20190925 + 20191001

Greedy Algorithms

Suppose we are planning a trip from Kingston to Montreal (and suppose there is only one road so we don't have a choice of which route to take). Our car will need several stops to fill up the gas tank. We know the location of all the service stations on the highway between Kingston and Montreal, and we know exactly how far we can travel on a tank of gas. We want to minimize the number of stops we make along the way.

The intuitive solution, of course, is to go as far as possible on each tank - that is, stop at a service station iff we can't make it to the next one. It turns out that in this case intuition can be trusted - this is exactly the right solution.

But how can we ...

- prove that this algorithm always finds the correct (i.e. optimal) solution?
- determine the computational complexity of this algorithm?

We can formalize this idea into an algorithm as follows. I'll show a recursive formulation first – here we will define a function RT(i) which is supposed to find an optimal solution when we are leaving station s_i with a full tank of gas ... RT(0) will give us the complete solution.

We will assume that the problem has a solution (ie that there is no gap between consecutive stations that exceeds the distance we can travel on one tank of gas ... so when we fill up, we know we can reach the next station). It is easy to test to make sure this condition is satisfied. If it is not, we can immediately conclude there is no solution – we are done.

```
1.      Sort the stations according to how far they are from Kingston
        Let S = {s_0, s_1, s_2, ... s_n, s_{n+1}} be the sorted set, with s_0 = Kingston
                                                    and s_{n+1} = Montreal

2.  def RT(i)       # i is the number of the station we are leaving
                    # with a full tank

                    # We know we have enough gas to reach the next station
        t = i+1

        while t ≠ n+1  and station s_{t+1} is reachable
            # don't stop at station t
            t++

              if t == n+1:   STOP            # we have reached Montreal

        else :
              fill up at station s_t
              RT(t)
```

We find the solution by calling RT(0)

To complete our consideration of this problem, we need to do several things:

1. Show that the problem does have an optimal solution (some problems don't, either because there is no solution or there is no bound on the value of the potential solutions).

2. Show that the greedy algorithm finds an optimal solution.

3. Determine the complexity of the algorithm.

First, once the stations are sorted we can easily determine if there are any feasible solutions: the trip is possible iff there is no gap between stations too long for us to cover on one tank of gas. Assuming then that there are feasible solutions, it is clear that there is at least one optimal solution since we can rank all the feasible solutions according to their cardinality and choose one with minimum cardinality.

It is also clear that the algorithm will find some solution to the problem: it never lets us run out of gas, and it gets us to the destination.

Let's also dispose of the complexity question: sorting the n stations takes O($n * \log n$) time. Choosing the stations where we stop takes O(n) time since we look at each station exactly once, and decide in constant time whether or not to stop there. Thus the entire algorithm takes O($n * \log n$) time ... or just O(n) if the stations are given to us already sorted.

Now the proof of correctness. This is the interesting part. We use Proof by Induction.

Let n be the number of stations between Kingston and Montreal.

Base Case: If n = 0, there are no possible stops between Kingston and Montreal. If there is any feasible solution (ie if we can reach Montreal on a single tank), then the algorithm's action is correct (it makes no stops before Montreal). Thus for n = 0 the algorithm finds an optimal solution.

Inductive Hypothesis

We assume that the algorithm finds an optimal solution when we have a full tank and the number of service stations still ahead of us is $\leq$ k, for some k ≥ 0

Inductive Step

Suppose there are k+1 service stations still ahead of us and we have a full tank when we leave Kingston.

Let the algorithm's solution be $A = \{a_1, a_2, \ldots\}$ ... the solution lists the stations chosen for stops. So a_1 is the first stop and a_2 is the second stop, but we don't know how many stops the algorithm will make to complete the journey.

The proof proceeds in three stages:

Stage 1: we show that there exists at least one optimal solution that starts with the algorithm's first choice. To do this, let O be some optimal solution to the problem, i.e. O = $\{o_1, o_2, \ldots\}$ where each element is a station where we should fill up with gas, in order.

