Problem Set 6

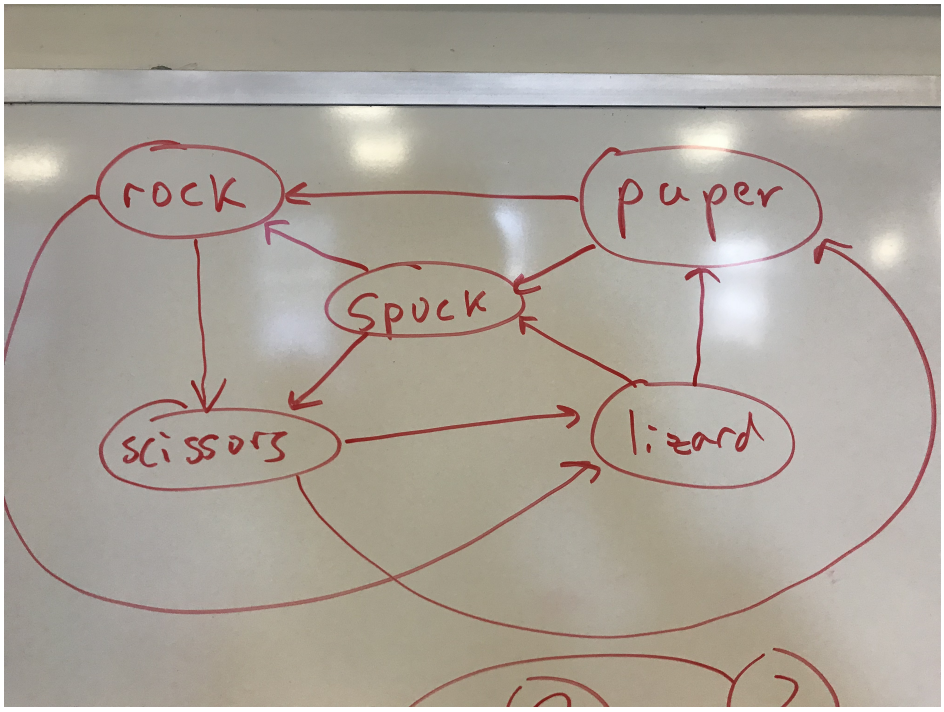
All parts are due Thursday, October 25 at 11PM.

