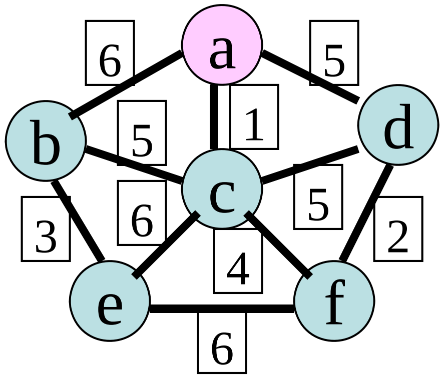
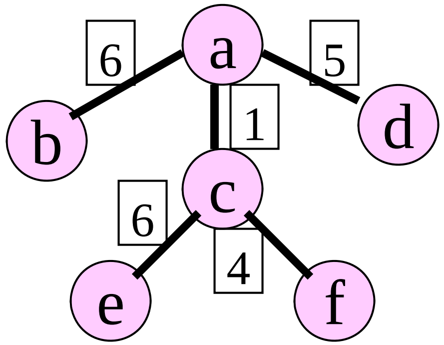
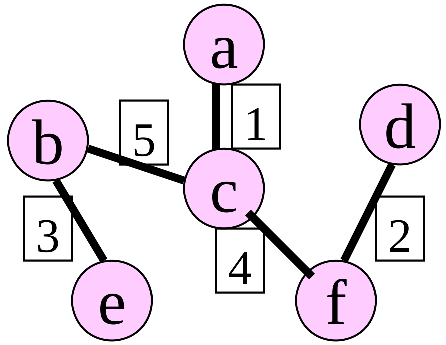
SPT versus MST



Graph



SPT – cost 22



MST – cost 15

counter example: in the MST the **path a to d** is NOT the shortest path (7 versus 5 in the SPT)

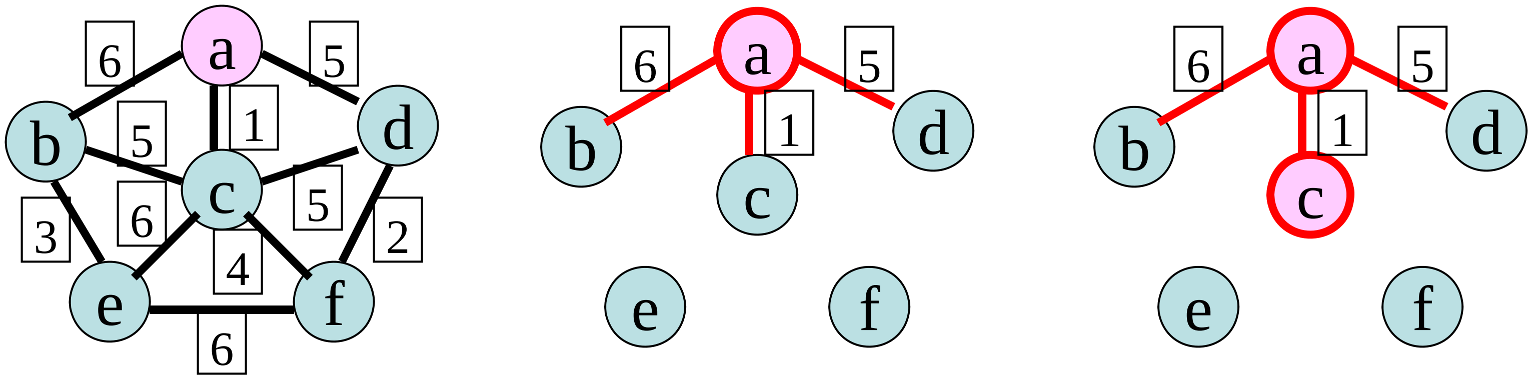
ditto: **a to e** (9 versus 7 in the SPT)

Dijkstra – worked example

- Principle
  - Given a **path x → z** check if there exists a node **y** such that the **path length x → y → z** is shorter than the currently calculated **path length x → z**
  - Node **y** is chosen to be the **shortest path** from **x**
  - An example using the above graph follows

