CSE 150 Programming Assignment 1 Report

Group member:
Huicheng Zhuang A99075768
There used to be another member, but he dropped.

1.Problem Description and Algorithm Used:

The problem is finding the shortest prime path between two prime numbers. A prime path is a path consists only prime numbers and each of the prime numbers results from changing only one digit from the previous one.

Problem 1

Breadth First Search

```
Procedure Breadth First Search (S, G)
Begin:
    push S to queue
    While (queue is not empty)
            tmpNode  = queue.get()
            If tmpNode == G
                    Return tmpNode
            tmpList = getPossibleAction(tmpNode)
            For (x in tmpList):
                    If x == G:
                            Return x
                    If x not in closed list:
                            Add x to closed list
                            Push x to queue
    print("UNSOLVABLE")
End
```

Problem 2

```
Depth Limited Search (S, G)
Begin:
    tmpDepth = 0
    push S to stack
    While (queue is not empty)
            tmpNode  = stack.pop()
            tmpDepth = tmpNode.depth
            If tmpNode == G
                    Return tmpNode
            If tmpDepth == 5 : continue to next loop
            tmpList = getPossibleAction(tmpNode)
            For (x in tmpList):
                    If x == G:
                            Return x
                    Push x to stack
    print("UNSOLVABLE")
End
```

Problem 3

```
Iterative Deepening Depth-First Search (S, G)
Begin:
    tmpDepth = 0
    tmpDepthLimit = 0
    For depth 0 to 8
            New stack
            push S to stack
            While (queue is not empty)
                    tmpNode  = stack.pop()
                    tmpDepth = tmpNode.depth
                    If tmpNode == G
                            Return tmpNode
                    If tmpDepth == 5 : continue to next loop
                    tmpList = getPossibleAction(tmpNode)
                    For (x in tmpList):
                            If x == G:
                                    Return x
                            Push x to stack
    print("UNSOLVABLE")
End
```

Problem 4

```
uses Bidirectional Search
  Procedure Bidirectional Search (S, G)
  Begin:
      push S to queue_start
     push G to queue_target
      While (both of the queues are not empty)
            tmpStart = queue_start.get()
            tmpTarget = queue_target.get()
            tmpList_start = getPossibleAction(tmpStart)
            tmpList_target = getPossibleAction(tmpTarget)
            For (x in tmpList_start):
                  If x in fring of queue_target:
                        Return x
                  If x not in closed list:
                        Add x to closed list
                        Push x to queue
            For (x in tmpList_target):
                  If x in fring of queue_start:
                        Return x
                  If x not in closed list:
                        Add x to closed list
                        Push x to queue
      print("UNSOLVABLE")
  End
```

Problem 5

```
uses A* search
  A* Search (S, G)
  Begin:
      push S to priority_queue pq
      While (pq is not empty)
            tmpNode  = queue.top() (That is the node with lowest hamming distance)
            If tmpNode == G
                  Return tmpNode
            tmpList = getPossibleAction(tmpNode)
            For (x in tmpList):
                  If x == G:
                        Return x
                  If x not in closed list:
                        Add x to closed list
                        Push x to queue
      print("UNSOLVABLE")
  End
```

2. Data Structures Used:

Problem 1 uses a queue
Problem 2 uses a stack
Problem 3 uses a stack as well
Problem 4 uses a queue since I implemented the bidirectional as two bfs
Problem 5 uses a priority queue

3.Performance Analysis:

Test pair 1: (2,7)

Algorithm	Time	Nodes visited	Path length	Solvable(y/n)
BFS	0.00008	1	2	y
DFS	0.00004	1	2	y
IDFS	0.00004	2	2	y
Bidirectional	0.00013	2	2	y
A*	0.00009	1	2	y

Test pair 2:(23, 37)

Algorithm	Time	Nodes visited	Path length	Solvable(y/n)
BFS	0.00029	10	4	y
DFS	0.00035	10	6	y
IDFS	0.00087	40	4	y
Bidirectional	0.00026	6	4	y
A*	0.00024	6	4	y

Test pair 3:(103,269)

Algorithm	Time	Nodes visited	Path length	Solvable(y/n)
BFS	0.00076	26	4	y
DFS	0.00124	54	6	y
IDFS	0.00435	199	4	y
Bidirectional	0.00034	10	4	y
A*	0.00038	10	4	y

Test pair 4: (1051, 2237)

Algorithm	Time	Nodes visited	Path length	Solvable(y/n)
BFS	0.01076	217	5	y
DFS	0.00378	147	6	y
IDFS	0.01890	778	5	y
Bidirectional	0.00057	14	5	y
A*	0.00047	14	5	y

