Shortest Path – DAG + All Pairs Shortest Path

For use in CSE6010 only

Not for distribution

Shortest Paths

- Last class we looked at two algorithms to find shortest paths in a graph. Their complexities are (for each vertex):
  - Bellman-Ford: O(E * V)
  - Dijkstra: $O(V^2)$ (for adjacency matrix; we did not derive this; can do better with adjacency lists), but we cannot use for graphs with negative weights
- In some cases we can do better.
- In particular, let's consider dags (directed acyclic graphs).
- A couple weeks ago we saw that we could calculate a topological sort of a dag in O(E+V) time (DFS recursive + stack for completed vertices).
- Using the topological sort ensures each edge needs to be considered only once.

Shortest Paths in Dags

Algorithm

Topologically sort vertices (DFS, stack for completed vertices)

Set distance to each vertex to ∞, pred of each vertex to NULL (for path)

Set distance to selected source vertex to 0

For each vertex u taken in topological order

    For each vertex v in Adj[u]

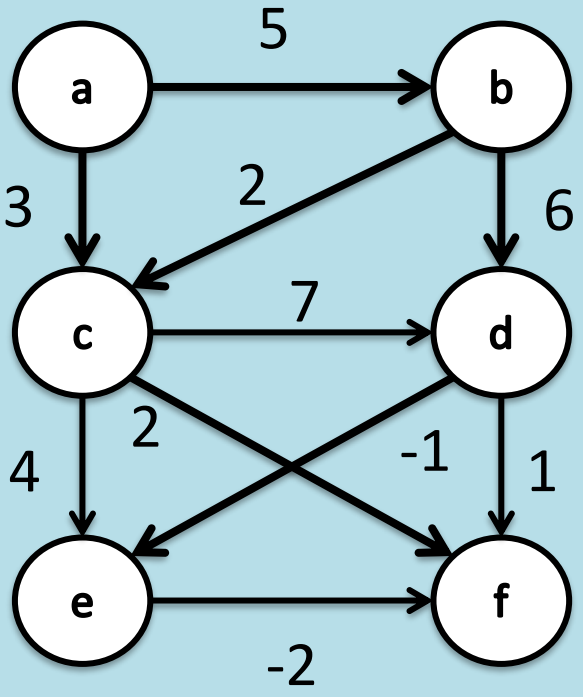
        Relax(u, v)

Example: Shortest Paths in Dags

Run through the Dag-shortest-path algorithm on the following dag using b as the source. Remember to find a topological sort first. You may find it helpful to redraw the graph as a horizontal line of vertices with arrows pointing only right.

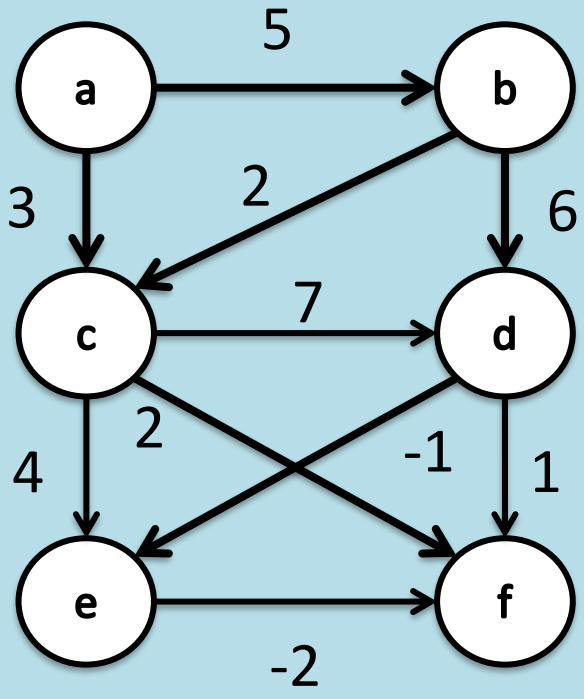
Topologically sort vertices
Set distance to each vertex to ∞
Set pred of each vertex to NULL (for path)
Set distance to selected source vertex to 0

For each vertex u taken in topological order
    For each vertex v in Adj[u]
        Relax(u, v)

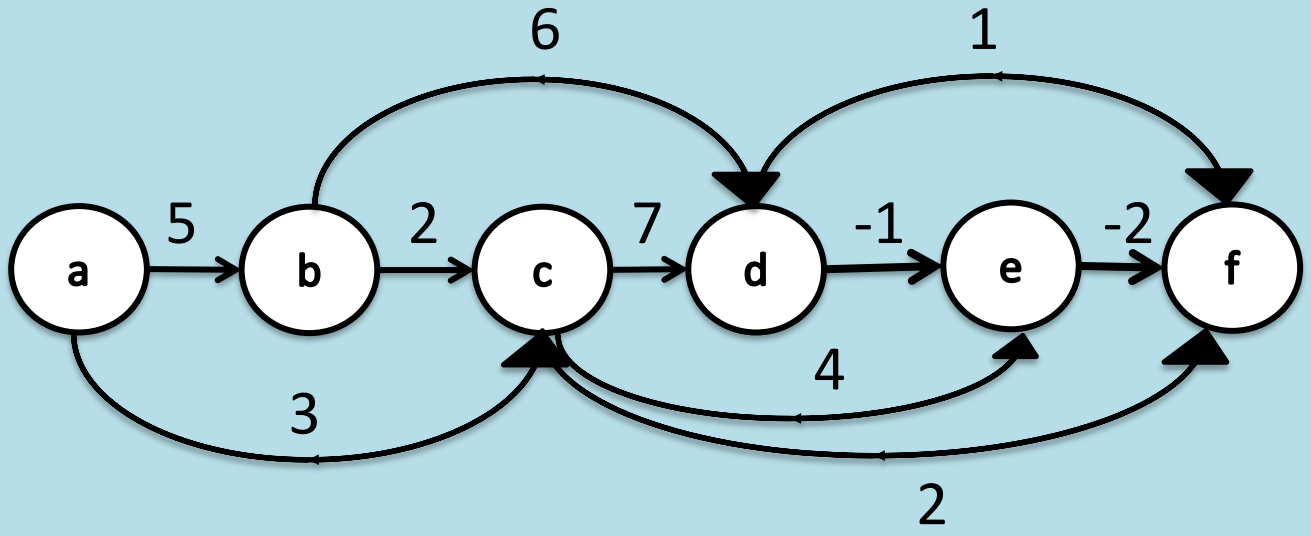


Example: Shortest Paths in Dags

Run through the Dag-shortest-path algorithm on the following dag using b as the source. Remember to find a topological sort first. You may find it helpful to redraw the graph as a horizontal line of vertices with arrows pointing only right.



