(1) (12 + 30 + 3 = 45 pts.)

Consider the following decision problem:

**EDGE-DISJOINT PATHS**

**INSTANCE:** A directed graph $G$ and nodes $s_1, s_2, t_1, t_2 \in V(G)$.

**QUESTION:** Do there exist a path $P_1$ from $s_1$ to $t_1$ and a path $P_2$ from $s_2$ to $t_2$ such that no edge connects a vertex used in $P_1$ to a vertex used in $P_2$?

(a) Find and explain the flaw(s) in this attempt to classify the problem into $\mathsf{NL}$: “Design an NTM $N$ with two worktapes used to guess the next vertex $u$ in the path $P_1$ and the next vertex $v$ in the path $P_2$. Initially $u = s_1$ and $v = s_2$, and $N$ can also write $t_1$ and $t_2$ on separate tapes as the targets without violating the $O(\log n)$ space bound. Using one or two more tapes to maintain indices $i, j$ of nodes in the input graph, $N$ can guess a $u'$ that has an out-edge from $u$ as the next step of $P_1$, and then guess a $v'$ going out from $v$ as the next step of $P_2$. Finally, $N$ can go once more to the input graph to check if there is an edge between $u'$ and $v'$ (in either direction). If so, then the current nondeterministic branch dies (maybe some other branch will make luckier guesses); while if this never happens and we eventually get $u' = t_1$ and $v' = t_2$ then $N$ accepts.”

(b) Prove that this problem is $\mathsf{NP}$-complete, using a reduction from 3SAT. You must use a standard “rungs and clause gadgets” type architecture, though it will be different from the “size $k$” problems used in lectures. It will need some extra framework so that you can run one path “up the ladder” and the other path through the clause gadgets.

(c) Name one or more “drastic” consequences if the language of this problem were to belong to $\mathsf{NL}$ after all.

(d) Added: And for 15 points extra credit, explain why the logical analysis of your answer to (b) would go wrong if $G$ is an undirected graph.

**Answer:**

(a) One flaw is that the algorithm is only checking for shared edges that are the *same* step $m$ in both paths. (The diagram on the board showed an edge that was step 1 on one path and step 2 on the other.) A second flaw is that the algorithm is not checking that the vertex sets $\{u_m\}$ and $\{v_m\}$ used by the respective paths are completely distinct.

(b) First to show the problem is in $\mathsf{NP}$, when the answer is “yes” we need only guess $P_1$ and $P_2$ and verify that (their vertex sets are distinct and) no edges cross from one path to the other.

Given a 3CNF formula $\phi = C_1 \land \cdots \land C_m$, build an undirected graph $G_\phi$ as follows: In the “left half,” allocate one node for each literal $x_i$ and $\overline{x_i}$, $i = 1$ to $n$. Connect $s_1$ to both $x_1$ and $\overline{x_1}$, each $x_i$ to both $x_{i+1}$ and $\overline{x_{i+1}}$, each $\overline{x_j}$ ditto for $i \leq n - 1$, and $x_n$ and $\overline{x_n}$ both to $t_1$. Then each path from $s_1$ to $t_1$ has no incentive to “double back” to any previous literal, so without loss of generality $P_1$ has exactly $n + 1$ steps and corresponds to a unique truth assignment to the variables.

On the “right half,” connect $s_2$ to the first of $m$ “clause triads” of three nodes each. Each clause-triad node connects to each of the ltree nodes in the next triad, and the bottom
triad’s nodes connect to \( t_2 \). Each triad node is labeled by a literal \( x_i \) or \( \overline{x}_i \) occurring in the corresponding clause. Finally we add “crossing edges” from \( x_i \) in a clause triad to \( \overline{x}_i \) in the \( i \)-th “rung” of the “left half,” and \( \overline{x}_i \) in a clause similarly to \( x_i \) in a rung. This finishes the construction of \( G_\phi \) and the distinguished nodes \( s_1, s_2, t_1, t_2 \). Here is an example for a formula used often in lectures:

\[
\phi = (x_1 \lor \overline{x}_2 \lor x_3) \land (x_1 \lor x_2 \lor \overline{x}_3) \land (\overline{x}_1 \lor \overline{x}_3 \lor \overline{x}_4)
\]

Note how the assignment 0010 blocks the second clause.

**Complexity:** The graph has size \( 2n + 3m + 4 \) nodes and is a one-pass translation from the formula (so in fact it is clearly a log-space reduction).

