

## CS4750/7750 HW #2 Report

Prepared by Jonah Marz, Ke Li, Xiaoyi Li

The programming language and hardware:

Python3.7/Inter i5-5200U 16G

Cost function description:

We used a recursive version of the search pseudocode given to us in class in order to implement these algorithms. We store a numeric value for each node in order for the Graph searches to be able to remember closed nodes. Each iteration of the algorithm loops through all possible actions and adds viable actions to the fringe up to the depth limit. Each time a recursive call is made the fringe is updated and sorted based on the criteria of the search algorithm (cost for uniform cost and LIFO for depth first search).

Our costs were calculated with each individual step so that each iteration would be able to make the best decisions going forward. The program was implemented in python on our personal laptops. There were some issues with hardware in that the scripts could not run past a depth of 6. There was simply not enough computational power on either of our machines to handle a depth of 10, so our solutions are based on a depth of 6. When calculated by hand smaller depths produced the optimal solutions to the problem for the algorithms. The algorithms are implemented as similarly as possible in order to reduce confusion in reading. The only real difference is that the graph searches maintain a “closed” list of nodes that have been visited so it does not visit them again, and graph searches performed differently as a result of that.

The results of the test cases are printed below just modified to handle a depth of 6.

a) uniform cost tree search

### Test Case #1

```
In [13]: #Start CPU Time  
start = time.clock()
```

```
In [14]: Result = UCS(ds1,vcl1,6)
```

```
The Frontier is [[11, 0, 3, 2], [12, -0.2, 3, 2], [13, -1, 3, 1], [14, -1.1, 3, 3], [15, -1.2, 2, 2], [16, -1.3, 4, 2]]  
The Frontier is [[211, 0, 3, 2], [12, -0.2, 3, 2], [212, -0.2, 3, 2], [13, -1, 3, 1], [213, -1, 3, 1], [14, -1.1, 3, 3], [214, -1.1, 3, 3], [15, -1.2, 2, 2], [215, -1.2, 2, 2], [16, -1.3, 4, 2], [216, -1.3, 4, 2]]  
The Frontier is [[3111, 0, 3, 2], [12, -0.2, 3, 2], [212, -0.2, 3, 2], [3112, -0.2, 3, 2], [13, -1, 3, 1], [213, -1, 3, 1], [3113, -1, 3, 1], [14, -1.1, 3, 3], [214, -1.1, 3, 3], [3114, -1.1, 3, 3], [15, -1.2, 2, 2], [215, -1.2, 2, 2], [3115, -1.2, 2, 2], [16, -1.3, 4, 2], [216, -1.3, 4, 2], [3116, -1.3, 4, 2]]  
The Frontier is [[41111, 0, 3, 2], [12, -0.2, 3, 2], [212, -0.2, 3, 2], [3112, -0.2, 3, 2], [41112, -0.2, 3, 2], [13, -1, 3, 1], [213, -1, 3, 1], [3113, -1, 3, 1], [41113, -1, 3, 1], [14, -1.1, 3, 3], [214, -1.1, 3, 3], [3114, -1.1, 3, 3], [41114, -1.1, 3, 3], [15, -1.2, 2, 2], [215, -1.2, 2, 2], [3115, -1.2, 2, 2], [41115, -1.2, 2, 2], [16, -1.3, 4, 2], [216, -1.3, 4, 2], [3116, -1.3, 4, 2], [41116, -1.3, 4, 2]]  
The Frontier is [[511111, 0, 3, 2], [12, -0.2, 3, 2], [212, -0.2, 3, 2], [3112, -0.2, 3, 2], [41112, -0.2, 3, 2], [511112, -0.2, 3, 2], [13, -1, 3, 1], [213, -1, 3, 1], [3113, -1, 3, 1], [41113, -1, 3, 1], [511113, -1, 3, 1], [14, -1.1, 3, 3], [214, -1.1, 3, 3], [3114, -1.1, 3, 3], [41114, -1.1, 3, 3], [511114, -1.1, 3, 3], [15, -1.2, 2, 2], [215, -1.2, 2, 2], [3115, -1.2, 2, 2], [41115, -1.2, 2, 2], [511115, -1.2, 2, 2], [16, -1.3, 4, 2], [216, -1.3, 4, 2], [3116, -1.3, 4, 2], [41116, -1.3, 4, 2], [511116, -1.3, 4, 2]]  
The Frontier is [[6111111, 0, 3, 2], [12, -0.2, 3, 2], [212, -0.2, 3, 2], [3112, -0.2, 3, 2], [41112, -0.2, 3, 2], [511112, -0.2, 3, 2], [6111112, -0.2, 3, 2], [13, -1, 3, 1], [213, -1, 3, 1], [3113, -1, 3, 1], [41113, -1, 3, 1], [511113, -1, 3, 1], [6111113, -1, 3, 1], [14, -1.1, 3, 3], [214, -1.1, 3, 3], [3114, -1.1, 3, 3], [41114, -1.1, 3, 3], [511114, -1.1, 3, 3], [6111114, -1.1, 3, 3], [15, -1.2, 2, 2], [215, -1.2, 2, 2], [3115, -1.2, 2, 2], [41115, -1.2, 2, 2], [511115, -1.2, 2, 2], [6111115, -1.2, 2, 2], [16, -1.3, 4, 2], [216, -1.3, 4, 2], [3116, -1.3, 4, 2], [41116, -1.3, 4, 2], [511116, -1.3, 4, 2], [6111116, -1.3, 4, 2]]
```

```
In [15]: #End CPU Time  
end = time.clock()  
CPU_Time = end - start
```

```
In [16]: cost = []  
for i in Result:  
    cost.append(i)  
  
cost = Sort(cost)
```

```
In [17]: print("b.The best solution " + str(cost[0][0])  
        + " And the point is " + str(cost[0][1]))  
print("c.The number of nodes (6 depths) " + str(len(cost)))  
print("d.CPU time is " + str(CPU_Time))  
  
b.The best solution 6352521 And the point is 4.1999999999999999  
c.The number of nodes (6 depths) 34699  
d.CPU time is 34.920571
```

