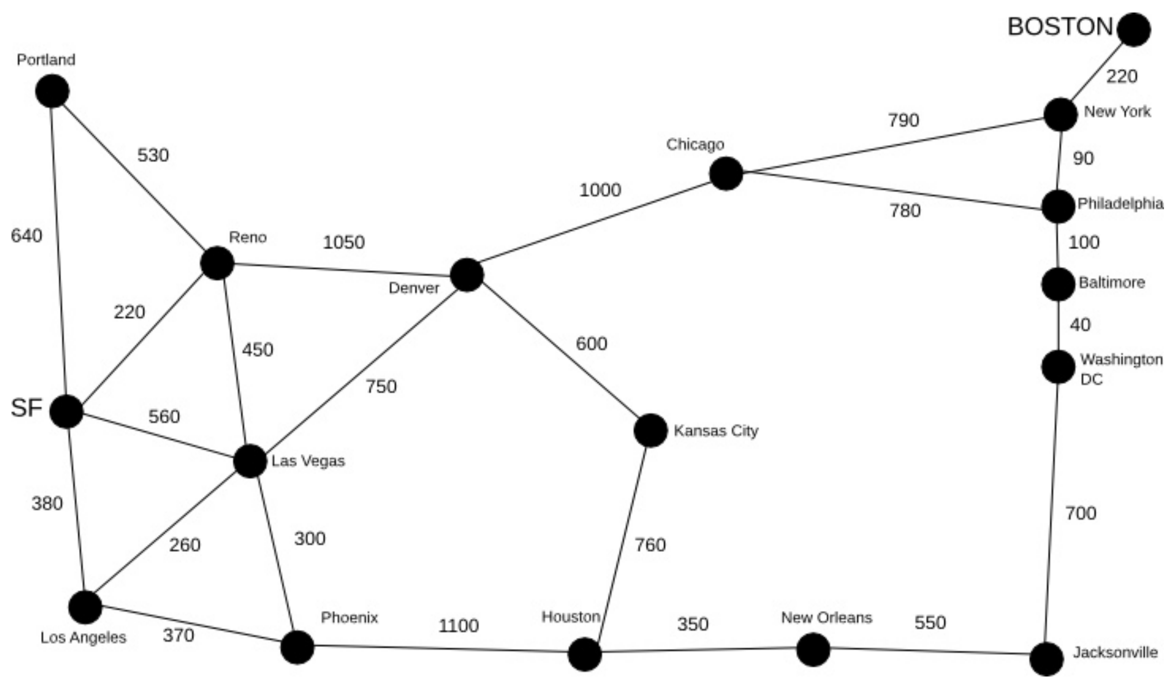
10.4 Shortest Path Algorithms

Weighted Graphs

A weighted graph is a graph $G(V,E)$ together with a function
$w : E \to [0, \infty)$

For $e \in E$, the number $w(e)$ is the weight of e.



Weighted Graphs

Graph	Vertices	Edges	Weights
communication	computers	fiberoptic cables	response time
air travel	airports	flights	flight times
car travel	street corners	streets	distances
Kevin Bacon	actors	common movies	1
stock market	stocks	transactions (directed edges)	cost
operations research	projects	dependencies (directed edges)	times

Distance Problems

Traveling Salesman Problem. Given a list of cities to visit, what is the minimum distance you need to travel?

