CS/ECE 374 A ✧ Fall 2021

Homework 8

Due Tuesday, October 26, 2021 at 8pm Central Time

---

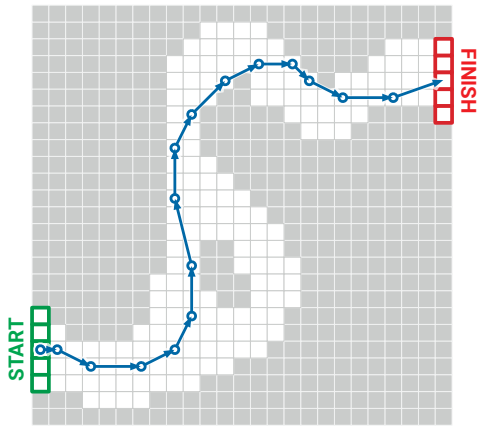
1. ***Racetrack*** (also known by several other names, including *Graph Racers* and *Vector Rally*) is a two-player paper-and-pencil racing game that Jeff played on the bus in 5th grade.[1] The game is played with a track drawn on a sheet of graph paper. The players alternately choose a sequence of grid 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.[2] 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. *[Hint: Build a graph. No, not that graph, a different one. What are the vertices? What are the edges? What problem is this?]*

velocity	position
$(0, 0)$	$(1, 5)$
$(1, 0)$	$(2, 5)$
$(2, -1)$	$(4, 4)$
$(3, 0)$	$(7, 4)$
$(2, 1)$	$(9, 5)$
$(1, 2)$	$(10, 7)$
$(0, 3)$	$(10, 10)$
$(-1, 4)$	$(9, 14)$
$(0, 3)$	$(9, 17)$
$(1, 2)$	$(10, 19)$
$(2, 2)$	$(12, 21)$
$(2, 1)$	$(14, 22)$
$(2, 0)$	$(16, 22)$
$(1, -1)$	$(17, 21)$
$(2, -1)$	$(19, 20)$
$(3, 0)$	$(22, 20)$
$(3, 1)$	$(25, 21)$



A 16-step Racetrack run, on a 25×25 track. This is *not* the shortest run on this track.

---

[1] The actual game is a bit more complicated than the version described here. See http://harmmade.com/vectorracer/ for an excellent online version.

[2] However, it is not necessary for the line between the old position and the new position to lie entirely within the track. Sometimes Speed Racer has to push the A button.

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.

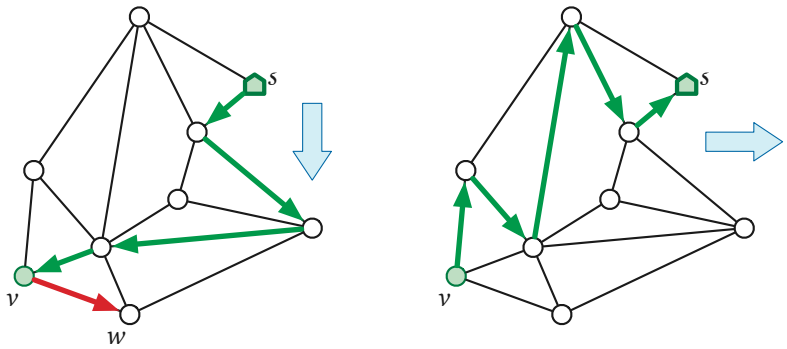
   Jeff has encoded a map of all bike-safe roads in Champaign-Urbana into an undirected graph $G = (V, E)$ whose vertices represent intersections and sharp corners, and whose edges represent straight road segments. Jeff's home is represented in G by a special vertex s. Every edge of G is labeled with its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

   (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.

   In both cases, your input consists of the graph G and the start vertex s. Despite overpowering evidence to the contrary, assume that Jeff can ride infinitely fast, and that no roads in Champaign-Urbana are oriented exactly north-south or exactly east-west.

