Problem Set 8 Solutions

Problem 8-1. No left turns

You've just stolen a brand-new Nexus Nano, but the owner has locked the car with *The Spade*, a mechanical lock on the steering wheel, which prevents you from making left turns. In addition, to avoid car accidents and otherwise attracting attention, you aren't going to make any U-turns either. You want to reach your chop-shop[1] as quickly as possible, and since you can drive down straight streets very quickly, your primary goal is to minimize the number of (right) turns you make. But if two paths have the same number of right turns, then you want to minimize the straight-line distance.

Fortunately, you have a map of the city, so you can plan out a route that uses no left turns and no U-turns. Even better, the map comes in electronic form as an $m \times n$ grid of cells, each marked as either ***empty*** (navigable) or ***blocked*** (unnavigable). At each cell you visit, you can continue in the direction you were going, or turn right. Two examples of no-left-turn maps are shown in Figure 1.

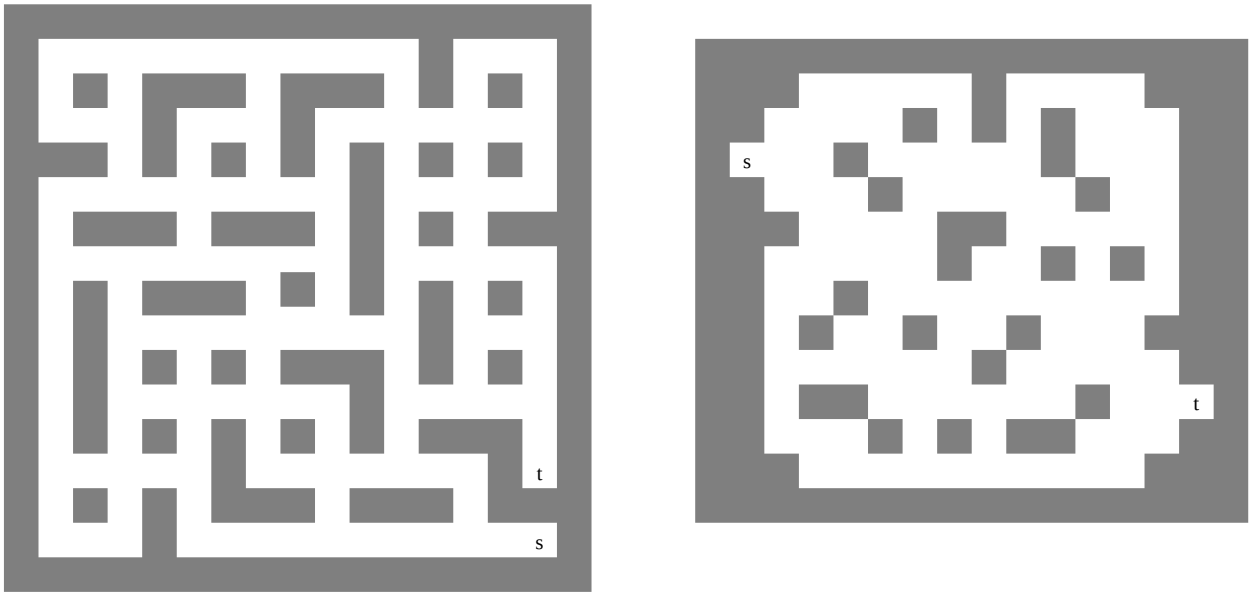


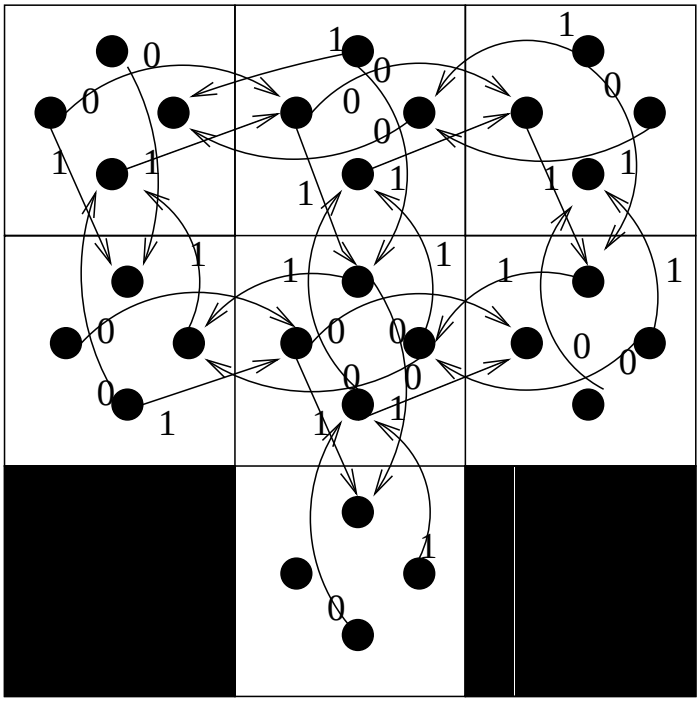
Figure 1: Two no-left-turn maps. Find a path from s to t with as few right turns as possible, and no left- or U-turns. You can play with these maps using a Java applet on `http://www.clickmazes.com/noleft/ixnoleft.htm`.

(a) Give an efficient algorithm to find a no-left-turn path through an $m \times n$ map using the minimum number of right turns, or report that no such path exists. Among all

[1] A garage that illegally disassembles cars in order to sell the parts.

paths with the minimum number of right turns, your algorithm should return the path minimizing the straight-line distance (i.e., the number of straight steps). (*Hint:* For intuition, solve the left-hand map in Figure 1.)

Solution: The basic idea in solving this problem is to create a graph representation of the problem and then use a shortest paths algorithm to find the optimal no-left-turn path. Each edge (u, v) in the graph has two weights, $w_1(u, v)$ and $w_2(u, v)$. We set $w_1(u, v) = 1$ if the edge represents a right turn and $w_2(u, v) = 1$ if the edge represents a straight line path of distance 1.



..

Figure 2: Graph Representation.

One way to turn this problem into a graph G is shown in Figure 2. Our map of cells is converted into a weighted, directed graph. For each of the mn cells of the map, we create 4 nodes. If the car enters a particular cell from the cell below it on the map it would enter the "south" node associated with that particular cell. After creating the nodes, we assign directed edges to them. Edges that correspond to a righthand turn have an edge weight $w_1(u, v) = 1$ and $w_2(u, v) = 0$ and edges that correspond to driving straight have a weight $w_1(u, v) = 0$ and $w_2(u, v) = 1$. Edges that would represent left hand turns or uturns are not included in the graph. An edge which goes from one orientation to another (example south to west) represents a right turn. It will take us $O(mn)$ time to create this graph representation.

Now that we have our graph representation we need to determine a path that minimizes right-hand turns, and then minimizes the straight-line distance. Instead of having a single real-number weight on every edge, however, the weight of an edge $w(u, v)$ is now a pair of numbers, $(w_1(u, v), w_2(u, v))$. We compare two weights by looking at the first weight, and then breaking ties using the second weight. In other words, two weights $w = (w_1, w_2)$ and $w' = (w'_1, w'_2)$ satisfy $w < w'$ exactly when either $w_1 < w'_1$, or $w_1 = w'_1$ and $w_2 < w'_2$.[2]

Alternatively, we can get by in this problem with a single edge weight on every edge: a right turn is given some weight $w > |E|$ (i.e., $4mn + 1$), and straight-line edges are given weight 1. Then, the shortest path in the graph will always contain the minimum number of right turns, because making a right turn is always more expensive than making the maximum possible number of straight-line movements.[3]

Even though these path and edge weights w are pairs instead of real numbers, we can use Dijkstra's algorithm because the edge weights are still "positive." In other words, given a path p from s to u with weight $w(p)$, the weight of a path p' formed by adding an edge (u, v) to p is never smaller, i.e., $w(p') \geq w(p) + w(u, v)$. Since Dijkstra's algorithm runs in $0(V \lg V + E)$ time, this will give us a running time of $O(mn \lg mn)$.

If we don't care about breaking ties by straight-line distance, and we just want a path with the minimum number of right turns, then it is possible to optimize the priority queue for Dijkstra's algorithm to support priority queue operations in $O(1)$ time. This reduces the runtime of the algorithm to $O(mn)$.

One simple approach for doing this is to notice that the path lengths are always integers, and the maximum path weight is $4mn$ (plus nodes at ∞) Thus, it is possible to maintain the priority queue as $O(mn)$ linked lists, one for each possible path weight.

In fact, we actually need only two linked lists if we use a breadth-first search-like algorithm. The only edge weights are 0 for straight-line travel, and 1 for right turns. New nodes that we discover through a weight-0 edge are inserted into one list, while nodes that are discovered through a weight-1 edge are inserted into the other. We leave the precise statement of this algorithm and the proof of correctness as an exercise for the reader.

