CS/ECE 374 A (Spring 2022)
Homework 8 Solutions

Problem 8.1: We are given a set P of n points $\{p_1, \ldots, p_n\}$ in two dimensions. Each point p_i has coordinates (x_i, y_i). (In this question, the x_i's and y_i's may not be distinct!)

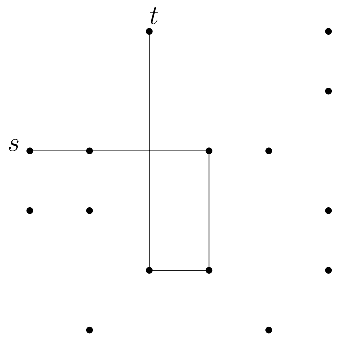
(a) Given a source point $s \in P$ and a destination point $t \in P$, we want to determine whether there exists a sequence $p_{i_0} p_{i_1} p_{i_2} \cdots p_{i_k}$ with $p_{i_0} = s$ and $p_{i_k} = t$, such that each segment $p_{i_j} p_{i_{j+1}}$ is either horizontal or vertical. (Paths are allowed to self-intersect.) Describe an efficient algorithm to solve this problem.

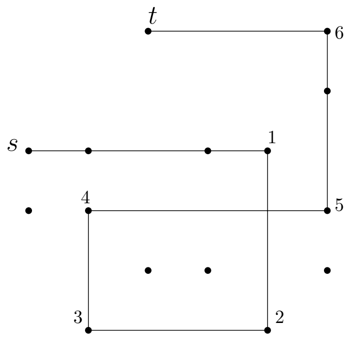
(b) Next, given a source $s \in P$ and a destination $t \in P$ and a value L, we want to determine whether there exists a sequence $p_{i_0} p_{i_1} p_{i_2} \cdots p_{i_k}$ with $p_{i_0} = s$ and $p_{i_k} = t$, such that (i) each segment $p_{i_j} p_{i_{j+1}}$ is either horizontal or vertical, and (ii) $d(p_{i_j}, p_{i_{j+1}}) + d(p_{i_{j+1}}, p_{i_{j+2}}) \geq L$ for all $j = 0, \ldots, k-2$. (The motivation is in finding a path that avoids quick turns.) Describe an efficient algorithm to solve this problem.

You should solve the above problems by transforming the input into a graph and applying a standard algorithm you have seen in class. In these type of graph questions, remember to give mathematically precise definitions of the vertices and edges of your graph. Analyze the overall running time (the time to construct the graph plus the time to run the standard algorithm) as a function of n.

[Hint: For (a), partial credit will be given for an $O(n^2)$-time algorithm, but there is an $O(n \log n)$-time algorithm. For (b), construct a graph with $O(n^2)$ vertices...]

Below, the left picture shows one path that is a feasible solution for (a). The right picture shows a path that might work better for (b), depending on the value of L.





First Solution for (a): Define an undirected graph $G = (V, E)$:

- The vertices are the input points, i.e., $V = P$.

- For each point $p_i \in P$, define its *right neighbor* $N_R(p_i)$ to be a point p_j with the smallest x_j such that $y_j = y_i$ and $x_j > x_i$; define its *left neighbor* $N_L(p_i)$ to be a point p_j with the largest x_j such that $y_j = y_i$ and $x_j < x_i$; define its *up neighbor* $N_U(p_i)$ to be a point p_j with the smallest y_j such that $x_j = x_i$ and $y_j > y_i$; define its *down neighbor* $N_D(p_i)$ to be a point p_j with the largest y_j such that $x_j = x_i$ and $y_j < y_i$. Add the edges from p_i to its four neighbors $N_R(p_i)$, $N_L(p_i)$, $N_U(p_i)$, and $N_D(p_i)$ (if they exist).

We run BFS or DFS from s, to decide whether there exists a path from s to t in G.

(Correctness is not difficult to see. Suppose there is a sequence $p_{i_0}p_{i_1}\cdots p_{i_k}$ with $p_{i_0} = s$ and $p_{i_k} = t$, such that each segment $p_{i_j}p_{i_{j+1}}$ is either horizontal or vertical. For each j, since p_{i_j} and $p_{i_{j+1}}$ have the same x-coordinate or the same y-coordinate, there is a path from p_{i_j} and $p_{i_{j+1}}$ by following a series of up/down neighbors or a series of left/right neighbors, and so there is a path from s to t in G. Conversely, if there is path $p_{i_0}p_{i_1}\cdots p_{i_k}$ from s to t in G, then $p_{i_0}p_{i_1}\cdots p_{i_k}$ obviously satisfies the stated condition.)

Running time analysis. The graph G has n vertices. Since each vertex has degree at most 4, the number of edges is $m \leq 4n$.

To construct the graph, we sort all the input points lexicographically by x coordinates, and for points with the same x, by y-coordinates. This takes $O(n \log n)$ time. Then for the points with the same x, we can identify their top and bottom neighbors by a linear scan in the y-sorted list (the top and bottom neighbors are precisely the successor and predecessor in the y-sorted list). The total time for the linear scans is $O(n)$. Similarly, we can identify the left and right neighbors of all points in $O(n)$ time, this time by sorting by y-coordinates first, and for points with the same y, by x-coordinates.

Afterwards, the running time of BFS or DFS is $O(m + n)$, which is $O(n)$ since $m \leq 4n$.

Total time: $O(n \log n)$.

(*Remark.* For a more naive solution, we can just add an edge between p_i and p_j iff $x_i = x_j$ or $y_i = y_j$. But this graph may have quadratically many edges, resulting in an $O(n^2)$-time algorithm.)

Second Solution for (a): We describe an alternate solution that is even simpler. Define an undirected (bipartite) graph $G' = (V, E)$:

- Let X be all the x-coordinates of P, and Y be all the y-coordinates of P. The vertices of our new graph will be the coordinates rather than the points. More precisely, $V = \{(x, 1) : x \in X\} \cup \{(y, 2) : y \in Y\}$.
- For each point $p_i = (x_i, y_i) \in P$, we add an edge between $(x_i, 1)$ and $(y_i, 2)$.

Suppose $s = p_a$ and $t = p_b$. We run BFS or DFS from $(x_a, 1)$, to decide whether there exists a path from $(x_a, 1)$ to $(x_b, 1)$ in G'.

Correctness. Suppose there is a sequence $p_{i_0}p_{i_1}\cdots p_{i_k}$ with $i_0 = a$ and $i_k = b$, such that each segment $p_{i_j}p_{i_{j+1}}$ is either horizontal or vertical. For each j, there is a path from $(x_{i_j}, 1)$ to $(x_{i_{j+1}}, 1)$ in G', namely, the path $(x_{i_j}, 1)(y_{i_j}, 2)(x_{i_{j+1}}, 1)$ if $p_{i_j}p_{i_{j+1}}$ is horizontal (i.e., $y_{i_j} = y_{i_{j+1}}$), or the trivial path of length 0 if $p_{i_j}p_{i_{j+1}}$ is vertical (i.e., $x_{i_j} = x_{i_{j+1}}$). It follows that there is path from $(x_a, 1)$ to $(x_b, 1)$ in G'.

Conversely, suppose there is path from $(x_a, 1)$ to $(x_b, 1)$ in G', say, $(x_{i_0}, 1)(y_{i_0}, 2)(x_{i_1}, 1)(y_{i_1} 2) \cdots (x_{i_k}, 1)$ with $i_0 = a$ and $i_k = b$. Then $s, (x_{i_0}, y_{i_0}), (x_{i_1}, y_{i_0}), (x_{i_1}, y_{i_1}), \ldots, (x_{i_k}, y_{i_{k-1}}), t$ form a sequence of points in P and alternate between vertical and horizontal segments.

Running time analysis. The graph G' has at most $2n$ vertices and n edges. Thus, the running time of the BFS or DFS is $O(n)$. We also need to bound the time to construct the graph G'. To construct X and Y, one needs to identify and remove duplicate coordinate values. This requires either hashing, which takes $O(n)$ expected time (if coordinates are integers), or sorting, which takes $O(n \log n)$ time. (After duplicates are identified, we technically also need to map coordinate values to indices, so that we can build an adjacency list representation of the graph.) The total time is at most $O(n \log n)$ either way.

