1. ***Racetrack*** is a two-player paper-and-pencil racing game that Jeff played on the bus in 5th grade. The game is played with a track drawn on a sheet of graph paper. The players alternately choose a sequence of grid points that represent the motion of a car around the track, subject to certain constraints explained below.

   Each car has a *position* and a *velocity*, both with integer x- and y-coordinates. A subset of grid squares is marked as the *starting area*, and another subset is marked as the *finishing area*. The initial position of each car is chosen by the player somewhere in the starting area; the initial velocity of each car is always $(0, 0)$. At each step, the player optionally increments or decrements either or both coordinates of the car's velocity; in other words, each component of the velocity can change by at most 1 in a single step. The car's new position is then determined by adding the new velocity to the car's previous position. The new position must be inside the track; otherwise, the car crashes and that player loses the race. The race ends when the first car reaches a position inside the finishing area.

   Suppose the racetrack is represented by an $n \times n$ array of bits, where each 0 bit represents a grid point inside the track, each 1 bit represents a grid point outside the track, the "starting area" is the first column, and the "finishing area" is the last column.

   Describe and analyze an algorithm to find the minimum number of steps required to move a car from the starting line to the finish line of a given racetrack.

   Solution: First, we claim that the horizontal speed of the car cannot exceed $\sqrt{2n}$. Suppose the car starts in column 0 of the grid, with velocity zero, and accelerates as quickly as possible to the right. After t steps, the car has velocity $(t, 0)$ and lies in column $\sum_{i=1}^{t} i = t(t+1)/2$. Because the car cannot go past the right edge of the grid, we must have $t(t+1)/2 \le n$, which implies that $t \le \sqrt{2n}$. Similar arguments imply that the car's vertical speed cannot exceed $\sqrt{2n}$.

   Let $T[1..n, 1..n]$ be the input bitmap. We construct a directed graph G with $O(n^3)$ vertices and $O(n^3)$ edges as follows:

   - G has a vertex for each integer vector $(x, y, \Delta x, \Delta y)$ that represents a legal position and velocity for the car. The integer vector (x, y) is a legal position if and only if $T[x, y] = 0$ (and therefore $1 \le x \le n$ and $1 \le y \le n$). Similarly, as we argued above, $(\Delta x, \Delta y)$ is a legal velocity if and only if $-\sqrt{2n} \le \Delta x \le \sqrt{2n}$ and $-\sqrt{2n} \le \Delta y \le \sqrt{2n}$.
   - G has a directed edge for each legal move by the car. Specifically, G contains the directed edge $(x, y, \Delta x, \Delta y) \to (x', y', \Delta x', \Delta y')$ if and only if both endpoints are legal vertices and all of the following conditions are satisfied:

     $$x' = x + \Delta x \qquad \Delta x' \in \{\Delta x - 1, \Delta x, \Delta x + 1\}$$
     $$y' = y + \Delta y \qquad \Delta y' \in \{\Delta y - 1, \Delta y, \Delta y + 1\}$$

   - G also has an artificial starting vertex s, with $O(n)$ outgoing edges to every vertex of the form $(1, y, 0, 0)$ such that the point $(1, y)$ is in the starting area.
   - Finally, G has an artificial target vertex t, with $O(n^2)$ incoming edges from every vertex $(n, y, \Delta x, \Delta y)$ such that the point (n, y) is in the finishing area.

By construction, every directed path from s to t in G represents a sequence of steps that begins with the car at the starting line with velocity $(0,0)$ and ends with the car at the finish line. Moreover, a path of length ℓ corresponds to a sequence of $\ell-2$ steps. Thus, the shortest sequence of steps corresponds to the ***shortest path*** from s to t in G. We can compute this shortest path in $O(V+E) = \boldsymbol{O(n^3)}$ ***time*** via ***breadth-first search***. ■

Rubric: 10 points: standard graph-reduction rubric (brute force).