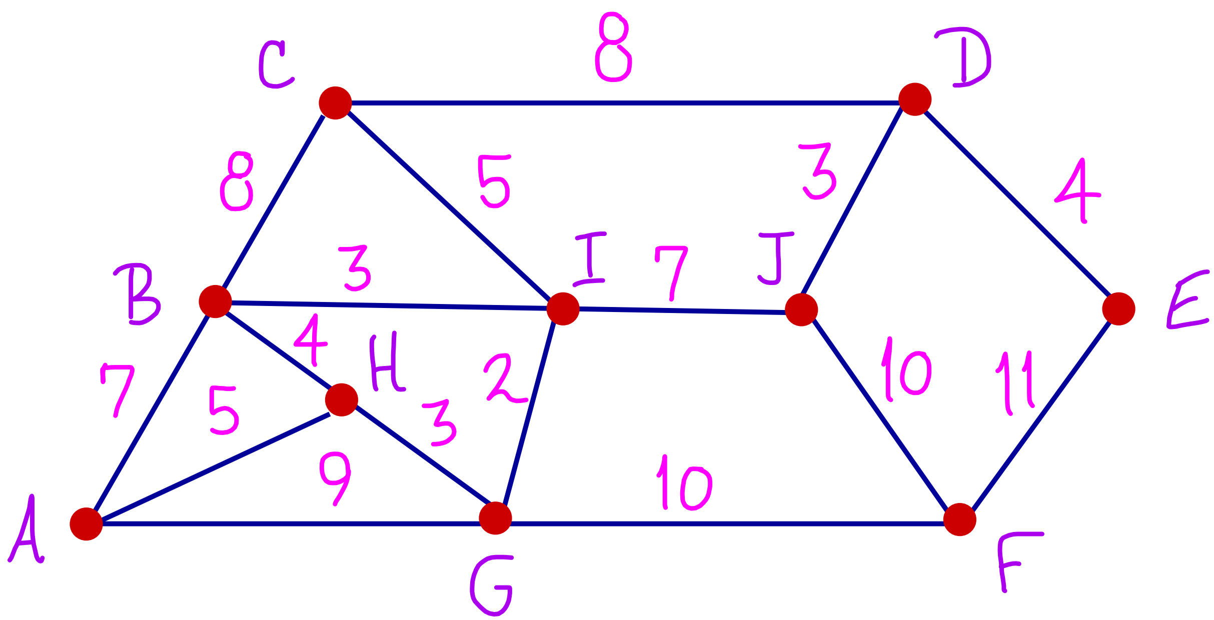
TSP is really a question about weighted graphs.

Easier Problem. Given two vertices in a weighted graph, what is their "distance."

The length of a walk is the sum of the weights of the edges traversed, and the distance between two vertices is the minimum length of a walk between them.

EXAMPLE

PROBLEM. Find the distance between A and E.



How to find the shortest path in general?

Dijkstra's Algorithm

To find the distances from a given vertex A in a weighted graph to all other vertices, do the following.

First, give A the permanent label 0, and give all other vertices the temporary label ∞.

Then repeat the following step:


Edsger Dijkstra

Find the vertex v with the newest permanent label.

For each vertex v' adjacent to v with a temporary label, check if

label of v + $w(vv')$ $\leq$ label of v'

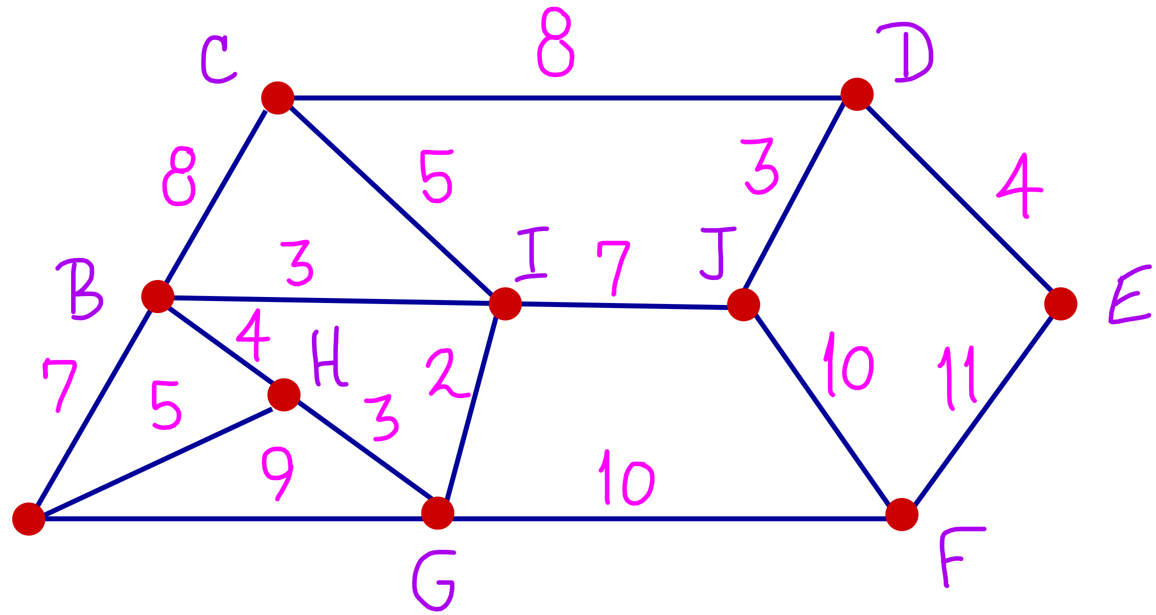
If so, change the temporary label of v'.

Make the smallest temporary label permanent.

Permanent labels are the distances from A.

DIJKSTRA'S ALGORITHM

Find the distance from A to each other vertex.



Dijkstra's Algorithm

Why does Dijkstra's algorithm work?

Use induction on the number of edges needed to walk from A to the other vertices.

Base case: 0 — The only such vertex is A, whose permanent label and distance from A are both 0.

Inductive step. Suppose we need to cross at least one edge to get from A to v. If there is a path of length d from A to v, there is a path of length $d' < d$ from A to some vertex v' that is adjacent to v and is one "step" closer to A. By induction, the permanent label of v' is its distance from A. It then follows that v will get the correct permanent label. (Why?)

Dijkstra's Algorithm

What is the complexity of Dijkstra's algorithm, if size is measured in the number of vertices and cost is measured in terms of number of operations (=additions and comparisons)?

At k^{th} step, there are $n-k$ vertices without a permanent label.

$\rightsquigarrow$ at most $n-k$ additions, $n-k$ comparisons.

Then need $n-k-1$ comparisons to find the smallest temporary label.

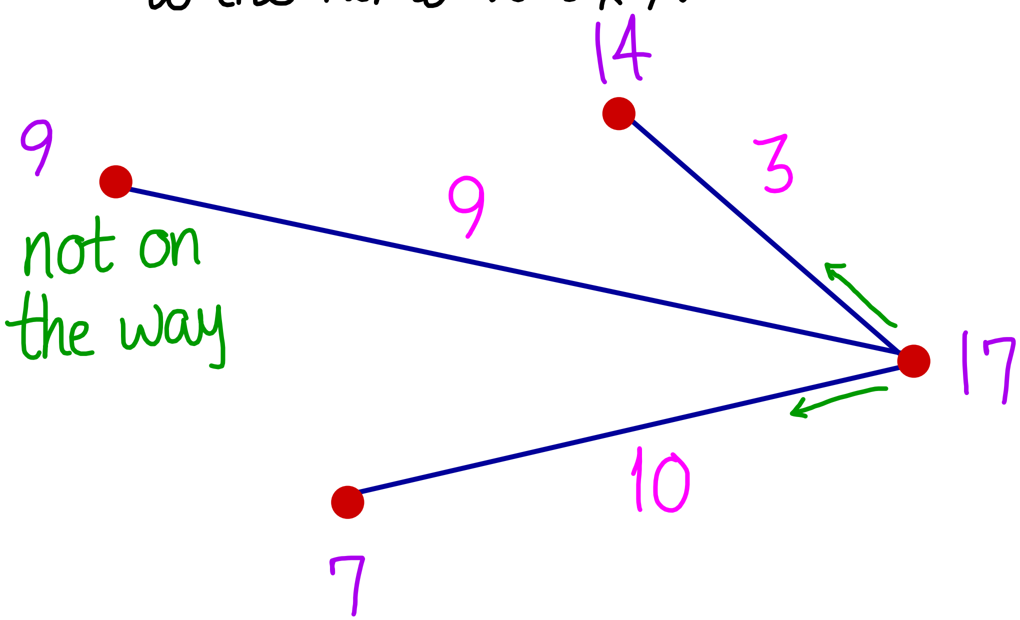
$$f(n) = \sum_{k=1}^{n-1} \Big(2(n-k) + (n-k+1) \Big)$$

$$= \frac{3}{2}n^2 - \frac{5}{2}n + 1 = O(n^2)$$

Dijkstra's Algorithm

What if we further want to find a walk between two vertices with the shortest length (not just the distance between the two vertices)?

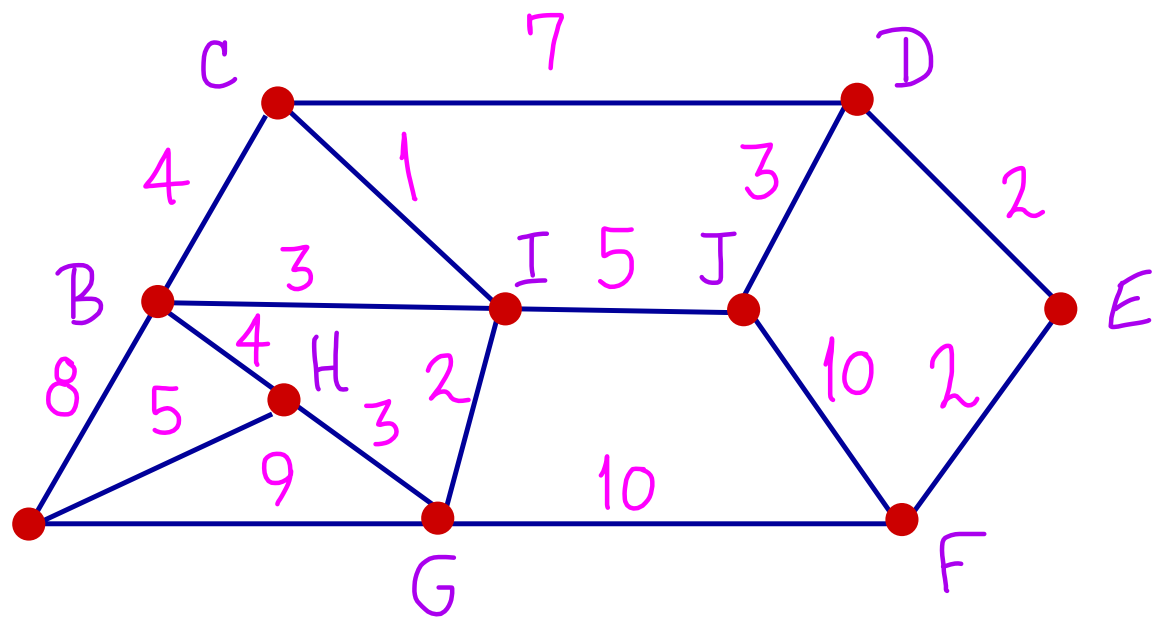
Idea: Every time we make a label permanent, draw a little arrow from that vertex to all other vertices that are "en route" to the home vertex A



Then, follow the arrows to find all shortest walks home.

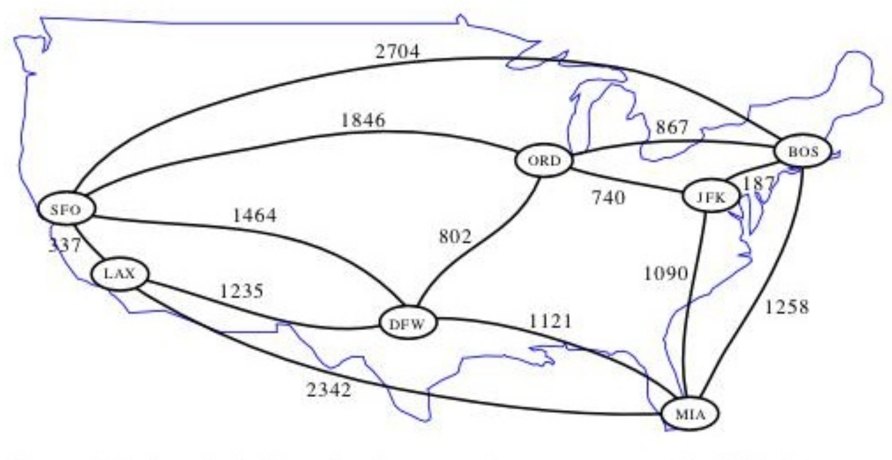
Dijkstra's Algorithm

Find all shortest paths from A to E.

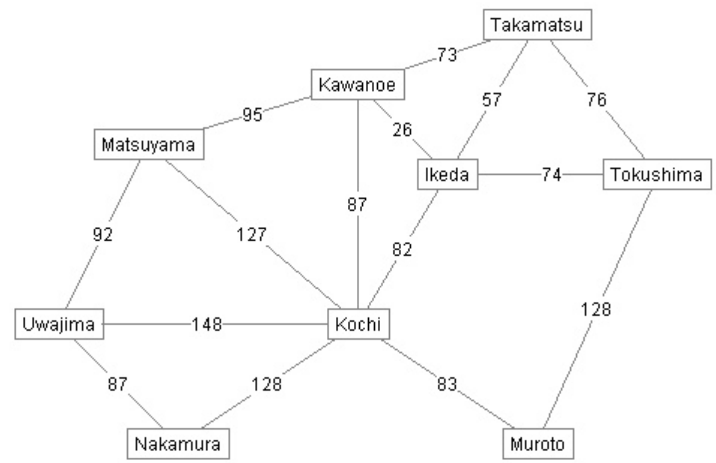


Dijkstra's Algorithm

Find the shortest paths...



from LAX to JFK



from Nakamura to Tokushima

Floyd-Warshall Algorithm

Idea: Number the vertices $v_1, \dots, v_n$.

