Problem Set 7

All parts are due on November 1, 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 7-1. [10 points] **cdkx**

In this problem we'll attempt to topologically sort the directed, 13-vertex graph illustrated below.[1] For this problem we can ignore **parallel** or duplicate edges: for example, there are three separate edges from 8 to 0, but we only need to list this connection once.

```
Adj = {0:[], 1:[0], 2:[1], 3:[1,2,4,6], 4:[1,5,6,7], 5:[0], 6:[0,10],
       7:[4,6,9], 8:[0], 9:[0,5,7], 10:[0,5], 11:[0,9,10], 12:[7,8,9,11]}
```

(a) [2 points] Give a one-sentence reason why this graph can't be topologically sorted unless at least two edges are deleted.

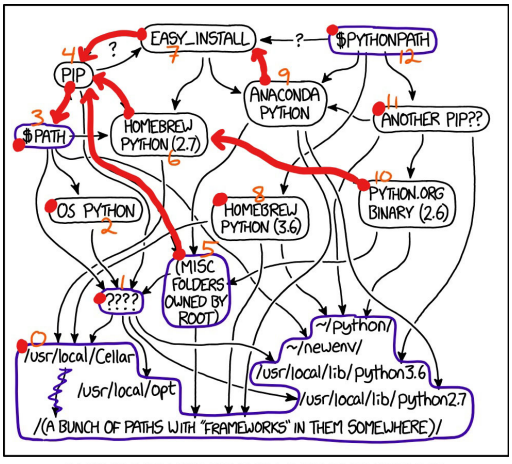
Solution: The two cycles, $4, 7$ and $7, 9$, both need to be broken to make the graph a DAG. (Note that the originally posted graph contained edge $(1, 5)$. That graph contained three disjoint cycles, each of which would need to be broken.)

Rubric:

- 1 point per edge (identifying each cycle)

(b) [5 points] Let's try anyway! Run a full DFS to search the entire graph, where an outer loop depth-first searches from unvisited vertices in order from 0 up to 12, and neighbors are always visited in increasing numerical order. Draw a graph of the parent pointers, and write down a list `F` of vertices arranged by finishing time.

Solution: Nodes $0, 1, 2, 3, 8, 11, 12$ are their own parents, and the other six parent pointers are illustrated below. The finishing time order is $0, 1, 2, 5, 10, 6, 9, 7, 4, 3, 8, 11, 12$. (Note that the originally posted graph contained edge $(1, 5)$, which would change the parent pointer of 5 to 1, and switch 2 and 5 in the finishing order.)



MY PYTHON ENVIRONMENT HAS BECOME SO DEGRADED
THAT MY LAPTOP HAS BEEN DECLARED A SUPERFUND SITE.

[1] Adapted from `http://xkcd.com/1987`, ©2018 Randall Munroe.

Rubric:

- 2 points for drawing of parent tree
- (Listing correct parent pointers without drawing the tree is OK)
- -1 point for each incorrect edge, minimum zero points
- 3 points for finishing time order
- -1 point per inversion from correct, minimum zero points

(c) [3 points] The list `reversed(F)` is almost a topological ordering. Which edges would need to be removed from the graph to make it a valid topological ordering?

Solution: Edges $(7, 4)$ and $(9, 7)$ are the only ones that point from a node to an earlier node in `reversed(F)`. Removing these two edges makes `reversed(F)` into a topological ordering. (Note that the originally posted graph contained directed edge $(1, 5)$. If student answered with respect to that graph, that edge would also need to be removed.)

Rubric:

- 3 points for correct identification of edges
- -2 points for each incorrect edge, minimum zero points

Problem 7-2. [10 points] **Early Termination**

Suppose you are given a weighted directed graph $G = (V, E, w)$ with positive and negative edge weights, a vertex $s \in V$, and a number k. You are told that every vertex $v \in V$ has a minimum-weight path from s that uses at most k edges. Design an algorithm that solves weighted single-source shortest-paths from s in $O(|V| + k|E|)$ time.

Solution: Perform Bellman-Ford, but stop after k rounds instead of $V - 1$. This requires $O(|V| + |E|)$ for initialization and $O(k|E|)$ for the k rounds, for a total of $O(|V| + k|E|)$.

