All-pairs shortest paths

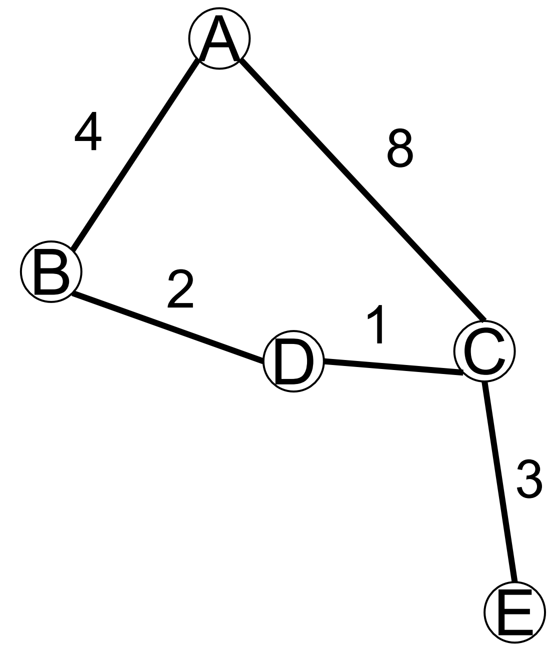
	Dingle *An Daingean*	Annascaul *Abhainn an Scáil*	Ballydavid *Baile na nGall*	Ballyferriter *Baile an Fheirtéaraigh*	Brandon *Cé Bhréanainn*	Camp *An Com*	Castlegregory *Caisleán Ghriaire*	Cloghane *An Clochán*	Conor Pass *An Conair*	Dunquin *Dún Chaoin*	Feohanagh *An Fheothanach*	Inch *An Inse*	Kinard *Ceann Áird*	Lispole *Lios Póil*	Minard *Minn Aird*	Ventry *Ceann Trá*
Dingle ***An Daingean***		17.4	11.5	11.7	20.0	32.5	25.8	14.9	8.3	20.9	12.2	23.7	9.1	8.3	13.9	7.5
Annascaul *Abhainn an Scáil*	10.8		28.8	29.0	42.2	15.1	24.2	37.1	25.7	38.2	29.6	7.1	13.6	9.1	8.4	24.9
Ballydavid *Baile na nGall*	7.1	17.9		7.2	31.4	43.9	37.3	26.4	19.8	15.8	3.9	35.1	20.7	19.8	25.3	11.4
Ballyferriter *Baile an Fheirtéaraigh*	7.3	18.0	4.5		31.6	44.2	37.5	26.6	20.0	8.6	10.3	35.3	20.9	20.0	25.5	5.9
Brandon *Cé Bhréanainn*	12.4	26.2	19.5	19.6		25.7	19.8	5.0	11.7	40.8	31.2	39.9	29.2	28.3	33.9	27.5
Camp *An Com*	20.2	9.4	27.3	27.4	16.0		9.1	22.0	23.4	52.5	42.9	14.2	28.7	24.2	23.5	40.0
Castlegregory *Caisleán Ghriaire*	16.1	15.0	23.2	23.3	12.3	5.7		16.2	17.5	46.6	37.1	23.3	37.7	33.3	32.6	33.3
Cloghane *An Clochán*	9.3	23.1	16.4	16.5	3.1	13.7	10.0		6.6	35.7	26.2	34.9	23.8	23.3	28.8	22.5
Conor Pass *An Conair*	5.2	16.0	12.3	12.4	7.3	14.5	10.9	4.1		29.1	19.5	32.0	24.4	16.6	22.2	15.8
Dunquin *Dún Chaoin*	13.0	23.7	9.8	5.3	25.3	32.6	29.0	22.2	18.1		18.9	44.5	29.9	29.2	34.7	13.4
Feohanagh *An Fheothanach*	7.6	18.4	2.4	6.4	19.4	26.7	23.0	16.3	12.1	11.7		35.9	21.3	20.5	26.1	17.1
Inch *An Inse*	14.7	4.4	21.8	22.0	24.8	8.8	14.5	21.7	19.9	27.7	22.3		20.1	15.4	14.7	31.2
Kinard *Ceann Áird*	5.6	8.4	12.9	13.0	18.2	17.8	23.4	14.8	15.2	18.6	13.2	12.5		4.4	8.4	16.6
Lispole *Lios Póil*	5.2	5.6	12.3	12.4	17.6	15.0	20.7	14.5	10.3	18.1	12.8	9.5	2.7		3.4	15.8
Minard *Minn Aird*	8.6	5.2	15.7	15.9	21.0	14.6	20.3	17.9	13.8	21.6	16.2	9.1	5.2	5.5		21.4
Ventry *Ceann Trá*	4.7	15.5	7.1	3.7	17.1	24.9	20.7	14.0	9.8	8.3	10.6	19.4	10.3	9.8	13.3	

KILOMETRES

MILES

Given a weighted graph, compute the length of the shortest paths between all pairs of vertices.

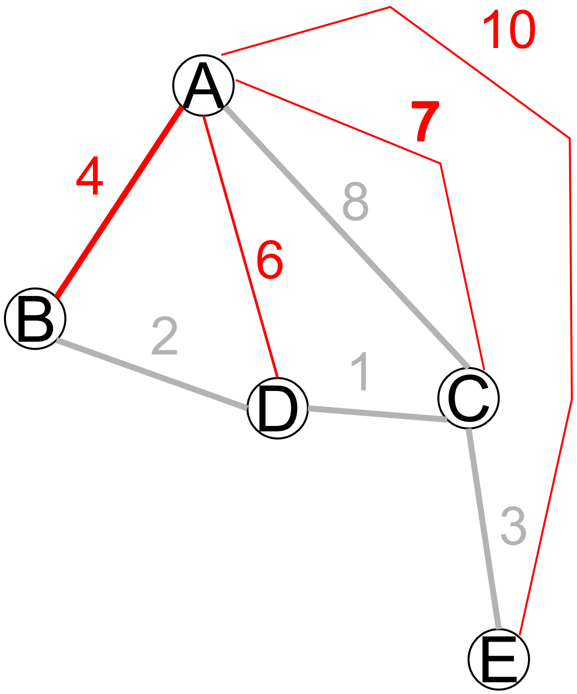
Algorithm?



Given a weighted graph, compute the length of the shortest paths between all pairs of vertices.

Repeated runs of Dijkstra?

Source A:	B	C	D	E
	4	7	6	10

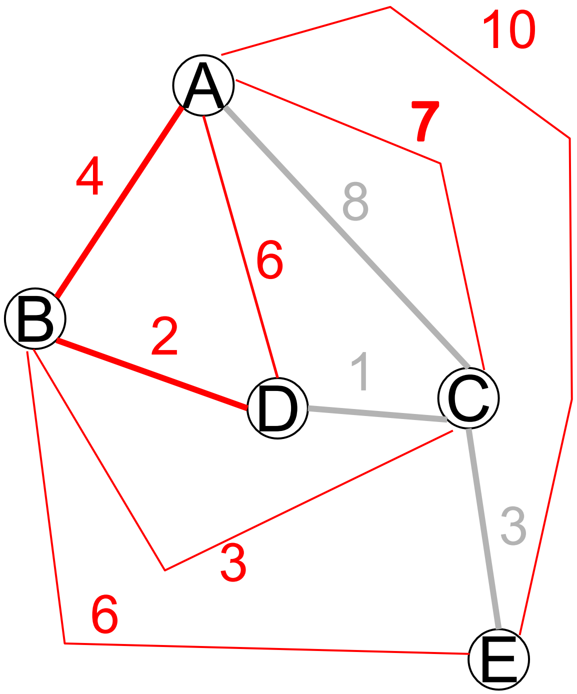


Given a weighted graph, compute the length of the shortest paths between all pairs of vertices.