   For example, suppose Jeff has the graph G shown below. On the first Sunday, Jeff can ride from s to w along the path shown on the left, including the red edge from v to w. On the second Sunday, Jeff can ride from s to v along the green path on the left in the morning, and then from v back to s along the green path on the right in the afternoon; however, he cannot ride to w, because every path from w to s requires riding into the wind at least once, and Jeff's wife is tired of driving out to the middle of nowhere to rescue him.



3. This problem is intended as a practice run for future homeworks, the second midterm, the final exam. **Each student must submit individually.**

   On the course Gradescope site, you will find an assignment called "Homework 8.3". Do not open this Gradescope assignment until you have the following items:

   - Two blank white sheets of paper. (In particular, *not* lined notebook paper.)
   - A pen with dark ink, preferably blue or black. (In particular, *not* a pencil.)
   - A fully-charged and working cell phone with a scanning app installed. (Gradescope recommends Scannable for iOS devices and Genius Scan for Android devices.)
   - A well-lit environment for scanning.

   The assignment will ask you to write/draw something, scan the paper, convert your scan to a PDF file, and upload the PDF to Gradescope. (Gradescope will automatically assign pages of your uploaded PDF to corresponding subproblems.)

   Alternatively, you can write/draw on a tablet and a note-taking app, export your note as a PDF file, upload the PDF to Gradescope.

   Once you open the Gradescope assignment, you will have 15 minutes to complete the submission process.

   The precise content to be written/drawn will be revealed in the Gradescope assignment. (Don't worry, we won't ask for anything technical. The actual writing/drawing should take less than 60 seconds.) If you are not used to your scanning app, we strongly recommend practicing the entire scanning process before starting the assignment.

   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. Yes, this actually counts.

Rubrics

Standard rubric for graph reduction problems. For problems out of 10 points:

- + 1 for correct vertices, *including English explanation for each vertex*
- + 1 for correct edges
  - —½ for forgetting "directed" if the graph is directed
- + 1 for stating the correct **problem** (For the solved problem below: "shortest path in G from $(0, 0, 0)$ to any target vertex")
  - — "Breadth-first search" is not a problem; it's an algorithm!
- + 1 for correctly applying the correct **algorithm**. (For the solved problem below, "breadth-first search from $(0, 0, 0)$ and then examine every target vertex")
  - —½ for using a slower or more specific algorithm than necessary
- + 1 for time analysis in terms of the input parameters.
- + 5 for other details of the reduction
  - – If your graph is constructed by naive brute force, you do not need to describe the construction algorithm. In this case, points for vertices, edges, problem, algorithm, and running time are all doubled.
  - – Otherwise, apply the appropriate rubric to the construction algorithm. For example, for an algorithm that uses dynamic programming to build the graph quickly, apply the standard dynamic programming rubric.

Solved Problem

4. Professor McClane takes you out to a lake and hands you three empty jars. Each jar holds a positive integer number of gallons; the capacities of the three jars may or may not be different. The professor then demands that you put exactly k gallons of water into one of the jars (which one doesn't matter), for some integer k, using only the following operations:

   (a) Fill a jar with water from the lake until the jar is full.

   (b) Empty a jar of water by pouring water into the lake.

   (c) Pour water from one jar to another, until either the first jar is empty or the second jar is full, whichever happens first.

   For example, suppose your jars hold 6, 10, and 15 gallons. Then you can put 13 gallons of water into the third jar in six steps:

   - Fill the third jar from the lake.
   - Fill the first jar from the third jar. (Now the third jar holds 9 gallons.)
   - Empty the first jar into the lake.
   - Fill the second jar from the lake.
   - Fill the first jar from the second jar. (Now the second jar holds 4 gallons.)
   - Empty the second jar into the third jar.

   Describe and analyze an efficient algorithm that either finds the smallest number of operations that leave exactly k gallons in any jar, or reports correctly that obtaining exactly k gallons of water is impossible. Your input consists of the capacities of the three jars and the positive integer k. For example, given the four numbers 6, 10, 15, and 13 as input, your algorithm should return the number 6 (the length of the sequence of operations listed above).