To prove correctness, first observe that every node *has* a shortest path, so $\delta(s, v)$ is finite for every v and we therefore don't have to worry about negative cycles. For any node v, consider a shortest path $(s = v_0) \to v_1 \to \cdots \to (v_t = v)$ with $t \leq k$ edges, which is promised to exist. We are guaranteed to relax these edges sequentially, possibly with other relaxations interspersed: we relax edge (v_0, v_1) in the first round, edge (v_1, v_2) in round 2, and so on, finally relaxing (v_{t-1}, v_t) in round $t \leq k$. As shown in class, this is enough to prove that $d[v] = \delta(s, v)$ after round t and in all future rounds.

Rubric:

- 6 points for description of a correct $O(|V| + k|E|)$-time algorithm
- 2 points for a correct argument of correctness
- 2 points for a correct running time analysis

Problem 7-3. [10 points] **Transfiguration**

Germione Hranger is taking a class in transfiguration at the Warthogs School of Wizardcraft and Witchery. She has learned many spells, where each spell transforms one material into another material in a $1:r$ mass ratio for some $r > 0$. For example, she has learned a spell to transform x grams of salt into $3.2x$ grams of dragon scales, and spell to transform x grams of nightshade into $0.6x$ grams of gold. Given a list of all spells Germione has learned, describe an efficient[2] algorithm to determine the maximum amount of gold she can produce from s grams of salt.

Solution: Construct a graph G with: a vertex for each of the n materials appearing in any spell; and one directed edge for each of the m spells, where spell i converting 1 gram of material a into r_i grams of material b corresponds to a directed edge from material a to b with edge weight $-\log(r_i)$. This graph can be constructed in $O(n+m)$ time by performing constant work per vertex or edge. Then a minimum weight path $\pi = (v_0, \ldots, v_k)$ in graph G from salt (vertex s) to gold (vertex t), where r_i is the ratio of the corresponding spell from v_{i-1} to v_i, has weight $\sum_{i=1}^{k} -\log(r_i)$. Minimizing this quantity is the same as maximizing $\log(\prod_{i=1}^{k} r_i)$, i.e., the logarithm of a product of ratios converting salt via a sequence of transformations into gold. Note that since $\log(x)$ monotonically increases, maximizing $\log(x)$ corresponds to maximizing x. If no path exists from s to t, Germione cannot convert any salt into gold. If s is reachable from t via a negative weight cycle, then the maximum amount of gold Germione can convert is infinite. Thus we can use Bellman-Ford to compute the weight of a minimum weight path$\delta(s,t)$ from s to t in G, and return $2^{-\delta(s,t)}$ in $O(nm)$ time.

Rubric:

- 3 points for construction of a relevant graph
- 3 points for description of a correct algorithm
- 2 points if correct algorithm is $O(nm)$
- 2 point for a correct running time analysis
- Partial credit may be awarded

Problem 7-4. [12 points] **Escape to the Surface**

After solving PS6, Lelda has successfully reached the underground cave where Zink was being held, and now they must escape via one of the above-ground cave entrances. As before, some caves are marked **dangerous**, where Zink and Lelda will need to fight enemies in order to pass. With their combined fighting abilities, Zink and Lelda can win any fight without losing a single heart, but Zink's sword will take damage during the scuffle.[3] If ever Zink's sword takes a total of k damage, it will break. Beetle's map indicates the strength of enemies at any cave, from which they can estimate how much damage will be dealt to Zink's sword in that cave. To maintain morale, they decide only to traverse tunnels that make **vertical upward progress** to the surface, i.e. they

[2] By efficient, we mean that faster correct algorithms will receive more points than slower ones.

[3] Lelda uses a bow and arrow which does not take damage.

will only traverse a tunnel from cave a to cave b if cave b has strictly higher elevation than cave a (Beetle's map also indicates cave elevations). Describe an efficient algorithm to determine whether Lelda and Zink can escape to the surface always traveling up, without ever breaking Zink's sword.

