Purnil Chaitanya

Unit 1 Searches Day 2 Note

More about BFS:

- Think about BFS algorithm generates children nodes for each expansion. Given the branch factor 'b' and the depth 'd', what is the total number of nodes generated?

$1 + b + b^2 + b^3 + \ldots + b^d$ nodes are generated

~~expanded~~ with $O(b^{d+1})$ efficiency

- Time efficiency: $O(b^{d+1})$ b/c goal is detected after depth d is expanded.
- Space complexity: $O(b^d)$ b/c there will be $O(b^{d-1})$ nodes in explored set and $O(b^d)$ in frontier.

Depth	Nodes	Time	Memory
2	110	.11 milliseconds	107 kilobytes
4	11,110	11 milliseconds	10.6 megabytes
6	10^6	1.1 seconds	1 gigabyte
8	10^8	2 minutes	103 gigabytes
10	10^{10}	3 hours	10 terabytes
12	10^{12}	13 days	1 petabyte
14	10^{14}	3.5 years	99 petabytes
16	10^{16}	350 years	10 exabytes

More about DFS:

- Time efficiency: $O(b^m)$, m is the maximum depth so possible to be m > d (the depth of shallowest solution)

- Space efficiency: $O(bm)$

Which search algorithm is optimal?

BFS because it usually finds

On smaller data sets BFS is optimal, but

on much larger datasets DFS is optimal

Analysis:

Task 1: Find the all solvable 8-puzzle initial states, and store them in "solvable_8_puzzle.txt"

- How many initial states did you find? 9!/2
- How did you implement this task? Briefly explain your algorithm:

Start from the goal state and permute all children and remove duplicates

Task 2: Go back to your Lab 1. When the input state was not solvable, how many nodes were expanded?

- The maximum number of states: 9!/2
- Compare the numbers you found in Task 1 and Task 2. What's your thought? (You may explain this mathematically.)

Because the most you have to check every permutation of the board to conclude that the current state is not solvable, and there are 9!/2 solvable states, which means there are also 9!/2 unsolvable states

Task 3: Go back to your Lab 1. Why is it, usually, not the shortest path when you use DFS? Explain with diagrams.

DFS goes as deep as possible and returns the first solution it finds, so on average you usually will get a solution, not the shortest solution

Task 4: Let's calculate a cost from each state to the goal and use the cost to search the goal. If we can call this method(algorithm) as the cheapest first search and we use a priority queue for the frontier,

- is it guaranteed to reach the goal? (Y/N) some initial states aren't solvable
- Is it guaranteed to find the shortest path? (Y/N) but it will find a path if one exists
- What should be the benefit to calculate costs in each state?

If you have the costs for each state then it becomes much more trivial to look for the cheapest option and go there.