CS 240: Introduction to Discrete Mathematics 6/18/2007

Lecture 1: Introduction - Mazes

Instructor: Jeff Kinne TA: Mike Kowalczyk

In the first lecture, we give an introduction to the course. After reviewing the syllabus, we introduce a problem that we will come back to multiple times during the course - navigating through a maze. This problem is nice because many of the techniques we learn in this course can be used to analyze and solve it.

We follow a few general steps to study the problem: 1) think about the problem and come up with initial solutions, 2) formalize the problem into mathematical objects, 3) see if we can do better than before by modeling the problem into mathematical objects. These steps will also apply to other problems we consider throughout the semester. A common theme is that we can solve problems very well once we model them in the appropriate way.

1 Navigating a Maze

There are a number of situations that can be modeled as needing to navigate a maze: mouse finding cheese in a laboratory maze, treasure hunters looking for treasure in a pyramid maze, robot exploring Mars and looking for a debris-free path to a far-off mountain, ... We will draw inspiration from two other examples from myth and fairy tale: the labyrinth of the Minotaur and Hansel and Gretel. We point out a key property that all of the above have in common - the person navigating the maze does not have a map of the maze beforehand. This makes finding the way through the maze more difficult than if a map were provided. Our goal is thus as follows.

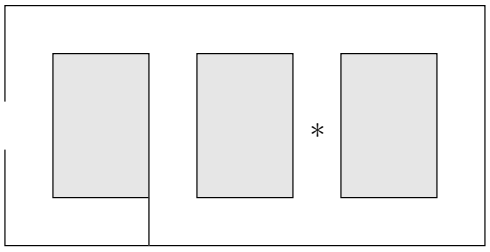

Figure 1: An example of a maze with treasure inside it. The * represents the location of the treasure.

Definition 1 (Maze, informal)**.** *For the problem of* navigating a maze*, we enter a maze and wish to navigate the maze to find treasure hidden within, recover the treasure, and find our way back to the entrance of the maze.*

An example maze is given in Figure 1. Of course, we would not have this map available to us when entering the maze. Here are a few strategies that have been suggested throughout history for successfully navigating mazes.

1. *Right Hand Rule:* When there is a choice of which direction to go, always go to the right. If you see the treasure, pick it up. If you see the entrance to the maze, leave the maze.

2. *String:* (Invented by Daedalus, and used by Theseus to navigate the Labyrinth to slay the Minotaur.) Fix the beginning of a long string to the entrance of the maze. At the first intersection, choose one of the directions to try first. Go along this direction, all the while unraveling the string. At each intersection, choose some direction to try first.

   If we either come to a dead-end or an intersection that already has string going through it, we backtrack along the string (winding it back up as we do so) to the last intersection. We then try a different route out of that intersection. If we have tried all routes out of that intersection, we backtrack yet further to try a new route on the next-last intersection along the string.

   If we happen upon the treasure, we pick it up and backtrack along the string back out of the maze. If our search eventually results in backtracking all the way back to the entrance without finding the treasure, we give up and leave the maze.

3. *Breadcrumbs:* (Used by Hansel and Gretel to find their way out of the forest.) For this strategy, we use breadcrumbs rather than string. We start by entering the maze and laying down breadcrumbs as we walk. When we come to an intersection, we pick one of the directions to try first.

   We want to avoid making a loop with the breadcrumbs because then we might not be able to find our way back out of the maze. So, we always run ahead to the next intersection to see if it is already bread-crumbed. If not, we lay breadcrumbs to it, and choose some direction out of the new intersection to explore first. If we look ahead and the intersection is already bread-crumbed, we do not breadcrumb to it but instead consider a different direction out of the current intersection.

   If we come to a dead-end we backtrack along the breadcrumbs to the last intersection. We already mentioned how we backtrack if we see an intersection that already has breadcrumbs. After we have tried all routes out of an intersection, we backtrack to the previous intersection.

   If we see the treasure, we pick it up and backtrack along the breadcrumbs out of the maze. If our search eventually results in backtracking all the way back to the entrance without finding the treasure, we give up and leave the maze.

4. *Random Walk:* When there is a choice of which direction to go, pick at random. If you see the treasure, pick it up. If you see the entrance and have the treasure, leave the maze. If you see the entrance and don't have the treasure, try again.

Notice that the right hand rule fails to find the treasure in the maze of Figure 1. Convince yourself that the string and breadcrumbs strategy do find the treasure.

2 Evaluating Maze Strategies

There are two basic properties we want from a strategy: correctness and efficiency. For correctness, we want the following:

1. If there is treasure within the maze, we always find it.

2. If there is treasure in the maze, we find our way out after finding it. If there is no treasure, we figure this out eventually and find our way back out of the maze.

