CS180 Homework 2

Due: April 29, 11:59pm

1. (20 pt) (Noor) Prove or disprove the following statement:

   For a graph G with n nodes (n is even), if every node of G has degree at least $n/2$, then G is connected.

   Solution: This is true.
   If a graph is disconnect, there will be a component with $\geq n/2$ nodes, then any node in another component will have degree $< n/2$.
   Another way of proving through contradiction is to assume that G is not connected i.e. it has at least two components, call them c_1 and c_2. Since every node must have degree of at least $\frac{n}{2}$, a node in c_1 is connected to at least $\frac{n}{2}$ other nodes i.e. there are at least $\frac{n}{2}+1$ nodes in c_1. Similarly, in c_2 there must be at least $\frac{n}{2}+1$ nodes. This gives total number of nodes $\frac{n}{2}+1+\frac{n}{2}+1 = n+2$ which is a contradiction since we have only n nodes. Hence G must be connected.

2. (20 pt) (Xiangning) A number of stories in the press about the structure of the Internet and the Web have focused on some version of the following question: How far apart are typical nodes in these networks? If you read these stories carefully, you find that many of them are confused about the difference between the *diameter* of a network and the *average distance* in a network; they often jump back and forth between these concepts as though they're the same thing.

   As in the text, we say that the distance between two nodes u and v in a graph $G = (V, E)$ is the minimum number of edges in a path joining them; we'll denote this by $dist(u, v)$. We say that the diameter of G is the maximum distance between any pair of nodes; and we'll denote this quantity by $diam(G)$.

   Let's define a related quantity, which we'll call the *average pairwise distance* in G (denoted $apd(G)$). We define $apd(G)$ to be the average, over all $\binom{n}{2}$ sets of two distinct nodes u and v, of the distance between u and v. That is,

$$apd(G) = \left[\sum_{\{u,v\}\subseteq V} dist(u, v)\right] / \binom{n}{2}.$$

   Here's a simple example to convince yourself that there are graphs G for which $diam(G) \neq apd(G)$. Let G be a graph with three nodes u, v, w, and with the two edges $\{u, v\}$ and $\{v, w\}$. Then

$$diam(G) = dist(u, w) = 2,$$

   while

$$apd(G) = [dist(u, v) + dist(u, w) + dist(v, w)]/3 = 4/3.$$

   Of course, these two numbers aren't all *that* far apart in the case of this three-node graph, and so it's natural to ask whether there's always a close relation between them. Here's a claim that tries to make this precise.

   Claim: There exists a positive natural number c so that for all connected undirected graphs G (with arbitrary number of nodes), it is the case that

$$\frac{diam(G)}{apd(G)} \leq c.$$

   Decide whether you think the claim is true of false, and give a proof of either the claim or its negation.

Solution: There doesn't exist a constant c for any n (we can construct a graph to make the ratio arbitrarily large). For example, consider a graph with fully connected n nodes extended with c nodes on it (like a lollipop). When n is large enough, the average approaches 1, so the ratio will be arbitrarily close to $c+1$ (so the ratio $> c$ when n large enough). We can then control c to make the ratio arbitrarily large.

Consider there is n fully connected nodes and c extended nodes. Then $diam(G)$ would be $c+1$, which is the distance from one of the $n-1$ nodes in the fully connected part (exclude the one connecting with the extended nodes) to the tail of extended part. The $apd(G) = \frac{\binom{n}{2}+nf(c)}{\binom{n}{2}}$ since the distance between every node pair of the fully connected part is 1 and f(c) is a function only related to c. So this value would approach 1 when n grows arbitrarily large. $\frac{diam(G)}{apd(G)}$ approaches $c+1$, so there does not exist a constant c.

