Graphs

Contents

Basics

Definitions

Graph	A collection of nodes and edges
Node (or vertex)	A point on a graph
Edge (or arc)	A line segment connecting two nodes
Weighted graph (or network)	A graph with weights associated with its edges
Directed graph (or digraph)	A graph where the edges only go one way
Subgraph	A subset of the nodes and vertices from the original graph
Degree (or valency, or order)	The number of edges connected to a given vertex
Walk	A route through a graph, connecting vertices along edges
Path	A walk in which no *vertex* is visited more than once
Trail	A walk in which no *edge* is visited more than once
Eulerian circuit	A trail which starts where it ends and traverses every edge
Cycle	A walk which ends where it started, and no other vertex is visited more than once
Hamiltonian cycle	A cycle that includes every vertex
Connected (vertices)	There is a path between the vertices
Connected (graph)	All the vertices in the graph are connected
Loop	An edge that starts and finishes at the same vertex
Simple graph	A graph where there are no loops and there is at most one edge connecting any pair of vertices
Tree	A connected graph with no cycles
Spanning tree (of graph **G**)	A subgraph of **G** which contains all vertices and is a tree
Complete graph	A graph in which every vertex is directly connected by a single edge to all other vertices
Isomorphic graphs	Graphs which represent the same information but are drawn differently

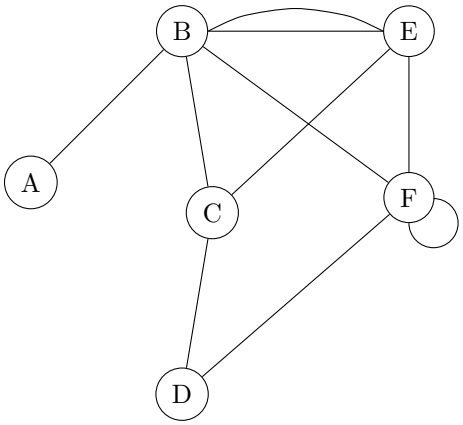
Matrices

A graph can be represented by an adjacency matrix, and a weighted graph can be represented by a distance matrix. The edge or distance *from* A *to* B is the number on row A and column B - read from the left.

Adjacency matrices

Each entry in an adjacency matrix describes the number of edges joining the corresponding nodes.

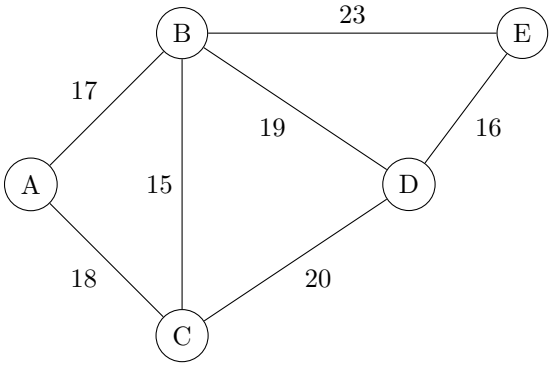
	A	B	C	D	E	F
A	0	1	0	0	0	0
B	1	0	1	0	2	1
C	0	1	0	1	1	0
D	0	0	1	0	0	1
E	0	2	1	0	0	1
F	0	1	0	1	1	2



Distance matrices

Each entry in a distance matrix describes the weight of the edge joining the corresponding nodes, if any.

	A	B	C	D	E
A	–	17	18	–	–
B	17	–	15	19	23
C	18	15	–	20	–
D	–	19	20	–	16
E	–	23	–	16	–



Minimum Spanning Trees

A minimum spanning tree (MST) is a subgraph with contains all nodes but no cycles, and has the minimum total length.