- −1 for using Dijkstra's algorithm instead of BFS.
- −1 for describing BFS in detail instead of just writing "breadth-first search" or "BFS".
- −1 for only showing $O(n^4)$ time instead of $O(n^3)$ time for this algorithm.
- Max 8 points for an algorithm that actually runs in $\Theta(n^4)$ time (for example, separately BFS from each start position); scale partial credit.
- Max 5 points for a slower polynomial-time algorithm; scale partial credit..
- Max 2½ points for an exponential-time algorithm; scale partial credit.
- No points for "try all possible paths".

2. Jeff likes to go on a long bike ride every Sunday, but because he is lazy, he absolutely refuses to ever ride into the wind.

   Jeff has encoded a map of all bike-safe roads in Champaign-Urbana into an undirected graph $G = (V, E)$ whose vertices represent intersections and sharp corners, and whose edges represent straight road segments. Jeff's home is represented in G by a special vertex s. Every edge of G is labeled with its length and orientation.

   (a) One Sunday the weather forecast predicts wind from due north all day long, which means Jeff can only ride along each road segment in the direction that tends south. Describe and analyze an algorithm to determine the longest total distance Jeff can ride, without ever riding into the wind, before he has to call his wife to come pick him up in the car.

Solution: Let $G = (V, E)$ denote the input graph. We create a directed version $G' = (V, E')$ of the input graph G by directing every edge from north to south. G' has no cycles, because every directed walk moves monotonically further south.

We need to compute the longest weighted path (maximizing the sum of the edge lengths, not just the number of edges) in the dag G' that starts at vertex s. We can compute this longest path in $O(V + E') = \boldsymbol{O(V + E)}$ ***time*** using the dynamic programming / topological sort algorithm described in the textbook. ■

Rubric: 5 points: standard graph-reduction rubric (scaled). —½ for writing out the longest-path algorithm instead of just citing the textbook.

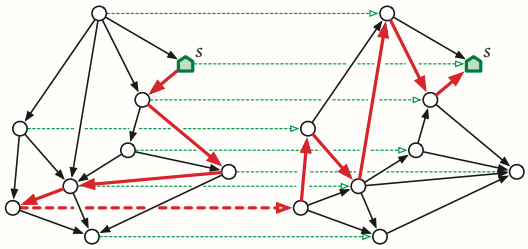
(b) The following Sunday's weather forecast predicts wind from due north in the morning, followed by wind from due west in the afternoon. Describe and analyze an algorithm to find the longest total distance Jeff can ride if he starts at home, rides out to some destination in the morning, eats lunch at noon (while the wind shifts), and then rides home in the afternoon, all without ever riding into the wind.

Solution (one graph): Let $G = (V, E)$ denote the input graph, and let $|e|$ denote the length of any edge e. We create a new directed graph $G' = (V', E')$ as follows:

- $V' = V \times \{N, W\}$. To simplify notation slightly, I'll use subscripts instead of air notation, for example, writing u_N instead (u, N).
- E' contains three types of edges:
  - For every edge uv in G, an edge between u_N and v_N, directed from north to south, with weight $|uv|$.
  - For every edge uv in G, an edge between u_W and v_W, directed from east to west, with weight $|uv|$.
  - For each vertex v of G, a directed edge $v_N \to v_W$ with weight zero.

We can construct G' by brute force in $O(V' + E') = O(V + E)$ time. This graph is actually a dag; we can construct a topological order by listing all vertices v_N in order from north to south, followed by all vertices v_W in order from west to east.

We need to find the longest weighted path in G' from s_N to s_W. We can solve this problem in $O(V' + E') = O(V + E)$ time using the longest-path algorithm in the textbook. Altogether our algorithm runs in ***O(V + E) time***.



Longest path from s_N to s_W in G'; dashed edges have weight 0.

