**[Exercise: Iterative Deepening Search](https://courses.cs.northwestern.edu/325/exercises/graham-exs.php" \l "gen-ids)**

Function names: **ids**, **dls**, **shortest-path**

Test cases: use the **shortest-path** tests, as described below.

Simple breadth-first search on large very bushy networks is impractical. If there are dozens or hundreds of branches, the queue quickly grows to 1000s of elements. But if you're looking for an optimal solution, e.g., the shortest path, what else can you do?

One answer that seems at first glance to be even worse than breadth-first search is this:

* Try to find an answer with depth-first search limited to a depth of 1.
* If that fails, start over and try to find an answer with depth-first search limited to a depth of 2.
* If that fails, start over and try to find an answer with depth-first search limited to a depth of 3
* Repeat until either you find an answer.

The iterative deepening algorithm seems like it will be too expensive because step 2 will repeat all the work done in step 1, step 3 will repeat step 2 and so on.

However, while this repeated work does happen, the combinatorics of trees, specifically that there as many or more nodes at layer N than in the entire tree above layer N, turns out to make [**the total runtime cost of this algorithm very competitive with BFS**](https://en.wikipedia.org/wiki/Iterative_deepening_depth-first_search), with much smaller memory costs. The same idea can be applied [**to more intelligent search algorithms**](http://www.sciencedirect.com/science/article/pii/0004370285900840).

Define the function **ids** (iterative deepening search) to take the same arguments as **[bfs](https://courses.cs.northwestern.edu/325/exercises/graham-exs.php" \l "gen-bfs)**. **ids** should repeatedly call a depth-first search function -- call it **dls** for depth-limited search -- that does a simple depth-first search to find an extension of *path*. **dls** should take an additional parameter *n*, and should

* generate new states by applying *gen* to *path*
* stop and return the first *path* found such that *pred* is true of the CAR of *path*
* stop and fail if the depth of search reaches *n*

**dls** should not need a queue. **dls** should avoid cycles, i.e, it should not add states already in path.

There is one tricky point. If you look at the algorithm as described on [**Wikipedia**](https://en.wikipedia.org/wiki/Iterative_deepening_depth-first_search) and elsewhere, it has a problem: it loops forever if there is no solution!

One option would be to pass a depth-limit parameter to **ids** to tell it when to stop trying. This is useful to have. It's necessary for both **ids** and **bfs** if infinitely long paths are possible. But it's not a general or satisfying approach when infinitely long paths are not possible.

**bfs** can handle searches that fail as long as there there no infinite paths, because, eventually, nothing new can be added to the queue, and all options have been explored.

To do the same thing, **ids** and **dls** need to be able to distinguish two kinds of search failure:

* True failure, when search could not extend a path any further
* Depth failure, when search reached the depth limit

If all the depth-first searches have true failure at depth N, then there's no point to going any deeper.

Implement **ids** and **dls**, and any subfunctions you need. Test by defing **shortest-path** to call **ids** and confirm that all tests pass, including those where search fails. Your code should be efficient and clean, including the handling of the two kinds of failure.