**Correctness:** Given a satisfying assignment \( a \), we can make \( P_1 \) follow \( a \). Then \( P_2 \) can dodge all the edges radiating out from nodes in \( P_1 \) by always selecting the node in each clause triad that corresponds to a literal made true by \( a \). Conversely, given \( P_1 \) and a \( P_2 \) dodging it among the clause gadgets, the assignment corresponding to \( P_1 \) must satisfy each clause at the point where \( P_2 \) goes through its triad. Thus the answer to the mapped instance of Edge Disjoint Paths is “yes” if and only if \( \phi \) is satisfiable, so \( 3\text{SAT} \leq^p \text{ Edge Disjoint Paths} \) which is thus \( \text{NP} \)-complete. (The crossing edges cannot be used by \( P_2 \) because they are one-way and \( P_2 \) would conflict with \( P_1 \) from then on.)

(c) If Edge Disjoint Paths were in \( \text{NL} \) then we would have \( \text{NP} = \text{NL} \), which implies \( \text{NP} = \text{P} \).

(d) If the crossing edges (in particular) are allowed to be two-way, then the paths can “cheat” as follows: Suppose we have an instance where the first clause \( C_1 \) and last clause \( C_m \) have the same variable \( x_5 \) and the two middle clauses have \( x_1 \) and \( x_n \), respectively. Then \( P_1 \) can jump from \( x_1 \) to those middle clauses and out to \( x_n \) and \( t_1 \). Meanwhile, \( P_2 \) can jump from \( C_1 \) to \( x_5 \) (or to \( \overline{x}_5 \) depending on how you route the crossing edges) and then straight to \( C_m \) and exit at \( t_2 \). This can happen regardless of whether the (rest of the) original formula \( \phi \) is satisfiable, so it destroys the correctness property.

[In fact, with undirected edges, and for any \( k \geq 2 \), the problem of whether there are \( k \) edge-disjoint paths belongs to \( \text{P} \). The reason is deep. The property of not having such paths...}
is closed under the operations of (a) deleting an edge and (b) contracting an edge, meaning that the edge’s two vertex endpoints become one vertex and join their neighborhoods together. Every such property is known to belong to roughly cubic time—but often with huge constants. If you are curious, see https://en.wikipedia.org/wiki/Robertson-Seymour_theorem. So there is no way to fix the reduction without using directed edges somewhere. This is actually an error in the ALR notes, not caught until now!

(2) $3 \times 6 = 18$ pts. total

A collection $\{C_i\}$ of complexity classes forms a proper hierarchy if given any $C_i$ and $C_j$ with $i \neq j$, one of them is properly contained in the other. Which of the following collections are proper hierarchies? Justify your answers, mainly by verifying the relevant “little-o” or “$\Theta$” relations between time bounds. Here $Q^+$ stands for the positive rational numbers.

(a) $\{\text{DTIME}[n^c] : c \in Q^+, c \geq 1\}$. Not graded: see presentation option (2)
(b) $\{\text{DTIME}[(n + c)^3] : c \in Q^+, c \geq 1\}$.
(c) $\{\text{DTIME}[2^n] : c \in Q^+\}$.
(d) $\{\text{DTIME}[2^{n^{1/c}}] : c \in Q^+, c \geq 1\}$.

Answer: All of the functions involved are time constructible, since $c$ and $d$ are rational. Hence we need only check, for each case of a time bound $t(n, c)$ and given $c < d$, whether $t(n, c) \log t(n, c) = o(t(n, d))$, i.e. whether $t(n, c) \log t(n, c)/t(n, d) \to 0$ as $n \to \infty$. For convenience we assume that logs are to base $e$; in none of these cases does the base of the logarithms matter.

(a) $\{\text{DTIME}[n^c] : c \in Q^+, c \geq 1\}$: We get $n^c \log(n^c)/n^d = c(\log n)/n^{d-c}$. One application of L’Hôpital’s Rule turns the ratio into $(c'/n)/(d-c)n^{d-c-1}$, where $c' = c$ give-or-take a factor of $\log_2 e$ I-forget-which, and this equals $(c'/(d-c))$ times $1/(n \cdot n^{d-c-1})$. Ignoring the constants leaves $1/n^{d-c}$, which goes to $0$ since $d > c$. So this is a proper hierarchy.

(b) $\{\text{DTIME}[(n + c)^3] : c \in Q^+, c \geq 1\}$: For all $c$ and $d$, $(n + c)^3$ and $(n + d)^3$ are both $\Theta(n^3)$, since the highest-powered term is $n^3$. By the “Linear Speed-Up Theorem,” all of these classes are equal, so there is no proper hierarchy.