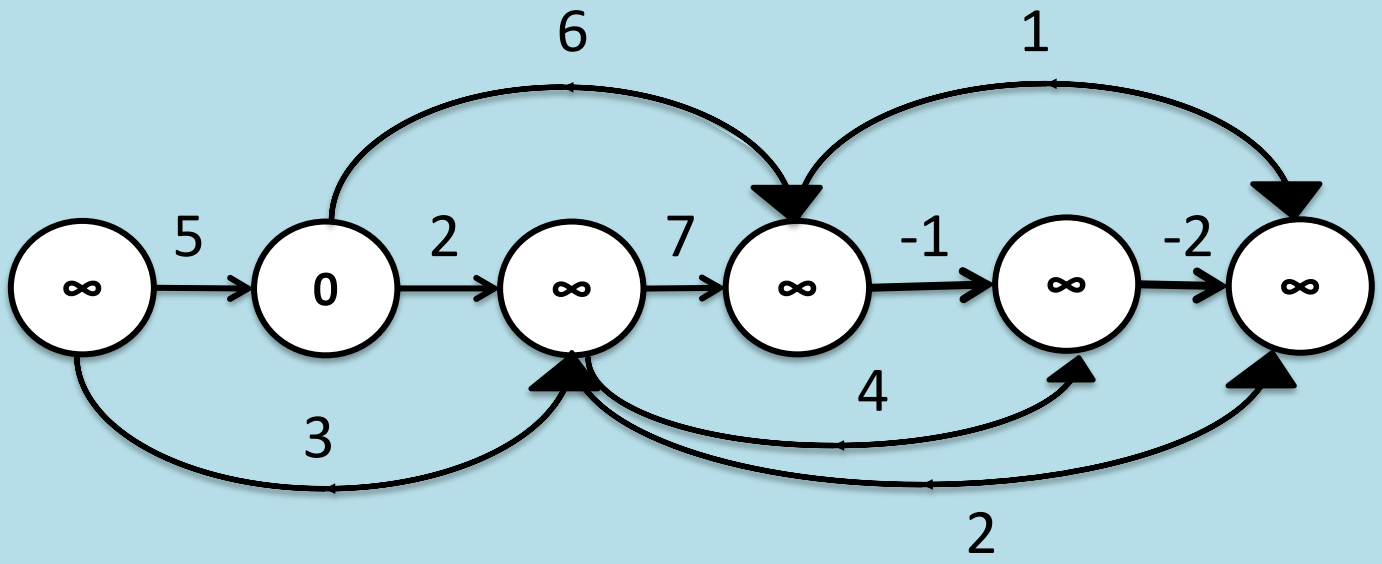
For the topological sort: it turns out that alphabetical order is a topological sort so we will use it.



Example: Shortest Paths in Dags

Set distance to each vertex to ∞
Set pred of each vertex to NULL (for path)
Set distance to selected source vertex to 0

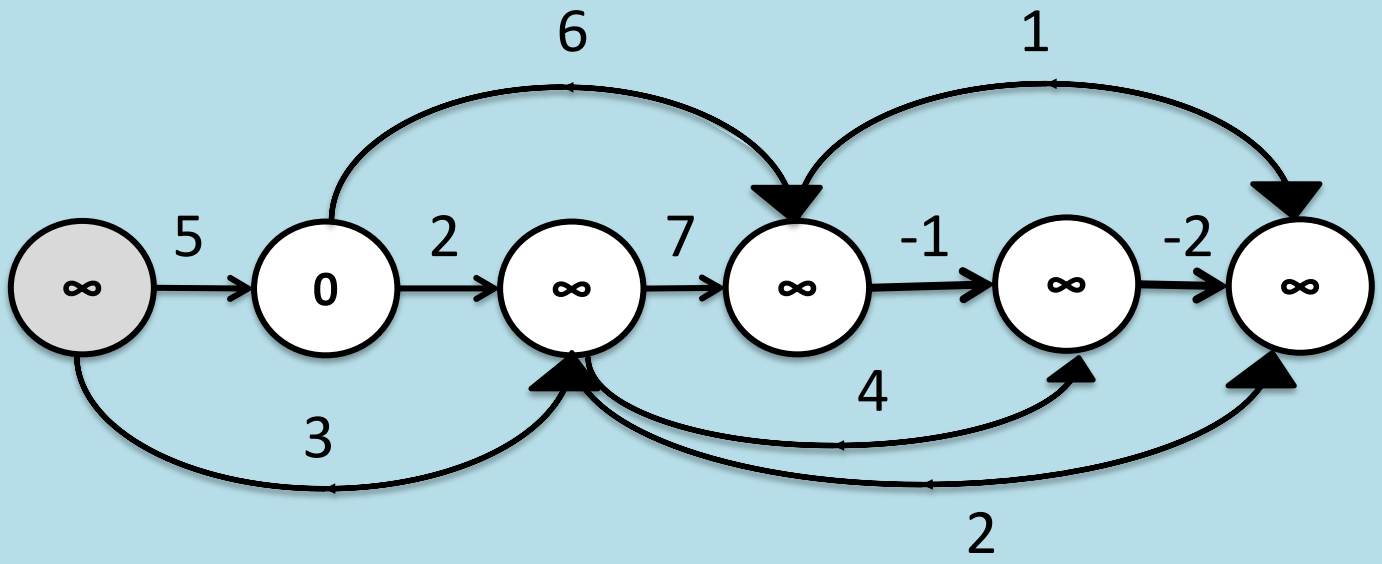
For each vertex u taken in topological order
    For each vertex v in Adj[u]
        Relax(u, v)



Example: Shortest Paths in Dags

Set distance to each vertex to ∞
Set pred of each vertex to NULL (for path)
Set distance to selected source vertex to 0

