Lecturer: Andrew Parkes
http://www.cs.nott.ac.uk/~pszajp/

COMP2054 ADE
Shortest Paths:
Floyd-Warshall (FW)

All-Pairs Shortest Paths

- Suppose that wanted to find the SP between **all** pairs of start and end nodes
  - Dijkstra's algorithm finds all shortest path costs from one given starting node
  - [Not assessed] The complexity of Dijkstra is $O(|V| * (|V| * \log(|V|) + |E|))$
  - We could run Dijkstra from every node
  - Would then be a factor of $O(|V|)$ worse than Dijkstra, e.g. $O(|V| * (|V| * \log(|V|) + |E|))$
- But may be better to run a specific algorithm
  - We will do the "Floyd-Warshall" (FW) algorithm

FW All-Pairs SPs

- The basic method has similarities to the methods for “change giving” in an earlier lecture
  - “Build best answers for a set of coins, and then add the effects of coins one at a time”
- “Build the optimal answers using a subset of the nodes. Then add the effects of other nodes one at a time”

FW All-Pairs SPs: data structure

Main data structure definition:

d(i,j,k) =

shortest distance between nodes i and j,

but using only the nodes {1,…,k} as potential allowed intermediary points

- E.g. d(2,5,3) =
  "shortest distance from n2 to n5 using only {n1,n2,n3} as potential intermediate points'."
  - Usage of all or any of these intermediate points is not forced, but other points, such as n4 …, cannot be used

FW: Initialisation of data structure

Initialisation:

d(i,j,0) = best distance between nodes i and j, but not using any intermediate nodes,

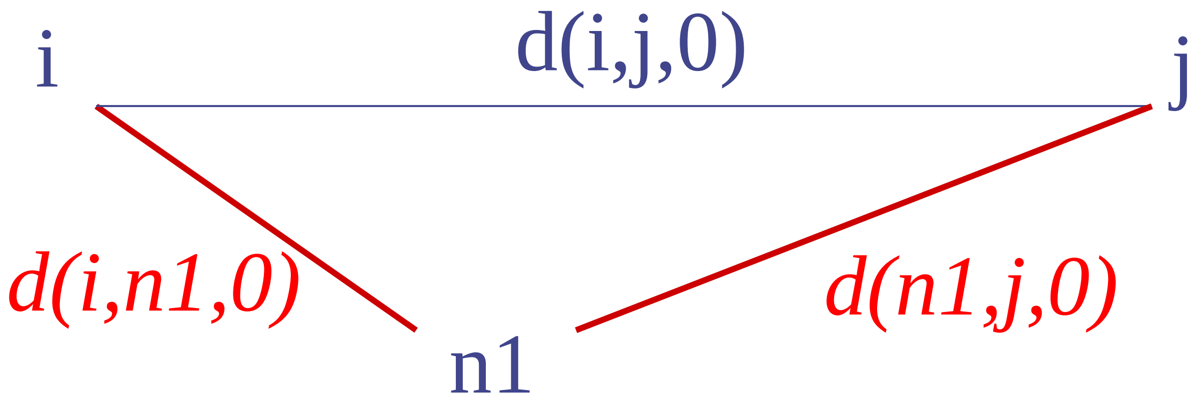
so only using a single edge, Hence

d(i,j,0) = w(i,j) if there is an edge i to j
= Inf otherwise

where Inf is 'infinity', and could be implemented as a special value, or a number large enough to act as infinity (bigger than all other numbers encountered), etc.

