

# Shortest Path I: Dijkstra

Instructor: Meng-Fen Chiang

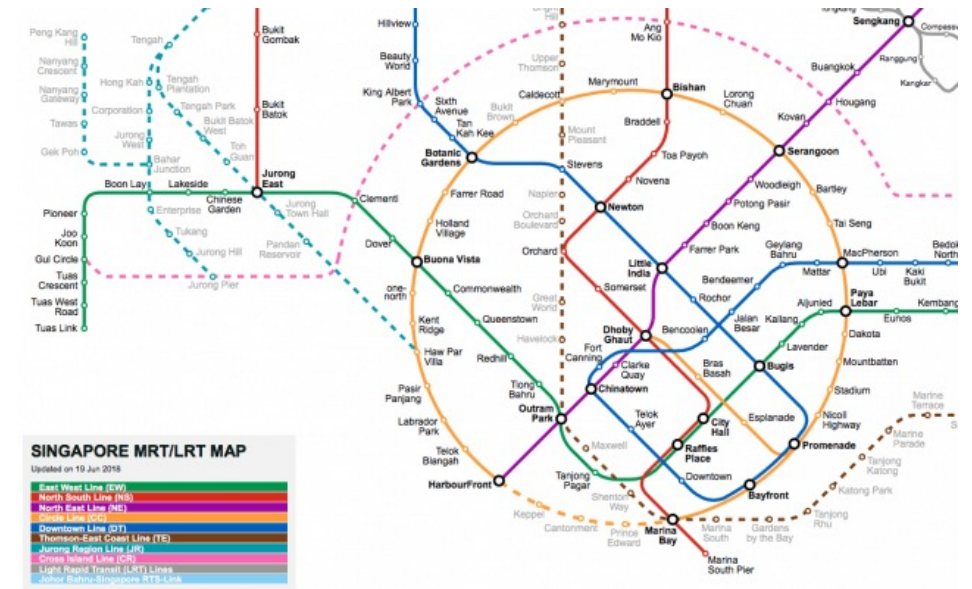
COMPCSI220: WEEK 11



Slides adapted from Giovanni Russello, Nevil Brownlee, and Ulrich Speidel

# OUTLINE

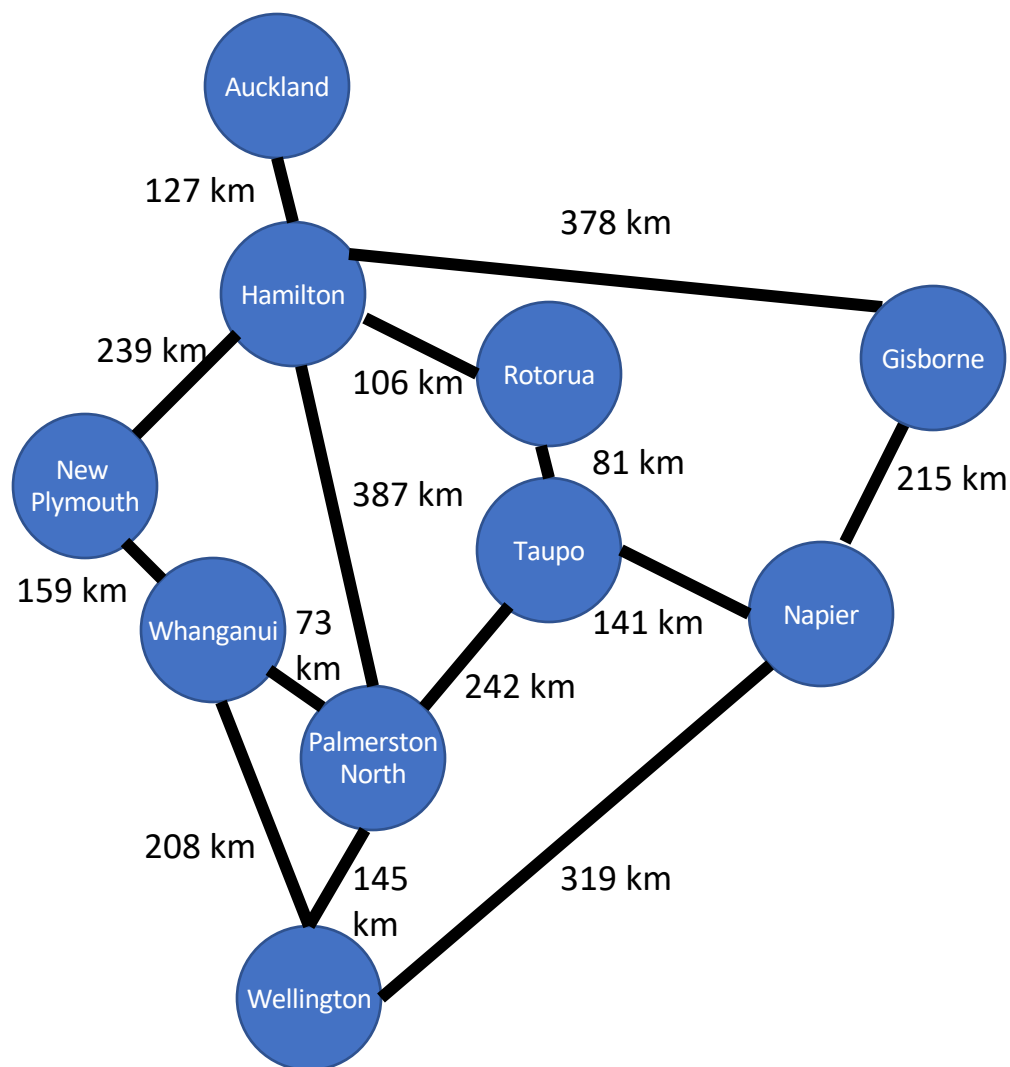
- Weighted Graphs
  - Representation
  - Weight as cost functions
- Algorithms on Weighted Graphs
  - Dijkstra
  - Bellman-Ford
  - Floyd-Warshall



# Weighted (Di)graphs

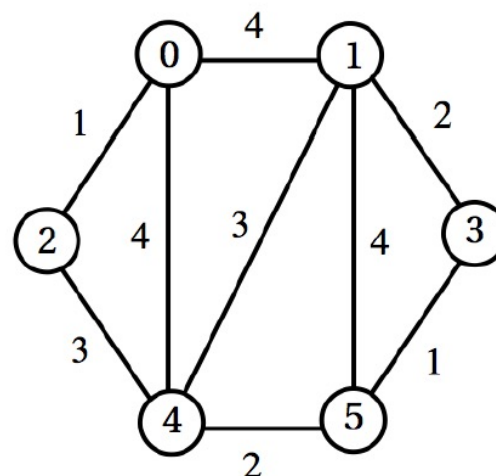
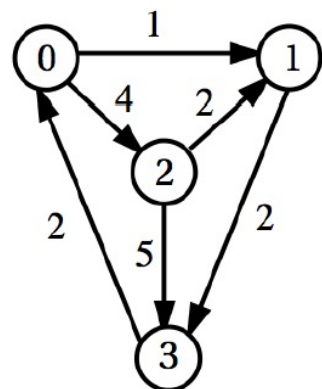
- Very common in applications, also called “networks”. Optimization problems on networks are important in operations research.
- Each arc carries a real number “weight”, usually positive, can be  $+\infty$ . Weight typically represents cost, distance, time. We may use the words “weight” and “cost” interchangeably in the slides.
- Representation: weighted adjacency matrix or double adjacency list.
- Standard problems concern finding a minimum or maximum weight path between given nodes (covered here), spanning tree, cycle or tour (e.g TSP), matching, flow, etc.

# Example: Weighted Graph



Source of distances: Google Maps

# Computer Representations of Weighted Digraphs

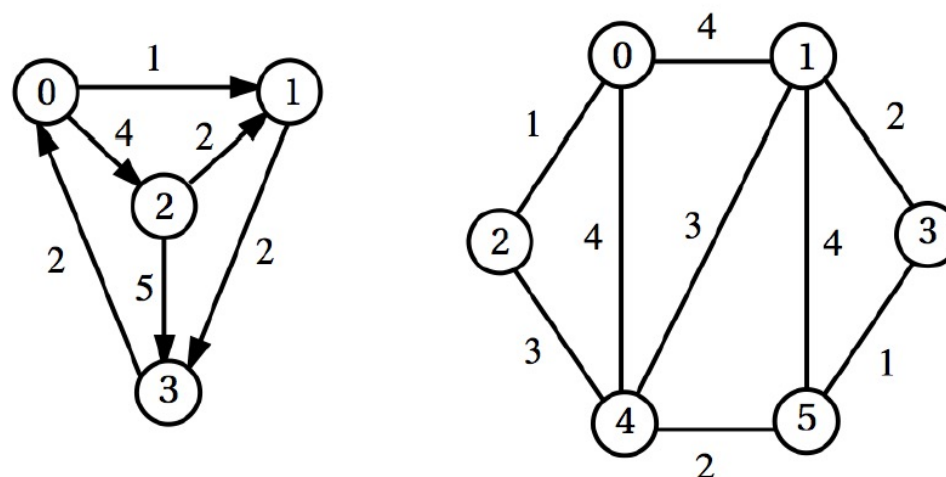


Weighted Adjacency Matrices (Cost Matrices):

$$\begin{bmatrix} 0 & 1 & 4 & 0 \\ 0 & 0 & 0 & 2 \\ 0 & 2 & 0 & 5 \\ 2 & 0 & 0 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 0 & 4 & 1 & 0 & 4 & 0 \\ 4 & 0 & 0 & 2 & 3 & 4 \\ 1 & 0 & 0 & 0 & 3 & 0 \\ 0 & 2 & 0 & 0 & 0 & 1 \\ 4 & 3 & 3 & 0 & 0 & 2 \\ 0 & 4 & 0 & 1 & 2 & 0 \end{bmatrix}$$

# Computer Representations of Weighted Digraphs



Weighted (Double) Adjacency Lists:

1	1	2	4
3	2		
1	2	3	5
0	2		

1	4	2	1	4	4		
0	4	3	2	4	3	5	4
0	1	4	3				
1	2	5	1				
0	4	1	3	2	3	5	2
1	4	3	1	4	2		

# Example: Distance Matrix Representation

	Auckland	Gisborne	Hamilton	Napier	New Plymouth	Palmerston North	Rotorua	Taupo	Wellington	Whanganui
Auckland	0	$\infty$	127	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$
Gisborne	$\infty$	0	378	215	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$
Hamilton	127	378	0	$\infty$	239	387	106	$\infty$	$\infty$	$\infty$
Napier	$\infty$	215	$\infty$	0	$\infty$	$\infty$	$\infty$	141	319	$\infty$
New Plymouth	$\infty$	$\infty$	239	$\infty$	0	$\infty$	$\infty$	$\infty$	$\infty$	159
Palmerston North	$\infty$	$\infty$	387	$\infty$	$\infty$	0	$\infty$	242	145	73
Rotorua	$\infty$	$\infty$	106	$\infty$	$\infty$	$\infty$	0	81	$\infty$	$\infty$
Taupo	$\infty$	$\infty$	$\infty$	141	$\infty$	242	81	0	$\infty$	$\infty$
Wellington	$\infty$	$\infty$	$\infty$	319	$\infty$	145	$\infty$	$\infty$	0	208
Whanganui	$\infty$	$\infty$	$\infty$	$\infty$	159	73	$\infty$	$\infty$	208	0

# Example: Adjacency List Representation

Auckland: Hamilton, 127

Gisborne: Hamilton, 378, Napier, 215

Hamilton: Auckland, 127, Gisborne 378, New Plymouth, 239, Palmerston North, 387, Rotorua, 106

Napier: Gisborne, 215, Taupo, 141, Wellington, 319

New Plymouth: Hamilton, 239, Whanganui, 159

Palmerston North: Hamilton, 387, Taupo, 242, Whanganui, 73, Wellington, 145

Rotorua: Hamilton, 106, Taupo, 81

Taupo: Napier, 141, Palmerston North, 242, Rotorua, 81

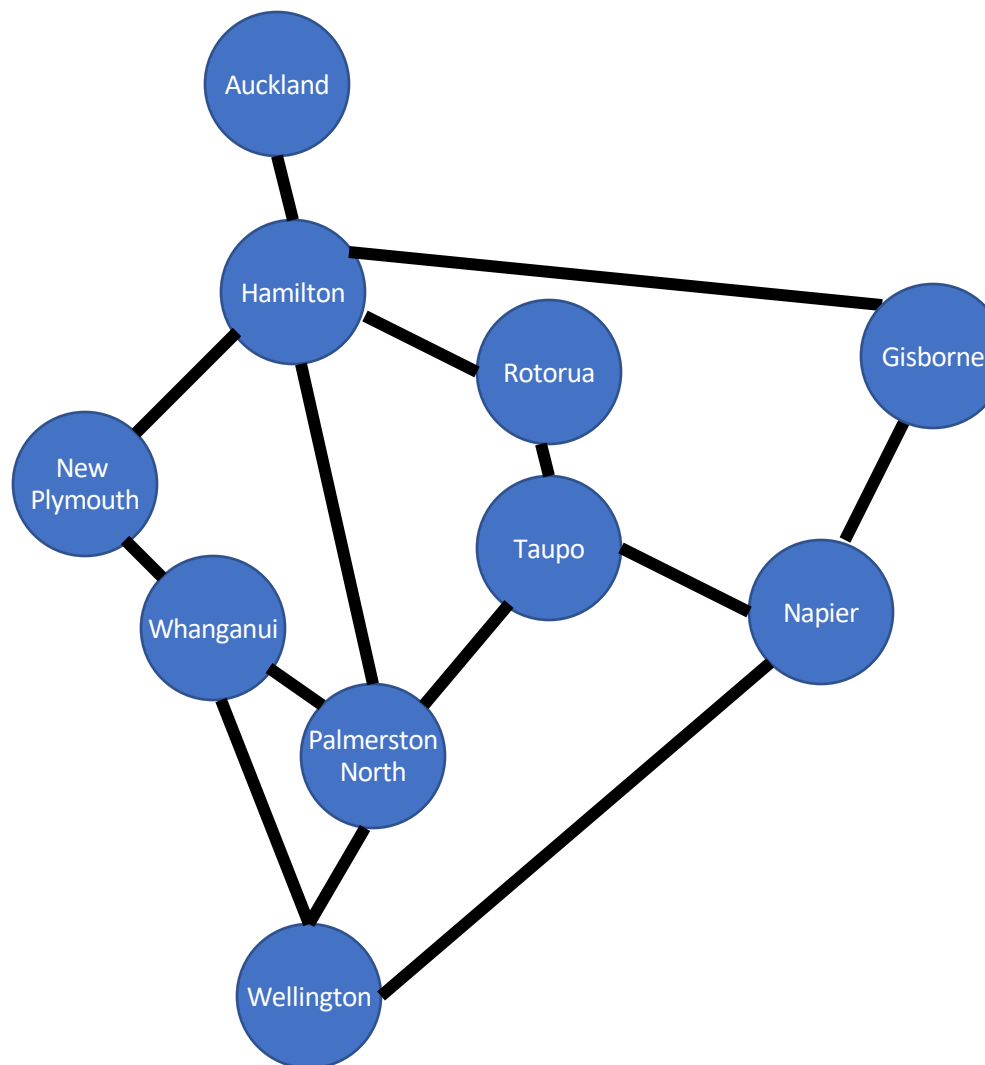
Wellington: Napier, 319, Palmerston North, 145, Whanganui, 208

Whanganui: New Plymouth, 159, Palmerston North, 73, Wellington, 208

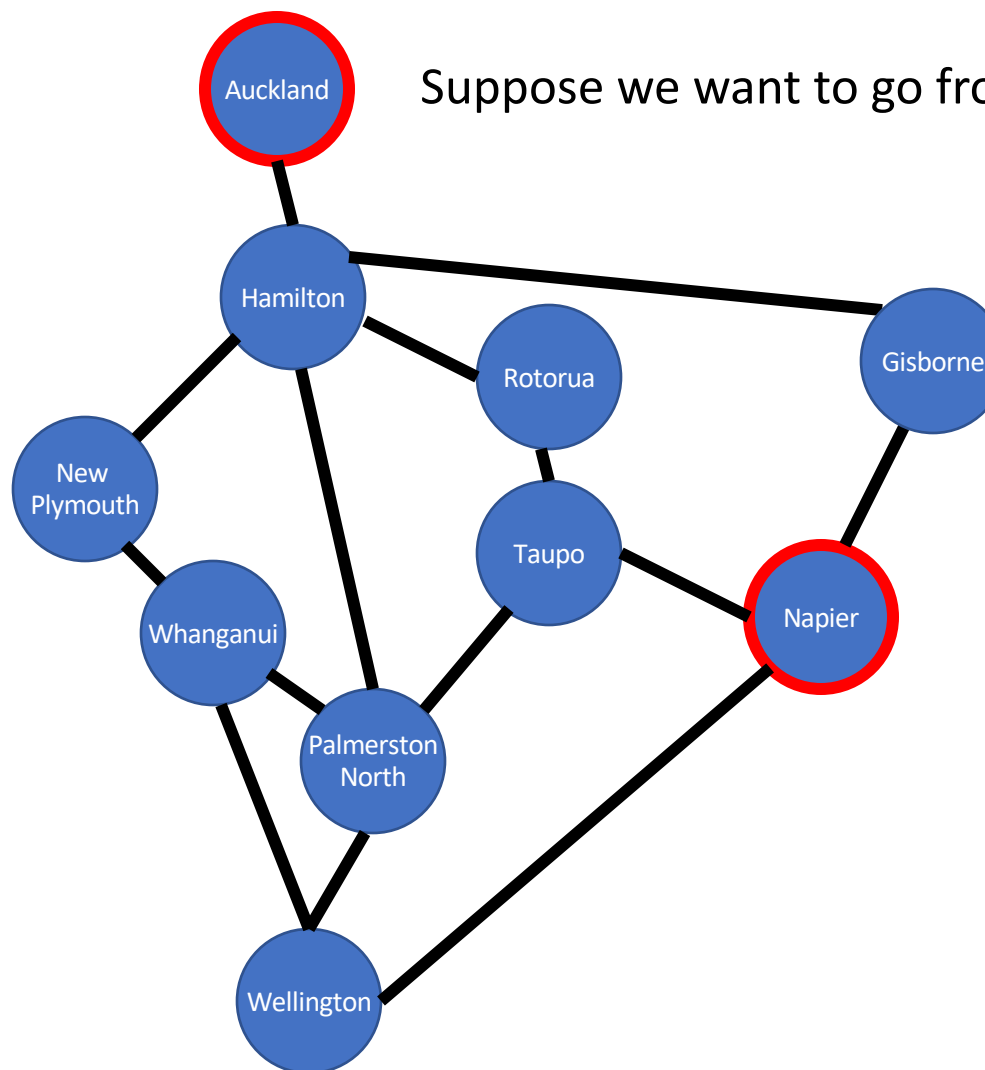
Essentially, we need the same number of extra storage spaces as there are objects, so the fundamental complexity does not change!



