**CS 599: Algorithms** **(Due: November 3rd, 2020)**

# Submission Homework 5

*Instructor:* Prof. Prasad Tadepalli *Name:* Rashmi Jadhav, *Student ID:* 934-069-574

## 1: Implementation of BFS to solve sliding tile 8-puzzle

The breadth first search algorithm starts by traversing from the goal state, maintains *seen_state_space* to keep track of visited nodes and uses BFS queue to keep track of level order traversal. We encode each state which is a 2D matrix in the form of string representation of its flattened 1D form. This helps in checking hash for whether the node was visited or not. We start popping nodes from BFS queue and try to find neighbor states of the popped state by checking 4 valid directions U, D, L, and R of the wherever empty spot 0 is present. All the valid neighbors are then pushed onto queue as immediate neighbors of current state, provided they were not already present in the state space. At each explored node, we also store string representation of the path that led from goal state to the node state. We keep searching new nodes level by level until the initial state is found. This is our solution. Throughout the recursion, if none of the states ever matched the initial state, we do not have a solution. Since we tracked path from goal state to the start state, we need to reverse it to get the actual solution.

(a) Problem 1:

```
init state [1, 2, 3, 4, 5, 6, 8, 7, 0] -> goal state [1, 2, 3, 8, 0, 4, 7, 6, 5]
  LEVELS| NODES AT LEVEL|STATE SPACE
       0|              1|          1
       1|              4|          5
       2|              8|         13
       3|              8|         21
       4|             16|         37
       5|             32|         69
       6|             60|        129
       7|             72|        201
       8|             55|        256
Solution: DRRULLDR, total nodes searched: 256, time taken: 0.010179996490478516 s
```

Problem 2:

```
init state [2, 4, 7, 1, 5, 3, 0, 8, 6] -> goal state [1, 2, 3, 8, 0, 4, 7, 6, 5]
  LEVELS| NODES AT LEVEL|STATE SPACE
       0|              1|          1
       1|              4|          5
       2|              8|         13
       3|              8|         21
       4|             16|         37
       5|             32|         69
       6|             60|        129
       7|             72|        201
       8|            136|        337
       9|            200|        537
      10|            376|        913
      11|            512|       1425
      12|            964|       2389
      13|           1296|       3685
      14|           2368|       6053
      15|           3084|       9137
      16|           5482|      14619
      17|           6736|      21355
      18|           9708|      31063
Solution: LDLDRULURDLDRRUULD, total nodes searched: 31063, time taken: 0.6672661304473877 s
```

Problem 3:

```
init state [0, 1, 6, 8, 4, 2, 5, 7, 3] -> goal state [1, 2, 3, 8, 0, 4, 7, 6, 5]
  LEVELS| NODES AT LEVEL|STATE SPACE
       0|              1|          1
       1|              4|          5
       2|              8|         13
       3|              8|         21
       4|             16|         37
       5|             32|         69
       6|             60|        129
       7|             72|        201
       8|            136|        337
       9|            200|        537
      10|            376|        913
      11|            512|       1425
      12|            964|       2389
      13|           1296|       3685
      14|           2368|       6053
      15|           3084|       9137
      16|           5482|      14619
      17|           6736|      21355
      18|          11110|      32465
Solution: UULDRDLLUURDDLUURD, total nodes searched: 32465, time taken: 0.6828031539916992 s
```

Problem 4:

```
init state [0, 5, 7, 6, 2, 8, 3, 4, 1] -> goal state [1, 2, 3, 8, 0, 4, 7, 6, 5]
  LEVELS| NODES AT LEVEL|STATE SPACE
       0|              1|          1
       1|              4|          5
       2|              8|         13
       3|              8|         21
       4|             16|         37
       5|             32|         69
       6|             60|        129
       7|             72|        201
       8|            136|        337
       9|            200|        537
      10|            376|        913
      11|            512|       1425
      12|            964|       2389
      13|           1296|       3685
      14|           2368|       6053
      15|           3084|       9137
      16|           5482|      14619
      17|           6736|      21355
      18|          11132|      32487
      19|          12208|      44695
      20|          18612|      63307
      21|          18444|      81751
      22|          24968|     106719
      23|          19632|     126351
      24|          22289|     148640
      25|          13600|     162240
      26|          11842|     174082
      27|           4340|     178422
      28|           2398|     180820
      29|            472|     181292
      30|            144|     181436
Solution: UULLDDRURDLUURDLLDRRULULDRURDL, total nodes searched: 181436, time taken: 6.093797922134399 s
```