For case#1, if we go through "right->down->suck->right->down->suck->down->suck->Noop->Noop", the score is -  $1.1-1.3+4-0.2-1.1-1.3+4-0.2-1.3+4-0.2+0+0=5.3$ .

For this algorithm,  $4.1999 < 5.3$ , so the best solution found is not the optimal solution.

## Test Case #2

```
In [13]: #Start CPU Time  
start = time.clock()
```

```
In [14]: Result = UCTS(ds2,vcl2,6)
```

```
IOPub data rate exceeded.  
The notebook server will temporarily stop sending output  
to the client in order to avoid crashing it.  
To change this limit, set the config variable  
`--NotebookApp.iopub_data_rate_limit`.  
  
Current values:  
NotebookApp.iopub_data_rate_limit=1000000.0 (bytes/sec)  
NotebookApp.rate_limit_window=3.0 (secs)
```

```
The Frontier is [[43151, -2.2, 2, 2], [513151, -2.2, 2, 2], [531151, -2.2, 2, 2], [3441, -2.2, 3, 5], [41441, -2.2, 3, 5],  
[511441, -2.2, 3, 5], [44141, -2.2, 3, 5], [514141, -2.2, 3, 5], [541141, -2.2, 3, 5], [3531, -2.2, 2, 2], [41531, -2.2, 2,  
2], [511531, -2.2, 2, 2], [42331, -2.2, 3, 1], [512331, -2.2, 3, 1], [521331, -2.2, 3, 1], [43231, -2.2, 3, 1], [513231, -2.  
2, 3, 1], [531231, -2.2, 3, 1], [45131, -2.2, 2, 2], [515131, -2.2, 2, 2], [523131, -2.2, 3, 1], [532131, -2.2, 3, 1], [5511  
31, -2.2, 2, 2], [43321, -2.2, 3, 1], [513321, -2.2, 3, 1], [531321, -2.2, 3, 1], [533121, -2.2, 3, 1], [43511, -2.2, 2, 2],  
[513511, -2.2, 2, 2], [6113511, -2.2, 2, 2], [236, -2.3, 4, 2], [3136, -2.3, 4, 2], [41136, -2.3, 4, 2], [511136, -2.3, 4,  
2], [3316, -2.3, 4, 2], [41316, -2.3, 4, 2], [511316, -2.3, 4, 2], [43116, -2.3, 4, 2], [513116, -2.3, 4, 2], [511116, -2.3,  
4, 2], [245, -2.3, 2, 4], [3145, -2.3, 2, 4], [41145, -2.3, 2, 4], [511145, -2.3, 2, 4], [3415, -2.3, 2, 4], [41415, -2.3,
```

```
In [15]: #End CPU Time  
end = time.clock()  
CPU_Time = end - start
```

```
In [16]: cost = []  
for i in Result:  
    cost.append(i)  
  
cost = Sort(cost)
```

```
In [17]: print("b.The best solution " + str(cost[0][0])  
          + " And the point is " + str(cost[0][1]))  
print("c.The number of nodes (6 depths) " + str(len(cost)))  
print("d.CPU time is " + str(CPU_Time))  
  
b.The best solution 6425211 And the point is 5.300000000000001  
c.The number of nodes (6 depths) 38968  
d.CPU time is 41.641845999999994
```

