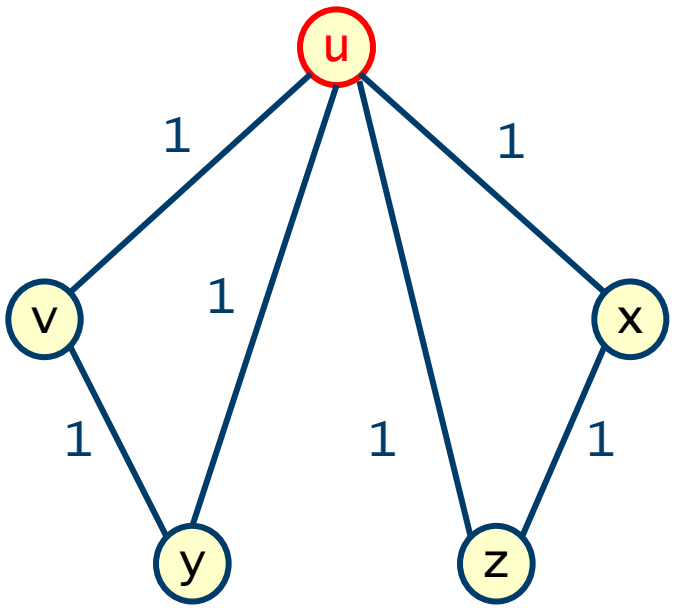
Dijkstra's algorithm versus DFS

Compute shortest path with u

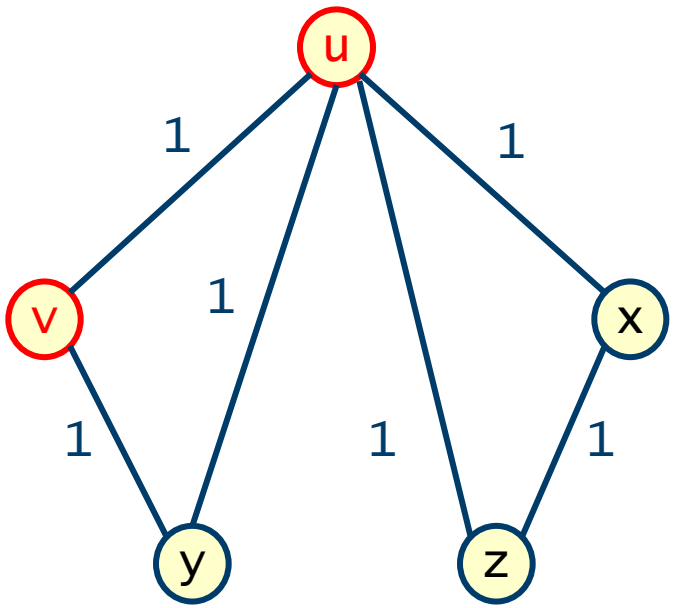
- If we perform DFS from u



Dijkstra's algorithm versus DFS

Compute shortest path with **u**

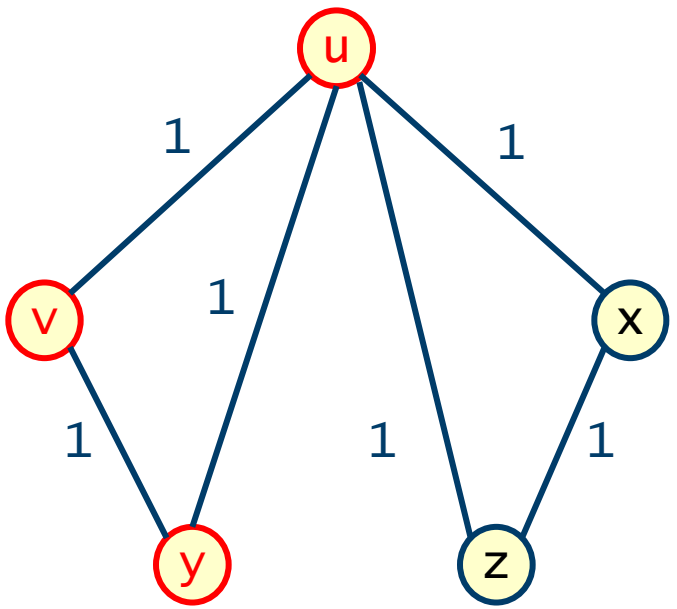
- If we perform DFS from u
  - mark v as visited
  - predecessor is u



Dijkstra's algorithm versus DFS

Compute shortest path with **u**

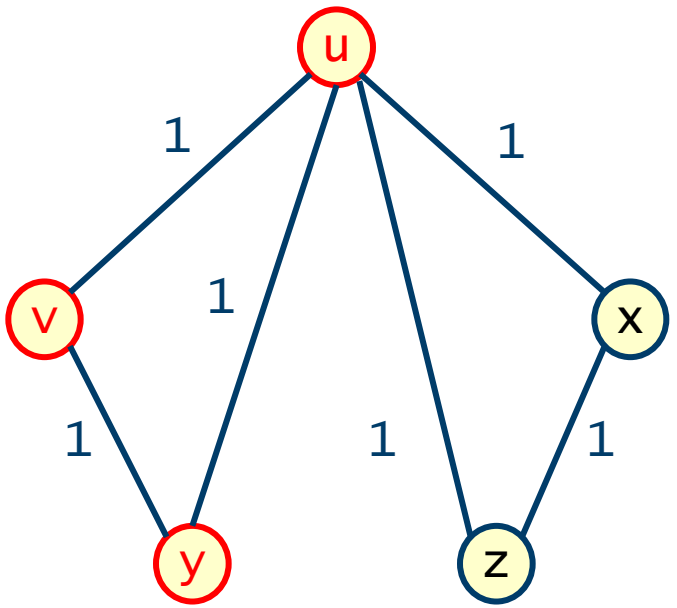
- If we perform DFS from u
  - mark v as visited
  - predecessor is u
  - mark y as visited
  - predecessor is v



