AQA Computer Science A-Level

4.3.6 Optimisation algorithm

Intermediate Notes

Specification:

4.3.6.1 Dijkstra’s shortest path algorithm

Understand and be able to trace Dijkstra’s shortest path algorithm

Be aware of applications of shortest path algorithm

Optimisation Algorithms

An optimisation algorithm finds the best possible solution to the problem posed. The only one you must be aware of is Dijkstra's (pronounced dyke-struh's) algorithm, which finds the shortest path from a starting node to every other node in a network.

These points may be modelled as nodes in a weighted graph. You will not be asked to recall the steps of the algorithm in an exam, although you may be asked to identify it and state its purpose and/or trace the code.

Algorithm

An **algorithm** is a finite **sequence of instructions** that can be followed to complete a task and that **always terminates**.

Synoptic Link

Graphs can be used as **visual representations** of **complex relationships**. **Weighted Graphs** have values assigned to each **edge**.

Graphs are covered in **Graphs** under **Fundamentals of Data Structures**.

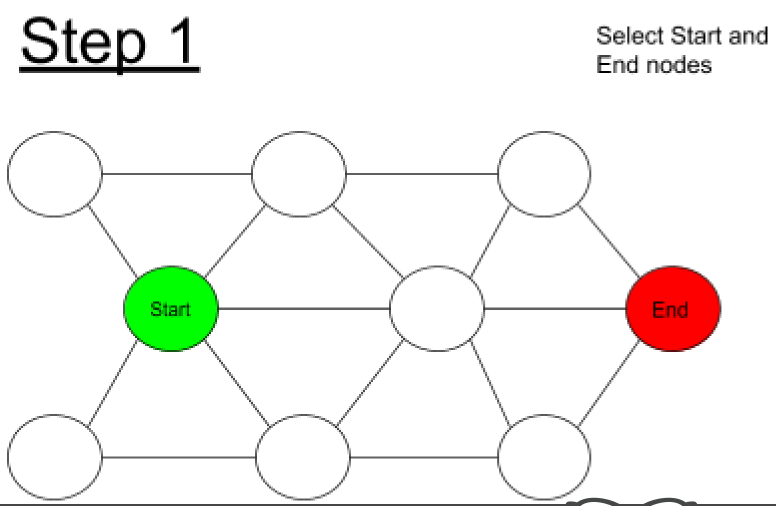
Dijkstra's Algorithm

As mentioned above, Dijkstra's algorithm is used to find the shortest path between two nodes in a graph. If you take maths for A level, you may have already come across this algorithm.

However, in maths Dijkstra you will normally only be asked for the finished path, whereas computer science Dijkstra will require you to have a full understanding of the code, often leading to filling in a dry run table.

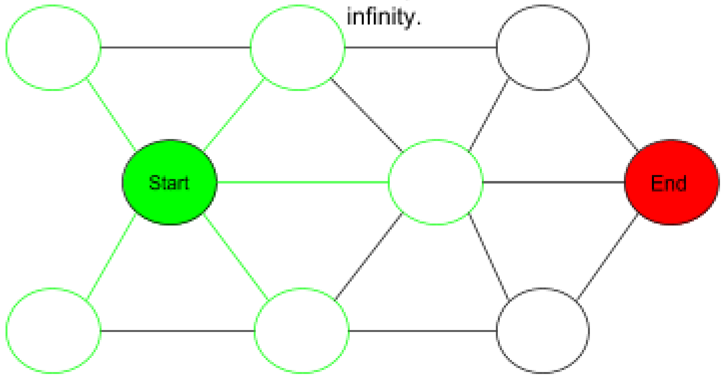
Satellite navigation systems use Dijkstra's algorithm to find the shortest route between two geographical destinations.

Dijkstra's Algorithm Overview:



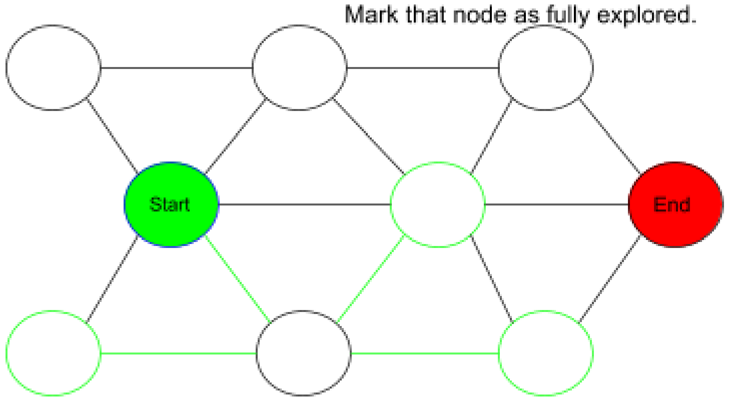
Step 2

Make note of the distances of the nodes from the start position. At this point, only the nodes connected to start will have a value, the others will have a distance of infinity.