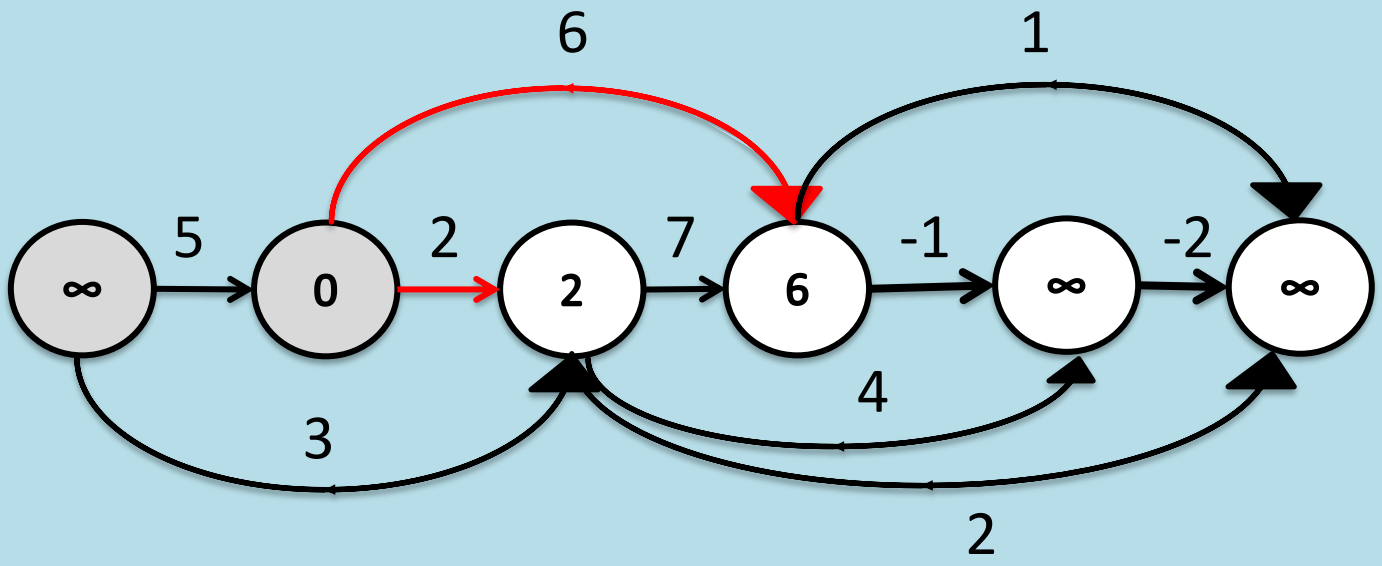
For each vertex u taken in topological order
    For each vertex v in Adj[u]
        Relax(u, v)



Example: Shortest Paths in Dags

Set distance to each vertex to ∞
Set pred of each vertex to NULL (for path)
Set distance to selected source vertex to 0

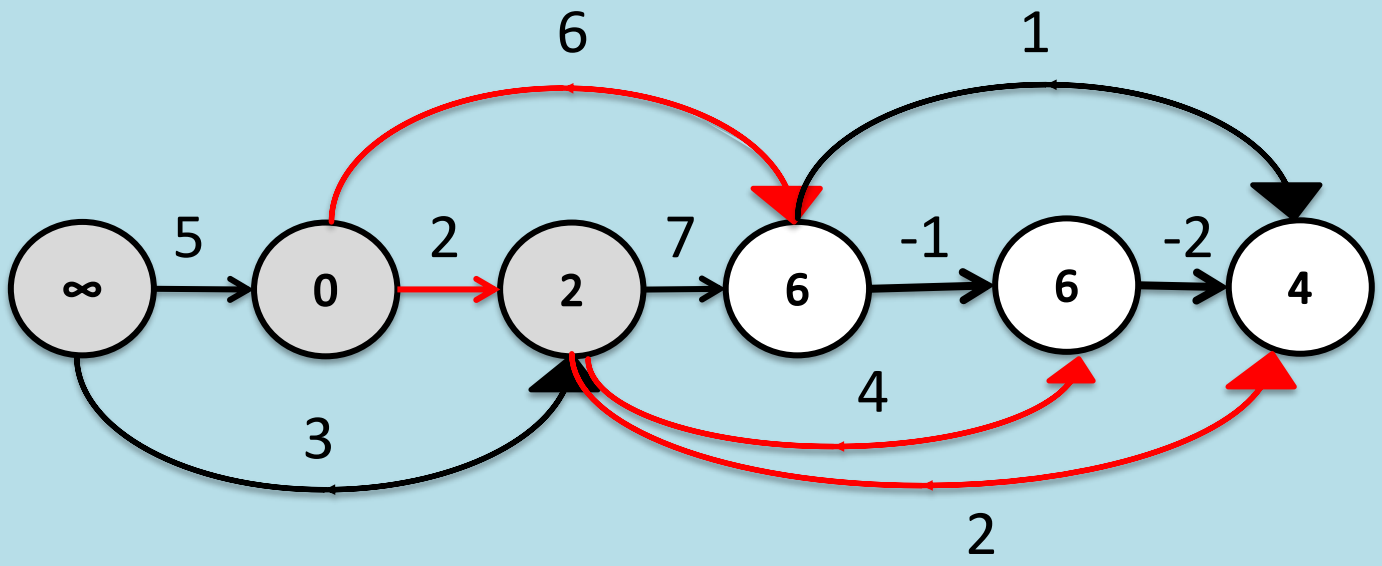
For each vertex u taken in topological order
    For each vertex v in Adj[u]
        Relax(u, v)



Example: Shortest Paths in Dags

Set distance to each vertex to ∞
Set pred of each vertex to NULL (for path)
Set distance to selected source vertex to 0