# Graph: North Island Road Network

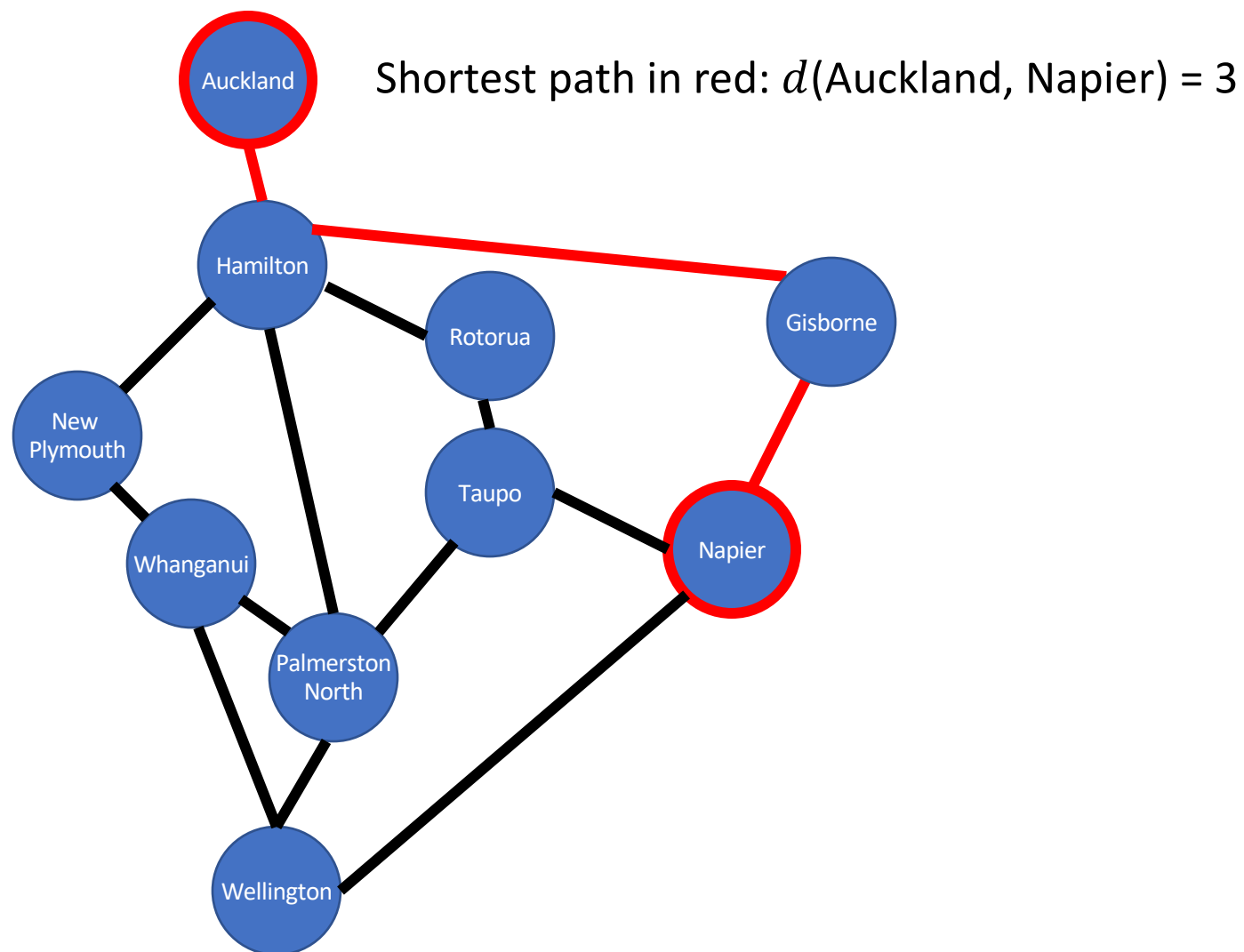


# Graph: North Island Road Network

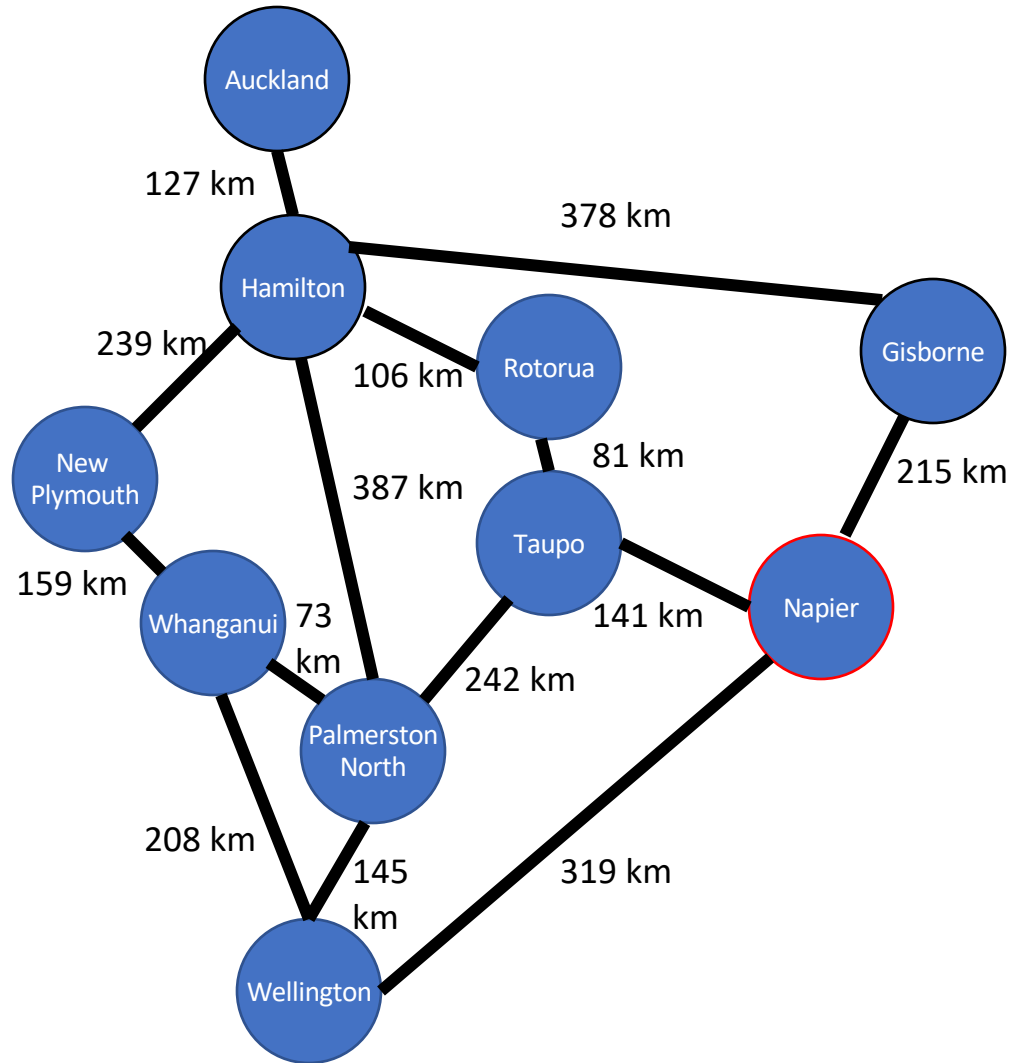


Suppose we want to go from **Auckland** to **Napier**

# Graph: North Island Road Network

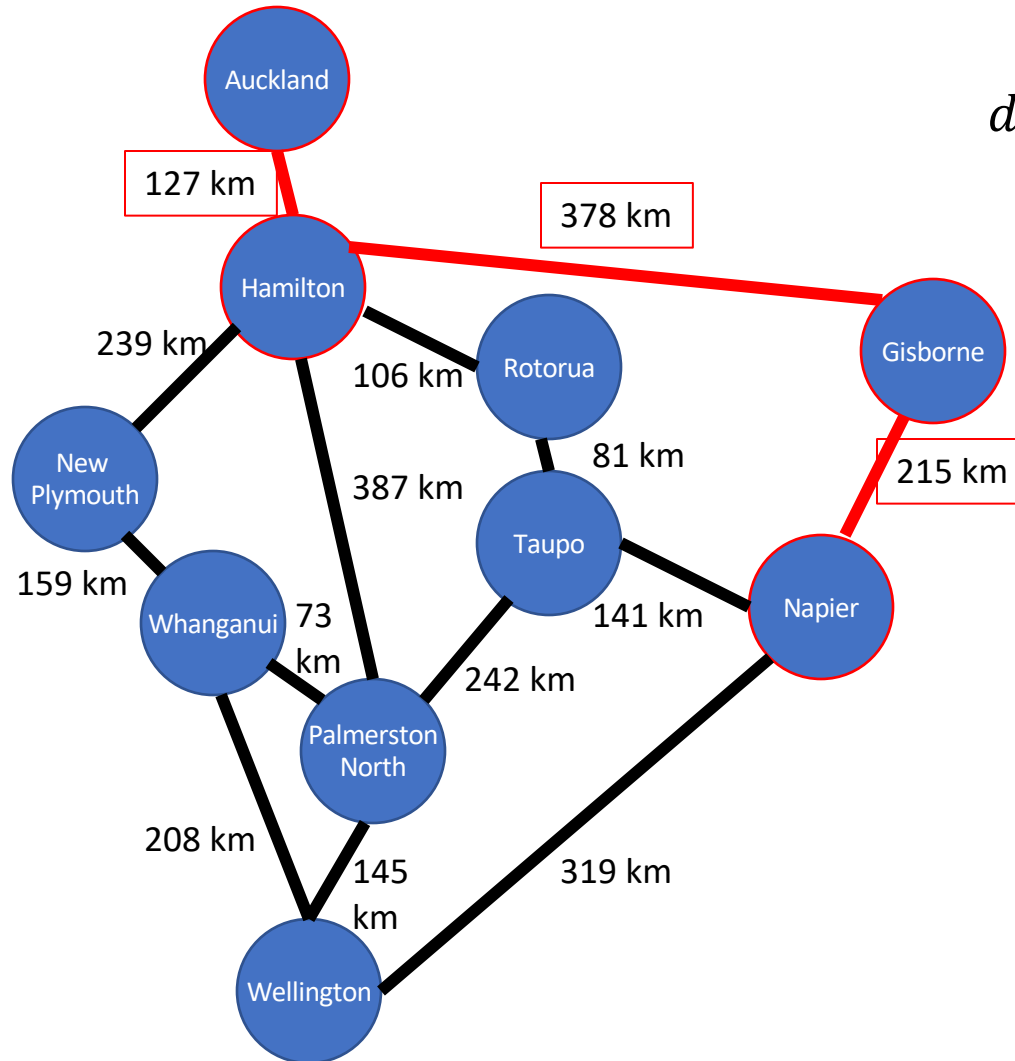


# Weighted Graph: North Island Road Network



Source of distances: Google Maps

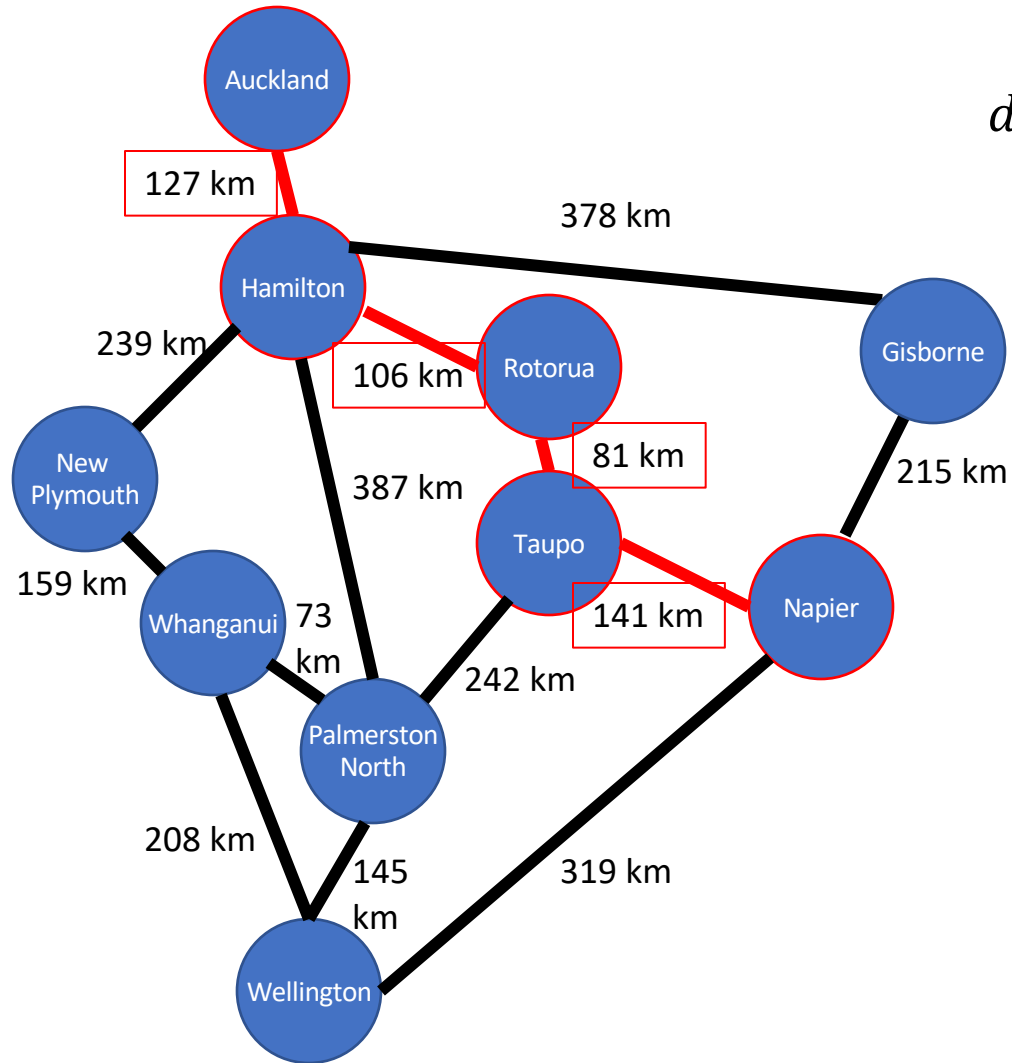
# Weighted Graph: North Island Road Network



$d(\text{Auckland, Napier}) = 720 \text{ km}$

Source of distances: Google Maps

# Weighted Graph: North Island Road Network



$d(\text{Auckland, Napier}) = 455 \text{ km}$

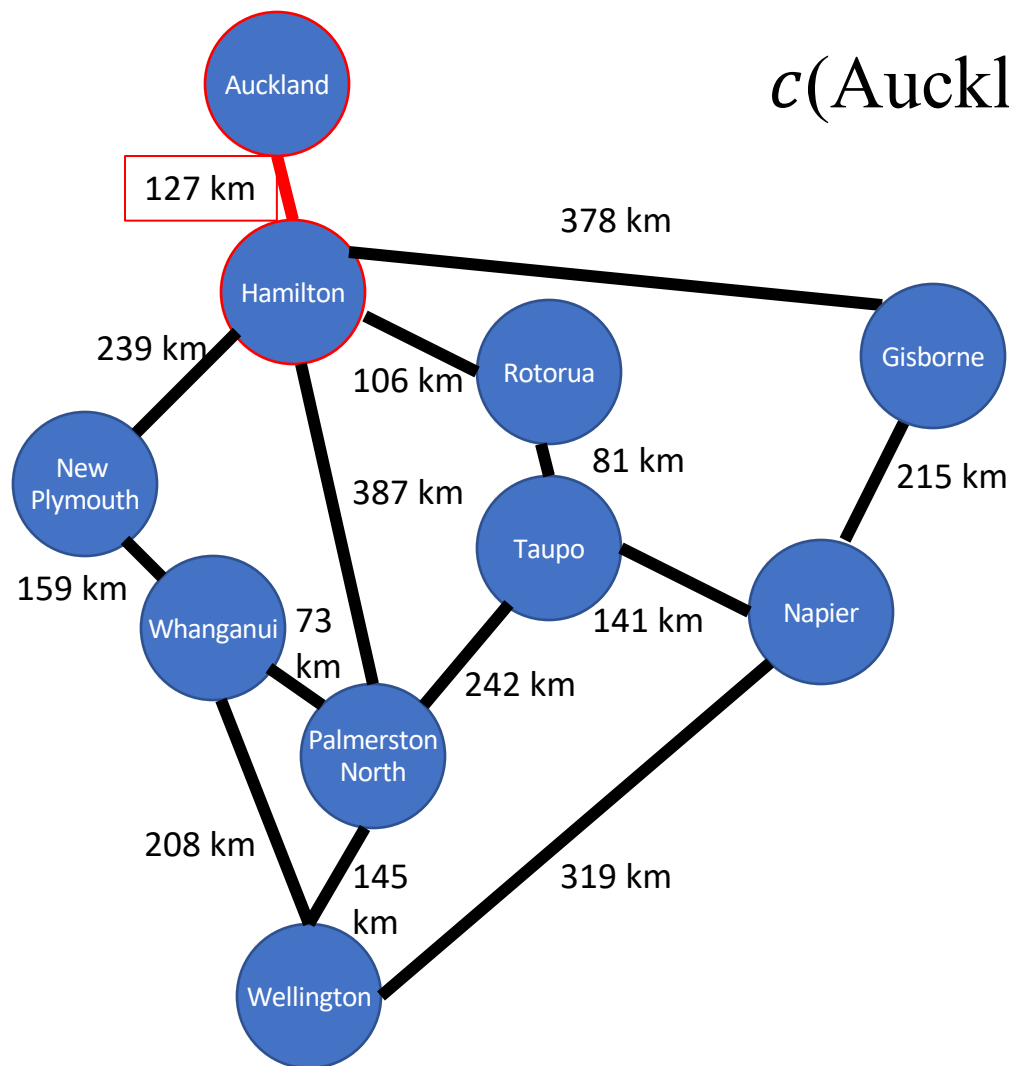
Source of distances: Google Maps

# Paths/Distances (revisited)

- **Definition.** For a digraph  $(V, E)$  with arc weights  $\{c(u, v) | (u, v) \in E\}$  we say that the **distance**  $d(u, v)$  between two vertices  $u$  and  $v$  of  $V$  is the minimum cost of a path between  $u$  and  $v$ . The **cost** of a walk/path  $v_0, v_1, \dots, v_k$  is  $\sum_{i=0}^{k-1} c(v_i, v_{i+1})$
- **Definition.** The diameter of a (di-)graph  $G = (V, E)$  is the maximum of  $d(u, v)$  over all pairs  $u, v \in V$ . If the (di-)graph is not (strongly) connected, the diameter of  $G$  is not defined.

# Example: Cost

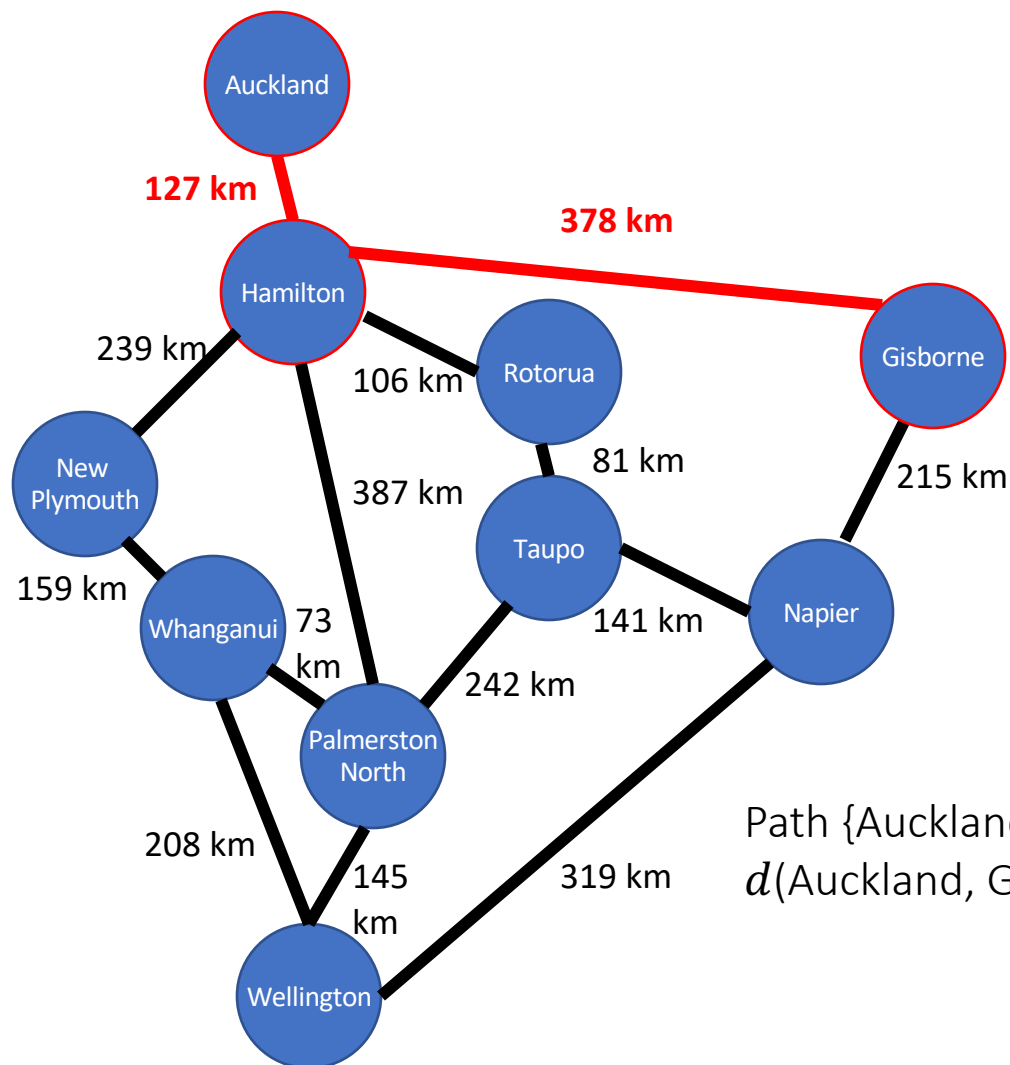
$$c(\text{Auckland, Hamilton}) = 127 \text{ km}$$



Source of distances: Google Maps



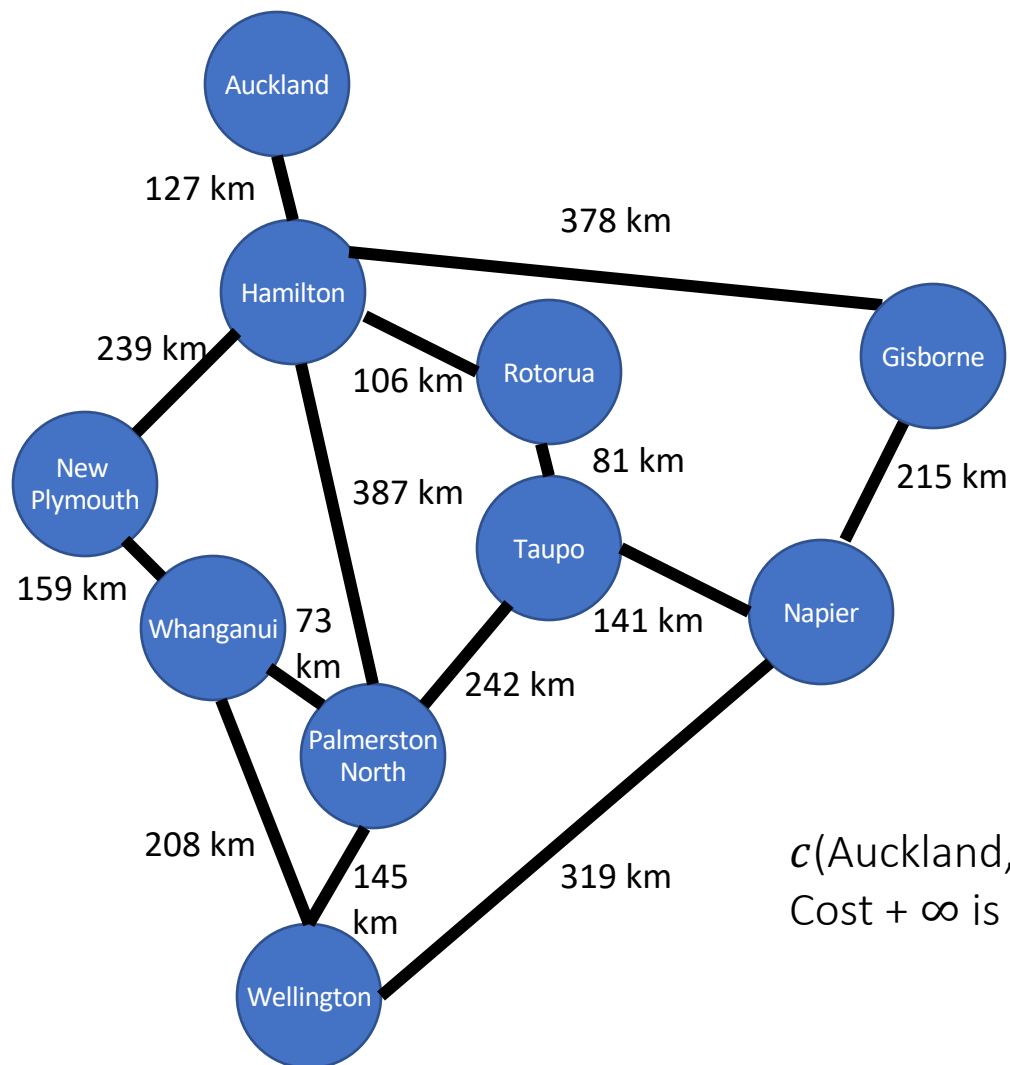
# Example: Distance v.s. Cost



Path {Auckland, Hamilton, Gisborne} then  
 $d(\text{Auckland, Gisborne}) = c(\text{Auckland, Hamilton}) + c(\text{Hamilton, Gisborne})$

Source of distances: Google Maps

# Example: Distance v.s. Cost

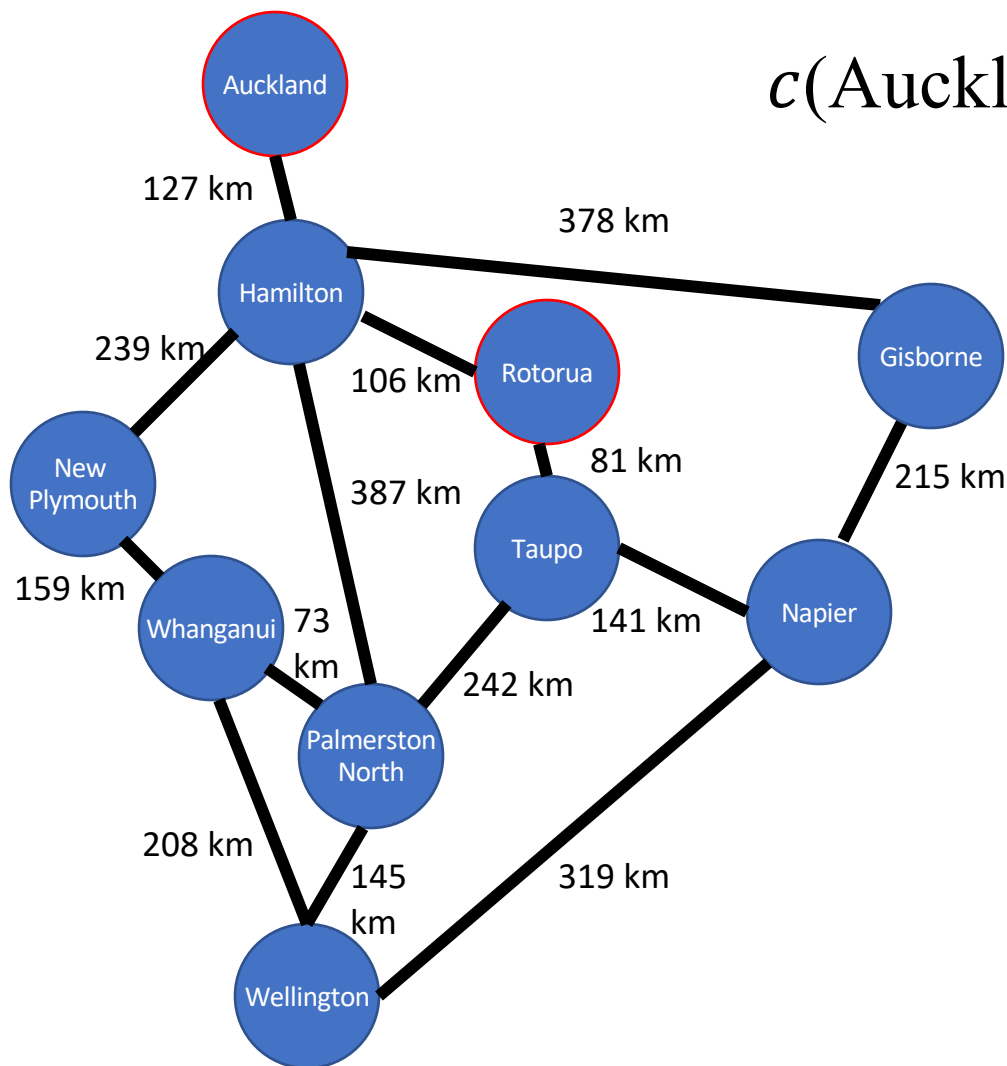


$c(\text{Auckland}, \text{Hamilton}) = 127$  and  $c(\text{Auckland}, \text{Gisborne}) = +\infty$   
 Cost +  $\infty$  is for cities that aren't directly connected.

