Preliminaries

Game board representation:

- Use list-like pattern to show pegs and disks (Fig. 1). Numbers are representing disks where smaller number means smaller disks. From left to right, each column means a peg.

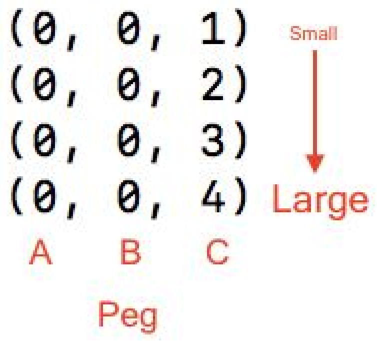
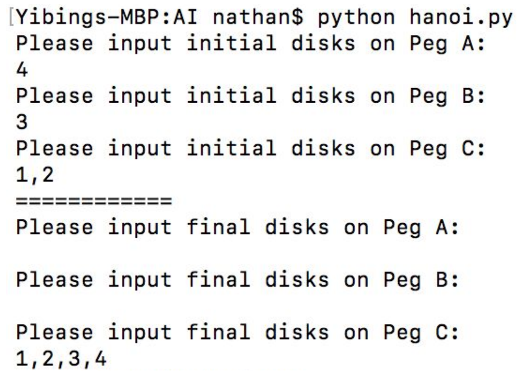


Fig. 1. Game Board Representation

Inputs:

- Will ask disks on each peg, for initial state and final state.
- Please use ',' to split, and do not add extra comma at the end.
- For a single stage, all pegs should have distinct disks(aka, all numbers should be different), and smaller numbers should always in front of larger ones on each peg.
- If one peg does not contain any disks, simply press enter and DO NOT input any value.
- A valid input shows in Fig. 2.



```
[Yibings-MBP:AI nathan$ python hanoi.py
Please input initial disks on Peg A:
4
Please input initial disks on Peg B:
3
Please input initial disks on Peg C:
1,2
============
Please input final disks on Peg A:

Please input final disks on Peg B:

Please input final disks on Peg C:
1,2,3,4
```

Fig 2. Valid Input

- The code will show your input right after using game board representation type.
- You can specify a final state that all disks are on one peg if you like, as long as it's valid input.
- PLEASE NOTE that this code assume all input is valid. If not, it will throw an error and it will stop.
- Input also contains the check of number of disks between start and final state. If it doesn't match it will require input again.

Code Insights

BFS:
A list that operates like a queue is used to achieve this, together with a dictionary as {children:parent} to represent a tree.

From start state, the hanoi() function will keep searching for all possible moves and put in the back of a queue called *nextState* until this queue is empty, or the first element popped from the queue matches with the final state. A list called *visitedState* is used for tracking all searched state, and a dictionary called *index* with {children:parent} will perform as a tree tracing structure. Finally when it reaches end state, we will trace back from the *final_index* back to zero, which is the start state.

DFS:
A tree structure is used here. Class *treeNode* to create node for each state, it contains a list of *children* , a *parent*, a *state*, a *level* to be used in DFS and a *distance* to be used in Best first search.

There are still two list used: *nextState* and *visitedState*. This time *nextState* works as a stack stores node of the tree and *visitedState* stores the state that has already been list for all its children. And with *targetNode.level*, we can check which path is shorter by comparing the current matched node level. For all possible states, if the state is already visited, then we compare if this path is shorter than former visited path, if so we link the node to the current path. Otherwise we push the new node to stack.

Best First Search:
My heuristic function compares differences of disks between input state with final state. It add all disk represent numbers which currently doesn't match. Like [[1,2,3],[],[]] vs [[3],[1],[2]] will return 3, [[2,3],[1],[]] vs [[1,2],[3],[]] will return 4. This is how I measure the distance between a state and a final state.

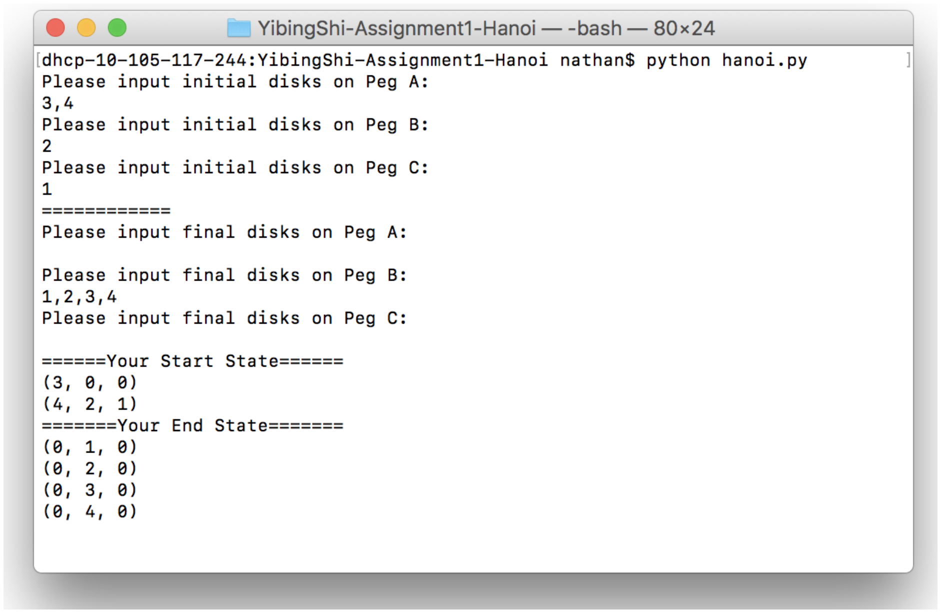
In this search, every time the current node is picked from a list where the first node has the shortest distance towards the final state. And every time if a possible move is not in the checked list, it will be added and calculate the distance.

