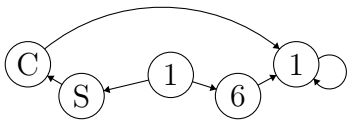
CS 161
Summer 2017

Problem Set 4
Due: July 28, 2017, 11:59 p.m.

Drawing graphs: You might try `http://madebyevan.com/fsm/` which allows you to draw graphs with your mouse and convert it into LaTeX code:



---

1. **Warmup with DFS/BFS.** (4 points)

   Give one example of a directed graph on four vertices, A, B, C, and D, such that both depth-first search and breadth-first search discover the vertices in the same order when started at A. Give one example of a directed graph where DFS and BFS discover the vertices in a different order when started at A. "Discover" means the time that the algorithm first reaches the vertex, referred to as `start_time` during lecture. Assume that both DFS and BFS iterate over outgoing neighbors in alphabetical order.

   [We are expecting a drawing of your graphs and an ordered list of vertices discovered by DFS and BFS.]

2. **Warmup with Dijkstra.** (9 points)

   Let $G = (V, E)$ be a weighted directed graph. For the rest of this problem, assume that $s, t \in V$ and that **there exists a directed path from s to t.** The weights on G could be anything: **negative, zero, or positive.**

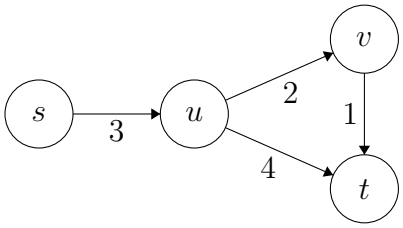
   For the rest of this problem, refer to the implementation of Dijkstra's algorithm given by the pseudocode below.

```
dijkstra_st_path(G, s, t):
  for all v in V, set d[v] = Infinity
  for all v in V, set p[v] = None

  // we will use p to reconstruct the shortest s-t path at the end
  d[s] = 0
  F = V
  D = []
  while F isn't empty:
    x = vertex v in F such that d[v] is minimized
    for y in x.outgoing_neighbors:
      d[y] = min( d[y], d[x] + weight(x,y) )
      if d[y] was changed in the previous line, set p[y] = x
    F.remove(x)
    D.add(x)

  // use p to reconstruct the shortest s-t path
  path = [t]
  current = t
  while current != s:
    current = p[current]
    add current to the front of the path
  return path, d[t]
```

Notice that the pseudocode above differs from the pseudocode in the notes. The variable `p` maintains the "**p**arents" of the vertices in the shortest s-t path, so it can be reconstructed at the end.

(a) (1 point) Step through `dijkstra_st_path`(G, s, t) on the graph G shown below. Complete the table below to show what the arrays `d` and `p` are at each step of the algorithm, and indicate what path is returned and what its cost is.

G



[We are expecting the table below filled out, as well as the final shortest path and its cost. No further justification is required.]

	$\texttt{d}[s]$	$\texttt{d}[u]$	$\texttt{d}[v]$	$\texttt{d}[t]$	$\texttt{p}[s]$	$\texttt{p}[u]$	$\texttt{p}[v]$	$\texttt{p}[t]$
When entering the first while loop for the first time, the state is:	0	∞	∞	∞	None	None	None	None
Immediately after the first element of D is added, the state is:	0	3	∞	∞	None	s	None	None
Immediately after the second element of D is added, the state is:								
Immediately after the third element of D is added, the state is:								
Immediately after the fourth element of D is added, the state is:								

(b) (1 point) **Prove or disprove**: In every such graph G, the shortest path from s to t exists. Here, a *path* from s to t is formally defined as a sequence of edges

$$(u_0, u_1), (u_1, u_2), (u_2, u_3), \ldots, (u_{M-1}, u_M)$$

such that $u_0 = s$, $u_M = t$, and $(u_i, u_{i+1}) \in E$ for all $i = 0, \ldots, M-1$. A *shortest path* is a path $((u_0, u_1), \ldots, (u_{M-1}, u_M))$ such that

$$\sum_{i=0}^{M-1} \text{weight}(u_i, u_{i+1}) \leq \sum_{i=0}^{M'-1} \text{weight}(u'_i, u'_{i+1})$$

for all paths $((u'_0, u'_1), \ldots, (u'_{M'-1}, u'_{M'}))$.

(c) (2 points) **Prove or disprove**: In every such graph G in which the shortest path from s to t exists, $\texttt{dijkstra\_st\_path}(G, s, t)$ returns a shortest path between s and t in G.

(d) (2 points) **Prove or disprove**: In every such graph G in which there is a negative-weight edge, and for all s and t, $\texttt{dijkstra\_st\_path}(G, s, t)$ does not return a shortest path between s and t in G.

(e) (3 points) Your friend offers the following way to patch up Dijkstra's algorithm to deal with negative edge weights. Let G be a weighted graph, and let w^* be the smallest weight that appears in G. (Notice that w^* may be negative). Consider a graph $G' = (V, E')$ with the same vertices, and such that E' is as follows: for every edge $e \in E$ with weight w, there is an edge $e' \in E'$ with weight $w - w^*$. Now all of the weights in G' are non-negative, so we can apply Dijkstra's algorithm to that:

```
modified_dijkstra(G,s,t):
  Construct G' from G as above.
  return dijkstra_st_path(G',s,t)
```

Prove or disprove: Your friend's approach will always correctly return a shortest path between s and t if it exists.

