CSE 150 Homework Assignment 1

Part 6 Report

Lingyi Kong	l1kong@ucsd.edu	A97010449
Utkrisht Chennanchetty Ra	urajkuma@ucsd.edu	A91060509
Yinji Lu	yil261@ucsd.edu	A91085115

Problem 1:

- Algorithm: Breadth First Search
- Implementation: Using a queue(FIFO) data structure to perform breadth first search. Using an array that will keep track of discovery flag as well as depth. Take each digit of the first argument, and find all the primes that can be generated by altering just one digit. Do the same for all the digits. This is the first layer. Then, take each of the newly generated primes (pop them off the queue), and expand them again by generating all the primes that can be generated by changing one digit at a time. Keep traversing all the layers until the goal node is reached.
- Completeness: Yes
- Optimality: Yes, given identical step costs.
- Space Complexity: Since each node is expanded, there will be nodes expanded.
- Time Complexity:

Problem 2:

- Algorithm: Depth Limited Search
- Implementation: Using stack (FILO) data structure to perform depth search. Using a dictionary to keep track of discovery flag as well as depth. In this implementation, our depth limit is 5 (with root having depth of 0). Thus our code will terminate one search path if the depth of current path is more than 5
- Completeness: In general no, yes if the length of shortest path is no greater than depth limit

- Optimality: In general no, it will only return an optimal path if the length of shortest path is exactly the same as the depth limit
- Space Complexity: This is essentially depth first search, so space complexity should be fairly small—O(bl)
- Time Complexity: since the max depth is limited, the max number of nodes visited are also capped—O(b^l)

Problem 3:

- Algorithm: Iterative Deepening Depth First Search
- Implementation: Basic data structure same as Depth Limited Search, except we use a for loop to run it over different max depth
- Completeness: If length of shortest path is smaller than max depth, then yes
- Optimality: whenever it finds a path, it's optimal
- Space Complexity: For each iteration, its space complexity is the same as depth limited search, thus space complexity is small—O(bl)
- Time Complexity: It's a constant proportion to depth limited search, thus max number of nodes visited are also capped—O(b^l)

Problem 4:

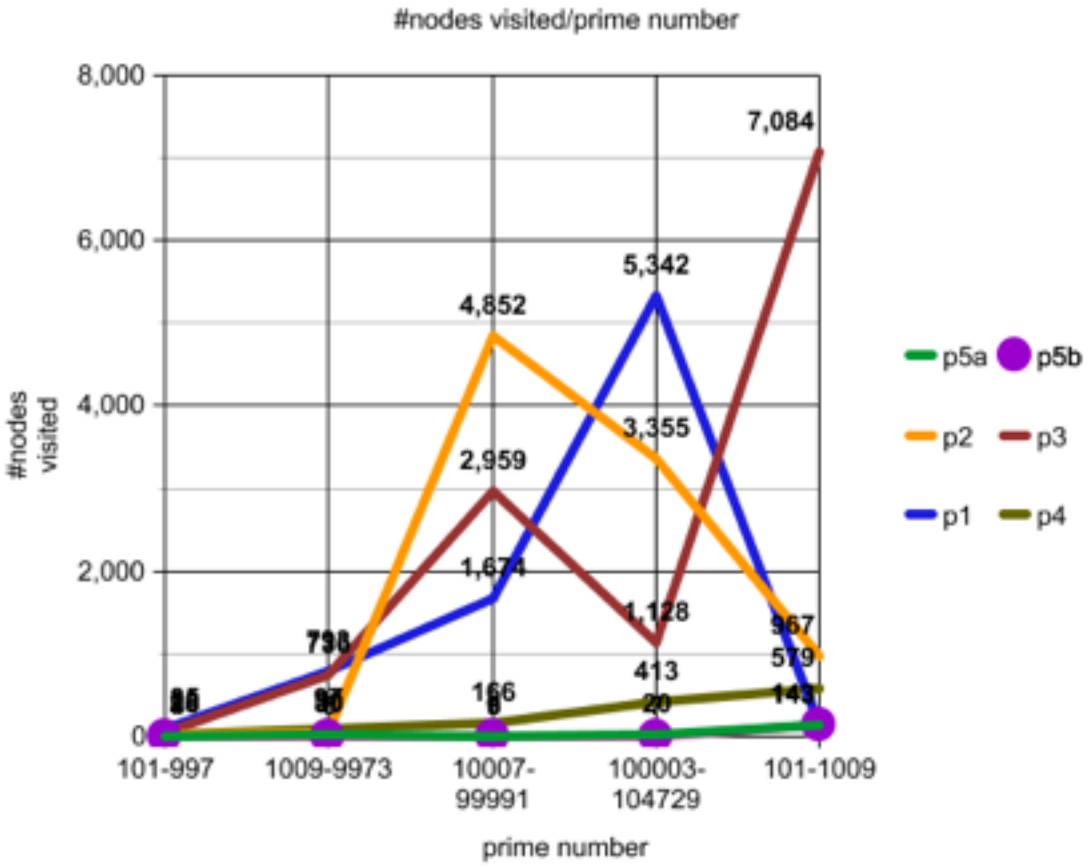
- Algorithm: Bi-directional search
- Implementation: Used 2 queues(FIFO) and 2 sets to perform bi-directional search. We are essentially running two bread first searches simultaneously, hence the two queues. Using 2 arrays that will keep track of discovery flag as well as distances. We will iterate from the initial state to goal state AND from goal state to initial state. We are using two set data structures to store all the visited nodes from each of the two bfs. If a popped node from one bfs is found in the set of the other bfs, then we have completed a connection from the initial state to goal state, this will be the middle node.
- Completeness: Yes
- Optimality: Yes, we have identical step costs and both directions use bfs
- Space Complexity: Space is O(b^(d/2)) because it will start searching from both side
- Time Complexity: Time is O(b^(d/2)) because it will start searching from both side