FW All-Pairs SPs

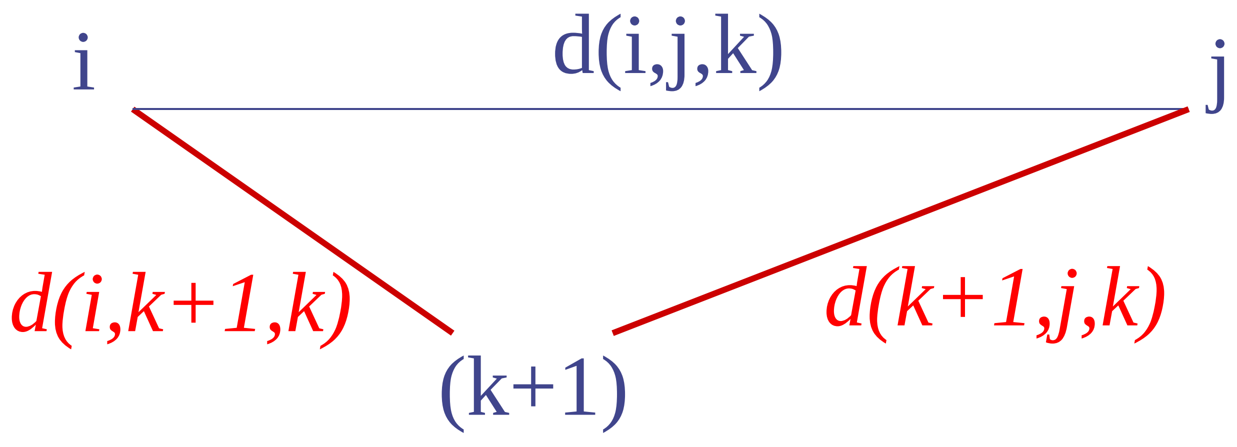
- Now suppose that we add the node 'n1' to the set of nodes that can be intermediates, i.e. consider k = 1
- Best path is now the best of "either direct, or via n1."
- d(i,j,1) = min (d(i,j,0),   w(i,n1) + w(n1,j)   )
  = min (d(i,j,0),   d(i,n1,0) + d(n1,j,0)   )



FW All-Pairs SPs

- Now suppose that we add the new node "(k+1)" to the set of "via nodes" that can be intermediates, but have already considered k of them
- Best path is now either direct using only the k 'via nodes' already accounted for, or else also via node 'k+1' (and using the previous k via's)

$$d(i,j,k+1) = \min\left(d(i,j,k),\ \ d(i,k+1,k) + d(k+1,j,k) \right)$$



FW equations

d(i,j,0) = w(i,j) if there is an edge i to j
= Inf otherwise

d(i,j,k+1) = min (d(i,j,k),
d(i,k+1,k) + d(k+1,j,k))

Note:

- The rhs depends only on d(-,-,k)
- The lhs gives d(-,-,k+1)
- So solve by “start at k=0 and iteratively increase k”

(These are an example of “Belmann Equations” and can be solved using DP.)

FW equations – Comment

k is a dummy variable, and so we could equally well write

$$d(i,j,k) = \min(\ d(i,j,k-1),\ \ d(i,k,k-1) + d(k,j,k-1)\ )$$

Of course, in practice, when implementing, then have to be more careful about the range of k. So that it correctly captures the idea of

- Start with no “via nodes” allowed
- Add “via nodes” one at a time

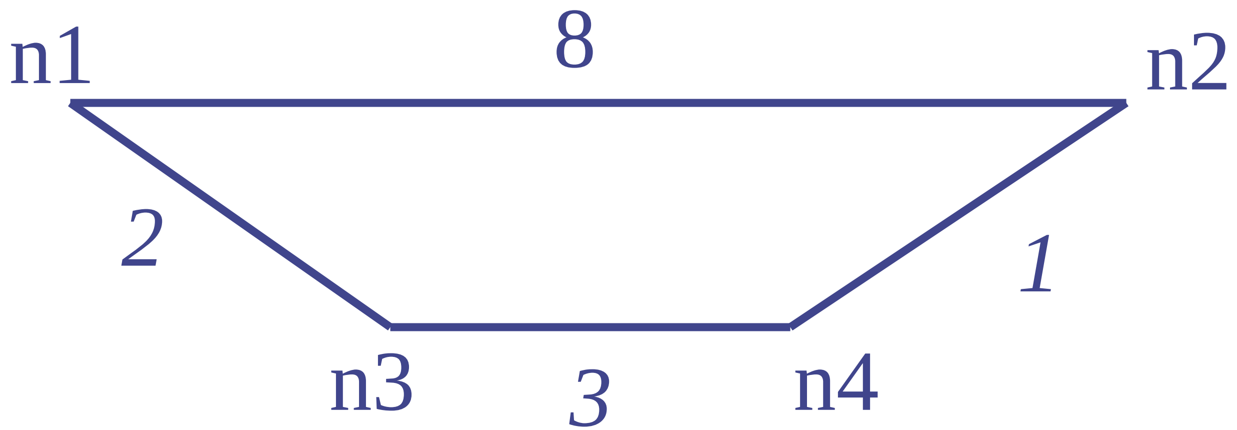
Self-edges?