For case#2, if we go through "down->suck->left->suck->left->up->suck->Noop->Noop->Noop", the score is  $-1.3+4-0.2-1+4-0.2-1-1.2+4-0.2+0+0+0=6.9$ .

For this algorithm,  $5.3 < 6.9$ , so the best solution found is not the optimal solution.

b) uniform cost graph search

```
Test Case 1

In [12]: #Start CPU Time
start = time.clock()

In [13]: Result=UCGS(ds1,vcl1,6)

Action is 1
Action is 2
Action is 3
Action is 4
Action is 5
Action is 6
The Frontier is [[11, 0, 3, 2], [12, -0.2, 3, 2], [13, -1, 3, 1], [14, -1.1, 3, 3], [15, -1.2, 2, 3], [16, -1.3, 4, 2]]
Action is 1
Action is 2
Action is 3
Action is 4
Action is 5
Action is 6
The Frontier is [[211, 0, 3, 2], [12, -0.2, 3, 2], [212, -0.2, 3, 2], [13, -1, 3, 1], [213, -1, 3, 1], [14, -1.1, 3, 3],
[214, -1.1, 3, 3], [15, -1.2, 2, 3], [215, -1.2, 2, 3], [16, -1.3, 4, 2], [216, -1.3, 4, 2]]
Action is 1
Action is 2
Action is 3
Action is 4
Action is 5
Action is 6

In [14]: #End CPU Time
end = time.clock()
CPU_Time = end - start

In [15]: cost = []
for i in Result:
    cost.append(i)

cost = Sort(cost)

In [16]: print("b.The best solution " + str(cost[0][0])
+ " And the point is " + str(cost[0][1]))
print("c.The number of nodes (6 depths) " + str(len(cost)))
print("d.CPU time is " + str(CPU_Time))

b.The best solution 6542421 And the point is 4.2
c.The number of nodes (6 depths) 36103
d.CPU time is 79.115439
```

For this algorithm, the best solution found is not the optimal solution, because  $4.2 < 5.3$ .

## Test Case 2

```
In [12]: #Start CPU Time
start = time.clock()
```

```
In [13]: Result=UCGS(ds2,vcl2,6)
```

```
Action is 1
Action is 2
Action is 3
Action is 4
Action is 5
Action is 6
The Frontier is [[11, 0, 3, 3], [12, -0.2, 3, 3], [13, -1, 3, 2], [14, -1.1, 3, 4], [15, -1.2, 3, 4], [16, -1.3, 4, 3]]
Action is 1
Action is 2
Action is 3
Action is 4
Action is 5
Action is 6
The Frontier is [[211, 0, 3, 3], [12, -0.2, 3, 3], [212, -0.2, 3, 3], [13, -1, 3, 2], [213, -1, 3, 2], [14, -1.1, 3, 4],
[214, -1.1, 3, 4], [15, -1.2, 3, 4], [215, -1.2, 3, 4], [16, -1.3, 4, 3], [216, -1.3, 4, 3]]
Action is 1
Action is 2
Action is 3
Action is 4
Action is 5
```

```
In [14]: #End CPU Time
end = time.clock()
CPU_Time = end - start
```

```
In [15]: cost = []
for i in Result:
    cost.append(i)

cost = Sort(cost)
```

```
In [16]: print("b.The best solution " + str(cost[0][0])
          + " And the point is " + str(cost[0][1]))
print("c.The number of nodes (6 depths) " + str(len(cost)))
print("d.CPU time is " + str(CPU_Time))
```

```
b.The best solution 6426211 And the point is 5.2
c.The number of nodes (6 depths) 32920
d.CPU time is 68.571494
```