Source of distances: Google Maps

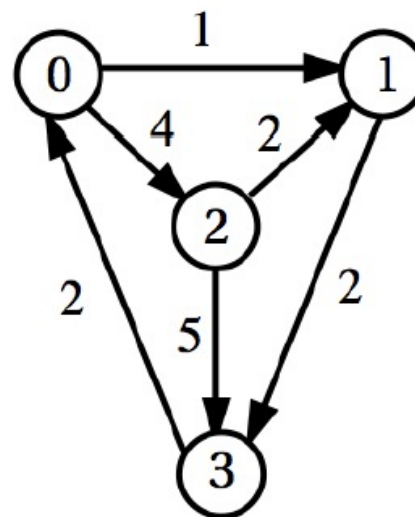
# Example: The Cost of no Edge

$$c(\text{Auckland, Rotoura}) = +\infty$$



Source of distances: Google Maps

# Example: Diameter



weighted adjacency matrix:

$$\begin{bmatrix} 0 & 1 & 4 & 0 \\ 0 & 0 & 0 & 2 \\ 0 & 2 & 0 & 5 \\ 2 & 0 & 0 & 0 \end{bmatrix}$$

distance matrix:

$$\begin{bmatrix} 0 & 1 & 4 & 3 \\ 4 & 0 & 8 & 2 \\ 6 & 2 & 0 & 4 \\ 2 & 3 & 6 & 0 \end{bmatrix}$$

Hence, the diameter is 8.

# Issue: Negative Weights

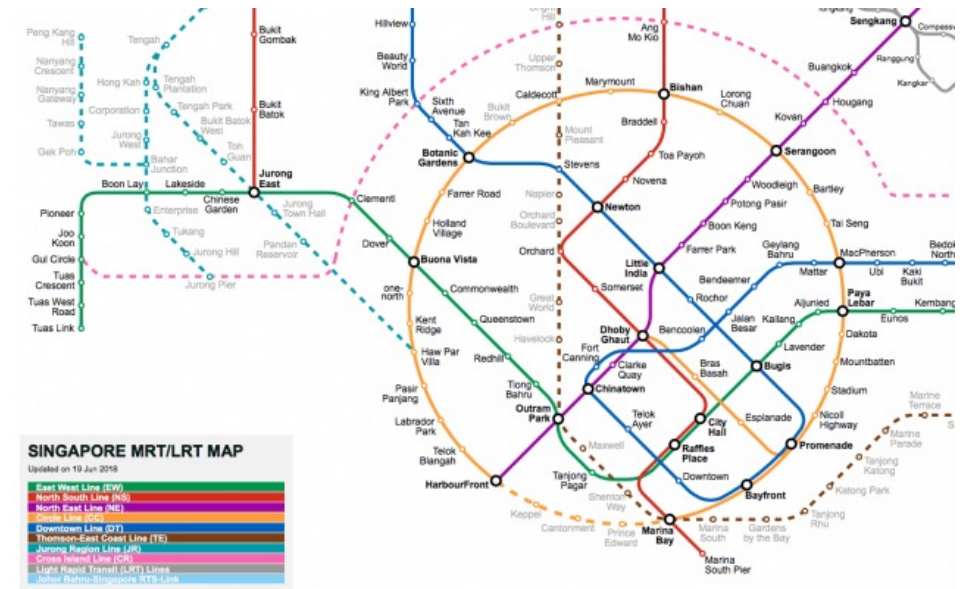
- The vast majority of weighted graph problems have positive weights
- However, exceptions exist where the weight of an edge/arc can be negative.

# Example: Negative Weights

- Consider going on a holiday tripping around the world.
- Between some cities, you may need to buy a train/ferry ticket or an airfare (=cost, positive weight),
- but between others you might actually have the opportunity to earn money by working as crew on a ship, train, or plane (=gain, negative weight).

# OUTLINE

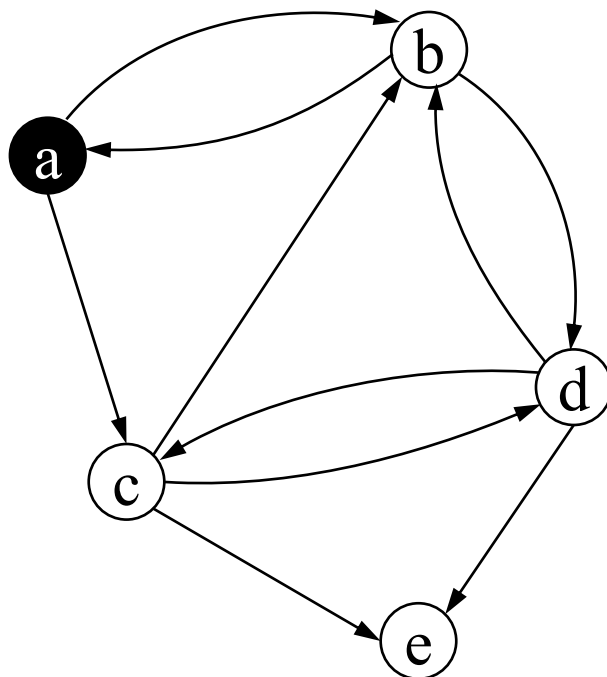
- Weighted Graphs
  - Representation
  - Weight as cost functions
- Algorithms on Weighted Graphs
  - Dijkstra
  - Bellman-Ford
  - Floyd-Warshall



# Single-source Shortest Path Problem (SSSP)

- Given an originating node  $v$ , find shortest (minimum weight) path to each other node.  
If all weights are equal then BFS works, otherwise not.

Unweighted digraph

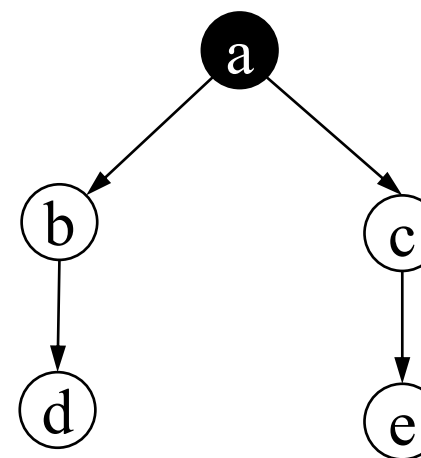


$d(a, b) = 1$ , path: a, b

$d(a, c) = 1$ , path: a, c

$d(a, d) = 2$ , path: a, b, d or a, c, d

$d(a, e) = 2$ , path: a, c, e



If BFS is used:

$d(a, b) = 1$ , path: a, b

$d(a, c) = 1$ , path: a, c

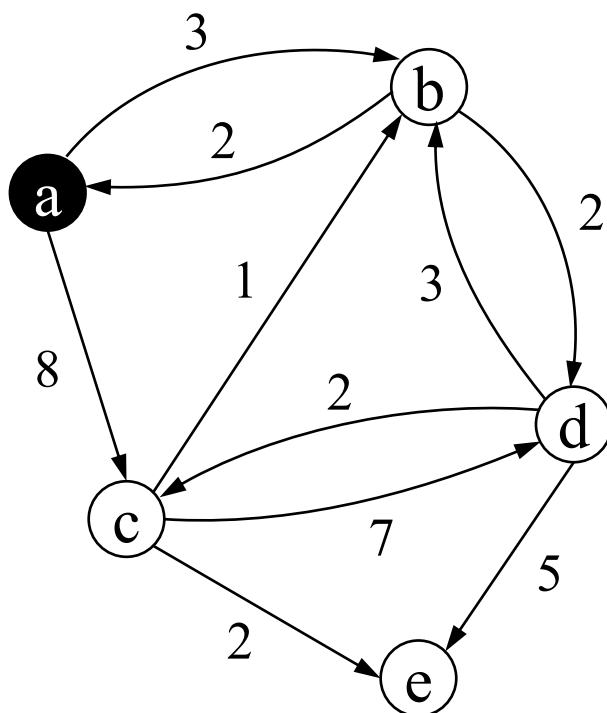
$d(a, d) = 2$ , path: a, b, d

$d(a, e) = 2$ , path: a, c, e

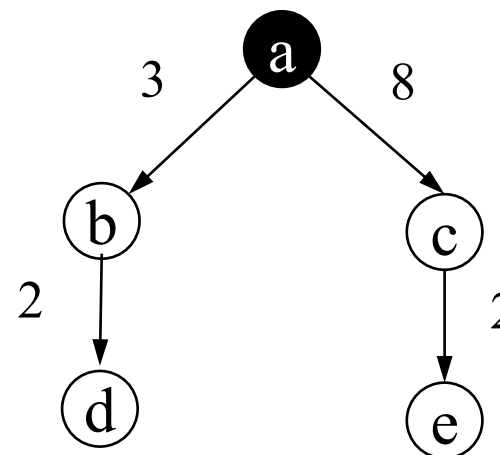


# Single-source Shortest Path Problem (SSSP)

- Given an originating node  $v$ , find shortest (minimum weight) path to each other node.  
If all weights are equal then BFS works, otherwise not.



$d(a, b) = 3$ , path: a, b  
 $d(a, c) = 7$ , path: a, b, d, c  
 $d(a, d) = 5$ , path: a, b, d  
 $d(a, e) = 9$ , path: a, b, d, c, e



If BFS is used:

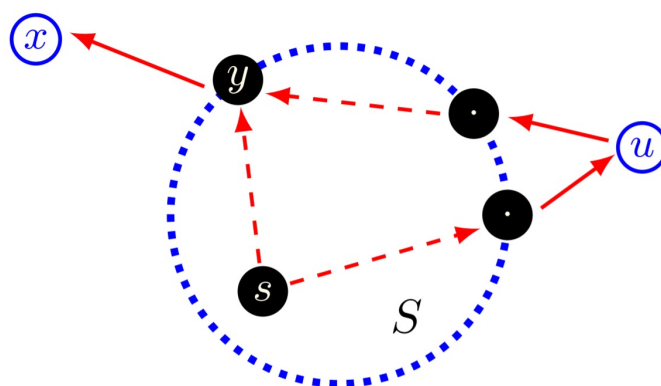
$d(a, b) = 3$ , path: a, b  
 $d(a, c) = 8$ , path: a, c  
 $d(a, d) = 5$ , path: a, b, d  
 $d(a, e) = 10$ , path: a, c, e

# Algorithms on Weighted Graphs

- **Dijkstra** (pronounced "Dyke-strā"): Used to find the cost to each destination vertex from a single source ("single source shortest path" - SSSP). Cannot handle negative weights.
- **Bellman-Ford**: solves SSSP as well, slower than Dijkstra but can handle negative weights
- **Floyd-Warshall**: solves all-pairs shortest path (APSP) problem – minimal cost between any given pair of vertices

# Single-source Shortest Path Problem (SSSP)

- Several algorithms are known; we present one, **Dijkstra's** algorithm. An example of a **greedy algorithm**; locally best choice is globally best. **Doesn't work if weights can be negative.**
  1. Maintain list  $S$  of visited nodes.
  2. Choose an unvisited node  $u$  with shortest  $S$ -path and put it to  $S$ .
  3. Update distances (of remaining unvisited nodes) from starting node  $s$  in case adding  $u$  has established shorter paths.
  4. Repeat.



Let an  **$S$ -path** be a path starting at node  $s$  and ending at node  $u$  with all nodes in  $S$  except possibly  $u$ .

# Dijkstra's Algorithm

---

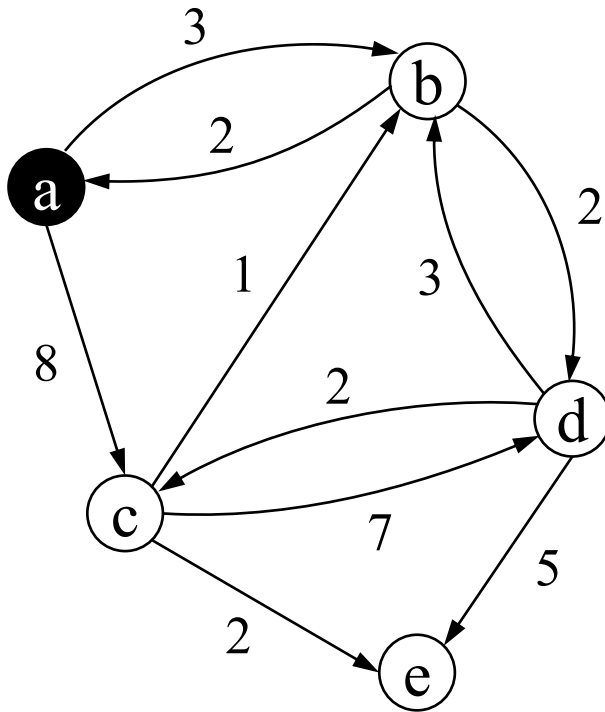
**Algorithm 1** Dijkstra's algorithm.

---

```
1: function DIJKSTRA(weighted digraph( $G, c$ ); node  $s \in V(G)$ )
2:   array  $colour[0..n - 1], dist[0..n - 1]$ 
3:   for  $u \in V(G)$  do
4:      $dist[u] \leftarrow c(s, u); colour[u] \leftarrow \text{WHITE}$ 
5:    $dist[s] \leftarrow 0; colour[s] \leftarrow \text{BLACK}$ 
6:   while there is a white node do
7:     find a white node  $u$  so that  $dist[u]$  is minimum
8:      $colour[u] \leftarrow \text{BLACK}$ 
9:     for each  $x$  adjacent to  $u$  do
10:      if  $colour[x] = \text{WHITE}$  then
11:         $dist[x] \leftarrow \min\{dist[x], dist[u] + c(u, x)\}$ 
12:   return  $dist$ 
```

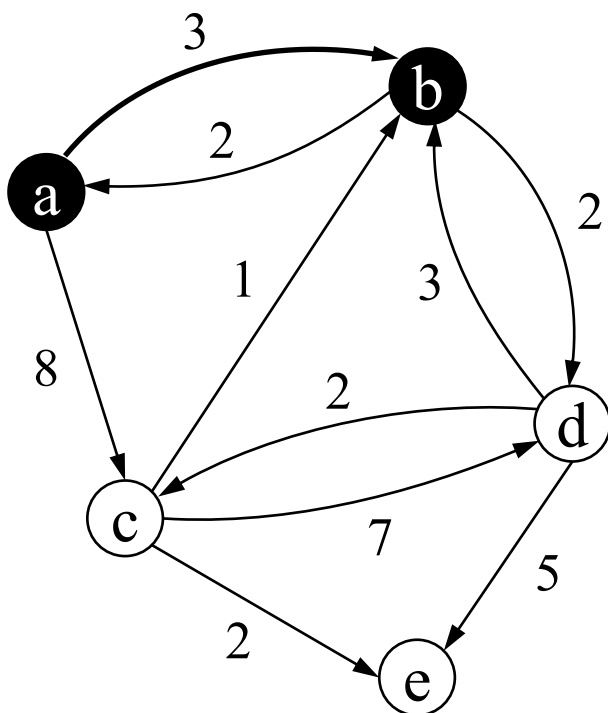
---

# Illustrating Dijkstra's algorithm



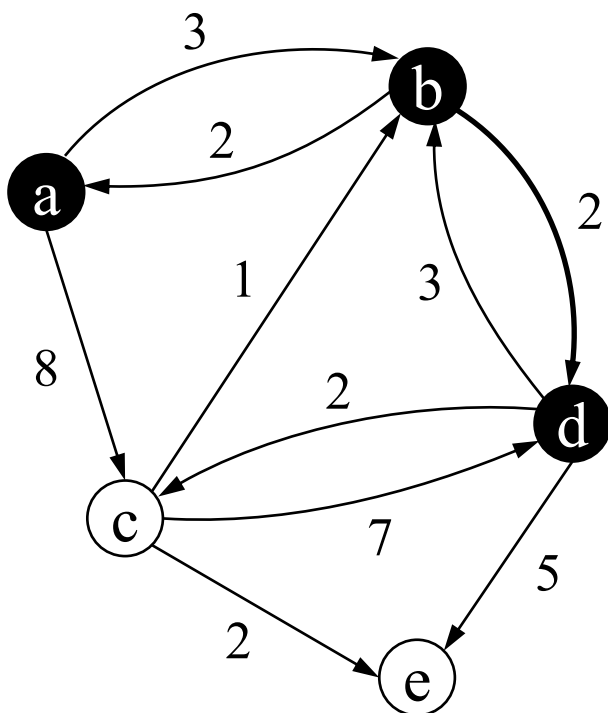
BLACK	$dist[x]$
	a, b, c, d, e
a	0, <b>3</b> , 8, $\infty$ , $\infty$

# Illustrating Dijkstra's algorithm



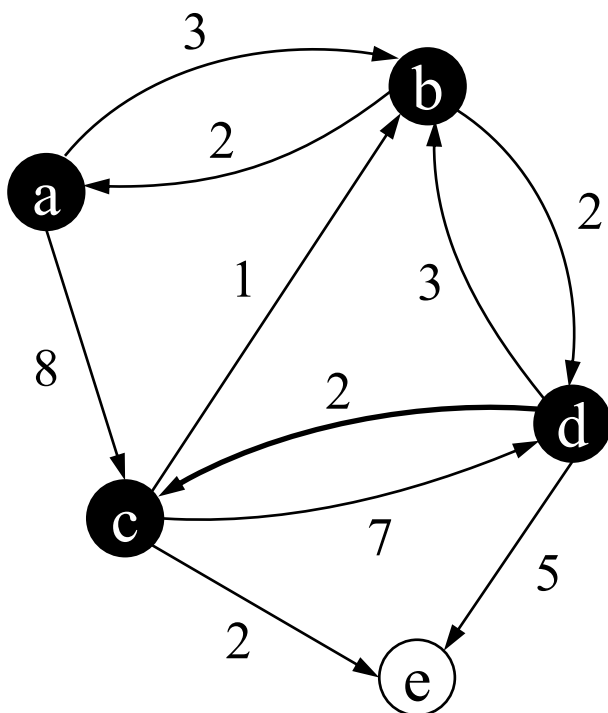
BLACK	$dist[x]$
	a, b, c, d, e
a	0, <b>3</b> , 8, $\infty$ , $\infty$
a, b	0, 3, 8, $3 + 2 = \mathbf{5}$ , $\infty$

# Illustrating Dijkstra's algorithm



BLACK	$dist[x]$
	a, b, c, d, e
a	0, <b>3</b> , 8, $\infty$ , $\infty$
a, b	0, 3, 8, $3 + 2 = \mathbf{5}$ , $\infty$
a, b, d	0, 3, $3 + 2 + 2 = \mathbf{7}$ , 5, 10

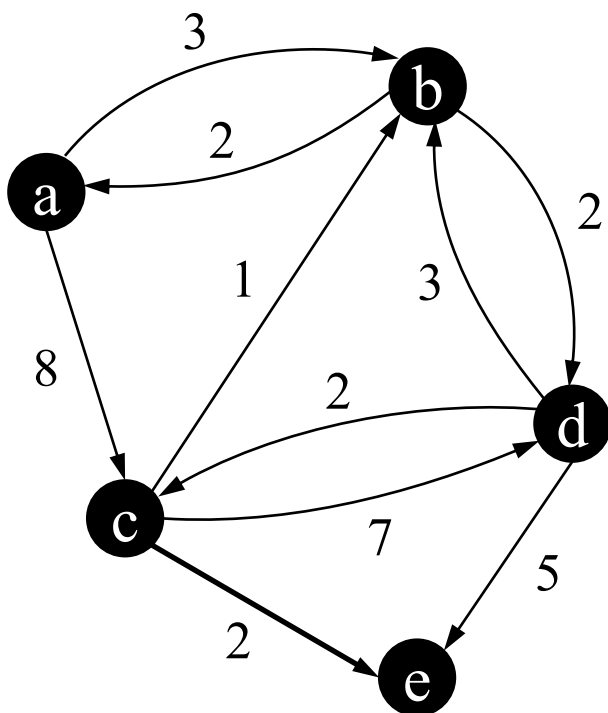
# Illustrating Dijkstra's algorithm



BLACK	$dist[x]$
	a, b, c, d, e
a	0, <b>3</b> , 8, $\infty$ , $\infty$
a, b	0, 3, 8, $3 + 2 = \mathbf{5}$ , $\infty$
a, b, d	0, 3, $3 + 2 + 2 = \mathbf{7}$ , 5, 10
a, b, c, d	0, 3, 7, 5, $7 + 2 = 9$

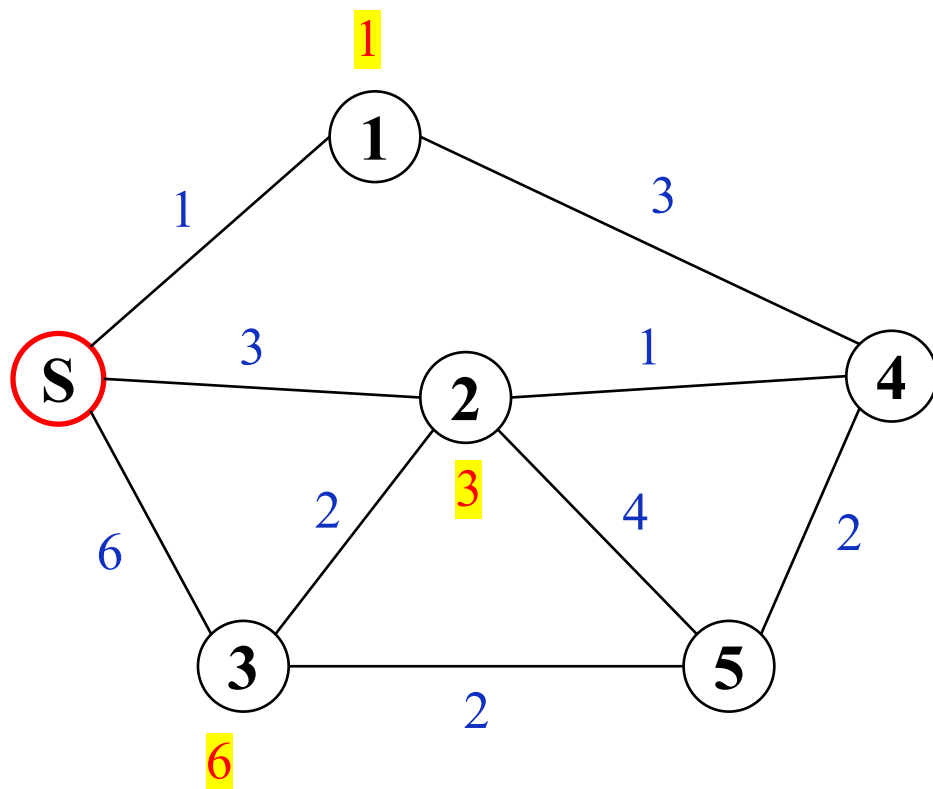


# Illustrating Dijkstra's algorithm



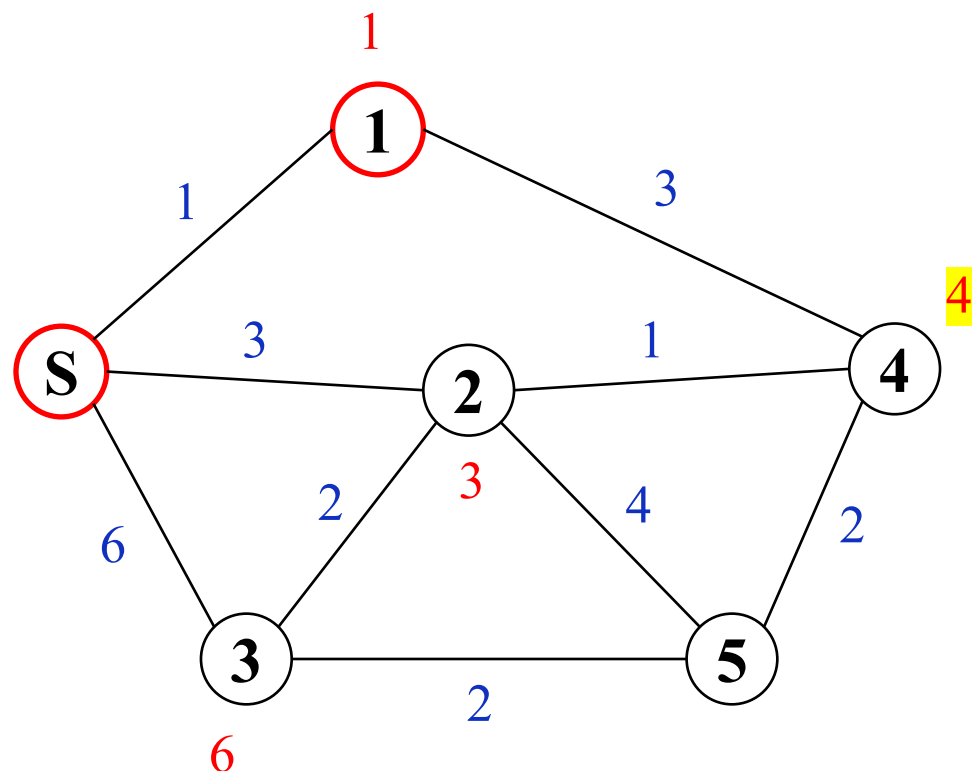
BLACK	$dist[x]$
	a, b, c, d, e
a	0, <b>3</b> , 8, $\infty$ , $\infty$
a, b	0, 3, 8, $3 + 2 = \mathbf{5}$ , $\infty$
a, b, d	0, 3, $3 + 2 + 2 = \mathbf{7}$ , 5, 10
a, b, c, d	0, 3, 7, 5, $7 + 2 = 9$
$V(G)$	