Problem 5a:

- Algorithm: A* Search
- Implementation: Using priority queue data structure to perform search. It uses f(n) = g(n) + h(n) where g(n) is the cost to reach the prime and h(n) is a function that measures the hamming distance between the current prime and the final prime. It always is using a dictionary to keep track of where does the prime come from as well as the cost so far to reach the prime.
- Completeness: In general, yes. It would always reach the final prime.
- Optimality: In general, yes. It would always be an optimal path
- Space Complexity: O(b^m): Potentially keeps all nodes in memory
- Time Complexity: Exponential

Problem 5b:

- Algorithm: Alternative A* Search
- Implementation: Using priority queue data structure to perform search. It uses f(n) = g(n) + 1.5h(n) where g(n) is the cost to reach the prime and h(n) is a function that measures the hamming distance between the current prime and the final prime. It always is using a dictionary to keep track of where does the prime come from as well as the cost so far to reach the prime. The reason why I added 1.5 is because we need to reduce the amount of time we restart exploring. In another word, once we start on a route we gotta go to the end and see if there is light instead of quitting in the middle of the road and start on another similar path. So since f(n) = h(n) + g(n) and g(n) mostly increases 1 at a time. f(n) is playing the main roles here. However, like I stated above, I would want to explore more on the paths I have gone along rather than restart on a similar path. This means I will try to give g(n) less weights so that when the path that is closer to the goal is being explored we will not lose them because its g(n) is high...and instead I give h(n) higher weights so that will differentiate those similar paths so that we will always stay on the track of the longest path so far.
- Completeness: In general, yes. It would always reach the final prime.
- Optimality: In general, yes but sometimes it might not be optimal because it might overestimate the distance between two prime numbers and result in a longer path. For example, if the actual distance between two primes are their hamming distance.
- Space Complexity: O(b^m): Potentially keeps all nodes in memory



- Time Complexity: Exponential

Data Analysis:

To test in a short but extensive way, our team chooses to test each algorithm on groups of prime with distinct number of digits. We choose the smallest and the largest prime number in each number group and run our algorithm on them. As it can be seen from the data collected above in the graph, A* search or the enhanced A* search are significantly faster than all other search algorithm. This is because A* search tends to go in a straight path as it is informed search unlike other uninformed search where the search algorithm will simply blindly explore its neighbors until it finds the target. Their search speed which is defined by the number of nodes visited vary at different range and this is because sometimes path length is not restricted by the number of digits in the primes. However, generally speaking, the more number of digits are in two primes, the longer the path between the two primes will be. Iterative deepening is significantly more slower when exploring the entire graph is needed especially when there is no path between two given numbers. This is because it will have to iterate through a lot of nodes that are not the target for numerous times. However, we also observe that most of the time, the actual runtime is not as bad as we predict with big O analysis. Our reasoning is that between two given prime, there should always be some paths that will help terminate the search before it has to go through all of the prime in the range of its number of digits.

Work contributed by each member:

Lingyi – Pair programmed with the other members on all 5 problems. Debugged the code extensively, made it more efficient, helped on analysis

Utkrisht – Pair programmed with the other members on all 5 problems, helped with analysis

Yinji – Pair programmed with the other members on all 5 problems, helped with analysis