For this algorithm, the best solution found is not the optimal solution, because  $5.2 < 6.9$ .

c) depth-limited depth-first tree search

### Test Case #1

```
In [14]: #Start CPU Time
start = time.clock()

Result = DLDFTS(ds1,vcl1,6)

#End CPU Time
end = time.clock()
CPU_Time = end - start
```

```
The Frontier is [[16, -1.3, 4, 2], [15, -1.2, 2, 2], [14, -1.1, 3, 3], [13, -1, 3, 1], [12, -0.2, 3, 2], [11, 0, 3, 2]]
The Frontier is [[266, -2.6, 5, 2], [265, -2.5, 3, 2], [264, -2.4000000000000004, 4, 3], [263, -2.3, 4, 1], [262, -1.5, 4, 2], [261, -1.3, 4, 2], [15, -1.2, 2, 2], [14, -1.1, 3, 3], [13, -1, 3, 1], [12, -0.2, 3, 2], [11, 0, 3, 2]]
The Frontier is [[3665, -3.8, 4, 2], [3664, -3.7, 5, 3], [3663, -3.6, 5, 1], [3662, -2.8000000000000003, 5, 2], [3661, -2.6, 5, 2], [265, -2.5, 3, 2], [264, -2.4000000000000004, 4, 3], [263, -2.3, 4, 1], [262, -1.5, 4, 2], [261, -1.3, 4, 2], [15, -1.2, 2, 2], [14, -1.1, 3, 3], [13, -1, 3, 1], [12, -0.2, 3, 2], [11, 0, 3, 2]]
The Frontier is [[46656, -5.1, 5, 2], [46655, -5.0, 3, 2], [46654, -4.9, 4, 3], [46653, -4.8, 4, 1], [46652, -4.0, 4, 2], [46651, -3.8, 4, 2], [3664, -3.7, 5, 3], [3663, -3.6, 5, 1], [3662, -2.8000000000000003, 5, 2], [3661, -2.6, 5, 2], [265, -2.5, 3, 2], [264, -2.4000000000000004, 4, 3], [263, -2.3, 4, 1], [262, -1.5, 4, 2], [261, -1.3, 4, 2], [15, -1.2, 2, 2], [14, -1.1, 3, 3], [13, -1, 3, 1], [12, -0.2, 3, 2], [11, 0, 3, 2]]
The Frontier is [[566565, -6.3, 4, 2], [566564, -6.1999999999999999, 5, 3], [566563, -6.1, 5, 1], [566562, -5.3, 5, 2], [566561, -5.1, 5, 2], [46655, -5.0, 3, 2], [46654, -4.9, 4, 3], [46653, -4.8, 4, 1], [46652, -4.0, 4, 2], [46651, -3.8, 4, 2], [3664, -3.7, 5, 3], [3663, -3.6, 5, 1], [3662, -2.8000000000000003, 5, 2], [3661, -2.6, 5, 2], [265, -2.5, 3, 2], [264, -2.4000000000000004, 4, 3], [263, -2.3, 4, 1], [262, -1.5, 4, 2], [261, -1.3, 4, 2], [15, -1.2, 2, 2], [14, -1.1, 3, 3], [13, -1, 3, 1], [12, -0.2, 3, 2], [11, 0, 3, 2]]
The Frontier is [[6665656, -7.6, 5, 2], [6665655, -7.5, 3, 2], [6665654, -7.4, 4, 3], [6665653, -7.3, 4, 1], [6665652, -6.5, 4, 2], [6665651, -6.3, 4, 2], [566564, -6.1999999999999999, 5, 3], [566563, -6.1, 5, 1], [566562, -5.3, 5, 2], [566561, -5.1, 5, 2], [46655, -5.0, 3, 2], [46654, -4.9, 4, 3], [46653, -4.8, 4, 1], [46652, -4.0, 4, 2], [46651, -3.8, 4, 2], [3664, -3.7, 5, 3], [3663, -3.6, 5, 1], [3662, -2.8000000000000003, 5, 2], [3661, -2.6, 5, 2], [265, -2.5, 3, 2], [264, -2.4000000000000004, 4, 3], [263, -2.3, 4, 1], [262, -1.5, 4, 2], [261, -1.3, 4, 2], [15, -1.2, 2, 2], [14, -1.1, 3, 3], [13, -1, 3, 1], [12, -0.2, 3, 2], [11, 0, 3, 2]]
```

```
In [15]: cost = []
for i in Result:
    cost.append(i)

cost = Sort(cost)
```

```
In [16]: print("b.The best solution " + str(cost[0][0])
        + " And the point is " + str(cost[0][1]))
print("c.The number of nodes (6 depths) " + str(len(cost)))
print("d.CPU time is " + str(CPU_Time))

b.The best solution 6111111 And the point is 0
c.The number of nodes (6 depths) 34699
d.CPU time is 2.733506
```

