

## Pathfinding Algorithms

Isaac Student Booster

**DO NOW:**

Think: how did you work out the quickest route to school or college today?

Would that change if the traffic was busy, or the bus broke down?

What would the algorithm look like for this problem?

---

---

---

---

---


---

---

---

## Learning Objectives

- Understand how the Dijkstra and A\* pathfinding algorithms work, including the use of heuristics
- Use Dijkstra and A\* on a given graph to find the shortest path
- Understand the benefits and drawbacks of A\* over Dijkstra




---

---

---

---

---


---

---

---

## From the OCR A-level Spec

- Candidates need to understand how the Dijkstra and A\* shortest path algorithms work (2.3.1)
- For both Dijkstra and A\* they need to be able to:
  - calculate the shortest path in a graph or tree
  - read and trace code – **not covered today**
- Candidates should apply their knowledge of heuristics to solve problems (2.2.2)




---

---

---

---

---

---

---

---

## From the AQA A-level Spec

- Understand and be able to trace Dijkstra's shortest path algorithm (4.3)
- Be aware of applications of Dijkstra's shortest path algorithm (4.3)
- Understand heuristic methods are often used when tackling intractable problems (4.4)

Students will not be expected to recall the steps in Dijkstra's algorithm, and don't need to know the A\* algorithm, although students must understand heuristics.



## Shortest Path

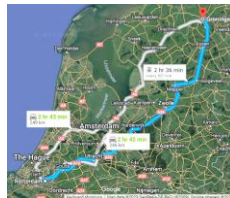
Many problems require us to find the **shortest path** between a set of points:

- satellite navigation
- internet packet routing
- finding the shortest length of wire needed to connect pins on a circuit board



## Dijkstra's shortest path algorithm

Invented by Dutch computer scientist Edsger Dijkstra in 1956 to calculate the shortest route from Rotterdam to Gröningen.



"In computing, elegance is not a dispensable luxury but a quality that decides between success and failure"

Edsger Dijkstra




---

---

---

---

---

---

---

---

## Edsger Dijkstra

Dijkstra was a computing pioneer, contributing to the development of ALGOL-60, which influenced Pascal and C and hence many modern languages.

He was a colourful character, rarely using a computer in his academic work and assessed his students through hours-long interviews instead of papers or exams!




---

---

---

---

---

---

---

---

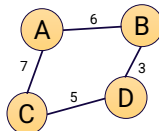
## The graph data structure

Pathfinding algorithms process data held in a **weighted graph** data structure.

**A graph consists of nodes** (also called vertices)

Nodes are connected by **edges**.  
If these edges can be travelled in both directions, this is an **undirected graph**.

In a **weighted graph**, edges carry a **weight** which could be the distance between cities, or the delay (latency) between devices.




---

---

---

---

---

---

---

---

## Dijkstra's Algorithm




---

---

---

---

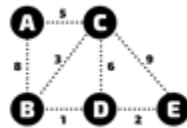
---

---

---

---

### Dijkstra's Algorithm



- Dijkstra's algorithm has one motivation: to find the set of **shortest paths** from a start node to each of the other nodes on the graph.
- The **cost** of a path that connects two nodes is calculated by adding the weights of all the edges that belong to the path.
- The **shortest path** is the sequence of nodes, in the order they are visited, which results in the **minimum cost** to travel between the start node and a given node.




---

---

---

---

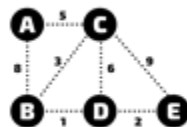
---

---

---

---

### Dijkstra's Algorithm



When the algorithm has finished running, it produces a list that holds the following information for **each node**:

- The **node** label
- The **cost** of the shortest path to **that node** (from the start node)
- The **previous** node's label, in the shortest path

You can then backtrack through the previous nodes to return the **shortest path and its cost**.




---

---

---

---

---

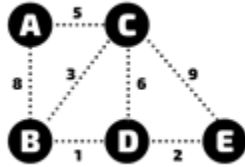
---

---

---

## Worked Example – Dijkstra

We will use Dijkstra now to find the shortest path from A to every other node in this graph:



### Step 1 – create the list

Create a table with headings: **node**, **cost** and **previous node**.

