A5 Optimization (Task 6)
James Tu (jt737)
Lam Le (ldl56)
Hours Worked: 20

<u>Changes from checkpoint McDiver:</u>

- maxScram()
  - changed name from maxScramGreedy() to maxScram()
    - Deleted old maxScram() method and it's helper method findBestPath()
  - Fixed bug where McDiver would not move if there were no coins in the maze
  - Implemented heap and used heap to make a priority queue for maxScram()
  - Added comments and specifications to maxScram and its helper method pathToCoin()
- optimizedScram()
  - Added a function optimizedScram() which optimizes maxScram for large mazes with 0.99 coin density → uses a dfs iteratively
  - Also added a function optimizedScram2() which should do the same thing as optimizedScram() but instead uses a dfs recursively
    - Data for part 3 were collected using optimizedScram()
- vanillaScram()
  - Changed name from basicScram() to vanillaScram()
  - Used a heap to implement dijsktra's algorithm used to find the path from start to exit

<u>Part 2a: Score Optimization</u>

Handout Seeds Configuration: default (MIN_ROWS = 8, MAX_ROWS = 25, MIN_COLS = 12, MAX_COLS = 40, CoinDensity = 0.6)	Vanilla Scram	Ref solution	Max Scram
-280019746129361794	Coins collected: 2,475 Bonus multiplier: 1.3 Score: 3,217	Coins collected: 8,156 Bonus multiplier: 1.3 Score: 10,602	Coins collected: 8,156 Bonus multiplier: 1.3 Score: 10,602

1908492650781828577	Coins collected: 1169 Bonus multiplier: 1.28 Score: 1,500	Coins collected: 10,233 Bonus multiplier: 1.28 Score: 13,139	Coins collected: 10,233 Bonus multiplier: 1.28 Score: 13,139
-3026730162232494481	Coins collected: 0 Bonus multiplier: 1.13 Score: 0	Coins collected: 0 Bonus multiplier: 1.13 Score: 0	Coins collected: 0 Bonus multiplier: 1.13 Score: 0
-4004310660161599891	Coins collected: 395 Bonus multiplier: 1.3 Score: 513	Coins collected: 395 Bonus multiplier: 1.3 Score: 513	Coins collected: 395 Bonus multiplier: 1.3 Score: 513
8035820871068432943	Coins collected: 369 Bonus multiplier: 1.04 Score: 384	Coins collected: 1,239 Bonus multiplier: 1.04 Score: 1,290	Coins collected: 369 Bonus multiplier: 1.04 Score: 384
2805343804353418701	Coins collected : 0 Bonus multiplier: 1.09 Score: 0	Coins collected: 0 Bonus multiplier: 1.09 Score: 0	Coins collected: 0 Bonus multiplier: 1.09 Score: 0

Additional Seeds (Not from the handout, same configuration)	Vanilla Scram	Ref solution	Max Scram
-5914125006077862352	Coins collected: 2473 Bonus multiplier: 1.28 Score: 3159	Coins collected: 22284 Bonus multiplier: 1.28 Score: 28474	Coins collected: 21335 Bonus multiplier: 1.28 Score: 27261
6073158090997163953	Coins collected: 4421 Bonus multiplier: 1 Score: 4421	Coins collected: 37240 Bonus multiplier: 1 Score: 37240	Coins collected: 36732 Bonus multiplier: 1 Score: 36732
5423013533187951654	Coins collected: 3974 Bonus multiplier: 1.26 Score: 5012	Coins collected: 33157 Bonus multiplier: 1.26 Score: 41824	Coins collected: 32620 Bonus multiplier: 1.26 Score: 41146
-6968229237334723624	Coins collected: 660 Bonus multiplier: 1.3 Score: 858	Coins collected: 8148 Bonus multiplier: 1.3 Score: 10592	Coins collected: 5148 Bonus multiplier : 1.3 Score: 6692
-7988817235456776368	Coins collected: 4334	Coins collected: 15622	Coins collected : 15622

	Bonus multiplier: 1.29 Score: 5598	Bonus multiplier: 1.29 Score: 20178	Bonus multiplier: 1.29 Score: 20178