Solution: Construct a graph G where each of the C caves corresponds to a vertex, having a directed edge for each of the T tunnels, adding an edge from cave a to cave b if: there is a tunnel from a to b and the elevation of cave b is strictly higher then the elevation of cave a, weighted by the amount of damage that would be done to Zink's sword at cave a. In addition, add a single vertex t, and a directed edge from each above-ground cave entrance to t, also weighted by the amount of damage that would be done at the corresponding cave entrance. Let vertex s correspond to the cave where Lelda and Zink begin. Then Lelda and Zink can escape to the surface without breaking Zink's sword if and only if a minimum weight path from s to t in G has weight less than k. G is acyclic because edges only connect to strictly higher caves, so compute $k > \delta(s, t)$ via DAG shortest paths in $O(C + T)$ time, relaxing edges from vertices in a topological sort order.

Rubric:

- 4 points for construction of a relevant graph
- 4 points for description of a correct algorithm
- 2 points if correct algorithm runs in linear time
- 2 points for a correct running time analysis
- Partial credit may be awarded

Problem 7-5. [13 points] **Psychic Psweets**

It's Halloween, and Lee Evan, a special child with special powers, wants to bike from Jopper's shed to Whike Meeler's house along roads in her town, so together they can devour trick-or-treat candy. But she has to be careful: many enemy agents lie in wait to capture her and study her powers. Whenever she encounters an agent on a road, Lee must create a distraction using her psychic powers in order to escape, after which she must eat one piece of candy to replenish her energy. After Lee escapes, agents will not pursue her; they will remain at their assigned posts. In addition, Lee will trick-or-treat at any house on the **right side of any street**[4] she travels on, where she will gain one additional piece of candy. Once she is biking on a road in one direction, she will continue in that direction until reaching a road intersection, so biking along a road having a agents in a direction where h_R houses are on her right (and h_L on her left) will change her supply of candy by precisely $h_R - a$ pieces (or $h_L - a$ pieces if she travels the road in the opposite direction). Lee may trick-or-treat at a house more than once, but each time she does, she would need to bike along the **entire road** having that house on the right. Lee can start her journey with as much candy as she wants, and since candy is small, you may assume that there is no limit to the amount of candy Lee can carry. Lee knows where all the roads, houses, and agents are in the town (she is psychic after all). Describe an efficient algorithm to determine whether Lee can reach Whike's house with **strictly more candy** than when she started.

[4] Because left turns in the middle of a street are dangerous.

Solution: Define a **stretch of road** to be either a road between two intersections having no other intersection between them, or a road between an intersection and either Jopper's shed or Whike's house. Assume that Lee can provide the number of agents on any stretch of road and the number of houses on either side in constant time. Construct a graph G with a vertex for each of the n intersection in the town, and a vertex s for Jopper's shed and vertex t for Whike's house. Then for each of the m stretches of road between locations p and q, containing a agents and h_L (h_R) houses on the left (right) side of the street when biking from p to q, add a directed edge to G from p to q with weight $a - h_R$, and a directed edge from q to p with weight $a - h_L$. These weights correspond to the **negative** net change in candy resulting from Lee traveling down a corresponding stretch of road along a particular direction. Lee will be able to reach Whike's house with strictly more candy, if and only if the minimum weight of any path from s to t is strictly negative. Then we can use Bellman-Ford to compute $\delta(s, t) < 0$ in $O(nm)$ time.

Rubric:

- 5 points for construction of a relevant graph
- 4 points for description of a correct algorithm
- 2 points if correct algorithm runs in $O(nm)$
- 2 points for a correct running time analysis
- Partial credit may be awarded

Problem 7-6. [45 points] **Bingle!**

Parry Lage is developing an amazing new search engine called Bingle! Every night, he needs to build source files S (from scratch) into a final executable file t via a collection of **code transformations**. Example code transformations include:

- converting a Python source file into a Python byte code file,
- converting a Java source file and its dependencies into a class interface file,
- converting a CoffeeScript source file into a JavaScript source file,
- converting a C source file into object code, etc.

Each code transformation (F, o, m) contains:

- a list F of input files (each either an original source file or an output from another code transformation) that need to exist before the transformation can be done,
- a unique output file o generated by the transformation,
- and the number of milliseconds m it takes to run the transformation.

Luckily, Parry has an extremely large number of machines, so he can **simultaneously** run as many code transformations as he wants in parallel. Help Parry determine the minimum amount of time needed to run code transformations to produce a final executable.