Repeated runs of Dijkstra?

Source A:	B	C	D	E
	4	7	6	10

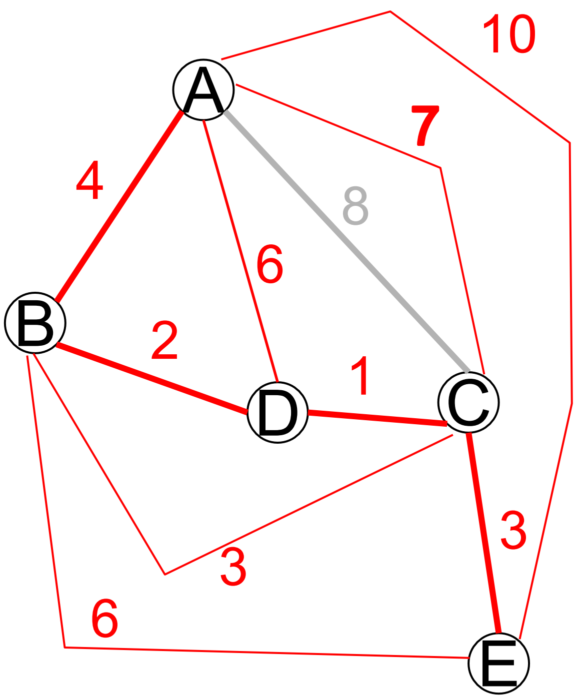
Source B:	A	C	D	E
	4	3	2	6



Given a weighted graph, compute the length of the shortest paths between all pairs of vertices.

Repeated runs of Dijkstra?

Source A:	B	C	D	E
	4	7	6	10
Source B:	A	C	D	E
	4	3	2	6
Source C:	A	B	D	E
	7	3	1	3



Given a weighted graph, compute the length of the shortest paths between all pairs of vertices.

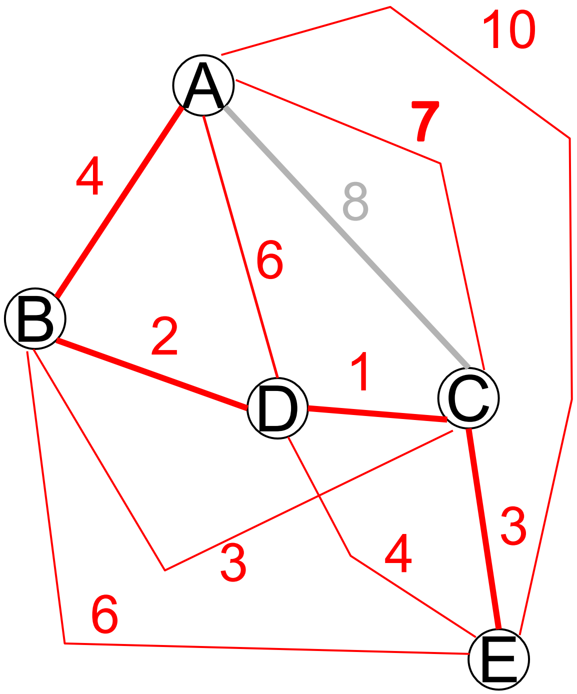
Repeated runs of Dijkstra?

Source A:	B	C	D	E
	4	7	6	10

Source B:	A	C	D	E
	4	3	2	6

Source C:	A	B	D	E
	7	3	1	3

Source D:	A	B	C	E
	6	2	1	4



Given a weighted graph, compute the length of the shortest paths between all pairs of vertices.

Repeated runs of Dijkstra?

Source A:	B	C	D	E
	4	7	6	10

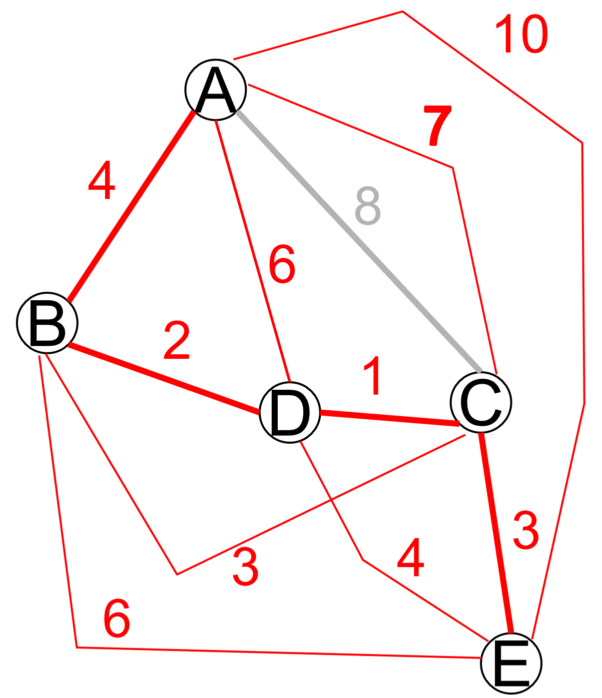
Source B:	A	C	D	E
	4	3	2	6

Source C:	A	B	D	E
	7	3	1	3

Source D:	A	B	C	E
	6	2	1	4

Source E:	A	B	C	D
	10	6	3	4

(don’t need to run Dijkstra for this one …)



Maintain as a dictionary of dictionaries?

How should we compute 'all pairs shortest paths'?

For each vertex v in the graph
compute the shortest path from v to all other vertices using Dijkstra?

Our Dijkstra implementation found paths from source to all other vertices in its component

- therefore, running Dijkstra *n-1* times will find all pairs path costs in an undirected graph (need *n* runs for a directed graph)

Single run of Dijkstra :

	Sparse graph	Dense graph
Dijkstra-heap	**$O(n \log n)$**	$O(n^2 \log n)$
Dijkstra-list	$O(n^2)$	**$O(n^2)$**

So for all pairs: (O(n) runs of Dijkstra …)

	Sparse graph	Dense graph
Dijkstra-heap	**$O(n^2 \log n)$**	$O(n^3 \log n)$
Dijkstra-list	$O(n^3)$	**$O(n^3)$**

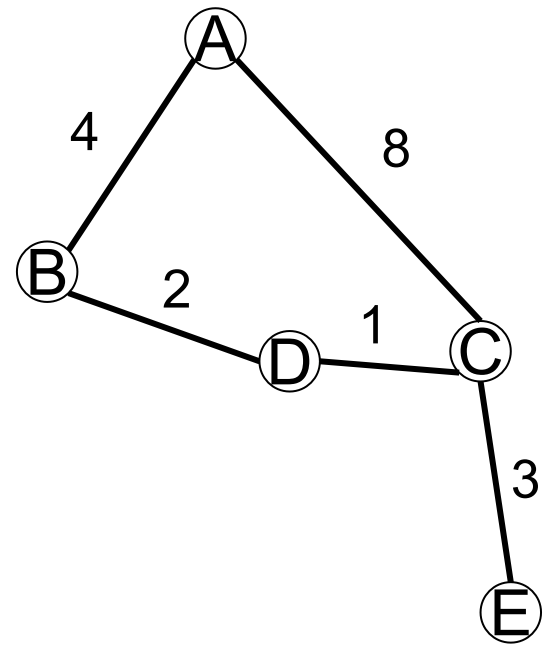
Given a weighted graph, compute the length of the shortest paths between all pairs of vertices.