For this algorithm, the best solution found is not the optimal solution, because  $0 < 5.3$ .

## Test Case #2

```
In [14]: #Start CPU Time
start = time.clock()

Result = DLDFTS(ds2,vcl2,6)

#End CPU Time
end = time.clock()
CPU_Time = end - start
```

```
The Frontier is [[16, -1.3, 4, 3], [15, -1.2, 2, 3], [14, -1.1, 3, 4], [13, -1, 3, 2], [12, -0.2, 3, 3], [11, 0, 3, 3]]
The Frontier is [[266, -2.6, 5, 3], [265, -2.5, 3, 3], [264, -2.4000000000000004, 4, 4], [263, -2.3, 4, 2], [262, -1.5, 4, 3], [261, -1.3, 4, 3], [15, -1.2, 2, 3], [14, -1.1, 3, 4], [13, -1, 3, 2], [12, -0.2, 3, 3], [11, 0, 3, 3]]
The Frontier is [[3665, -3.8, 4, 3], [3664, -3.7, 5, 4], [3663, -3.6, 5, 2], [3662, -2.8000000000000003, 5, 3], [3661, -2.6, 5, 3], [265, -2.5, 3, 3], [264, -2.4000000000000004, 4, 4], [263, -2.3, 4, 2], [262, -1.5, 4, 3], [261, -1.3, 4, 3], [15, -1.2, 2, 3], [14, -1.1, 3, 4], [13, -1, 3, 2], [12, -0.2, 3, 3], [11, 0, 3, 3]]
The Frontier is [[46656, -5.1, 5, 3], [46655, -5.0, 3, 3], [46654, -4.9, 4, 4], [46653, -4.8, 4, 2], [46652, -4.0, 4, 3], [46651, -3.8, 4, 3], [3664, -3.7, 5, 4], [3663, -3.6, 5, 2], [3662, -2.8000000000000003, 5, 3], [3661, -2.6, 5, 3], [265, -2.5, 3, 3], [264, -2.4000000000000004, 4, 4], [263, -2.3, 4, 2], [262, -1.5, 4, 3], [261, -1.3, 4, 3], [15, -1.2, 2, 3], [14, -1.1, 3, 4], [13, -1, 3, 2], [12, -0.2, 3, 3], [11, 0, 3, 3]]
