We started with a sketch of the proof that Dijkstra’s Algorithm correctly finds the least-weight paths from A to all other vertices in the graph. We use induction to prove this result.

Claim: after each iteration of the main while-loop, the algorithm has correctly identified the least-weight path from A to one more vertex.

Base Case: after the first iteration the algorithm has moved A to the set R, with a computed distance of 0 and a path containing no edges. This is correct because it costs nothing to travel from A to A.

Inductive Step: Suppose that after the ith iteration, for some \( i \geq 1 \), the algorithm has correctly computed the distance and least-weight paths to all vertices in R. If there is no following iteration, the claim is true. If there is a following iteration, let \( x \) be the vertex chosen from C, and let \( \{v_1, v_2, ..., v_t\} \) be the other vertices in C. Suppose there is a path from A to \( x \) that has lower total weight than the path chosen by the algorithm. Then that preferable path would have to start at A, continue through some vertices in R, then jump to one of the \( v_i \) vertices, then continue on to \( x \). But \( x \) was chosen because \( B(x) \leq B(v_i) \ \forall \ i \). Thus the supposed preferable path must have weight \( \geq B(x) \), and so it cannot actually be a lower weight path from A to \( x \). Thus the algorithm’s decision to move \( x \) to R is correct.

We examined the relationship between Dijkstra’s Algorithm and Prim’s Algorithm. The two algorithms have **exactly** the same structure. The only difference is in how the \( B(v) \) values are computed. Where Dijkstra’s Algorithm has
if $B(x) + w(x,y) < B(y)$:
    if $B(y) = \infty$:
        add $y$ to $C$
    $B(y) = B(x) + w(x,y)$

Prim’s Algorithm has

if $w(x,y) < B(y)$:
    if $B(y) = \infty$:
        add $y$ to $C$
    $B(y) = w(x,y)$

Thus Prim’s Algorithm ignores the weight of the path and focuses completely on the weight of the individual edges.

The result of this tiny change is that Prim’s Algorithm discovers a Minimum Weight Spanning Tree, instead of least-weight paths from $A$.

(Dijkstra actually discovered Prim’s Algorithm independently a couple of years after Prim did ... and another researcher named Jarnik had discovered it before either of them, so the algorithm goes by several different names.)

This is an excellent example of how we can take an algorithm and adapt it to solve new problems.

We discussed whether or not Dijkstra’s Algorithm can be adapted to find greatest-weight paths rather than least-weight paths. It may seem plausible that we could do this by changing all the $\leq$ in the algorithm to $\geq$, or by multiplying all the edge weights by -1 and applying the algorithm as
is. Unfortunately these tactics do not work – you may want to construct examples of graphs on which they fail to find greatest-weight paths.

It turns out that we have very strong reasons to believe that no simple modification of Dijkstra’s Algorithm can convert it to finding greatest-weight paths. This is a very powerful result that follows immediately from our study of NP-Completeness, which we will begin on Tuesday.

But before finishing with Dijkstra’s Algorithm, let’s look at how we can adapt it to solve one more problem: the maximum band-width problem presented earlier this week as Problem 1.

To recap: we have a network represented by a graph with weighted edges, where the weight of an edge represents the rate at which data can flow along that edge. For any path in this graph, the rate of flow along the path is equal to the minimum edge weight in the path: the edge with the smallest weight in the path acts as a bottleneck for the whole path. The goal is to start at some vertex A and determine the paths from A to all other vertices that maximize the flow rate from A to the target vertex.

As with the conversion of Dijkstra’s Algorithm to Prim’s Algorithm, the changes need to solve this new problem are minimal. First, since we initially have no flow rate from A to any other vertex, we initialize $B(v) = 0$ for all vertices other than A, and we initialize $B(A) = \infty$.

In Dijkstra’s Algorithm we compute the cost of a path that ends with the edge $(x, y)$ as $B(x) + w(x, y)$ ... this makes sense because we are interested in the total weight of the path. But in this new problem we are interested in the minimum edge-weight in the path. For each vertex $y$, we want $B(y)$ to be the minimum edge-weight in the path from A to $y$. So the value of a path that ends with the edge $(x, y)$ is $\min\{B(x), w(x, y)\}$, because the lowest-weight edge in the path is either $(x, y)$ or it is somewhere in the path from
A to x ... and that will be the value of B(x).

The last change we have to make is that instead of choosing the candidate with the lowest B() value, we choose the candidate with the highest. This is because (using reasoning exactly analogous to the inductive proof of correctness for Dijkstra’s Algorithm) we know there cannot be a better path to this candidate: all the other paths include edges limit their flow rate to \( \leq \) the one we are choosing.

Putting all of this together, we initialize B as

\[
\begin{align*}
B(A) &= \infty \\
B(v) &= 0 \text{ for all } v \neq A
\end{align*}
\]

We replace

let x be the vertex in C with minimum B() value

with

let x be the vertex in C with maximum B() value

and we take the part of Dijkstra’s Algorithm

\[
\begin{align*}
\text{if } B(x) + w(x,y) &< B(y): \\
&\quad \text{if } B(y) == \infty: \\
&\quad\quad \text{add } y \text{ to } C \\
&\quad B(y) = B(x) + w(x,y)
\end{align*}
\]
and replace it by

```python
if min(B(x), w(x,y)) > B(y):
    if B(y) == 0:
        add y to C
    B(y) = min(B(x), w(x,y))
```

Having earlier asserted that Dijkstra’s Algorithm cannot be adapted to find greatest-weight paths, it may seem odd that we have now successfully adapted it to find greatest-flow paths. However, there it is. The power and limits of Dijkstra’s Algorithm are suitable topics for deep meditation, and may either serve as a cure for insomnia or to induce it.