Throughout this MST topic, we will use the following graph for all our examples:

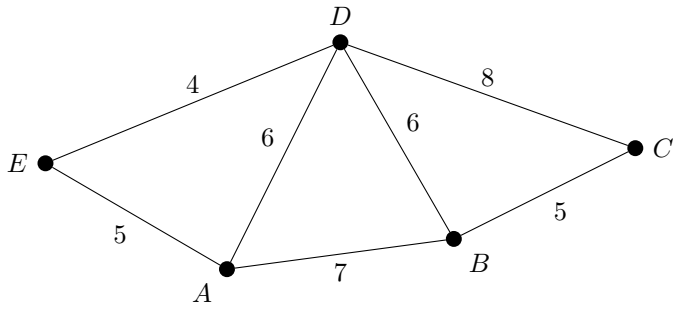


Figure 1: The graph **G**

	A	B	C	D	E
A	-	7	-	6	5
B	7	-	5	6	-
C	-	5	-	8	-
D	6	6	8	-	4
E	5	-	-	4	-

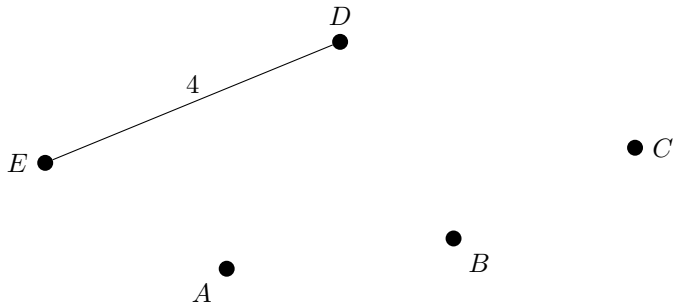
Figure 2: The distance matrix of **G**

Kruskal's algorithm

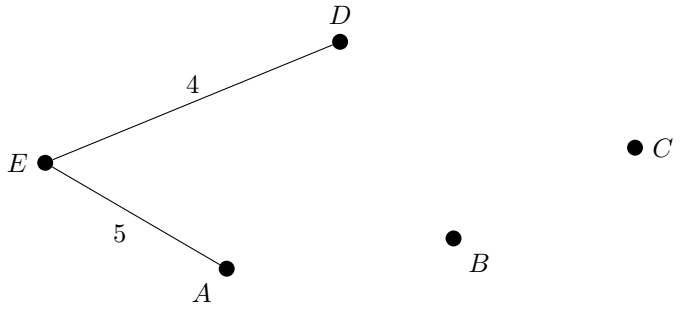
An algorithm to find an MST is Kruskal's algorithm. It goes as follows:

1. Sort all the edges into order of ascending weight
2. Select the edge of least weight to start the tree
3. Consider the next shortest edge. If it would form a cycle, then reject it, else add it to the tree
4. Repeat step 3 until all nodes have been added

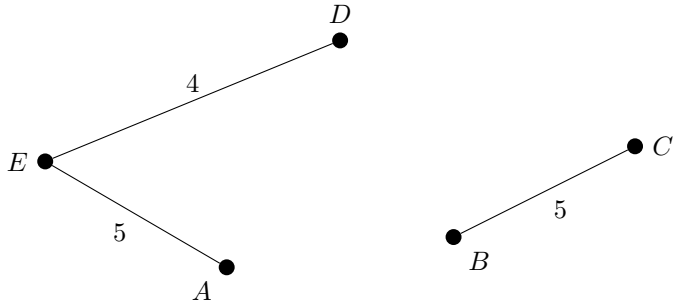
Using the graph **G** as our example, we order the edges to get $DE(4), AE(5), BC(5), AD(6), BD(6), AB(7), CD(8)$, so we start with DE.



AE won't form a cycle, so we can add it next.

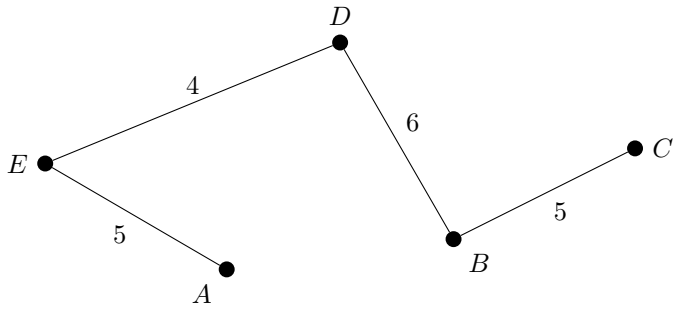


BC also won't form a cycle, so we can add it next.