Consider a_1 : the algorithm's first stop. From the definition of the algorithm we know it is not possible to reach any station beyond a_1 on the first tank of gas. Therefore $o_1 \leq a_1$

But now consider o_2, the second stop in O. It must be reachable from o_1 ... which means it is also reachable from a_1. Thus we can create a new feasible solution:

Let O* = $\{a_1, o_2, \ldots.\}$

Note that |O*| = |O|, so O* is also optimal. Thus there is an optimal solution that contains a_1.

Now we know that the algorithm's first decision is 'safe' - ie there is at least one optimal solution that matches this decision.

Stage 2: we show that the rest of the algorithm's solution is optimal for the reduced problem (after we stop at a_i, there are fewer stations left between us and Montreal)

When we make a decision about the first stop, the problem reduces in size to $\leq$ k (i.e. the number of remaining stations between us and Montreal is reduced, and our tank is full again). By the IH, the algorithm will find an optimal solution to this reduced problem.

Stage 3: The last step of our proof is to show that when we combine the algorithm's first choice with its solution to the reduced problem, we get an optimal solution to the original problem.

This is the stage of the proof that often perplexes people. Since we know that the algorithm's first decision is part of an optimal solution, and we also know that the algorithm finds an optimal solution on the reduced problem, ***why*** can't we just conclude that the combination of optimal first choice and optimal "rest of the solution" must be optimal?

The reason we have to do this is because it is conceivable that the algorithm's first choice may only belong to one optimal solution. When we reduce the problem and get an optimal solution to the reduced problem, we might get an optimal solution different from the one that the first choice belongs to. (Remember that it is possible for the problem to have many optimal solutions.) In this situation it might be possible that the combination of the first choice and the selected optimal solution to the reduced problem might not even be a feasible solution, let alone optimal.

If you are struggling to see how this can happen with this road-trip problem, you can pat yourself on the back and say "Truly the intuition is strong in this one" ... because the situation I just described cannot happen in this problem. We still have to prove it though ... so let's do that.

We know that there is at least one optimal solution that matches the algorithm's first choice s_i – we identified O* above as one such, so let's use that one again: O* = $\{a_1, o_2, o_3, \ldots\}$

Let the algorithm's solution be A = $\{a_1, a_2, \ldots\}$, as before.

Due to the $^{s_j, \ldots s_{n+1}}$structure of the algorithm, we know A is a feasible solution.

Observe that $\{a_2, \ldots\}$ and $\{o_2, \ldots\}$ are both solutions to the problem of getting from station a_1 to Montreal, based on the decision made to fill up at station a_1.

In fact, we know $\{a_2, \ldots\}$ is an **optimal** solution to this reduced problem of getting from station a_1 to Montreal.

Thus $|\{a_2, \ldots\}| \leq |\{o_2, \ldots\}|$. Since adding a_1 to both these sets gives A and O* , it is clear that $|A| \leq |$O*$|$. Thus A is also an optimal solution.

This completes the proof that RT(0) finds an optimal solution to the Road Trip problem.

Based on this example we can state the general Greedy Algorithm paradigm:

The fundamental principle of greedy algorithms is "take the best thing first".

More formally, suppose we are trying to solve an **optimization problem** that involves choosing objects from a set, subject to some feasibility constraint. We want to choose the objects that will maximize (or minimize) the total value of the solution, while satisfying the constraint. The **Greedy Paradigm** looks like this:

1. Sort the objects according to some criterion
2. repeat
   select the next item in the list if it does not violate the feasibility constraint
   until no more selections can be made

For the Road Trip problem, the algorithm can be phrased as

1. Sort the objects according to the distance from the start
2. repeat
   skip the next station if that doesn't cause us to run out of gas
   until we reach the destination

Our original goal was to minimize the number of stations we stop at ... but this is exactly equivalent to maximizing the number stations we skip.

The Greedy Algorithm is such a simple idea that we cannot expect it to find optimal solutions for all problems (for example, it is very easy to define greedy algorithms that find solutions to some instances of a class of very difficult problems problems ... but there will be instances of the problem for which the algorithm's solutions won't be optimal).

However, there is a huge class of problems for which simple greedy algorithms **will** always find the optimal solution. This means that our big job with respect to Greedy Algorithms will not be coming up with the algorithm, but finding a proof that the algorithm gives the optimal solution for all instances of the problem.