**(Recursive) Backtracking**

* Many programming problems can be solved by systematically searching a ***set of possibilities***.
  + If you want to find a path through a maze from a starting point to an exit point, you can explore **all possible paths** through the maze until you find one that works.
  + If you want to find all possible moves and countermoves in a game of tic-tac-toe, you can explore all possible moves to see if there is some move that guarantees that you a win.
* Many of these **exhaustive search problems** can be solved with an approach called *backtracking*. It is a problem-solving approach that is nicely expressed using **recursion**.
* As a result, it is sometimes referred to as *recursive backtracking*.

**Formal Definition**

* Recursive Backtracking is a general algorithm for finding solutions to a problem by **exploring possible candidate solutions and abandoning (“backtracking”) once a given candidate is deemed unsuitable**.
* Every backtracking problem involves a ***solution space***of possible answers that you want to explore.
* We try to view the problem as a sequence of ***choices***, which allows us to think of the solution space as a ***decision tree***.
* Backtracking involves searching all possibilities, so it can be an expensive technique to use.
* But many problems are small enough in scope that they are nicely solved with a backtracking approach.

**A Simple Example: Traveling North/East**

* To introduce the basic concepts and terminology of backtracking, let’s explore a simple example.
* Consider a standard Cartesian plane with (*x*, *y*) coordinates.
* Suppose that you start at the origin, (0, 0), and you are allowed to repeatedly make one of these three moves:
  + You can move North (abbreviated “N”), which will increase your *y*-coordinate by 1.
  + You can move East (abbreviated “E”), which will increase your *x*-coordinate by 1.
  + You can move North-East (abbreviated “NE”), which will increase both your *x*-coordinate and *y*-coordinate by 1.
* Starting from the origin, these three different moves would leave you in the locations shown in Figure 12.7.

Diagram

Description automatically generated with medium confidence

* We can think of this as a traveling problem where we can make a series of moves that take us from the origin to some other (*x*, *y*) point.
  + For example, the sequence of moves N, NE, N would leave us at (1, 3).
* For our traveling problem the ***choices*** are the sequence of moves that we make.
* Here is a a ***decision tree*** illustrating the ***solution space*** of all the possible ways to make two moves.
* These decision trees can be quite large even for a small problem like this.

Diagram

Description automatically generated

* Consider the problem of traveling from the origin to the point (1, 2).
* What are the possible sequences of moves that would get you there?
* People are fairly good at solving problems like these, so you can probably pretty easily come up with all five of the possibilities:
  + N, N, E
  + N, E, N
  + N, NE
  + E, N, N
  + NE, N
* How would we write a computer program to find these solutions?
* For this simple problem, we could devise a specialized algorithm that takes into account the properties of these paths, but backtracking provides a convenient way to exhaustively search all possibilities.
* For most backtracking solutions we end up writing two methods.

1. We generally want a public method that is passed the details of the problem to be solved.
2. But we almost always need extra parameters for the backtracking, so we almost always have an extra private method that does the actual backtracking.

* The basic form that our backtracking solution will take is a method to explore all possible choices recursively:

Logo

Description automatically generated with low confidence

* Because we are using recursion, we need to identify
  + base cases

and

* + recursive cases
* Backtracking solutions generally involve **two different base cases**. You tend to stop the backtracking when you find a solution, so that becomes one of our base cases. Often what you want to do when you find a solution is to report it: