Problem Set 5

All parts are due on October 18, 2019 at 6PM. 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 5-1. [15 points] **Graph Mechanics**

(a) [8 points] Draw the **directed** graph associated with each of the following graph representations. `G1` is provided as direct access array of adjacency lists, while `G2` is provided as an adjacency matrix.

```
G1 = [[3], [0, 2, 3], [5], [], [1, 3], [0]]

G2 = [[0, 0, 1, 0, 0],
      [0, 0, 1, 0, 1],
      [0, 0, 0, 0, 0],
      [0, 1, 1, 0, 0],
      [0, 0, 1, 1, 0]]
```

(b) [3 points] Let the ***k*-prime-sum** graph be the undirected graph on vertices $\{1, \ldots, k\}$ having an undirected edge between distinct vertices u and v if and only if they sum to a prime number. For example, the 4-prime-sum graph contains edges $\{1, 4\}$ and $\{2, 3\}$ (and others), but not edges $\{1, 3\}$ or $\{2, 4\}$. Draw a picture of the 8-prime-sum graph, and write its graph representation as a direct access array of adjacency lists, as in `G1` above. Hint: your graph should have 11 edges.

(c) [4 points] Run breadth-first search and depth-first search on the 8-prime-sum graph, starting from node 7. While performing each search, visit the neighbors of each vertex in increasing order. For the search, list the order in which nodes are first visited.

Problem 5-2. [10 points] **Graph Radius**

In any undirected graph $G = (V, E)$, the **eccentricity** $\epsilon(u)$ of a vertex $u \in V$ is the shortest distance to its farthest vertex v, i.e., $\epsilon(u) = \max\{\delta(u, v) \mid v \in V\}$. The **radius** $R(G)$ of an undirected graph $G = (V, E)$ is the smallest eccentricity of any vertex, i.e., $R(G) = \min\{\epsilon(u) \mid u \in V\}$.

(a) [5 points] Given connected undirected graph G, describe an $O(|V||E|)$-time algorithm to determine the radius of G.

(b) [5 points] Given connected undirected graph G, describe an $O(|E|)$-time algorithm to determine an upper bound R^* on the radius of G, such that $R(G) \leq R^* \leq 2R(G)$.

Problem 5-3. [10 points] **Internet Investigation**

MIT has heard complaints regarding the speed of their WiFi network. The network consists of r routers, some of which are marked as **entry points** which are connected to the rest of the internet. Some pairs of routers are directly connected to each other via bidirectional wires. Each wire w_i between two routers has a known length ℓ_i measured in a positive integer number of feet. The **latency** of a router in the network is proportional to the minimum feet of wire a signal from the router must pass through to reach an entry point. Assume the latency of every router is finite and there is at most $100r$ feet of wire in the entire network. Given a schematic of the network depicting all routers and the lengths of all wires, describe an $O(r)$-time algorithm to determine the sum total latency, summed over all routers in the network.

Problem 5-4. [10 points] **Quadwizard Quest**

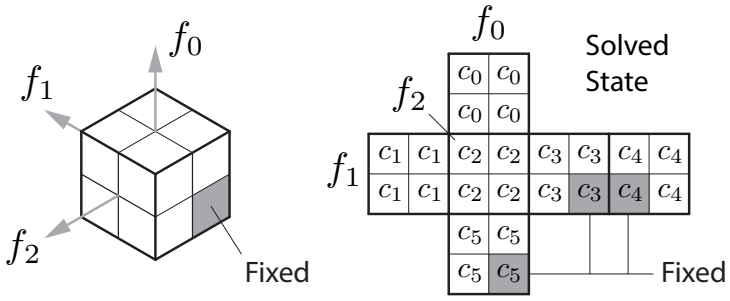
Wizard Potry Harter and her three wizard friends have been tasked with searching every room of a Labyrinth for magical artifacts. The Labyrinth consists of n rooms, where each room has at most four doors leading to other rooms. Assume all doors begin closed and every room in the Labyrinth is reachable from a specified entry room by traversing doors between rooms. Some doors are protected by evil enchantments that must be **disenchanted** before they can be opened; but all other doors may be opened freely. Given a map of the Labyrinth marking each door as enchanted or not, describe an $O(n)$-time algorithm to determine the minimum number of doors that must be disenchanted in order to visit every room of the Labyrinth, beginning from the entry room.

Problem 5-5. [10 points] **Purity Atlantic**

Brichard Ranson is the founder of Purity Atlantic, an international tour company that specializes in planning luxury honeymoon getaways for newlywed couples. To book a customized tour, a couple submits their home city, and the names of three touring cities they would like to visit during their honeymoon. Then Purity will arrange all accommodations, including a **flight itinerary**: a sequence of flights from their home to each touring city (in any order), then returning back to their home. Unfortunately, it's not always possible to fly directly between any two cities, so multiple flights may be required. While cost and time are not a factor, couples prefer to minimize the number of direct flights they will have to take during their honeymoon. Given a list of c cities and a list of all f available direct flights, where each direct flight is specified by an ordered pair of cities (origin, destination), describe an efficient algorithm to determine a flight itinerary for a given couple that minimizes the number of direct flights they will have to take.

Problem 5-6. [45 points] **Pocket Cube**

A Pocket Cube[1] is a smaller $2 \times 2 \times 2$ variant of the traditional $3 \times 3 \times 3$ Rubik's cube, consisting of eight corner cubes, each with a different color on its three visible faces. The **solved** configuration is when each 2×2 face of the Pocket Cube is monochromatic. We reference each color c_i with an index $i \in \{0, \ldots, 5\}$. Without loss of generality, we fix the position and orientation of one of the corner cubes and only allow single-turn rotations about the normals of the three faces of the Pocket Cube $\{f_0, f_1, f_2\}$ that do not contain the fixed corner cube; specifically, a **move** is described by tuple (j, s) corresponding to a single-turn rotation of face f_j, clockwise when $s = 1$ and counterclockwise when $s = -1$. Breadth-first search can be used to solve puzzles like the Pocket Cube by searching a graph whose vertices are possible configurations of the puzzle, with an edge between two configurations if one can be reached from the other via a single move. Instead of storing these adjacencies explicitly, one can compute the neighbors of a given configuration by applying all possible single moves to the configuration.



(a) [4 points] Argue that the number of distinct configurations of a Pocket Cube is less than 12 million (try to get as tight a bound as you can using combinatorics).

(b) [2 points] State the max and min degree of any vertex in the Pocket Cube graph.

(c) [2 points] In your problem set template is code that fully explores the Pocket Cube graph from a given configuration using breadth-first search, and then returns a sequence of moves that solves the Pocket Cube (assuming the solved configuration is reachable). However, this solver is very slow[2]. Run the code provided and state the number of configurations the search explores. How does this number compare to your upper bound from part (a)?

(d) [2 points] State the max number of moves needed to solve any solvable Pocket Cube.

(e) [10 points] Let d be the max degree of the Pocket Cube graph, and let w be the max number of moves needed to solve any solvable Pocket Cube. Describe an algorithm to find a shortest sequence of moves to solve any Pocket Cube configuration (or return no such sequence exists) that visits no more than $2d^{\lceil w/2 \rceil}$ configurations.

(f) [25 points] Rewrite the `solve(config)` function in the template code provided, based on your algorithm from part (e). You can download the code template and some test cases from the website. Submit your code online at `alg.mit.edu`.

[1] `http://en.wikipedia.org/wiki/Pocket_Cube`

[2] Please note that the code requires a couple minutes and considerable memory (over 400 Mb) to complete.