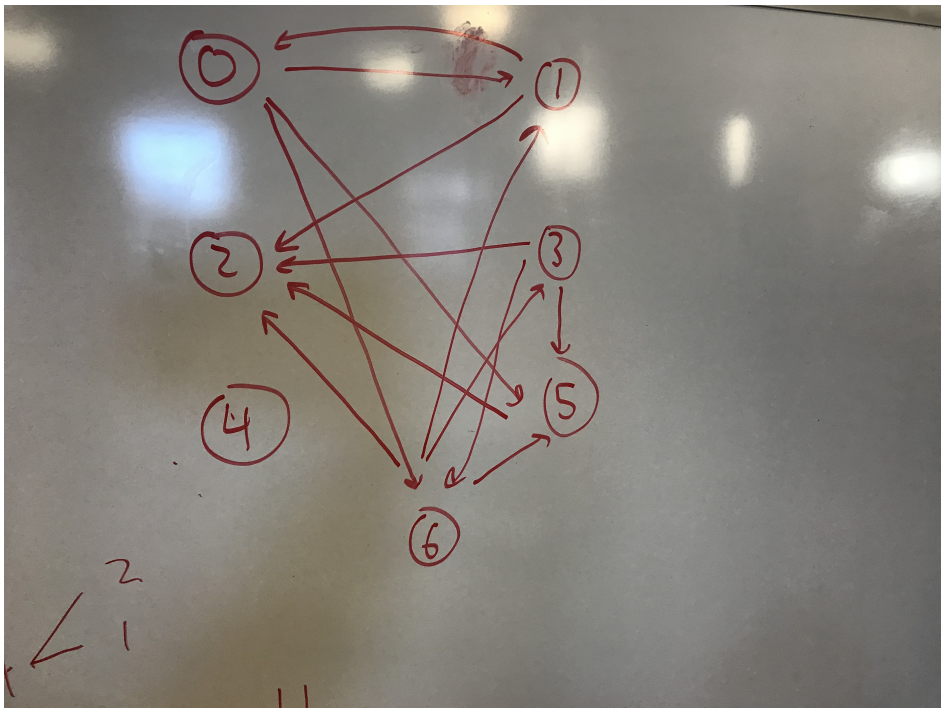
Name: Kevin Jiang

Collaborators: Suraj Srinivasan, Jason Lu

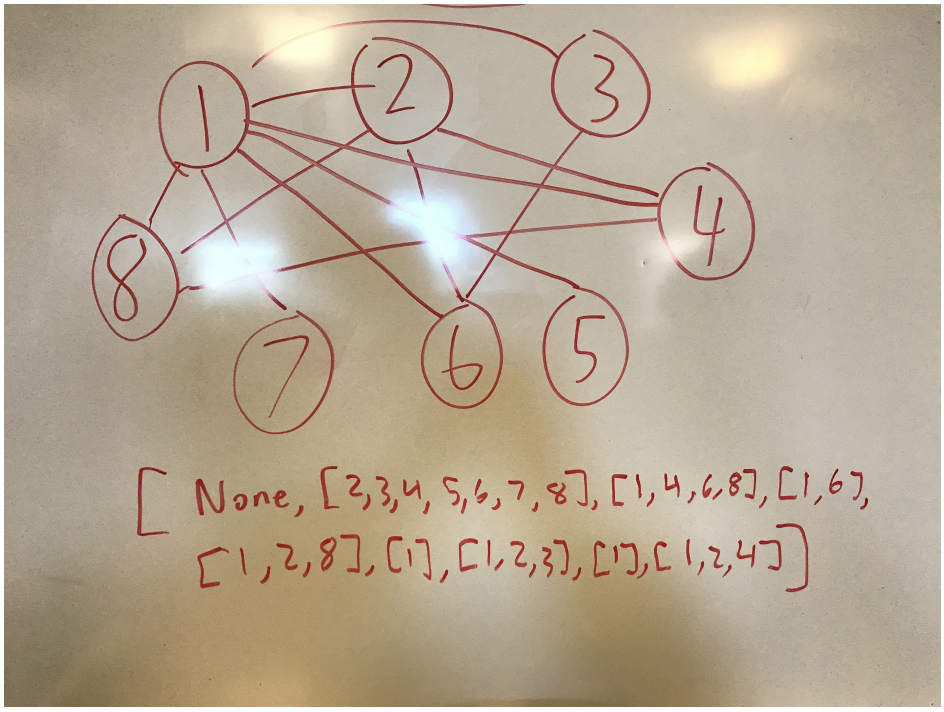
Problem 6-1.

(a)