There are a variety of measures of efficiency. The following are some goals we would like to achieve:

1. Spend as little time in the maze as possible.
2. Use a strategy that does not require us to remember very much.
3. Find a path to the treasure that is as short as possible.

We will evaluate the maze strategies on each of these correctness and efficiency criterion below. First, we point out that we can model a maze as a graph. Each intersection corresponds to a vertex of the graph, and direct paths between intersections correspond to edges. We also add vertices for dead-ends and the location of the treasure (if it is not at a dead-end or intersection). The graph for the maze of Figure 1 is given in Figure 2. Finding the treasure in the maze, then, corresponds to finding a particular vertex in the graph. We call the graph we are looking at G, and suppose it has n vertices and m edges.

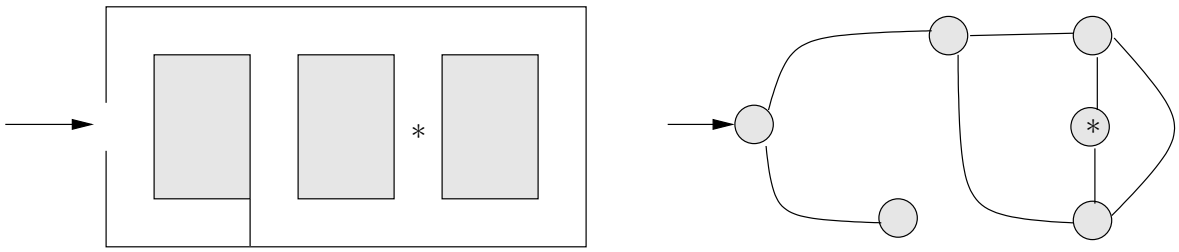

Figure 2: We can model the task of navigating the maze on the left as needing to search in a graph, given on the right.

After modeling the maze as a graph, some would call the problem solved already by invoking graph search algorithms such as depth first search and breadth first search. We point out that for a graph search algorithm to work in the current setting of not having a map beforehand, it must have the property that it works in phases by discovering new vertices and only uses information about vertices already discovered (initially just the entrance/start vertex). Breadth first search and depth first search do satisfy these properties, so they can be used to navigate a maze, but would you actually be able to implement them if you physically were in a maze and needed to get out? We consider the strategies mentioned above that are physically realizable and somewhat intuitive, and see how well they perform. The conclusions we make about time spent in the maze and memory required translate as expected if we were to implement the strategies on a computer - time spent in the maze is roughly the running time, and memory required is roughly the memory usage.

2.1 Right-Hand Rule

The right-hand rule fails as a viable general purpose strategy because it is not guaranteed to find the treasure in all mazes. The graph given in Figure 2 corresponds to a maze where the right-hand rule fails to find the treasure. The right-hand rule does have a few desirable qualities (we state these without proof):

- You are guaranteed to find your way back to the entrance.

- For mazes with no free-standing structures (all walls are ultimately connected to the exterior walls), you are guaranteed to find the treasure.
- Very small memory is needed - we only need to remember what the entrance looks like. In graph terms, we need to remember the label of the entrance vertex, so we need only $O(\log n)$ space.

Many mazes (including for example, many corn mazes) do not have free-standing structures, and for these the right-hand rule is a good first option to try. If you fail to find the treasure using the right-hand rule, you can then try one of the other methods.

However, because the right-hand rule is not guaranteed to find the treasure for general mazes, it fails as a general-purpose strategy. We also point out that the right hand rule is not guaranteed to find the shortest path to the treasure.

2.2 String Strategy

We claim that the string strategy is correct - that it always finds the treasure if there is any. The intuition is that by using the string strategy, you will eventually consider all possible paths between the entrance and the rest of the vertices within the maze. As the treasure must be at one of these vertices, correctness follows. We formalize this intuition by defining a layout of the graph.

Definition 2. *Let G be the graph of a maze with entrance vertex v. A* layout *of the maze is a path from v through the maze with no repeating vertices except for potentially the last one.*

The correctness of the string strategy follows from the following claim, which we do not prove today.

Claim 1. *Let G be the graph for a maze with no treasure. Then the string of the string strategy eventually assumes all possible layouts of the maze.*

Let us see how correctness follows from the claim. The proof is by contradiction. Suppose there is a graph G for a maze and a location t for the treasure within G so that the string strategy fails to find t. Consider the maze corresponding to G but with no treasure. Claim 1 then guarantees that the string strategy assumes all possible layouts for G. In particular, the string assumes the shortest path between the entrance and t at some point. This means that the string strategy must have found t, contradicting our assumption that it did not.