# Dijkstra's algorithm - Pairwise



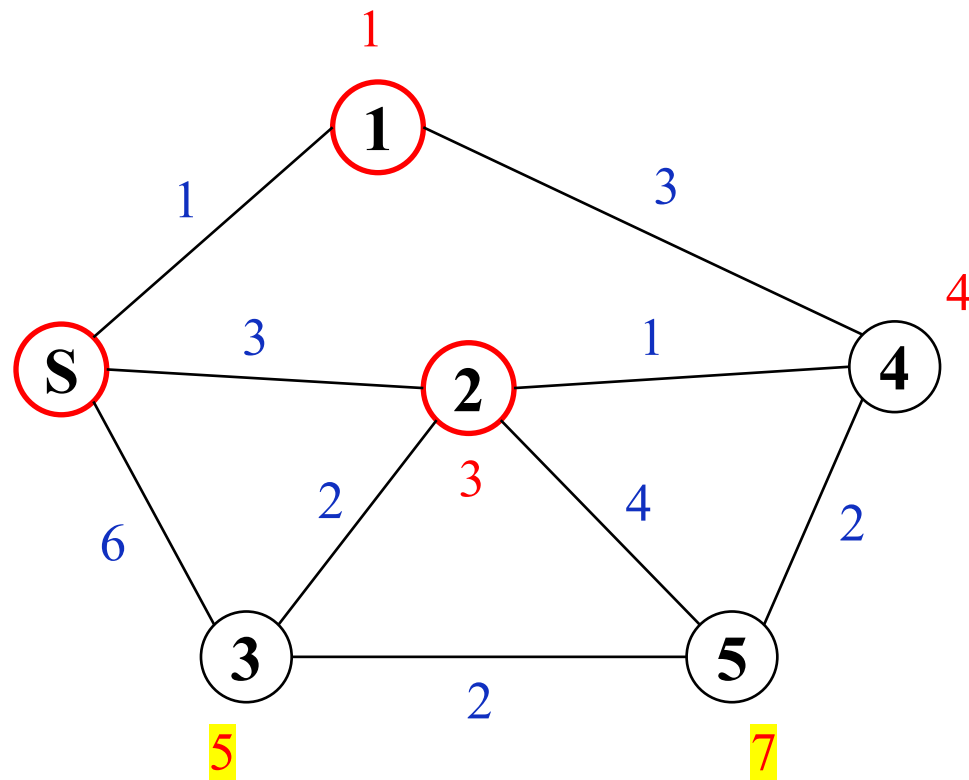
BLACK	$dist[x]$
	S, 1, 2, 3, 4, 5
S	0, 1, 3, 6, $\infty$ , $\infty$

# Dijkstra's algorithm - Pairwise



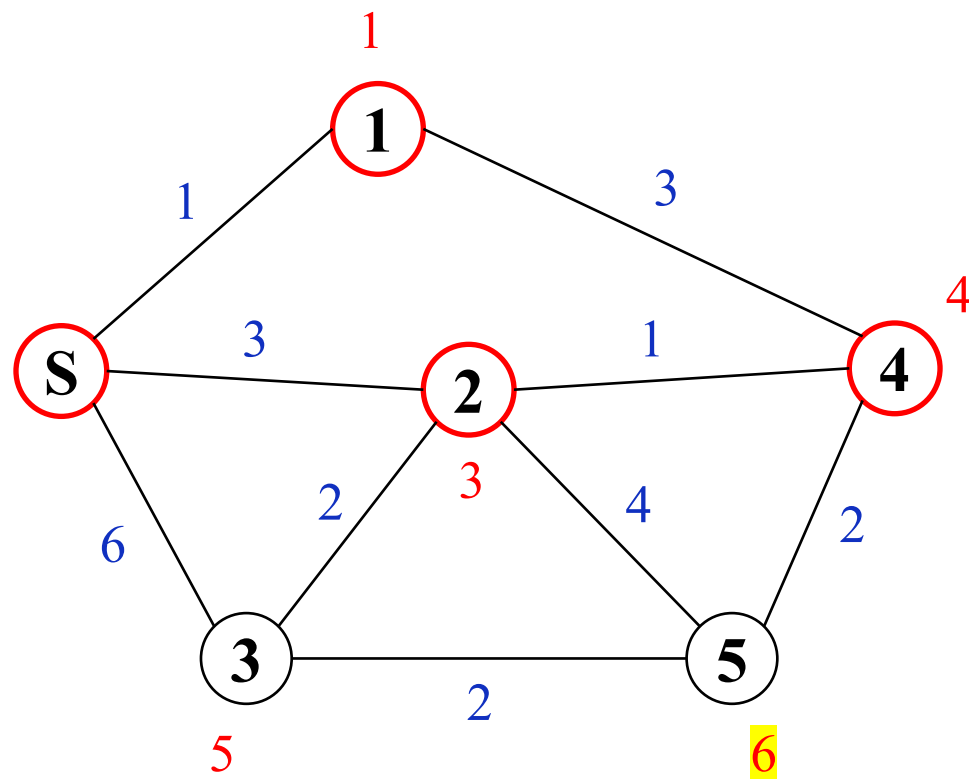
BLACK	$dist[x]$
	S, 1, 2, 3, 4, 5
S	0, 1, 3, 6, $\infty$ , $\infty$
S, 1	0, 1, 3, 6, 4, $\infty$

# Dijkstra's algorithm - Pairwise



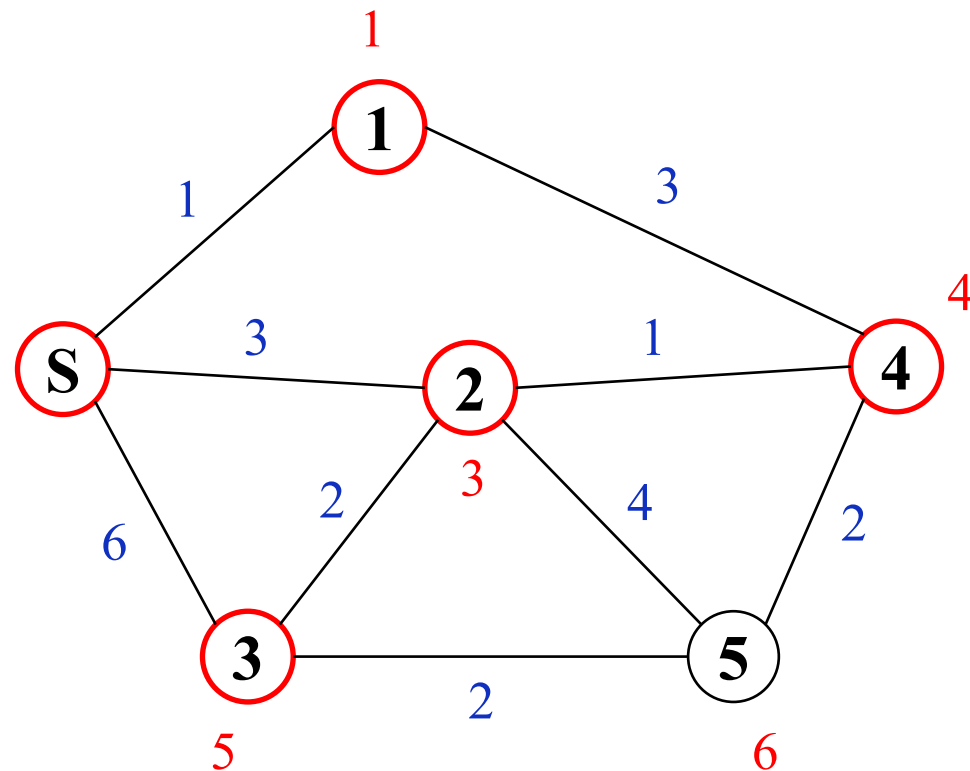
BLACK	$dist[x]$
	S, 1, 2, 3, 4, 5
S	0, 1, 3, 6, $\infty$ , $\infty$
S, 1	0, 1, 3, 6, 4, $\infty$
S, 1, 2	0, 1, 3, 5, 4, 7

# Dijkstra's algorithm - Pairwise



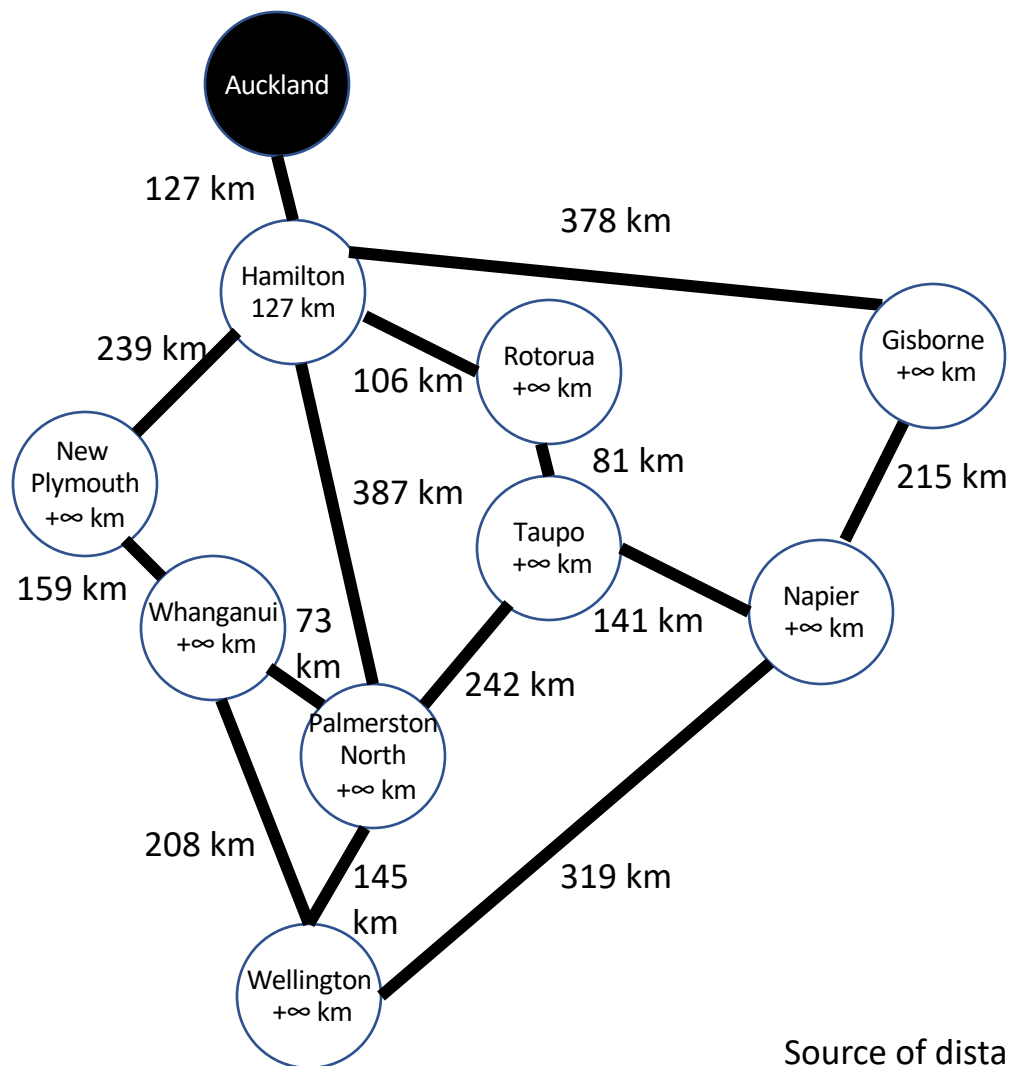
BLACK	$dist[x]$
	S, 1, 2, 3, 4, 5
S	0, <b>1</b> , 3, 6, $\infty$ , $\infty$
S, 1	0, 1, <b>3</b> , 6, 4, $\infty$
S, 1, 2	0, 1, 3, 5, <b>4</b> , 7
S, 1, 2, 4	0, 1, 3, <b>5</b> , 4, 6

# Dijkstra's algorithm - Pairwise



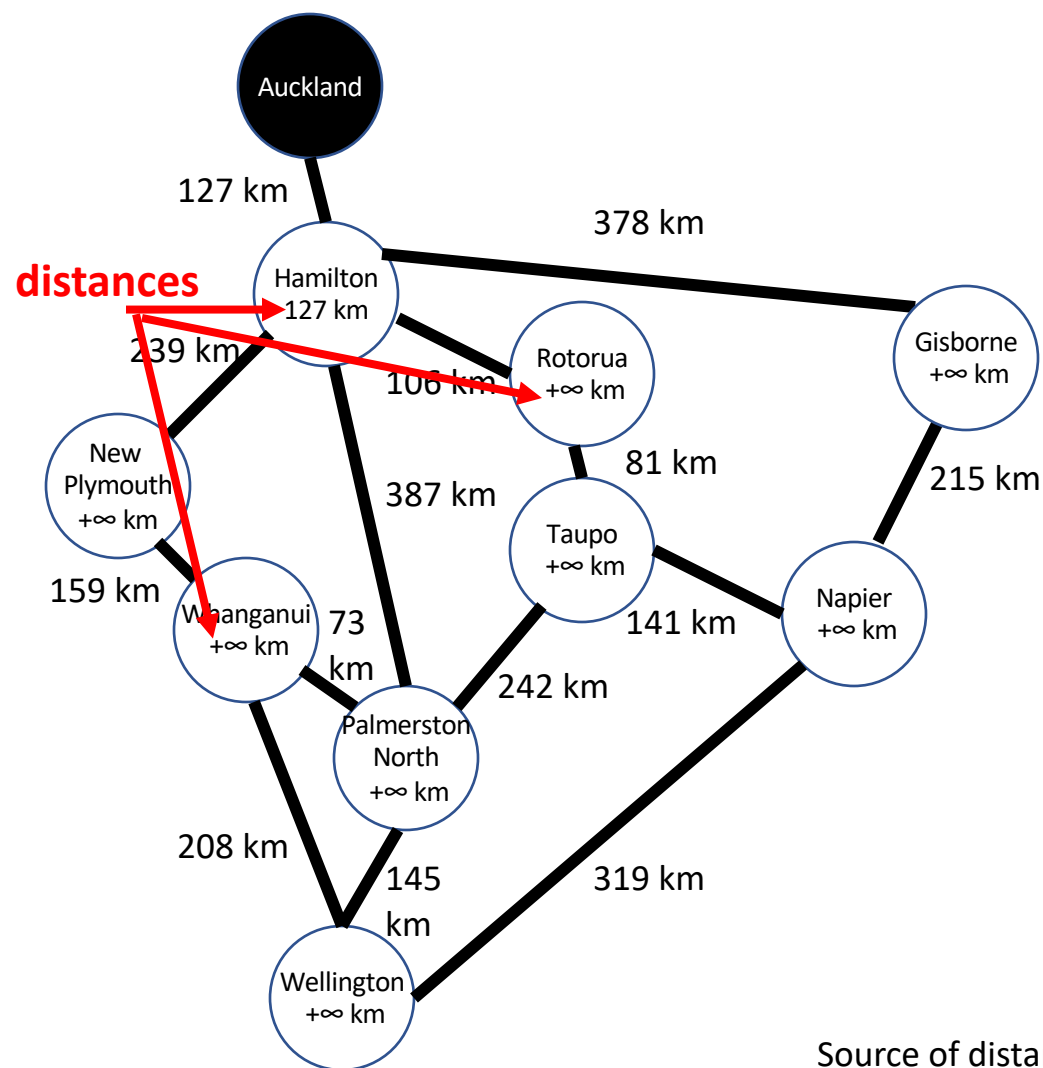
BLACK	$dist[x]$
	S, 1, 2, 3, 4, 5
S	0, 1, 3, 6, $\infty$ , $\infty$
S, 1	0, 1, 3, 6, 4, $\infty$
S, 1, 2	0, 1, 3, 5, 4, 7
S, 1, 2, 4	0, 1, 3, 5, 4, 6
S, 1, 2, 4, 5	0, 1, 3, 5, 4, 6

