Cosc 201
Algorithms and Data Structures
Lecture 19 (6/5/2026)
Minimum Spanning Trees

Brendan McCane
brendan.mccane@otago.ac.nz

Reminder: Dijkstra's shortest path algorithm

```
1:  procedure DIJKSTRA(G, v)
2:      dist[v] ← 0
3:      for each edge e = (v, u) from v do
4:          q.add(e, e.weight)
5:      end for
6:      while q is not empty do
7:          e = (x, y) ← q.remove()
8:          if dist[y] is not known then
9:              dist[y] ← dist[x] + e.weight
10:             y.parent ← x
11:             for each edge e = (y, z) from y do
12:                 if dist[z] is not known then
13:                     q.add(e, dist[y] + e.weight)
14:                 end if
15:             end for
16:         end if
17:     end while
18: end procedure
```

Prove that Dijkstra’s algorithm computes the shortest distances

We only update distances when we remove a vertex from the queue. We need to show that when we remove a vertex from the queue, the current distance must be the shortest.

By induction:

- Base case: at the start of the while loop, the only valid distance is $dist[v] = 0$ which is the shortest possible distance and therefore the base case is true.

Prove that Dijkstra's algorithm computes the shortest distances

Inductive step:

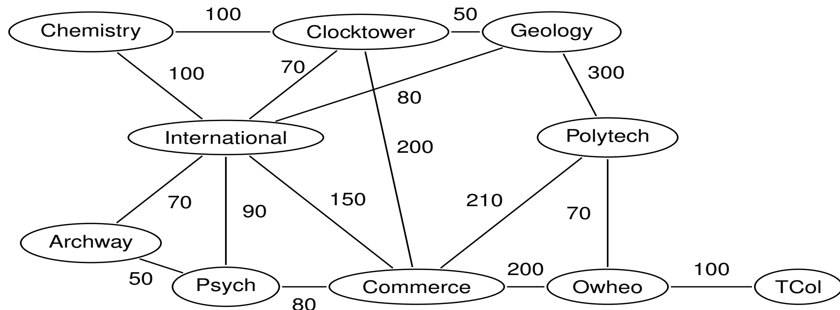
- Assume that the distances up to some intermediate point are all correct. In particular, the distances for all source vertices of edges in the queue are correct.
- We remove edge $e = (x, y)$ from the queue.
- There are two possible cases for y: we have previously removed an edge with y as the target, and hence have already set $dist[y]$; we have not previously removed an edge with y as the target.
- In the first case, we can only go wrong if $dist[x] + e.weight < dist[y]$. Let's say that $dist[y] = dist[z] + (z, y).weight$. We have already visited both x and z and therefore all of their edges have been added to the queue. But, (z, y) has already been removed from the queue, and therefore must have had higher priority than (x, y). Therefore $dist[x] + e.weight \geq dist[y]$, and we have not gone wrong.

Proof continued

- In the second case, we have not yet set $dist[y]$. y is not a source of any edge in the queue (else we would have already set $dist[y]$). Of the edges in the queue with y as a target, $dist[x] + e.weight$ must have the highest priority in the queue and is therefore the shortest distance seen so far.
- Taking the first and second cases together, we can conclude that if the *dist* map contained only shortest distances before the top of the loop, it has only shortest distances at the bottom of the loop.
- Therefore, by induction, the *dist* map contains only shortest distances.

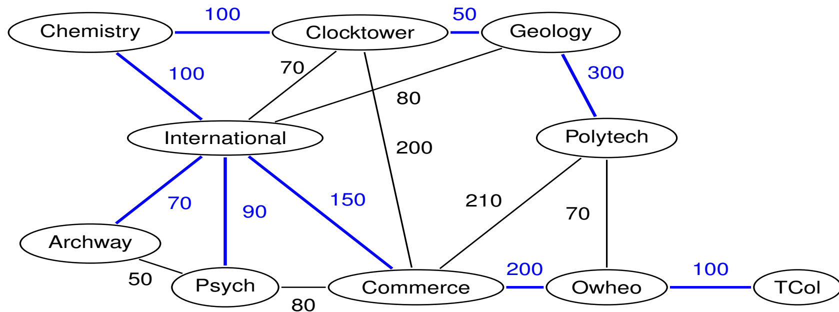
A problem

Imagine you are tasked with connecting the Dunedin campus buildings with a wired network. The wires are expensive, and you want to connect all the buildings with the minimum total length of wire. Here is a graph of the buildings and the distances between them. What is the connectivity pattern you choose?



A problem

Here is the shortest path tree starting from Chemistry (in blue).

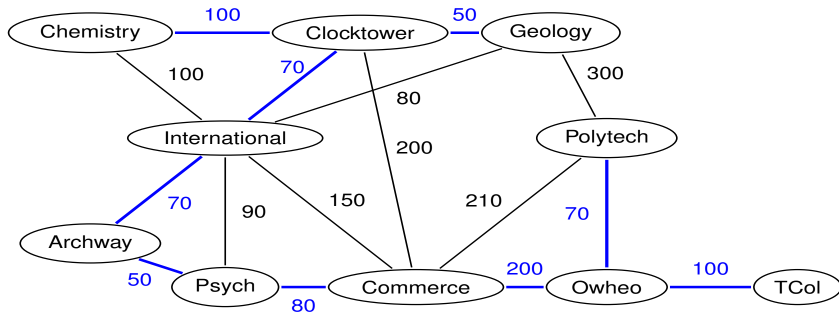


Total distance = 1160. Can we do better?

Prim's algorithm

```
1:  procedure PRIM(G, v)
2:      dist[v] ← 0
3:      for each edge e = (v, u) from v do
4:          q.add(e, e.weight)
5:      end for
6:      while q is not empty do
7:          e = (x, y) ← q.remove()
8:          if dist[y] is not known then
9:              dist[y] ← e.weight
10:             y.parent ← x
11:             for each edge e = (y, z) from y do
12:                 if dist[z] is not known then
13:                     q.add(e, e.weight)
14:                 end if
15:             end for
16:         end if
17:     end while
18: end procedure
```

The minimum spanning tree



Total distance = 790.

Matters of efficiency

- How large a graph do you think we could find all the single-source shortest paths in in a reasonable time?
- **Scenario 1**: For any two vertices v and w there's an edge from v to w of random weight (and the weight from v to w versus w to v might differ). How large a graph can we analyse? There are $n \times (n - 1)$ edges to consider.
- **Scenario 2**: The graph is an $n \times n$ grid in which every vertex is connected to its orthogonal neighbours by edges of random weight. There are approximately $2n^2$ edges. How large a graph can we analyse?