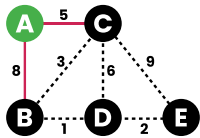
Add all nodes to the list, giving start node (A) a **cost of zero**, all other costs are **infinity ( $\infty$ )**

Set all **previous nodes** to **none**.

Node	Cost (from start)	Previous
A	0	none
B	$\infty$	none
C	$\infty$	none
D	$\infty$	none
E	$\infty$	none

### Step 2a – choose lowest

Make the lowest cost unvisited node the **current node**.

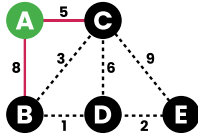


Examine nodes that can be visited from A, which are **B and C**.

Node	Cost (from start)	Previous
A	0	none
B	$\infty$	none
C	$\infty$	none
D	$\infty$	none
E	$\infty$	none

## Step 2b – process node

The cost of going from A to B is 8, while A to C is 5.



Update the list with these costs and previous node A.

Node	Cost (from start)	Previous
A	0	none
B	8	A
C	5	A
D	$\infty$	none
E	$\infty$	none

---

---

---

---

---

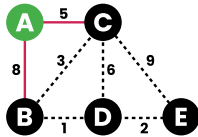
---

---

---

## Step 2c – mark node visited

You have now evaluated all routes from the current node



Mark node A **visited**.

Node	Cost (from start)	Previous
	0	none
B	8	A
C	5	A
D	$\infty$	none
E	$\infty$	none

---

---

---

---

---

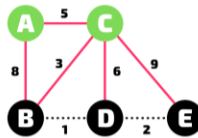
---

---

---

## Step 3a – choose lowest again

Again, make the lowest unvisited node the **current node**.



C has the lowest cost. Now evaluate its neighbours...

Node	Cost (from start)	Previous
	0	none
B	8	A
C	5	A
D	$\infty$	none
E	$\infty$	none

---

---

---

---

---

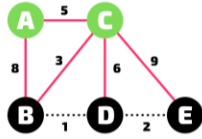
---

---

---

### Step 3b – process node

C costs 5 so we add this to the cost **from C to its neighbours**



C-B costs  $5 + 3 = 8$   
 C-D costs  $5 + 6 = 11$   
 C-E costs  $5 + 9 = 14$

Node	Cost (from start)	Previous
A	0	none
B	8	A
C	5	A
D	$\infty$ 11	<del>none</del> C
E	$\infty$ 14	<del>none</del> C

---

---

---

---

---

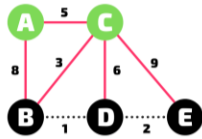
---

---

---

### Step 3b – mark node visited

You have now evaluated all routes from the current node



Mark node C **visited**.

Node	Cost (from start)	Previous
A	0	none
B	8	A
C	5	A
D	$\infty$ 11	<del>none</del> C
E	$\infty$ 14	<del>none</del> C

---

---

---

---

---

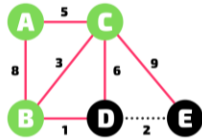
---

---

---

### Step 4a – choose lowest again

Again, make the lowest unvisited node the current node.



**B** has the lowest cost.  
 Now evaluate its neighbours...

Node	Cost (from start)	Previous
A	0	none
B	8	A
C	5	A
D	11	C
E	14	C

---

---

---

---

---

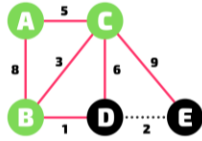
---

---

---

### Step 4b – process node

B costs 8 so we add this to the cost **from B to its neighbours**



Node	Cost (from start)	Previous
A	0	none
B	8	A
C	5	A
D	14	B
E	14	C

D costs  $8 + 1 = 9$

We don't revisit A and C




---

---

---

---

---

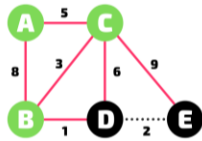
---

---

---

### Step 4c – mark node visited

You have now evaluated all routes from the current node



Node	Cost (from start)	Previous
A	0	none
B	8	A
C	5	A
D	9	B
E	14	C

Mark node B **visited**.




---

---

---

---

---

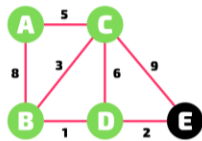
---

---

---

### Step 5a – get lowest yet again!

Again, make the lowest unvisited node the current node



Node	Cost (from start)	Previous
A	0	none
B	8	A
C	5	A
D	9	B
E	14	C

D is the lowest cost unvisited node. We evaluate its neighbours...




---

---

---

---

---

---

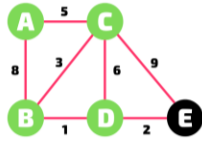
---

---



## Step 5b – process node

D costs 9, so we add this to the cost from D to its neighbours...



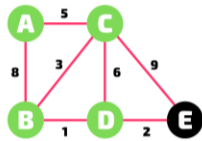
E costs  $9 + 2 = 11$

We update E in the table with 11 and D:

Node	Cost (from start)	Previous
A	0	none
B	8	A
C	5	A
D	9	B
E	11	D

## Step 5c – mark visited

You have now evaluated all routes from the current node

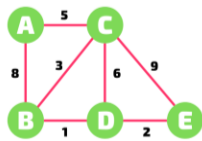


Mark node D **visited**.

Node	Cost (from start)	Previous
A	0	none
B	8	A
C	5	A
D	9	B
E	11	D

## Step 6 – and again!

Only E remains, which has no unvisited neighbours.

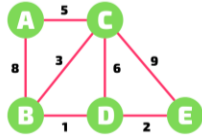


So we can just mark node E visited.

Node	Cost (from start)	Previous
A	0	none
B	8	A
C	5	A
D	9	B
E	11	D

## The final table

All nodes have now been visited...



Node	Cost (from start)	Previous
A	0	none
B	8	A
C	5	A
D	9	B
E	11	D

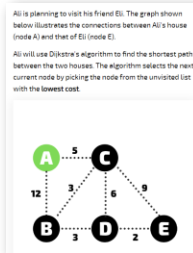
We can work backward from any node to find the shortest path from A e.g. **E's previous node is D** etc. The shortest path to E is thus **ABDE** costing **11**

## Activity 1 – Isaac Questions

Open [bit.ly/iapath23](https://bit.ly/iapath23) and answer the two questions.

You will need to sign in to **Isaac Computer Science** or register for a free account if not done already.

The questions are provided on **Handout 1** if this is not possible.

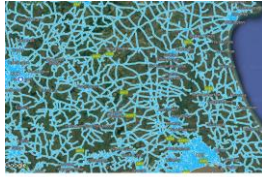


## A\* Algorithm

Dijkstra plus *heuristics*

## The problem with Dijkstra

This is just a small segment of Google Maps.  
How long would Dijkstra's algorithm take to traverse this?



Dijkstra's algorithm becomes **intractable** on large graphs. How can we change the algorithm to make the problem **tractable** again?




---

---

---

---

---

---

---

---

## The solution - heuristics

Heuristics help obtain **good enough** solutions where a perfect solution would take too long (or is **intractable**). In this diagram we can see the **Manhattan distance** (red line) and **Euclidean distance** (green line) which give estimates of the remaining distance.

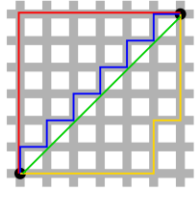


Image by Psychonaut via wikipedia




---

---

---

---

---

---

---

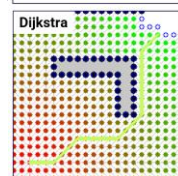
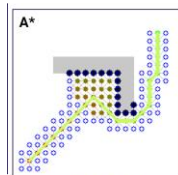
---

## Heuristics: Pros and Cons

Using heuristics trades accuracy for speed.