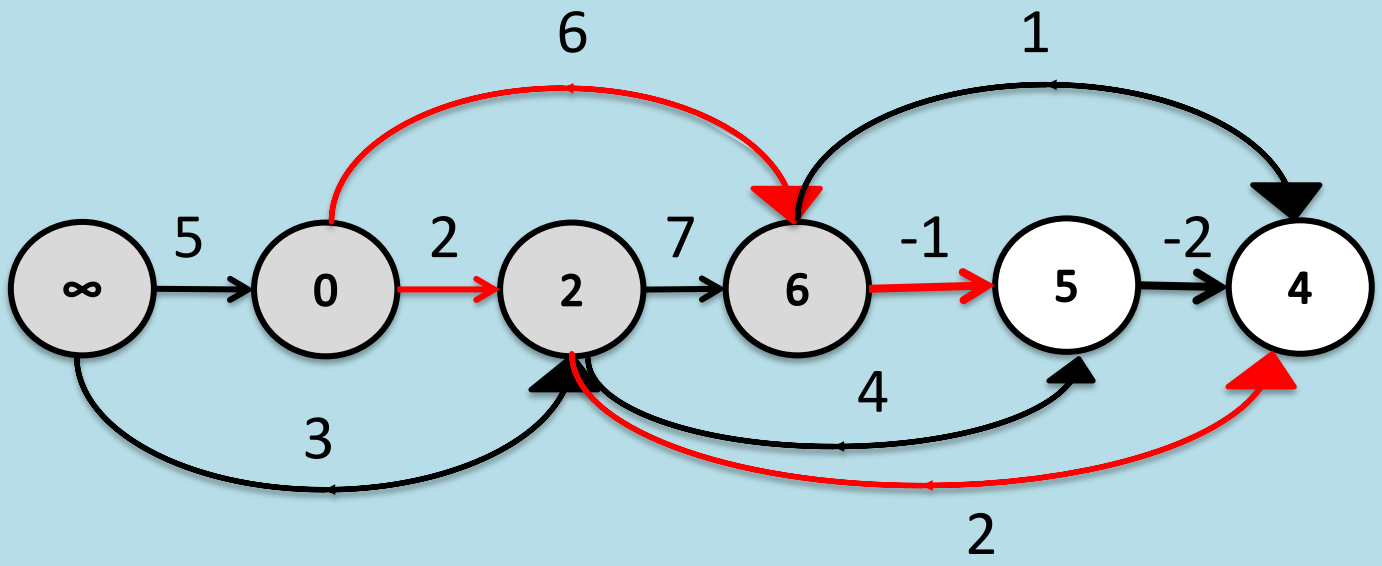
For each vertex u taken in topological order
    For each vertex v in Adj[u]
        Relax(u, v)



Example: Shortest Paths in Dags

Set distance to each vertex to ∞
Set pred of each vertex to NULL (for path)
Set distance to selected source vertex to 0

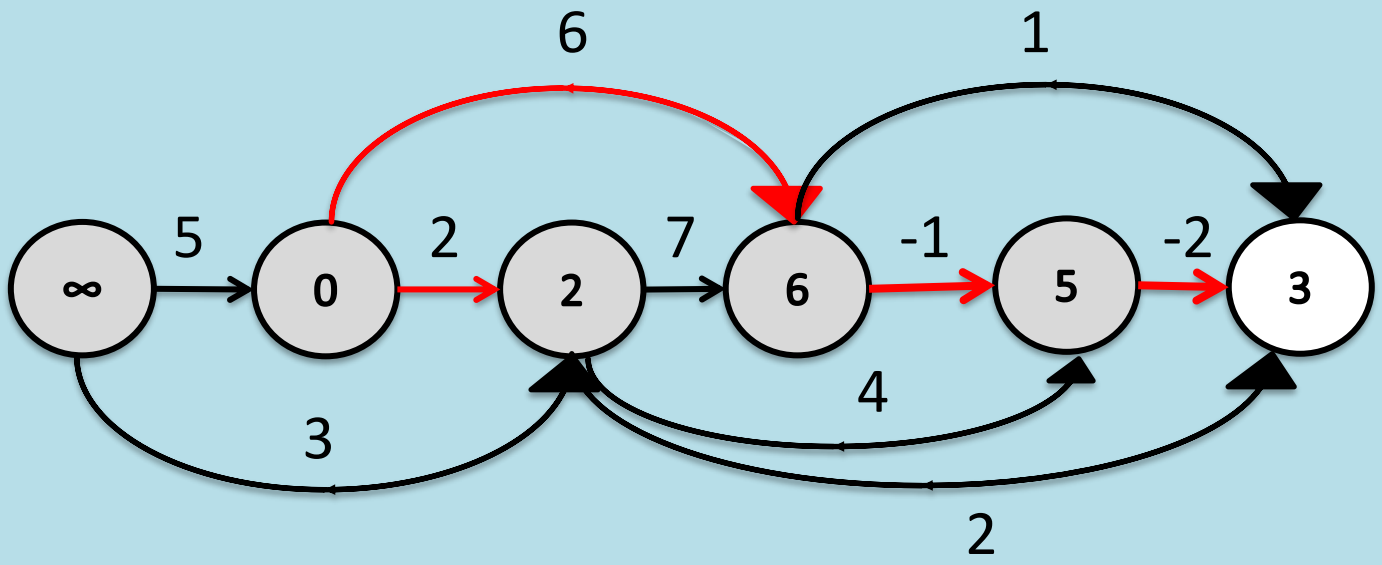
For each vertex u taken in topological order
    For each vertex v in Adj[u]
        Relax(u, v)



Example: Shortest Paths in Dags

Set distance to each vertex to ∞
Set pred of each vertex to NULL (for path)
Set distance to selected source vertex to 0

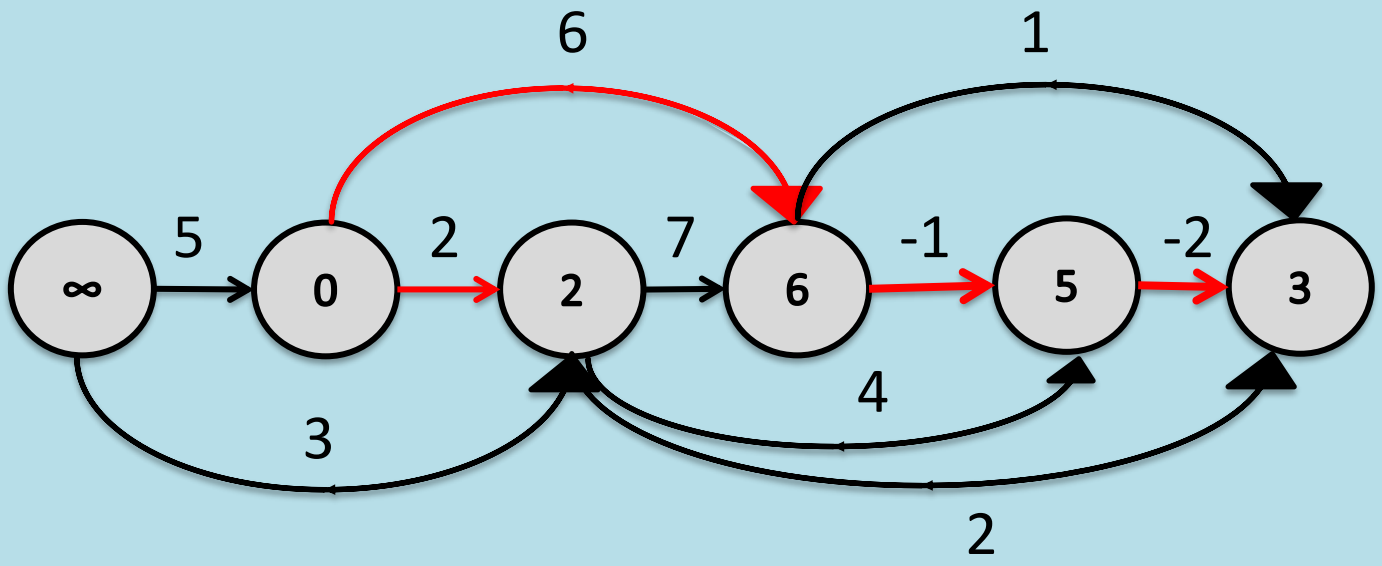
For each vertex u taken in topological order
    For each vertex v in Adj[u]
        Relax(u, v)