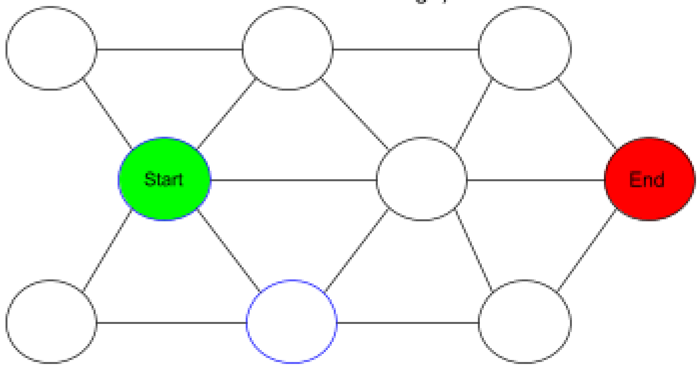
Step 3

Note the start node as fully explored. Choose the node closest to the start node, and update the table to reflect the shortest distance from A to each node. Mark that node as fully explored.



Step 4

Repeat step 3, always selecting the node the shortest distance from A which hasn't been fully explored. Continue until each node has been fully explored (and hence each edge).



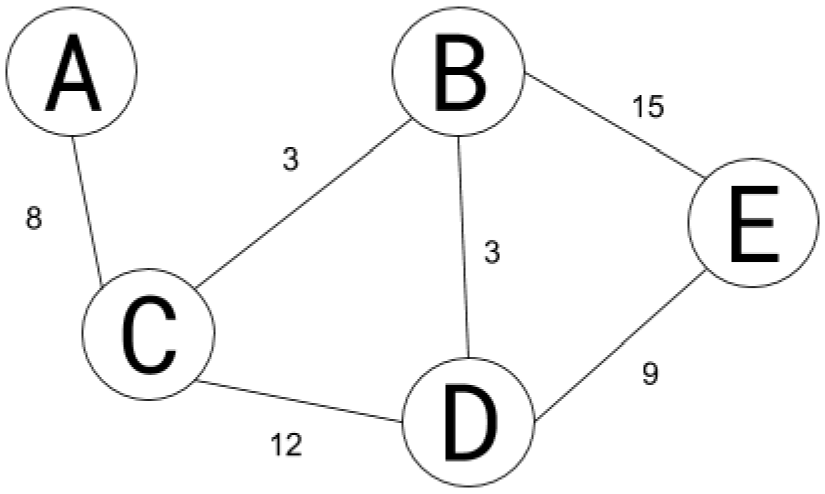
Dijkstra's Algorithm Example:

Here is a graph with nodes A to E. The numbers represent the time in minutes it would take travel along each edge.

Our task is to get from A to E in the shortest amount of time. The first step is to set up a table containing all the vertices.

Edge

Edges / arcs are the connections between each node / vertex.



Starting from A, we list all the connections to other nodes. Where A is not directly connected to the node, we write ∞ because there is effectively no way to get to them - it would take an infinite time to reach them.

V	A	B	C	D	E
A	0_A	∞_A	8_A	∞_A	∞_A

The subscript A next to each value tells us that the weight is relative to A. This comes in useful at the end of the algorithm, when we trace back our steps to find the route that we took through the network.

Node A has now been inspected, we'll identify this in the table by shading it yellow.

The next step is to select the node in the table with the lowest weight which hasn't yet been inspected. In this case, that is C with a weight of 8. We add a row in the table for C and can now begin to fill in weights for C.

V	A	B	C	D	E
A	0_A	∞_A	8_A	∞_A	∞_A
C					

We now look at the graph for the new weights and add them to the table if they are less than the value currently under the node when added to the shortest weight to the chosen node (8 in this case).

There is no connection from A to C with a value of less than 0, so we leave A's weight as 0 from A.

The connection from C to B has a weight of 3, which leaves a total weight of 11 when added to our current weight to E (8). The current value of infinity is higher, so we replace it with 11 from C.

Moving on to C, we cannot include connections from one node to itself so we leave the value as 8 from A.

Now at D, there is a connection of weight 12 to C. Adding this to our current weight for C (8) gives us 20. 20 is lower than infinity so we place 20 in the table, and mark it as from C.

Finally, C to E is left as infinity as there is no direct connection between these nodes.

We can now mark C as visited. Our table now looks like this:

V	A	B	C	D	E
A	0_A	∞_A	8_A	∞_A	∞_A
C	0_A	11_C	8_A	20_C	∞_A

Again, we select the node in the table with the lowest weight which hasn't yet been inspected. In this case, that is B with a weight of 11. We add a row in the table for B and can now begin to fill in weights for B.

V	A	B	C	D	E
A	0_A	∞_A	8_A	∞_A	∞_A
C	0_A	11_C	8_A	20_C	∞_A
B					

We now look at the graph for the new weights and add them to the table if they are less than the value currently under the node when added to the shortest weight to the chosen node (11 in this case).

