1 NP-complete problems

NP-complete problems are hard to solve quickly. There doesn't really exist an algorithm to do this. We can get approximate solutions, but the exact solution has to be obtained through brute-force. There are no known fast solutions for NP-complete problems, the best thing to do is to use some approximate algorithm (e.g. a greedy algorithm). Examples of NP-complete problems are:

- set cover problem
- travelling salesman problem

To give a more concrete idea, consider the following situations:

- A postman needs to deliver to 20 homes. He needs to find the shortest route that goes to all 20 homes (example of travelling salesman)
- Find the largest clique in a set of people (where clique is a set of people who all know each other)
- You're making a map of the world and need to colour adjacent countries with different colours. You have to find the minimum number of colours you need so that no two adjacent countries are the same colour.

All the above examples are NP-complete.

1.1 Travelling salesman

The travelling salesman problem is an NP-hard problem and is formulated as follows: *'given a list of cities and distances between each pair of cities, what is the shortest possible route that visits each city and returns to the original city?'*

How would we go about solving this problem? Well, suppose we have n cities. There are n different cities we can choose from to start. For each starting choice, there are now $n-1$ choices for the first city to visit from our starting city. After we've visited the first city, there are $n-2$ cities left to choose from, etc.. etc.. Therefore, there are $n!$ possible options to choose from. To find the optimal option, we'd have to try out every option, calculate the total distance, and pick the option with minimum distance. This means our problem gets solved in $O(n!)$.

Can we do better than this? Well, not really, if we want to be sure to get the optimal solution. However, if we just want a 'good' solution, then we have other options.

1.1.1 Using a greedy algorithm

We can pick the starting city randomly. Then, we pick the closest unvisited city. Easy enough.

1.2 Set cover problem

Given a set of elements $\{1, 2, \ldots, n\}$ (the universe), and a collection of m sets whose union equals the universe, what is the smallest sub-collection of sets whose union equals the universe?

For example, suppose you're starting a radio show in the US. You want peole from all 50 states to be able to listen to you. You have to decide which radio stations to use. You have a list of stations and for each of them you know which states they cover. For example, station one might cover {Kentucky, Indiana, Tennessee}, station two covers {Tennessee, Georgia, Mississippi}, station three covers {Nevada}, etc.. etc.. Now your task is: *what is the smallest set of stations you can pay on to cover all 50 states?*.

There set problem can be weighted, meaning that the sets are assigned weights. For example, it might be more expensive to use certain stations than others, and you're trying to find the cheapest set of stations to cover all 50 states.

The set-covering problem has a running order of $O(2^n)$ because there are 2^n subsets in total.

1.2.1 Greedy algorithm

Again, a greedy algorithm can help us. Using the example above, we can:

1. pick the station that covers the most states that have not been covered yet
2. repeat until all statiosn have been covered

This greedy algorithm runs in $O(n^2)$ time (we have to search through every one of the n possible stations to find the one which covers the most yet-uncovered states, and we have to do this in each step, i.e. $n \times n$)

2 TODO

- Binary search (write proof, do coding too)
- Binary search trees (summarize, do coding too)
- read hash table section, summarize what makes a good hash table, read about SHA.