We conjecture that there exists an $O(mn)$ algorithm to solve this problem when we do break ties by straight-line distance. If you have an $O(mn)$ solution *with a complete correctness proof*, please let us know!

(b) After mapping the route produced by your algorithm from part (a), you realize that you may lead the police right to the chop-shop if they are following you. You decide that the best strategy to avoid pursuit is to make two right turns in quick succession,

[2] To be completely rigorous, we should in fact show that this is a reasonable defi nition for edge weights.

[3] Note that the approach of having two edge weights on every vertex can be thought of as having a weight on right turn edges that approaches ∞ compared to straight-line edges.

i.e., in adjacent squares, the 90-degree rotations intervened by only one unit of straight travel.

So now you want to find a route that makes at least two consecutive right turns, while still making no left turns, making as few right turns as possible, and among all such paths, minimizing the straight-line distance. Give an efficient algorithm to find such a path through the grid, or report that no such path exists.

Solution: Create a new graph G' that consists of two copies $G_0 = (V_0, E_0)$ and $G_1 = (V_1, E_1)$ of the graph G from the previous problem. Insert an edge from $u_0 \in V_0$ to $v_1 \in V_1$ if you can get from u to v using two consecutive right turns in G. This edge has weight $w_1(u_0, v_1) = 2$ and $w_2(u_0, v_1) = 0$. Then, find a shortest path in this new graph G' from s_0 to t_1.

Since the source only exists in G_0 and the sink only exists in G_1, you are guaranteed to make two consecutive right turns at least once. G' has twice as many nodes as the original graph G (i.e., $8mn$). We double the number of edges, and then add at most $8mn$ edges representing two consecutive right turns (for every node in the original graph, we can get to at most one other node making two consecutive right turns). Therefore, asymptotically, searching G' using Dijkstra's algorithm still takes $O(mn \lg mn)$ time.

Problem 8-2. ***Video Game Design***

Professor Cloud has been consulting in the design of the most anticipated game of the year: ***Takehome Fantasy XII***. One of the levels in the game is a maze that players must navigate through multiple rooms from an entrance to an exit. Each room can be empty, contain a monster, or contain a life potion. As the player wanders through the maze, points are added or subtracted from her ***life points*** L. Drinking a life potion increases L, but battling a monster decreases L. If L drops to 0 or below, the player dies.

As shown in Figure 3, the maze can be represented as a digraph $G = (V, E)$, where vertices correspond to rooms and edges correspond to (one-way) corridors running from room to room. A vertex-weight function $f : V \to \mathbb{Z}$ represents the room contents:

- If $f(v) = 0$, the room is empty.
- If $f(v) > 0$, the room contains a life potion. Every time the player enters the room, her life points L increase by $f(v)$.
- If $f(v) < 0$, the room contains a monster. Every time the player enters the room, her life points L drop by $|f(v)|$, killing her if L becomes nonpositive.

The ***entrance*** to the maze is a designated room $s \in V$, and the ***exit*** is another room $t \in V$. Assume that a path exists from s to every vertex $v \in V$, and that a path exists from every vertex $v \in V$ to t. The player starts at the entrance s with $L = L_0 > 0$ life points. For simplicity, assume that the entrance is empty: $f(s) = 0$.

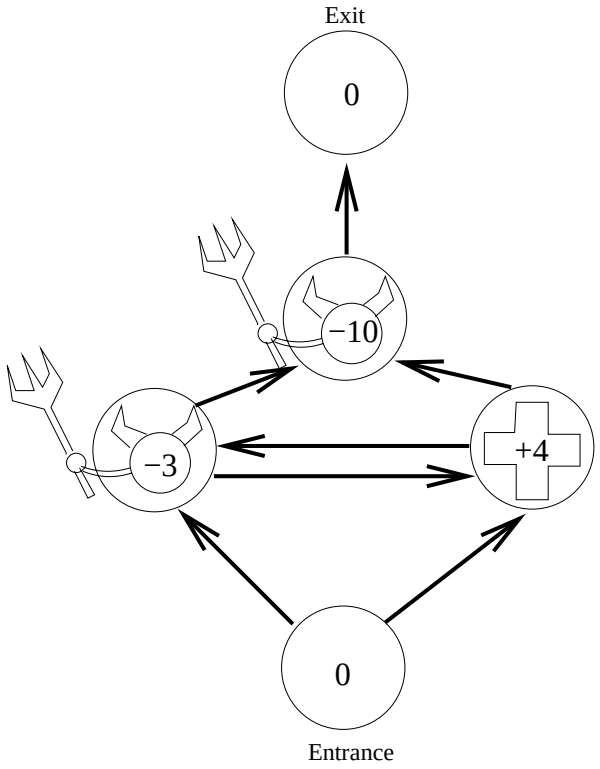


Figure 3: An example of a 1-admissible maze.