We conclude that the string strategy always finds the treasure, but how much time is spent in the maze? We first point out that for finite graphs, the strategy is guaranteed to exit the maze eventually (in finite time). This is due to the fact (which we do not prove - see a common theme today...) that the string strategy never repeats a layout of the maze more than twice, and there are finitely many layouts of any finite maze. Given this, we also have a bound on the time spent in the maze - the time spent in the maze is roughly the same as $L(G, v)$, where $L(G, v)$ is the number of distinct layouts in a maze G with entrance vertex v.

So how big can $L(G, v)$ be? The graph in Figure 3 is an example that has exponentially many layouts. At each vertex, the string can go either of two directions, so the number of shortest paths between v and v_i is 2^i. The total number of paths in the graph is thus $\geq 2^n$. We conclude that the time spent in the maze for the string strategy can depend exponentially on the size of the maze - this is bad.

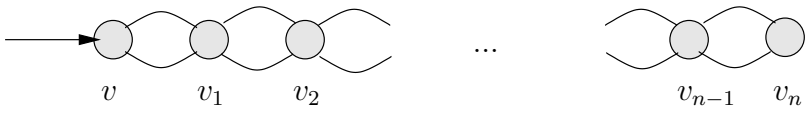

Figure 3: A graph of a maze that has exponentially many different layouts.

So the string strategy may take a long time to complete, how about the memory required? To run the strategy, we need to keep track of which edges we have already tried out of each vertex touching the string. As there are n vertices in the graph, we need at least n bits of memory to accomplish this. This is much more than the $O(\log n)$ bits required to use the right hand rule strategy.

We summarize the properties of the string strategy in the following theorem.

Theorem 1. *The string strategy finds the treasure within a graph G with n vertices within time roughly 2^n, and always finds its way back out of the maze. There are mazes for which time 2^n is required, and the memory usage of the string strategy is at least n.*

The string strategy is also not guaranteed to find the shortest path to the treasure.

2.3 Breadcrumbs Strategy

The thread strategy performed so poorly (with respect to time) because it is very repetitive. By always rolling up the thread, we effectively lose the information gained by exploring parts of the maze. The breadcrumbs strategy tries to correct this problem by leaving the breadcrumbs behind. The correctness and efficiency of the breadcrumbs strategy follows from the following claim.

Claim 2. *Let G be the graph for a maze with no treasure. By using the breadcrumbs strategy, each edge is traversed at least twice and at most four times during the strategy.*

Correctness then follows by a similar argument as given for the string strategy. Notice that the claim also tells us that if G has m edges, then the time spent in the maze is $O(m)$. This is much better than the performance of the string strategy. As with the string strategy, we need to remember for each vertex which edges we have already tried out of it, so the amount of memory needed is at least n.

Theorem 2. *The breadcrumbs strategy finds the treasure within a graph G with m edges within time $O(m)$, and always finds its way back out of the maze. The memory usage of the breadcrumbs strategy is at least n.*

We point out that the breadcrumbs strategy is essentially the same as depth first search, so shares the same properties with it. Among these is that we are not guaranteed to find the shortest path to the treasure.

2.4 Random Walk

For the random walk strategy, there are certainly random choices that will result in traveling in a loop forever (consider the random choices of continually moving between two vertices). Then the random walk strategy is not correct in the same sense of the string and breadcrumb strategies.

But we point out that the probability of picking random choices that result in continually moving between two vertices is very unlikely.

For the random walk strategy, we need a different notion of what it means to be correct. As it is a randomized strategy, it is natural to only require the the strategy succeeds with very high probability. For a fixed ϵ, it can be shown that the random walk strategy successfully finds the treasure and exits the maze with probability at least $1 - \epsilon$ in $O(m^2)$ steps. This is worse than the $O(m)$ required of the breadcrumbs strategy, so why should we care about the random walk?

The great advantage of the random walk is that it is very efficient with respect to memory. All we need to remember is where we currently are and what the entrance to the maze looks like (i.e. two vertices in the graph), so the memory required of the random walk is $O(\log n)$. This matches the memory required of the right-hand rule.

We summarize the properties of the random walk in the following theorem.

Theorem 3. *For any fixed ϵ, the random walk uses $O(m^2)$ steps to with probability at least $1 - \epsilon$ find the treasure and exit the maze, for a maze with m edges. The memory required is $O(\log n)$, where n is the number of vertices.*

As with the other strategies we have considered, the random walk is not guaranteed to find the shortest path. If finding the shortest path is important, breadth first search could be used, although we have not given a "physically realizable" implementation of breadth first search.

3 Conclusion

We will come back to the problem of navigating mazes later in the course - giving a more formal treatment and proving some of the unproven claims of today's lecture.