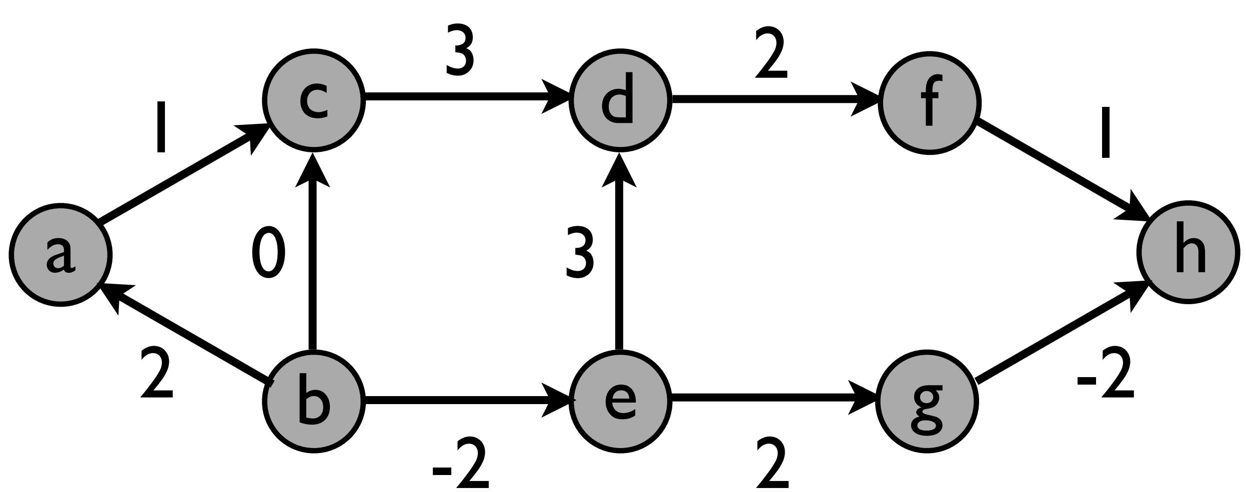
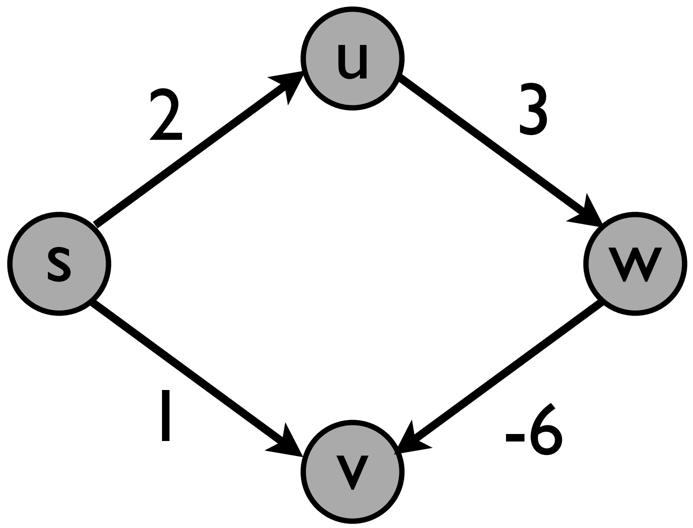
Part taken from slides by Kamalika Chaudhuri

All Pairs Shortest Paths

Problem: Given n nodes and distances d_{ij} (which could be negative, or 0, or positive) on all edges, find shortest path distances between all pairs of nodes.

