FINAL REPORT

Teammate: Yi-Ling Lin, Student ID - 100958447

1) We have chosen array or list as we coded in Python as the representation of our problem. The rationale behind choosing it was to use a data structure which could simplify our operation. The first element in the array/list is the number of the biggest disk, followed by disks with smaller numbers. We can easily remove the outermost disk and add a disk to the array/list. The operations would be trickier if we used classes or dictionary etc.

2) The search algorithms we have used in the projects are Breadth-first search and A-star search.

Breadth-first search expands each state at the same depth level d, and if a solution is not found, it moves to the depth level d+1 and continues its search for the goal in the same fashion. Breadth-first search always finds a solution with a minimal number of actions, because when it is generating nodes at depth d, it has already generated all the nodes at depth d-1. If the branching factor is finite, then Breadth-First search can be said to be complete.

A-star search is an extension of best-first search, and it assigns a cost specified by a heuristic function which is the estimated cost of the shortest path from current to a goal state. It provides a direction for our search to find the solution. It doesn't follow a greedy approach like Best-first search. A-star search can lead to the shortest path/solution if our heuristic function is admissible.

3) We have devised two heuristic functions, one of which is admissible and the other is not.

The admissible heuristic function- To create the admissible heuristic function, we have calculated the out-of-place disks by taking the difference of the disks in the goal state of peg 4 from the disks in-place in the current state of peg 4. For example, if we have four disks in our final goal state as [4, 3 ,2, 1] and one disk at current peg four as [1] our difference/ heuristic value is four, but if the current peg four state is as [4] our difference/ heuristic value will be three which means we are getting closer to our final goal state. This function is admissible because we never overestimate the true number of moves when we use this approach. This statement is also strengthened by the provided table, which shows the number of moves required for different n, the path is similar to the one chosen by the breath-first search.

The inadmissible heuristic function- Unlike the admissible function, where we calculated the heuristic based on the goal state of peg four, we have taken the final state of all the pegs into account and have found out of place disks according to the final state of all the pegs. This gives us a different heuristic function. It is overestimating our path, but as n grows more, it solves our

problem in lesser time and lesser number of expansions than the admissible heuristic we devised. The number of moves proves this point. I am confirming it with the whole experimental data in the tabular form in point 5.

4) When we use the admissible function, we can find the solution optimally for n =8 under 10 minutes.

When we use the inadmissible function, we can find the solution sub-optimally for n = 8 under 10 minutes.

For n >8 the time to solve the problem rises at a very sharp rate.

5) This table is a combined table, which covers the partial answer to point 4 and point 6 and answer for this point (point 5).

Breadth-first search

Number of disks	Number of moves/ length of solutions	States expanded	Time taken in seconds
1	1	4	0.0036
2	3	14	0.006
3	5	56	0.038
4	9	256	0.56
5	13	1019	7.85
6	17	4095	127.08
7	25	16381	2117

A-star Search using Admissible heuristic

Number of disks	Number of moves/ length of solutions	States expanded	Time taken in seconds
1	1	2	0.002
2	3	4	0.003
3	5	8	0.005
4	9	33	0.03
5	13	93	0.14
6	17	317	1.31
7	25	1266	17.6
8	33	4576	226
9	41	13820	2050

A-star search using inadmissible heuristic

Number of disks	Number of moves/ length of solutions	States expanded	Time taken in seconds
1	1	4	0.003
2	3	11	0.004
3	5	26	0.014
4	9	85	0.101
5	15	188	0.47
6	27	497	3.15
7	49	1206	19
8	93	3102	140
9	179	8213	1080

6) For max K = 8, the optimal length found was 33, expanded states.

For max K = 8, the sub-optimal length found is 93.

For other values of K, the length of solutions is given in a combined tabular data in the previous answer to avoid redundancy of data.

Extension

1) PDB is a lookup table constructed using the subset of the problem. In general, a PDB is built by running a breadth-first search in the pattern space backwards from the goal pattern until the entire pattern space is spanned. The rationale behind using the PDB is to find the minimum value of heuristic cost from each state and use that heuristic value whilst doing A-star search. Doing this gives us a better direction to find the result. It gives very accurate results and is amongst the most efficient types of heuristic function. The expense of this search is amortised over subsequent problem instances.

2) For our problem, we have used a compressed PDB approach which can save a lot of memory. For example, a pattern database of 15 disks requires 4^15 = 2^30 bytes or a gigabyte of memory. Therefore, we are using the value computed for one random pattern assuming this would still give us fine results. We could solve this problem using subsets of pattern database as well. We have used PDB for removing m smallest disks to the goal state.

The biggest m we found under 5 minutes for our PDB is 5.

3) Since our PDB model is a very simple one due to compression, it's not as efficient as the other sophisticated PDB heuristic function. It still does a fine job of finding the path and is better than using the Breadth-search.

For m= 5

For n= 6, time taken = 129 seconds, moves = 17 (shortest path according to Breadth-first)

For n= 7, time taken = more than 10 minutes. Therefore, not specifying it.

The PDB function also appears to be admissible cos it's choosing the lower bound as actually moving all the disks would way more moves.