# Example: Dijkstra at Work



Source of distances: Google Maps

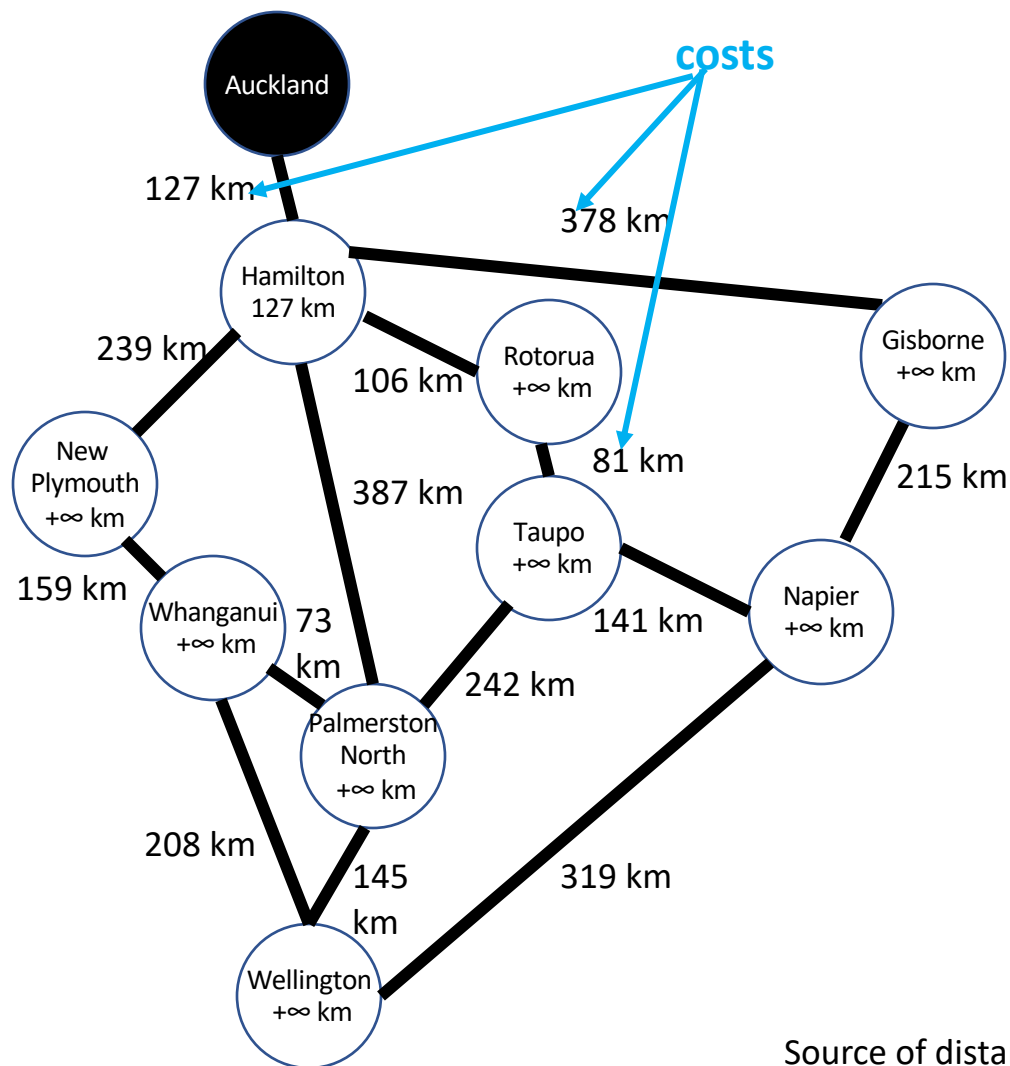
# Example: Dijkstra at Work



Source of distances: Google Maps

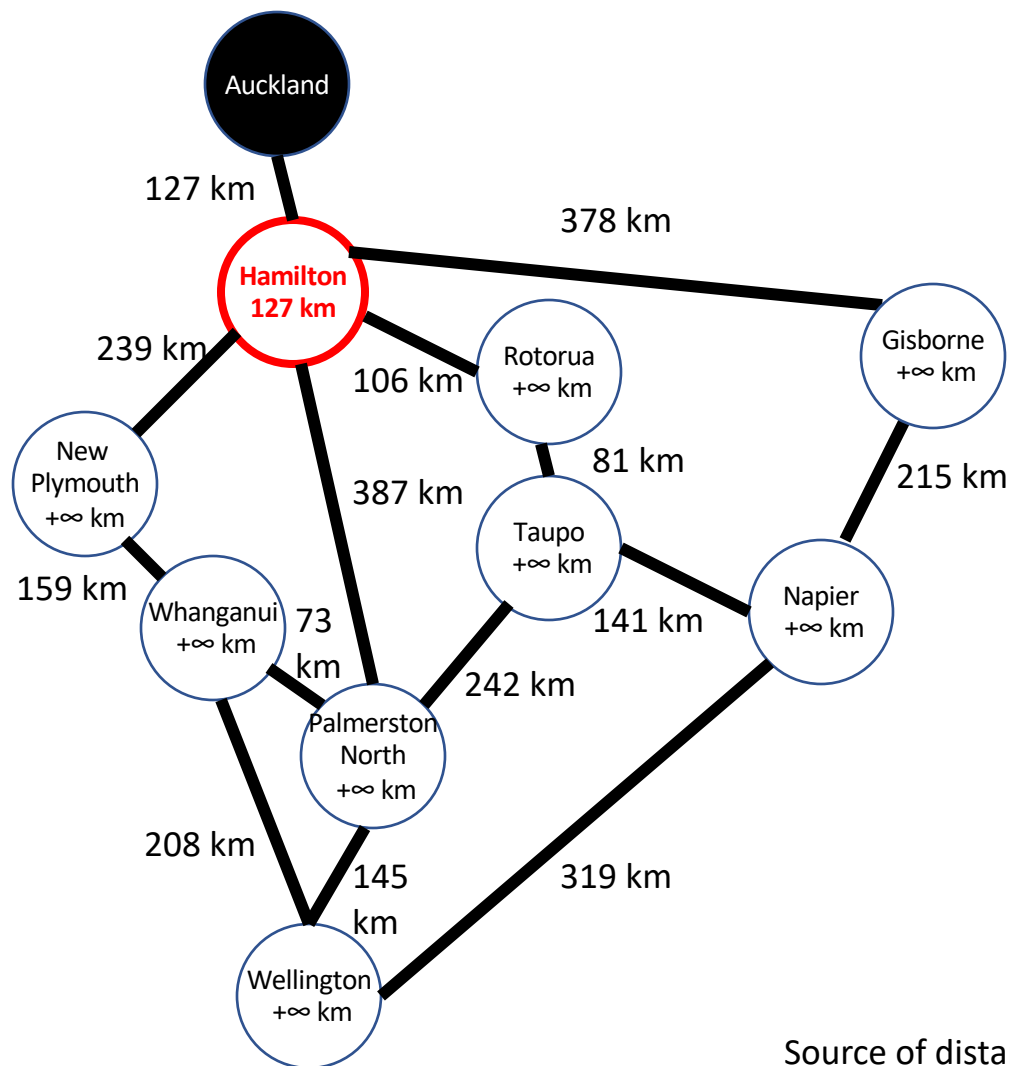


# Example: Dijkstra at Work



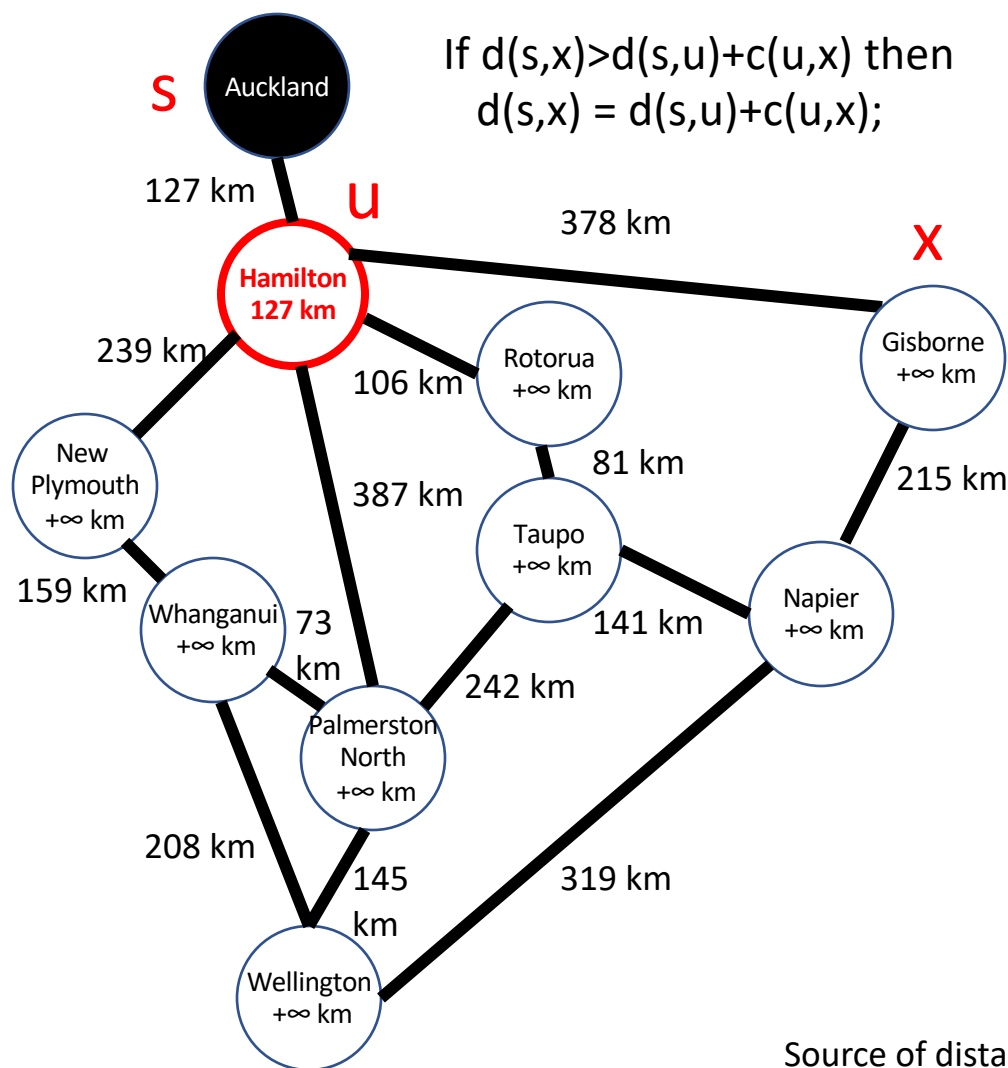
Source of distances: Google Maps

# Example: Dijkstra at Work



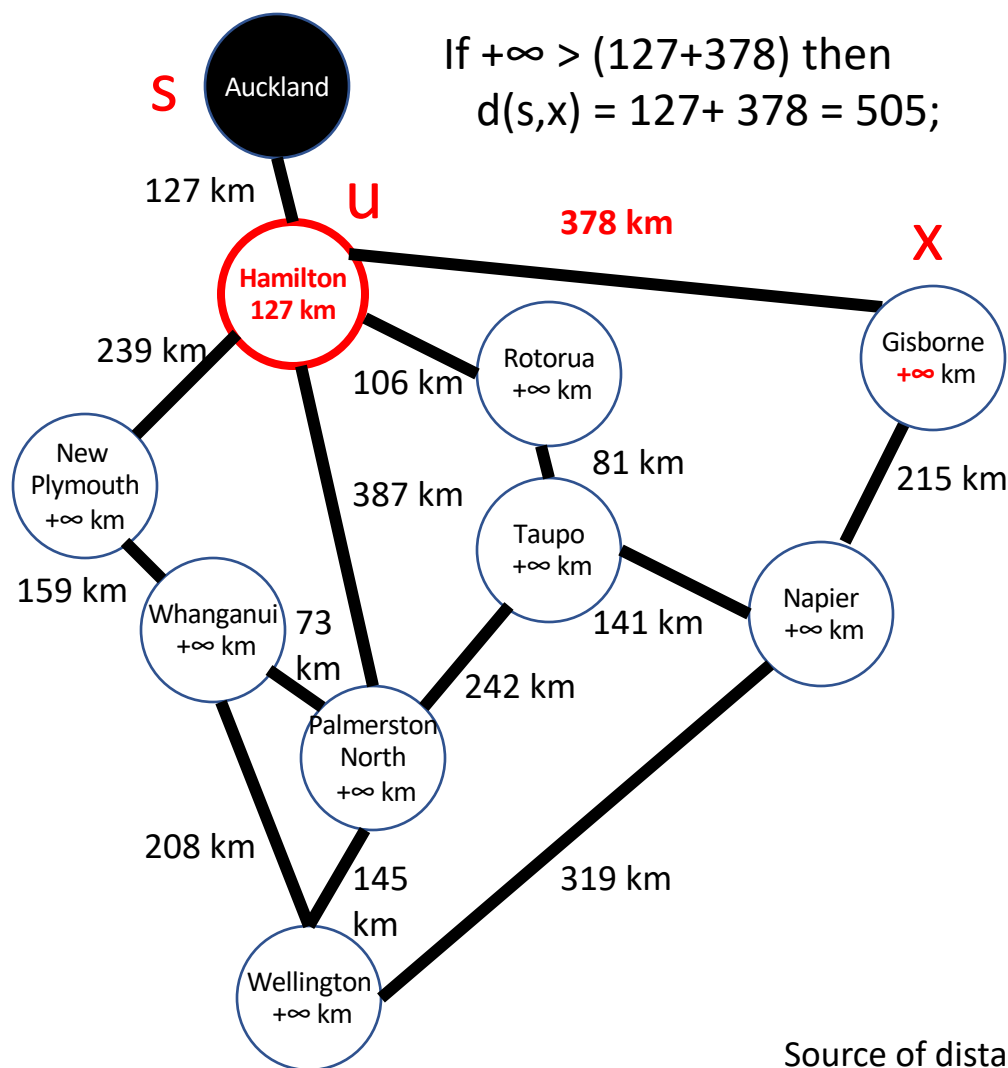
Source of distances: Google Maps

# Example: Dijkstra at Work



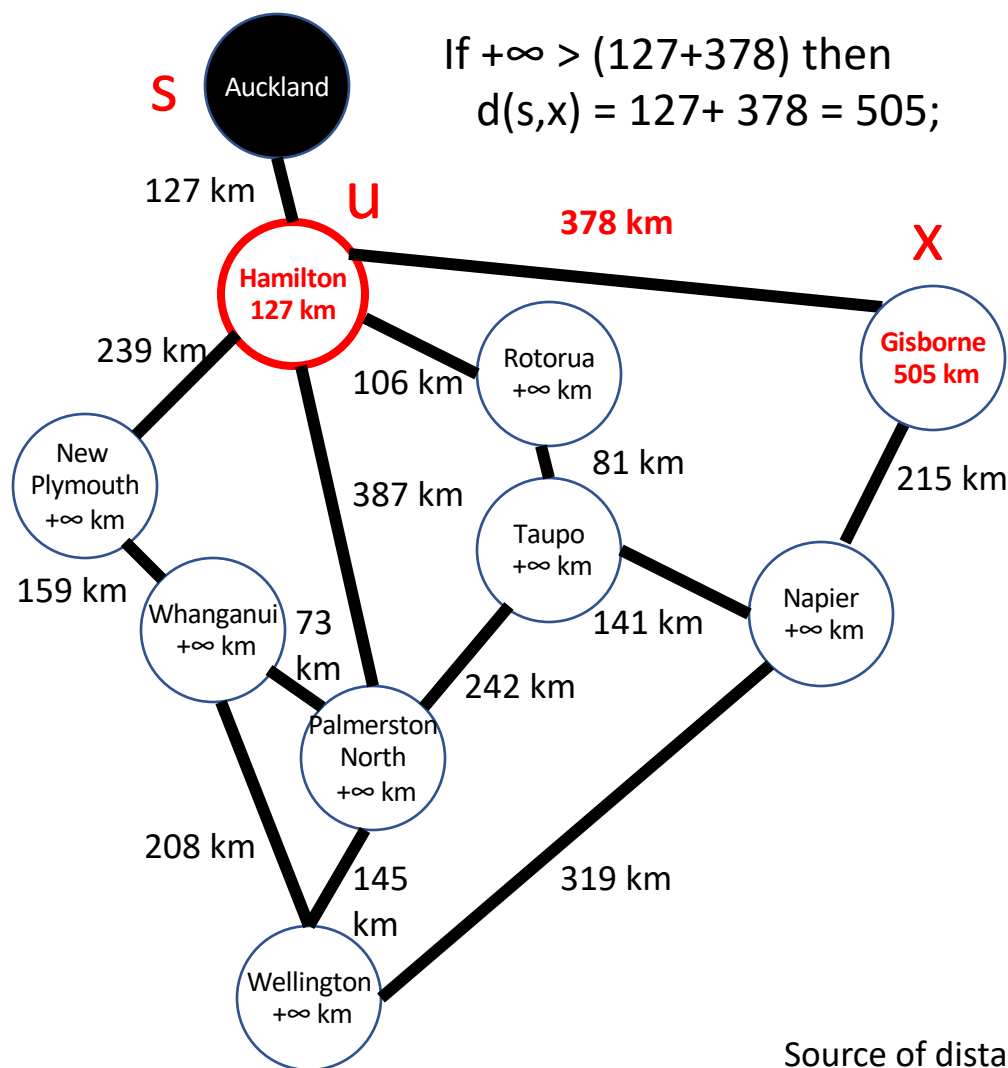
Source of distances: Google Maps

# Example: Dijkstra at Work



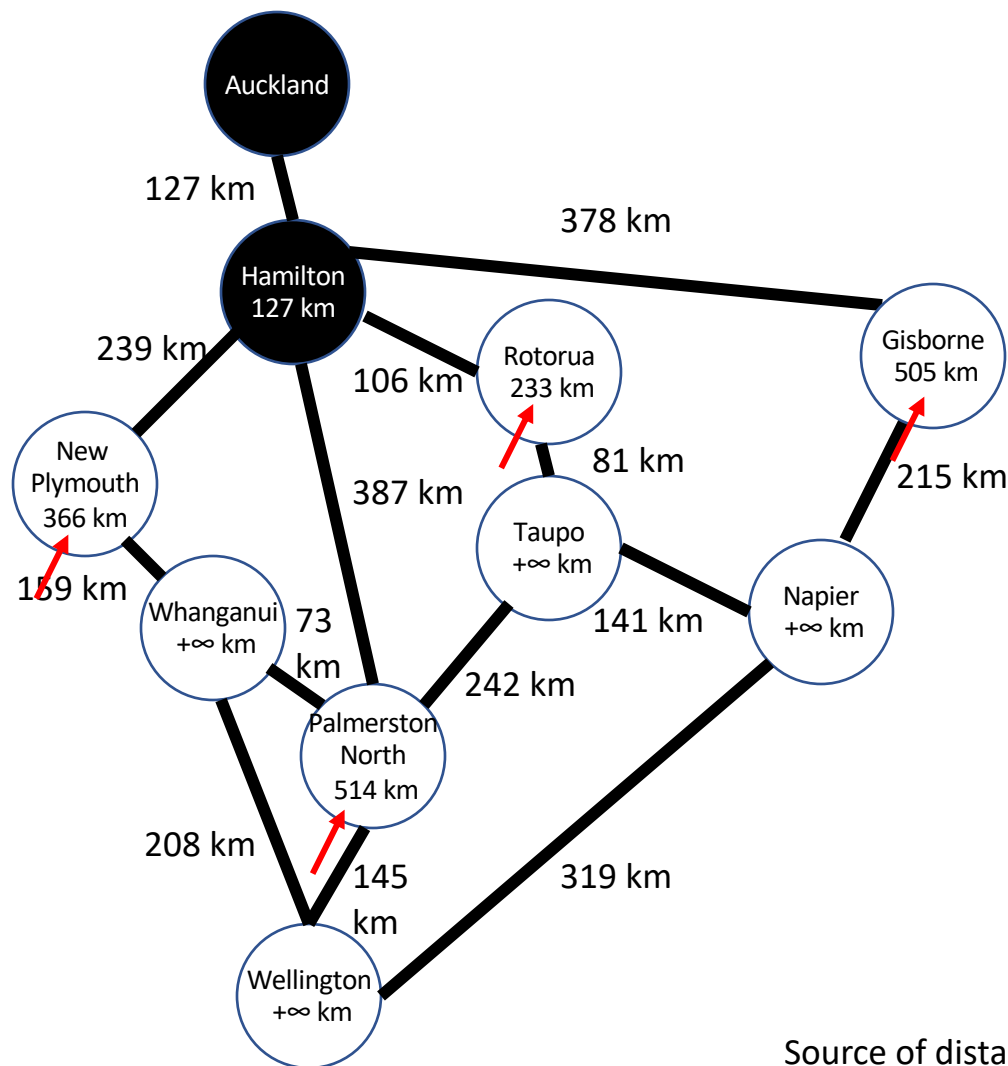
Source of distances: Google Maps

# Example: Dijkstra at Work



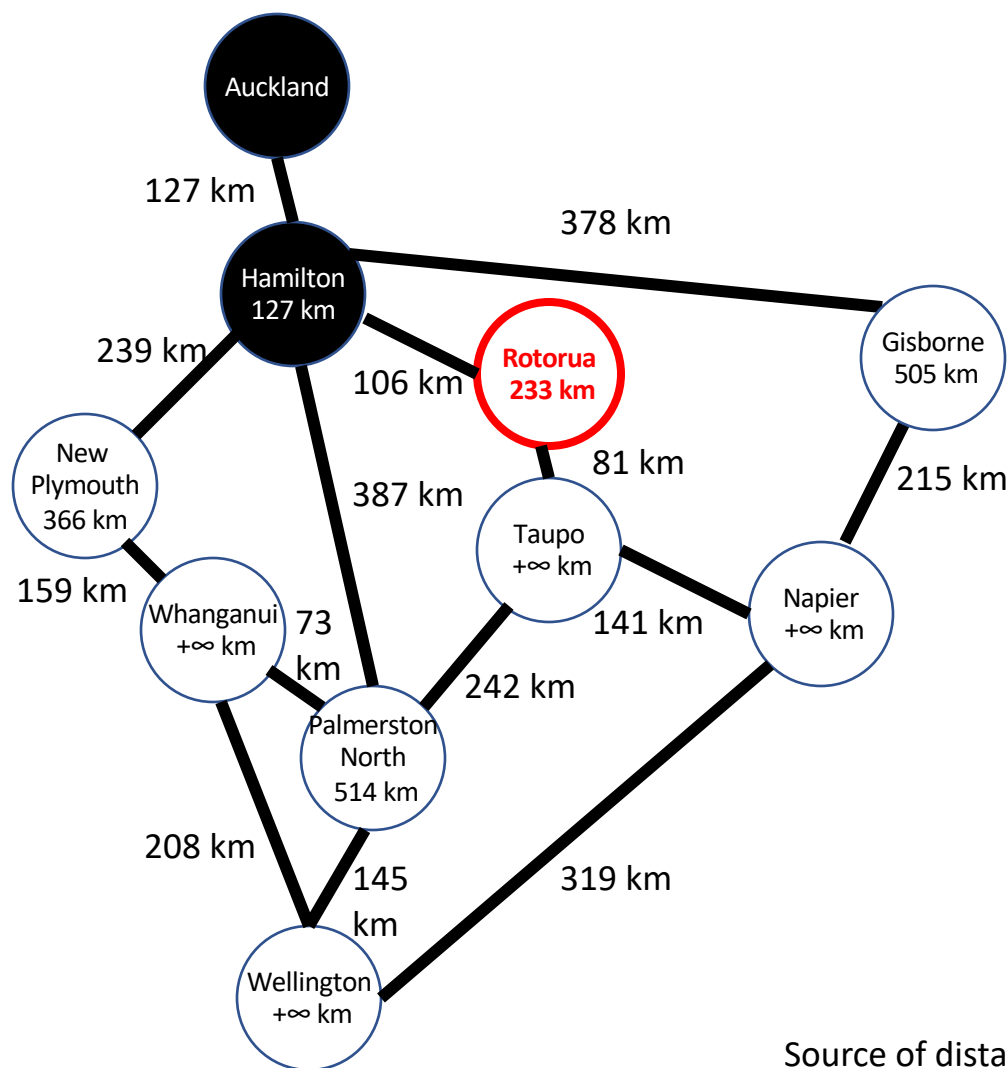
Source of distances: Google Maps

# Example: Dijkstra at Work



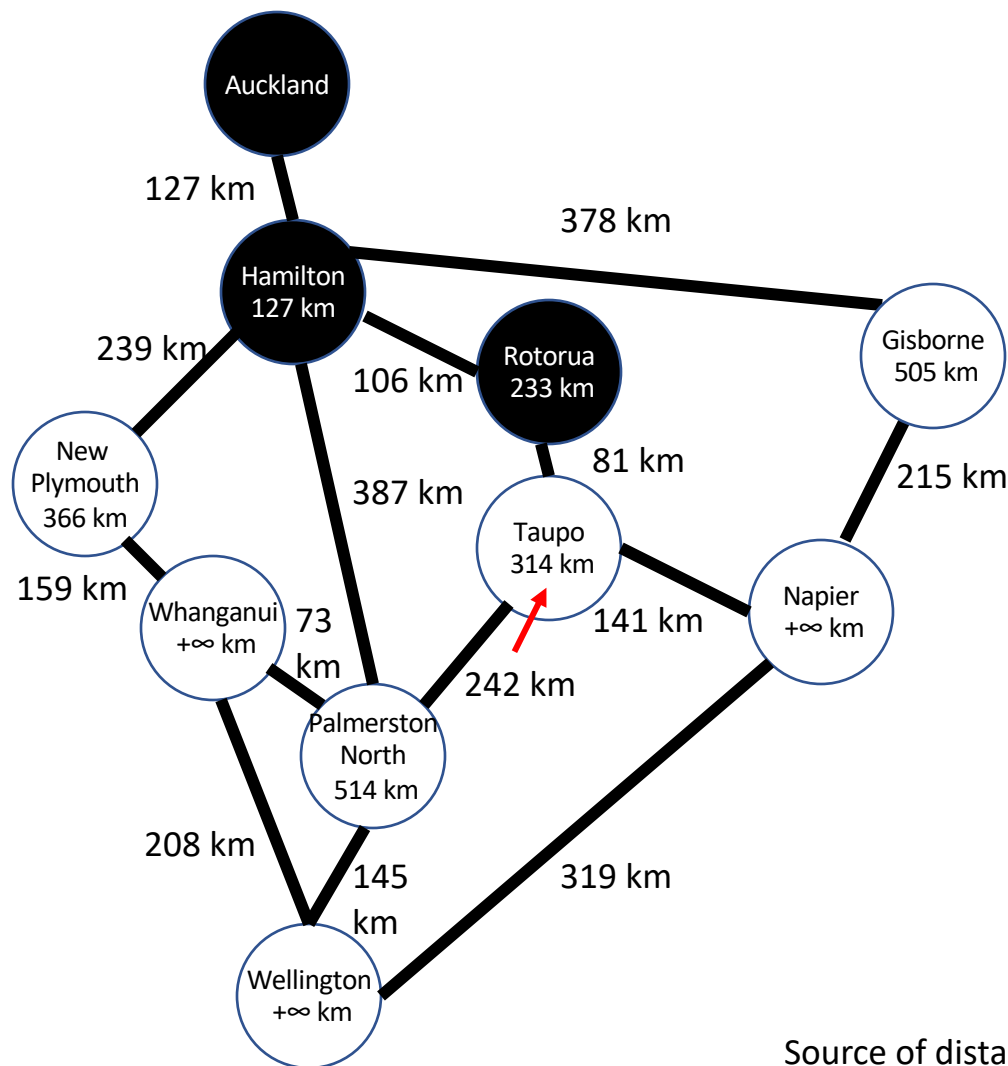
Source of distances: Google Maps

# Example: Dijkstra at Work



Source of distances: Google Maps

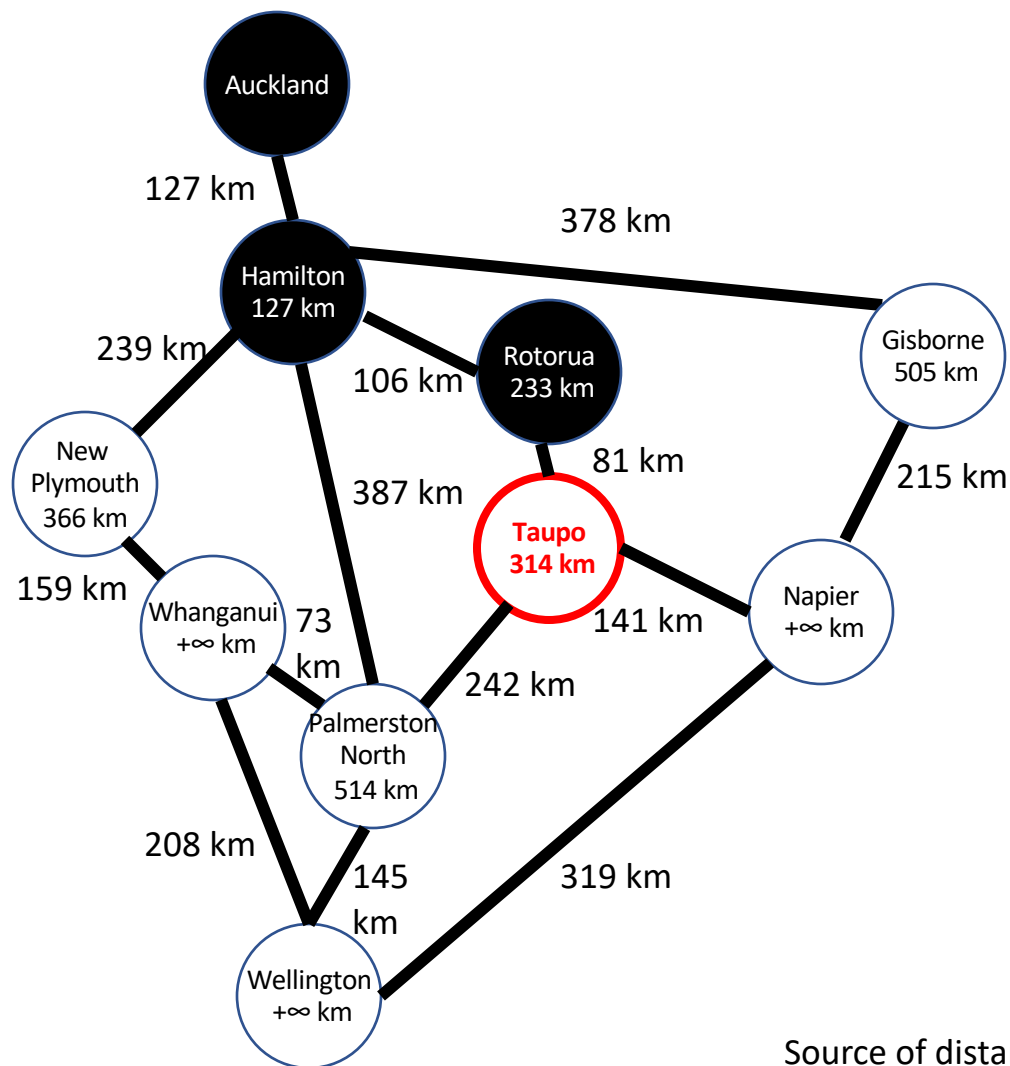
# Example: Dijkstra at Work



Source of distances: Google Maps

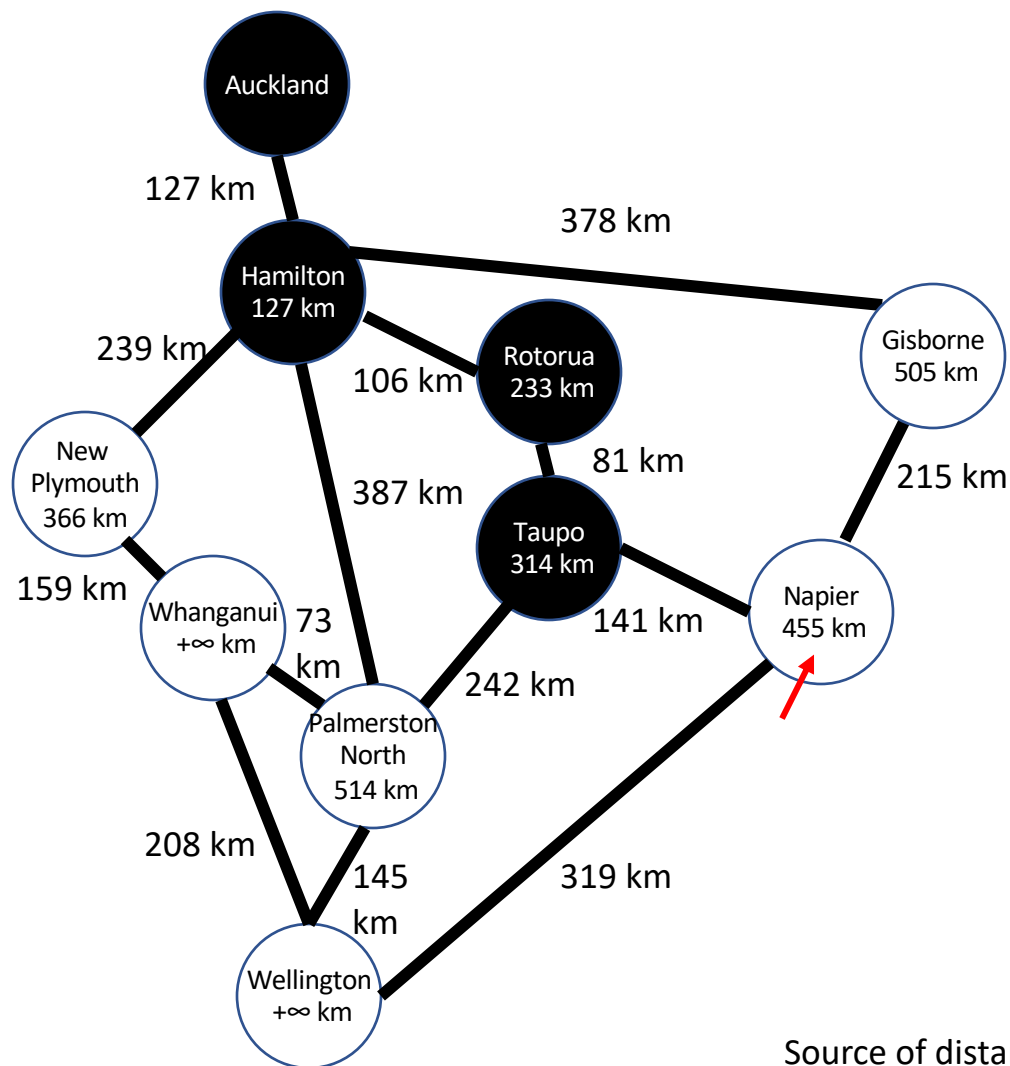


# Example: Dijkstra at Work



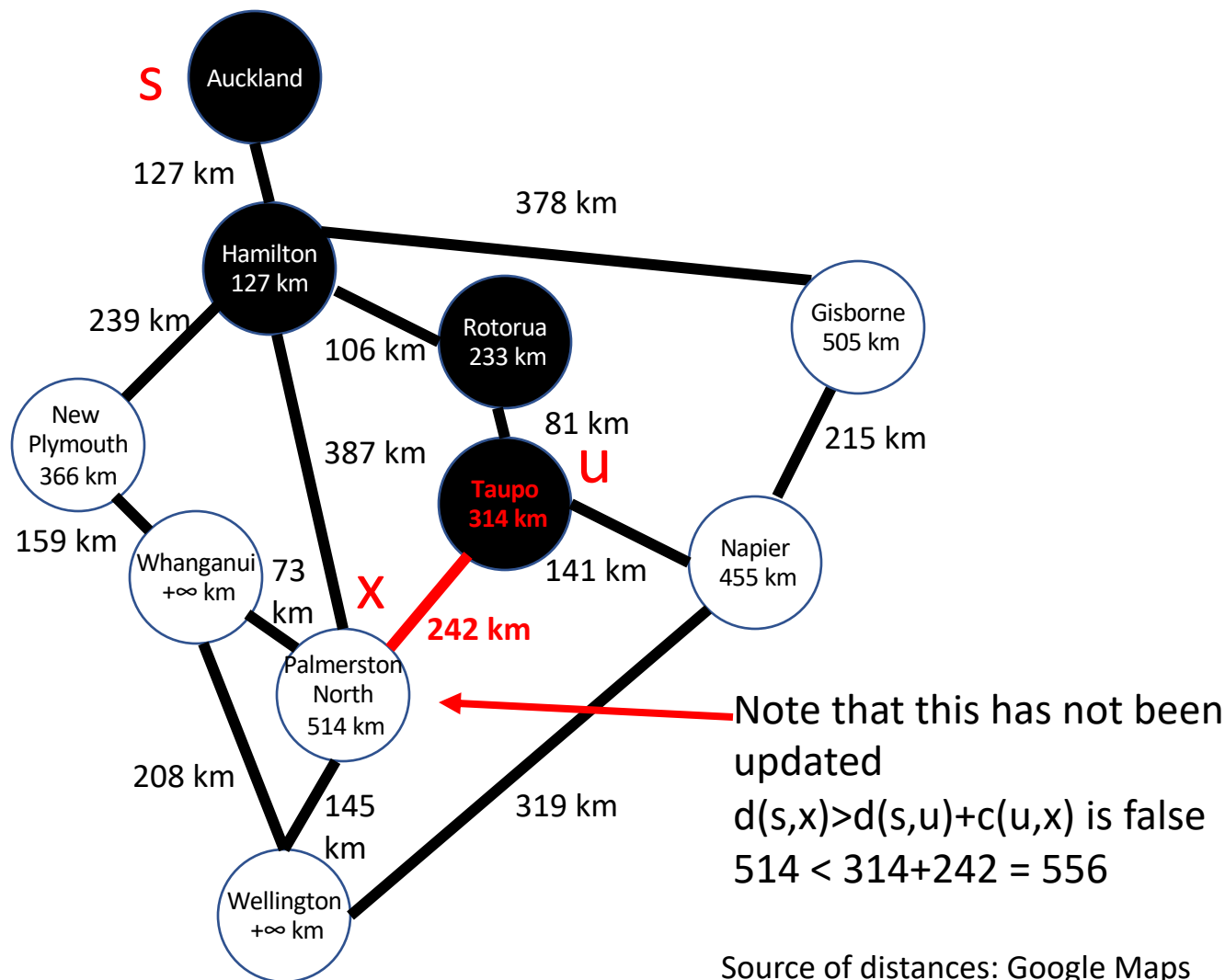
Source of distances: Google Maps

# Example: Dijkstra at Work



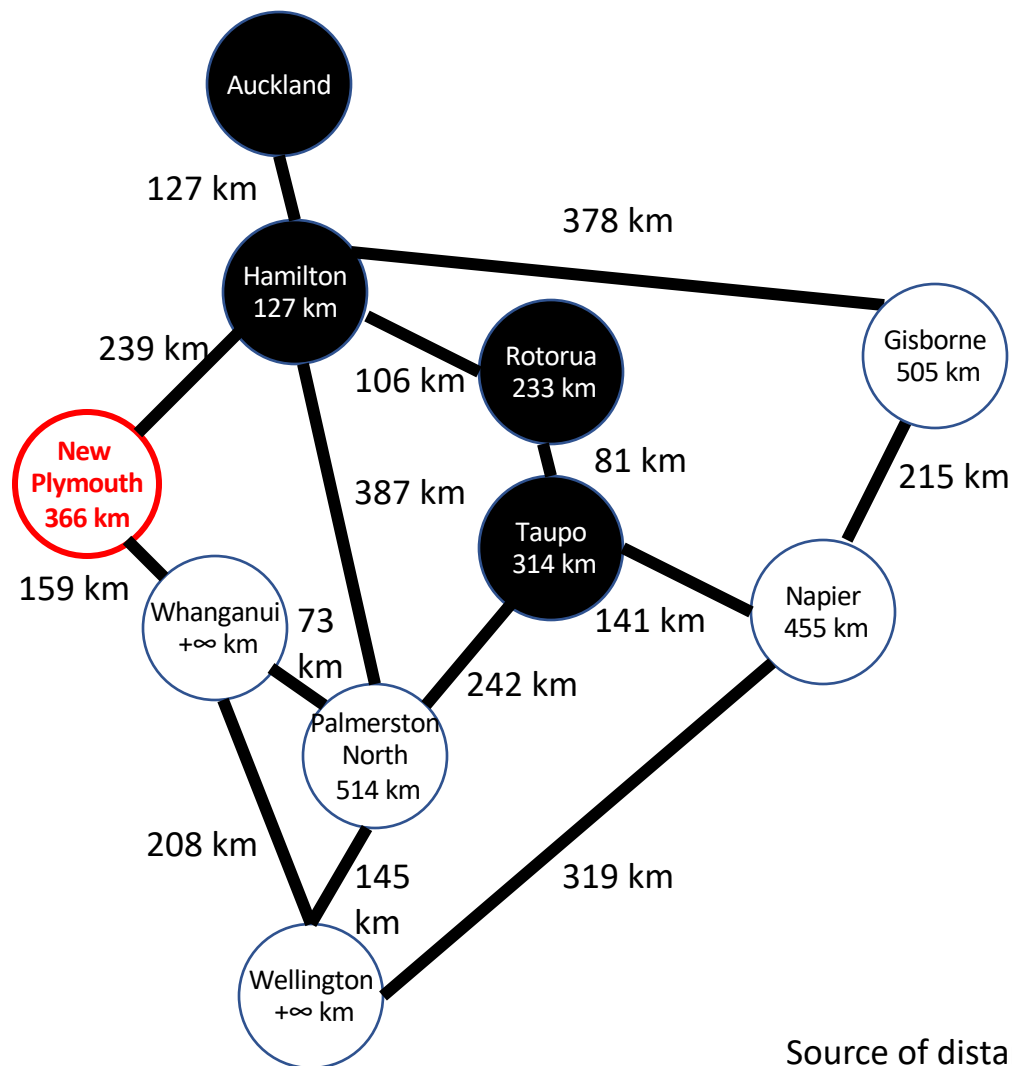
Source of distances: Google Maps

# Example: Dijkstra at Work



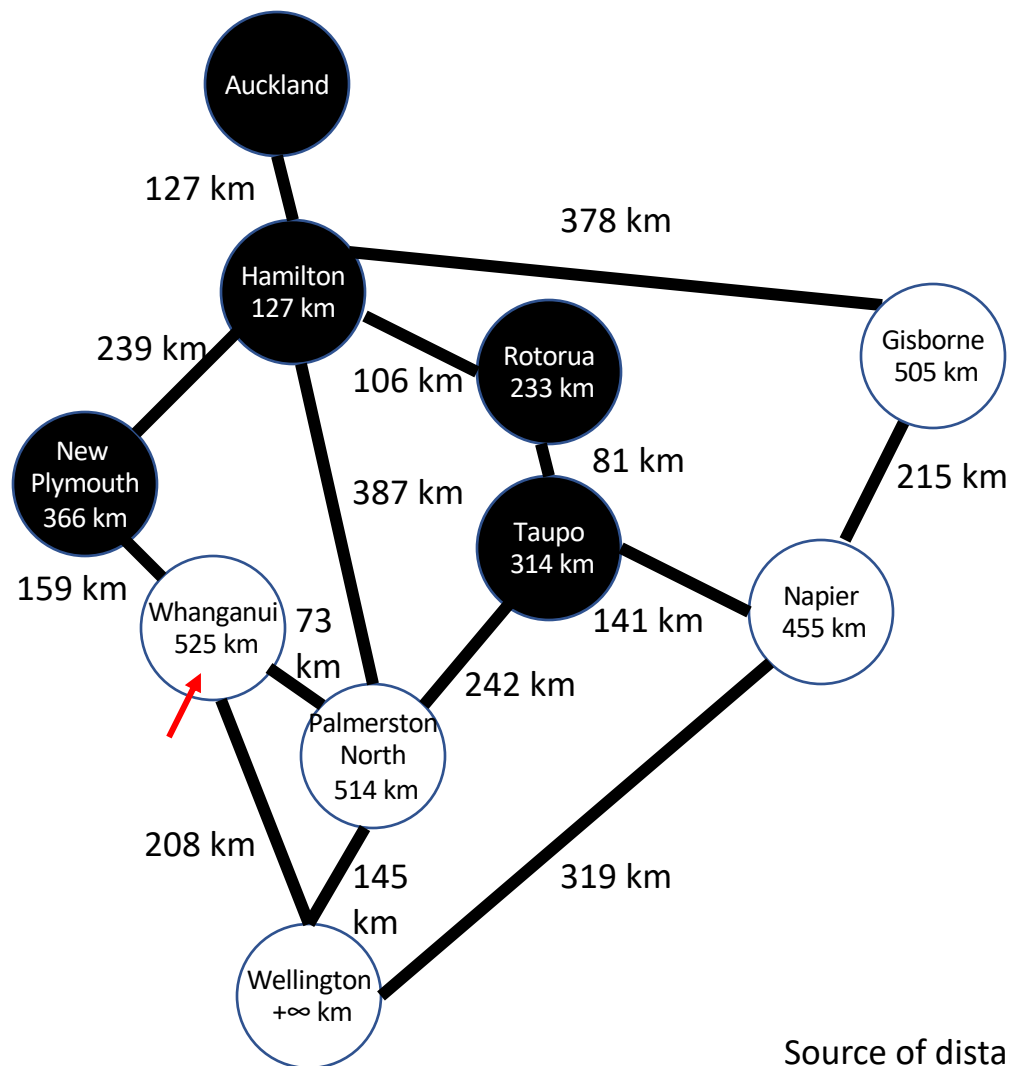
Source of distances: Google Maps

# Example: Dijkstra at Work



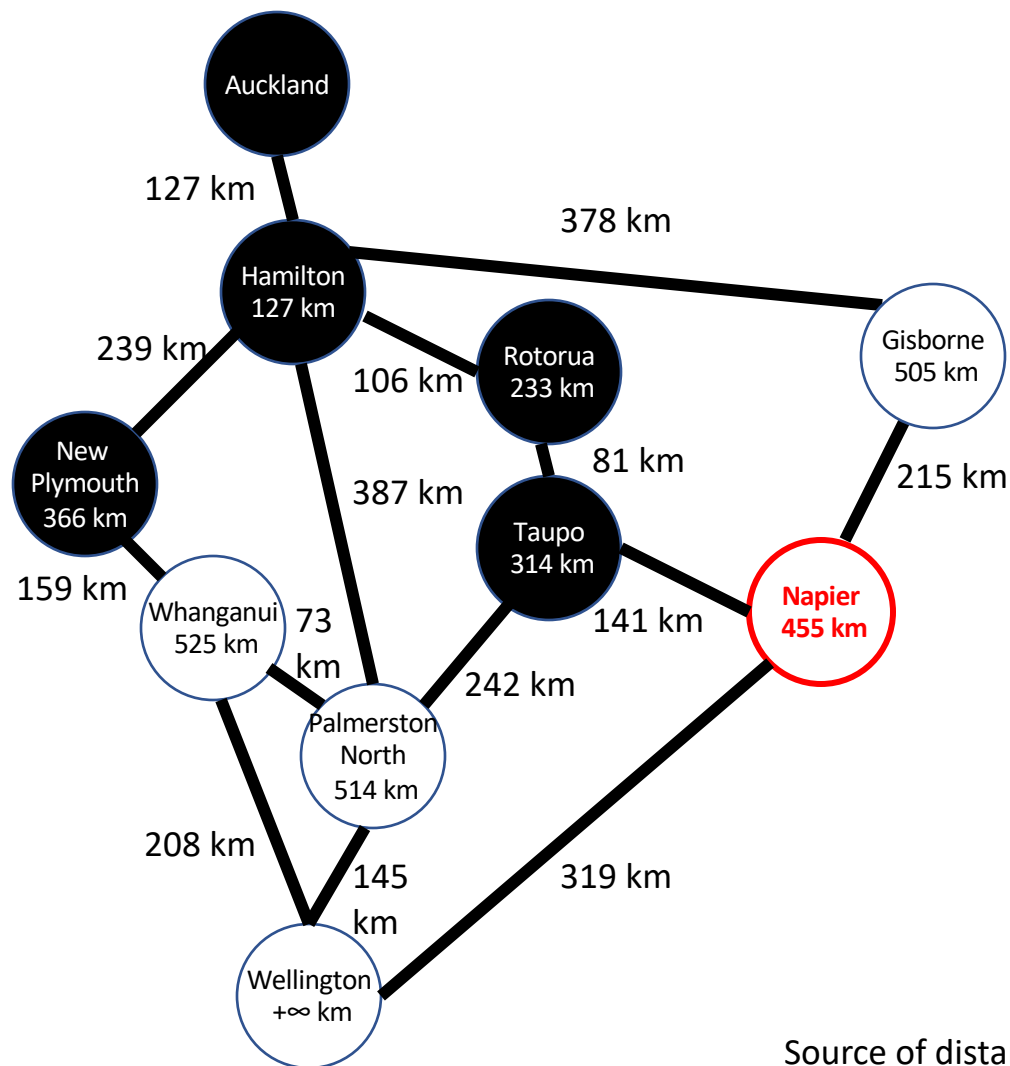
Source of distances: Google Maps

# Example: Dijkstra at Work



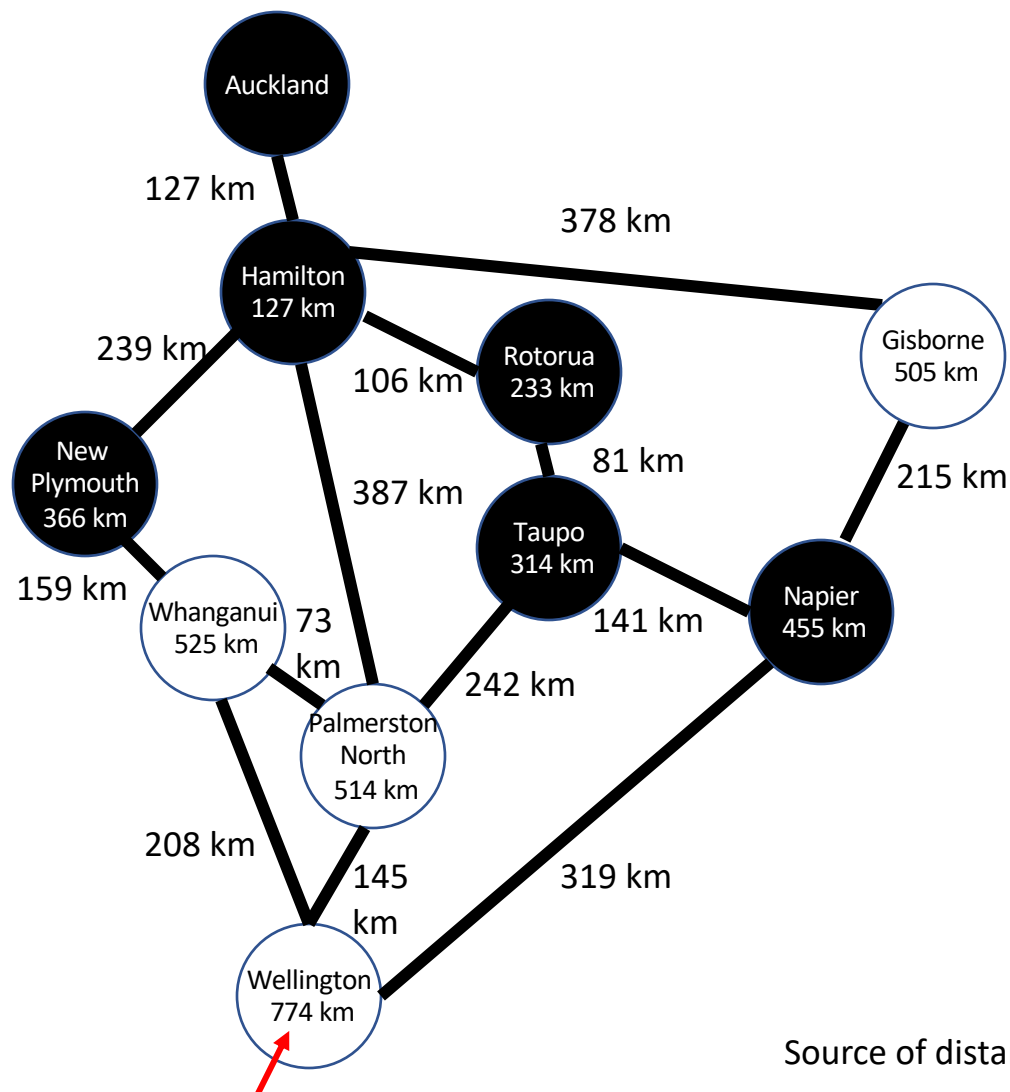
Source of distances: Google Maps

# Example: Dijkstra at Work

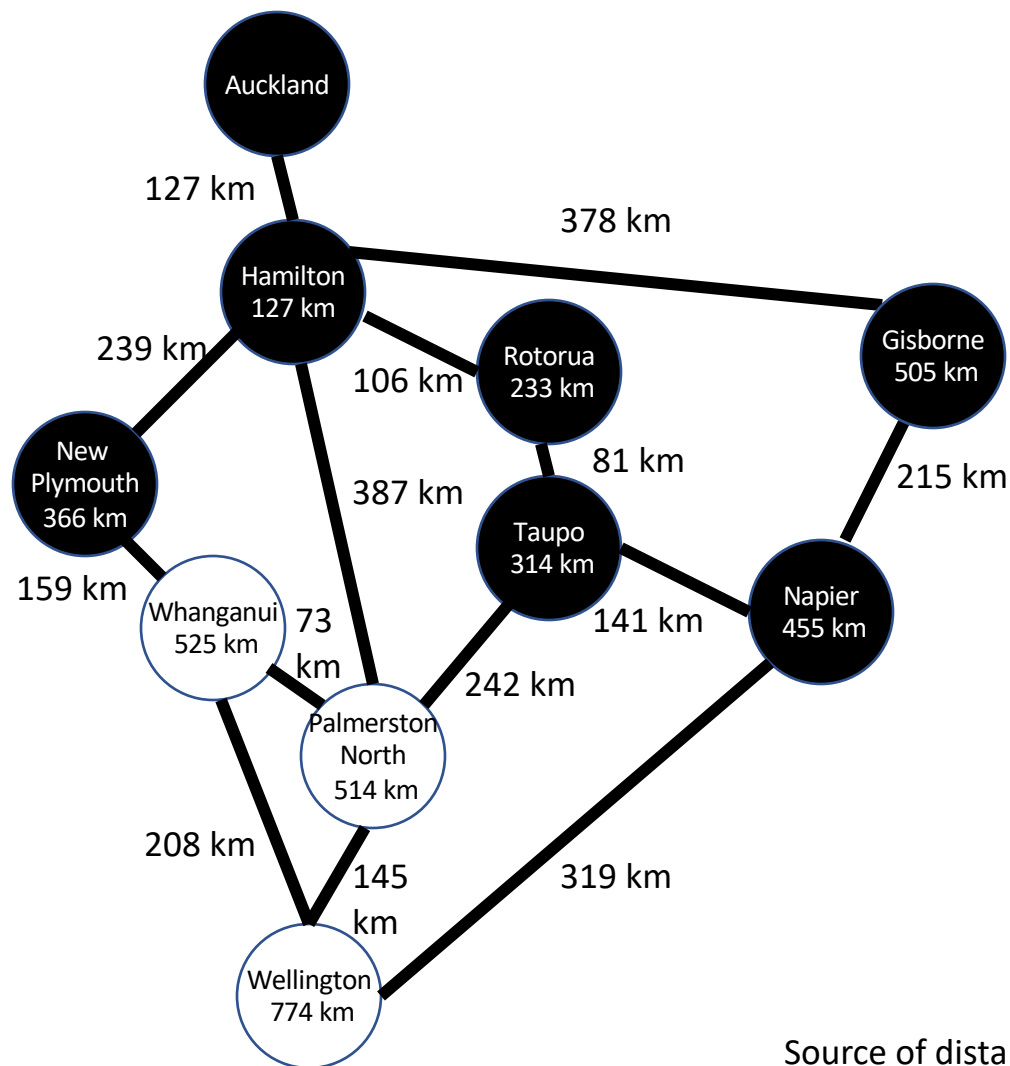


Source of distances: Google Maps

# Example: Dijkstra at Work



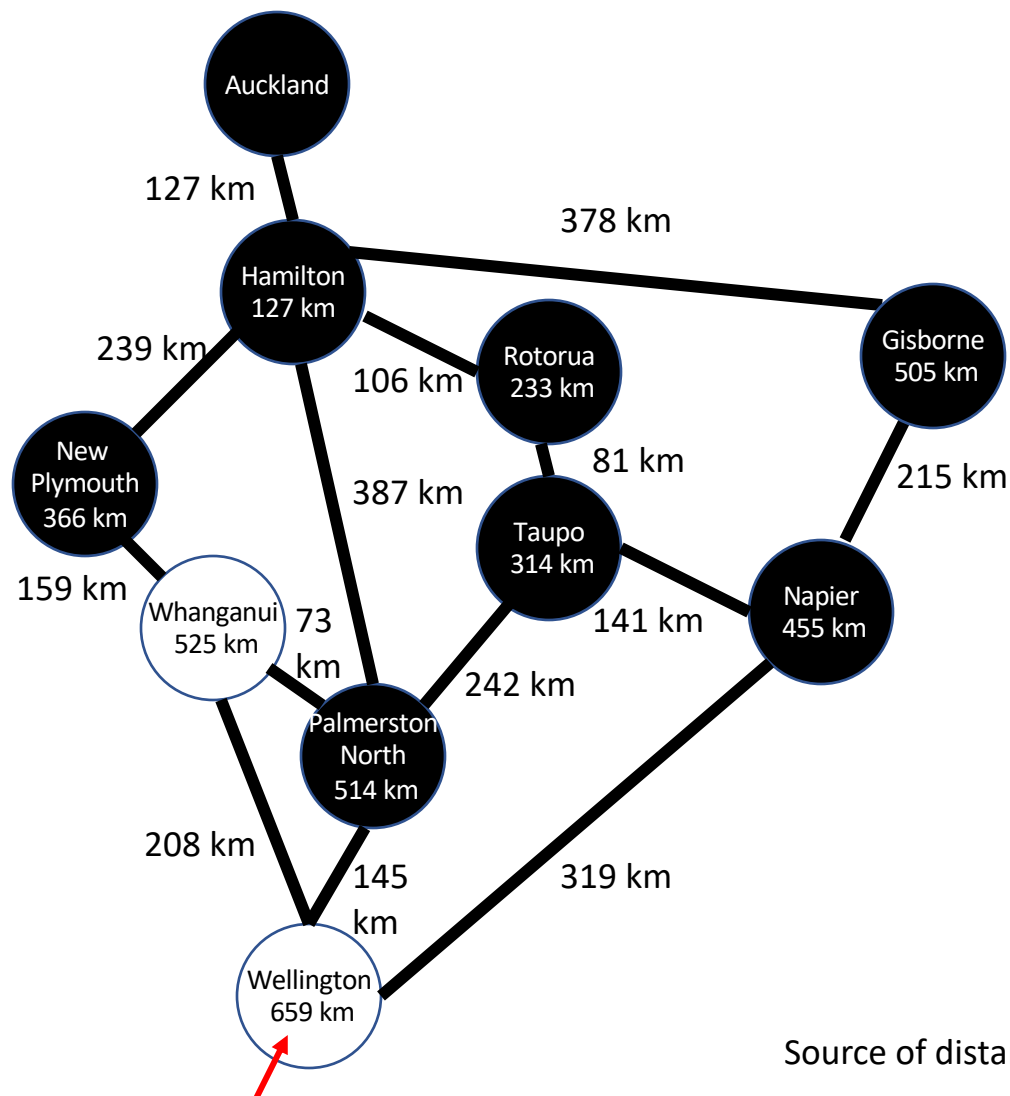
# Example: Dijkstra at Work



Source of distances: Google Maps

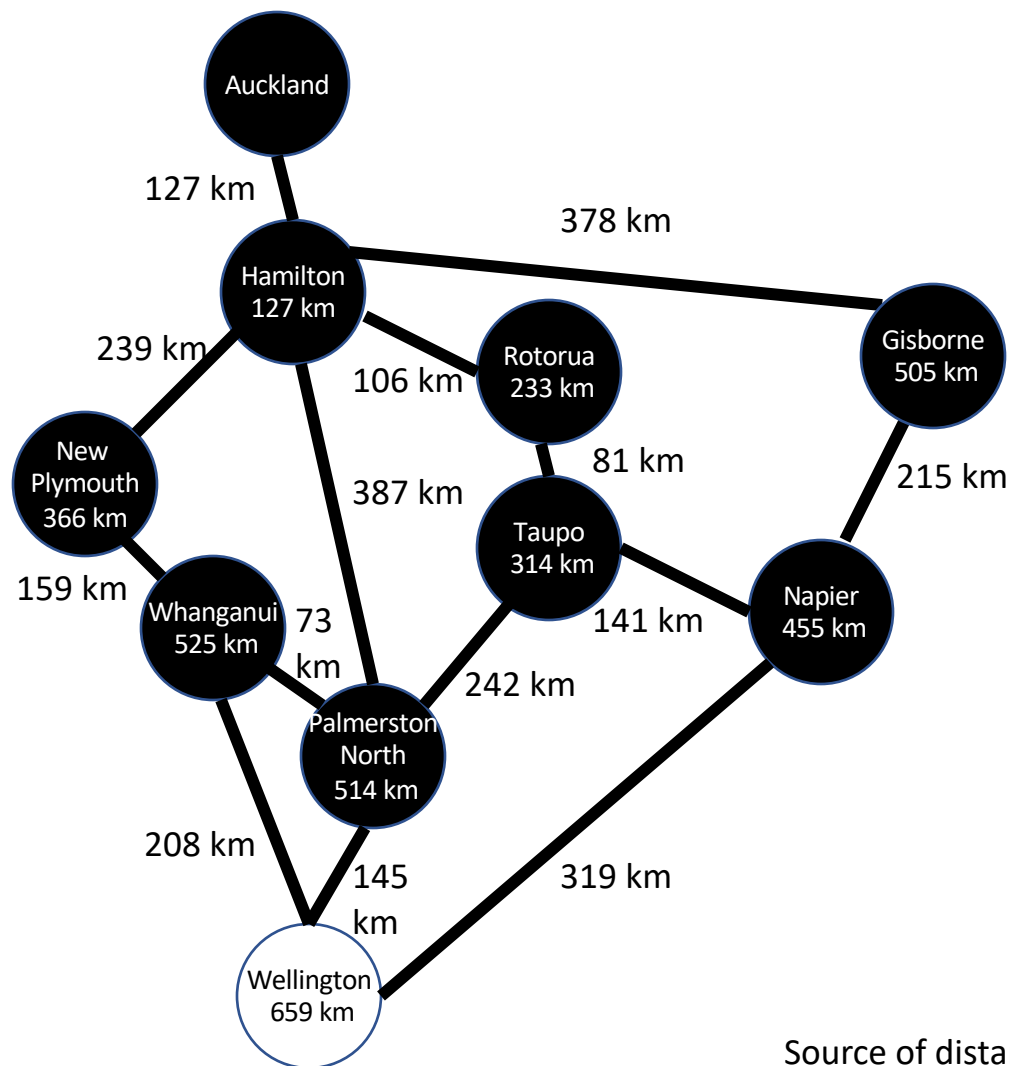


# Example: Dijkstra at Work



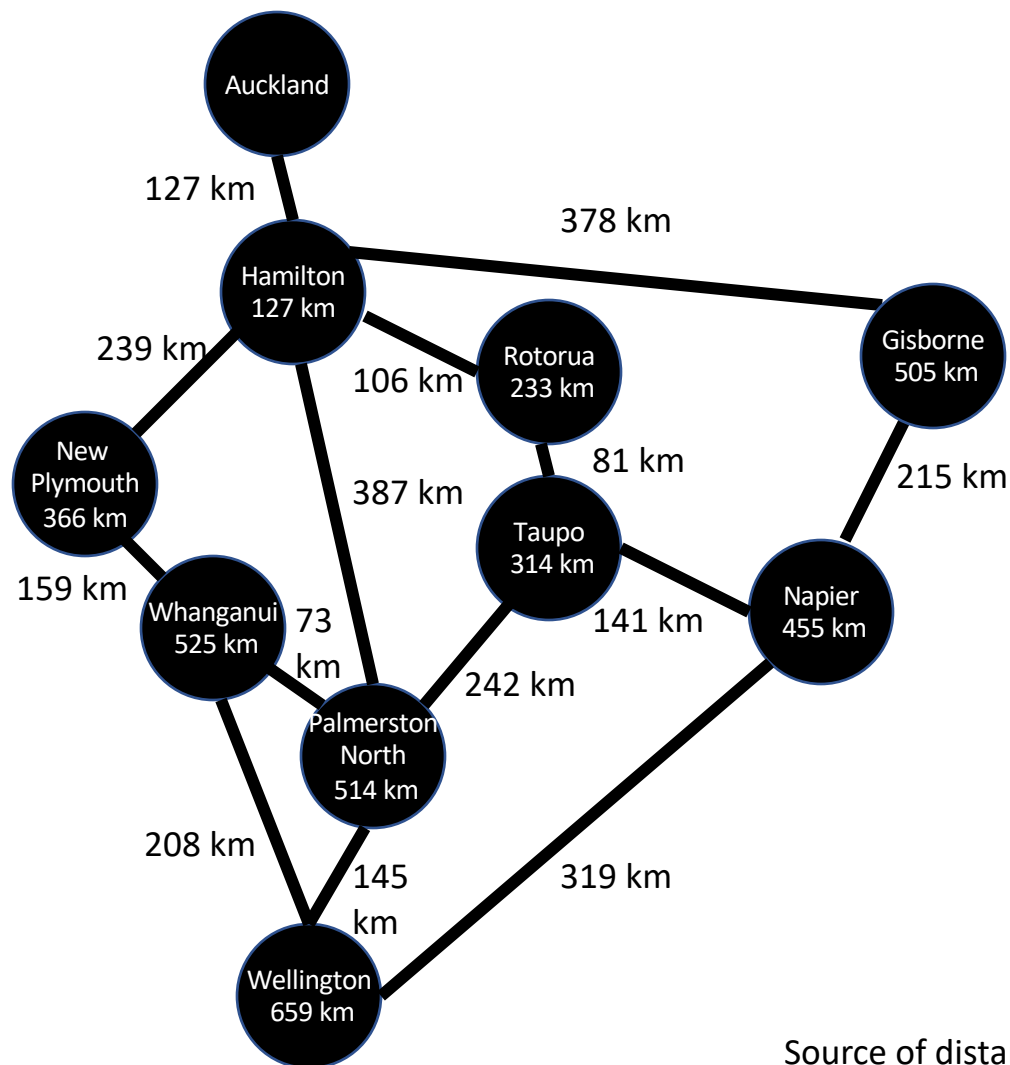
Source of distances: Google Maps

# Example: Dijkstra at Work



Source of distances: Google Maps

# Example: Dijkstra at Work

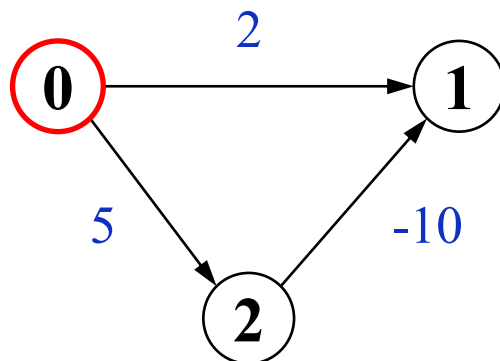


Source of distances: Google Maps

# Time Complexity Analysis: Dijkstra's Algorithm

- Complexity depends on data structures used, especially for priority queue;  $O((m + n) \log n)$  is possible.

**Fact:** Dijkstra's algorithm does not work with negative weights!



starting vertex **0** :

- shortest path from 0 to 2 is 5
- shortest path from 0 to 1 is  $5 + (-10) = -5$

**1<sup>st</sup> iteration:** **1** is colored black

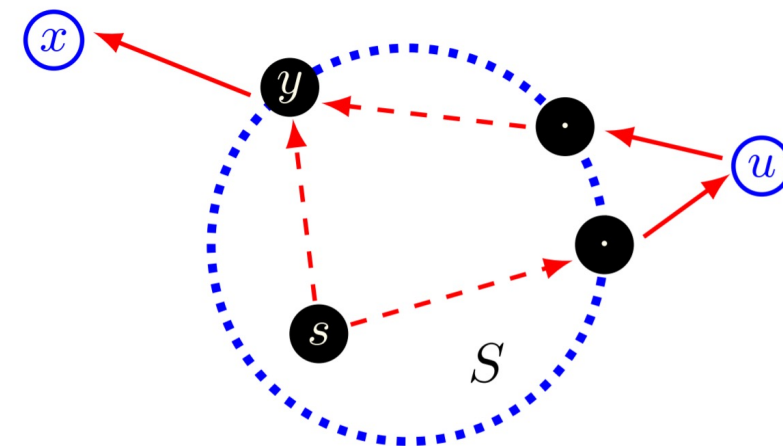
**2<sup>nd</sup> iteration:** **2** is colored black

Shortest path from 0 to 1 is not updated since 1 is already black!