3. (25 pt) (Kevin) We have defined the diameter of a graph in Problem 2. Finding the diameter of a graph is not an easy problem on a general graph, but we consider a special case when G is a tree:

   (a) Assume G is a tree (undirected, connected graph without any cycle). Assume we are given a function $A(\cdot)$ such that given an input u, it will return a node v that maximizes dist(u, v) and the corresponding distance. There exists an algorithm that can compute the diameter of a tree using a constant calls of the function A without using any other graph traversal algorithms (e.g., BFS/DFS). Try to design such algorithm and prove the correctness of your algorithm.

   (b) Which graph traversal algorithm we introduced in the class can be used for implementing the function A? And what will be the overall time complexity for this diameter computing algorithm for trees in part (a)?

Solution: a) Pick an arbitrary node u and run $A(u)$ to get the farthest node v from u. Then run $A(v)$ to find the farthest point w from v. The distance v, w will then be the diameter of the tree.

To show this, we show that for any random diameter path xy, there is another diameter path that has v as an endpoint. This means that $v, A(v) = w$ is a diameter path, since it's the longest path starting v, and one such path is a diameter path.

To prove this, we aribtrarily root the tree at u (so v will be a node with maximum height). Now consider some arbitrary diameter path xy. We by considering two cases (1) path xy goes through u, and (2) path xy doesn't go through u.

(Notation: Here xy denotes a path from x to y, and saying xzy means xz then zy, with no backtracking, i.e. z is a node on xy. Then this means $|xzy| = |xz| + |zy|$)

For the first case, we have xy goes through u, i.e. it goes xuy. By construction, $|uy| \leq |uv|$ by A, so path $|xuv| \geq |xuy|$. Hence xuv must also be a diameter path.

Otherwise, xy doesn't go through u. Then there is some point z on the path that is the closest to u, meaning our path can be written as xzy. Now consider uv. For contradiction, suppose uv does not go through z. That means that after xz, the path $|zuv| = |zu| + |uv| \geq |uy|$. And hence $xzuv = xv$ is a longer path than xy, contradiction. Therefore, uv goes through z. Note that x and y must be beneath two different children of z, call them x' and y'. If they were beneath the same child of z, i.e. $x' = y'$, then the path would go x' to z and back to $y' = x'$, which can't happen. Then, the path from v to z can go through at most one of these children $x' \neq y'$ of z. Without loss of generality, suppose vz does not go through x'. Then since $|uv| = |uzv| \geq |uy| = |uzy|$, this means $|zv| \geq |zy|$. Therefore after xz, it's at least as long to go zv as zy, so xv is at least as long as xy, and so xv is a diameter path. (We have to argue x' and y' to make sure that xzv does not backtrack).

So there always exists a diameter path going through v, and hence $A(v)$ find a diameter path.

b) A(u): Run BFS(u), and return a point on the last layer, since BFS provides shortest distance (aka actual distance) to the root. The distance is just the layer number. A runtime O(m + n), so with constant calls the total runtime is O(m + n).

4. (20 pt) (Xuanqing) You're helping a group of ethnographers analyze some oral history data they've collected by interviewing members of a village to learn about the lives of people who've lived there over the past two hundred years.

From these interviews, they've learned about a set of n people (all of them now deceased), whom we'll denote $P_1, P_2, \ldots, P_n$. They've also collected facts about when these people lived relative to one another. Each fact has one of the following two forms:

- For some i and j, person P_i died before person P_j was born; or
- for some i and j, the life spans of P_i and P_j overlapped at least partially.

Naturally, they're not sure that all these facts are correct; memories are not so good, and a lot of this was passed down by word of mouth. So what they'd like you to determine is whether the data they've collected is at least internally consistent, in the sense that there could have existed a set of people for which all the facts they've learned simultaneously hold.

Give an efficient algorithm to do this: either it should produce proposed dates of birth and death for each of the n people so that all the facts hold true, or it should report (correctly) that no such dates can exist–that is, the facts collected by the ethnographers are not internally consistent. Assume there are n people and m facts, your algorithm should run in $O(n+m)$ time.