Problem 5(No solution):

```
init state [1, 2, 3, 4, 5, 6, 7, 8, 0] -> goal state [1, 2, 3, 8, 0, 4, 7, 6, 5]
  LEVELS| NODES AT LEVEL|STATE SPACE
       0|              1|          1
       1|              4|          5
       2|              8|         13
       3|              8|         21
       4|             16|         37
       5|             32|         69
       6|             60|        129
       7|             72|        201
       8|            136|        337
       9|            200|        537
      10|            376|        913
      11|            512|       1425
      12|            964|       2389
      13|           1296|       3685
      14|           2368|       6053
      15|           3084|       9137
      16|           5482|      14619
      17|           6736|      21355
      18|          11132|      32487
      19|          12208|      44695
      20|          18612|      63307
      21|          18444|      81751
      22|          24968|     106719
      23|          19632|     126351
      24|          22289|     148640
      25|          13600|     162240
      26|          11842|     174082
      27|           4340|     178422
      28|           2398|     180820
      29|            472|     181292
      30|            148|     181440
Solution: No solution, total nodes searched: 181440, time taken: 6.146442174911499 s
```

(b) The hardest problem tried that was solved has start state as: $[0, 5, 7, 6, 2, 8, 3, 4, 1]$. It explored a total of **181436 nodes** and **31 levels** during recursive BFS calls. It took around **5.4 seconds** to run. The solution for the same is:
$U, U, L, L, D, D, R, U, R, D, L, U, U, R, D, L, L, D, R, R, U, L, U, L, D, R, U, R, D, L$ which consists of **30 moves**.

```
------------------------------------------------------------------------------------------------------
init state [1, 2, 3, 4, 5, 6, 8, 7, 0] -> goal state [1, 2, 3, 8, 0, 4, 7, 6, 5]
Solution: DRRULLDR, total nodes searched: 256, time taken: 0.012455940246582031 s
------------------------------------------------------------------------------------------------------
init state [2, 4, 7, 1, 5, 3, 0, 8, 6] -> goal state [1, 2, 3, 8, 0, 4, 7, 6, 5]
Solution: LDLDRULURDLDRRUULD, total nodes searched: 31063, time taken: 0.6474981307983398 s
------------------------------------------------------------------------------------------------------
init state [0, 1, 6, 8, 4, 2, 5, 7, 3] -> goal state [1, 2, 3, 8, 0, 4, 7, 6, 5]
Solution: UULDRDLLUURDDLUURD, total nodes searched: 32465, time taken: 0.6202759742736816 s
------------------------------------------------------------------------------------------------------
init state [0, 5, 7, 6, 2, 8, 3, 4, 1] -> goal state [1, 2, 3, 8, 0, 4, 7, 6, 5]
Solution: UULLDDRURDLUURDLLDRRULULDRURDL, total nodes searched: 181436, time taken: 5.3974058628082275 s
------------------------------------------------------------------------------------------------------
init state [1, 2, 3, 4, 5, 6, 7, 8, 0] -> goal state [1, 2, 3, 8, 0, 4, 7, 6, 5]
Solution: No solution, total nodes searched: 181440, time taken: 5.475801944732666 s
```

**2: Dijkstra analysis on negatively weighted edges**

Dijkstra's algorithm is a single source shortest path algorithm for graphs that greedily selects a vertex that has shortest distance from source at any point. Once a vertex is selected/relaxed, it is said to have optimal solution as per Dijkstra and thus it is not visited again. If the graph has all positive distances, we can claim that once a vertex is relaxed, we cannot find any better answer / shorter distance. However, if a graph has negative edges, the algorithm cannot be guaranteed to give the shortest path as it may mark a vertex visited only to realize later that we had a shorter path with negative edge that gave a better answer.
If were given that the only negative edges are those that leave the starting node $S$ and all other edges are positive, we will need to consider cases for different types of graphs.