Compare and Contrast:

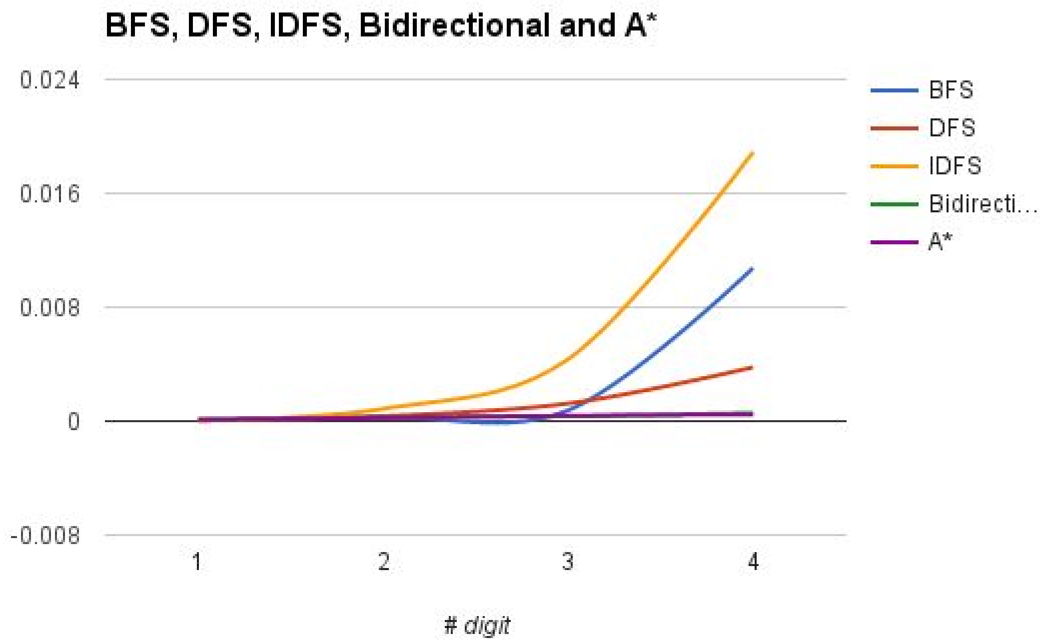
Due to the big O time complexity of these searches.

BFS has O(b^d), Depth-limited has O(b^l) and Iterative Depth-limited has O(b^d) where d is the depth of solution and l is the depth limit. Since the problem we are trying to solve here does not need huge amount of operations, and the actual terminating depth of search is sometimes even smaller than the given depth limit, so the performance difference between them is not so obvious which seems to be reasonable.

Bidirectional Search, on the other hand, performs way better than the first three algorithms and that is also as we expected since bidirectional has time complexity of O(b^d/2)

A* search , as well as bidirectional, performs way better than the first three and for most of the cases, it outperforms bidirectional and I think that's because the hamming distance heuristic did a great job on simplifying the problem and reducing the number of work the search has to do.

4. Analysis discussion



The chart reflexes the time taken to solve test pairs of different number of digits. We cannot see the line for bidirectional because it is just too close to that of A*.

(I don't know why one of the lines goes below 0 , I tried to fix it but failed, I am just not good at google excel)

We can easily conclude that Bidirectional and A* performs a lot better than other 3, and for the other 3 algorithms, it looks like the iterative deepening search has a much higher growth rate than BFS and DFS ,and that might be due to my implementation. Theoretically, these three have almost the same performance.

For optimality (this is not shown on the graph) we can see that BFS, Iterative-deepening , bidirectional and A* all give optimal solutions to the problem and Depth-limited search may not always generate optimal result. This discovery is consistent with the definitions of all the algorithms.

5. Group contribution

My partner dropped the class last Friday, and I asked professor if I could go on finish this PA by myself and he said yes. So I did everything.

(This is not the last page)

6.My heuristic

I made a function for calculating the number of prime numbers that can be generated by modifying one digit from current number. Let’s denote it N. Then in the priority queue, if two nodes have exactly the same value of f(n), I break the ties by comparing their N, where the number having greater N will be given higher priority. I think this may improve the perform of the algorithm because the more prime numbers that can be made from the number, the more likely the number will lead to a solution.

My heuristic seems to be optimal based on my own test results but I actually cannot prove it theoretically since there might be some special cases.

The performance of my heuristic , in terms of time, is a little worse than the original heuristic, which only uses the hamming distance. And that’s because calculating N takes some time.

But in terms of nodes visited, my heuristic turned out to be 20% more efficient than the original based on the fact that it generally visites 20% less nodes.