My Output will contain shortest path found in all three methods, separated by "=" and other notable comments.

Results:

> Test Case 1:

Input:



```
dhcp-10-105-117-244:YibingShi-Assignment1-Hanoi nathan$ python hanoi.py
Please input initial disks on Peg A:
3,4
Please input initial disks on Peg B:
2
Please input initial disks on Peg C:
1
============
Please input final disks on Peg A:

Please input final disks on Peg B:
1,2,3,4
Please input final disks on Peg C:

======Your Start State======
(3, 0, 0)
(4, 2, 1)
=======Your End State=======
(0, 1, 0)
(0, 2, 0)
(0, 3, 0)
(0, 4, 0)
```

Output:

BFS:

```
================================
======Start Game with BFS======
Initial Step 0
(3, 0, 0)
(4, 2, 1)
---
Step 1
(3, 1, 0)
(4, 2, 0)
---
Step 2
(0, 1, 0)
(4, 2, 3)
---
Step 3
(1, 0, 0)
(4, 2, 3)
---
Step 4
(1, 0, 2)
(4, 0, 3)
---
Step 5
(0, 0, 1)
(0, 0, 2)
```

```
(4, 0, 3)
---
Step 6
(0, 0, 1)
(0, 0, 2)
(0, 4, 3)
---
Step 7
(0, 1, 2)
(0, 4, 3)
---
Step 8
(0, 1, 0)
(2, 4, 3)
---
Step 9
(1, 0, 0)
(2, 4, 3)
---
Step 10
(1, 3, 0)
(2, 4, 0)
---
Step 11
(0, 3, 0)
(2, 4, 1)
---
Step 12
(0, 2, 0)
(0, 3, 0)
(0, 4, 1)
---
Step 13
(0, 1, 0)
(0, 2, 0)
(0, 3, 0)
(0, 4, 0)
---
=======End Game with BFS=======
```

DFS:

```
======Start Game with DFS======
Initial Step 0
(3, 0, 0)
(4, 2, 1)
---
Step 1
(3, 1, 0)
(4, 2, 0)
---
Step 2
(0, 1, 0)
(4, 2, 3)
---
Step 3
(1, 0, 0)
(4, 2, 3)
---
```

```
Step 4
(1, 0, 2)
(4, 0, 3)
---
Step 5
(0, 0, 1)
(0, 0, 2)
(4, 0, 3)
---
Step 6
(0, 0, 1)
(0, 0, 2)
(0, 4, 3)
---
Step 7
(0, 1, 2)
(0, 4, 3)
---
Step 8
(0, 1, 0)
(2, 4, 3)
---
Step 9
(1, 0, 0)
(2, 4, 3)
---
Step 10
(1, 3, 0)
(2, 4, 0)
---
Step 11
(0, 3, 0)
(2, 4, 1)
---
Step 12
(0, 2, 0)
(0, 3, 0)
(0, 4, 1)
---
Step 13
(0, 1, 0)
(0, 2, 0)
(0, 3, 0)
(0, 4, 0)
---
=======End Game with DFS=======
```

