***assignment1\_p6:***

author = jim027@ucsd.edu, A99075314,

siz001@ucsd.edu, A99076798,

yuc036@ucsd.edu, A91112915

**Problem Description:**

Problem 1:

This problem is to find the shortest possible (optimal) path by being given an initial prime number and a final prime number, and to print a sequence of intermediate prime numbers which are generated by changing one digit at a time. However, if the puzzle is not solvable, we need to print UNSOLVABLE.

Here, we use the breadth-first search algorithm to solve this problem.

The data structure used in the breadth-first algorithm is queue.

Problem 2:

This problem is to find the shortest possible (optimal) path by being given an initial prime number and a final prime number, and to print a sequence of intermediate prime numbers which are generated by changing one digit at a time. However, if the puzzle is not solvable, we need to print UNSOLVABLE.

Here, we use the depth-first search algorithm to solve this problem.

The data structure used in the depth-first algorithm is stack.

Problem 3:

This problem is to find the shortest possible (optimal) path by being given an initial prime number and a final prime number, and to print a sequence of intermediate prime numbers which are generated by changing one digit at a time. However, if the puzzle is not solvable, we need to print UNSOLVABLE. This problem we use absolute depth limit of 8, to make sure the code will run in reasonable time even for unsolvable cases.

Here, we use the iterative deepening depth-first search algorithm to solve this problem.

The data structure used in the the iterative deepening depth-first search algorithm is stack.

Problem 4:

This problem is to find the shortest possible path (optimal) consists of a sequence of intermediate prime numbers which are generated by changing one digit at a time by being given an initial prime number and a final prime number, and to print the path followed from both start and end element till the middle element in separate lines. However, if the puzzle is not solvable, we need to print UNSOLVABLE.

Here, we use the bidirectional search algorithm to solve this problem.

The data structure used in the bidirectional search algorithm is dictionary.

Problem 5:

This problem is to find the shortest possible (optimal) by being given an initial prime number and a final prime number, and to print a sequence of intermediate prime numbers which are generated by changing one digit at a time. However, if the puzzle is not solvable, we need to print UNSOLVABLE.

Here, we use the A\* algorithm with hamming distance as the heuristic for 5(a), and for 5(b), we use our own heuristic algorithm that to get the distance from the first prime to goal by calculating the number of different digits they have, and the heuristic length of its children are minus 1 if parent's length becomes 0 but still not same as the goal, the child has distance as hamming distance with the goal. Otherwise, it will automatically discriminate by 1 from parent's heuristic length since hamming distance is consistent and admirable, the length less that hamming distance is also admirable and consistent.

The data structure used in the both algorithms are priority queue.

**Analysis on the efficacy of the different algorithms with different puzzles:**

Test Inputs1:

Test 1: 2 2

Test 2: 2 3

Test 3: 103 199

Test 4: 103 269

Test 5: 103 3221

Test 6: 3221 3319

Test 7: 100003 190633

number of nodes visited:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Algorithm/Test** | Test 1 | Test 2 | Test 3 | Test 4 | Test 5 | Test 6 | Test 7 |
| BFS | 0 | 1 | 2 | 15 | 143 | 45 | 79 |
| DFS | 0 | 9 | 43 | 32 | 5022 | 134 | 5950 |
| iterative deepening DFS | 0 | 1 | 1 | 1 | 0 | 1 | 1 |
| bidirectional search | 0 | 5 | 10 | 53 | 879 | 62 | 77 |
| A\* | 0 | 1 | 2 | 4 | 0 | 3 | 11 |
| own heuristics | 0 | 1 | 2 | 15 | 0 | 38 | 205 |

maximum size of the queue:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Algorithm/Test** | Test 1 | Test 2 | Test 3 | Test 4 | Test 5 | Test 6 | Test 7 |
| BFS | 0 | 1 | 8 | 41 | 73 | 140 | 385 |
| DFS | 0 | 7 | 24 | 24 | 31 | 34 | 38 |
| iterative deepening DFS | 0 | 4 | 15 | 23 | 57 | 27 | 1 |
| bidirectional search | 1 | 5 | 10 | 36 | 478 | 43 | 55 |
| A\* | 0 | 1 | 13 | 20 | 0 | 19 | 58 |
| own heuristics | 0 | 1 | 13 | 47 | 0 | 152 | 1025 |

path length given by algorithm:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Algorithm/Test** | Test 1 | Test 2 | Test 3 | Test 4 | Test 5 | Test 6 | Test 7 |
| BFS | 1 | 2 | 3 | 4 | 0 | 4 | 5 |
| DFS | 1 | 4 | 5 | 5 | 0 | 5 | 5 |
| iterative deepening DFS | 1 | 2 | 3 | 4 | 0 | 4 | 5 |
| bidirectional search | 1 | 2 | 3 | 4 | 0 | 4 | 5 |
| A\* | 1 | 2 | 3 | 4 | 0 | 4 | 5 |
| own heuristics | 1 | 2 | 3 | 5 | 0 | 6 | 7 |