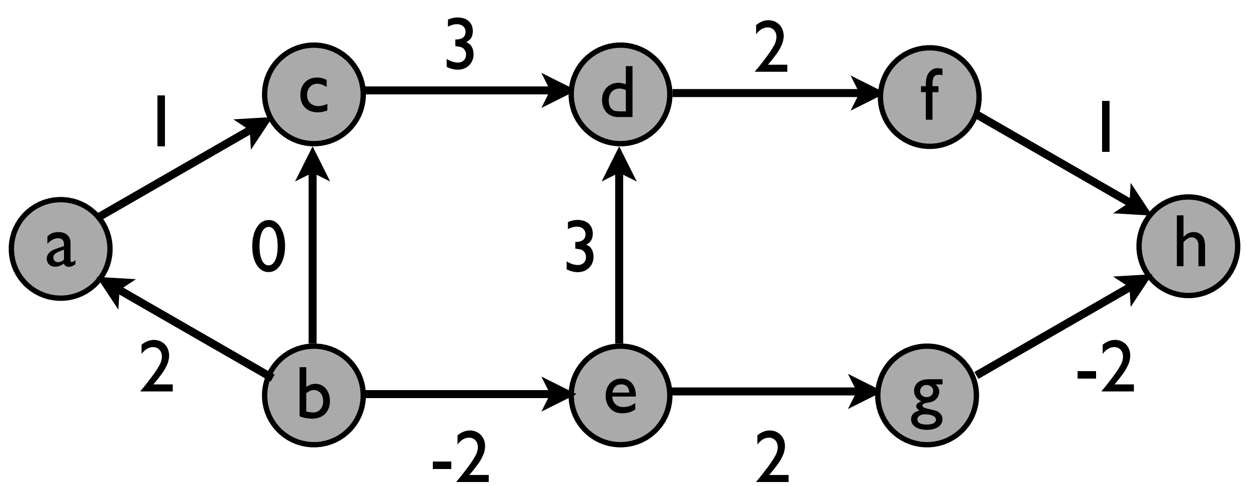


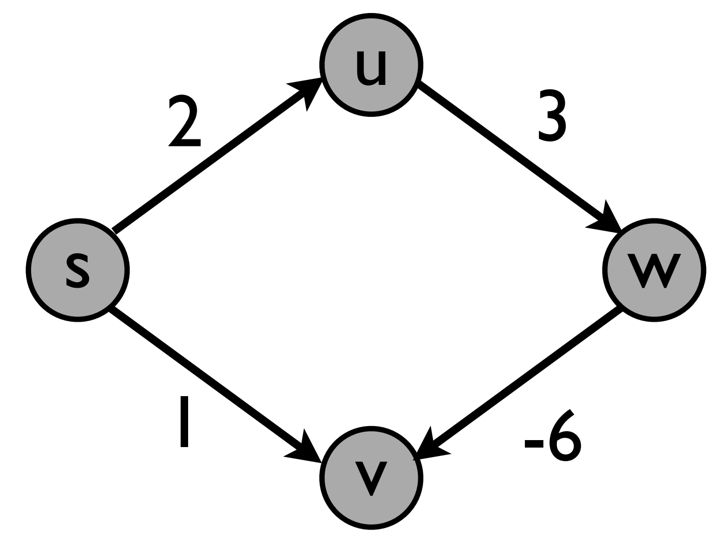


Does Dijkstra's algorithm work?

All Pairs Shortest Paths

Problem: Given n nodes and distances d_{ij} (which could be negative, or 0, or positive) on all edges, find shortest path distances between all pairs of nodes.





Does Dijkstra's algorithm work?

Ans: No! Example: s-v Shortest Paths

All Pairs Shortest Paths (APSP)

Problem: Given n nodes and distances d_{ij} (which could be negative, or 0, or positive) on all edges, find shortest path distances between all pairs of nodes.