- The FW format does not in itself make any assumptions about d(i,i)
  - i.e. whether "self-edges" are always assumed to exist.
- However, we will assume that

$$d(\, i, i \,) = 0 \quad \text{forall } i$$

  - that is, travelling from a node A to itself is possible at cost 0.
  - This is for simplicity, and also makes it compatible with Dijkstra which (typically) has the same assumption

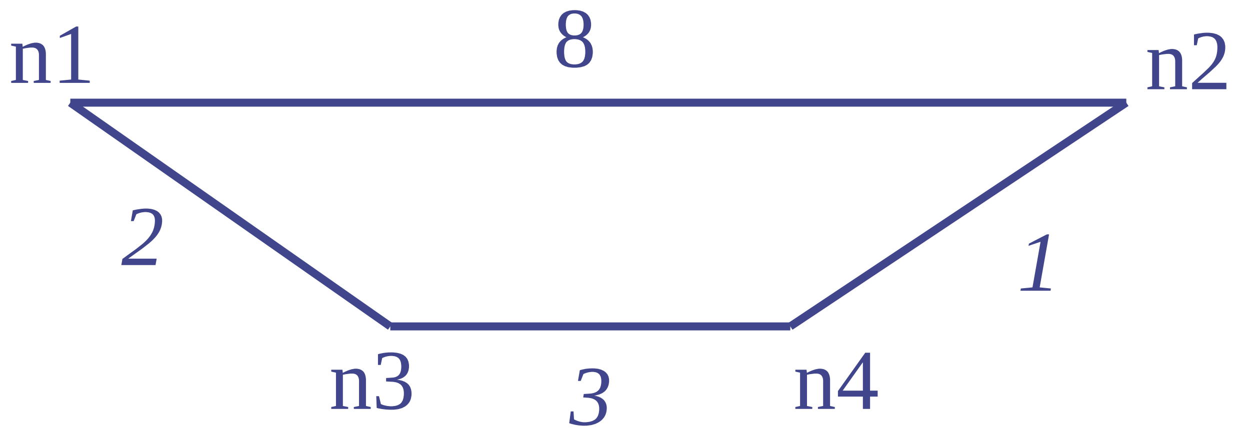
FW example : d(i,j,0)



- Initial d(i,j,0)
  - direct edges only
- Edges are undirected, hence matrix is symmetric

d(i,j,0)	n1	n2	n3	n4
n1	0	8	2	Inf
n2	8	0	Inf	1
n3	2	Inf	0	3
n4	Inf	1	3	0

FW example : d(i,j,1), partial



Set of intermediates = { n1 }

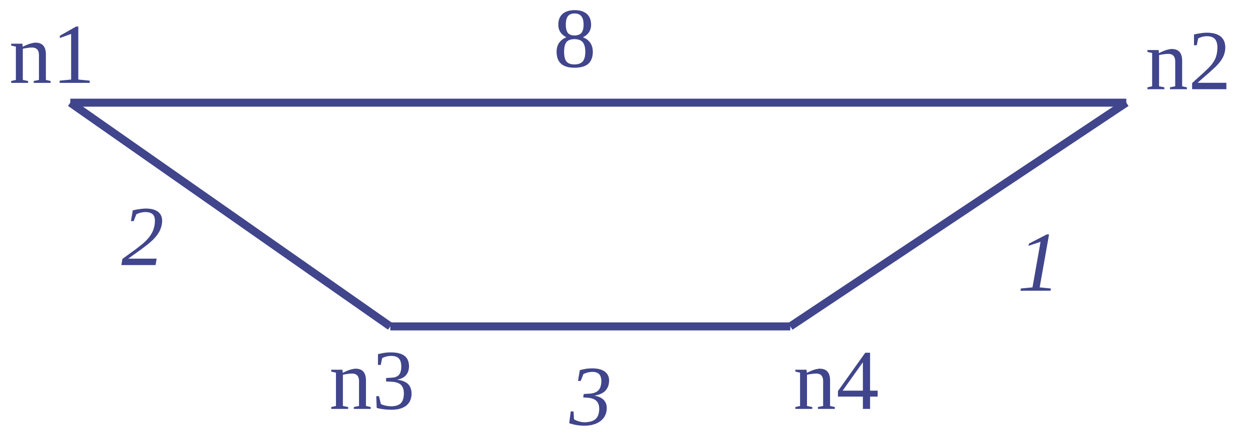
Example:

d(n3,n2,1)
= min(Inf, d(n3,n1,0) + d(n1,n2,0))
= min(Inf, 2+8) = 10

(We can "see" the optimal will be 3+1 via n4, but we have not considered n4 yet)

d(i,j,1)	**n1**	**n2**	**n3**	**n4**
n1	0	8	2	Inf
n2	8	0	~~**Inf**~~ **10**	1
n3	2	~~**Inf**~~ **10**	0	3
n4	Inf	1	3	0