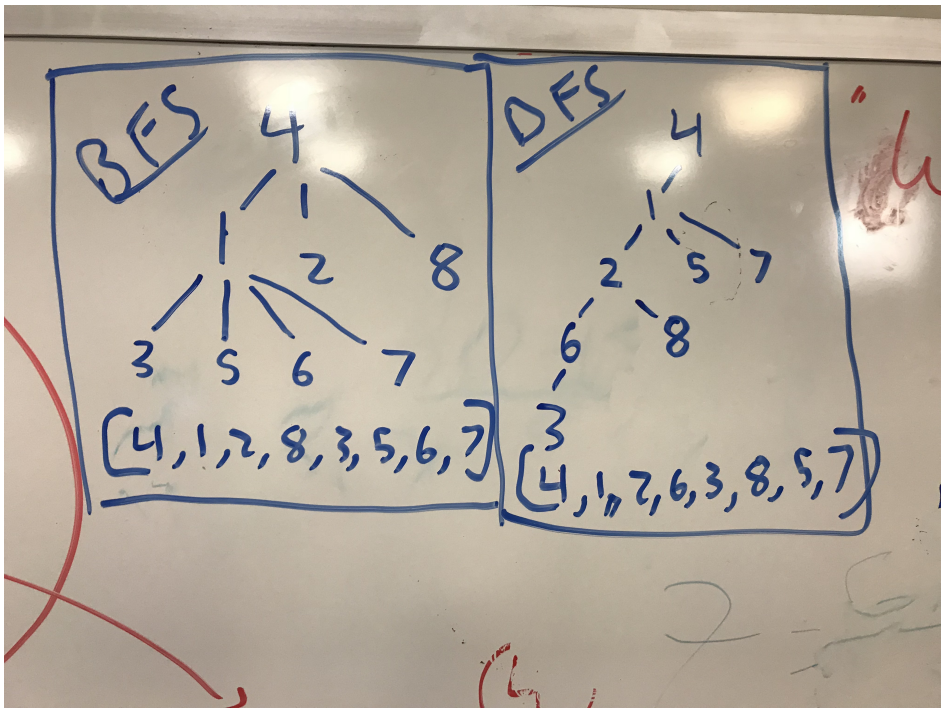




(b)



(c)



Problem 6-2. Let c denote the initial minimum radius, $R(G)$. We loop through each vertex $v \in V$ and find its radius $r(u)$ by looking through each edge. This takes $O(|E|)$ time since we are looping through at most all of the edges in the graph. If $r(u) < c$, then we set $c = r(u)$. We repeat these steps for each vertex in the graph. At the end of this loop, we will have found the smallest radius since we are comparing the radius of every vertex in the graph. Since we are doing $O(|E|) + O(1)$ work for V vertices, the total run time is $O(|V|(|E| + 1)) = O(|V||E| + |V|)$, Since the graph is connected, we know that $V = n = O(n)$ and $E = \binom{n}{2} = O(n^2)$ for some positive integer n. Therefore, the $O(|V||E|)$ dominates $O(|V|)$, so the run time is $O(|V||E|)$.

Problem 6-3. We first loop through the list of routes and create an adjacency list. This step takes worst case $O(E)$ time, where E is the number of edges (routes) in the graph. Now we create a parent array with each index corresponding to a city and None if the city has not been visited before. Creating the list takes worst case $O(V)$ time since we are looping through all of the cities.

Take the capital in our adjacency list and perform DFS on it. We change the corresponding value in our parent array from None to itself. After the first DFS has finished, we check for the next time a None appears in our parent list and perform DFS again. Other than the first DFS, the order we perform DFS on None nodes does not matter due to the graph being undirected. We repeat these steps until all of the elements in the DAA are not None, meaning that every city is reachable from the capital. The number of times we perform full DFS minus one (from the initial) gives us the number of new routes to create from the capital to the "separated" cities. The run time for these steps is worst case $O(V + E) + O(V)$ since all the DFS combined take worst case $O(V + E)$ time and checking for the next None value is $O(V)$ (looping through at most V cities).

The overall run time for the algorithm is worst case $O(E)+O(V)+O(V+E)+O(V) = O(V+E)$, which is linear.

Problem 6-4.

(a) We create a graph $G = \{V, E\}$, where each vertex is either a source or destination airport, and each edge is a flight that is weighted with an integer from 1 to k. From G, we construct a new graph G' by splitting each edge (u, v) into $w(u, v)$ edges of weight 1. Since the weight of each edge in G is $\leq k$, we can create k new dummy vertices for each edge. Thus, G' has $V + (k-1)E$ vertices and at most kE edges. Since all the edges in $G' = (V + (k-1)E, kE)$ are equally weighted 1, the run time of BFS is $O((V + (k-1)E) + kE)) = O(V + (2k-1)E) = O(V + kE)$. Since there are n airports and f flights, the run time is $O(n + kf)$.

