 54, 54, 46, 56, 60, 84, 84, 72, 67, 73, 80, 107, 101, 125, 20]
---------------------------------
Under estimate
10 0
goal reached
path length =  16
path     ->  [0, 2, 4, 6, 7, 8, 10, 11, 13, 15, 17, 25, 35, 36, 39, 1]
h-values ->  [10, 9, 8, 7, 6, 6, 5, 4, 3, 4, 5, 5, 4, 5, 3, 0]
g-values ->  [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
f-values ->  [10, 10, 10, 10, 10, 11, 11, 11, 11, 13, 15, 16, 16, 18, 17, 15]
---------------------------------
```

Discs = 5

Heuristic	Start State Cost	Path Length
Normal/monotone	30	32
Overestimate	105	68
Underestimate	15	32

```
5
Normal Funciton
30 0
goal reached
path length =  32
path     ->  [0, 3, 4, 6, 15, 16, 19, 21, 22, 24, 25, 33, 38, 92, 93, 95, 104, 105, 108, 110, 111, 115, 127, 141, 147, 149, 150, 152, 161, 162, 165, 1]
h-values ->  [30, 28, 26, 27, 21, 22, 20, 18, 14, 15, 19, 20, 23, 21, 19, 20, 18, 11, 9, 7, 10, 11, 15, 16, 12, 10, 8, 9, 3, 4, 2, 0]
g-values ->  [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31]
f-values ->  [30, 29, 28, 30, 25, 27, 26, 25, 22, 24, 29, 31, 35, 34, 33, 35, 26, 28, 27, 26, 30, 32, 37, 39, 36, 35, 34, 36, 31, 33, 32, 31]
--------------------------------
Over estimate
105 0
goal reached
path length =  68
path     ->  [0, 2, 4, 6, 10, 11, 18, 19, 25, 26, 28, 29, 31, 32, 43, 59, 60, 61, 63, 80, 83, 85, 86, 87, 90, 91, 93, 94, 96, 99, 109, 110, 112, 114, 119, 121, 123, 124, 132, 164, 165, 167, 170, 172, 173, 175, 176, 185, 191, 193, 195, 19
6, 202, 204, 205, 207, 209, 219, 221, 223, 225, 227, 228, 230, 232, 237, 241, 1]
h-values ->  [105, 103, 97, 105, 99, 96, 110, 113, 101, 93, 73, 71, 87, 94, 118, 106, 106, 113, 127, 141, 131, 115, 95, 101, 71, 63, 53, 60, 84, 91, 99, 93, 75, 79, 97, 87, 99, 107, 132, 122, 120, 130, 106, 93, 85, 92, 114, 128, 140, 129
, 103, 108, 129, 117, 129, 138, 154, 149, 127, 141, 132, 116, 132, 138, 171, 167, 193, 0]
g-values ->  [0, 1, 5, 6, 9, 7, 5, 6, 14, 13, 15, 16, 12, 11, 14, 16, 18, 16, 18, 20, 25, 24, 20, 19, 16, 17, 21, 20, 18, 19, 11, 10, 12, 14, 11, 10, 6, 5, 10, 12, 14, 13, 19, 20, 18, 17, 21, 22, 26, 24, 22, 23, 17, 18, 16, 14, 18, 19, 2
3, 24, 27, 26, 22, 21, 24, 25, 29, 30]
f-values ->  [105, 104, 102, 111, 108, 103, 115, 119, 115, 106, 88, 87, 99, 105, 132, 122, 124, 129, 145, 161, 156, 139, 115, 120, 87, 80, 74, 80, 102, 110, 110, 103, 87, 93, 108, 97, 105, 112, 142, 134, 134, 143, 125, 113, 103, 109, 135
, 150, 166, 153, 125, 131, 146, 135, 145, 152, 172, 168, 150, 165, 159, 142, 154, 159, 195, 192, 222, 30]
--------------------------------
Under estimate
15 0
goal reached
path length =  32
path     ->  [0, 3, 4, 6, 7, 8, 10, 12, 13, 15, 17, 21, 25, 31, 38, 41, 42, 43, 52, 60, 61, 71, 78, 93, 96, 98, 109, 115, 127, 128, 140, 1]
h-values ->  [15, 14, 13, 12, 11, 10, 9, 9, 8, 9, 9, 8, 7, 9, 8, 5, 4, 7, 8, 6, 7, 8, 8, 7, 6, 6, 7, 8, 6, 7, 4, 0]
g-values ->  [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31]
f-values ->  [15, 15, 15, 15, 15, 15, 15, 16, 16, 18, 19, 19, 19, 22, 22, 20, 20, 24, 26, 25, 27, 29, 30, 30, 30, 31, 33, 35, 34, 36, 34, 31]
--------------------------------
```