Example: Shortest Paths in Dags

Set distance to each vertex to ∞
Set pred of each vertex to NULL (for path)
Set distance to selected source vertex to 0

For each vertex u taken in topological order
    For each vertex v in Adj[u]
        Relax(u, v)



Complexity of Dag Shortest Paths Algorithm

- What is the computational complexity?

Topologically sort vertices O(V+E)

…

For each vertex u taken in topological order

For each vertex v in Adj[u]

Relax(u, v)

This will examine each edge once

- Overall complexity is O(V+E).
- Savings over Bellman-Ford [O(V*E)] and Dijkstra [$O(V^2)$].

All Pairs Problem Statement

- Compute the minimum length path between all pairs of nodes in a graph with N nodes and adjacency matrix A
  - Nodes are labeled 0, 1, 2, 3, … N-1
  - A[i,j] indicates the weight of the edge from i to j
  - Assume A[i,j] ≥ 0 for all i, j
- Specifically: compute D where D[i,j] = minimum weight of a path from i to j
  - D[i,j] = ∞ if no path exists from i to j

Definition: the maximum length among the minimum length paths between all pairs of nodes in the graph is known as the graph's ***diameter*** (analogous to the diameter of a circle)

Diameter = max (D[i,j]), i,j = 0, … N-1

Diameter gives the worst-case distance between any pair of nodes

All Pairs Shortest Path

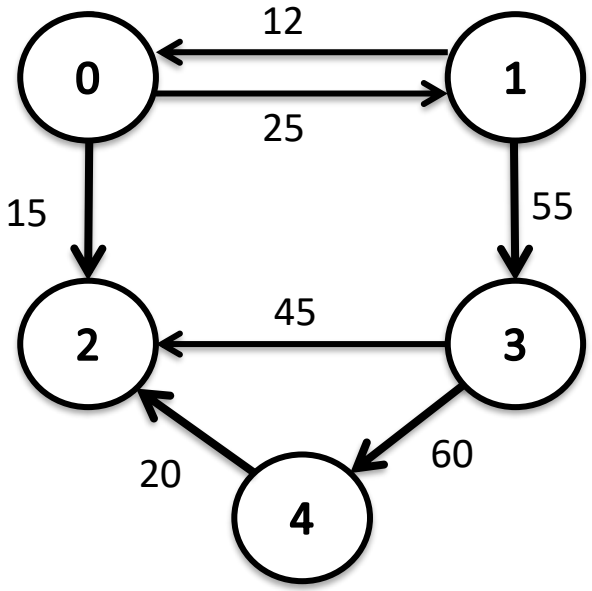
Can you guess one way to calculate all pairs distance?

All pairs distance could be computed by repeatedly applying Bellman-Ford or Dijkstra's algorithm.

```
for (i=0; i<N; i++)
   // invoke Dijkstra w/ source node i
   Dijkstra(i);
```

To develop a different approach, let's begin with some observations.

Adjacency Matrix: Observations

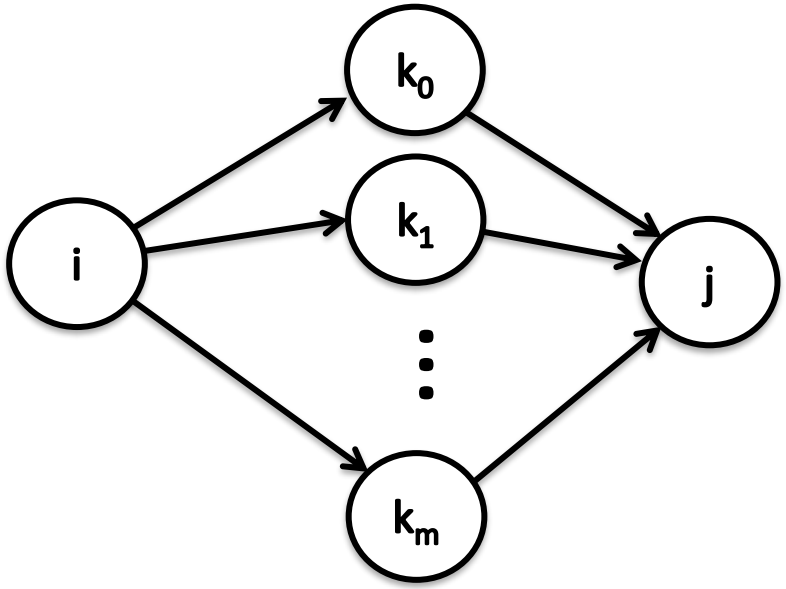


	0	1	2	3	4
0	-	25	15	-	-
1	12	-	-	55	-
2	-	-	-	-	-
3	-	-	45	-	60
4	-	-	20	-	-

- Row i indicates the outgoing edges for node i
- Column j indicates the incoming edges for node j
- Length of path from node i to j via k can be computed as A[i,k] + A[k,j]
- Example: Path from 0 to 3 via 1: A[0,1]+A[1,3] = 25+55 = 80
  This is an example of a two-hop path

Two-Hop Paths

What is the minimum weight path from node i to node j using two hops?



