INTRODUCTION TO

FORMAL REASONING ABOUT PROGRAMS

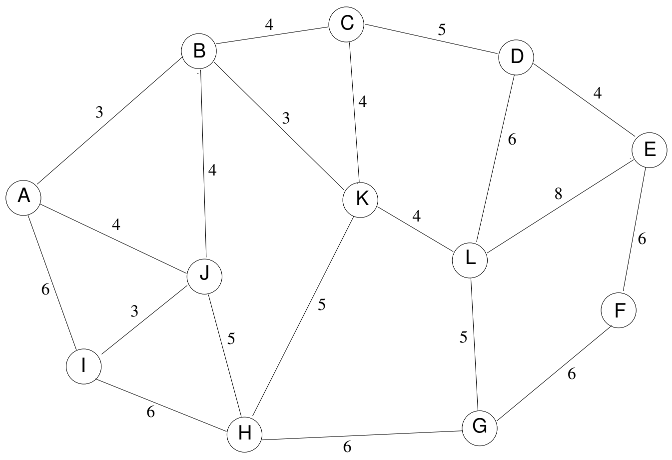
Paving Iqaluit

Problem: Iqaluit is a muddy place, and city council would like do some repaving. Their goal is to make sure that citizens can get from any intersection to any other intersection using only paved segments of road. And they want to do it as cheaply as possible.

On the next page is a diagram of Iqaluit.

- Each circle represents an intersection. They are labelled so we can conveniently refer to them.
- Each line represents a segment of road.
- The number on each line represents how expensive it would be to pave.

What is the cheapest solution?



Knowing you have the best answer

Best solution:

How many solutions did we consider?

How many possible solutions are there?

- There are 20 edges in this particular graph.
- Each edge can either be paved or not.
- So there are $2^{20} = 1,048,576$ possible pavings.
- (Some will not meet our requirement that you can get from anywhere to anywhere, but we won't know until we check.)

So how do we ever know we have the best solution? Hm.

A General Strategy

Suppose we believe we have the best paving for Iqualuit.

Can we come up with a strategy that will find the best paving for *any* city?

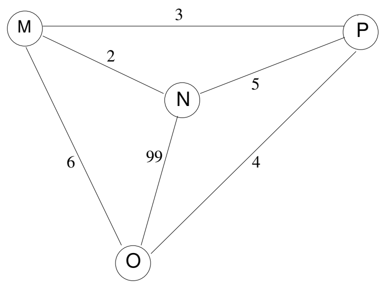
A proposed general strategy:

- Start from some arbitrary node.
- Pave a cheapest edge going out of that node. Now two nodes are "connected" (*i.e.*, reachable on paved roads).
- Repeat until every node is connected:
  - Pave the cheapest edge that links a connected node to an unconnected node

Is the strategy correct?

The strategy certainly seems reasonable.

See if it works on this graph for Wainfleet:



Do you believe it works on *every* graph you might give it?

Some Counter-evidence

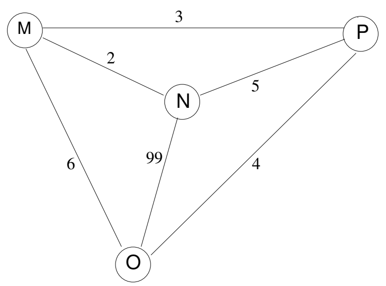
This is an example of a "greedy" strategy; it always grabs the best choice among those remaining.

Another problem: Find the shortest route that takes you to every intersection in the city and back to where you started, without visiting any intersection more than once.

(This is the "travelling salesperson problem", which is famous in computer science.)

Greedy strategy: Pick the cheapest edge to start. Then from wherever you are, always pick the cheapest edge that takes you to somewhere you haven't been before.

Use this greedy strategy to solve the Travelling Salesperson Problem for Wainfleet:



We were backed into a corner, and wound up with a terrible solution.

And note that this very same graph was not a problem for the greedy paving strategy!

Conclusion: A greedy strategy isn't necessarily guaranteed to always give the best solution.

So now we would be naive to trust that our greedy strategy for paving gives the best solution for any graph.

On the other hand, maybe it is indeed correct. What would it take to convince you?

Knowing your Strategy is Correct

In fact, our greedy paving strategy *does* work for all graphs. It is known as Prim's algorithm.

Here's a sketch for an argument that it works:

1. There must be one paving that is the best one (or more, if there is a tie).
2. Before we start choosing edges, we have chosen an (empty) set of edges that is (trivially) a subset of a best paving.
3. Every time we add another edge, we still have a subset of a best paving.
4. So when done, we must still have a subset of a best paving.
5. But everything is connected (or we wouldn't have stopped), so our "subset" of a best paving must be a complete best paving.

Crunchy nugget: Item (3) is hard to believe. But certainly *if* we believed it, we would know our greedy strategy works.

Assertions

Statements (2) and (3) have this general form:

> Whenever we reach this point in the algorithm, such-and-such is true.

Recall that such statements are called **assertions**.

Assertions form the basis of any argument that an algorithm is correct.

Efficiency of the Greedy Strategy

How much work is required to solve the paving problem using the greedy strategy?

The answer depends, of course, on how big the graph is – how many nodes and edges it has. Let n be the number of nodes in the graph and e be the number of edges.

Recap of the strategy:

- Start from some arbitrary node.
- Pave the cheapest edge going out of that node. Now two nodes are "connected" (*i.e.*, reachable on paved roads).
- Repeat until every node is connected:
  - Pave the cheapest edge that links a connected node to an unconnected node

Most of the work is done inside the loop. How many times does the repeat loop iterate:

And how much work is done to pick the next cheapest edge:

(This is an over-estimate; the work gets less each time because we have fewer edges left to consider.)

So in total, the work done is "on the order of":

How does e compare to n?

If the graph has no unconnected parts e is at least equal to $n-1$, and it can be a lot more: if the graph is complete, $e = \frac{n^2-n}{2}$.

Eff. of the greedy strategy, in terms of n

We determined that it takes time that is on the order of $n \times e$, and that $(n-1) \leq e \leq \frac{(n^2-n)}{2}$.

So in the worst case, the time taken is on the order of n^3 (making some crude mathematical simplifications).

In fact, we can speed up the greedy strategy using a simple trick, so that it takes time on the order of $\boxed{n^2}$.

Efficiency of a brute force strategy

How does this compare to a "brute force" strategy — one that simply works through every single possible paving and finds which is the cheapest?

How many possible pavings are there to check through:

So in the worst case, the time taken for the brute force strategy is on the order of $\boxed{2^{n^2}}$ (again making some crude mathematical simplifications).

Comparing n^2 to 2^{n^2}

	Approximate Value for $n =$				
function	10	100	1,000	10,000	10^5
n^2	100	10,000	10^6	10^8	10^{10}
2^n	1024	10^{30}	10^{300}	10^{3000}	$10^{30,000}$
2^{n^2}	10^{30}	10^{300}	humungous!		

Note that n can be very large — much larger than 10^5 — in many real-world applications of the paving problem.

So the greedy approach is *vastly* more efficient than a brute force approach.

(Too bad greedy approaches don't always work!)

Major Conclusions

Formal reasoning about correctness

There are some algorithms whose correctness can only be established with careful reasoning. We need techniques for reasoning about correctness so we can be confident that our software works.

You will learn about this in subsequent courses.

Formal reasoning about efficiency

For a given problem, there may be some solutions that are vastly more efficient than others. We need techniques for analyzing efficiency so that we can write software that runs quickly.

You will learn more about this now. ...