# Why Dijkstra's Algorithm Works

- **Fact.** Suppose that all weights are non-negative. At the top of *while* loop, these properties hold for all  $x \in V(G)$ :
- **P1:**  $dist[x]$  is the minimum weight of an S-path to  $x$ .
- **P2:** if  $color[x]=BLACK$ ,  $dist[x]$  is the minimum weight.

Let an **S-path** be a path starting at node  $s$  and ending at node  $u$  with all nodes in  $S$  except **possibly**  $u$ .



## In Other Words ...

- P1 guarantees that, in each iteration of the algorithm, we find a minimum weight path from  $s$  to a vertex  $w \in V(G)$  that only uses vertices in  $S$  (black vertices) except possibly  $w$ .
- P2 guarantees that, for each black vertex  $w$ , we have already found a minimum weight path from  $s$  to  $w$ .
- Taken together, these two facts imply that Dijkstra's algorithm solves the single-source shortest path problem for weighted digraphs that do not have any **negative arc weights**.

# Proof by Mathematical Induction

- Induction on the number of times  $k$  of going through the while-loop. We use  $S_k$  to denote the set of  $S$  at the  $k$ -th loop:  $S_0 = \{s\}$ ;  $dist[s] = 0$

*P1:*  $dist[x]$  is the minimum weight of an  $S$ -path to  $x$ .

*P2:* if  $color[x] = \text{BLACK}$ ,  $dist[x]$  is the minimum weight.

- **Base case:** both P1 and P2 hold when  $k=0$
- If  $k=0$ , then only the starting vertex  $s$  is black and  $dist[s]=0$ . Hence, both properties hold.
- **Inductive hypothesis:** P1 and P2 hold for  $k \geq 0$ ;  $S_{k+1} = S_k \cup \{u\}$ .



# Inductive Step for P1

- Inductive hypothesis: P1 and P2 hold for  $k \geq 0$ ;  $S_{k+1} = S_k \cup \{u\}$ .

*P1:*  $dist[x]$  is the minimum weight of an  $S$ -path to  $x$ .

*P2:* if  $color[x] = \text{BLACK}$ ,  $dist[x]$  is the minimum weight.

- In iteration  $k+1$ , the algorithm colors  $u$  black.
- Let  $x \in V(G)$  and  $\gamma$  be any  $S_{k+1}$ -path from starting vertex  $s$  to  $x$ .
- We want to show that  $|\gamma| \geq dist[x]$ , then  $|\gamma| = dist[x]$  for min path

**Case 1:**  $u$  is not a vertex of  $\gamma$ .

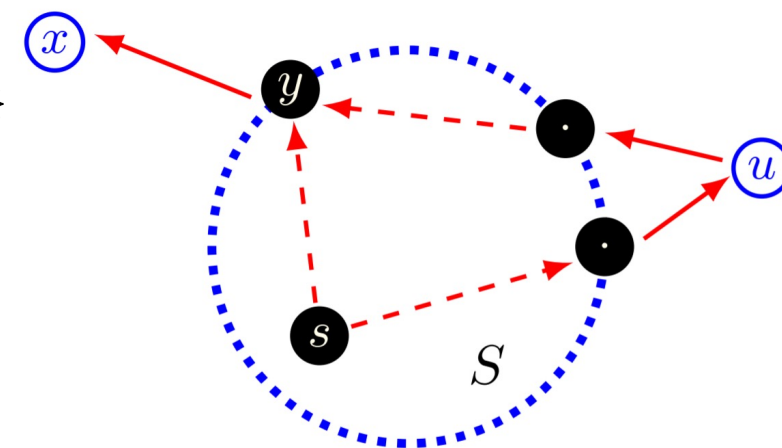
- Then  $\gamma$  is an  $S_k$ -path and the result follows from the induction hypothesis:  $dist[x]$  is the minimum weight of an  $S_{k+1}$ -path from the starting vertex to  $x$ .

# Inductive Step for P1 (Contd.)

- Inductive hypothesis: P1 and P2 hold for  $k \geq 0$ ;  $S_{k+1} = S_k \cup \{u\}$

*P1:*  $dist[x]$  is the minimum weight of an  $S$ -path to  $x$ .

*P2:* if  $color[x] = \text{BLACK}$ ,  $dist[x]$  is the minimum weight.