(a) [10 points] Given a weighted directed acyclic graph $G = (V, E, w)$, describe a linear-time algorithm to find a **maximum**-weight path from any vertex in a subset $S \subset V$ to a vertex t.

Solution: Add a new vertex s' to G with a directed edge of weight zero from s; to each vertex in S, and then negate all edge weights, to construct a new graph G'. This graph is directed and acyclic because so was G. Let t' be the vertex in G' corresponding to the vertex t from G. Then we can compute a minimum-weight path π' in G' from s' to t' in $O(|V| + |E|)$ time using DAG shortest paths by relaxing edges from vertices in topological sort order. From path π', we can construct a path π in G by listing vertices from G that correspond to each vertex in G' from path π' in order, except for the starting vertex s' which corresponds to no vertex in G. Then path π is a maximum-weight path from any vertex from S to vertex t, since the second vertex visited by π' must correspond to a vertex from S, and negating edge weights changes maximization to minimization.

Rubric:

- 8 points for description of a correct algorithm
- 2 point for a correct running time analysis
- Partial credit may be awarded

(b) [10 points] Given a list of S original source files, a list T of transformations, and a target file t, show how to use part (a) to compute the minimum number of milliseconds needed to build target file t from the source files S.

Solution: Construct a graph G with: a vertex for each file name present in the source files and transformations, and a directed edge from file a to file b if file b is the output of some transformation τ having file a as an input, weighted by the time to perform transformation τ. It is reasonable to assume that file t is buildable and that building file t depends on every file mentioned in the input, which implies that G is acyclic and every source file is necessary to build t.

If you do not make this assumption, in order to build file t, every vertex having a path to t corresponds to a file that must be built by a transformation or exist as an original source file, while any other file does not need to be built. Remove all files from G that do not have a path to t in G, resulting in graph G'. We do this by reversing the direction of all edges in G, running BFS or DFS from t, and then removing any vertex from G not touched by the search. Now look at all vertices S' in G' that have no incoming edge. If any such vertex v does not correspond to a source file, then t is not buildable because v is also not buildable from the source files, and t depends on v. If $S' \subset S$, then t is buildable. We let G equal G' and proceed as follows.

Then given G, the minimum time to build t is limited by the maximum-weight of any path starting at some vertex from S ending at vertex t. We can then use the result from part (a) to return the minimum time to build t in linear time.

Rubric:

- 3 points for correctly constructing relevant graph
- 6 points for correct algorithm applying topological sort relaxation
- 3 points for correctness analysis
- 3 points for runtime analysis

(c) [25 points] Implement method `build_time` based on the algorithm you described in part (b). You can download a code template containing some test cases from the website. Submit your code online at `alg.mit.edu`.

Solution:

```
def dfs(Adj, s, parent = None, order = []):
    if parent is None:
        parent = [None for v in Adj]
        parent[s] = s
    for v in Adj[s]:
        if parent[v] is None:
            parent[v] = s
            dfs(Adj, v, parent, order)
    order.append(s)
    return parent, order

def topo_shortest_paths(Adj, w, s):
    _, order = dfs(Adj, s)
    order.reverse()
    d = [float('inf') for _ in Adj]
    parent = [None for _ in Adj]
    d[s], parent[s] = 0, s
    for u in order:
        for v in Adj[u]:
            if d[v] > d[u] + w(u, v):
                d[v] = d[u] + w(u, v)
                parent[v] = u
    return d, parent

def build_time(source_files, transformations, target_file):
    n, s = 0, len(source_files)
    file_idx = {}
    for f in source_files:
        file_idx[f] = n
        n += 1
    for (_, output, _) in transformations:
        file_idx[output] = n
        n += 1
    Adj = [[] for _ in range(n + 1)]
    w = {}
    for i in range(s):
        Adj[n].append(i)
        w[(n, i)] = 0
    for (input_files, output, time) in transformations:
        for f in input_files:
            f_idx, o_idx = file_idx[f], file_idx[output]
            Adj[f_idx].append(o_idx)
            w[(f_idx, o_idx)] = -1 * time
    dist, _ = topo_shortest_paths(Adj, lambda u,v: w[(u,v)], n)
    return -1 * dist[file_idx[target_file]]
```