Step k: Find the shortest path from v_i to v_j if you are only allowed to use $v_1, \dots, v_k$ as intermediate vertices (= pit stops).

Can write this info in a matrix M_k.

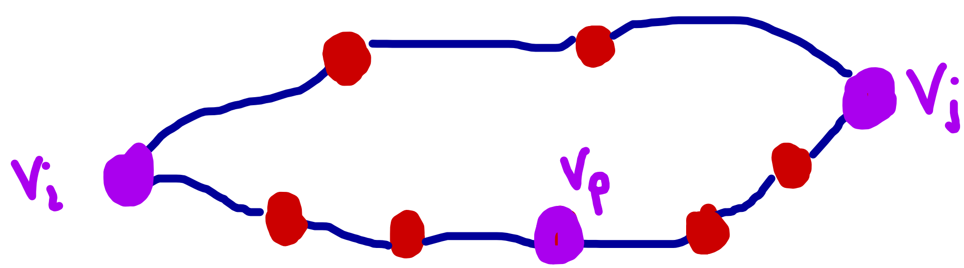
Write ∞ if there is no path.

Do this for $k = 0, \dots, n$. (At Step 0, no pit stops allowed.)

The ij-entry of M_n is the distance from v_i to v_j.

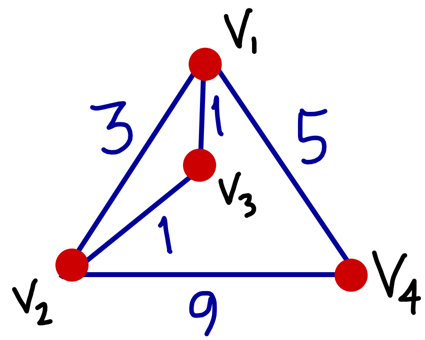
Key observation. For $k \geq 1$:

$$M_k(i,j) = \min_p \{ M_{k-1}(i,j),\ M_{k-1}(i,p) + M_{k-1}(p,j) \}$$



Floyd-Warshall Algorithm

Example.



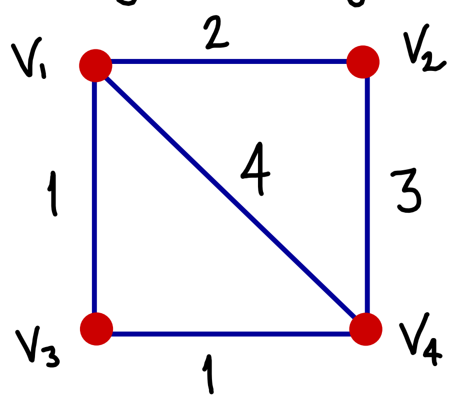
$$M_0 = \begin{pmatrix} 0 & 3 & 1 & 5 \\ & 0 & 1 & 9 \\ & & 0 & \infty \\ & & & 0 \end{pmatrix} \qquad M_1 = \begin{pmatrix} 0 & 3 & 1 & 5 \\ & 0 & 1 & 8 \\ & & 0 & 6 \\ & & & 0 \end{pmatrix} \qquad M_2 = \begin{pmatrix} 0 & 3 & 1 & 5 \\ & 0 & 1 & 8 \\ & & 0 & 6 \\ & & & 0 \end{pmatrix}$$

$$M_3 = \begin{pmatrix} 0 & 2 & 1 & 5 \\ & 0 & 1 & 7 \\ & & 0 & 6 \\ & & & 0 \end{pmatrix} \qquad M_4 = \begin{pmatrix} 0 & 2 & 1 & 5 \\ & 0 & 1 & 7 \\ & & 0 & 6 \\ & & & 0 \end{pmatrix}$$

Note: M_k has same row/col k as M_{k-1}.

Floyd-Warshall Algorithm

Find all distances using the Floyd-Warshall algorithm.



DIJKSTRA VS FLOYD-WARSHALL

To find distances for all pairs of vertices, we need to run Dijkstra's algorithm n times $\rightsquigarrow \mathcal{O}(n^3)$.

Floyd-Warshall is also $\mathcal{O}(n^3)$, but is quicker for large graphs.

One advantage to Floyd-Warshall is that it even works with negative edge weights.