The simplest possible RRT algorithm has been so successful because it is pretty easy to implement. Things tend to get complicated when you:

* need to visualise planning concepts in more than two dimensions
* are unfamiliar with the terminology associated with planning, and;
* in the huge number of variants of RRT that are have been described in the literature.

**Pseudo code**

The basic algorithm looks something like this:

1. Start with an empty search tree
2. Add your initial location (configuration) to the search tree
3. while your search tree has not reached the goal (and you haven't run out of time)

3.1. Pick a location (configuration), q\_r, (with some sampling strategy)

3.2. Find the vertex in the search tree closest to that random point, q\_n

3.3. Try to add an edge (path) in the tree between q\_n and q\_r, if you can link them without a collision occurring.

Although that description is adequate, after a while working in this space, I really do prefer the[pseudocode of figure 5.16 on RRT/RDT](http://planning.cs.uiuc.edu/node231.html) in Steven LaValle's book "Planning Algorithms".

**Tree Structure**

The reason that the tree ends up covering the entire search space (in most cases) is because of the combination of the sampling strategy, and always looking to connect from the nearest point in the tree. This effect is described as reducing the [Voronoi bias](http://msl.cs.uiuc.edu/~lavalle/papers/LinLav04.pdf).

**Sampling Strategy**

The choice of where to place the next vertex that you will attempt to connect to is the sampling problem. In simple cases, where search is low dimensional, uniform random placement (or uniform random placement biased toward the goal) works adequately. In high dimensional problems, or when motions are very complex (when joints have positions, velocities and accelerations), or configuration is difficult to control, sampling strategies for RRTs are still an open research area.

**Libraries**

The [MSL library](http://msl.cs.uiuc.edu/msl/) is a good starting point if you're really stuck on implementation, but it hasn't been actively maintained since 2003. A more up-to-date library is the [Open Motion Planning Library (OMPL)](http://ompl.kavrakilab.org/). You'll also need a good collision detection library.

**Planning Terminology & Advice**

From a terminology point of view, the hard bit is to realise that although lots of the diagrams you see in the (early years of) publications on RRT are in two dimensions (trees that link 2d points), that this is the*absolute* simplest case.

Typically, a mathematically rigorous way to describe complex physical situations is required. A good example of this is planning for a robot arm with n- linkages. Describing the end of such an arm requires a minimum of n joint angles. This set of minimum parameters to describe a position is a **configuration**(or some publications **state**). A single configuration is often denoted q

The combination of all possible configurations (or a subset thereof) that can be achieved make up a**configuration space** (or **state** space). This can be as simple as an unbounded 2d plane for a point in the plane, or incredibly complex combinations of ranges of other parameters.