Solution for (b): Define an directed (or undirected) graph $G = (V, E)$:

- $V = \{s, t\} \cup \{(p_i, p_j) : \ p_i, p_j \in P \text{ and } (x_i = x_j \text{ or } y_i = y_j)\}$. Notice that except for s and t, vertices are *pairs of points*!
- For each $p_i, p_j, p_k \in P$, if $(p_i, p_j), (p_j, p_k) \in V$, we add an edge from (p_i, p_j) to (p_j, p_k) iff $d(p_i, p_j) + d(p_j, p_k) \geq L$.
- For each $p_i \in P$, if $(s, p_i) \in V$, we add an edge from s to (s, p_i).
- For each $p_i \in P$, if $(p_i, t) \in V$, we add an edge from (p_i, t) to t.

We run BFS or DFS from s, to determine whether there is a path from s to t in G.

Correctness is obvious from the definition of the problem.

Running time analysis. The number N of vertices in G is at most $2 + n^2 = O(n^2)$. The number M of edges in G is at most $n^3 + 2n = O(n^3)$. The graph can be constructed in $O(n^3)$ time, by looping through all $O(n^3)$ combinations of p_i, p_j, p_k. The running time of BFS or DFS is $O(M + N)$, which is $O(n^3)$. Total time: $O(n^3)$.

(*Remark.* A faster algorithm is possible with more complicated ideas. I can get close to $O(n^2 \log n)$ time. I currently don't have a subquadratic-time algorithm, but suspect that there could be one...)

Problem 8.2: We are given a directed graph with n vertices and m edges and a source vertex s, where each edge e has a color $c(e)$ from $\{1, \ldots, k\}$.

(a) (25 pts) Describe an algorithm to determine whether there is a closed walk that contains s and uses at least 4 colors. (Recall that in a *walk*, repeated vertices are allowed.) For full credit, the running time should be $O(m + n)$.
[Hint: use the meta-graph.]

(b) (75 pts) Describe an algorithm to determine whether there is a walk (not necessarily closed) that starts at s and uses at least 4 colors. For full credit, the running time should be $O(k^3(m + n))$ or better.
[Hint: one approach is to define a new graph with $O(k^3 n)$ vertices. You will still get partial credit if you obtain $O(k^4(m + n))$ time instead.]

Solution:

(a) The algorithm is simple:

  (a) First compute the strongly connected components.
  (b) Return true iff the component containing s contains at least 4 colors.

Runtime. We can compute the strongly connected components in $O(m)$ time by the algorithm from class. We can check whether a component uses at least 4 colors, by examining each edge one by one, and adding its color to the current list, and stopping as soon as the list has size 4.

(Remark: actually, the strongly connected components algorithm from class is unnecessary, since we only want the component containing s. A simpler way is to run BFS from s in G, and run BFS from s in the reverse graph of G. Vertices in the component of s are precisely those that are both reachable from s in G and reachable from s in the reverse graph.)

Correctness. If a component contains edges $(u_1, v_1), \ldots, (u_k, v_k)$ using at least 4 colors, we can connect these edges together to form a closed walk (since by definition of strongly connected components, there are paths from v_1 to u_2, v_2 to u_3, etc.).

Conversely, if there is a closed walk using at least 4 colors, all vertices in the walk belong to the same strongly connected component, so this component uses at least 4 colors.

(b) Let $G = (V, E)$ be the given directed graph. Construct a new directed graph G':

- For each $v \in V$ and each subset $S \subseteq \{1, \ldots, k\}$ of size at most 3, create a new vertex (v, S) in G'. Also create an extra vertex t'.
- For each edge $(u, v) \in E$ with color $c(u, v)$ and each subset $S \subseteq \{1, \ldots, k\}$ of size at most 3, create an edge from (u, S) to $(v, S \cup \{c(u, v)\})$ in G' if $|S \cup \{c(u, v)\}| \leq 3$, and an edge from (u, S) to t' in G' if $|S \cup \{c(u, v)\}| = 4$.

We then run BFS or DFS in G' from $(s, \emptyset)$, and return true iff there is a path from $(s, \emptyset)$ to t' in G'.

Runtime. G' has at most $O(k^3 n)$ vertices (since there are $O(k^3)$ subsets of $\{1, \ldots, k\}$ of size at most 3) and $O(k^3 m)$ edges. The BFS/DFS takes $O(k^3(m + n))$ time.

Justification. For each subset S of size at most 3, a vertex (v, S) is reachable from $(s, \emptyset)$ in G' iff there is a walk in G from s to v and uses precisely the colors in S. And t' is reachable from $(s, \emptyset)$ iff there is a walk from s using at least 4 colors.