1 Return to Shortest Paths

Let’s look at the ever-popular shortest path problem.

- Given a weighted, directed graph $G = (V, E)$, a source vertex $s$, and a target vertex $t$, find a shortest path in $G$ from $s$ to $t$.
- For fun, let’s allow $G$ to have negative-weight edges.
- (Clearly, these aren’t distances – could be bonuses/penalties)
- A “shortest path” is now one with weight less than all others (can be negative)
- (Note that Dijkstra’s algorithm cannot deal with this case – it breaks if edge weights are $< 0$.)
- However, we will forbid negative-weight cycles
- If a cycle with weight $< 0$ exists, then a path can take arbitrarily many trips around it to get arbitrarily negative weight!
- Without negative-weight cycles, shortest paths are WLOG acyclic.

Let’s consider a DP approach to this problem.

2 A Stab at an Algorithm

How might you attack shortest paths via DP?

- Let the first choice identify an intermediate vertex $k$ on a shortest path between $s$ and $t$, or assert that there is no such vertex (i.e. the path is just the edge $s \rightarrow t$).

- Complete Choice: Consider a shortest path $\pi$ from vertex $s$ to vertex $t$.
- Either $\pi$ is a single edge $s \rightarrow t$ (no subproblem), or there is an intermediate vertex $k$ on the path.

- Inductive Structure: If we say that $\pi$ from $s$ to $t$ passes through $k$, we are left to determine how to get from $s$ to $k$ and from $k$ to $t$. These look like inputs to shortest path problem, and they are unconstrained given choices of $s$, $t$, and $k$. 
• (We’re allowing repeated vertices – it’s OK, paths with cycles are feasible, though they are not in general shortest.)

• **Optimal Substructure**: Suppose that for choice \( k \), shortest subpaths have lengths \( \ell(s, k) \) and \( \ell(k, t) \).

• Observe that total length of concatenated path is
  \[
  \ell(s, t) = \ell(s, k) + \ell(k, t).
  \]

• Apply contradiction argument to show that this path is shortest among all those that pass through \( k \). QED

Nifty. Let’s develop a recurrence.

• **General subproblem**: let \( D(i, j) \) be score of shortest path from vertex \( i \) to vertex \( j \).

• Using our decomposition by choices,
  \[
  D(i, j) = \min \left\{ w(i, j) \min_k D(i, k) + D(k, j) \right\}
  \]

• And the base case is... (ask)

• We can’t actually find one! There is no starting point, as even path from vertex 1 to vertex 2 could go through vertex 37.

• So much for this problem formulation!

• The real issue: our “subproblems” are not actually smaller in any sense – inductive structure argument was incorrect!

Now what do we do?

## 3 Alternate Plan B

Following algorithm is due to Floyd and Warshall.

• Number the vertices of \( G \) \( 1 \ldots n \).

• A shortest path from \( s \) to \( t \) has some largest intermediate vertex \( k \) (not the end-points).

• (If \( k = 0 \), path must be a single edge from \( s \) to \( t \).)

• **Complete Choice**: choose largest intermediate vertex on path to be one of \( 0 \ldots n \)

• (Could skip \( s \) and \( t \) if desired, since WLOG path need not have cycles.)
• **Inductive Structure**: If shortest path from \( s \) to \( t \) has largest intermediate vertex \( k > 0 \), then it decomposes into shortest subpaths from \( s \) to \( k \) and from \( k \) to \( t \).

• Again, no constraints given \( s \), \( t \), and \( k \).

• **Check**: do we have a basis for induction now?

• WLOG, shortest path from \( s \) to \( t \) is acyclic, since no cycle can decrease its total length.

• Hence, \( k \) does not occur twice, and so *two subpaths have largest intermediate vertex* \( \leq k - 1 \).

• **Optimal Substructure**: Suppose shortest subpaths given max vertex \( k \) have lengths \( \ell(s, k) \) and \( \ell(k, t) \).

• Total length of resulting path is \( \ell(s, k) + \ell(k, t) \).

• Contradiction argument shows that this path must be a shortest path from \( s \) to \( t \) given max vertex \( k \). QED

OK, recurrence time again.

• Arbitrary subproblem involves getting from node \( i \) to node \( j \) with max intermediate vertex equal to \( k \).

• Three parameters, so index problem as \([i, j, k]\).

• Let \( D[i, j, k] \) be length of shortest path from \( i \) to \( j \) with largest intermediate vertex \( k \).

• By substructure, we can write

\[
D[i, j, k] = \min_{p, q < k} (D[i, k, p] + D[k, j, q])
\]

\[
= \min_{p < k} D[i, k, p] + \min_{q < k} D[k, j, q].
\]

• **Base case**: \( D[i, j, 0] \) is \( w(i, j) \) (which may be \( \infty \) if there is no edge from \( i \) to \( j \))

• **Base case**: \( D[i, i, 0] \) is by defn 0.

• **Goal**: \( \min_{k=0}^{n} D[s, t, k] \)

• (Note that goal is not one point but rather \( \min \) over \( \Theta(n) \) points.)

Ordering and cost?

• Must compute \( D[i, j, k] \) for all \( i, j, k \leq n \).
• Hence, must compute $\Theta(n^3)$ domain points.
• Base case lookups in $G$ are $O(1)$, provided we use adjacency matrix.
• Each other point takes $\Theta(n)$ time for the two independent mins.
• Conclude that total cost is $\Theta(n^4)$.

4 We Can Do Better

Can we get this running time down?

• Note that for each $k$, we are computing
  \[
  \min_{p<k} D[i, k, p] + \min_{q<k} D[k, j, q]
  \]

• This min operation is extremely redundant!
• When moving from $k$ to $k + 1$, each min includes all the terms we’ve already looked at, plus a few more (for $p$ or $q = k + 1$).
• Is there a way to avoid so much repetition?

Idea: compute min progressively!

• Let $C[i, j, k] = \min_{p \leq k} D[i, j, p]$
• Then we may write
  \[
  D[i, j, k] = C[i, k, k - 1] + C[k, j, k - 1].
  \]

• Also, we have that
  \[
  C[i, j, k] = \min\{C[i, j, k - 1], D[i, j, k]\}
  \]

• We can rewrite the last expression fully in terms of $C$ by subbing in definition of $D$:
  \[
  C[i, j, k] = \min \left\{ \frac{C[i, j, k - 1]}{C[i, k, k - 1] + C[k, j, k - 1]} \right\}
  \]

• Base case: $C[i, j, 0] = D[i, j, 0] = w(i, j)$, or 0 if $i = j$.
• Goal point: $C[s, t, n]$
• Analysis: still $\Theta(n^3)$ cells as before, but now each cell takes only $O(1)$ time to compute.
• Total cost is $\Theta(n^3)$!

Moral: keep an eye out for opportunities to implement progressive min and max. You could also reformulate choice initially to build in the min: shortest path has largest intermediate vertex at most $k$, rather than exactly $k$. 
5 Why Do This?

What’s so great about cubic time?

- Observe that one run of DP algorithm actually computes $C[s, t, n]$ for every pair of vertices $s, t$, not just the pair specified in the input.

- Hence, in $\Theta(n^3)$ time, computes all-pairs shortest paths.

- Compare to $\Theta(n^2 \log n)$ to use Dijkstra for single-source shortest paths on a dense graph.