Solution: For each person P_i, construct two nodes u_i, v_i where u_i will indicate P_i's birth date and v_i is the death date. We then construct a directed graph according to the constraints:

(a) Draw a directed edge from u_i to v_i for all $i = 1, 2 \ldots, n$.

(b) If P_i dies before P_j was born, then we add an edge (v_i, u_j), indicating $v_i \leq u_j$.

(c) If the life spans of P_i and P_j overlapped, then we know $u_i \leq v_j$ and $u_j \leq v_i$, so we add two edges (u_i, v_j) and (u_j, v_i).

Eventually we get a graph G of $2n$ vertices and $\leq n+2m$ edges. After constructing the graph G, we run the topological sort on G to decide whether this is a DAG (this takes $O(n+m)$ time). If a cycle is found, then it means that there are contradictions among the facts; otherwise we output the nodes in topological order.

5. (15 pt) (Yuanhao) Some of your friends have gotten into the burgeoning field of *time-series data mining*, in which one looks for patterns in sequences of events that occur over time. Purchases at stock exchanges–what's being bought are one source of data with a natural ordering in time. Given a long sequence S of such events, your friends want an efficient way to detect certain "patterns" in them–for example, they may want to know if the four events

   `buy Yahoo, buy eBay, buy Yahoo, buy Oracle`

   occur in this sequence S, in order but not necessarily consecutively.

   They begin with a collection of possible events (e.g., the possible transactions) and a sequence S of n of these events. A given event may occur multiple times in S (e.g., Yahoo stock may be bought many times in a single sequence S). We will say that a sequence S' is a subsequence of S if there is a way to delete certain of the events from S so that the remaining events, in order, are equal to the sequence S' . So, for example, the sequence of four events above is a subsequence of the sequence

   `buy Amazon, buy Yahoo, buy eBay, buy Yahoo, buy Yahoo, buy Oracle`

   Their goal is to be able to dream up short sequences and quickly detect whether they are subsequences of S. So this is the problem they pose to you: Give an algorithm that takes two sequences of events–S' of length m and S of length n, each possibly containing an event more than once–and decides in time $O(m+n)$ whether S' is a subsequence of S.

Solution: Let the sequence S consist of $s_1, \ldots, s_n$ and the sequence S' consist of $s'_1, \ldots, s'_m$. We give a greedy algorithm that finds the first event in S that is the same as s'_1, matches these two events, then finds the first event after this that is the same as s'_2, and so on. We will use $k_1, k_2, \ldots$ to denote the match we have found so far, i to denote the current position in S, and j the current position in S'.

The running time is $O(n)$: one iteration through the while loop takes $O(1)$ time, and each iteration increments i, so there can be at most n iterations.

```
Initially i = j = 1
While i ≤ n and j ≤ m
    If s_i is the same as s'_j, then
        let k_j = i
        let i = i + 1 and j = j + 1
    otherwise let i = i + 1
EndWhile
If j = m + 1 return the subsequence found:  k_1, ..., k_m
Else return that "S' is not a subsequence of S"
```

Figure 1: Algorithm for Problem 5.

★ Homework assignments are due on the exact time indicated. Please submit your homework using the Gradescope system. Email attachments or other electronic delivery methods are not acceptable. To learn how to use Gradescope, you can:

- 1. Watch the one-minute video with complete instructions from here: https://www.youtube.com/watch?v=-wemznvGPfg
- 2. Follow the instructions to generate a PDF scan of the assignments: http://gradescope-static-assets.s3-us-west-2.amazonaws.com/help/submitting_hw_guide.pdf
- 3. **Make sure you start each problem on a new page.**

★ We recommend to use LaTeX, LyX or other word processing software for submitting the homework. This is not a requirement but it helps us to grade the homework and give feedback. For grading, we will take into account both the correctness and the clarity. Your answer are supposed to be in a simple and understandable manner. Sloppy answers are expected to receiver fewer points.

★ Unless specified, you should justify your algorithm with proof of correctness and time complexity.