FW example : d(i,j,1)



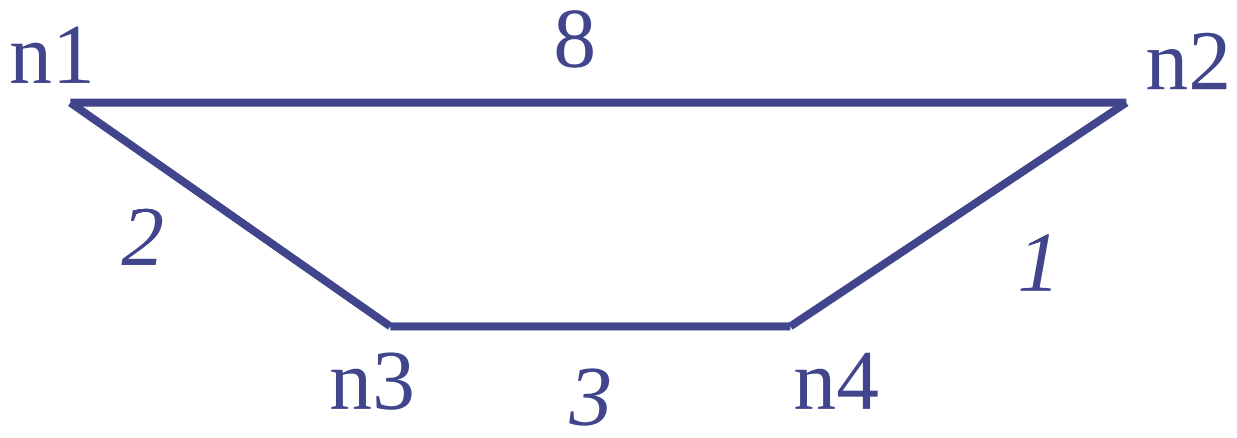
Set of intermediates = { n1 }

Filling in the other entries – check that no other entries change

The matrix is symmetric so will now just give the lower-triangle.

d(i,j,1)	n1	n2	n3	n4
n1	0			
n2	8	0		
n3	2	10	0	
n4	Inf	1	3	0

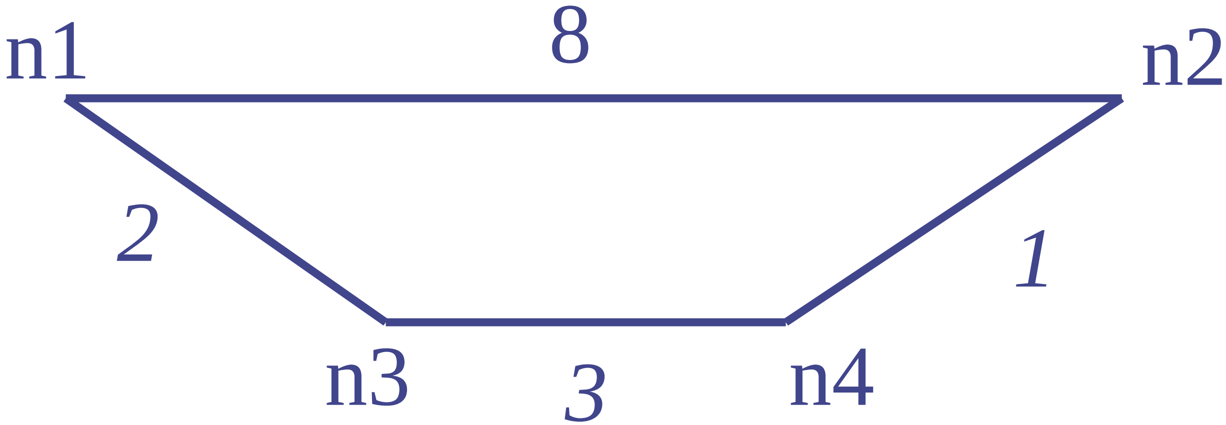
FW example : d(i,j,2)