A heuristic algorithm chooses paths that are *likely* to yield a good result (but we *may* thus discount the optimal path!)

Compare these two animations, what do you notice?



From Wikipedia by Subh3 - Own work, CC BY 3.0




---

---

---

---

---

---

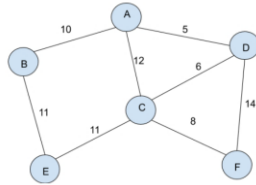
---

---

## Worked Example – A\*

We will use A\* now to find the shortest path from A to F in this diagram.

(Note that A\* cannot cost all routes to all nodes, only a route to a chosen destination)




---

---

---

---

---

---

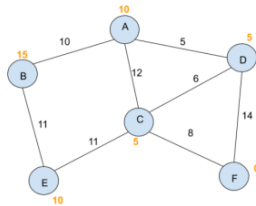
---

---

## Adding the heuristics

First, we need to add heuristics. In this example they are provided for you.

You may need to calculate them yourself (e.g. using Manhattan distance) – an exam question will state this.




---

---

---

---

---

---

---

---

## g-score and f-score

In Dijkstra's algorithm, you kept track of the cost of the shortest path so far to a given node. We called this "cost from start".

In the A\* algorithm, this is called the **g-score**.

node	g-score	previous
A	0	none
B	$\infty$	none
C	$\infty$	none
D	$\infty$	none
E	$\infty$	none
F	$\infty$	none




---

---

---

---

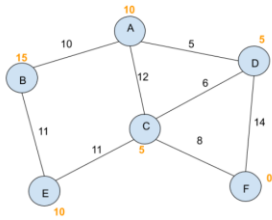
---

---

---

---

## g-score and f-score



We also need an **f-score**: the **sum** of the **cost so far** (g-score) and the **heuristic** (the estimated remaining cost *from this node*).

It is this **f-score** that we use to select the next current node.




---

---

---

---

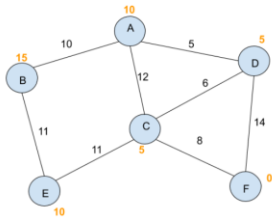
---

---

---

---

## Question: Finding g-score and f-score



### Question:

When finding the shortest path from A to F, we visit B first. What g-score and f-score does B get?

### Answer:

g-score: 10

f-score: 25




---

---

---

---

---

---

---

---

## Step 1 – create the list

Create a list similar to Dijkstra but with columns for:

- **g-score**  
cost so far to this node, or  $g(\text{node})$
- **f-score**  
current best guess of cost to target *via this node*, also called  $f(\text{node})$

Node	g-score	f-score	Previous
A	0	10	none
B	∞	∞	none
C	∞	∞	none
D	∞	∞	none
E	∞	∞	none
F	∞	∞	none




---

---

---

---

---

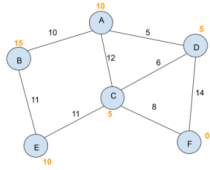
---

---

---

## Step 2a – choose lowest f()

In the unvisited list, A has the lowest f-score



Node	g-score	f-score	Previous
A	0	10	none
B	∞	∞	none
C	∞	∞	none
D	∞	∞	none
E	∞	∞	none
F	∞	∞	none

Check A's unvisited neighbours: **B, C, D**.




---

---

---

---

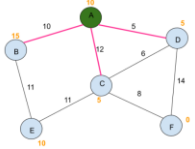
---

---

---

---

## Step 2b – process node



$$\begin{aligned} g(B) &= g(A) + AB = 0 + 10 = 10 \\ f(B) &= g(B) + h(B) = 10 + 15 = 25 \\ g(C) &= g(A) + AC = 0 + 12 = 12 \\ f(C) &= g(C) + h(C) = 12 + 5 = 17 \\ g(D) &= g(A) + AD = 0 + 5 = 5 \\ f(D) &= g(D) + h(D) = 5 + 5 = 10 \end{aligned}$$

Node	g-score	f-score	Previous
A	0	10	none
B	10	25	A
C	12	17	A
D	5	10	A
E	∞	∞	none
F	∞	∞	none




---

---

---

---

---

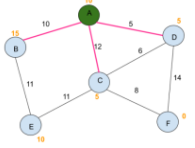
---

---

---

## Step 2c – mark node visited

You have now evaluated all routes from the current node



Mark node A **visited**.