Solution: Let A, B, C denote the capacities of the three jars. We reduce the problem to breadth-first search in a directed graph $G = (V, E)$ defined as follows:

- $V = \{(a, b, c) \mid 0 \le a \le A \text{ and } 0 \le b \le B \text{ and } 0 \le c \le C\}$. Each vertex corresponds to a possible ***configuration*** of water in the three jars. There are $(A+1)(B+1)(C+1) = O(ABC)$ vertices altogether.
- G contains a directed edge $(a, b, c) \to (a', b'c')$ whenever it is possible to move from the first configuration to the second in one step. Specifically, G contains an edge from (a, b, c) to each of the following vertices (except those already equal to (a, b, c)):
  - $(0, b, c)$ and $(a, 0, c)$ and $(a, b, 0)$ — dumping a jar into the lake
  - (A, b, c) and (a, B, c) and (a, b, C) — filling a jar from the lake
  - $\begin{cases} (0, a+b, c) & \text{if } a+b \le B \\ (a+b-B, B, c) & \text{if } a+b \ge B \end{cases}$ — pouring from jar 1 into jar 2
  - $\begin{cases} (0, b, a+c) & \text{if } a+c \le C \\ (a+c-C, b, C) & \text{if } a+c \ge C \end{cases}$ — pouring from jar 1 into jar 3
  - $\begin{cases} (a+b, 0, c) & \text{if } a+b \le A \\ (A, a+b-A, c) & \text{if } a+b \ge A \end{cases}$ — pouring from jar 2 into jar 1
  - $\begin{cases} (a, 0, b+c) & \text{if } b+c \le C \\ (a, b+c-C, C) & \text{if } b+c \ge C \end{cases}$ — pouring from jar 2 into jar 3
  - $\begin{cases} (a+c, b, 0) & \text{if } a+c \le A \\ (A, b, a+c-A) & \text{if } a+c \ge A \end{cases}$ — pouring from jar 3 into Jar 1
  - $\begin{cases} (a, b+c, 0) & \text{if } b+c \le B \\ (a, B, b+c-B) & \text{if } b+c \ge B \end{cases}$ — pouring from jar 3 into jar 2

  Because each vertex has at most 12 outgoing edges, there are at most $12(A+1) \times (B+1)(C+1) = O(ABC)$ edges altogether.

To solve the jars problem, we need to find the ***shortest path*** in G from the start vertex $(0, 0, 0)$ to any target vertex of the form $(k, \cdot, \cdot)$ or $(\cdot, k, \cdot)$ or $(\cdot, \cdot, k)$. We can compute this shortest path by calling ***breadth-first search*** starting at $(0, 0, 0)$, and then examining every target vertex by brute force. If BFS does not visit any target vertex, we report that no legal sequence of moves exists. Otherwise, we find the target vertex closest to $(0, 0, 0)$ and trace its parent pointers back to $(0, 0, 0)$ to determine the shortest sequence of moves. The resulting algorithm runs in $O(V + E) = \boldsymbol{O(ABC)}$ ***time***.

We can make this algorithm faster by observing that every move leaves at least one jar either empty or full. Thus, we only need vertices (a, b, c) where either $a = 0$ or $b = 0$ or $c = 0$ or $a = A$ or $b = B$ or $c = C$; no other vertices are reachable from $(0, 0, 0)$. The number of non-redundant vertices and edges is $O(AB + BC + AC)$. Thus, if we only construct and search the relevant portion of G, the algorithm runs in $\boldsymbol{O(AB + BC + AC)}$ ***time***. ■

Rubric: 10 points: standard graph reduction rubric

- Brute force construction is fine.

— 1 for calling Dijkstra instead of BFS

- max 8 points for $O(ABC)$ time; scale partial credit.