Time (microseconds) it took to run the code:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Algorithm/Test** | Test 1 | Test 2 | Test 3 | Test 4 | Test 5 | Test 6 | Test 7 |
| BFS | 274 | 37 | 211 | 761 | 6591 | 4633 | 44316 |
| DFS | 214 | 83 | 378 | 347 | 37483 | 1828 | 351346 |
| iterative deepening DFS | 226 | 71 | 391 | 4026 | 154022 | 2449 | 423518 |
| bidirectional search | 284 | 98 | 316 | 1072 | 96054 | 1340 | 7361 |
| A\* | 68 | 184 | 334 | 541 | 32 | 555 | 5500 |
| own heuristics | 62 | 134 | 359 | 1272 | 15 | 7333 | 141328 |

Total number of nodes:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Algorithm/Test** | Test 1 | Test 2 | Test 3 | Test 4 | Test 5 | Test 6 | Test 7 |
| BFS | 1 | 1 | 9 | 54 | 142 | 184 | 463 |
| DFS | 1 | 9 | 43 | 32 | 5022 | 134 | 5950 |
| iterative deepening DFS | 1 | 4 | 45 | 628 | 2833864 | 198 | 6730 |
| bidirectional search | 2 | 6 | 19 | 50 | 1143 | 63 | 104 |
| A\* | 1 | 2 | 15 | 24 | 1 | 22 | 69 |
| own heuristics | 1 | 2 | 15 | 62 | 1 | 190 | 1230 |

Big O Time Complexity:

|  |  |
| --- | --- |
| BFS | O(bd) |
| DFS | O(bl) |
| iterative deepening DFS | O(bd) |
| bidirectional search | O(bd/2) |
| A\* | O(bd) |
| own heuristics | O(bd) |

(d be the solution depth, and l be the limit)

Test Inputs2:

Test 1: 2 3

Test 2: 13 29

Test 3: 103 269

Test 4: 3221 3319

Test 5: 14107 32057

Test 6: 100003 190633

Time (microseconds) it took to run the code:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Algorithm/Digit** | 1 Digit | 2 Digit | 3 Digit | 4 Digit | 5 Digit | 6 Digit |
| BFS | 288 | 286 | 797 | 4558 | 345928 | 38243 |
| DFS | 335 | 135 | 390 | 1826 | 160393 | 373151 |
| iterative deepening DFS | 591 | 247 | 5688 | 2771 | 515773 | 822462 |
| bidirectional search | 328 | 188 | 724 | 1340 | 31077 | 18501 |
| A\* | 234 | 350 | 1381 | 1283 | 30464 | 7531 |
| own heuristics | 210 | 293 | 1543 | 5183 | 97626 | 129085 |

Efficiency:

From the comparison of the graphs of time taken to run the code, we can found that the A\* algorithm with hamming distance as the heuristic is the most efficient as the tendency of it is almost linear when the number of digits of input is increasing.

For the other algorithms, the efficiency will be placed in the order from the most efficient to the least efficient as: bidirectional > BFS > DFS > iterative deepening DFS. Their graph is growing like an exponential function as well, which is fitted with its big O time complexity.

To conclude, the A\* is the most efficient as it is the fasted and could output the most optimal path.

然后那个要比较visited nodes和total nodes我没写==

analysis on the efficacy of the different algorithms with different puzzles

measure the number of nodes visited / maximum size of the queue / time it took to run the code/ path length given by algorithm etc., for each case

How do they compare against what you expect from the Big-O analysis? How do they compare against each other?

Results of the analysis and a short discussion. It should include at least one graph with proper labels that shows how the quantity you measured changes with what you varied.

**Author Contribution:**

Jing Ma: I focused on the algorithm part of this assignment, especially the first three parts. And I spent a lot of time to improve the efficiency. At the first, I used list instead of set to implement the visited, and every time it takes O(n) to confirm if an element is in the list or not, so it’s very slow. Also, for BFS (p1), I added another set called added to make sure one value will be only added to the queue once, in this way the run time will be reduced.

Siyao Zheng:

After learning all the algorithms, I decided the data structure and figured out the relatively optimal solution and the differences when coding different problems. I take the most responsibility in problem 4 and 5's coding, testing, optimization. I also take a little part in analyzing test results. I found its so different for different algorithms to do the same thing, which makes machine to think very differently but still we do bot come up a most optimal solution both for time and shortest path. I see the potential improvement scientists have to figure out.

Yunxin Chen:

I watched and read through tutorials about Python, and learning different algorithms from reading books. After we discussed to get a general algorithm design, I debugged after the code was firstly written. I take the most responsibility in debugging, test design, testing and optimization. I also did the record and analysis of the testing data. After that, I wrote the report to show the combined result of our assignment.