Algorithms & Data — Problem Set 2
(due 11:59pm, Friday February 12th)

Instructions:

- This assignment should be submitted via Blackboard. Late assignments will not be accepted.
- You are encouraged to solve the problems on your own. You are permitted to study with friends and discuss the problems; however, *you must write up your own solutions, in your own words.*
- If you do collaborate with any of the other students on any problem, you must list all the collaborators in your submission for each problem.
- Finding solutions to homework problems on the web, or by asking students not enrolled in the class (or the class staff) is strictly prohibited.
- We require that all homework submissions be neat, organized, and *typeset*. You may use plain text or a word processor like Microsoft Word or LaTeX for your submissions. If you need to draw any diagrams, however, you may draw them with your hand.

1. Adjacency Lists (10 pts)

The reverse of a directed graph $G = (V, E)$ is another directed graph $G^R = (V, E^R)$ on the same vertex set, but with all edges reversed; that is, $E^R = \{(v, u) : (u, v) \in E\}$. Give a linear time algorithm to compute the adjacency list for G^R.

Solution: Denote by L_G the original adjacency list of the graph $G = (V, E)$. Technically, L_G is an array of linked lists; each entry in the array corresponds to a vertex $v \in V$ and holds the pointer to the linked list of v's neighbors in G. Our task it to build an analogous array L_{G^R} for the graph G^R.

1. First, we create an array L_{G^R} of length $|V|$ whose entries correspond to vertices in V and hold pointers to initially empty linked lists.
2. We scan the original array L_G entry by entry, that is, linked list by linked list. In particular, for each vertex $v \in V$, we traverse the list L_v of $v's$ neighbors. A vertex w encountered in L_v signifies the fact that $(v, w) \in E$. Hence
   (a) we locate the entry corresponding to w in L_{G^R}
   (b) we append v to the list L_w of w's neighbors in G^R. Our doing so will encode in L_{G^R} the fact that $(w, v) \in E^R$.

(c) After L_G has been so traversed, L_{G^R} will form an adjacency list for G^R.

Note that the operation under point 1. takes $O(n)$ time. The operation 2 a) takes O(1) time because L_{G^R} is an array and locating an entry corresponding to a given vertex takes constant time. Appending to the end of a given linked list in 2 b) is a constant time operation as well. Hence the running time of this algorithm is proportional to the number of items encountered during the scan of L_G, i.e., $O(|V| + |E|)$.

---

2. Degrees (20 pts = 10+5+5)

True or false? Prove the given statement or give a counterexample showing that it is untrue.

i. An undirected graph G on 2016 vertices is given in which every vertex has the degree of at least 1008. G is connected.

Solution: True. Suppose for a moment that G is not connected. We will show that this assumption leads to a contradiction. If G is not connected, then it consists of a least 2 connected components and one of them, say C, must contain at most 1008 vertices (if all the components contained $>$ 1008 vertices, we would have $>$ than 2 *1008=2016 vertices in G—a contradiction). Because C is a connected component of G with at most 1008 vertices, each of its vertices can be connected to at most 1007 other vertices of C and no other vertices in G. We know, however, that every vertex in G has degree of at least 1008. Thus, our assumption that G is not connected has led to a contradiction.

---

ii. Let G be a graph in which every vertex has the degree ≥ 2. G contains a cycle.

Solution: True. Pick a vertex $x_1 \in G$. Since x_1 has the degree ≥ 2, we can follow any of the edges adjacent to it and arrive at a new vertex x_2. Since $deg(x_2) \geq 2$, we can follow an edge adjacent to x_2 which is different from the one we followed to arrive at x_2. That will lead us to a new vertex x_3 which we can then leave through an edge different from that that led us to x_3. Proceeding in this way, we see that the fact that every vertex has degree ≥ 2 allows us to generate a path $P = (x_1, x_2, x_3, \ldots)$ which must self intersect at some point as otherwise we would have infinitely many vertices in G. The first

self-intersection of P creates a cycle which shows that the statement ii. is true.

---

iii. Let G be a graph with no cycles. G contains a vertex of degree ≤ 1.