Dijkstra – worked example

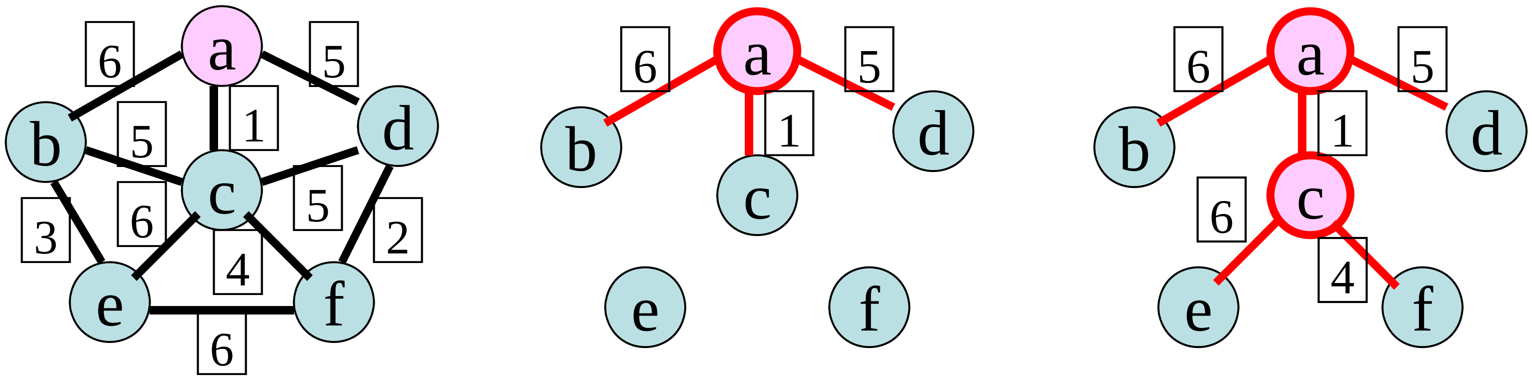
- Graph & initialisation (edges) from node a
- **Path** a-1-c (cost 1) is the cheapest **path**



- Now calculate alternative **paths** via c

Dijkstra – worked example

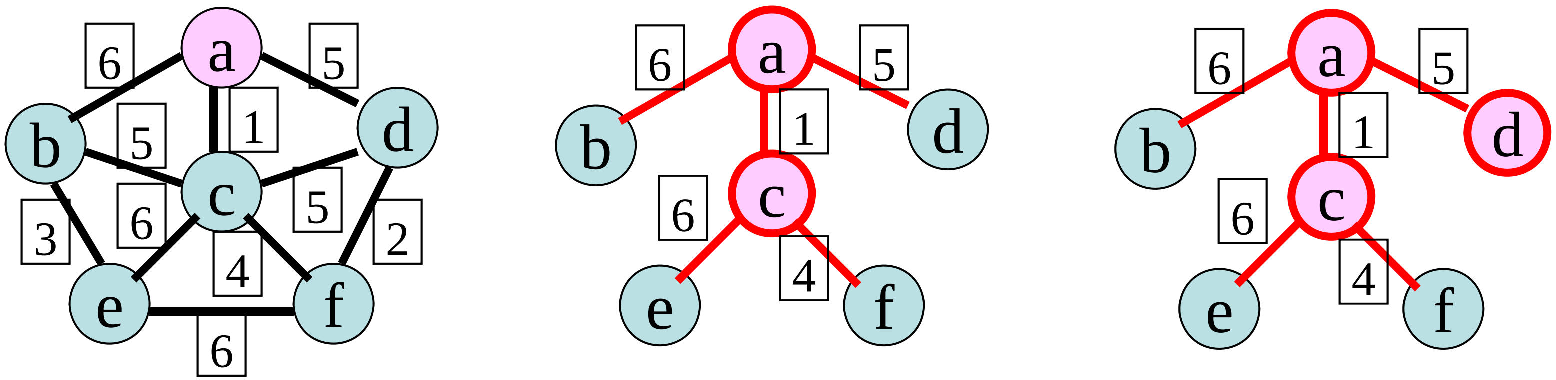
- **Calculate paths via c to unvisited = {b, d, e, f}, visited = {a, c}**



- **a-1-c-5-b (cost 6) – not cheaper than a-6-b (cost 6)**
- **a-1-c-5-d (cost 6) – not cheaper than a-5-d (cost 5)**
- **a-1-c-6-e (cost 7) – cheaper than a-§-e (no path)**
- **a-1-c-4-f (cost 5) – cheaper than a-§-f (no path)**