■

Solution (two graphs): Let $G = (V, E)$ denote the input graph. We create two directed versions $G_1 = (V, E_1)$ and $G_2 = (V, E_2)$ of the input graph G, where G_1 is defined by directing every edge in G from north to south, and G_2 is defined by directing every edge in G from west to east. Both G_1 and G_2 are actually dags; sorting the vertices from north to south gives a topological order for G_1, and sorting the vertices from west to each gives a topological order for G_2. We then proceed as follows:

- For every node v, let $\ell_1(v)$ denote the length of the longest path in G_1 from s to v. We can compute $\ell_1(v)$ for all nodes v in $O(V + E_1) = O(V + E)$ time using the longest-path algorithm in the textbook.
- For every node v, let $\ell_2(v)$ denote the length of the longest path in G_2 from v

to s. We can compute $\ell_2(v)$ for all nodes v in $O(V + E_2) = O(V + E)$ time using the longest-path algorithm in the textbook.

- Finally, we need to compute $\max_v(\ell_1(v) + \ell_2(v))$. We can compute this value in $O(V)$ additional time by brute force.

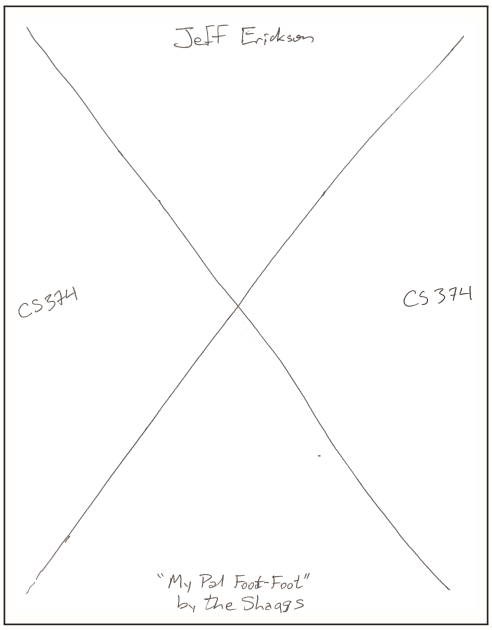
Altogether our algorithm runs in $\boldsymbol{O(V+E)}$ ***time***. ■

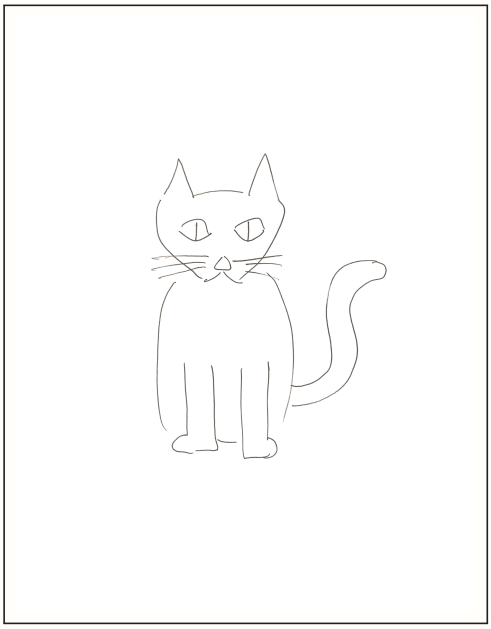
Rubric: 5 points: standard graph-reduction rubric (scaled).

- −½ for writing out the longest-path algorithm instead of just citing the textbook.
- Max 4 points for a slower polynomial-time algorithm (for example, independently compute the longest path from s to each other vertex); scale partial credit.

3. Follow the instructions for "Homework 8.3" on the course Gradescope site.

Solution:





■

Rubric: 10 points = 1 for using blank white paper + 2 for using a dark pen + 2 for submitting a scan instead of a raw photo + 3 for a *good* scan (in focus, high contrast, properly aligned, no keystone effect, no shadows, no background) + 2 for following content instructions. This is not the only correct solution. Yes, this actually counts.