Answer: min (A[i,k]+A[k,j])

k=0 … N-1

k≠i, k≠j

What if there is no link from i to k (or k to j)?

A[p,q] = ∞ if there is no link from p to q

	0	1	2	3	4
0	-	25	15	∞	∞
1	12	-	∞	55	∞
2	∞	∞	-	∞	∞
3	∞	∞	45	-	60
4	∞	∞	20	∞	-

More to come about the diagonal…

Two-Hop (or less) Paths

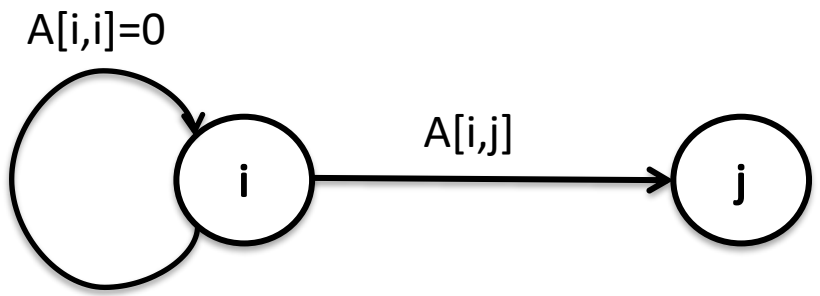
What is the minimum weight of a path from node i to node j using at most two hops?

Direct solution: min (min (A[i,k]+A[k,j]), A[i,j])

2-hop 1-hop

k=0 … N-1
k≠i, k≠j

Better solution: We can view a one-hop path as a two-hop path where each node has a zero-weight link to itself (A[i,i] = 0 for all i)



"Two-hop" path from i to j: i -> i -> j
(or equivalently path i->j->j)
Weight = A[i,i] + A[i,j] = A[i,j]

	0	1	2	3	4
0	0	25	15	∞	∞
1	12	0	∞	55	∞
2	∞	∞	0	∞	∞
3	∞	∞	45	0	60
4	∞	∞	20	∞	0

distance matrix with A[i,i] = 0

Answer: min (A[i,k]+A[k,j])

k=0 … N-1
~~k≠i, k≠j~~

Shortest Path & Matrix Multiplication

$A^2 =$

	0	1	2	3	4
0	0	25	15	∞	∞
1	12	0	∞	55	∞
2	∞	∞	0	∞	∞
3	∞	∞	45	0	60
4	∞	∞	20	∞	0

A

*

	0	1	2	3	4
0	0	25	15	∞	∞
1	12	0	∞	55	∞
2	∞	∞	0	∞	∞
3	∞	∞	45	0	60
4	∞	∞	20	∞	0

A

Minimum distance 2 hops or less from i to j:

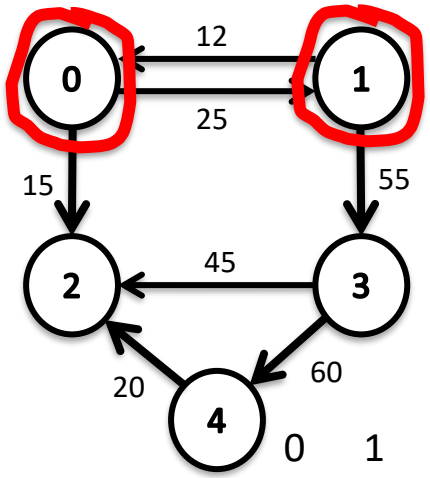
$D_2[i,j] = \min_{k=0 \ldots N-1} (A[i,k]+A[k,j])$

Matrix Multiply ($B = A^2 = A*A$):

$B[i,j] = \sum_{k=0}^{N-1} (A[i,k]*A[k,j])$

Observation: The minimum distance between any two nodes of at most 2 hops can be computed using matrix multiply (A^2) where “minimum” is used rather than addition and addition is used rather than multiplication in each dot product of the matrix multiplication.

Matrix Multiplication Example



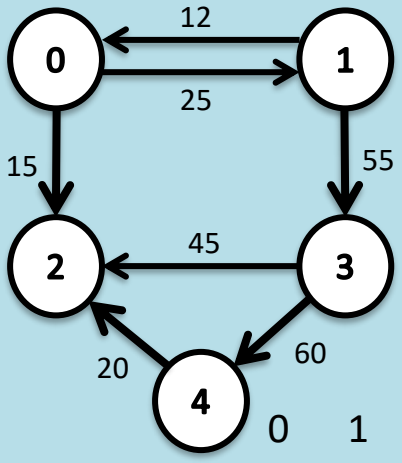
	0	1	2	3	4
0	0	25	15	∞	∞
1	12	0	∞	55	∞
2	∞	∞	0	∞	∞
3	∞	∞	45	0	60
4	∞	∞	20	∞	0

A * A = $D_2 = A^2$

	0	1	2	3	4
0	0	25			
1					
2					
3					
4					

$D_2[i,j]$ = length of minimum path of length 2 or less from i to j

- $D_2[0,0]$ = min (0+0, 25+12, 15+∞, ∞+∞, ∞+∞) = 0
- $D_2[0,1]$ = min (0+25, 25+0, 15+∞, ∞+∞ ∞+∞) = 25



Matrix Multiplication Example

	0	1	2	3	4
0	0	25	15	∞	∞
1	12	0	∞	55	∞
2	∞	∞	0	∞	∞
3	∞	∞	45	0	60
4	∞	∞	20	∞	0

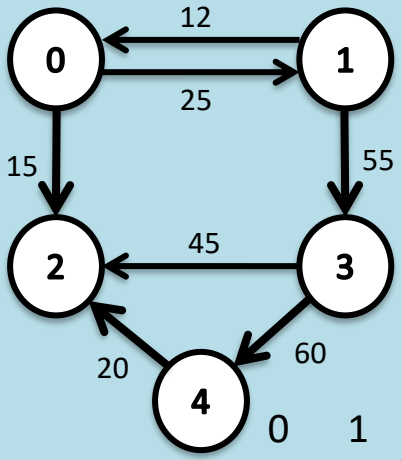
A * A = $D_2 = A^2$

	0	1	2	3	4
0	0	25	15	∞	∞
1	12	0	∞	55	∞
2	∞	∞	0	∞	∞
3	∞	∞	45	0	60
4	∞	∞	20	∞	0

	0	1	2	3	4
0	0	25			
1					
2					
3					
4					

Complete the matrix multiplication by finding the remaining entries in $D_2=A^2$.