Node	g-score	f-score	Previous
A	0	10	none
B	10	25	A
C	12	17	A
D	5	10	A
E	∞	∞	none
F	∞	∞	none




---

---

---

---

---

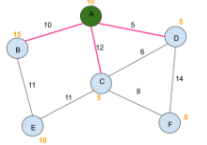
---

---

---

### Step 3a – choose lowest f()

D now has the lowest f-score.

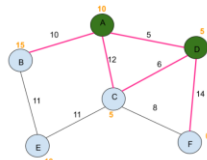


Node	g-score	f-score	Previous
A	0	10	none
B	10	25	A
C	12	17	A
D	5	10	A
E	∞	∞	none
F	∞	∞	none

Check D's unvisited neighbours: **C, F**.



### Step 3b – process node



$$g(C) = g(D) + DC = 5 + 6 = 11$$

$$f(C) = g(C) + h(C) = 11 + 5 = 16$$

$$g(F) = g(D) + DF = 5 + 14 = 19$$

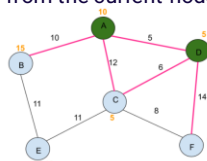
$$f(F) = g(F) + h(F) = 19 + 0 = 19$$

Node	g-score	f-score	Previous
A	0	10	none
B	10	25	A
C	<del>12</del> 11	<del>17</del> 16	<del>A</del> D
D	5	10	A
E	∞	∞	none
F	<del>∞</del> 19	<del>∞</del> 19	<del>none</del> D



### Step 3c – mark node visited

You have now evaluated all routes from the current node



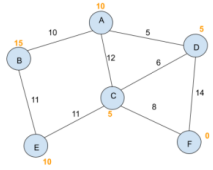
Mark node D visited.

Node	g-score	f-score	Previous
A	0	10	none
B	10	25	A
C	<del>12</del> 11	<del>17</del> 16	<del>A</del> D
D	5	10	A
E	∞	∞	none
F	<del>∞</del> 19	<del>∞</del> 19	<del>none</del> D



## Step 4a – choose lowest f()

C now has the lowest f-score.

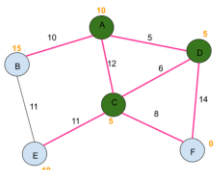


Node	g-score	f-score	Previous
A	0	10	none
B	10	25	A
C	11	16	D
D	5	10	A
E	$\infty$	$\infty$	none
F	19	19	D

Check C's unvisited neighbours: E, F.



## Step 4b – process node



Node	g-score	f-score	Previous
A	0	10	none
B	10	25	A
C	11	16	D
D	5	10	A
E	$\infty$ 22	$\infty$ 32	none C
F	19	19	D

$$g(E) = g(C) + CE = 11 + 11 = 22$$

$$f(E) = g(E) + h(E) = 22 + 10 = 32$$

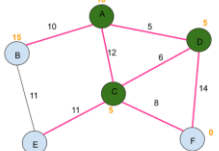
$$g(F) = g(C) + CF = 22 + 8 = 30$$

30 is greater than F's current g-score so don't change F row.



## Step 4b – process node

You have now evaluated all routes from the current node



Node	g-score	f-score	Previous
A	0	10	none
B	10	25	A
C	11	16	D
D	5	10	A
E	$\infty$ 22	$\infty$ 32	none C
F	19	19	D

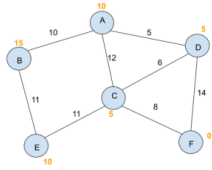
Mark node C visited.





## Step 5 – choose lowest f()

The unvisited node with the lowest f-score is now F, our **target node**.



