Programming Project: Assignment Part I

ThankYouEnjoy

Woo Jung
Thomas Tarantino

October
2019

Group declaration: FILL IN

Level	Frontier	Time	Memory Used	Solution length	States Generated
SAD1	BFS	0.201 s	24.33 MB	19	80
SAD1	DFS	0.204 s	13.95 MB	27	75
SAD2	BFS	—	exceeded	—	—
SAD2	DFS	0.184 s	10.85 MB	25	86
SAfriendofDFS	BFS	—	exceeded	—	—
SAfriendofDFS	DFS	0.225 s	31.83 MB	60	305
SAfriendofBFS	BFS	0.686 s	138.64 MB	3	1,227
SAfriendofBFS	DFS	—	exceeded	—	—
SAFirefly	BFS	—	exceeded	—	—
SAFirefly	DFS	—	exceeded	—	—
SACrunch	BFS	—	exceeded	—	—
SACrunch	DFS	—	exceeded	—	—

Table 1: Benchmarks table for Exercise 1

1 Exercise 1

(a) Refer to Table 1.

(b) There are three boxes adjacent to the agent in the initial state. Using BFS, the agent is forced to consider neighbor boxes at the present depth prior to moving on in each iteration. This means, the agent has significantly more states to iterate through before moving onto a higher depth state.

(c) Our implementation of DFS differs from the implementation of BFS in that it uses a stack instead of a queue since DFS requires LIFO data structure.

(d) Our BFS algorithm has a much better performance on this level because the initial state of the level is close to the goal state in the sense that it does not require much moving of the boxes. On the other hand, our DFS algorithm performed poorly because the algorithm has to iterate through each state until success or failure which resulted in a much longer search time – well it didn't end up solving.

(e) Our BFS algorithm performs poorly because there are too many boxes in the map so the search space becomes exponential. Since DFS is not limited to searching the entire depth level's search space before progressing, the DFS is not bound by the number of boxes in the map.

(f) Refer to Table 1.

Level	Frontier	Time	Memory Used	Solution length	States Generated
SAD1	BFS	0.041 s	3.12 MB	19	80
SAD1	DFS	0.045 s	2.40 MB	27	75
SAD2	BFS	14.458 s	563.12 MB	19	635,190
SAD2	DFS	0.032 s	2.62 MB	25	86
SAfriendofDFS	BFS	1.718 s	53.12 MB	8	89,112
SAfriendofDFS	DFS	0.039 s	2.90 MB	60	305
SAfriendofBFS	BFS	0.071 s	4.62 MB	3	1,227
SAfriendofBFS	DFS	25.781 s	1894.12 MB	981,528	2,953,986
SAFirefly	BFS	18.280 s	2052.00 MB	60	1,961,416
SAFirefly	DFS	30.897 s	3859.00 MB	2,517,074	4,089,953
SACrunch	BFS	389.379 s	4831.51 MB	98	9,285,293
SACrunch	DFS	6.096 s	664.00 MB	380,992	1,023,377

Table 2: Benchmarks table for Exercise 2

2 Exercise 2

(1) For the new benchmarks, refer to Table 2.

(2) The improvement is quite significant in comparison to the first run. We saw improvements of 10 times in run-time. For instance, the `BFS` frontier on level `SAfriendofBFS` took 0.686 seconds in exercise 1 but only 0.071 seconds in exercise 2 — this amounts to an improvement of $\frac{0.686}{0.071} = 9.66$ times. Furthermore, we saw huge improvements in memory use. For example, many of the levels in exercise 1 could not complete due to exceeding the memory limit (around 4 5 GB) but exercise 2 was able to finish all levels without exceeding the same memory limit.

(3) We modified the code in three main ways: First, we changed the location of walls and goals in the `State` class into static variables so that they can be shared across instances. This significantly reduces the memory usage of the program since each instance of the class does not need its own walls and goals object. Next, we made the size (width and height) of the arrays dynamic — not fixed to a length of 130. Finally, we made a similar change to the `Color` arrays by limiting the size to the number of agents and boxes.

Level	Time	Memory Used	Solution length	States Generated
SAsoko1_04	0.037 s	2.31 MB	2	3
SAsoko1_08	0.038 s	2.25 MB	6	15
SAsoko1_16	0.034 s	2.50 MB	14	68
SAsoko1_32	0.098 s	2.81 MB	30	266
SAsoko1_64	0.106 s	5.28 MB	62	1,034
SAsoko1_128	0.151 s	9.01 MB	126	4,128
SAsoko2_04	0.027 s	2.31 MB	2	44
SAsoko2_08	0.070 s	4.00 MB	6	660
SAsoko2_16	0.303 s	42.63 MB	14	8,580
SAsoko2_32	3.907 s	571.00 MB	30	118,093
SAsoko2_64	—	Memory exceeded	—	—
SAsoko2_128	—	Memory exceeded	—	—
SAsoko3_04	0.478 s	25.20 MB	8	17,016
SAsoko3_08	—	Memory exceeded	—	—
SAsoko3_04	—	Memory exceeded	—	—
SAsoko3_04	—	Memory exceeded	—	—
SAsoko3_04	—	Memory exceeded	—	—
SAsoko3_04	—	Memory exceeded	—	—

Table 3: BFS Benchmarks table for Exercise 3

3 Exercise 3

(1) Refer to Table 3.

(2) There is a big difference between the biggest size levels that the algorithm can handle among the three types of maps. This can be attributed to the differences in the growth of the state space sizes.

First, the `SAsoko1` map is a simple horizontal map of height 1 with one agent and one box. This limits the agent's possible actions only to moving West or East and either pushing the box East or pulling the box West. Furthermore, the increase in the level's size only increases the width of the map whereas the height of the map is constant. Thus the increase in level size has a linear relationship with the size of the state space. Let ℓ be the size of the level such that `SAsoko1_04` would have $\ell = 4$. Let N be the number of agents, M the number of boxes, and A the number of possible actions. Then, it follows that the size of the state space has $O(\ell NMA)$. Since N, M, A are constant for all ℓ, this can be simplified to $O(\ell)$, a linear relationship.

Second, the `SAsoko2` map is a square map of size $\ell \times \ell$ with one agent $N = 1$ and one box $M = 1$. Since the map is not limited as in the case with `SAsoko1`, the number of possible actions is higher. However, since there is one agent and one box, the size of the action set increases with the size of the map. Thus it follows that the size of the state space has $O(\ell^2 NMA)$. Since $N = 1$ and $M = 1$, the number of possible actions A remains constant. Thus the size of the state space resembles quadratic growth $O(\ell^2)$.

Lastly,

Level	Time	Memory Used	Solution length	States Generated

Table 4: BFS benchmarks for selected Sokoban levels

4 Exercise 4

- Refer to Table 4 for benchmarks for the selected Sokoban levels.