For some of the seeds given in the handout, all three methods gave the same score. This is because, for some of these seeds, the maze doesn't have any coins or has very little coins, so vanilla scram would score the same amount of points as maxScram and the reference solution (eg. seeds -4004310660161599891 and 2805343804353418701). For the additional seeds, our Max Scram performed consistently better than our Vanilla Scram in terms of score and coins collected. This is because, for these seeds, the mazes are larger and have much more coins than the given seeds in the handout. Thus, our maxScram has much more opportunities to collect coins before going to exit (See explanation below on how our maxScram works). Thus, we can conclude that for larger mazes with more coins, maxScram will score much higher than vanilla scram, since, unlike maxScram, vanilla scram does not take into account the coin's value or distances. It is worth noting that for some smaller/medium size mazes (eg. -7988817235456776368), our maxScram scores the same as the instructor solution, but for others (-6968229237334723624), the instructor solution is performs better. This may be because unlike in the reference solution, in maxScram, when McDiver goes to the exit, he ignores picking up coins, leading to a loss in the score. This slight loss in coins may also explain why for larger mazes (like the first 3 additional seeds), the instructor solution also performs slightly better than maxScram(), as the McDiver in the reference solution does pick up coins while he is going to the exit.

In our maxScram, we optimized McDiver to score the most amount of points possible before going to the exit only when he's right about to run out of steps. McDiver collects coins by looking at every single tile in the maze and using Dijkstra's to calculate the shortest path from his current location to that tile. These paths are put in a priority queue implemented by a maxHeap where the priorities are calculated by the tile's coin value divided by the distance from McDiver to the tile. Thus, the coins of the highest values that are the closest to McDiver will be prioritized over coins that are worth less and are further away. After putting each of these paths in the priority queue, McDiver will choose the best path that's stored in the priority queue and walk along that path. Once McDiver is done walking that path, he will repeat this process by looking at all the tiles in the maze once again, ordering the paths from his current location to each tile in a priority queue, and walking along that path. While McDiver is walking along any path, before McDiver moves to any tile, the program checks whether he has enough remaining steps to go to that tile and take the shortest path from that tile to the exit. If so, the program will continue and McDiver will go to that tile. If not, McDiver will take the shortest path to the exit from his current tile. This ensures that McDiver only goes to the exit when he has no more remaining steps to collect any additional coins.

Part 2b: Runtime Optimization

Configuration (--no graphics) MIN_COL: 800 MAX_COL: 800 MIN_ROW: 800 MAX_ROW: 800 Coin Density: 0.6	SlowPQueue (Vanilla Scram)	Heap (Vanilla Scram)
Seed: -2800197461293617940	12.152 seconds	10.503 seconds
Seed: 1908492650781828577	13.546 seconds	9.83 seconds
Seed: -3026730162232494481	22.249 seconds	21.088 seconds
Seed: -4004310660161599891	10.59 seconds	9.461 seconds
Seed: 8035820871068432943	9.257 seconds	8.203 seconds
Seed: 2805343804353418701	15.639 seconds	13.121 seconds

For each seed, using a heap implementation of Dijkstra's algorithm for Vanilla Scram led to faster runtimes. This is because the slow priority queue implementation used an ArrayList to store the values and performed a linear search to implement add(), extractMin(), and changePriority(). Since the slow priority queue used linear search for these methods, their worst-case time complexity would be O(N) where N is the size of the priority queue. Our heap implementation of Dijkstra's algorithm uses a heap to maintain a priority queue rather than an ArrayList. Unlike the ArrayList implementation of add(), extractMin(), and changePriority(), the heap implementation of these methods has a worst-case time complexity of O(logN). This is because, when rearranging the values of the heap, the values have to be bubbled down or up at most the height of the tree which is logN, as a heap is a binary tree. For smaller graphs (such as the ones used in part a), the heap and slow priority queue implementations has roughly the same runtime (around 0.1 seconds apart). This is because for smaller graphs, less elements would need to be put in the priority queue for Dijkstra's, which decreases the overall search space, making the O(n) runtime of the slow priority queue roughly the same as the O(logN) runtime of the heap. This difference in runtime is only made more apparent for larger graphs (800x800), since the number of elements that need to be put in the priority queue would be much greater, and thus the search space would be much bigger. For larger priority queues, the heap would be consistently faster (as shown by the table), since the O(logN) work to do methods such as add() or changePriority(), would be significantly less than the O(N) work to do these methods using the slow priority queue.