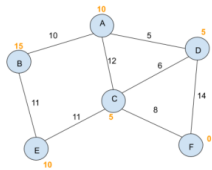
Node	g-score	f-score	Previous
A	0	10	none
B	10	25	A
C	11	16	D
D	5	10	A
E	22	32	C
F	19	19	D



## Step 5a – mark visited

As F is the target, no need to evaluate its neighbours.

Just mark F visited.

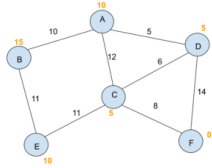


Node	g-score	f-score	Previous
A	0	10	none
B	10	25	A
C	11	16	D
D	5	10	A
E	22	32	C
F	19	19	D



## The final table

Note that we don't need to visit B and E, those paths have been "pruned" using the heuristics.



Node	g-score	f-score	Previous
A	0	10	none
B	10	25	A
C	11	16	D
D	5	10	A
E	22	32	C
F	19	19	D

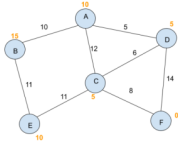


## Shortest Path from A → F

F's previous node is D,  
D's previous node is A

Therefore, the  
"shortest path" is

**A → D → F**  
costing  $g(F) = 19$ .



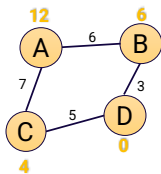
Node	g-score	f-score	Previous
A	0	10	none
B	10	25	A
C	11	16	D
D	5	10	A
E	22	32	C
F	19	19	D



## Another Worked Example of A\* in action



## Worked Example 2 – A\* Traversal from A to D



Node	g-score	f-score	Previous
A	0	12	none
B	6	12	A
C	7	11	A
D	∞	∞	none

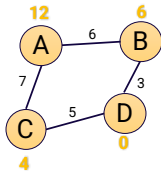
This is a partially-completed traversal.

We have visited A.

**Activity:** Verify the table is correct so far.



## Activity 2 – Complete the A\* Traversal on Handout 1



Node	g-score	f-score	Previous
A	0	12	none
B	6	12	A
C	7	11	A
D	$\infty$	$\infty$	none

Now complete the table using the A\* algorithm to find a route from A to D.

Use **Handout 1 Activity 2**




---

---

---

---

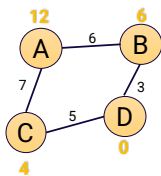
---

---

---

---

## Activity 2 – Solution



Node	g-score	f-score	Previous
A	0	12	none
B	6	12	A
C	7	11	A
D	12	12	C

We evaluate C and reach our destination D, so we stop and return the route A → C → D costing 12.

What do you notice?




---

---

---

---

---

---

---

---

## Practice Gameboard

---




---

---

---

---

---

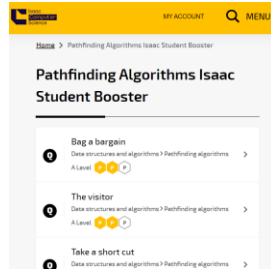
---

---

---

## Activity 3 – Isaac Gameboard

Now complete the gameboard on Isaac Computer Science at [bit.ly/iapathfinding](https://bit.ly/iapathfinding)  
Use the hints to help you.




---

---

---

---

---

---

---

---

## Recap Learning Objectives

- Understand how the Dijkstra and A\* pathfinding algorithms work, including the use of heuristics
- Use Dijkstra and A\* on a given graph to find the shortest path
- Understand the benefits and drawbacks of A\* over Dijkstra

## Questions?

---

---

---

---

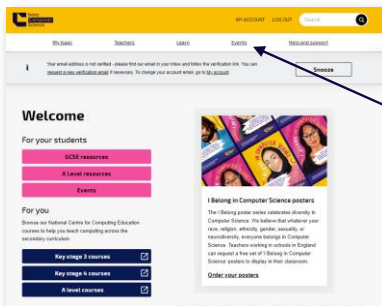
---

---

---

---

## Check for more ISAAC boosters



Keep an eye out for more student booster events

---

---

---

---

---

---

---

---



---

---

---

---

---

---

---

---