(b) We construct a new graph $G' = \{V', E'\}$ by taking each vertex v in $G = \{V, E\}$ and creating v_1, v_2, v_3, $\cdots$, v_k in G'. For each set of vertices $v_j \in V'$, where $1 \leq j \leq k$, we create a path from v_1 to v_k by creating edges $(v_m, v_m + 1)$ for $1 \leq m \leq k-1$. For each edge $(u, v) \in E$ of length n, we construct the edge $(u_n, v_1) \in E'$.

In G', there are kV vertices and $(k-1)V + E$ edges. Thus, running BFS on $G' = (kV, (k-1)V+E)$ takes $O(kV+(k-1)V+E) = O((2k-1)V+E) = O(kV+E) = O(kn + f)$ time.

Problem 6-5. Given a graph $G = \{V, E\}$ where V is the number of caves, and E is the number of tunnels, we construct a new graph $G' = \{V', E'\}$ by creating vertices v_3, v_2, v_1, and v_0 for each vertex $v \in V$. The subscript of v indicates how many lives Lelda has left right after she has exited that cave.

Note that finding a path from one of the entrance caves to Zink is the same as finding a path from Zink to one of the entrance caves. Therefore, we construct the edges in the graph by first taking the vertex Zink is at, $k \in V$, and setting it to $k_3 \in V'$. Starting with this node, for each edge $(u, v) \in E$ we construct the edge in G' based on the number of lives u has. If we are going from a safe or dangerous cave to a dangerous cave, then we construct the edge $(u_k, v_{k-1}) \in G'$. If we are going from a safe or dangerous cave to a safe cave, then we construct the edge $(u_k, v_k) \in G'$. If at any point $k = 0$, then we don't construct the rest of the edges in that path.

To find the route from Zink to an entrance cave, we run DFS starting at Zink's vertex. If we stop our path at a vertex $p_k \in V'$ with $k = 0$, then we have to backtrack. If we stop at a vertex with $k > 0$ lives, then we have found an entrance cave and have successfully won. If DFS finishes without finding an entrance cave, then we return the 'Game Over' situation.

G' has at most $4V$ vertices and E edges since we are creating 4 new vertices for each $v \in V$ and looking at most E edges. Since $G' = \{4V, E\}$, DFS takes $O(4V + E) = O(V + E)$ time, or linear time.

Problem 6-6.

(a) We simply loop through each element in the tuple of tuples and check if the numbers increase properly from 1 to k^2. Since we are looping through k^2 elements, the run time is $O(k^2)$.

(b) Given a k x k configuration, there are $2k$ possible single moves to change the configuration. For each of these $2k$ moves, we can just loop through each element to compute the configuration. Since computing each of the $2k$ configurations take $O(k^2)$ time, the overall run time to compute all configurations reachable from a given Rubiks configuration via a single move is $O(2k \cdot k^2) = O(k^3)$.

(c) For each square in a Rubik's k-Square, there are four possible locations after any number of swaps: reflected across column $k/2$, reflected across row $k/2$, reflected across column $k/2$ and across row $k/2$, and its original position. Since there are four locations for each of the k^2 squares, there are at most 4^{k^2} distinct configurations.

(d) We create a graph $G = (V, E)$ that has every possible distinct Rubik's k-Square configuration as vertices and 1 swap moves as edges. From part c, we know that there are at most 4^{k^2} distinct configurations. From part b, we know that every configuration has $2k$ possible one moves to get to a different configuration. Therefore, there are $V = 4^{k^2}$ vertices and $E = 4^{k^2} \cdot 2k$ edges. Since we want the shortest path, BFS runs in $O(V + E) = O(4^{k^2} + 4^{k^2} \cdot 2k) = O(4^{k^2}k)$ time. However, for each edge, we have to check if we are in a solved state, which takes $O(k^2)$ time, so the overall run time is $O(k^2) \cdot O(4^{k^2}k) = O(4^{k^2}k^3)$.

(e) Submit your implementation to `alg.mit.edu/PS6`