Algorithm?

Adapt the Floyd Warshall algorithm for finding the transitive closure?

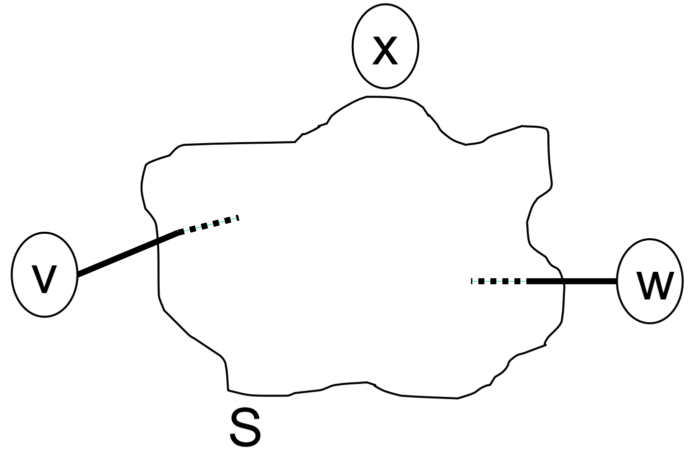
1. ~~Find all-pairs shortest paths that only transit through vertices in {A}~~
2. ~~Find all pairs shortest paths that only transit through vertices in {A,B}~~

etc



1. Find all-pairs shortest paths in G0 that only transit through A, and if they are better, add an edge for those paths, to get G1
2. Find all pairs shortest paths in G1 that only transit through B, and if they are better, add an edge for those paths, to get G2

etc



Let P be the shortest path from v to w which only visits vertices in S, and we will say cost(P) = *sp(v,w,S).*

Now let Q be the shortest path from v to w which only visits vertices in $S \cup \{x\}$

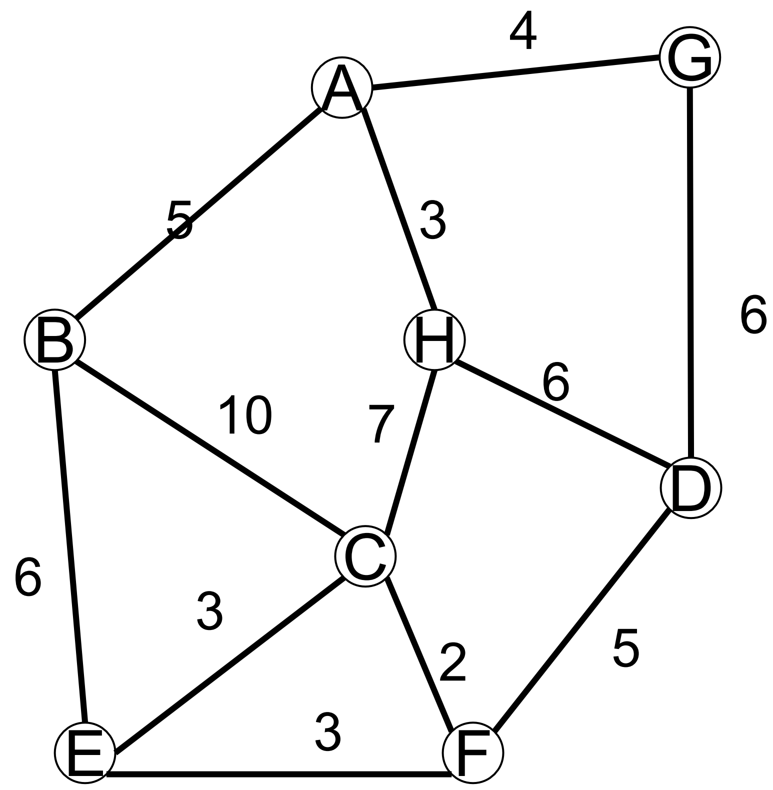
Either Q = P, or Q is made up of the shortest path from v to x which only visits vertices in S, then the shortest path from x to w which only visits vertices in S.

So if we have sp(v,w,S), and sp(v,x,S) and sp(x,w,S), we can compute $sp(v,w,S \cup \{x\}) = \text{minimum}(\ sp(v,w,S),\ \ sp(v,x,S)+sp(x,w,S)\ )$

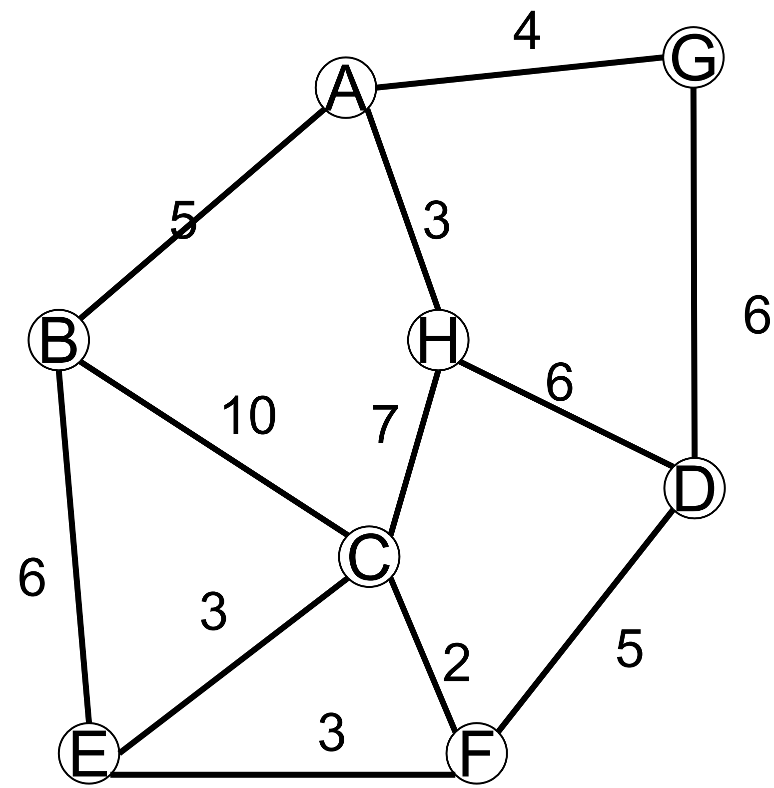
floydwarshall(): pseudocode

```
initialise an nxn 2D structure with value ∞ for each entry
for each v in graph
    assign   table[v][v] = 0     #cost of path from v to v = 0
#begin visitable set S = {}, so shortest paths visiting only S start as just edge costs
for each pair of vertices (x,y)
    if {x,y} is an edge in the graph
        table[x][y] = w((x,y))      #initial cost of path is the edge weight, if edge exists
        table[y][x] = w((x,y))
    else
        table[x][y] = infinity  # i.e. no path yet found
        table[y][x] = infinity
for each v in the graph       #each time round the loop, add v to S
    for each w in the graph #but not v
        for each x in the graph #but not v
            if table[w][x] > table[w][v] + table[v][x]:     #if path via v is cheaper
                table[w][x] = table[w][v] + table[v][x]     #record that as shortest path
return table
```

Given a weighted graph, compute the length of the shortest paths between all pairs of vertices.