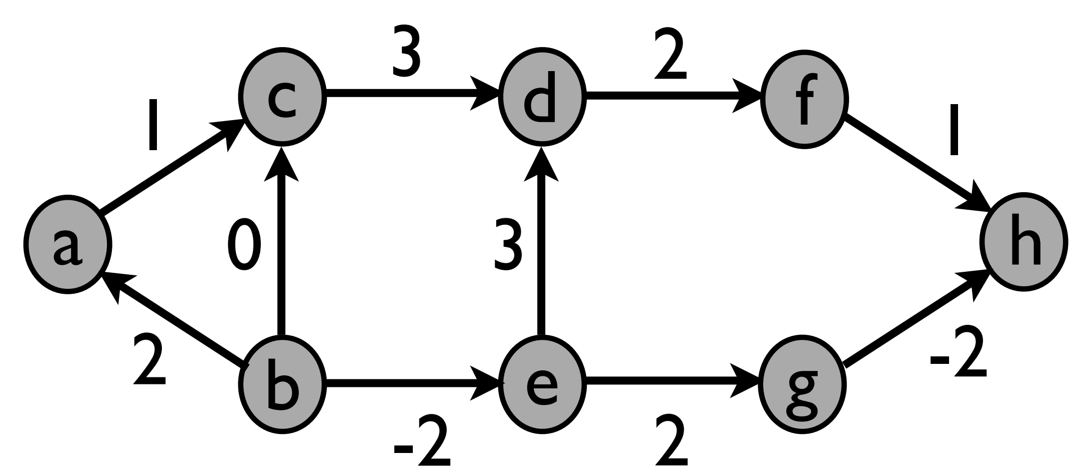
Structure:

For all x, y:

either $SP(x, y) = d_{xy}$

Or there exists some z s.t

$SP(x, y) = SP(x, z) + SP(y, z)$



All Pairs Shortest Paths (APSP)

Problem: Given n nodes and distances d_{ij} (which could be negative, or 0, or positive) on all edges, find shortest path distances between all pairs of nodes.

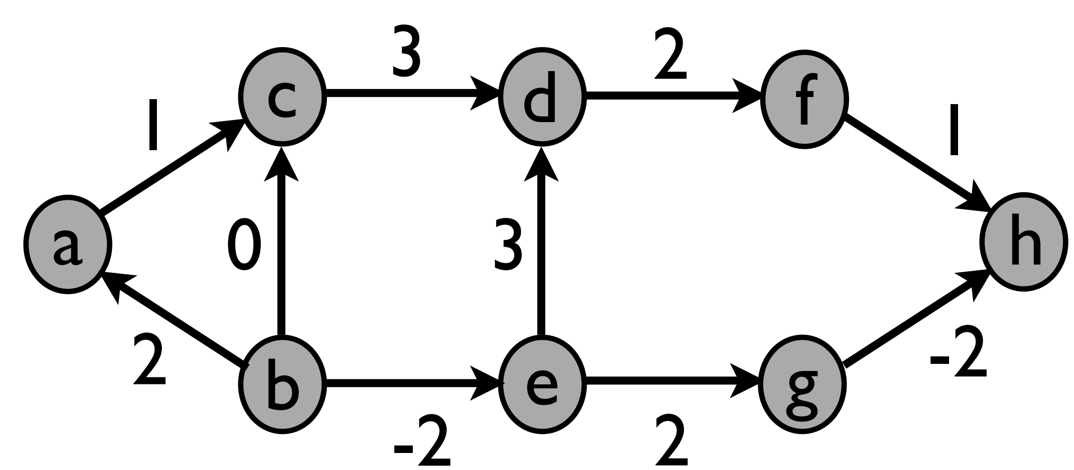
Structure:

For all x, y:

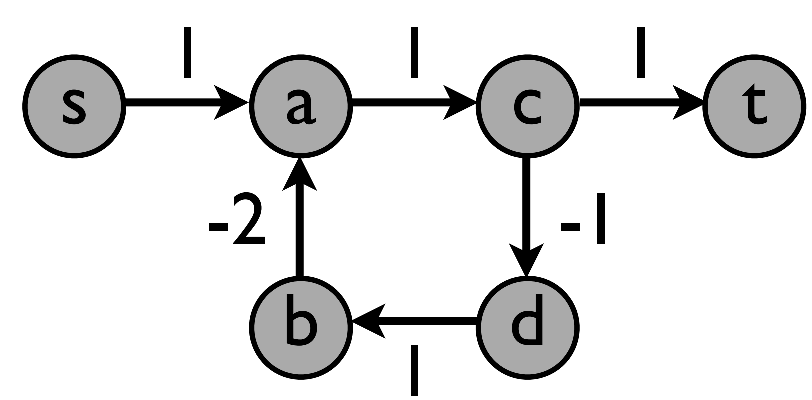
either $SP(x, y) = d_{xy}$

Or there exists some z s.t

$SP(x, y) = SP(x, z) + SP(y, z)$



Property: If there is no negative weight cycle, then for all x, y, SP(x, y) is simple (that is, includes no cycles)