**Case 2:**  $u$  is a vertex of  $\gamma$ .

- Subcase 2a:  $\gamma = (s \dots u, x)$  Then let  $\gamma_1 = (s \dots u)$ . So  $|\gamma| = |\gamma_1| + c(u, x) \geq dist[x]$
- Subcase 2B:  $\gamma = (s \dots u \dots y, x)$

Let  $\beta$  be the minimum  $S_k$  path from  $s$  to  $y$  (not including  $u$ ):  $|\beta| = dist[y]$

By the induction hypothesis:

$$|\gamma| = |\gamma_1| + c(y, x) \geq |\beta| + c(y, x) = dist[y] + c(y, x) \geq dist[x].$$

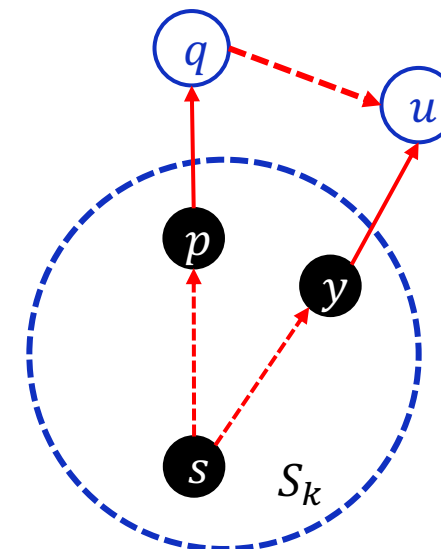
# Inductive Step for P2

- Inductive hypothesis: P1 and P2 hold for  $k \geq 0$ ;  $S_{k+1} = S_k \cup \{u\}$ .

*P1:*  $dist[x]$  is the minimum weight of an  $S$ -path to  $x$ .

*P2:* if  $color[x] = \text{BLACK}$ ,  $dist[x]$  is the minimum weight.

**Case 1:** If  $x \in S_{k+1}$  and  $x \neq u$ , P2 holds by hypothesis

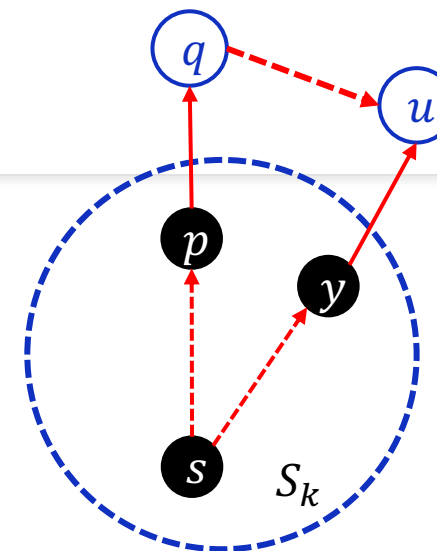


# Inductive Step for P2 (Contd.)

- Inductive hypothesis: P1 and P2 hold for  $k \geq 0$ ;  $S_{k+1} = S_k \cup \{u\}$ .

*P1*:  $dist[x]$  is the minimum weight of an  $S$ -path to  $x$ .

*P2*: if  $color[x] = \text{BLACK}$ ,  $dist[x]$  is the minimum weight.



**Case 2:** If  $x=u$ , for any  $S_k$ -path, we have  $dist[u]$  be the shortest by hypothesis. We only need to show that there is no shorter path to  $u$  that contains any node that is not in  $S_{k+1}$ .

Let  $\gamma = (s, \dots, p, q, \dots, u)$  be any of the above path to  $u$ , and  $q$  is the first node in  $\gamma$  that leaves  $S_k$ . Let  $\gamma_1 = (s, \dots, p)$ ,  $\gamma_2 = (q, \dots, u)$

- By hypothesis (P1):  $|\gamma_1| + c(p, q) \geq dist[q]$  ( $\gamma_1$  is an  $S_k$  path)
- By non-negative weights:  $|\gamma_2| \geq 0$

By 1 and 2, we have  $|\gamma| = |\gamma_1| + c(p, q) + |\gamma_2| \geq dist[q]$ . At time  $k+1$ , we choose  $u$  not  $q$ , this implies  $dist[q] \geq dist[u]$ , thus  $|\gamma| \geq dist[u]$ .

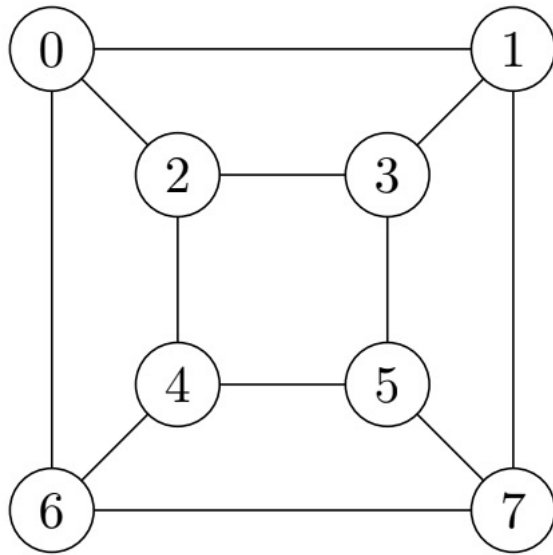
**Example 28.3.** Draw the weighted graph given by the weighted matrix below.

$$\begin{bmatrix} 0 & 3 & 4 & 0 \\ 3 & 0 & 1 & 3 \\ 4 & 1 & 0 & 2 \\ 0 & 3 & 2 & 0 \end{bmatrix}$$

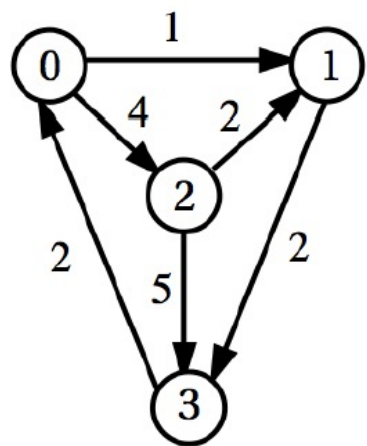
Draw the weighted digraph given by the weighted list representation below.

0	1	3	2	4
1	0	2	3	2
2	1	3		
3	2	1		

**Example 28.7.** What is the diameter of the 3-cube in Example 28.2?



**Example 28.11.** An application of Dijkstra’s algorithm on the digraph below for each starting vertex  $s$ . Complete the table for the starting vertex 2.



The table illustrates that the distance array is updated at most  $n - 1$  times (only before a new vertex is selected and added to  $S$ ). Thus we could have omitted the lines with  $S = \{0, 1, 2, 3\}$ .