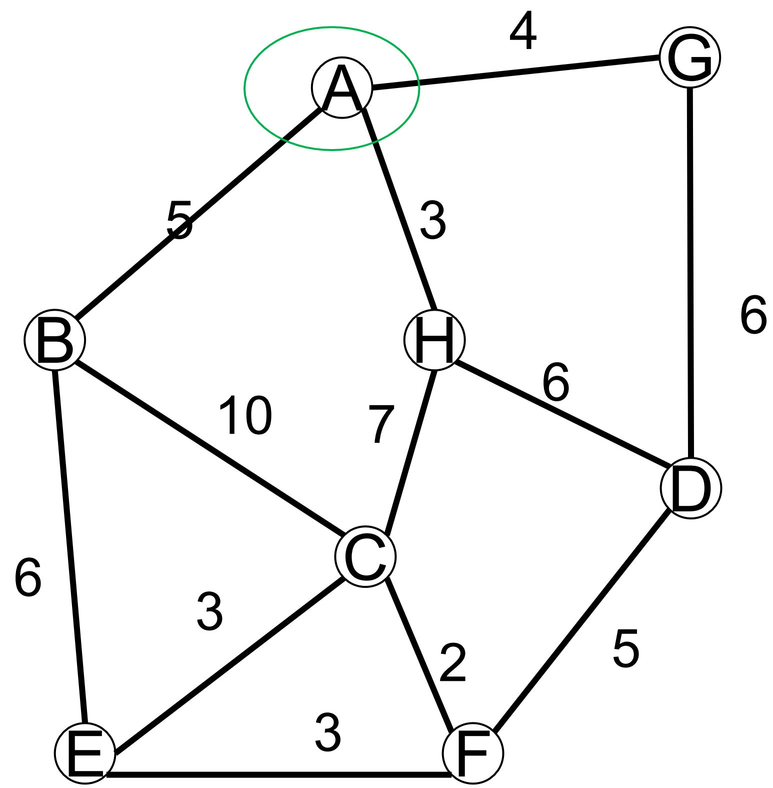
Given a weighted graph, compute the length of the shortest paths between all pairs of vertices.



Initialised – cost of edges in graph, or 0 for the main diagonal, ∞ if no edge

	A	B	C	D	E	F	G	H
A	0	5	∞	∞	∞	∞	4	3
B	5	0	10	∞	6	∞	∞	∞
C	∞	10	0	∞	3	2	∞	7
D	∞	∞	∞	0	∞	5	6	6
E	∞	6	3	∞	0	3	∞	∞
F	∞	∞	2	5	3	0	∞	∞
G	4	∞	∞	6	∞	∞	0	∞
H	3	∞	7	6	∞	∞	∞	0

Given a weighted graph, compute the length of the shortest paths between all pairs of vertices.



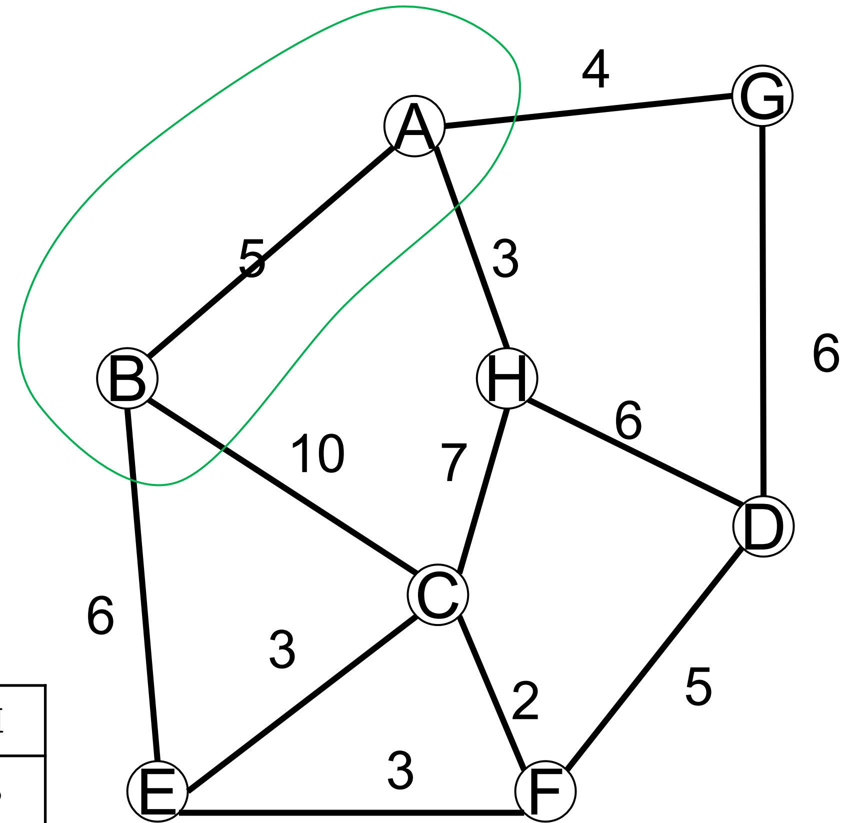
Augmented with all paths via: {A}

	A	B	C	D	E	F	G	H
A	0	5	∞	∞	∞	∞	4	3
B	5	0	10	∞	6	∞	9	8
C	∞	10	0	∞	3	2	∞	7
D	∞	∞	∞	0	∞	5	6	6
E	∞	6	3	∞	0	3	∞	∞
F	∞	∞	2	5	3	0	∞	∞
G	4	9	∞	6	∞	∞	0	7
H	3	8	7	6	∞	∞	7	0

```
[B][A]+[A][G]= 5+4 = 9 < ∞
[B][A]+[A][H]= 5+3 = 8 < ∞
[G][A]+[A][H]= 4+3 = 7 < ∞

(and symmetric …)
```

Given a weighted graph, compute the length of the shortest paths between all pairs of vertices.