We can see here that the subgraph at this stage is not connected. This is fine, since the final MST will be all connected.

AD **would** form a cycle, so we reject it. BD will not form a cycle, so we add it next.



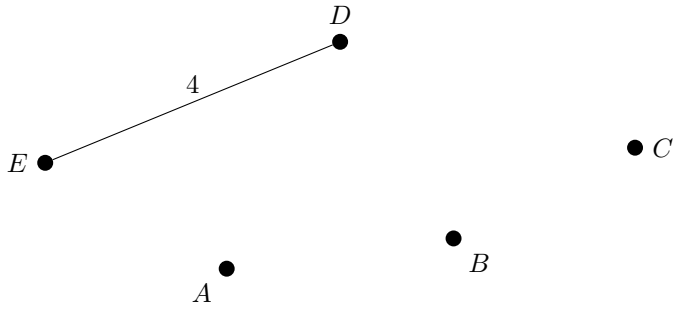
All the vertices are connected, so we've created an MST of **G**.

Prim's algorithm

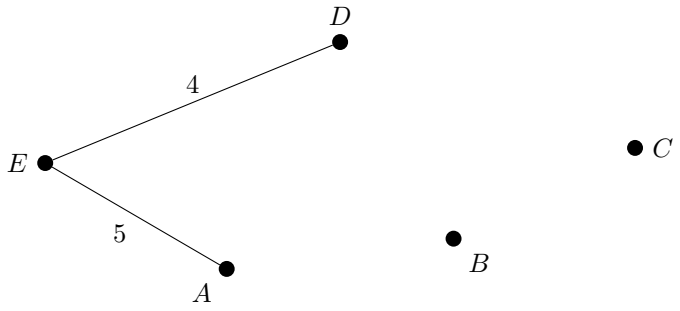
Another algorithm to find an MST is Prim's algorithm. It goes as follows:

1. Choose an arbitrary vertex to start the tree
2. Select the edge of least weight which joins a vertex in the tree to a vertex not yet in the tree
3. Repeat step 2 until all nodes have been added

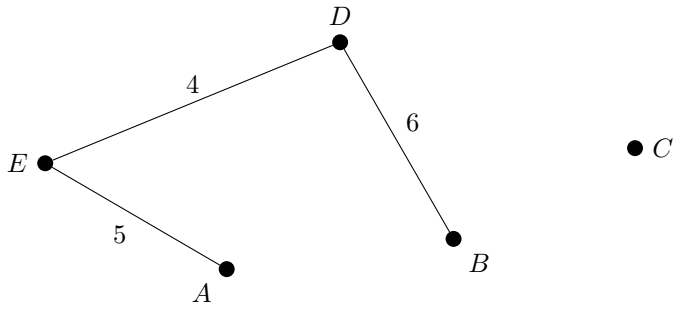
We will choose to start at D. The shortest edge from D is DE, so we will add it.



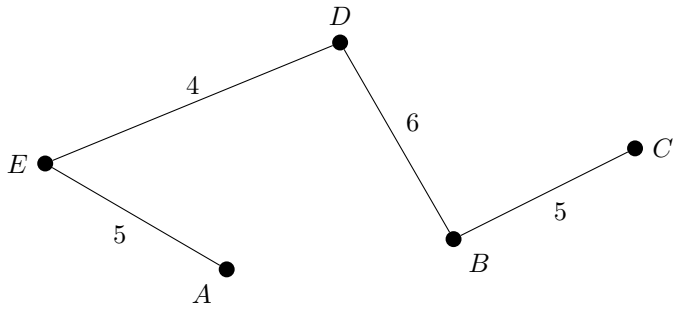
Now the shortest edge from the current subgraph is EA, so we will add it.



The shortest edge from the current subgraph is DA or DB but DA would create a cycle, so we will add DB.