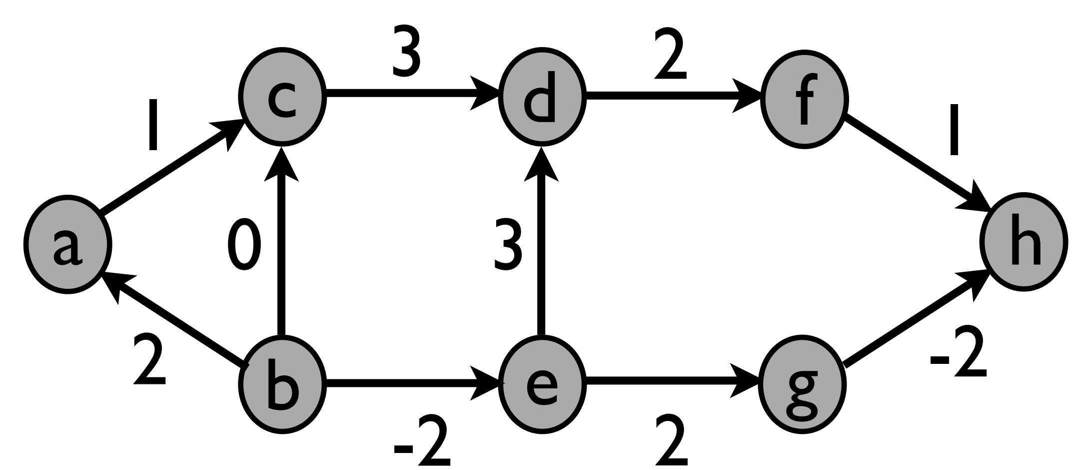


All Pairs Shortest Paths

Problem: Given n nodes and distances d_{ij} (which could be negative, or 0, or positive) on all edges, find shortest path distances between all pairs of nodes.

STEP 1: Define Subtasks

D(i,j,k) = length of shortest path from i to j with intermediate nodes in {1,2,...k}



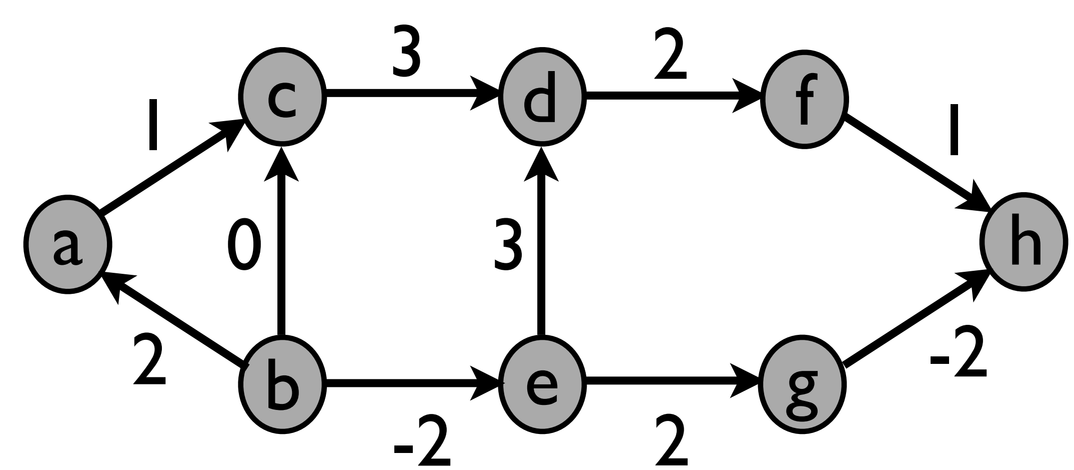
All Pairs Shortest Paths

Problem: Given n nodes and distances d_{ij} (which could be negative, or 0, or positive) on all edges, find shortest path distances between all pairs of nodes.