Matrix Multiplication Example



	0	1	2	3	4
0	0	25	15	∞	∞
1	12	0	∞	55	∞
2	∞	∞	0	∞	∞
3	∞	∞	45	0	60
4	∞	∞	20	∞	0

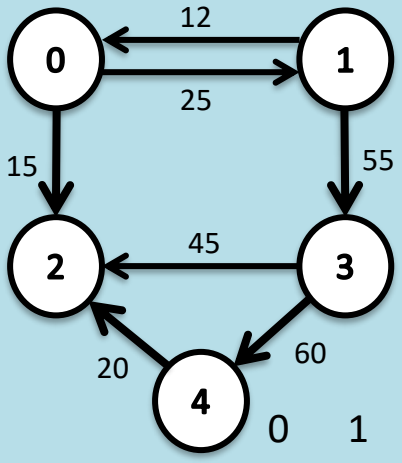
A * A =

	0	1	2	3	4
0	0	25	15	80	∞
1	12	0	27	55	115
2	∞	∞	0	∞	∞
3	∞	∞	45	0	60
4	∞	∞	20	∞	0

$D_2 = A^2$

- $D_2[0,2]$ = min (0+15, 25+∞, 15+0, ∞+45, ∞+20) = 15
- $D_2[0,3]$ = min (0+∞, 25+55, 15+∞, ∞+0, ∞+∞) = 80
- $D_2[0,4]$ = min (0+∞, 25+∞, 15+∞, ∞+60, ∞+0) = ∞

Matrix Multiplication Example



	0	1	2	3	4
0	0	25	15	∞	∞
1	12	0	∞	55	∞
2	∞	∞	0	∞	∞
3	∞	∞	45	0	60
4	∞	∞	20	∞	0

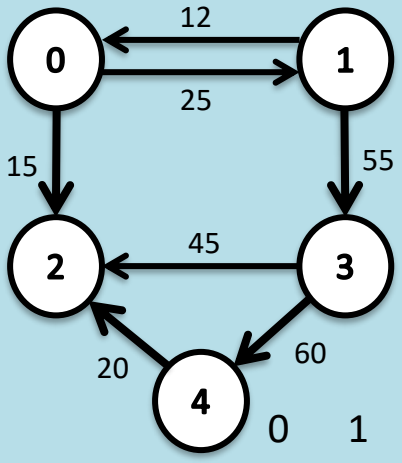
A * A =

	0	1	2	3	4
0	0	25	15	80	∞
1	12	0	27	55	115
2	∞	∞	0	∞	∞
3	∞	∞	45	0	60
4	∞	∞	20	∞	0

$D_2 = A^2$

- $D_2[1,0] = \min(12+0, 0+12, \infty+\infty, 55+\infty, \infty+\infty) = 12$
- $D_2[1,1] = \min(12+25, 0+0, \infty+\infty, 55+\infty, \infty+\infty) = 0$
- $D_2[1,2] = \min(12+15, 0+\infty, \infty+0, 55+45, \infty+20) = 27$
- $D_2[1,3] = \min(12+\infty, 0+55, \infty+\infty, 55+0, \infty+\infty) = 55$
- $D_2[1,4] = \min(12+\infty, 0+\infty, \infty+\infty, 55+60, \infty+0) = 115$

Matrix Multiplication Example



	0	1	2	3	4
0	0	25	15	∞	∞
1	12	0	∞	55	∞
2	∞	∞	0	∞	∞
3	∞	∞	45	0	60
4	∞	∞	20	∞	0

A * A =

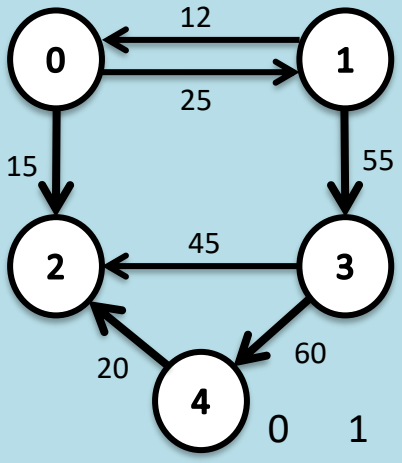
	0	1	2	3	4
0	0	25	15	∞	∞
1	12	0	∞	55	∞
2	∞	∞	0	∞	∞
3	∞	∞	45	0	60
4	∞	∞	20	∞	0

$D_2 = A^2$

	0	1	2	3	4
0	0	25	15	**80**	∞
1	12	0	**27**	55	**115**
2	∞	∞	0	∞	∞
3	∞	∞	45	0	60
4	∞	∞	20	∞	0

- $D_2[2,0] = \min(\infty+0, \infty+12, 0+\infty, \infty+\infty, \infty+\infty) = \infty$
- $D_2[2,1] = \min(\infty+25, \infty+0, 0+\infty, \infty+\infty, \infty+\infty) = \infty$
- $D_2[2,2] = \min(\infty+15, \infty+\infty, 0+0, \infty+45, \infty+20) = 0$
- $D_2[2,3] = \min(\infty+\infty, \infty+55, 0+\infty, \infty+0, \infty+\infty) = \infty$
- $D_2[2,4] = \min(\infty+\infty, \infty+\infty, 0+\infty, \infty+60, \infty+0) = \infty$

Matrix Multiplication Example



	0	1	2	3	4
0	0	25	15	∞	∞
1	12	0	∞	55	∞
2	∞	∞	0	∞	∞
3	∞	∞	45	0	60
4	∞	∞	20	∞	0

A * A =

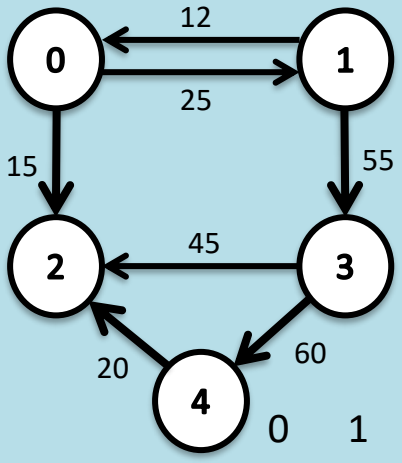
	0	1	2	3	4
0	0	25	15	∞	∞
1	12	0	∞	55	∞
2	∞	∞	0	∞	∞
3	∞	∞	45	0	60
4	∞	∞	20	∞	0