1) Directed Acyclic Graph(DAG): Let's assume that source $S$ has $k$ edges $e_1, e_2, e_3, ..., e_k$ going out from it to $k$ different nodes $v_1, v_2, v_3, ..., v_k$. If we order these as per distances from smallest to largest as $v_1, v_2, v_3, ..., v_k$, for each $v_i$ and $v_j$ where $i < j$, there can't be a path from $S$ to $v_i$ through $v_j$ that has a better cost than $e_i$. We can say this because there's no cycle in the graph. Thus, the order of selecting the first vertex will never change. Since this selection order makes sure to choose best first vertex, there's no way for an algorithm to enter a later vertex that was already marked visited before shortest path was found. In short, once an edge is in the "visited" we will never find a "shorter" path to it, since the negative edges are only from the source. Thus, Dijkstra would hold for a DAG.

2) Directed Cyclic Graph: If a directed cyclic graph has a cycle with positive edge that doesn't involve $S$, Dijkstra would hold just like a DAG. Consider a graph where there's a cycle with negative edge that starts from $S$. Assume a small 2 vertex graph such that $e_1 : S \to v_1 = -2$ and $e_2 : v_1 \to S = 1$. This means there is no minimum-weight path and the algorithm will loop forever trying to update min distance in a cyclic manner. Thus, Dijkstra fails for a cyclic graph that has a negative edge starting at source and a negative edge even otherwise.

**3: Design for finding length of the shortest cycle in a graph**

To find the length of shortest cycle in a directed graph, we can use Dijkstra's algorithm to find a shortest cycle going through any given node and run it for all the nodes.

```
find_len_shortest_cycle(G):
    len_shortest_cycle = inf
    for u in vertices:
        shortest_dist_from_u[] = dijkstra(G, len_of_edges, u)
            for v in vertices:
                if (v, u) belongs to e:
                    len_shortest_cycle = min(len_shortest_cycle, shortest_dist_from_u[v] + dist(v, u)
    if len_shortest_cycle == inf:
        print "no cycles"
    else:
        retutn len_shortest_cycle
```

Since we have n nodes and we run Dijkstra for n nodes, the time complexity is $O(n * (n^2)) = O(n^3)$.

**4: All pair shortest path with restriction**

In order to find shortest paths between all pairs of vertices with a restriction that they all must pass through vertex $A$, we can divide the problem into 2 shortest-path problems. For any two vertices $u$ and $v$, since the shortest path must go through $A$, the shortest path is of the form $u \to A \to v$. First, we find shortest paths from vertex $A$ to all other vertices using Dijkstra, let's call it $dist_A(u)\ where\ u \in vertices$. Now, we reverse the graph and find the shortest path from all other vertices to $A$, let's call it $dist_A^R(v)\ where\ v \in vertices$. Due to symmetry in $G$ and $G^R$ and strongly connected property, shortest path from $u$ to $v$ is $dist_A(u) + dist_A^R(v)$ where $u, v \in vertices$. Once we have all distances from $A$ to *other vertices* in graph and from *other vertices* to $A$ in reversed graph, for all *vertices* x *vertices*, we need to add the corresponding calculated distances. The time complexity for running Dijkstra on graph and reverse graph would be $O((|V| + |E|).log\ |V|)$. The time complexity for adding all calculated distances for *vertices* x *vertices* would be $O(|V|^2)$. Thus the time complexity of the proposed algorithm is $O((|V| + |E|).log\ |V| + |V|^2)$