STEP 1: Define Subtasks

D(i,j,k) = length of shortest path from i to j with intermediate nodes in {1,2,...k}

Shortest Path lengths = D(i,j,n)



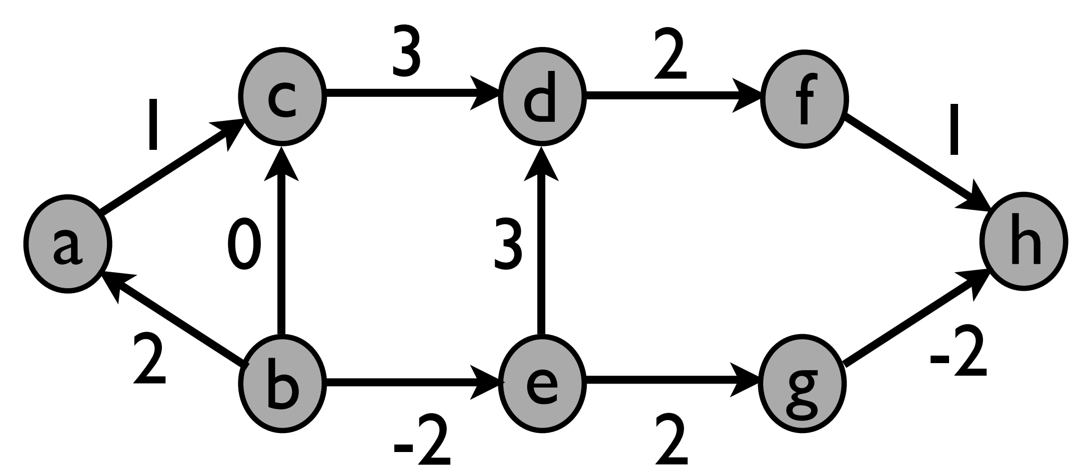
All Pairs Shortest Paths

Problem: Given n nodes and distances d_{ij} (which could be negative, or 0, or positive) on all edges, find shortest path distances between all pairs of nodes.

STEP 1: Define Subtasks

D(i,j,k) = length of shortest path from i to j with intermediate nodes in {1,2,...k}

Shortest Path lengths = D(i,j,n)



STEP 2: Express Recursively

D(i,j,k) = min{D(i,j,k-1), D(i,k,k-1) + D(k,j,k-1)}

Base case: D(i,j,0) = d_{ij}

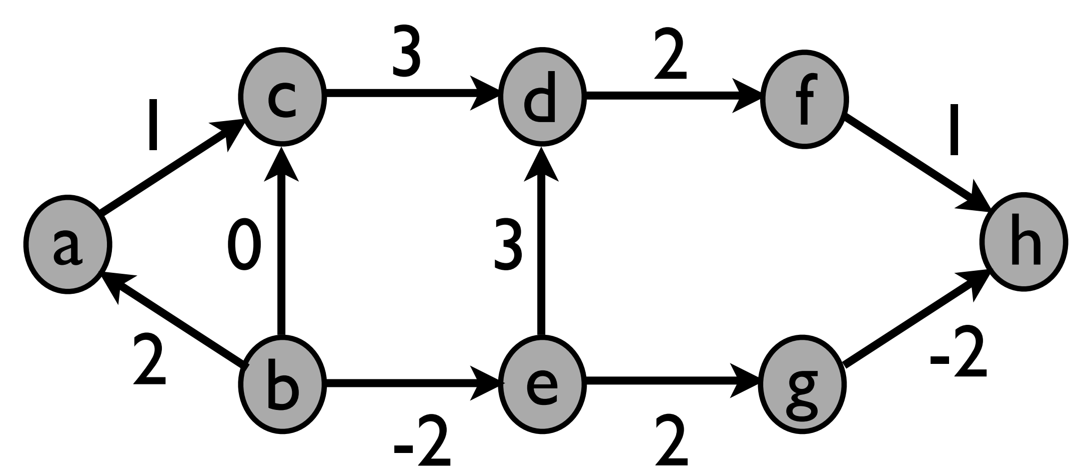
All Pairs Shortest Paths

Problem: Given n nodes and distances d_{ij} (which could be negative, or 0, or positive) on all edges, find shortest path distances between all pairs of nodes.