Now the shortest edge from the current subgraph is BC, so we will add it.



Prim's algorithm on a distance matrix

To apply Prim's algorithm to a distance matrix, use the following algorithm:

1. Choose an arbitrary vertex to start the tree

2. Delete the **row** of the matrix for the chosen vertex
3. Number the **column** of the matrix for the chosen vertex
4. Put a ring around the smallest undeleted entry in the numbered columns
5. The ringed entry becomes the next edge to join the tree
6. Repeat steps 2 to 5 until all rows have been deleted

We will do our example on the distance matrix of graph **G** and start at D. We will first delete its row and column, and then circle the smallest number in the numbered columns.

				1	
	A	B	C	D	E
A	-	7	-	6	5
B	7	-	5	6	-
C	-	5	-	8	-
~~D~~	~~6~~	~~6~~	~~8~~	~~-~~	~~4~~
E	5	-	-	(4)	-

We will now delete the E row and find the smallest number in the columns.

				1	2
	A	B	C	D	E
A	-	7	-	6	(5)
B	7	-	5	6	-
C	-	5	-	8	-
~~D~~	~~6~~	~~6~~	~~8~~	~~-~~	~~4~~
~~E~~	~~5~~	~~-~~	~~-~~	~~(4)~~	~~-~~

Now we will delete the A row and find the smallest number in the columns.

	3			1	2
	A	B	C	D	E
~~A~~	~~-~~	~~7~~	~~-~~	~~6~~	~~(5)~~
B	7	-	5	(6)	-
C	-	5	-	8	-
~~D~~	~~6~~	~~6~~	~~8~~	~~-~~	~~4~~
~~E~~	~~5~~	~~-~~	~~-~~	~~(4)~~	~~-~~

Here, the smallest number in any of the numbered columns is the circled 6 above. When we look for the smallest number, we have to look at *all* the numbered columns.

We now delete the B row and, again, find the smallest number.

	3	4		1	2
	A	B	C	D	E
~~A~~	~~-~~	~~7~~	~~-~~	~~6~~	~~(5)~~
~~B~~	~~7~~	~~-~~	~~5~~	~~(6)~~	~~-~~
C	-	(5)	-	8	-
~~D~~	~~6~~	~~6~~	~~8~~	~~-~~	~~4~~
~~E~~	~~5~~	~~-~~	~~-~~	~~(4)~~	~~-~~

We can now delete the final row and then we're done. The numbers on top of the columns show what order we added edges in.

	3	4	5	1	2
	A	B	C	D	E
A	-	7	-	6	(5)
B	7	-	5	(6)	-
C	-	(5)	-	8	-
D	6	6	8	-	4
E	5	-	-	(4)	-

Shortest Path

Throughout this shortest path topic, we will use the following graph for all our examples:

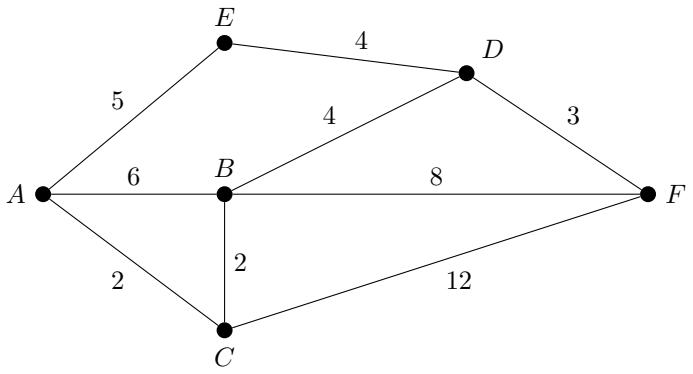


Figure 3: The graph **G**

	A	B	C	D	E	F
A	-	6	2	-	5	-
B	6	-	2	4	-	8
C	2	2	-	-	-	12
D	-	4	-	-	4	3
E	5	-	-	4	-	-
F	-	8	12	3	-	-

Figure 4: The distance matrix of **G**

Dijkstra's algorithm

Dijkstra's algorithm can be used to find the shortest path between two chosen vertices in a network.

