Problem Set 6

All parts are due on October 25, 2018 at 11PM. Please write your solutions in the LaTeX and Python templates provided. Aim for concise solutions; convoluted and obtuse descriptions might receive low marks, even when they are correct. Solutions should be submitted on the course website, and any code should be submitted for automated checking on `alg.mit.edu`.

Problem 6-1. [16 points] **Graph Mechanics**

(a) [4 points] Draw the **directed** graph associated with each of the following graph representations.

```
G1 = {"rock"     : ["scissors", "lizard"],
      "paper"    : ["rock"    , "spock"],
      "scissors" : ["paper"   , "lizard"],
      "lizard"   : ["paper"   , "spock"],
      "spock"    : ["rock"    , "scissors"]}

G2 = [[1,5,6], [0,2], [], [6,2,5], None, [2], [1,2,3,5]]
```

(b) [4 points] The ***order-k division graph*** is the undirected graph on vertices $\{1, \ldots, k\}$, with an edge connecting two distinct vertices if and only if one vertex is a factor of the other. For example, the order-4 division graph contains the edge $\{4, 2\}$ but not $\{4, 3\}$. Draw a picture of the order-8 division graph, and write its graph representation as a direct access array of adjacency lists, as in `G2` above. Double check your graph: it should have exactly 12 edges.

(c) [8 points] Run breadth-first search and depth-first search on the order-8 division graph, starting at node 4. When performing each search, visit the neighbors of each vertex in increasing order. Draw the parent tree constructed by each search, and list the order in which nodes were first visited.

Problem 6-2. [7 points] **Graph Radius**

In any undirected graph $G = (V, E)$, the ***radius*** $r(u)$ of a vertex $u \in V$ is the distance to the farthest other vertex v, i.e. $r(u) = \max\{\delta(u, v) \mid v \in V\}$; while the ***radius*** $R(G)$ of an undirected graph $G = (V, E)$ is the smallest radius of any vertex, i.e. $R(G) = \min\{r(u) \mid u \in V\}$. Describe a $O(|V||E|)$-time algorithm to compute the radius of a given **connected** undirected graph.

Problem 6-3. [8 points] **Tramak Railway**

Railey P. Trainston has acquired many local railroad networks across the country, and has combined them to form **Tramak**, a single national railway system. A Tramak ***route*** is a two-way train service between two cities having no other train stop between them. A city is ***reachable*** from another city on the Tramak network if there is a sequence of connected routes from one to the other. Each of the country's major cities participates in **at least** one route from some acquired local railroad, but not all cities are reachable from each other. Given a list of routes from the acquired local networks, describe a linear-time algorithm to determine the minimum number of new routes that Railey must add to ensure that every city is reachable from the capital. Specify whether your algorithm's running time is worst-case or expected, and state any assumptions you make.

Problem 6-4. [12 points] **Anti-Aloft Aviation**

The Wleft brothers have just launched a new airline featuring their own fixed-wing aircraft, with flights between n locations. They make a list of the f flights operated by their airline, where each flight is defined by a source airport, a destination airport, and the number of minutes of flying time: an integer between 1 and k inclusive. The Wleft brothers want to plan a route from their hometown of Dayton, Ohio to Cambridge, Massachusetts for their upcoming talk at MIT LSC. To maximize safety, they want to minimize the total number of minutes they spend flying, not caring about the number of connections nor any layover time.

(a) [4 points] Describe a $O(n + kf)$-time algorithm to help the brothers compute the minimum possible flying time from Dayton to Cambridge, using breadth-first search.

(b) [8 points] Describe a $O(kn + f)$-time algorithm to help the brothers compute the minimum possible flying time from Dayton to Cambridge, using breadth-first search. (Hint: replace each vertex with a path.)

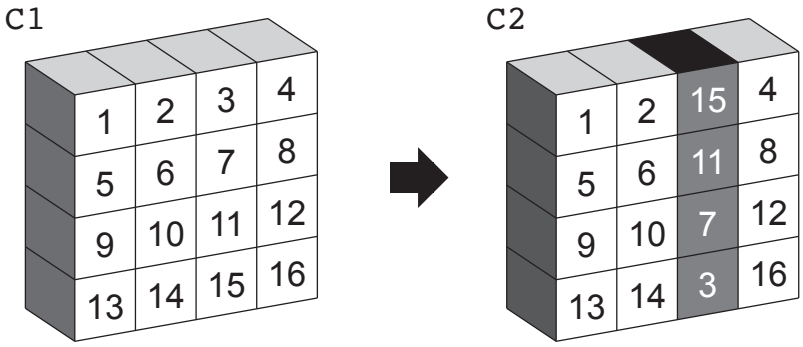
Problem 6-5. [12 points] **Three-Heart Challenge**