STEP 1: Define Subtasks

D(i,j,k) = length of shortest path from i to j with intermediate nodes in {1,2,...k}

Shortest Path lengths = D(i,j,n)



STEP 2: Express Recursively

$D(i,j,k) = \min\{D(i,j,k-1), D(i,k,k-1) + D(k,j,k-1)\}$

Base case: $D(i,j,0) = d_{ij}$

STEP 3: Order of Subtasks

By increasing order of k

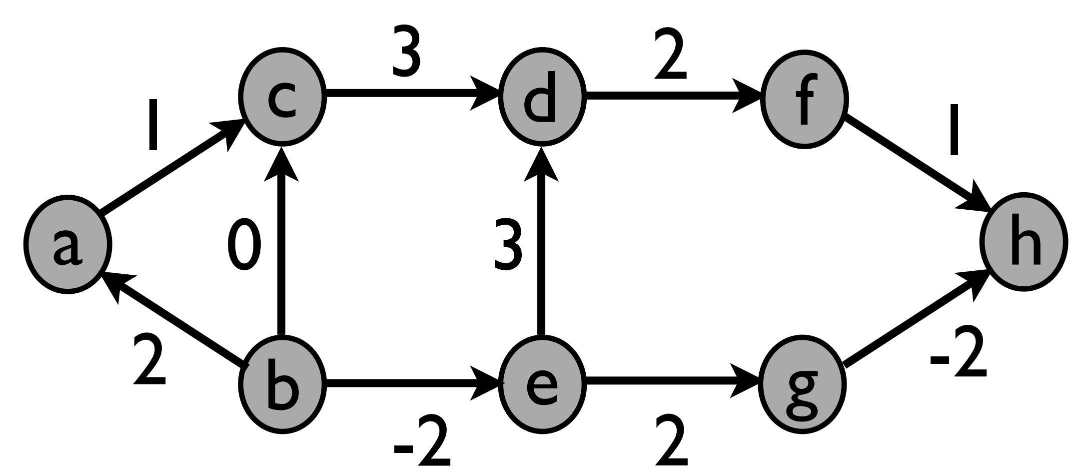
All Pairs Shortest Paths

Problem: Given n nodes and distances d_{ij} (which could be negative, or 0, or positive) on all edges, find shortest path distances between all pairs of nodes.

STEP 1: Define Subtasks

D(i,j,k) = length of shortest path from i to j with intermediate nodes in {1,2,...k}

Shortest Path lengths = D(i,j,n)



STEP 2: Express Recursively

D(i,j,k) = min{D(i,j,k-1), D(i,k,k-1) + D(k,j,k-1)}

Base case: D(i,j,0) = d_{ij}

STEP 3: Order of Subtasks

By increasing order of k

Running Time = $O(n^3)$

Exercise:

Reconstruct the shortest paths

Summary: Dynamic Programming

Main Steps:

1. Divide the problem into **subtasks**

2. Define the subtasks **recursively** (express larger subtasks in terms of smaller ones)

3. Find the **right order** for solving the subtasks (but do not solve them recursively!)

Summary: Dynamic Programming vs Divide and Conquer

Divide-and-conquer

A problem of size n is decomposed into a few subproblems which are significantly smaller (e.g. n/2, 3n/4,...)

Therefore, size of subproblems decreases geometrically.

eg. n, n/2, n/4, n/8, etc

Use a recursive algorithm.

Dynamic programming

A problem of size n is expressed in terms of subproblems that are not much smaller (e.g. n-1, n-2,...)

A recursive algorithm would take exp. time.

Saving grace: in total, there are only polynomially many subproblems.

Avoid recursion and instead solve the subproblems one-by-one, saving the answers in a table, in a clever explicit order.