Dijkstra uses boxes like this in place of nodes to make the working clear:

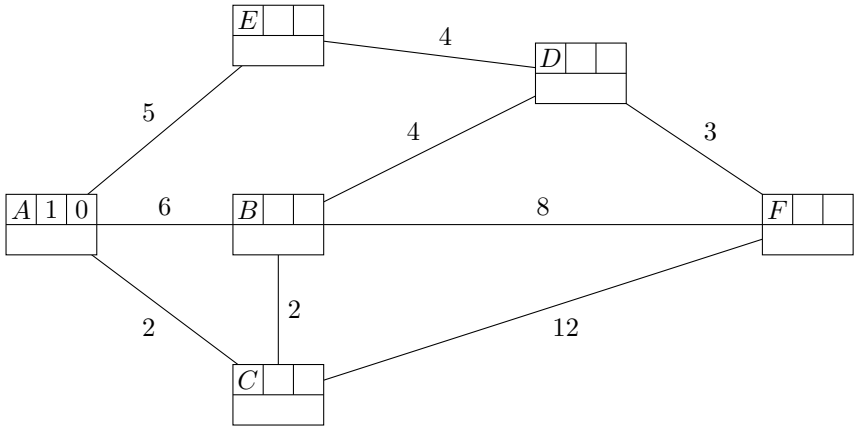
Vertex name	Order of labelling	Final label
Working values		

To find the shortest path from S to T, use the following algorithm:

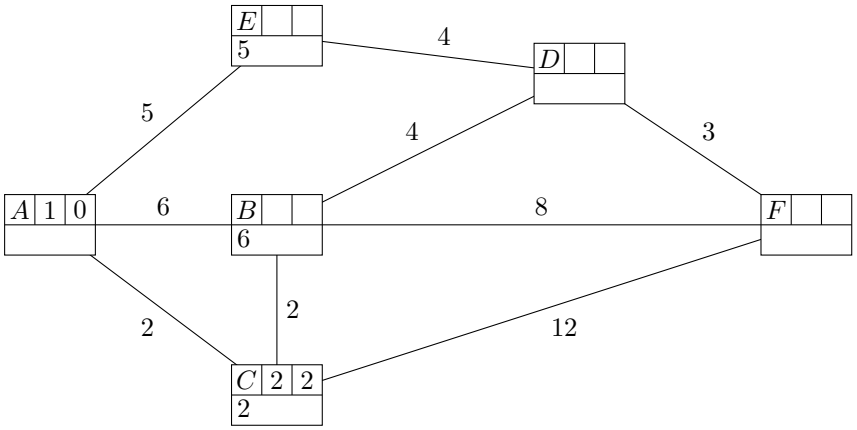
1. Label the start vertex, S with the final label 0
2. The vertex which just received its final label is X. For every unlabelled vertex Y which is connected to X:
   (a) The new working value of Y is the final label of X plus the distance between X and Y
   (b) But only replace Y's working value if the new one is smaller
3. Look at all vertices without final labels. Choose the one with the smallest working value. This becomes the final label for this vertex
4. Repeat steps 2 and 3 until the destination vertex T receives a final label
5. To find the shortest path, trace back from T to S. Include an edge only when its weight is the difference of the final labels of the vertices on each end of it

As an example, lets find the shortest path from A to F on graph **G** using Dijkstra's algorithm.

