SAI assignment 4

Ryan Sleeuwaegen, Sjouk Ketwaru
s3122166, s3287297

23 december 2022

Question 1

1.3.b)

After running value iteration we noticed that the agent chose to go for the higher reward, even though this takes more actions than the lower reward. The reason for this is that the extra amount of moves it takes to get to the higher reward is less then the gap between the two rewards. So in this case the difference in reward is 10. The extra amount of steps it takes to get to the high reward instead of the low reward is 9 more steps. Since these this punishment is less then the reward it will take the higher reward. If we moved the lower reward a few spaces closer it would take the lower reward since the difference between the 2 rewards wouldn't be 9 but more then 10, and since the reward difference is 10, it would go to the lower reward.

Question 2

2.4.a)

The formula to calculate the amount of unique states is:

$$s = l * 2^k$$

s = amount of unique locations
l = amount of free locations
k = amount of keys.

So in this case:

$$s = 10.000 * 2^{30}$$

$$s = 1,0737... * 10^{13}$$

2.4.b)

Since each value takes up 4 bytes. The amount of memory needed in bytes is:

$$m = 4 * 1,0737... * 10^{13}$$

$$m = 4,2949... * 10^{13}$$

m = amount of memory (in bytes)

We know 1 byte is roughly $1.0 * 10^{-12}$ terabyte. This means the total amount of memory needed is:

$$m = 4,2949... * 10^{13} * 1.0 * 10^{-12}$$

This comes down to roughly 42,9 terabytes.

2.4.c)

As we saw in question 2.4.b, a program like this would take up roughly 42.9 terabytes. A normal laptop with 16 gigabytes of ram would never be able to process a program like this, it would run out of memory.

But lets assume it would, then the following time would be needed:

$$2,60\, GHz = 2,60 * 10^9\, cycles/second$$

$$1\, cycle = 64\, bits = 8\, bytes$$

$$8 * 2,60 * 10^9\, cycles = 2,08 * 10^{10}\, bytes/second$$

$$4,2949... * 10^{13}\, bytes\ /\ 2,08 * 10^{10}\, bytes/second = 2064,85...\, seconds$$

$$2064,85...\, seconds\ /\ 60 = 34.4\, minutes$$

2.4.d)

Curse of dimensionality: The number of unique states grows exponentially in problem size (number of variables that the state describes)

The aspect of our problem definition that causes the exponential growth is the amount of keys. The reason is that, for every location, there is a unique state for every possible combination of keys. If we only have 2 keys that is only 2^2 different combinations (no key, key a, key b, key a and b) per location. However, if we have 30 keys, there are 2^{30} different combinations possible per location.

2.5.a)

Iterative deepening tree search is a depth first search algorithm where the search iteratively gets executed, with a higher depth limit till either a solution is found or the entire tree is searched. The depth of the shortest path towards the goal is: 20. because the minimal amount of steps to achieve the goal is 20. The time complexity for iterative deepening is,

$$O(b^d)$$

In general the b = breaching factor, d = depth. so in our case that would be b = 4, because there are 4 actions, d = 20, because that is the minimal search depth. which gives us a time complexity of:

$$O(b^d) = O(4^{20}) = O(1.099.511.627.776)$$

2.5.b)

When empirically observing how long Dynamic Programming took to solve the prison problem, we measured a time of less then a second. This is faster then the time complexity we estimated for Iterative deepening tree search.

2.5.c)

When comparing Dynamic Programming to Tree/Graph Search the main difference is in the memory requirement and speed.

Dynamic Programming is faster then Tree/Graph Search, but it requires more memory.
Tree/Graph Search on the other hand is slower then Dynamic Programming, but the memory requirement varies, and is usually lower then that of Dynamic Programming.