Solution: True. This a consequence of part ii. If there was no vertex of degree ≤ 1 in G, then every vertex would have the degree ≥ 2. By part ii. we would conclude then that G has a cycle—a contradiction with the assumptions given in the problem.

---

3. Paths (15 pts)

Consider an undirected graph $G = (V, E)$ on 2016 vertices containing vertices x, y whose distance is 1009 (i.e., the shortest path joining x, y has length 1009). Show that there is a vertex $z \in V, z \neq x, y$ with the property that any other path joining x, y passes through z.

Solution: We will use the following fact proved in the textbook:

> **Theorem.** Let $G = (V, E)$ be a connected graph and $x \in V$. Suppose that we execute the BFS algorithm starting at vertex x and obtain a tree T and a decomposition of V into a union of disjoint layers $V = L_0 \cup \cdots \cup L_k$. Then for every edge (u, v) in G we have:
>
> $$\text{if } u \in L_i \text{ and } v \in L_j \implies |i - j| \leq 1$$

This theorem states that any edge in G connects vertices of the same or consecutive layers. We have the following consequence.

> **Corollary.** Suppose that $G = (V, E)$, $x \in V$ and BFS has been run from x as above. Consider a path P starting at x which contains a vertex belonging to some layer L_i. Then P contains vertices belonging to all intermediate layers $\{0, 1, \ldots, i\}$.

Proof.

1. Let $0 < j < i$ be the very first layer that P skips. Consider the set P_1 of vertices on P belonging to $L_0 \cup \cdots \cup L_{j-1}$ and the set P_2 of vertices on P belonging to $L_{j+1} \cup \cdots \cup L_i$. Because P has no vertices in L_i, we have $P = P_1 \cup P_2$.

2. Since P connects the two sets, there must be an edge $e = (u, v) \in P$ such that $u \in P_1$ and $v \in P_2$.

3. u, v must belong to some layers L_s and L_t, respectively. By the definition of u, v, we must have $s \leq j - 1$ and $t \geq j + 1$.

4. By the theorem above, however, the layers of the vertices on the edge $e = (u, v)$ must satisfy $|s - t| \leq 1$ — a contradiction. $\square$

Having the two facts established, we can now solve the problem at hand. Let's launch the BFS algorithm from the vertex x in the given graph G. The algorithm will reach the vertex y and place it in the layer L_{1009} (because that layer is the locum of all vertices whose distance from x is 1009).

There must an intermediate layer $L_j, 0 < j < 1009$, containing just one vertex (otherwise, we would have 1008 layers $L_1, \ldots, L_{1008}$ with at least two vertices; those layers together with x and y (belonging to L_0 and L_{1009} respectively) would make G posses 2010 vertices — a contradiction). Let's call the single vertex belonging to that layer z.

Every path P joining x and y must pass through z. Indeed, by the corollary above all layers $\{0, 1, ..., 1009\}$ are represented on P. However, the layer j contains only vertex z, so z must belong to P.

---

4. Connectivity (25 pts = 10+5+10)

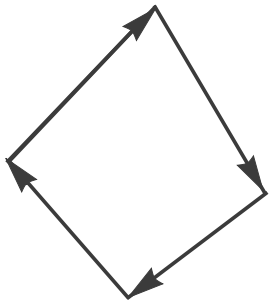
i. Prove that in any connected undirected graph $G = (V, E)$ there is a vertex $v \in V$ whose removal leaves G connected. (Hint: Consider the DFS search tree for G.)

Solution: Let x be the leaf (a vertex of degree 1) of the tree T produced by DFS (we know that since T has no cycles, such a vertex must exist). Removal of x from the graph G (and the edges adjacent to x), result in T's loosing only one edge — the edge joining x with its parent in T. Since T connects all the remaining vertices in G, $G \setminus \{x\}$ is connected.

---

ii. Give an example of a strongly connected directed graph $G = (V, E)$ such that, for every $v \in V$, removing v from G leaves a directed graph that is not strongly connected.

Solution: Here is a simple example:

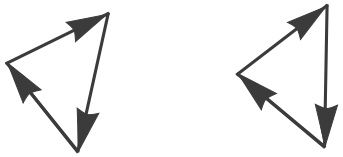


For every two vertices u, v in this grapth there is a path $u \rightsquigarrow v$ as well as $v \rightsquigarrow u$. After removal of any vertex, however, there is a path between any remaining pair of vertices only in one direction (hence strong connectivity has been lost).