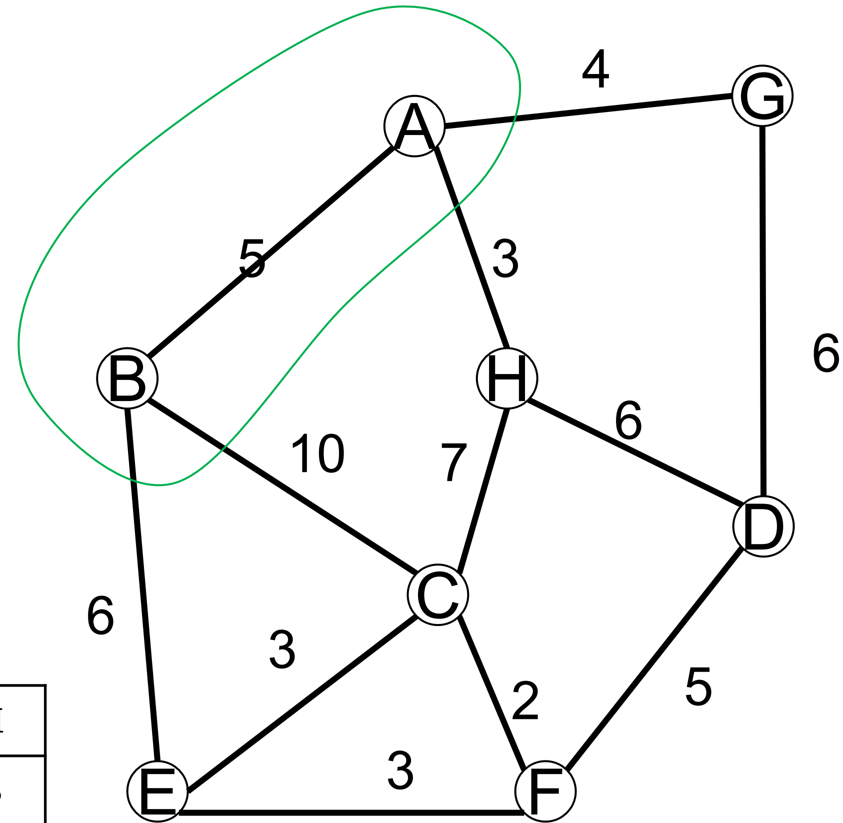


Augmented with all paths via: {A, B}

	A	B	C	D	E	F	G	H
A	0	5	∞	∞	∞	∞	4	3
B	5	0	10	∞	6	∞	9	8
C	∞	10	0	∞	3	2	∞	7
D	∞	∞	∞	0	∞	5	6	6
E	∞	6	3	∞	0	3	∞	∞
F	∞	∞	2	5	3	0	∞	∞
G	4	9	∞	6	∞	∞	0	7
H	3	8	7	6	∞	∞	7	0

E.g. [A][C] was ∞ `but [A][B]=5`
`+ [B][C]=10`
`= 15`
`So update [A][C] to 15`

Given a weighted graph, compute the length of the shortest paths between all pairs of vertices.



Augmented with all paths via: {A, B}

	A	B	C	D	E	F	G	H
A	0	5	∞	∞	∞	∞	4	3
B	5	0	10	∞	6	∞	9	8
C	∞	10	0	∞	3	2	∞	7
D	∞	∞	∞	0	∞	5	6	6
E	∞	6	3	∞	0	3	∞	∞
F	∞	∞	2	5	3	0	∞	∞
G	4	9	∞	6	∞	∞	0	7
H	3	8	7	6	∞	∞	7	0

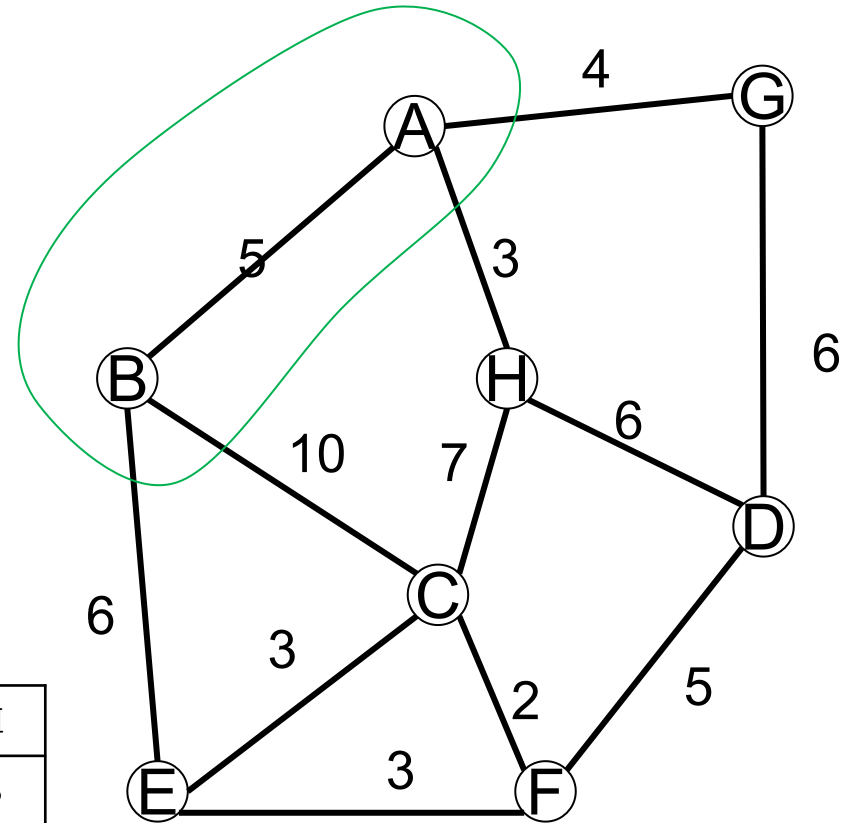
E.g. [C][H] was `7 and [C][B]=10`

`+ [B][H]=8`

`= 18`

`So do not update [C][H]`

Given a weighted graph, compute the length of the shortest paths between all pairs of vertices.

