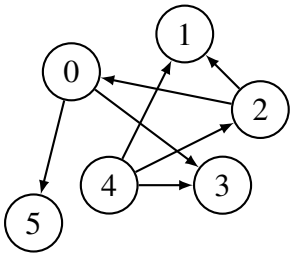
# Problem Set 6

**All parts are due on October 25, 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 6-1.** [10 points] **Topological Training**

Please answer the following questions about the unweighted directed graph $G$ below, whose vertex set is $\{0, 1, 2, 3, 4, 5\}$.




**(a)** [5 points] State a topological ordering of $G$. Then state and **justify** the number of distinct topological orderings of $G$.

**(b)** [5 points] State a single directed edge that could be added to $G$ to construct a simple[1] graph with no topological ordering. Then state and **justify** the number of distinct single edges that could be added to $G$ to construct a simple graph with no topological ordering.

**Problem 6-2.** [10 points] **DFS Relaxation**

Given an unweighted undirected graph $G = (V, E)$ and vertex $s \in V$, consider the following algorithm (DFS Relaxation):

- Initialize $d'[s] = 0$ and $d'[v] = \infty$ for all $v \in (V - \{s\})$
- Run some depth-first search (DFS) from $s$ (there can be multiple valid ways to DFS)
- Whenever the search **first visits** a vertex $v$ from a vertex $u$ set $d'[v] = 1 + d'[u]$

**(a)** [4 points] In the special case when $G$ is a **tree**, prove that any DFS Relaxation from a vertex $s$ in the tree correctly computes shortest path distances from $s$, i.e., $d'[v] = \delta(s, v)$ for every $v \in V$.

**(b)** [2 points] State an undirected graph $G = (V, E)$ on three vertices for which DFS Relaxation from some vertex $s \in V$ **incorrectly** computes the shortest path distance to some vertex, i.e., $d'[v] \neq \delta(s, v)$ for some $v \in V$.

**(c)** [4 points] An unweighted graph $G = (V, E)$ **fails spectacularly** if every DFS Relaxation on $G$ (from any vertex $s \in V$, using any valid depth-first search edge visitation order) results in some $d'[v]$ for which $d'[v] \geq \frac{1}{2}|V| \cdot \delta(s, v)$, i.e., $d'[v]$ is at least a factor of $\frac{1}{2}|V|$ away from the true shortest path distance from $s$. Given any integer $k > 3$, describe an undirected graph on $k$ vertices that fails spectacularly.

[1] A simple graph has no self-loops (i.e., each edge connects two different vertices) and has no multi-edges (i.e., there can be at most one directed edge from vertex $a$ to vertex $b$, though a directed edge from $b$ to $a$ may exist).

**Problem 6-3.** [10 points] **Never Return**

Bimsa is a young lioness whose evil uncle has just banished her from the land of Honor Stone, proclaiming: "Run away, Bimsa. Run away, and never return." Bimsa has knowledge of all landmarks and trails in and around Honor Stone. For each landmark, she knows its $x$ and $y$ coordinates and whether it is inside or outside of Honor Stone. For each trail, she knows its positive integer length and the two landmarks it directly connects. Each landmark connects to at most five trails, each trail may be traversed in either direction, and every landmark can be reached along trails from the **gorge**, the landmark where Bimsa is now. Bimsa wants to leave Honor Stone quickly, while only traversing trails in a way that **never returns**: traversing a trail from landmark $a$ to landmark $b$ never returns if the distance[2] from $a$ to the gorge is strictly smaller than the distance from $b$ to the gorge. If there are $n$ landmarks in Honor Stone, describe an $O(n)$-time algorithm to determine a shortest route that never returns, from the gorge to any landmark outside of Honor Stone.

