430.329 (001)
Introduction to Algorithms

HW#3
2018 Fall

Submit your answer and the code for the programming exercise to the ETL as "Student-ID.zip". e.g.) 2018-9999.zip

1. **[10 pts] [Greedy Algorithm]** Suppose you are given two sets A and B, each containing *n* positive integers. You can choose to reorder each set however you like. After reordering, let a_i be the ith element of set A, and let b_i be the ith element of set B. You then receive a payoff of $\prod_{i=1}^{n} a_i^{b_i}$. Give an algorithm that will maximize your payoff. Prove that your algorithm maximizes the payoff.

2. **[10 pts] [Minimum Spanning Tree]** Let *T* be a minimum spanning tree of a graph *G = (V, E)*, and let *V`* be a subset of *V*. Let *T`* be the subgraph of *T* induced by *V`*, and let *G`* be the subgraph of *G* induced by *V`*. Show that if *T`* is connected, then *T`* is a minimum spanning tree of *G`*.

3. **[20 pts] The Party Organizer**
You are planning the seating arrangement for a party given a list of guests, V.

(a) Assume that you have a table S where $S[u]$ for $u \in V$ is a list of guests that u knows. If u knows v, then v knows u. You have to arrange the seating satisfying that any guest at a table knows every other guest sitting at the same table either directly or through some other guests sitting at the same table. For example, if x knows y, and y knows z, then x, y, z can sit at the same table. Describe an efficient algorithm that, given V and T, returns the minimum number of tables needed to achieve this requirement. Analyze the running time of your algorithm.

(b) Assume that there are only two tables, and you are given a different lookup table S where $S[u]$ for $u \in V$ is a list of guests who are in a bad relationships with u. If v is in a bad relationship with u, then u is on bad relation with v. Your goal is to arrange the seating such that no pair of guests sitting at the same table are in a bad relationship with each other. Figure 1 below shows two graphs in which we present each guest as a vertex and an edge between two vertices means these two guests are on bad relation with each other. Figure 1(a) is an example where we can achieve the goal by having A, C sitting at one table and B, E, D sitting at another table. Figure 1(b) is an example where we cannot achieve the goal. Describe an efficient algorithm that, given V and S, returns TRUE if you can achieve the goal or FALSE otherwise. Analyze the running time of your algorithm.

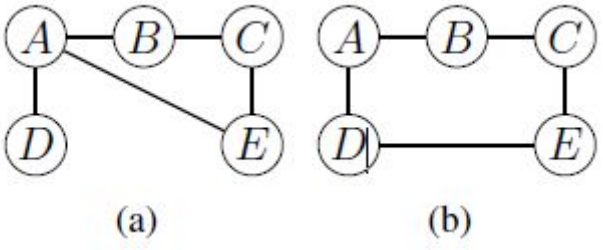
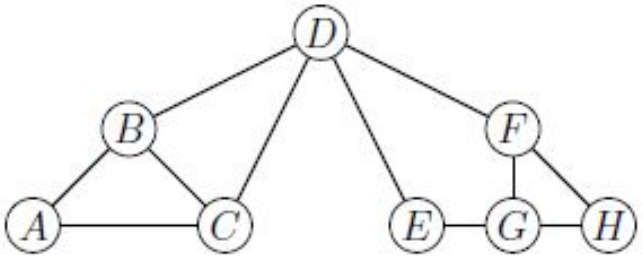


Figure 1: Examples of guest relationships presented as graphs

4. **[25 pts] Graph**

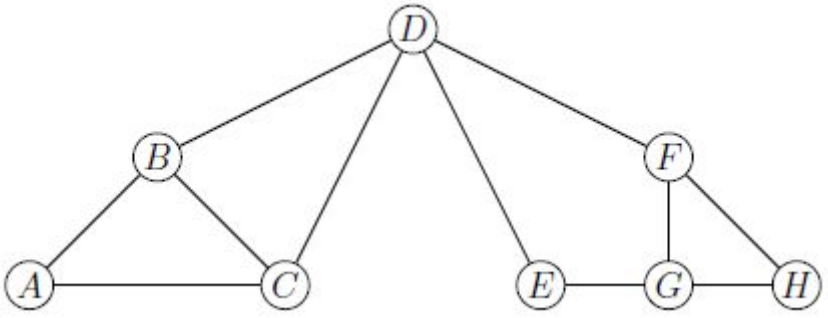
You are given a map, 'SNU-maze'. Realizing that you can convert a map into a graph with vertices representing cells and edges representing passages, you want to use your newly learned graph-search algorithms to navigate the map. Consider the following converted graph.