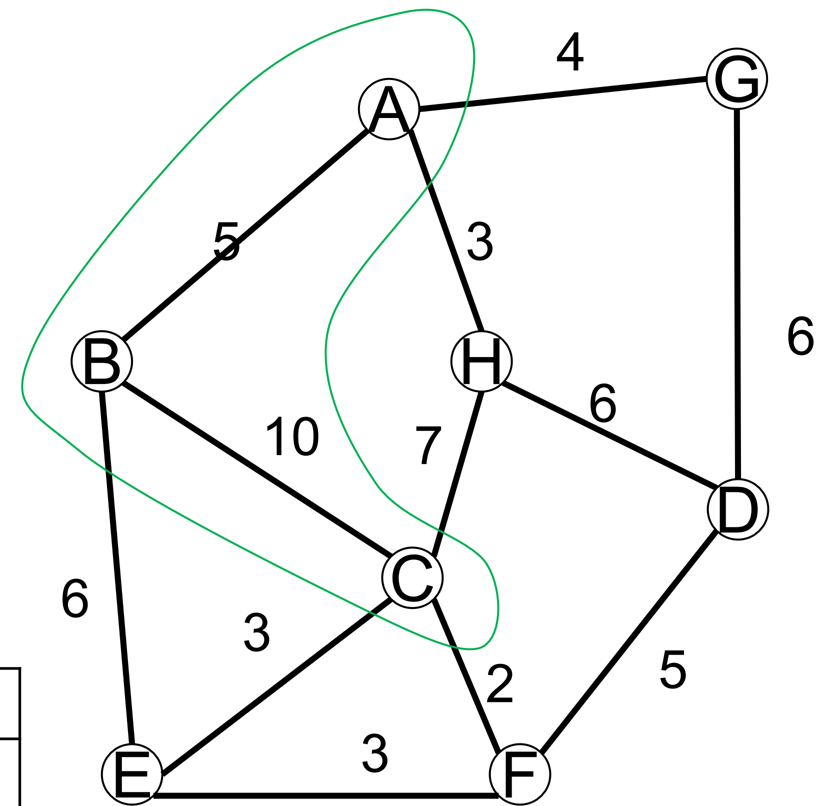


Augmented with all paths via: {A, B}

	A	B	C	D	E	F	G	H
A	0	5	15	∞	11	∞	4	3
B	5	0	10	∞	6	∞	9	8
C	15	10	0	∞	3	2	19	7
D	∞	∞	∞	0	∞	5	6	6
E	11	6	3	∞	0	3	15	14
F	∞	∞	2	5	3	0	∞	∞
G	4	9	19	6	15	∞	0	7
H	3	8	7	6	14	∞	7	0

[A][B]+[B][C]= 5+10=15 < ∞
[A][B]+[B][E]= 5+6 =11 < ∞
[C][B]+[B][G]= 10+9=19 < ∞
[E][B]+[B][G]= 6+9 =15 < ∞
[E][B]+[B][H]= 6+8 =14 < ∞

Given a weighted graph, compute the length of the shortest paths between all pairs of vertices.

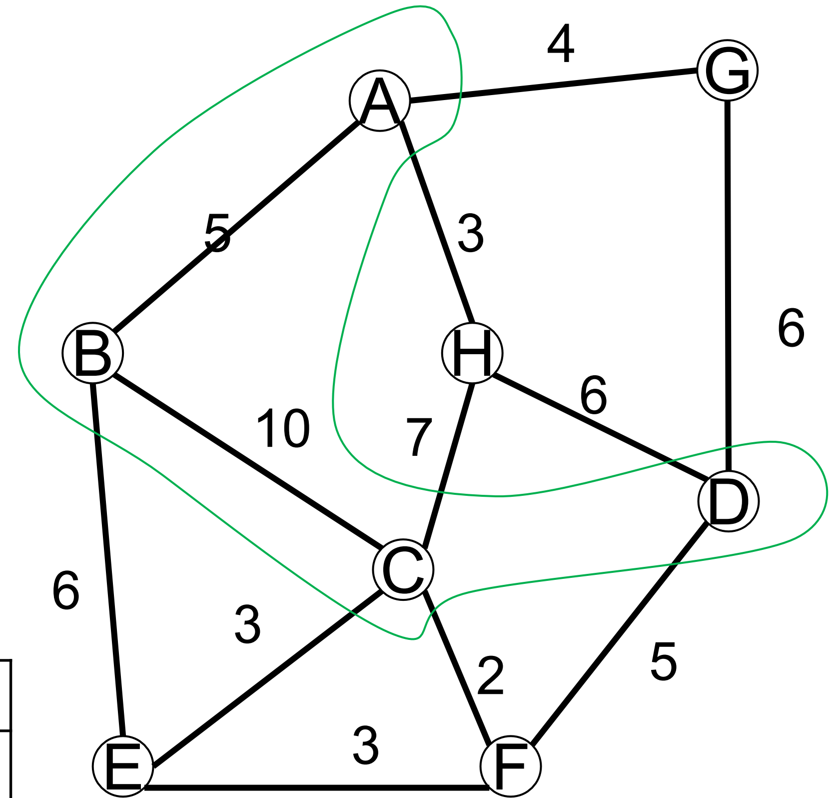


Augmented with all paths via: {A, B, C}

	A	B	C	D	E	F	G	H
A	0	5	15	∞	11	17	4	3
B	5	0	10	∞	6	12	9	8
C	15	10	0	∞	3	2	19	7
D	∞	∞	∞	0	∞	5	6	6
E	11	6	3	∞	0	3	15	10
F	17	12	2	5	3	0	21	9
G	4	9	19	6	15	21	0	7
H	3	8	7	6	10	9	7	0

[A][C]+[C][F]= 15+2=17 < ∞
[B][C]+[C][F]= 10+2=12 < ∞
[E][C]+[C][H]= 3+7 =10 < 14
[F][C]+[C][G]= 2+19=21 < ∞
[F][C]+[C][H]= 2+7 = 9 < ∞

Given a weighted graph, compute the length of the shortest paths between all pairs of vertices.

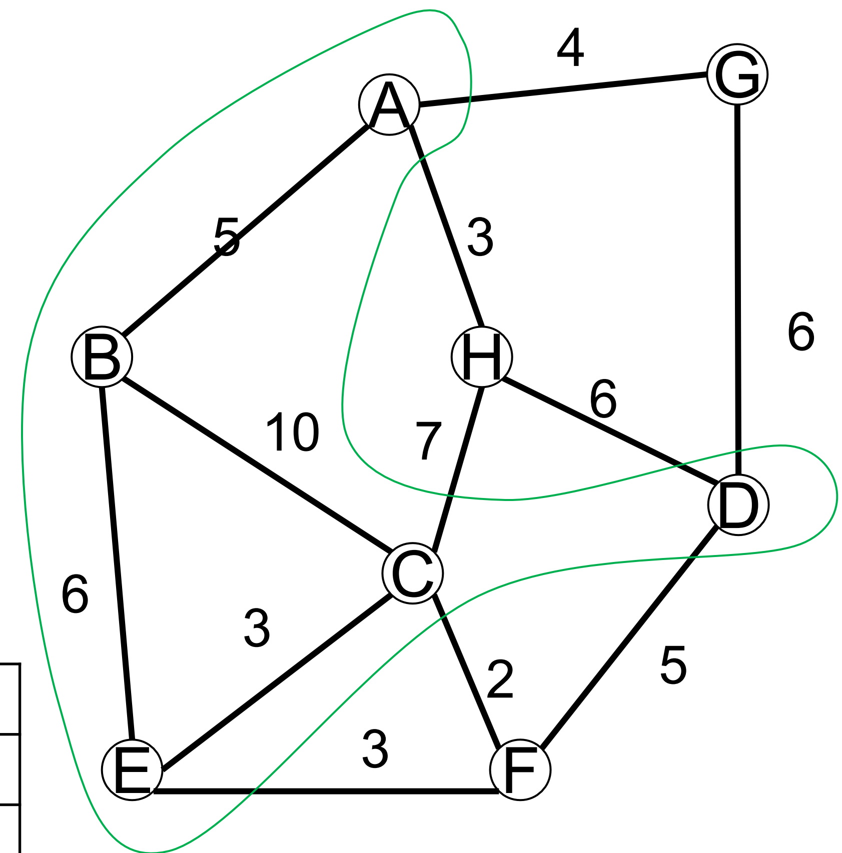


Augmented with all paths via:
{A, B, C, D}

	A	B	C	D	E	F	G	H
A	0	5	15	∞	11	17	4	3
B	5	0	10	∞	6	12	9	8
C	15	10	0	∞	3	2	19	7
D	∞	∞	∞	0	∞	5	6	6
E	11	6	3	∞	0	3	15	10
F	17	12	2	5	3	0	11	9
G	4	9	19	6	15	11	0	7
H	3	8	7	6	10	9	7	0

[F][D]+[D][G]= 5+6 =11 < 21

Given a weighted graph, compute the length of the shortest paths between all pairs of vertices.