[We are expecting for each "prove or disprove," either a proof or a (small) counterexample. In the previous sentence, "(small)" means no more than 5 vertices.]

3. **Fun with Reductions.** (3 points)

(a) (3 points) Suppose the economies of the world use a set of currencies $C_1, \ldots, C_n$; think of these as dollars, pounds, Bitcoin, etc. Your bank allows you to trade each currency C_i for any other currency C_j, and finds some way to charge you for this service. Suppose that for each ordered pair of currencies (C_i, C_j), the bank charges a flat fee of $f_{ij} > 0$ dollars to exchange C_i for C_j (regardless of the quantity of currency being exchanged).

Devise an efficient algorithm which, given a starting currency C_s, a target currency C_t, and a list of fees f_{ij} for all $i, j \in \{1, \ldots, n\}$, computes the cheapest way (that is, incurring the least in fees) to exchange all of our currency in C_s into currency C_t. Also, justify the correctness of your algorithm and its runtime.

[We are expecting a description or pseudocode of your algorithm as well as a brief justification of its correctness and runtime.]

(b) (2 points extra credit) Consider a robot that can take steps of k distinct lengths $s_1, s_2, \ldots, s_k \in \mathbb{Z}$ in feet. The robot starts at position p_0 on a circular track with circumference length $T \in \mathbb{N}$ in feet and it starts walking counter-clockwise around the track to calibrate itself. It takes one step for each of the lengths $s_1, s_2, \ldots, s_k$, bringing it to position p_1.

Describe an $O(T + kT)$-time algorithm that finds the fewest number of steps that the robot needs to take to get back to position p_0, assuming the robot can't "turn around" but can walk backwards (all steps $s_i > 0$ are taken in the counter-clockwise direction and all steps $s_i < 0$ are taken in the clockwise direction).

[We aren't expecting anything; this is a bonus problem.]

4. **Another topological sort algorithm.** (7 points)

   Consider a **complete directed acyclic graph** $G = (V, E)$ where $|V| = n$ and $|E| = \frac{n(n-1)}{2}$. Since the graph is complete, there exists an edge between each pair of the vertices $u, v \in V$ such that either $(u, v) \in E$ or $(v, u) \in E$ but not both (since the graph is acyclic).

   Suppose we define a function `contains(u,v)` which returns `True` if $(u, v) \in E$ and `False` if $(v, u) \in E$. Assume that we do not have access to the list of edges and can only access the graph through the `contains` function. (We do have access to the list of vertices though.)

   (a) (1 point) Give an asymptotically tight lower bound on the worst-case number of calls to `contains(u,v)` required to find a topological sort of G.
   [We are expecting an asymptotic tight lower bound like $\Omega(\ldots)$.]

   (b) (3 points) Prove that the bound is tight, i.e., that no algorithm can find a topological sort of G with fewer than that many calls to `contains(u,v)` in the worst-case.
   [We expecting a convincing argument.]

   (c) (3 points) Describe an algorithm that achieves this bound.
   [We are expecting a description or pseudocode of your algorithm.]

5. **Social engineering.** (9 points)

   Suppose we have a community of n people. We can create a directed graph from this community as follows: the vertices are people, and there is a directed edge from person A to person B if A would forward a rumor to B. Assume that if there is an edge from A to B, then A will always forward any rumor they hear to B. Notice that this relationship isn't symmetric: A might gossip to B but not vice versa. Suppose there are m directed edges total, so $G = (V, E)$ is a graph with n vertices and m edges.

   Define a person P to be *influential* if for all other people A in the community, there is a directed path from P to A in G. Thus, if you tell a rumor to an influential person P, eventually the rumor will reach everybody. You have a rumor that you'd like to spread, but you don't have time to tell more than one person, so you'd like to find an influential person to tell the rumor to.

   In the following questions, assume that G is the directed graph representing the community, and that you have access to G as an array of adjacency lists: for each vertex v, in $O(1)$ time you can get a pointer to the head of the linked lists v`.outgoing_neighbors` and v`.incoming_neighbors`. Notice that G is not necessarily acyclic. In your answers, you may appeal to any statements we have seen in class, in the notes, or in CLRS.

   (a) (1 point) Show that all influential people in G are in the same strongly connected component, and that everyone in this strongly connected component is influential.
   [We are expecting a short but formal proof.]

   (b) (5 points) Suppose that an influential person exists. Give an algorithm that, given G, finds an influential person in time $O(n + m)$.
   [We are expecting a description or pseudocode of your algorithm, a proof of correctness, and a short argument about the runtime.]

(c) (3 points) Suppose that you don't know whether or not an influential person exists. Use your algorithm from part (b) to give an algorithm that, given G, either finds an influential person in time $O(n+m)$ if there is one, or else returns "no influential person."

[We are expecting a description or pseudocode or your algorithm.]

6. **How is the course so far?** (0.5 points extra credit)

   Please complete the poll at `https://goo.gl/forms/8cKzJ2u08evIc6EM2`.