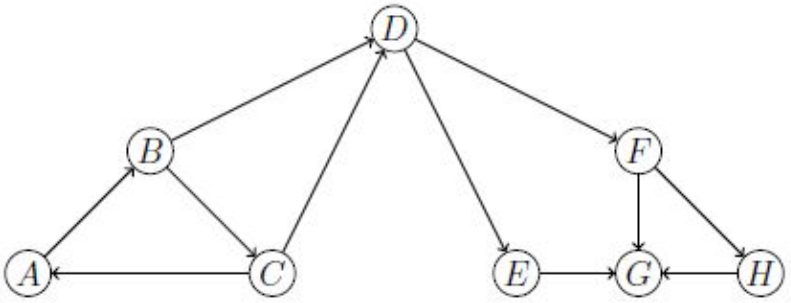
For the following questions, assume that the graph is represented using adjacency lists, and that all adjacency lists are sorted, i.e., the vertices in an adjacency list are always sorted alphabetically.

(a) Suppose that you want to find a path from *A* to *H*. If you use BFS, write down the resulting path as a sequence of vertices.

(b) To determine whether the maze has cycles or multiple paths to the same destination, you decide to use the edge classification of DFS. Run DFS on the graph reproduced below, starting from vertex *A*, and label every edge with *T* if it's a tree edge, *B* if it's a back edge, *F* if it's a forward edge, and *C* if it's a cross edge.
As a reminder, recall that an edge (*u, v*) is
- a tree edge (*T*) if v was first discovered by exploring edge (*u, v*) (tree edges form the DFS forest);
- a back edge (*B*) if v is u's ancestor in a depth-first tree;
- a forward edge (*F*) if v is u's descendant in a DFS tree; and
- a cross edge (*C*) if none of the above apply.



(c) Now suppose that the passages in the maze are directional. Rerun DFS in the directed graph below and label edges accordingly.



5. **[Programming Exercise]**

[35 pts] [Prime Path] Given two four-digit numbers, we need to find the shortest path from one to another by altering **only a single digit** at a time such that every number that we get after changing a digit is **prime**. For instance, let 1033 and 8179 are the two given prime numbers. Then the path will be 1033 → 1733 → 3733 → 3739 → 3779 → 8779 → 8179, which are all prime numbers, and the answer(number of the arrows) will be 6. This can be done by following two steps. First, build a graph of prime numbers whose edges connect the prime numbers which have only a single different digit. And then find the shortest path between two prime numbers.

1) **[15 points]** Design a function that returns an adjacency list representation of the graph of primes. The edges of the graph connect the prime numbers which have only a single different digit, such as 6329 - 6529.

2) **[20 points]** Design a function that returns the length of the shortest path from one to another.

The skeleton code(***hw3.py***) and the list of prime numbers(***prime.txt***) for the homework will be provided. Please implement your function in that file.
Please note that you are "not" allowed to use the modules that directly calculate the shortest path, such as "Dijkstar" or "NetworkX".

Example 1) :
Input: A = 1033
B = 8179
The shortest path : 1033 →1733 →3733 →3739 → 3779 →8779 →8179
Output: length(shortest path) = 6 (number of arrows)

Example 2) :
Input: A = 6329
B = 8537
The shortest path : 6329 →6529 →3529 → 3539 → 8539 →8537
Output: length(shortest path) = 5 (number of arrows)

Example 3) :
Input: A = 9721
B = 4079
The shortest path : 9721 →4721 →4021 → 4091 → 4099 →4079
Output: length(shortest path) = 5 (number of arrows)