Set of intermediates = { n1, n2 }
Node n2 is new, so e.g.
d(n4,n1,2)
= min(Inf, d(n4,n2,1) + d(n2,n1,1))
= min(Inf, 1+8) = 9

d(i,j,2)	n1	n2	n3	n4
n1	0			
n2	8	0		
n3	2	10	0	
n4	~~Inf~~ 9	1	3	0

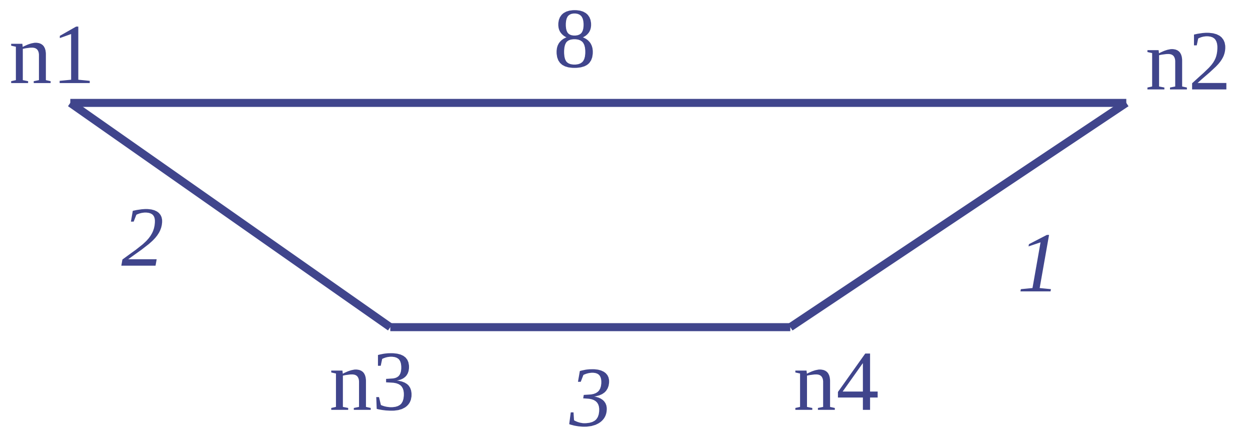
FW example : d(i,j,3)



Set of intermediates = { n1, n2, n3 }
Node n3 is new, so e.g.
d(n4,n1,3)
= min(9, d(n4,n3,2) + d(n3,n1,2))
= min(9, 3+2) = 5

d(i,j,3)	n1	n2	n3	n4
n1	0			
n2	8	0		
n3	2	10	0	
n4	~~9~~ 5	1	3	0

FW example : d(i,j,4)



Set of intermediates = {n1,n2,n3,n4}
Node n4 is new, so e.g.
d(n1,n2,4)
= min(8, d(n1,n4,3) + d(n4,n2,3))
= min(8, 5+1) = 6

d(i,j,4)	n1	n2	n3	n4
n1	0			
n2	~~8~~ 6	0		
n3	2	~~10~~ 4	0	
n4	5	1	3	0

Finished as all intermediates are now accounted for

FW code & complexity

The main loop after initialisation is:

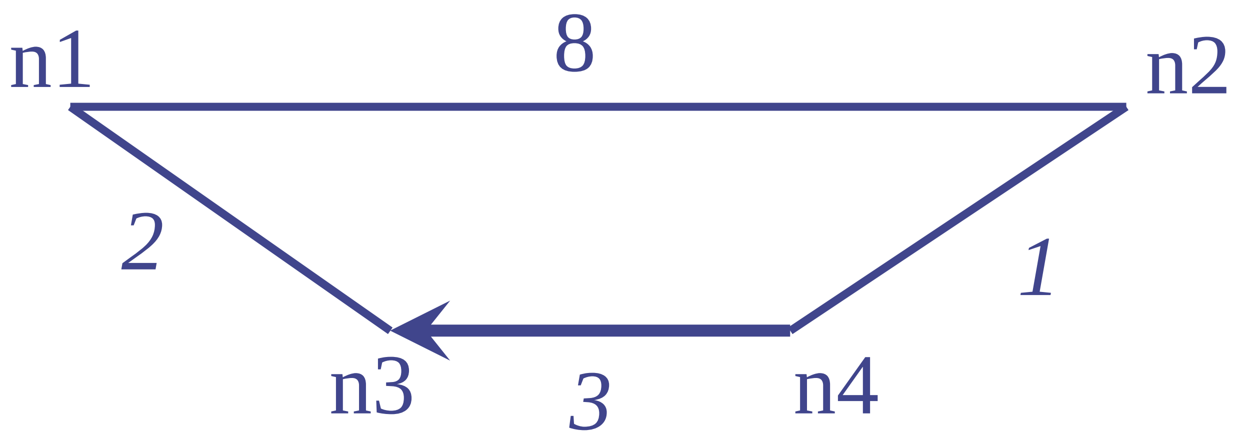
foreach k = 1, … (so "foreach $n_k \in V$")
 foreach $i \in V$
  foreach $j \in V$
   $d(i,j,k+1) = \min(d(i,j,k), d(i,k+1,k) + d(k+1,j,k))$

Note that is it is vital that 'k' is the outer loop.
Have 3 nested loops, of ranges |V|. **Hence is O($|V|^3$)**
(If the graph is sparse, then $|E| << |V|^2$, so then "all-starts Dijkstra" may be better. "<<" means "much less than".)