$D_2 = A^2$

	0	1	2	3	4
0	0	25	15	**80**	∞
1	12	0	**27**	55	**115**
2	∞	∞	0	∞	∞
3	∞	∞	45	0	60
4	∞	∞	20	∞	0

- $D_2[3,0] = \min(\infty+0, \infty+12, 45+\infty, 0+\infty, 60+\infty) = \infty$
- $D_2[3,1] = \min(\infty+25, \infty+0, 45+\infty, 0+\infty, 60+\infty) = \infty$
- $D_2[3,2] = \min(\infty+15, \infty+\infty, 45+0, 0+45, 60+20) = 45$
- $D_2[3,3] = \min(\infty+\infty, \infty+55, 45+\infty, 0+0, 60+\infty) = 0$
- $D_2[3,4] = \min(\infty+\infty, \infty+\infty, 45+\infty, 0+60, 60+0) = 60$

Matrix Multiplication Example



	0	1	2	3	4
0	0	25	15	∞	∞
1	12	0	∞	55	∞
2	∞	∞	0	∞	∞
3	∞	∞	45	0	60
4	∞	∞	20	∞	0

A * A =

	0	1	2	3	4
0	0	25	15	80	∞
1	12	0	27	55	115
2	∞	∞	0	∞	∞
3	∞	∞	45	0	60
4	∞	∞	20	∞	0

$D_2 = A^2$

- $D_2[4,0]$ = min (∞+0, ∞+12, 20+∞, ∞+∞, 0+∞) = ∞
- $D_2[4,1]$ = min (∞+25, ∞+0, 20+∞, ∞+∞, 0+∞) = ∞
- $D_2[4,2]$ = min (∞+15, ∞+∞, 20+0, ∞+45, 0+20) = 20
- $D_2[4,3]$ = min (∞+∞, ∞+55, 20+∞, ∞+0, 0+∞) = ∞
- $D_2[4,4]$ = min (∞+∞, ∞+∞, 20+∞, ∞+60, 0+0) =0

Does This Generalize?

- Does the same approach extend to a larger numbers of hops?
- The answer is yes
  - A^3 computes the minimum weight (length) of paths that are 3 hops or less
  - A^4 computes the minimum weight (length) of paths that are 4 hops or less
  - etc.
- Observation: It can be shown that the shortest path "matrix multiply" is associative
  - (A*A)*A = A*(A*A)
  - Doesn't matter what order the "multiplications" are performed

All Pairs Path Length

Given an adjacency matrix A with N vertices, how do we compute the distance between any pair of nodes?

- Can you think of a way to do this at all?
- Can you think of a way to do this efficiently? What is the execution time for your algorithm?

All Pairs Path Length

Given an adjacency matrix A with N vertices, how do we compute the distance between any pair of nodes?

- Maximum length path is N-1 hops (no repeats in path)
- To compute the minimum length path between all pairs of nodes, one need only compute $D_{N-1} = A^{N-1}$
- $D_{N-1}[i,j]$ = distance w/ N-1 or fewer hops from i to j
- Initial approach: perform N-1 multiplications to arrive at A^{N-1}

All Pairs Path Length

- For greater efficiency: Observe that subsequent multiplications yield same result

  $A^N = A^{N-1}$ (would require repeated node: cannot decrease length)

  $A^{N+1} = A^N = A^{N-1}$

  $A^{N+2} = A^{N+1} = A^N = A^{N-1}$

  …

Algorithm

Goal: Compute A^{N-1}

Approach: Keep squaring the matrix to compute A^x until $x \geq N-1$

Step 1: Compute $A*A = A^2$

Step 2: Compute $A^2 * A^2 = A^4$

Step 3: Compute $A^4 * A^4 = A^8$

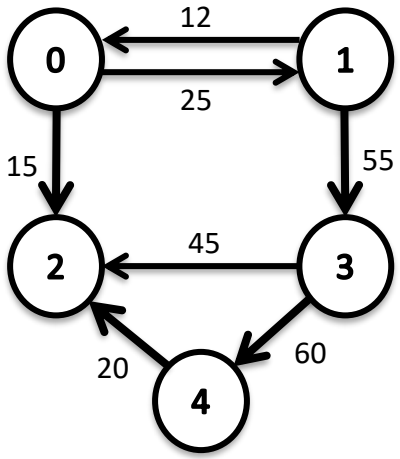
…

Until exponent is at least N-1

Analysis

- What is the execution time?
- Each matrix multiply requires $O(N^3)$
- Need $\lceil \log N-1 \rceil$ multiplication steps (log base2: log(x)/log(2))
- Runtime is $O(N^3 \log N)$

- Note that a limitation is that the algorithm computes the shortest path lengths for all pairs but does not record/ construct the paths!



Matrix Multiplication Example

- Let's finish our example. What we computed as $D_2 = A^2$ is the minimum path length between all pairs of vertices with at most two hops. This can be easily verified.
- With 5 vertices, the paths can have length at most 4, so for all-pairs shortest paths we need to compute $A^2 * A^2 = A^4$. Using successive squaring saves one multiply.

A^2

	0	1	2	3	4
0	0	25	15	80	∞
1	12	0	27	55	115
2	∞	∞	0	∞	∞
3	∞	∞	45	0	60
4	∞	∞	20	∞	0

*

A^2

	0	1	2	3	4
0	0	25	15	80	∞
1	12	0	27	55	115
2	∞	∞	0	∞	∞
3	∞	∞	45	0	60
4	∞	∞	20	∞	0

=

A^4

	0	1	2	3	4
0	0	25	15	80	**140**
1	12	0	27	55	115
2	∞	∞	0	∞	∞
3	∞	∞	45	0	60
4	∞	∞	20	∞	0

Summary: Graphs

- Graphs are a useful abstraction for many problems of practical interest
- Graphs can be represented in in software using various approaches (adjacency matrix, adjacency list); these can be implemented in C using dynamic memory allocation and linked lists
- Breadth first and depth first search are two approaches to visit all nodes of a graph
- Relaxation can be used to compute shortest paths (Bellman-Ford)
- A more efficient relaxation approach can be obtained by being careful about the order in which relaxation is applied to edges (Dijkstra's algorithm)
- Computing minimum-weight paths can be viewed as a matrix computation problem