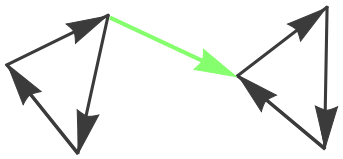
---

iii. In an undirected graph with 2 connected components it is always possible to make the graph connected by adding only one edge. Give an example of a directed graph with two strongly connected components such that no addition of one edge can make the graph strongly connected.

Solution:



The only edges that could join the two components and potentially make the resulting graph strongly connected start in one of the triangles and end in the other. Once we choose the direction of such an egde, going in the opposite direction will not be possible. Hence the vertices in the triangle that the added edge leaves will not be reachable from the triangle that the edge enters.



5. Consistency of constraints (15 pts = 5+10)

The following problem occurs in program analysis. For a set of n variables $x_1, \ldots, x_n$, you are given a set of m constraints, of one of two forms: *equality*, of the form $x_i = x_j$; *strict inequality*, of the form $x_i > x_j$. The goal of this problem is to determine whether a given set of constraints can be satisfied. For example, the following set of constraints can be satisfied.

$$x_1 = x_2; x_1 = x_3; x_2 > x_4; x_3 > x_5$$

One can set $x_1 = x_2 = x_3 = 1$, $x_4 = 0$, and $x_5 = 0$. The following set of constraints, however, cannot be satisfied.

$$x_1 = x_2; x_1 = x_3; x_2 > x_4; x_4 > x_5; x_5 > x_3$$

This is because the constraints yield the contradiction $x_2 > x_4 > x_5 > x_3 = x_2$.

i. Can the following set of constraints be satisfied?

$$x_2 = x_4; x_1 = x_3; x_5 = x_6; x_2 > x_3; x_1 > x_5; x_6 > x_4$$

ii. Give an $O(m + n)$ algorithm that takes as input m constraints over n variables and determines whether the constraints can be satisfied. Your algorithm only needs to return a "yes/no" answer ("yes" if the constraints can be satisfied; and "no" otherwise).

Justify your answers. *Hint.* One possible approach is to make use of a known $O(m + n)$ algorithm which identifies all strongly connected components in a directed graph. You can use that algorithm as a "black box".

Solution: Create a directed graph G whose vertices are the variables $x_1, \ldots, x_n$; for every equality constraint $x_i = x_j$ create edges in both directions (x_i, x_j) and (x_j, x_i); for every strict inequality constraint $x_i > x_j$

create a single edge (x_i, x_j). Creation of this graph takes $O(m+n)$ time.

The key observation is that

Fact. A system of constraints described above is satisfiable if and only if the strongly connected components of G do not contain any strict inequality edge.

Proof:
a) Existence of a path $x_p \rightsquigarrow x_q$ in G means that $x_p = x_q$ or $x_p > x_q$. Hence if a strict inequality edge (x_i, x_j) belonged to a strongly connected component of G, we would have on the one hand $x_i > x_j$, and on the other, there would be a path $x_j \rightsquigarrow x_i$ indicating that $x_j \geq x_i$ — a contradiction.

b) If strongly connected components of G contain only equality edges, then each component's vertices stand for the same number. The satisfiability of the constraints is reduced then to the component graph of G (the graph in which every strongly connected component of G has been reduced to a vertex) being acyclic. The component graph of a strongly connected graph is always acyclic (a cycle would connect several components into a larger strongly connected component). Hence the components of G can be topologically ordered, and the vertices in each component assigned the same number. □

Our algorithm proceeds as follows:

1. First, run the $O(m+n)$ algorithm which identifies all strongly connected components in G.
2. For every edge in G we check whether
   (a) the edge is a strict inequality
   (b) if yes, check whether the edge's ends belong to the same strongly connected component. If they do, the set of constraints is not satisfiable.

If the check 2 b) never returns an inequality edge in the same strongly connected component, the set of constraints is satisfiable. Clearly, 1 and 2 run in $O(m+n)$ time.