**Problem 6-4.** [15 points] **DigBuild**

Software company Jomang is developing **DigBuild**, a new 3D voxel-based exploration game. The game world features $n$ collectable block types, each identified with a unique integer from 1 to $n$. Some of these block types can be converted into other block types: a conversion $(d_1, b_1, d_2, b_2)$ allows $d_1$ blocks of type $b_1$ to be converted into $d_2$ blocks of type $b_2$; each conversion is animated in the game by repeatedly dividing blocks in half, then recombining them, so $d_1$ and $d_2$ are always constrained to be integer powers of two. Jomang wants to randomly generate allowable game conversions to increase replayability, but wants to disallow sets of game conversions through which players can generate infinite numbers of blocks via conversion. Describe an $O(n^3)$-time algorithm to determine whether a given a set of $\left\lfloor \frac{1}{5}n^2 \right\rfloor$ conversions should be disallowed. Assume that a starting world contains $D$ blocks of each type, where $D$ is the product of all $d_i$ appearing in any conversion. Given positive integer $x$, assume that its bit-length, $\log_2 x$, can be computed in $O(1)$ time.

**Problem 6-5.** [15 points] **Tipsy Tannister**

Lyrion Tannister is a greedy, drunken nobleman residing in Prince's Pier, the capital of Easteros, who wants to travel to the northern town of Summerrise. Lyrion has a map depicting all towns and roads in Easteros, where each road allows direct travel between a pair of towns, and each town is connected to at most seven roads. Each road is marked with the non-negative number of gold pieces he will be able to collect in taxes by traveling along that road in either direction (this number may be zero). Every town has a tavern, and Lyrion has marked each town with the positive number of gold pieces he would spend if he were to drink there. If Lyrion ever runs out of money on his trip, he will just leave a debt marker[3] indicating the number of gold pieces he owes. After leaving the tavern in Prince's Pier with no gold and zero debt, Lyrion's policy will be to drink at the tavern of every third town he visits along his route until reaching Summerrise. Given Lyrion's map depicting the $n$ towns in Easteros, describe an $O(n^2)$-time algorithm to compute the maximum amount of gold he can have (minus debts incurred) upon arriving in Summerrise, traveling from Prince's Pier along a route that follows his drinking policy.

[2] We mean Euclidean distance between landmark $(x_1, y_1)$ and landmark $(x_2, y_2)$, i.e., $\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$.

[3] The marker will be readily accepted because Tannisters always pay their debts.

**Problem 6-6.** [40 points] **Cloud Computing**

Azrosoft Micure is a cloud computing company where users can upload **computing jobs** to be executed remotely. The company has a large number of identical cloud computers available to run code, many more than the number of pieces of code in any single job. Any piece of code may be run on any available computer at any time. Each computing job consists of a code list and a dependency list.

A **code list** $C$ is an array of code pairs $(f, t) \in C$, where string $f$ is the file name of a piece of code, and $t$ is the positive integer number of microseconds needed for that code to complete when assigned to a cloud computer. Assume file names are short and can be read in $O(1)$ time.

A **dependency list** $D$ is an array of dependency pairs $(f_1, f_2) \in D$, where $f_1$ and $f_2$ are distinct file names that appear in $C$. A dependency pair $(f_1, f_2)$ indicates that the piece of code named $f_1$ must be completed before the piece of code named $f_2$ can begin. Assume that every file name exists in some dependency pair.

**(a)** [5 points] A job $(C, D)$ can be **completed** if every piece of code in $C$ can be completed while respecting the dependencies in $D$. Given job $(C, D)$, describe an $O(|D|)$-time algorithm to decide whether the job can be completed.

**(b)** [10 points] Azrosoft Micure wants to know how fast they can complete a given job. Given a job $(C, D)$, describe an $O(|D|)$-time algorithm to determine the minimum number of microseconds that would be needed to complete the job (or return that the job cannot be completed).

**(c)** [25 points] Write a Python function `min_time(C, D)` that implements your algorithm from (b). You can download a code template containing some test cases from the website. Submit your code online at `alg.mit.edu`.

```
def min_time(C, D):
    '''
    Input:  C | a list of code pairs
            D | a list of dependency pairs
    Output: t | the minimum time to complete the job,
                or None if the job cannot be completed
    '''
    t = None
    ##################
    # YOUR CODE HERE #
    ##################
    return t
```