Princess Lelda is on a quest to save her friend Zink from the evil force of Anon. Luckily, the roaming shopkeeper Beetle sold her a map of Anon's underworld headquarters, which consists of various ***caves*** and ***tunnels***: bidirectional connections between the caves. Lelda's goal is to reach the cave where Zink is being held, starting at one of many possible cave entrances. The map marks some caves as ***dangerous***, which indicates that if Lelda enters that cave, she would need to fight enemies resulting in exactly one heart of damage. Lelda can take at most two hearts of damage on her way to Zink; taking three hearts of damage would lead to a "game over" situation. Describe a linear-time algorithm to help Lelda find a route to Zink, if such a route exists.

Problem 6-6. [45 points] **Rubik's Squares**

The ***Rubik's cube*** is a puzzle which consists of a $3 \times 3 \times 3$ grid of smaller cubes called ***cubies*** which can be rotated relative to each other to form different multi-colored configurations of the cube. In the solved configuration, each 3×3 face of the Rubik's cube is monochromatic, and each of the six faces is a different color. The goal of the puzzle is to return the cube to the solved configuration via a sequence of moves.

In this problem, we will study ***Rubik's k-Square*** puzzles, consisting of a $k \times k \times 1$ grid of cubies, where the solved configuration consists of six monochromatic faces. From any Rubik's k-Square configuration, there are $2k$ possible ***moves*** to transform the configuration into a new configuration. Each move changes the position and orientation of exactly one row or column of cubies from a input configuration in the following way: to move row (or column) i of a configuration, reverse the order of the cubies within that row (or column), and flip the orientation of each cubie in the row (or column) upside-down. Below shows the result of a column-2 move applied to the solved configuration of a Rubik's 4-Square.



We will represent a Rubik's k-Square configuration via a length-k tuple of length-k tuples, where the y-th inner tuple corresponds to the y-th row of the configuration, and the x-th element of row y corresponds to the cubie positioned at row y and column x. We represent each cubie as an integer between 1 and k^2 inclusive, where integer i represents the cubie that exists in row $y = \lfloor (i-1)/k \rfloor$ and column $x = i - ky - 1$ in the solved configuration. In addition, we use positive i when the cubie in the configuration has the same orientation as in the solved configuration, and negative i when the cubie is upside-down. For example, the two configurations shown above have the following representations:

```
C1 = (( 1,  2,  3,  4),        #      C2 = (( 1,  2,-15,  4),
      ( 5,  6,  7,  8),        #            ( 5,  6,-11,  8),
      ( 9, 10, 11, 12),        #            ( 9, 10, -7, 12),
      (13, 14, 15, 16))        #            (13, 14, -3, 16))

C2 = move(C1, ("col", 2))
```

(a) [4 points] Describe an $O(k^2)$-time algorithm to determine whether a given Rubik's k-Square configuration is solved.

(b) [4 points] Describe an $O(k^3)$-time algorithm to compute all configurations reachable from a given Rubik's configuration via a single move.

(c) [4 points] Show that a Rubik's k-Square has at most c^{k^2} distinct configurations, for some constant c.

(d) [8 points] Describe an $O(c^{k^2}k^3)$-time algorithm to find the shortest sequence of moves to solve a Rubik's k-Square.

(e) [25 points] Implement your algorithm by writing functions `is_solved`, `move`, and `solve_ksquare` in the code template provided. The input to `solve_kquare` will be a Rubik's k-square configuration, and it should return a list of moves where: (1) each move is of the form `(s, i)` for some `s` $\in \{$`"row"`,`"col"`$\}$ and `i` $\in \{0, \ldots, k-1\}$, and (2) applying the sequence of moves in order to the input configuration results in the solved configuration. You can download a code template containing some test cases from the website. Submit your code online at `alg.mit.edu`.

```
def is_solved(config)
    "Return whether input config is solved"
    ##################
    # YOUR CODE HERE #
    ##################
    return True

def move(config, mv):
    "Return a new configuration made by moving config by move mv"
    k = len(config)
    (s, i) = mv          # s in ("row", "col") and i in range(k)
    ##################
    # YOUR CODE HERE #
    ##################
    return tuple([tuple(row) for row in config])

def solve_ksquare(config):
    "Return a list of moves that solves config"
    ##################
    # YOUR CODE HERE #
    ##################
    return []
```