Dijkstra's algorithm versus DFS

Compute shortest path with u

- If we perform DFS from u
  - mark v as visited
  - predecessor is u
  - mark y as visited
  - predecessor is v



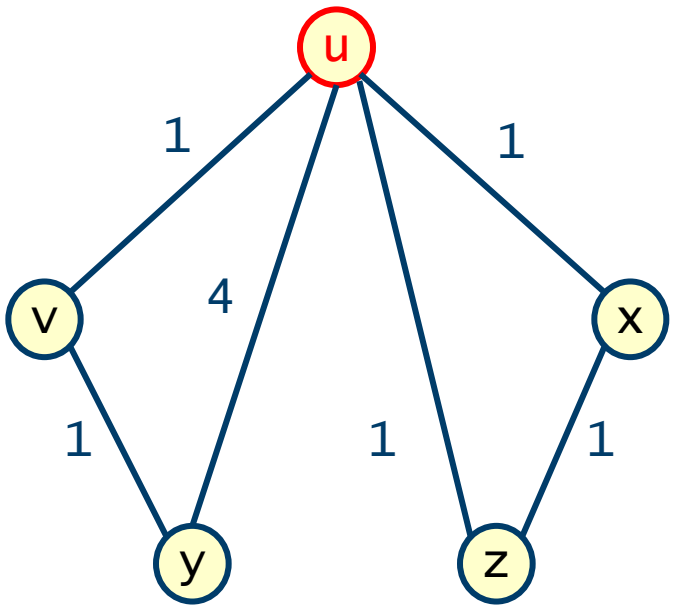
DFS therefore finds the path u→v→y between u and y

However, the shortest paths between u and y in terms of edges and distance (sum of weights) is the path u→y

Dijkstra's algorithm versus BFS

Compute shortest path with u

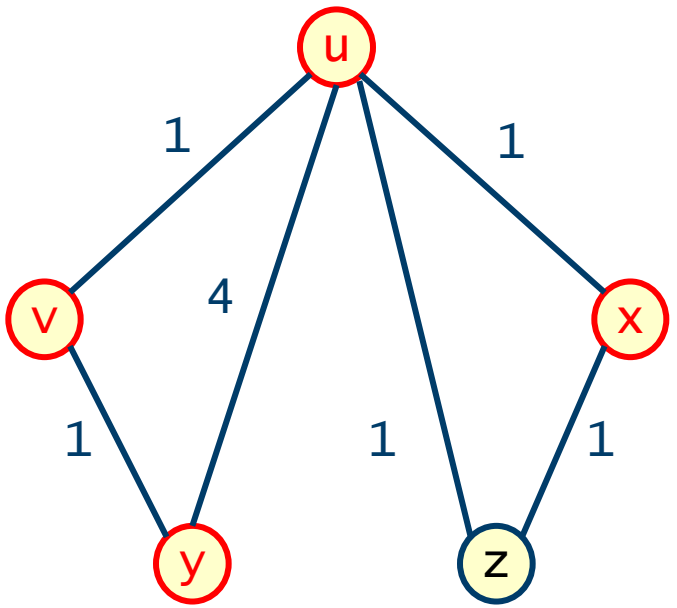
- If we perform BFS from u



Dijkstra's algorithm versus BFS

Compute shortest path with u

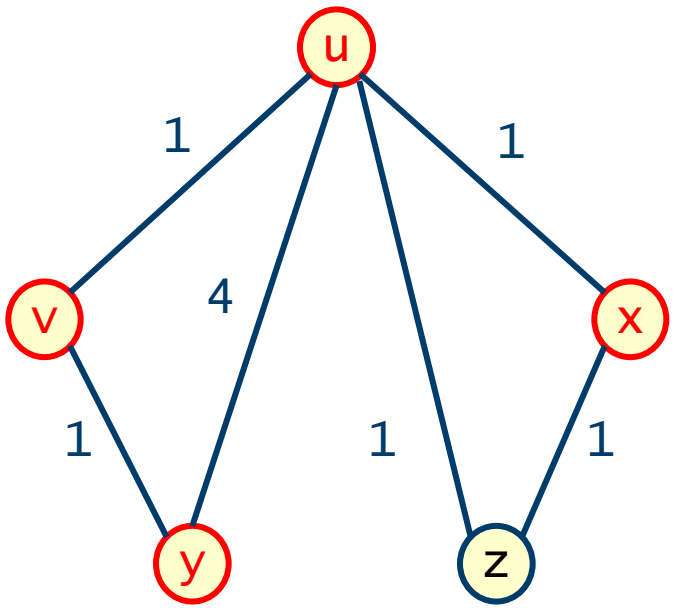
- If we perform BFS from u
  - mark v, y and x as visited and set their predecessor to be u



Dijkstra's algorithm versus BFS

Compute shortest path with u

- If we perform BFS from u
  - mark v, y and x as visited and set their predecessor to be u



BFS therefore finds the path u→y between u and y which is the shortest path in terms of edges

- using BFS is the most efficient method for finding such paths

However, the shortest paths between u and y in terms of distance (sum of weight is the path u→v→y