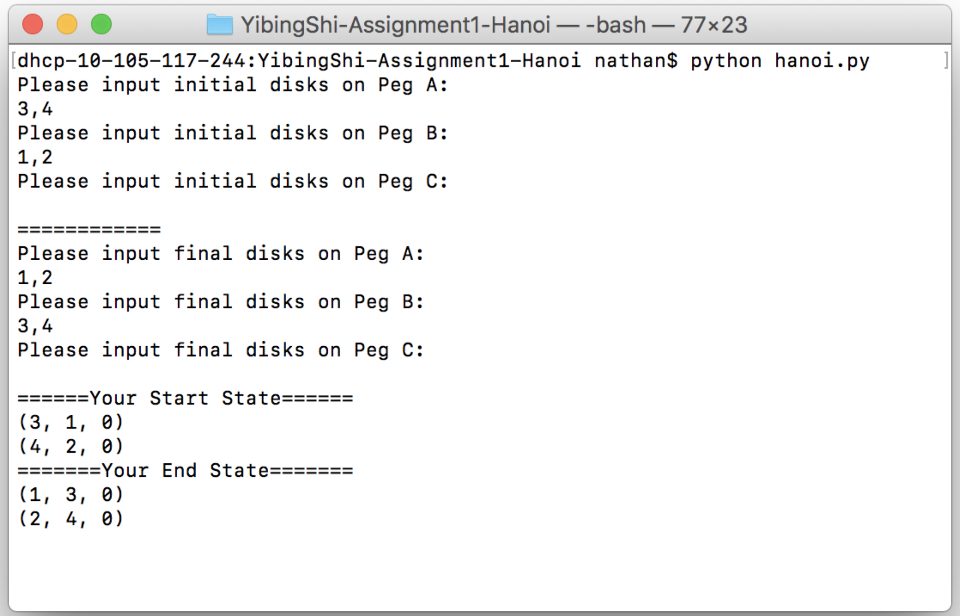
Best first search:

```
====Start Game with Best FS====
Initial Step 0
(3, 0, 0)
(4, 2, 1)
---
Step 1
(3, 1, 0)
(4, 2, 0)
---
Step 2
```

```
(0, 1, 0)
(4, 2, 3)
---
Step 3
(1, 0, 0)
(4, 2, 3)
---
Step 4
(1, 0, 2)
(4, 0, 3)
---
Step 5
(0, 0, 1)
(0, 0, 2)
(4, 0, 3)
---
Step 6
(0, 0, 1)
(0, 0, 2)
(0, 4, 3)
---
Step 7
(0, 1, 2)
(0, 4, 3)
---
Step 8
(0, 1, 0)
(2, 4, 3)
---
Step 9
(1, 0, 0)
(2, 4, 3)
---
Step 10
(1, 3, 0)
(2, 4, 0)
---
Step 11
(0, 3, 0)
(2, 4, 1)
---
Step 12
(0, 2, 0)
(0, 3, 0)
(0, 4, 1)
---
Step 13
(0, 1, 0)
(0, 2, 0)
(0, 3, 0)
(0, 4, 0)
---
=====End Game with Best FS=====
```

> Test Case 2:

Input:



```
YibingShi-Assignment1-Hanoi — -bash — 77×23
dhcp-10-105-117-244:YibingShi-Assignment1-Hanoi nathan$ python hanoi.py
Please input initial disks on Peg A:
3,4
Please input initial disks on Peg B:
1,2
Please input initial disks on Peg C:

============
Please input final disks on Peg A:
1,2
Please input final disks on Peg B:
3,4
Please input final disks on Peg C:

======Your Start State======
(3, 1, 0)
(4, 2, 0)
=======Your End State=======
(1, 3, 0)
(2, 4, 0)
```

Output:

BFS:

```
================================
======Start Game with BFS======
Initial Step 0
(3, 1, 0)
(4, 2, 0)
---
Step 1
(0, 1, 0)
(4, 2, 3)
---
Step 2
(1, 0, 0)
(4, 2, 3)
---
Step 3
(1, 0, 2)
(4, 0, 3)
---
Step 4
(0, 0, 1)
(0, 0, 2)
(4, 0, 3)
---
Step 5
(0, 0, 1)
```

```
(0, 0, 2)
(0, 4, 3)
---
Step 6
(0, 1, 2)
(0, 4, 3)
---
Step 7
(0, 1, 0)
(2, 4, 3)
---
Step 8
(1, 0, 0)
(2, 4, 3)
---
Step 9
(1, 3, 0)
(2, 4, 0)
---
=======End Game with BFS=======
```

DFS:

```
======Start Game with DFS======
Initial Step 0
(3, 1, 0)
(4, 2, 0)
---
Step 1
(0, 1, 0)
(4, 2, 3)
---
Step 2
(1, 0, 0)
(4, 2, 3)
---
Step 3
(1, 0, 2)
(4, 0, 3)
---
Step 4
(0, 0, 1)
(0, 0, 2)
(4, 0, 3)
---
Step 5
(0, 0, 1)
(0, 0, 2)
(0, 4, 3)
---
Step 6
(0, 1, 2)
(0, 4, 3)
---
Step 7
(0, 1, 0)
(2, 4, 3)
---
Step 8
```

```
(1, 0, 0)
(2, 4, 3)
---
Step 9
(1, 3, 0)
(2, 4, 0)
---
=======End Game with DFS=======
```

Best First Search:

```
====Start Game with Best FS====
Initial Step 0
(3, 1, 0)
(4, 2, 0)
---
Step 1
(0, 1, 0)
(4, 2, 3)
---
Step 2
(1, 0, 0)
(4, 2, 3)
---
Step 3
(1, 0, 2)
(4, 0, 3)
---
Step 4
(0, 0, 1)
(0, 0, 2)
(4, 0, 3)
---
Step 5
(0, 0, 1)
(0, 0, 2)
(0, 4, 3)
---
Step 6
(0, 1, 2)
(0, 4, 3)
---
Step 7
(0, 1, 0)
(2, 4, 3)
---
Step 8
(1, 0, 0)
(2, 4, 3)
---
Step 9
(1, 3, 0)
(2, 4, 0)
---
=====End Game with Best FS=====
```