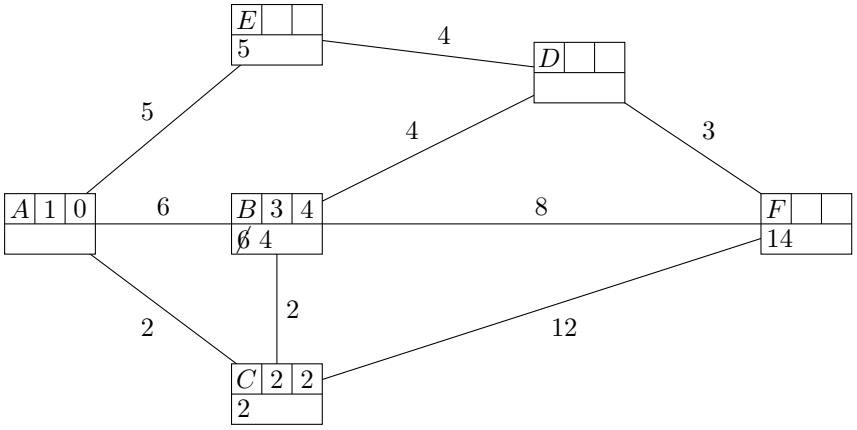
We first redraw the graph with the boxes and fill in box A with a final label of 0. This is the first box we labelled, so it gets an order number of 1.



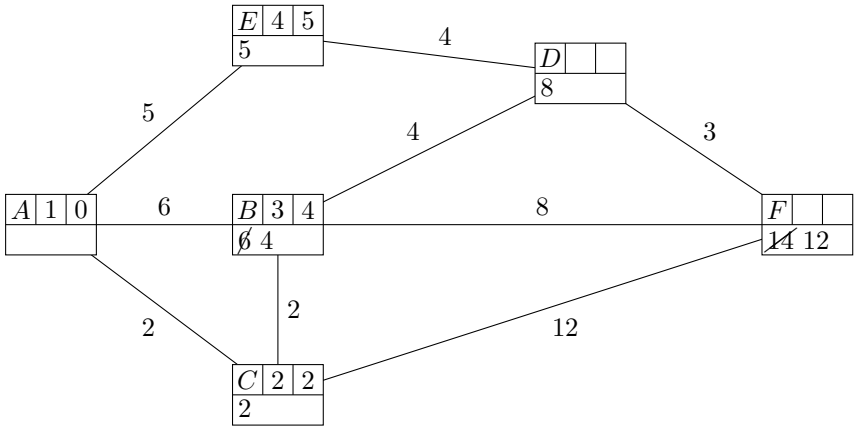
We now fill in the working values for all visit-able nodes and thus choose C as the next node to visit, so it gets a final label.



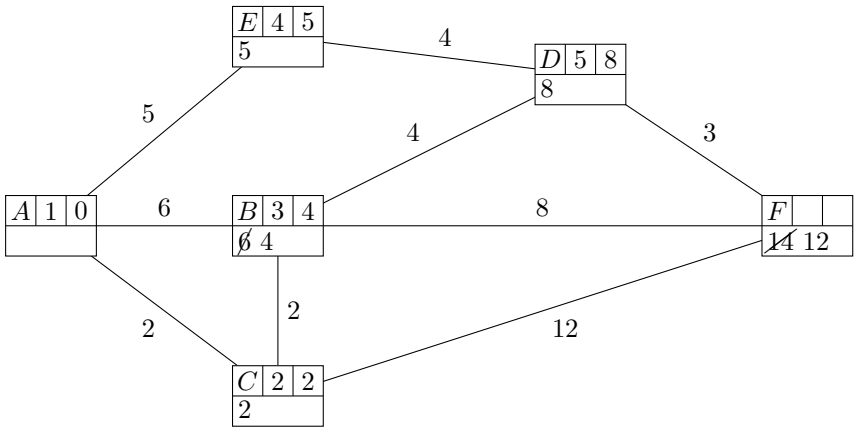
We now repeat that process and choose B as the next node to visit, so it gets a final label.



