CMPS 2200 Assignment 5

In this assignment we'll look at the greedy and dynamic programming paradigms.

To make grading easier, please place all written solutions directly in `answers.md`, rather than scanning in handwritten work or editing this file.

All coding portions should go in `main.py` as usual.

Part 1: Making Change

The pandemic is over and you decide to take a much needed vacation. You arrive in a city called Geometrica, and head to the bank to exchange N dollars for local currency. In Geometrica they have a currency that is 1-1 with U.S. Dollars, but they only have coins. Moreover the coins are in denominations of powers of 2 (e.g., k denominations of values 2^0, 2^1, ..., 2^k). You wonder why they have such strange denominations. You think about it a while, and because you had such a good Algorithms instructor, you realize that there is a very clever reason.

1a) Given a N dollars, state a greedy algorithm for producing as few coins as possible that sum to N. Please discuss if this algorithm is optimal or not.

enter answer in `answers.md`

1b) What is the work and span of your algorithm?

enter answer in `answers.md`

Part 2: Making Change Again

You get tired of Geometrica and travel to the nearby town of Fortuito. While Fortuito also has a 1-1 exchange rate to the US Dollar, it has an even stranger system of currency where any given bank has a completely arbitrary set of denominations (k denominations of values $D_0, D_2, \ldots, D_k$). There is no guarantee that you can even make change. So you wonder, given N dollars is it possible to even make change? If so, how can it be done with as few coins as possible?

2a) You realize the greedy algorithm you devised above doesn't work in Fortuito. Give a simple counterexample that shows that the greedy algorithm does not produce the fewest number of coins. Please discuss why greedy algorithm cannot work optionally.

enter answer in `answers.md`

2b) Use this optimal substructure property to design a dynamic programming algorithm for this problem. If you used top-down or bottom-up memoization to avoid recomputing solutions to subproblems, what is the work and span of your approach?

enter answer in `answers.md`

Part 3: Reachable Graph

3a) Let's assume we're using the "Map of Neighbors" representation for undirected graphs. The provided `make_undirected_graph` function will make a graph using this representation given a list of edge tuples.

We'll start by implementing the `reachable` function, which identifies the set of nodes that are reachable from a given `start_node`.

As discussed in lecture, we'll maintain a set called `frontier` that keeps track of which nodes we will visit next. We initialize the set to be the start node. We then perform a loop where we pop a single node off the frontier, visit its neighbors, and update the `result` and `frontier` sets appropriately. At the end of the loop, `result` should contain all the nodes that are reachable from `start_node`.

Complete the `reachable` implementation and test with `test_reachable`. Think about how to make this efficient and ensure we don't revisit nodes unnecessarily.

.
.
.
.
.
.

3b) Next, we will use the `reachable` function to determine if a graph is connected or not. Complete the `connected` function and test with `test_connected`.

.
.
.
.
.
.

3c) Next, we'll use `reachable` to determine the number of connected components in a graph. Complete `n_components` and test with `test_n_components`. Again, think about how to minimize the number of calles to `reachable` you must make.

.
.