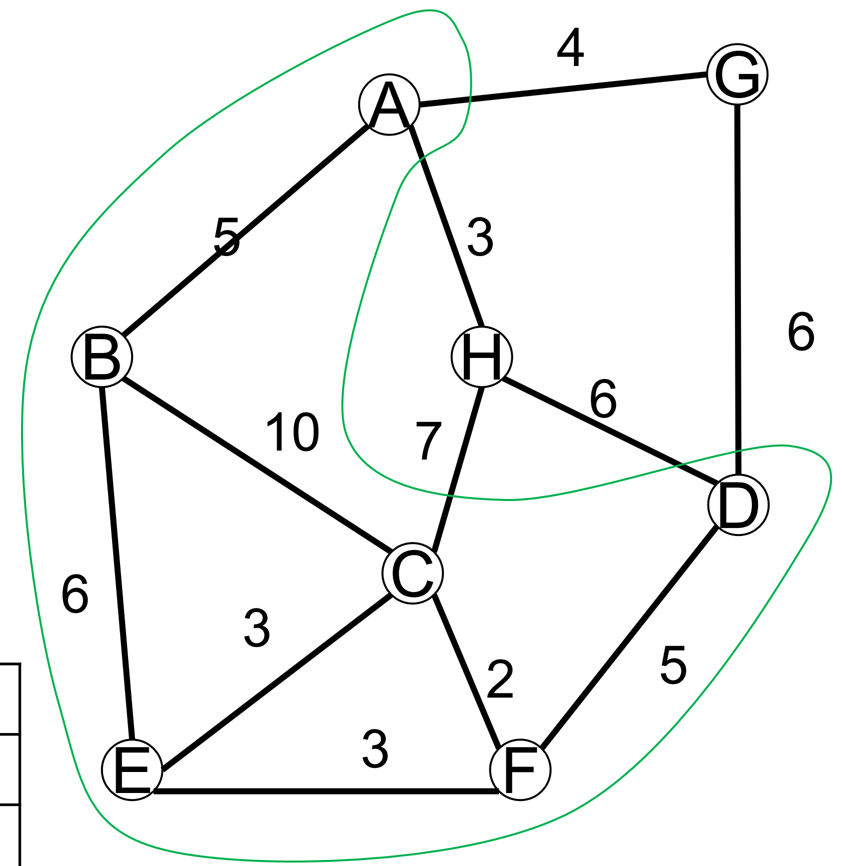


Augmented with all paths via:
{A, B, C, D, E}

	A	B	C	D	E	F	G	H
A	0	5	14	∞	11	14	4	3
B	5	0	9	∞	6	9	9	8
C	14	9	0	∞	3	2	19	7
D	∞	∞	∞	0	∞	5	6	6
E	11	6	3	∞	0	3	15	10
F	14	9	2	5	3	0	11	9
G	4	9	19	6	15	11	0	7
H	3	8	7	6	10	9	7	0

```
[A][E]+[E][C]= 11+3=14 < 15
[A][E]+[E][F]= 11+3=14 < 17
[B][E]+[E][C]= 6+3 = 9 < 10
[B][E]+[E][F]= 6+3 = 9 < 12
```

Given a weighted graph, compute the length of the shortest paths between all pairs of vertices.

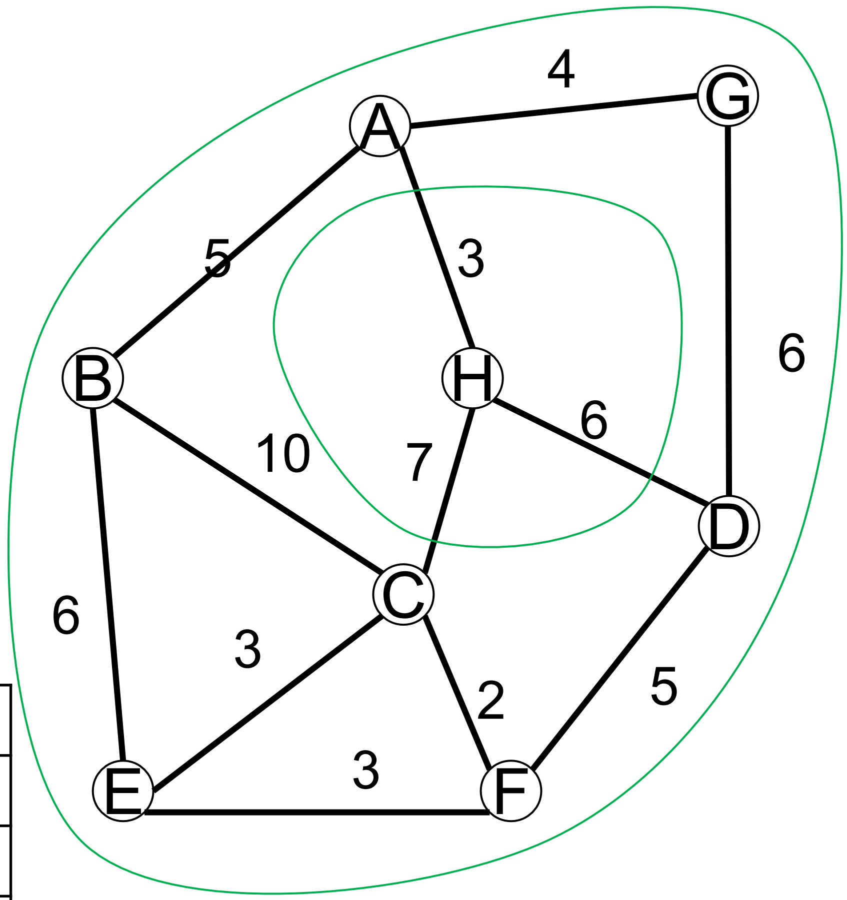


Augmented with all paths via:
{A, B, C, D, E, F}

	A	B	C	D	E	F	G	H
A	0	5	14	19	11	14	4	3
B	5	0	9	14	6	9	9	8
C	14	9	0	7	3	2	13	7
D	19	14	7	0	8	5	6	6
E	11	6	3	8	0	3	14	10
F	14	9	2	5	3	0	11	9
G	4	9	13	6	14	11	0	7
H	3	8	7	6	10	9	7	0

[A][F]+[F][D]= 14+5=19 < ∞
[B][F]+[F][D]= 9+5 =14 < ∞
[C][F]+[F][D]= 2+5 = 7 < ∞
[C][F]+[F][G]= 2+11=13 < 19
[D][F]+[F][E]= 5+3 = 8 < ∞
[E][F]+[F][G]= 3+11=14 < 15

Given a weighted graph, compute the length of the shortest paths between all pairs of vertices.