FW on digraphs

- FW also works the same on directed graphs
    - The initial matrix d(i,j,0) need not be symmetric, but then the remaining calculations use exactly the same formulas

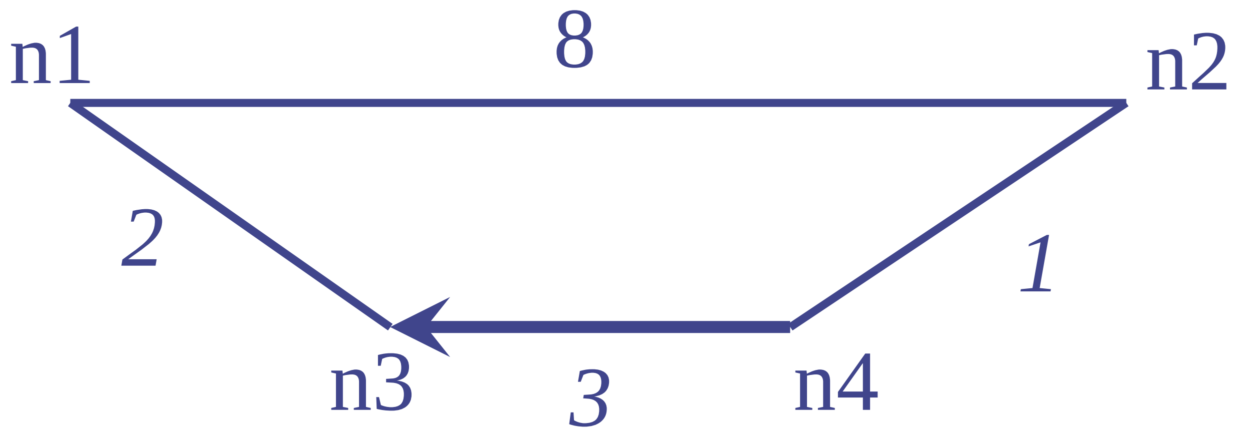
FW digraph example : d(i,j,0)



- Initial d(i,j,0)
  - direct edges only
  - edges with no arrows are bidirectional
- Matrix is no longer symmetric

d(i,j,0)	n1	n2	n3	n4
n1	0	8	2	Inf
n2	8	0	Inf	1
n3	2	Inf	0	**Inf**
n4	Inf	1	3	0

FW example : d(i,j,1), partial



Set of intermediates = { n1 }

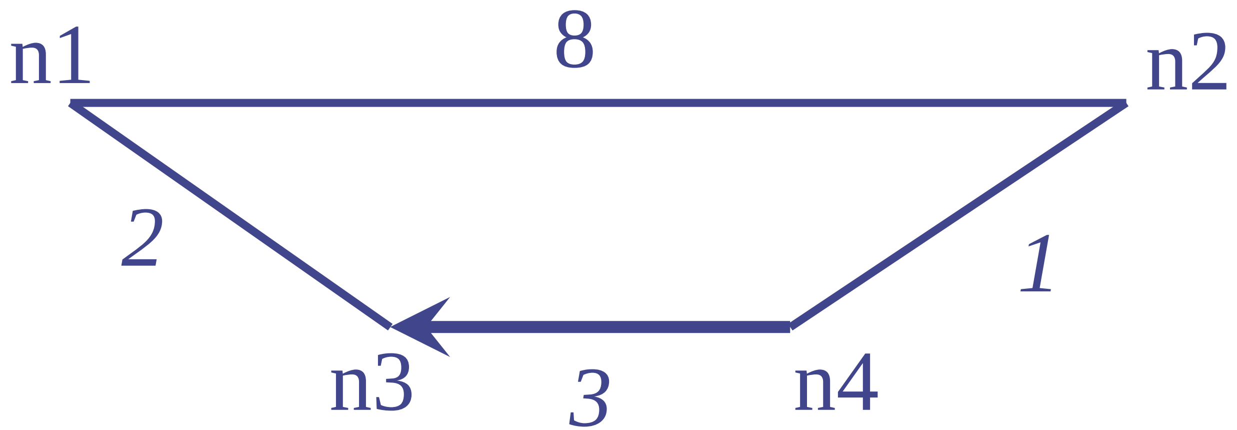
Example:

d(n2,n3,1)
= min(Inf, d(n2,n1,0) + d(n1,n3,0))
= min(Inf, 8+2) = 10

(We can "see" the optimal will be 3+1 via n4, but we have not considered n4 yet)

d(i,j,1)	n1	n2	n3	n4
n1	0	8	2	Inf
n2	8	0	~~Inf~~ **10**	1
n3	2	~~Inf~~ **10**	0	Inf
n4	Inf	1	3	0