There is no connection from B to A so we leave this as 0 from A.

Looking at node B, we cannot include connections from one node to itself so we leave the value as 11 from C.

Moving on to C, B and C are connected with a weight of 3. Adding this to the 11 we have already used on our way to B, we get 14 which is higher than the current 8. We leave the value as 8.

Now at D, there is a connection of weight 3 to B. Adding this to our current weight for B (11) gives us 14. 14 is lower than 20 so we place 14 in the table, and mark it as from B.

Finally, B and E are connected with a weight of 15. Adding this to 11 gives us 26. This is lower than the current value of infinity so we place 26 in the table and mark it as from B.

We can now mark B as inspected. Our table now looks like this:

V	A	B	C	D	E
A	0_A	∞_A	8_A	∞_A	∞_A
C	0_A	11_C	8_A	20_C	∞_A
B	0_A	11_C	8_A	14_B	26_B

Once again we select the node in the table with the lowest weight which hasn't yet been inspected. In this case, that is D with a weight of 14. We add a row in the table for D and can now begin to fill in weights for D.

V	A	B	C	D	E
A	0_A	∞_A	8_A	∞_A	∞_A
C	0_A	11_C	8_A	20_C	∞_A
B	0_A	11_C	8_A	14_B	26_B
D					

We now look at the graph for the new weights and add them to the table if they are less than the value currently under the node when added to the shortest weight to the chosen node (14 in this case). There is no connection from D to A so we leave this as 0 from A.

Looking at node B, there is a connection between D and B with weight 3. Adding this to 14 gives us 17 which is greater than the current weight of 11. We leave the table as it is.

Moving on to C, D and C are connected with a weight of 12. Adding this to the 14 we have already used on our way to B, we get 26 which is higher than the current 8. We leave the value as 8.

Now at D, we cannot consider connections from one node to itself so we move on.

Finally, D and E are connected with a weight of 9. Adding this to 14 gives 23 which is less than the 26 currently in place. We place 23 in the table., marking it as from D.

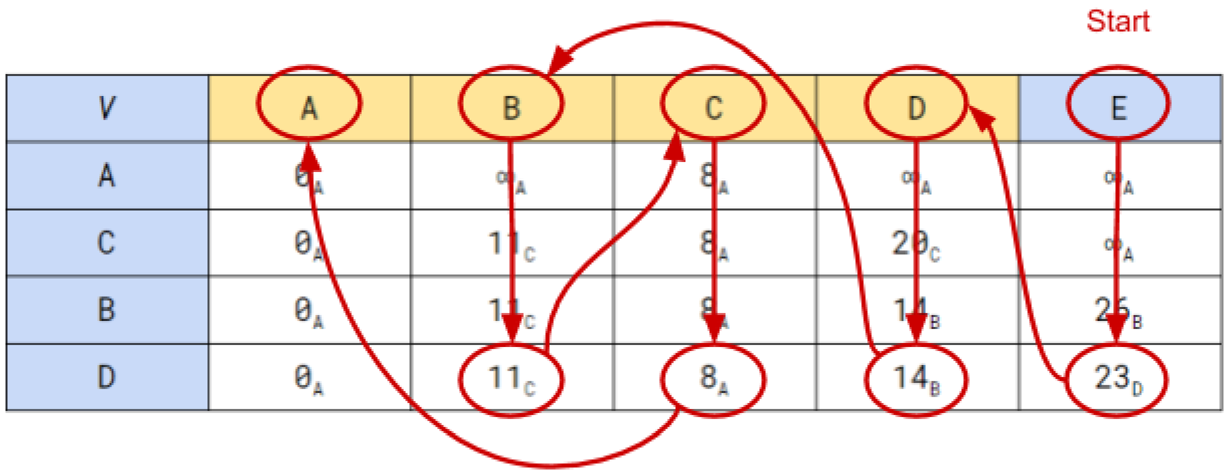
We can now mark D as inspected. Our table now looks like this:

V	A	B	C	D	E
A	0_A	∞_A	8_A	∞_A	∞_A
C	0_A	11_C	8_A	20_C	∞_A
B	0_A	11_C	8_A	14_B	26_B

D	0_A	11_C	8_A	14_B	23_D

We now know that the shortest possible path between the nodes A and E has a weight of 23. In order to find the path that we took, we make use of the subscript letters.

Start from the finishing node, E. The value of 23 at E is marked as from D. The final value in D’s column is 14 from B. Looking at B’s column, we see that it is 11 from C. Finally, C is marked as 8 from A.



V	A	B	C	D	E
A	0_A	∞_A	8_A	∞_A	∞_A
C	0_A	11_C	8_A	20_C	∞_A
B	0_A	11_C	8_A	14_B	25_B
D	0_A	11_C	8_A	14_B	23_D

This process, which the diagram above represents, results in the nodes E, D, B, C, A. Reverse this order to get the shortest path through the network.

The shortest path through the network from A to E is A, C, B, D, E and has a weight of 23 minutes.