The Frontier is [[566565, -6.3, 4, 3], [566564, -6.1999999999999999, 5, 4], [566563, -6.1, 5, 2], [566562, -5.3, 5, 3], [566561, -5.1, 5, 3], [46655, -5.0, 3, 3], [46654, -4.9, 4, 4], [46653, -4.8, 4, 2], [46652, -4.0, 4, 3], [46651, -3.8, 4, 3], [3664, -3.7, 5, 4], [3663, -3.6, 5, 2], [3662, -2.8000000000000003, 5, 3], [3661, -2.6, 5, 3], [265, -2.5, 3, 3], [264, -2.4000000000000004, 4, 4], [263, -2.3, 4, 2], [262, -1.5, 4, 3], [261, -1.3, 4, 3], [15, -1.2, 2, 3], [14, -1.1, 3, 4], [13, -1, 3, 2], [12, -0.2, 3, 3], [11, 0, 3, 3]]
The Frontier is [[6665656, -7.6, 5, 3], [6665655, -7.5, 3, 3], [6665654, -7.4, 4, 4], [6665653, -7.3, 4, 2], [6665652, -6.5, 4, 3], [6665651, -6.3, 4, 3], [566564, -6.1999999999999999, 5, 4], [566563, -6.1, 5, 2], [566562, -5.3, 5, 3], [566561, -5.1, 5, 3], [46655, -5.0, 3, 3], [46654, -4.9, 4, 4], [46653, -4.8, 4, 2], [46652, -4.0, 4, 3], [46651, -3.8, 4, 3], [3664, -3.7, 5, 4], [3663, -3.6, 5, 2], [3662, -2.8000000000000003, 5, 3], [3661, -2.6, 5, 3], [265, -2.5, 3, 3], [264, -2.4000000000000004, 4, 4], [263, -2.3, 4, 2], [262, -1.5, 4, 3], [261, -1.3, 4, 3], [15, -1.2, 2, 3], [14, -1.1, 3, 4], [13, -1, 3, 2], [12, -0.2, 3, 3], [11, 0, 3, 3]]
```

```
In [15]: cost = []
for i in Result:
    cost.append(i)

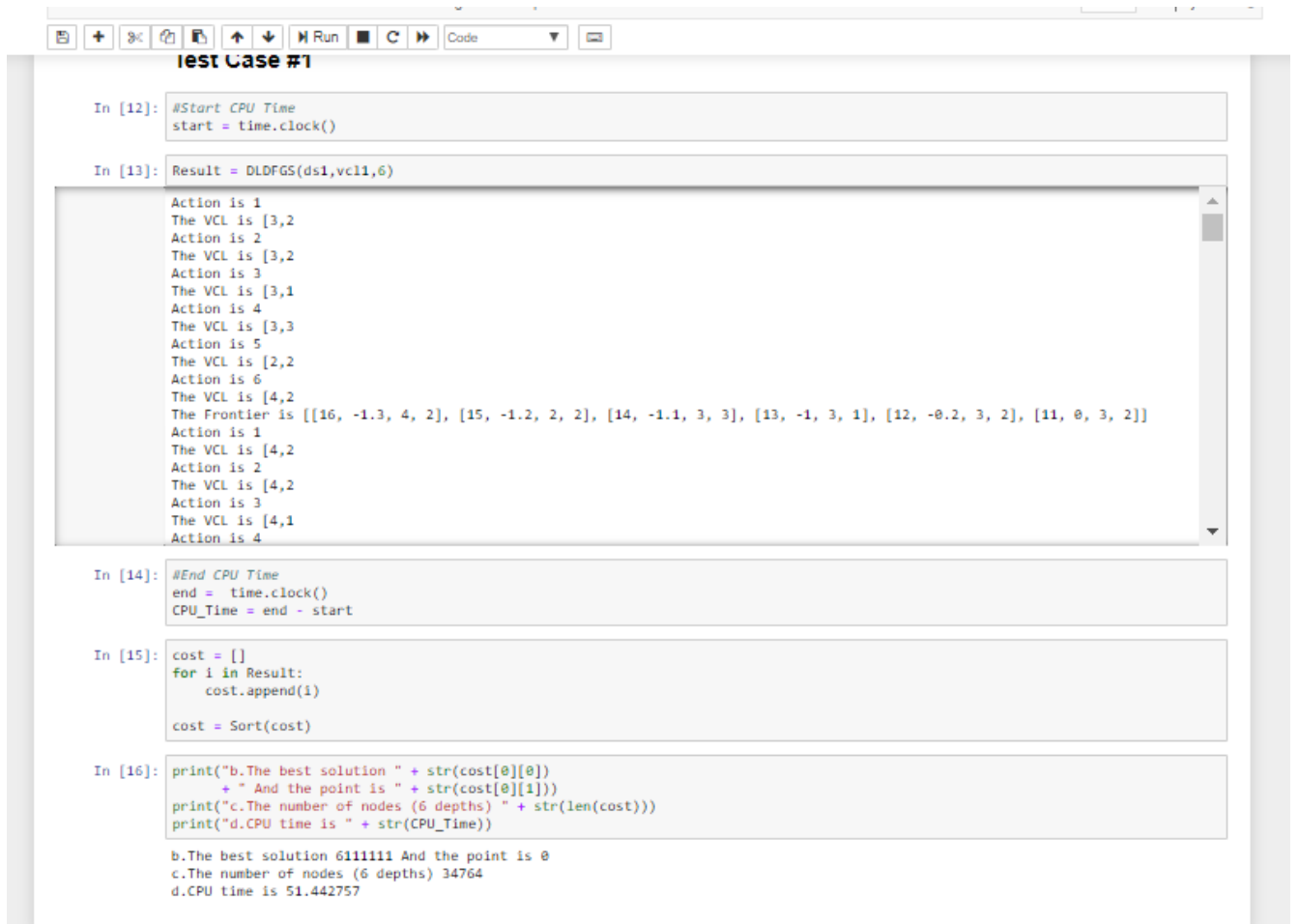
cost = Sort(cost)
```

```
In [16]: print("b.The best solution " + str(cost[0][0])
        + " And the point is " + str(cost[0][1]))
print("c.The number of nodes (6 depths) " + str(len(cost)))
print("d.CPU time is " + str(CPU_Time))
```

```
b.The best solution 6111111 And the point is 0
c.The number of nodes (6 depths) 38968
d.CPU time is 3.0909140000000006
```