FW example : d(i,j,1)

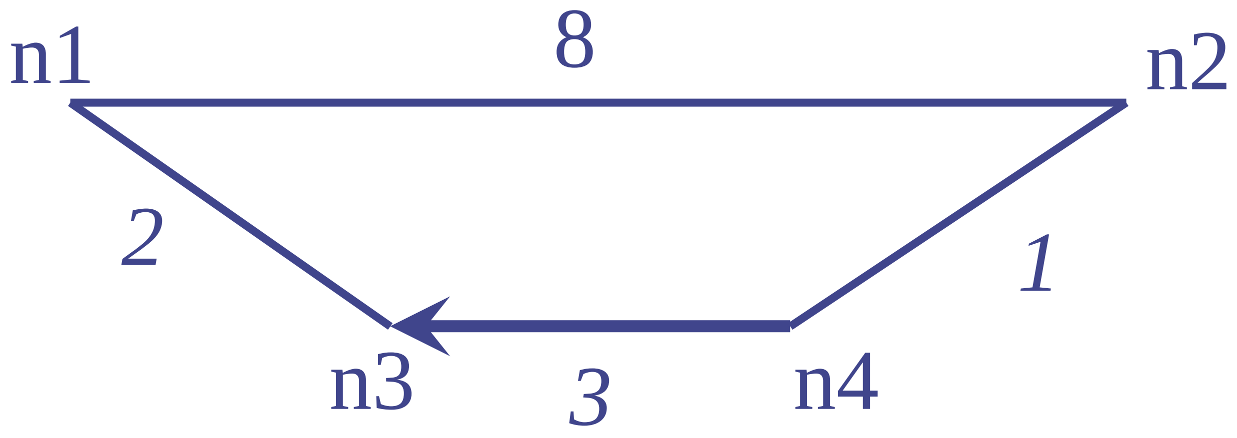


Set of intermediates = { n1 }

Filling in the other entries – check that no other entries change

d(i,j,1)	n1	n2	n3	n4
n1	0	8	2	Inf
n2	8	0	10	1
n3	2	10	0	Inf
n4	Inf	1	3	0

FW example : d(i,j,2)

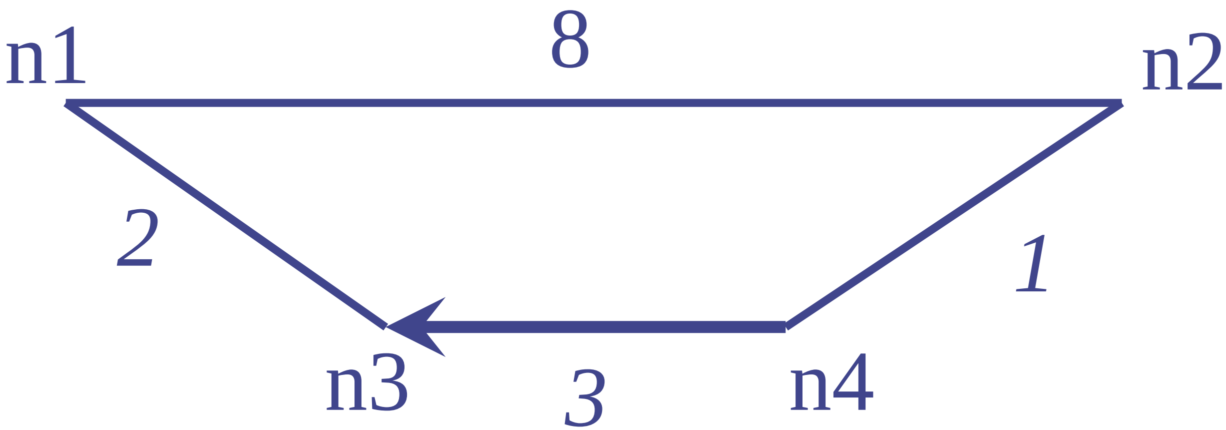


Set of intermediates = { n1, n2 }
Node n2 is new, so e.g.
d(n4,n1,2)
= min(Inf, d(n4,n2,1) + d(n2,n1,1))
= min(Inf, 1+8) = 9 etc

d(n3,n4,2)
= min(Inf, d(n3,n2,1) + d(n2,n4,1))
= min(Inf, 10+1) = 11

d(i,j,2)	n1	n2	n3	n4
n1	0	8	2	~~Inf~~ 9
n2	8	0	10	1
n3	2	10	0	~~Inf~~ 11
n4	~~Inf~~ 9	1	3	0