Professor Cloud has designed a program to put monsters and life potions randomly into the maze, but some mazes may be impossible to safely navigate from entrance to exit unless the player enters with a sufficient number $L_0 > 0$ of life points. A path from s to t is ***safe*** if the player stays alive along the way, i.e., her life points never become nonpositive. Define a maze to be ***r-admissible*** if a safe path through the maze exists when the player begins with $L_0 = r$ life points.

Help the professor by designing an efficient algorithm to determine the minimum value r for which a given maze is r-admissible, or determine that no such r exists.

(a) Find a safe path in the maze in Figure 3.

(b) Formulate the problem as an equivalent problem where the weights are on the edges, and prove equivalence.

Solution: If the entrance node has a weight, we create a new entrance with weight 0 and add an edge from the new entrance to the original entrance. Now move the weights from the nodes to the edges in the following manner. If node v has weight $f(v)$, then for all $(u, v) \in E$, we set $w(u, v) = f(v)$. The equivalent problem is to find a path through this edge weighted graph, so that the weight of every subpath is positive.

(c) Assume for this problem part that there are no cycles whose traversal gives a net increase in life points. Given r, how would you check whether the maze is r-admissible?

Solution: We use a modified version of Bellman-Ford algorithm. Given an r, for every node u we find the maximum (positive) number of points $q[u]$ the player can have when she reaches u. If $q[t]$ is positive, then the graph is r-admissible.

For each vertex $u \in V$, we maintain $p[u]$ which is a lower bound on $q[u]$. We initialize all the $p[u]$'s to $-\infty$, except the entrance, which is initialized to r. As we run the Bellman-Ford Algorithm and relax edges, the value of $p[u]$ increases until it converges to $q[u]$ (if there are no positive weight cycles). The important point to note is that reaching a node with negative points is as good as not reaching it at all. Thus, we modify $p[u]$ only if it becomes positive, otherwise $p[u]$ remains $-\infty$. We change the relaxation routine to incorporate this as follows.

```
V-RELAX(u, v)
1  if (u, v) ∈ E
2     then if ((p[v] > p[u] + w(u, v)) and (p[u] + w(u, v) > 0))
3             then   p[v] ← p[u] + w(u, v)
4             π[v] ← u
```

After all the edges have been relaxed V times, if there are no positive weight cycles, all $p[u]$'s will have converged to the corresponding $q[u]$'s (the maximmum number of points you can have on reaching vertex u). If $q[t]$ is positive at this point, then the player can reach there with positive life points and thus the graph is r-admissible.

(d) Now assume that there can be cycles in the graph whose traversal gives a net increase in life points. Given r, how would you check whether the maze is r-admissible?

Solution: If $p[t]$ is not positive after relaxing all the edges $V-1$ times, we relax all the edges one more time (just like Bellman-Ford). If $p[u]$ of any node changes, we have found a positive weight cycle which is reachable from s starting with r points. Thus the player can go around the cycle enough times to collect all the necessary points to reach t and thus the graph is r-admissible. If we dont find a reachable positive weight cycle and $p[t]$ is $-\infty$, then the graph is not r admissible. The correctness of the algorithm follows from the correctness of Bellman-Ford, and the running time is $O(VE)$.

(e) How can you find the minimum r for which the maze is r-admissible? **Solution:**

Given the above sub-routine, we now find the minimum r. We first check if the graph is 1-admissible. If it is, we return 1 as the answer. If it is not, then we check if it is 2-admissible and then 4-admissible and so on. Thus on the ith step we check if the graph is 2^{i-1}-admissible. Eventually, we find k such that the graph is not 2^{k-1}-admissible,

but it is 2^k-admissible. Thus the minimum value of r lies between these two values. Then we binary search between $r = 2^{k-1}$ and $r = 2^k$ to find the right value of r.

Analysis: The number of iterations is $k + O(\lg r) = O(\lg r)$, since $k = \lfloor \lg r \rfloor$. Thus you have to run Bellman-Ford $O(\lg r)$ times, and the total running time is $O(VE \lg r)$.