For this algorithm, the best solution found is not the optimal solution, because  $0 < 6.9$ .

d) depth-limited depth-first graph search



```
test Case #1

In [12]: #Start CPU Time
start = time.clock()

In [13]: Result = DLDFGS(ds1,vc11,6)

Action is 1
The VCL is [3,2
Action is 2
The VCL is [3,2
Action is 3
The VCL is [3,1
Action is 4
The VCL is [3,3
Action is 5
The VCL is [2,2
Action is 6
The VCL is [4,2
The Frontier is [[16, -1.3, 4, 2], [15, -1.2, 2, 2], [14, -1.1, 3, 3], [13, -1, 3, 1], [12, -0.2, 3, 2], [11, 0, 3, 2]]
Action is 1
The VCL is [4,2
Action is 2
The VCL is [4,2
Action is 3
The VCL is [4,1
Action is 4

In [14]: #End CPU Time
end = time.clock()
CPU_Time = end - start

In [15]: cost = []
for i in Result:
    cost.append(i)
cost = Sort(cost)

In [16]: print("b.The best solution " + str(cost[0][0])
+ " And the point is " + str(cost[0][1]))
print("c.The number of nodes (6 depths) " + str(len(cost)))
print("d.CPU time is " + str(CPU_Time))

b.The best solution 6111111 And the point is 0
c.The number of nodes (6 depths) 34764
d.CPU time is 51.442757
```

For this algorithm, the best solution found is not the optimal solution, because  $0 < 5.3$ .



## Test Case #2

```
In [12]: #Start CPU Time  
start = time.clock()
```

```
In [13]: Result = DLDFGS(ds2,vcl2,6)
```

```
Action is 1  
The VCL is [3,3  
Action is 2  
The VCL is [3,3  
Action is 3  
The VCL is [3,2  
Action is 4  
The VCL is [3,3  
Action is 5  
The VCL is [2,3  
Action is 6  
The VCL is [4,3  
The Frontier is [[16, -1.3, 4, 3], [15, -1.2, 2, 3], [14, -1.1, 3, 3], [13, -1, 3, 2], [12, -0.2, 3, 3], [11, 0, 3, 3]]  
Action is 1  
The VCL is [4,3  
Action is 2  
The VCL is [4,3  
Action is 3  
The VCL is [4,2  
Action is 4
```

```
In [14]: #End CPU Time  
end = time.clock()  
CPU_Time = end - start
```

```
In [15]: cost = []  
for i in Result:  
    cost.append(i)  
  
cost = Sort(cost)
```

```
In [16]: print("b.The best solution " + str(cost[0][0])  
        + " And the point is " + str(cost[0][1]))  
print("c.The number of nodes (6 depths) " + str(len(cost)))  
print("d.CPU time is " + str(CPU_Time))
```

```
b.The best solution 6111111 And the point is 0  
c.The number of nodes (6 depths) 40118  
d.CPU time is 64.070484
```

For this algorithm, the best solution found is not the optimal solution, because  $0 < 6.9$ .

## Reference

<https://www.geeksforgeeks.org/python-sort-list-according-second-element-sublist/>

<https://www.geeksforgeeks.org/find-first-last-digits-number/>