FW example : d(i,j,3)

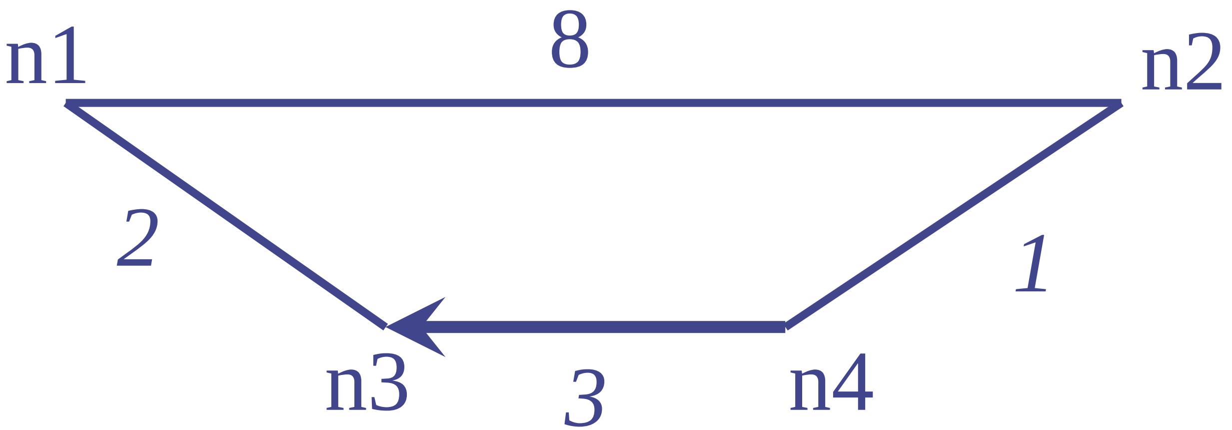


Set of intermediates = { n1, n2, n3 }
Node n3 is new, so e.g.
d(n4,n1,3)
= min(9, d(n4,n3,2) + d(n3,n1,2))
= min(9, 3+2) = 5

but
d(n1,n4,3)
= min(9, d(n1,n3,2) + d(n3,n4,2))
= min(9, 2+Inf) = 9

d(i,j,3)	**n1**	**n2**	**n3**	**n4**
n1	0	8	2	**9**
n2	8	0	10	1
n3	2	10	0	11
n4	~~9~~ **5**	1	3	0

FW example : d(i,j,4)



Set of intermediates = {n1,n2,n3,n4}
Node n4 is new, so e.g.
d(n2,n1,4)
= min(8, d(n2,n4,3) + d(n4,n1,3))
= min(8, 1 + 5) = 6

d(i,j,4)	**n1**	**n2**	**n3**	**n4**
n1	0	8	2	9
n2	~~8~~ **6**	0	~~10~~ **4**	1
n3	2	10	0	11
n4	5	1	3	0

Finished as all intermediates are now accounted for

FW with negative edges

- FW even works if some (directed) edge weights are negative
  - BUT it is essential that there are no cycles of total negative weight
  - Otherwise simply repeatedly following around the negative cycle may reduce lengths to be as negative as desired, so there is no shortest path

- Offline Exercise: repeat the previous example with d(n4,n3) = -1 (instead of +3)
  - The cycle n4-n3-n1-n2-n4 has weight -1+2+8+1=10 which is allowed

FW: Recall - key idea

Uses that SP does satisfy a nice decomposition of

- If P(A,B) is a shortest path, and goes via M then P(A,M) is optimal for A to M and P(M B) is optimal for M to B
- Hence uses a version of “dynamic programming”

Exercise

- You are **highly** recommended to
  - create some small to medium graphs (directed and undirected) and work through both (Dijkstra and FW) algorithms
  - repeat working examples until you **understand** them fully and can do it ‘by hand’ quickly and easily
    - Both algorithms are classics, and similar ideas appear in many other algorithms

Minimum Expectations

- Know and understand definition of all-pairs shortest path, and Floyd-Warshall algorithm
- Be able to apply FW, by hand, to small graphs