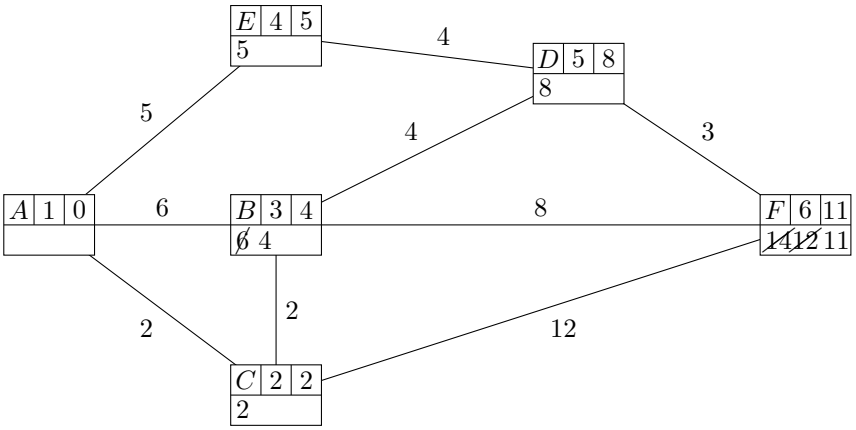
We then repeat that process once again and choose E as the next node.



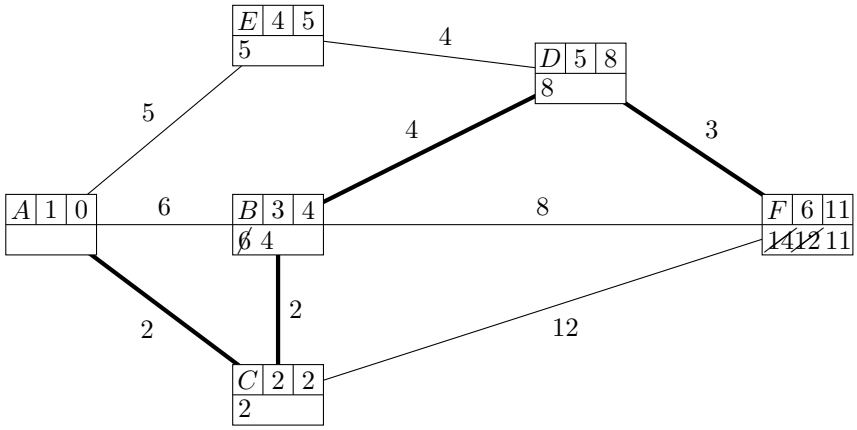
We choose D next, since E offers no new lengths.



And again.



The target node F now has a final label, so we have finished the algorithm. The shortest distance from A to F is 11 and to find the path, we backtrack from F to A.



The final labels represent the distance between the start vertex and that labelled vertex, and you can use the backtracking method to find the shortest path between the start node and any vertex with a final label. To find the shortest path between a different pair of vertices, you'll have to do Dijkstra's again. Or you could use Floyd's.

Floyd's algorithm

Floyd's algorithm can be used to find the shortest path between every pair of vertices in a network. It produces two tables - the first shows the minimum distance between any two vertices, and the second can be used to find the route.

1. Complete an initial **distance table** for the network. If there is no direct route from one vertex to another, label the distance with ∞
2. Complete an initial **route table** by making every entry in a column the same as the label at the top of the column
3. For the first iteration, copy the values in the first row and column in the tables into a new pair of tables. Shade these values
4. Consider each unshaded position in turn:
   (a) Compare the value in this position in the previous table to the sum of the corresponding shaded values. Choose the smaller of the two options
   (b) If you chose the sum, then copy the letter of this vertex into the corresponding position in the route table
   (c) Repeat until all unshaded cells have been filled in
5. For the second iteration, copy the second row and column from the previous tables into a new pair of tables and shade them. Repeat step 4
6. Continue iterating like this until you've done every row and column

We will use graph **G** as an example to perform Floyd's algorithm on, with the distance table on the left and the route table on the right.