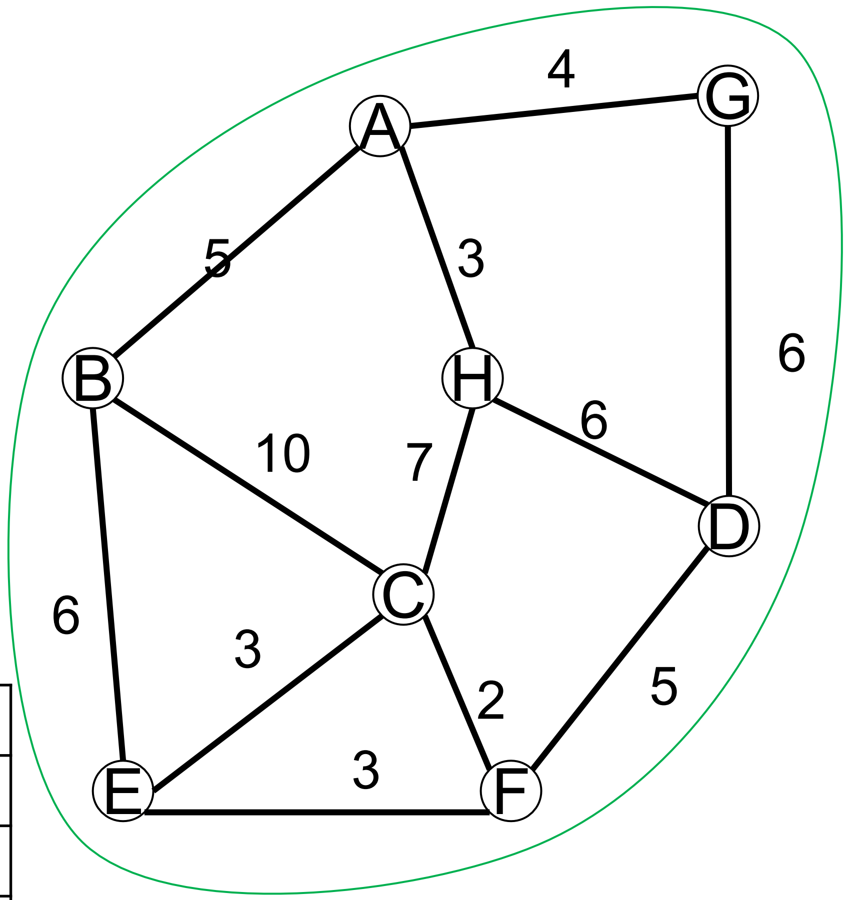


Augmented with all paths via:
{A, B, C, D, E, F, G}

	A	B	C	D	E	F	G	H
A	0	5	14	10	11	14	4	3
B	5	0	9	14	6	9	9	8
C	14	9	0	7	3	2	13	7
D	10	14	7	0	8	5	6	6
E	11	6	3	8	0	3	14	10
F	14	9	2	5	3	0	11	9
G	4	9	13	6	14	11	0	7
H	3	8	7	6	10	9	7	0

[A][G]+[G][D]= 4+6 =10 < 19

Given a weighted graph, compute the length of the shortest paths between all pairs of vertices.



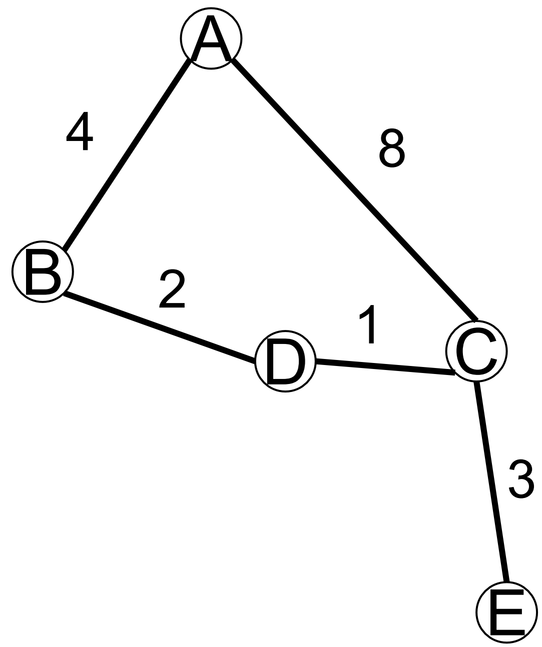
Augmented with all paths via:
{A, B, C, D, E, F, G,H}

	A	B	C	D	E	F	G	H
A	0	5	10	9	11	12	4	3
B	5	0	9	14	6	9	9	8
C	10	9	0	7	3	2	13	7
D	9	14	7	0	8	5	6	6
E	11	6	3	8	0	3	14	10
F	12	9	2	5	3	0	11	9
G	4	9	13	6	14	11	0	7
H	3	8	7	6	10	9	7	0

[A][H]+[H][C]= 3+7 =10 < 14
[A][H]+[H][D]= 3+6 = 9 < 10
[A][H]+[H][F]= 3+9 =12 < 14

H was last vertex to be added, so this is the final matrix

	A	B	C	D	E
A					
B					
C					
D					
E					



```python
def floydwarshall(self):

    allpairs = {}                    #create a dictionary, vertices as keys
    for v in self._structure:
        allpairs[v] = {}             #each value is a dictionary
        for w in self._structure:
            allpairs[v][w] = float('inf')
        allpairs[v][v] = 0

    for e in self.edges():
        (v,w) = e.vertices()
        allpairs[v][w] = e.element()
        allpairs[w][v] = e.element()

    for v in self._structure:
        for w in self._structure:
            for x in self._structure:
                if allpairs[w][x] > allpairs[w][v] + allpairs[v][x]:
                    allpairs[w][x] = allpairs[w][v] + allpairs[v][x]

    return allpairs
```

Complexity:
- initialising the 2D dictionary is $O(n^2)$
- adding the edge costs is $O(m)$, which is $O(n^2)$
- triple loop, n times round each loop: $O(n^3)$

So complete algorithm is $O(n^3)$

If graph is dense, this is lower complexity than repeated Dijkstra-heap (and same as Dijkstra-list, but tends to be faster)

If graph is sparse, this is higher complexity than repeated Dijkstra-heap, so don't use it for sparse graphs.

Final recommendation for all-pairs shortest path costs:

Sparse graph	**Dense graph**
$O(n^2 \log n)$ [Dijkstra-heap]	$O(n^3)$ [Floyd-Warshall]

Exercise:

Augment the algorithm to store, for each pair, the ‘intermediate’ vertex which last improved the cost of the path (i.e. what ‘v’ was being considered in the outer loop when the cost was last updated?).

How would you reconstruct the shortest path for any pair from the result?

Next lecture

Directed acyclic graphs
Topological Sort