(c) $\{\text{DTIME}[2^n] : c \in Q^+\}$: We get $2^n \cdot cn/2^{dn} = cn/2^{(d-c)n}$. A trip to the L’Hôpital eliminates the numerator and leaves the denominator $1/2^{(d-c)n}$ times some constants, and again this $\to 0$ as $n \to \infty$. So this is a proper hierarchy.

(d) $\{\text{DTIME}[2^{n^{1/c}}] : c \in Q^+, c \geq 1\}$: Note that here higher $d$ makes the time bound smaller, so we get $n^{1/d}/2^{n^{1/c}} = n^{1/d}/2^{n^{1/c}-n^{1/d}}$. L’Hôpital gets messy here, but a permissible handwave is to reason that for all $d > c$, there exists an $n_0$ such that for all $n > n_0$, $n^{1/d} < n^{1/c}/2$. For such $n$ the ratio is bounded above by $n^{1/d}/2^{n^{1/c}/2}$. Since $d > 1$ this is certainly also bounded above by $n/2^{n^{1/c}/2}$. Now L’Hôpital kills the numerator like in (c), and the denominator stays something that clearly goes to $\infty$ as $n$ does. Hence this bounding ration goes to $0$, taking the original ratio down with it. Thus this is a proper hierarchy.
(3) \((18 + 6 + 15 = 39\) pts., for 102 on the set)

Consider the following decision problem:

**Cycling DFA**

**Instance:** A DFA \(M = (Q, \Sigma, \delta, s, F)\) and a string \(x \in \Sigma^n\) where \(Q = \{1, 2, \ldots, n\}\).

**Question:** Does \(M\) on input \(x\) visit every one of its states and end up back at \(s\)?

(a) Sketch a deterministic Turing machine \(M\) that decides this problem in \(O(\log n)\) space. It is enough to diagram the worktapes of \(M\) and say what information each one maintains while sketching the algorithm in pseudocode.

(b) Also estimate the worst-case running time of your \(M\). (It is AOK to ignore \(O(\log n)\) factors by using \(\tilde{O}\) notation, meanwhile ignoring the difference between \(n\) and the true instance length which is \(N = \Theta(n \log n)\).)

(c) Now sketch a faster algorithm that uses linear space. You may use the fact that Turing machines can execute mergesort at full efficiency. Estimate its running time and compare with your answer to (b).

**Answer:** (a) The DTM \(M\) uses one tape to try each state \(q, 1 \leq q \leq n\), to verify that \(q\) occurs once on the path. To handle each \(q\) in the outer loop, it uses one tape to manage an inner loop over bits \(x_i\) of \(x\) \((i = 1 \text{ to } n)\) and one more tape to track the current state \(p\) in that loop. If \(q\) “checks out” by having \(p = q\) occur once and only once as the \(n\) steps are traced out, then \(M\) increments \(q := q + 1\) and re-starts the inner loop running the DFA on \(x\). If any state fails to check out then \(M\) rejects, but if they all check out and \(s\) is the last state in the path again (making it a cycle) then \(M\) accepts. The body of the inner loop requires one more tape to track the location \(j\) in the code of the DFA—if the next state is way-the-hell over on the other side of the code then the body can take all of \(O(n)\) steps.

(b) So the algorithm uses two loops from 1 to \(n\) with an \(O(n)\) body, for overall time \(O(n^3)\) (ignoring \(O(\log n)\) factors by treating each label as a unit the way a RAM does—note that this really is doable on a TM though).

(c) Do the entire trace of the DFA on \(x\) writing down each state that is encountered along the path. This can take \(O(n^2)\) time by the potential need to traverse the code per each step, but that’s still better than \(O(n^3)\)—albeit at the cost of linear space. Then sorting the list of states allows a final check that they are all distinct—i.e., we got \((1, 2, \ldots, n)\) as the sorted list. Even with a quadratic-time sort this is all in \(O(n^2)\) time, which improves on the \(O(n^3)\) from (b).

**Technotes:** On a RAM the latter time would really be \(O(n \log n)\) using optimal sorting and random-access processing of the code of the DFA. If you defined a “logspace RAM” you might be able to speed part (a) up to \(O(n^2)\) time, but that wouldn’t contradict the terms of the problem by using RAMs anyway.

The problem originally just said \(|Q| = n\) which would allow states to use any reasonable \(O(\log n)\)-sized labels they want. It turns out not to matter because the logspace-bounded DTM can calculate \(n\) from \(x\) and then can cycle through all the possible labels and verify that exactly \(n\) are used in the code. But fixing \(Q = \{1, 2, \ldots, n\}\) makes the essence clearer anyway.