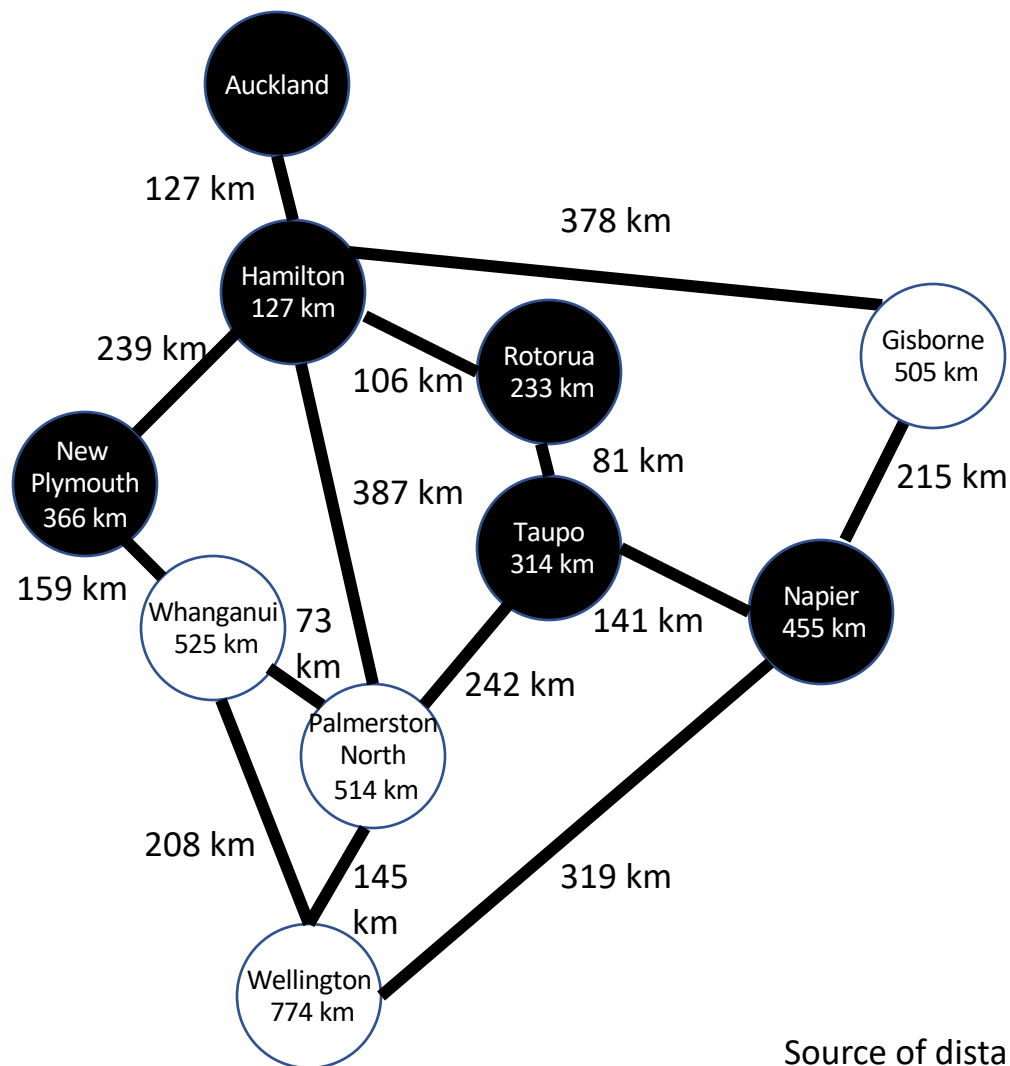
current $S \subseteq V$	distance vector dist
$\{0\}$ $\{0, 1\}$ $\{0, 1, 3\}$ $\{0, 1, 2, 3\}$	$0, 1, 4, \infty$ $0, 1, 4, 3$ $0, 1, 4, 3$ $0, 1, 4, 3$
$\{1\}$ $\{1, 3\}$ $\{0, 1, 3\}$ $\{0, 1, 2, 3\}$	$\infty, 0, \infty, 2$ $4, 0, \infty, 2$ $4, 0, 8, 2$ $4, 0, 8, 2$
$\{2\}$ $\{ \textcolor{red}{1}, \textcolor{blue}{2} \}$ $\{ \textcolor{red}{1}, \textcolor{blue}{2}, \textcolor{green}{3} \}$ $\{0, 1, 2, 3\}$	$\infty, \textcolor{red}{2}, 0, 5$ $\infty, 2, 0, \textcolor{green}{2} + \textcolor{green}{2}$ $2 + 2 + 2, 2, 0, 2 + 2$ $6, 2, 0, 4$
$\{3\}$ $\{0, 3\}$ $\{0, 1, 3\}$ $\{0, 1, 2, 3\}$	$2, \infty, \infty, 0$ $2, 3, 6, 0$ $2, 3, 6, 0$ $2, 3, 6, 0$

# Dijkstra Algorithm: Negative Weights

- One of the property of Dijkstra's algorithm is that once a node turns black its distance is not updated any longer
- This means that even if a new cheaper route is available, a black node's distance will not get updated – a case that would only be there when there are negative weights.

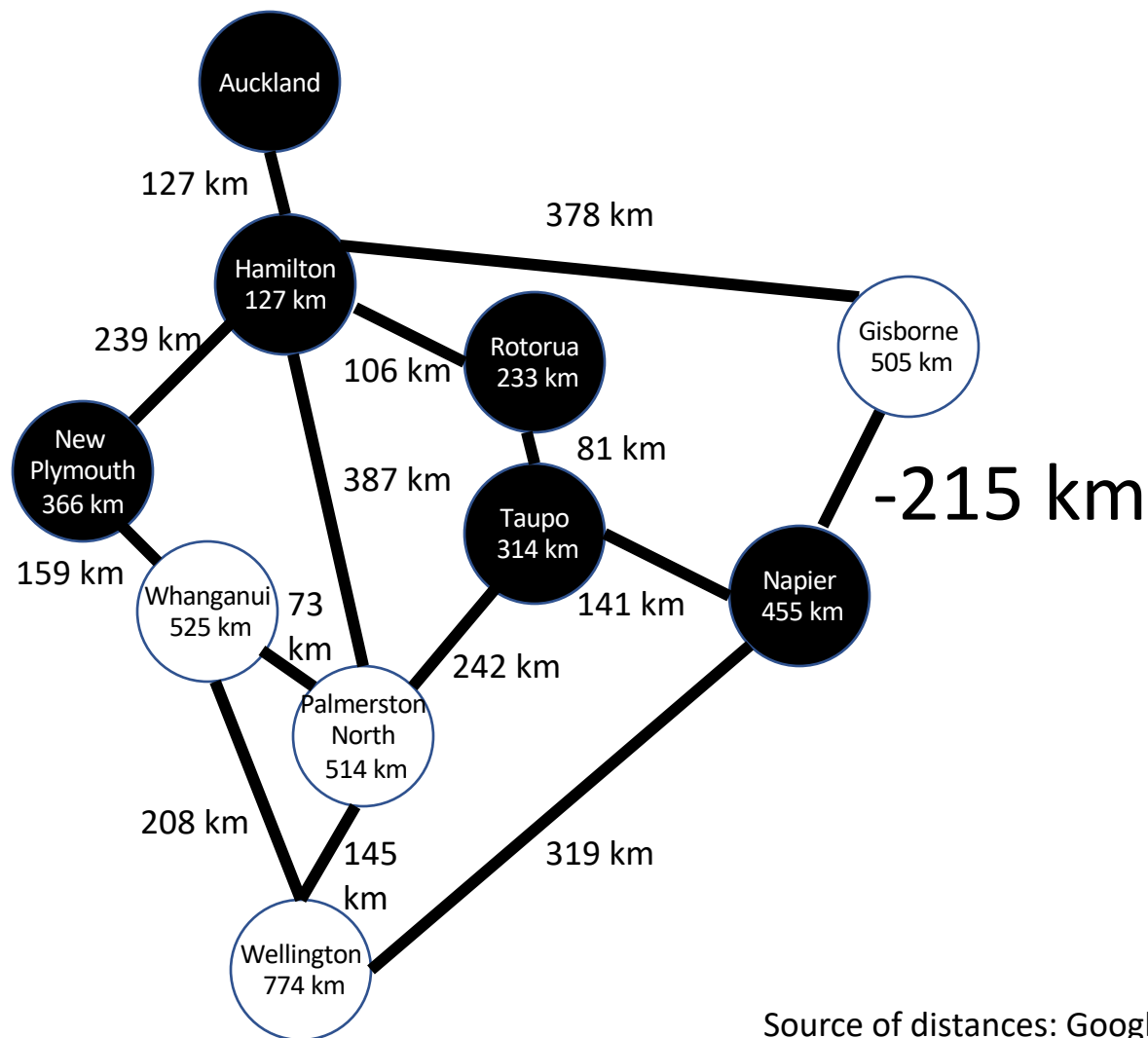


# Example: Dijkstra at Work



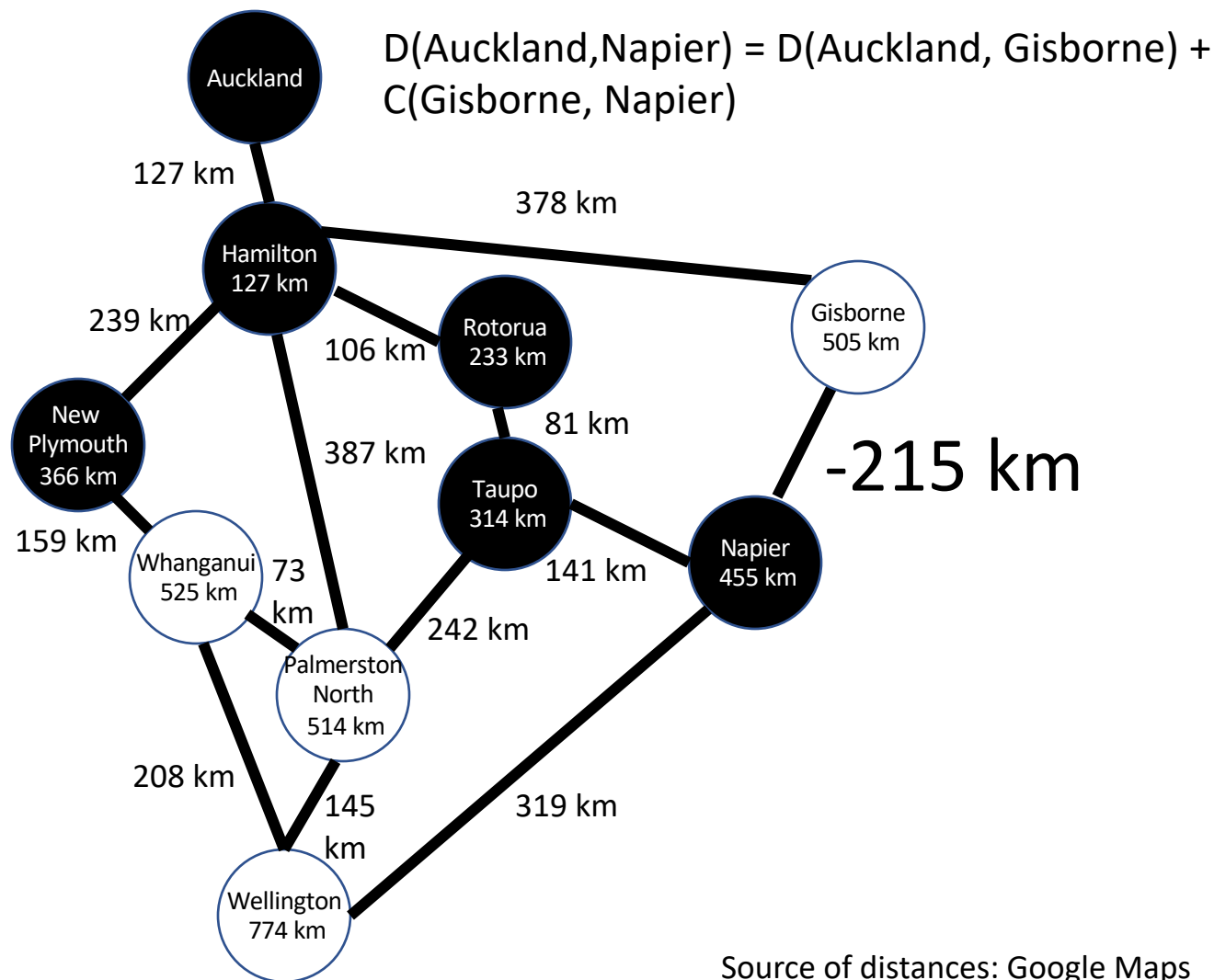
Source of distances: Google Maps

# Example: Adding a Negative Weight



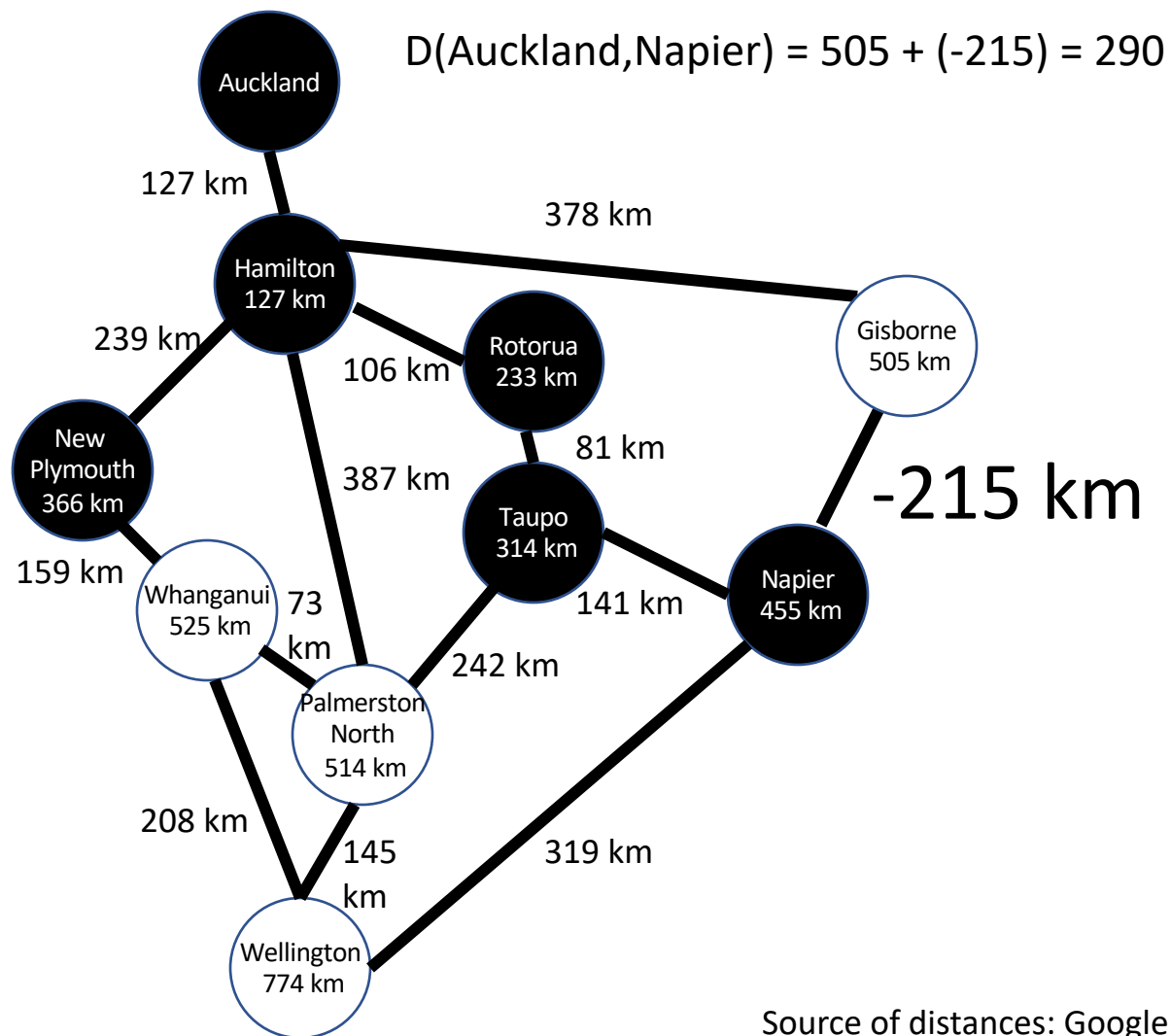
Source of distances: Google Maps

# Example: Adding a Negative Weight



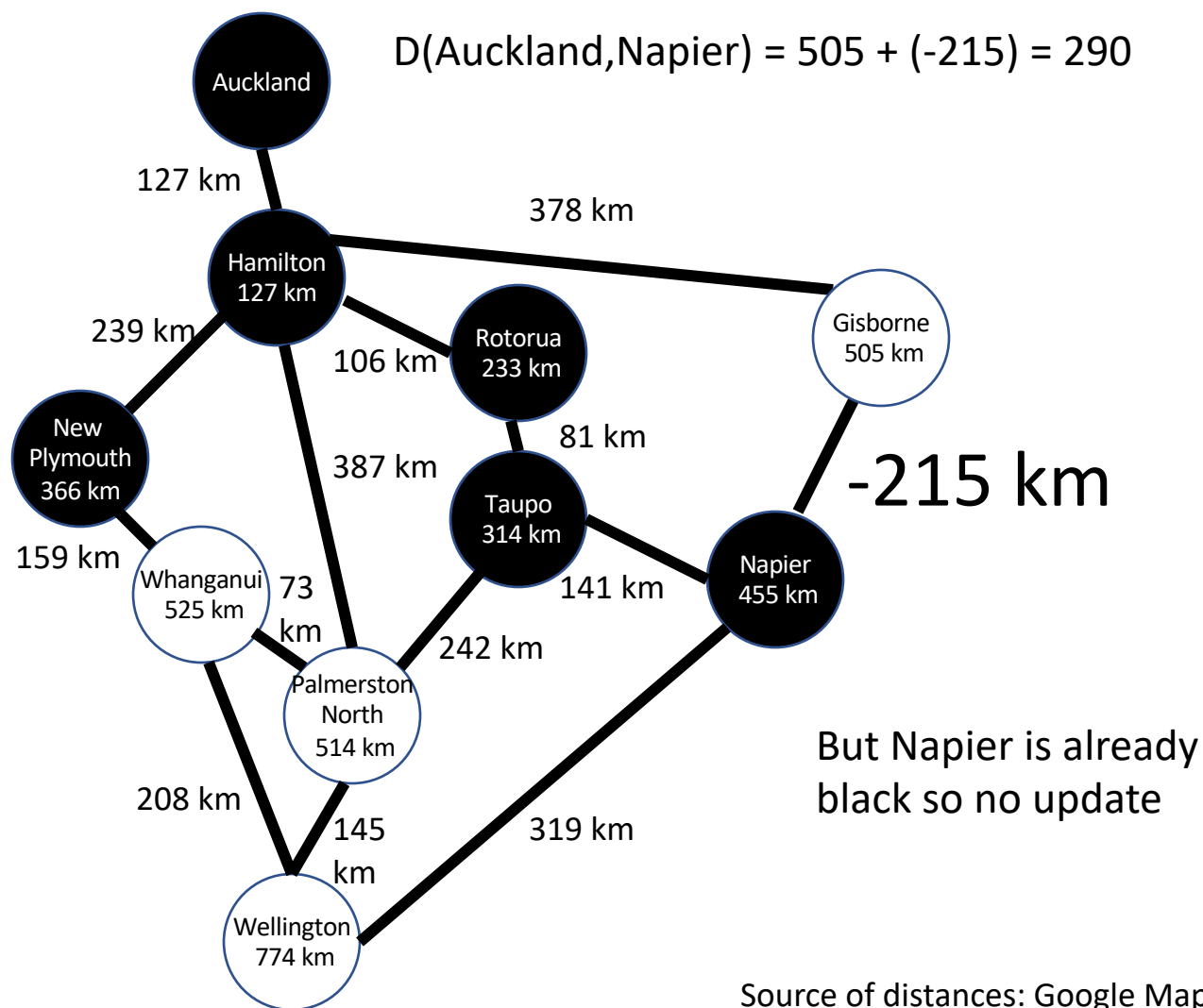
Source of distances: Google Maps

# Example: Adding a Negative Weight



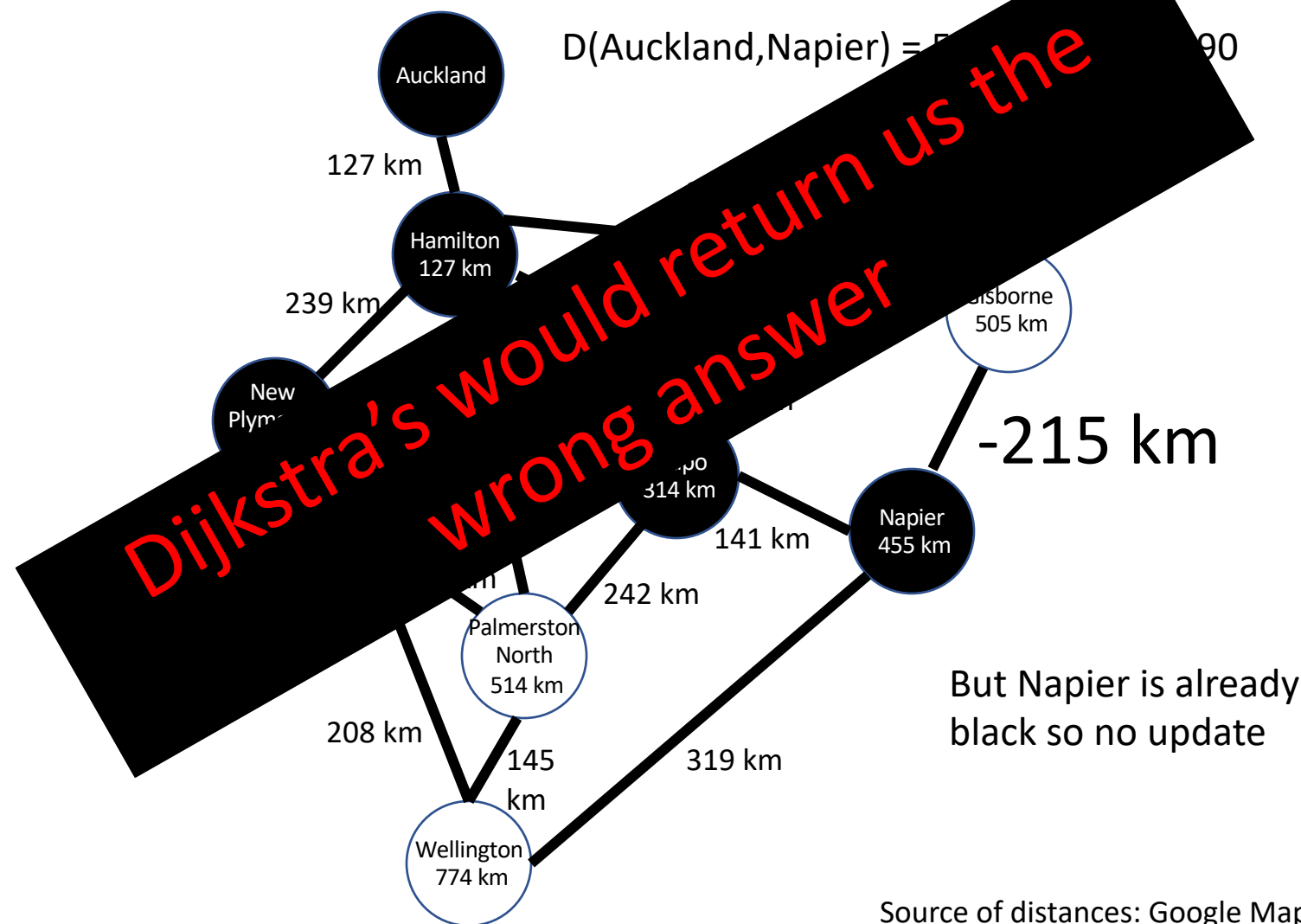
Source of distances: Google Maps

# Example: Adding a Negative Weight



Source of distances: Google Maps

# Example: Adding a Negative Weight



Source of distances: Google Maps

# SUMMARY

- Weighted Graphs
  - Representation
  - Weight as cost functions
- Algorithms on Weighted Graphs
  - Dijkstra
  - Bellman-Ford
  - Floyd-Warshall