	A	B	C	D	E	F
A	∞	6	2	∞	5	∞
B	6	∞	2	4	∞	8
C	2	2	∞	∞	∞	12
D	∞	4	∞	∞	4	3
E	5	∞	∞	4	∞	∞
F	∞	8	12	3	∞	∞

	A	B	C	D	E	F
A	A	B	C	D	E	F
B	A	B	C	D	E	F
C	A	B	C	D	E	F
D	A	B	C	D	E	F
E	A	B	C	D	E	F
F	A	B	C	D	E	F

For the first iteration, we will copy the A row and column to a new distance table and shade them, and then perform step 4. Changed values are highlighted in red. This is the result after the first iteration:

	A	B	C	D	E	F
A	∞	6	2	∞	5	∞
B	6	12	2	4	11	8
C	2	2	4	∞	7	12
D	∞	4	∞	∞	4	3
E	5	11	7	4	10	∞
F	∞	8	12	3	∞	∞

	A	B	C	D	E	F
A	A	B	C	D	E	F
B	A	A	C	D	A	F
C	A	B	A	D	A	F
D	A	B	C	D	E	F
E	A	A	A	D	A	F
F	A	B	C	D	E	F

And here's the result after the second iteration:

	A	B	C	D	E	F
A	12	6	2	10	5	14
B	6	12	2	4	11	8
C	2	2	4	6	7	10
D	10	4	6	8	4	3
E	5	11	7	4	10	19
F	14	8	10	3	19	16

	A	B	C	D	E	F
A	B	B	C	B	E	B
B	A	A	C	D	A	F
C	A	B	A	B	A	B
D	B	B	B	B	E	F
E	A	A	A	D	A	B
F	B	B	B	D	B	B

And here's the result after the third iteration:

	A	B	C	D	E	F
A	4	4	2	8	5	12
B	4	4	2	4	9	8
C	2	2	4	6	7	10
D	8	4	6	8	4	3
E	5	9	7	4	10	17
F	12	8	10	3	17	16

	A	B	C	D	E	F
A	C	C	C	C	E	C
B	C	C	C	D	C	F
C	A	B	A	B	A	B
D	C	B	B	B	E	F
E	A	C	A	D	A	C
F	C	B	B	D	C	B

And here's the result after the fourth iteration:

	A	B	C	D	E	F
A	4	4	2	8	5	11
B	4	4	2	4	8	7
C	2	2	4	6	7	9
D	8	4	6	8	4	3
E	5	8	7	4	8	7
F	11	7	9	3	7	6

	A	B	C	D	E	F
A	C	C	C	C	E	D
B	C	C	C	D	D	D
C	A	B	A	B	A	D
D	C	B	B	B	E	F
E	A	D	A	D	D	D
F	D	D	D	D	D	D

And here's the result after the fifth iteration (nothing changed):

	A	B	C	D	E	F
A	4	4	2	8	5	11
B	4	4	2	4	8	7
C	2	2	4	6	7	9
D	8	4	6	8	4	3
E	5	8	7	4	8	7
F	11	7	9	3	7	6

	A	B	C	D	E	F
A	C	C	C	C	E	D
B	C	C	C	D	D	D
C	A	B	A	B	A	D
D	C	B	B	B	E	F
E	A	D	A	D	D	D
F	D	D	D	D	D	D

And here's the result after the sixth iteration:

	A	B	C	D	E	F
A	4	4	2	8	5	11
B	4	4	2	4	8	7
C	2	2	4	6	7	9
D	8	4	6	6	4	3
E	5	8	7	4	8	7
F	11	7	9	3	7	6

	A	B	C	D	E	F
A	C	C	C	C	E	D
B	C	C	C	D	D	D
C	A	B	A	B	A	D
D	C	B	B	F	E	F
E	A	D	A	D	D	D
F	D	D	D	D	D	D

From these tables, we can see that the shortest path between A and F has a length of 11, and it goes like this (using $\mapsto$ for an indirect connection and $\to$ for a direct one):

$$A \mapsto F$$
$$A \mapsto D \mapsto F$$
$$A \mapsto C \mapsto D \to F$$
$$A \to C \mapsto B \mapsto D \to F$$
$$A \to C \to B \to D \to F$$

This is the same path and length we got using Dijkstra's, so we can assume it's correct.

When resolving a path, find the start node on the left and the target node on the top. The connection goes through the corresponding node in the table. If this is the target node, then the connection is direct. If it's a different node, then the path goes through that one. Continue resolving indirect connections until all connections in the path are direct.