Part 2c: Runtime Optimization for Scam (Optimization for 100x100, 0.99 Coin Density)

Configuration (--no graphics) MIN_COL: 100 MAX_COL: 100 MIN_ROW: 100 MAX_ROW: 100 Coin Density: 0.99	Optimized Scram	Vanilla Scram	Max Scram
-280019746129361794	Coins collected: 668,458 Bonus multiplier: 1.06 Score: 706,153 Time: 8.883 seconds	Coins collected: 40,738 Bonus multiplier: 1.06 Score: 43,035 Time: 0.763 seconds	Coins collected: 851,829 Bonus multiplier: 1.06 Score: 899,864 Time: 62.825 seconds
1908492650781828577	Coins collected: 674,660 Bonus multiplier: 1.17 Score: 787,103 Time: 7.918 seconds	Coins collected: 28,003 Bonus multiplier: 1.17 Score: 32,670 Time: 0.67 seconds	Coins collected: 915,354 Bonus multiplier: 1.17 Score: 1,067,913 Time: 66.309 seconds
-3026730162232494481	Coins collected: 647,487 Bonus multiplier: 1 Score: 647,487 Time: 8.845 seconds	Coins collected: 24,222 Bonus multiplier: 1 Score: 24,222 Time: 0.772 seconds	Coins collected: 847,820 Bonus multiplier: 1 Score: 847,820 Time: 54.603 seconds

-4004310660161599891	Coins collected: 681,987 Bonus multiplier: 1 Score: 681,987 Time: 9.114 seconds	Coins collected: 85,049 Bonus multiplier: 1 Score: 85,049 Time: 0.846 seconds	Coins collected: 889,866 Bonus multiplier: 1 Score: 889,866 Time: 59.955 seconds
8035820871068432943	Coins collected: 623,160 Bonus multiplier: 1 Score: 623,160 Time: 9.059 seconds	Coins collected: 21,933 Bonus multiplier: 1 Score: 21,933 Time: 0.774 seconds	Coins collected: 882,198 Bonus multiplier: 1 Score: 882,198 Time: 64.512 seconds
2805343804353418701	Coins collected : 682,055 Bonus multiplier: 1 Score: 682,055 Time: 9.217 seconds	Coins collected: 25,563 Bonus multiplier: 1 Score: 25,563 Time: 0.712 seconds	Coins collected: 924,315 Bonus multiplier: 1 Score: 924,315 Time: 64.689 seconds

For each graph, we chose the dimensions to be (100x100) since larger graphs would timeout when using maxScram(). For each of the seeds given in the handout, our optimizedScram() had a faster runtime than maxScram() while collecting more coins than vanillaScram(). We optimized maxScram() in optimizedScram() by moving McDiver randomly throughout the maze before walking him to the exit right when he's about to run out of steps. We kept the same logic of maxScram() where we only move to the exit after collecting all the coins possible by calling dijsktra's algorithm each step McDiver takes. However, unlike maxScram(), optimizedScram() did not use a priority queue to find the best path to a coin and instead only walks McDiver to a neighboring tile using a depth-first-search algorithm similar to seek(). However, unlike seek(), the dfs used for optimizedScram() used a stack rather than recursion in order to prevent any stack overflow errors for larger mazes. OptimizedScram() did not use a priority queue like maxScram() because in order to generate the priority queue, maxScram() needs to call dijisktra's to calculate the shortest path from McDiver to every coin on the maze, which decreases the runtime efficiency as the maze grows bigger. Thus, optimizedScram() should run faster since it only calls dijsktra's once every step (to check if he needs to go to the exit), while maxScram() does it twice (once to find the shortest paths from McDiver to every coin on the maze and once to check if he needs to go to the exit). However, this implementation of optimizedScram() did come at a cost of score when compared to maxScram() since unlike maxScram(), optimizedScram() does not take into consideration the value of the coins in the maze nor their distance from McDiver. Even though optimized Scram scored less than maxScram(), it still scored significantly more than Vanilla Scram(), as it forces McDiver uses all of his allotted steps to collect coins rather than only small portion like in Vanilla Scram.