Dijkstra – worked example

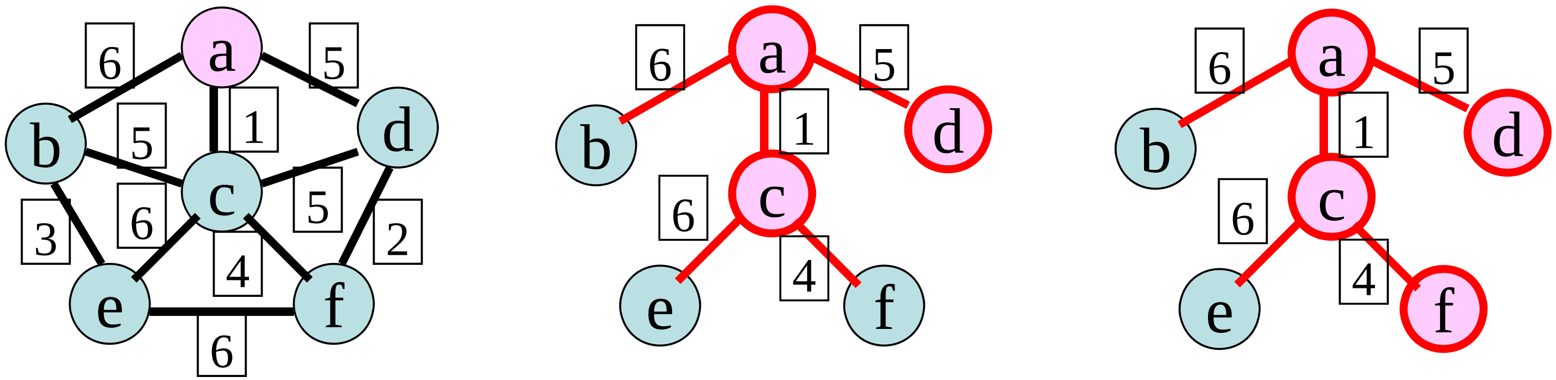
- **Calculate paths via d to unvisited = {b, e, f}, visited = {a, c, d}**
- **a-5-d (cost 5) is the cheapest path to an unvisited node**



- **a-5-d-§-b (cost §) – not cheaper than a-6-b (cost 6)**
- **a-5-d-§-e (cost §) – not cheaper than a-1-c-6-e (cost 7)**
- **a-5-d-2-f (cost 7) – not cheaper than a-1-c-4-f (cost 5)**
- **No change to the SPT**

Dijkstra – worked example

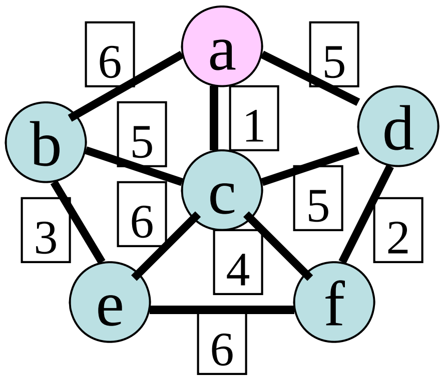
- **Calculate** **paths** **via f to unvisited = {b, e}, visited = {a, c, d, f}**
- **a-1-c-4-f (cost 5) is the cheapest** **path** **to an unvisited node**

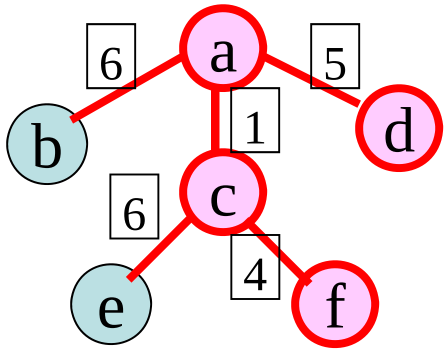


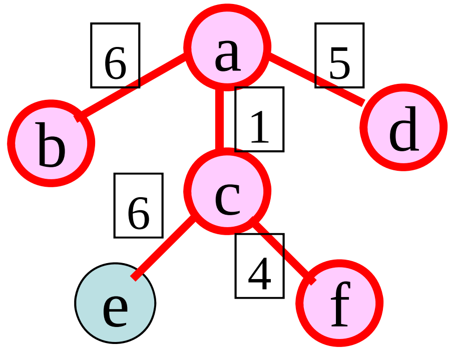
- **a-1-c-4-f-§-b (cost §)   – not cheaper than a-6-b (cost 6)**
- **a-1-c-4-f-6-e (cost 11) – not cheaper than a-1-c-6-e (cost 7)**
- **No change to the SPT**

Dijkstra – worked example

- **Calculate paths via b to unvisited = {e}, visited = {a, c, d, f, b}**
- **a-6-b (cost 6) is the cheapest path to an unvisited node**



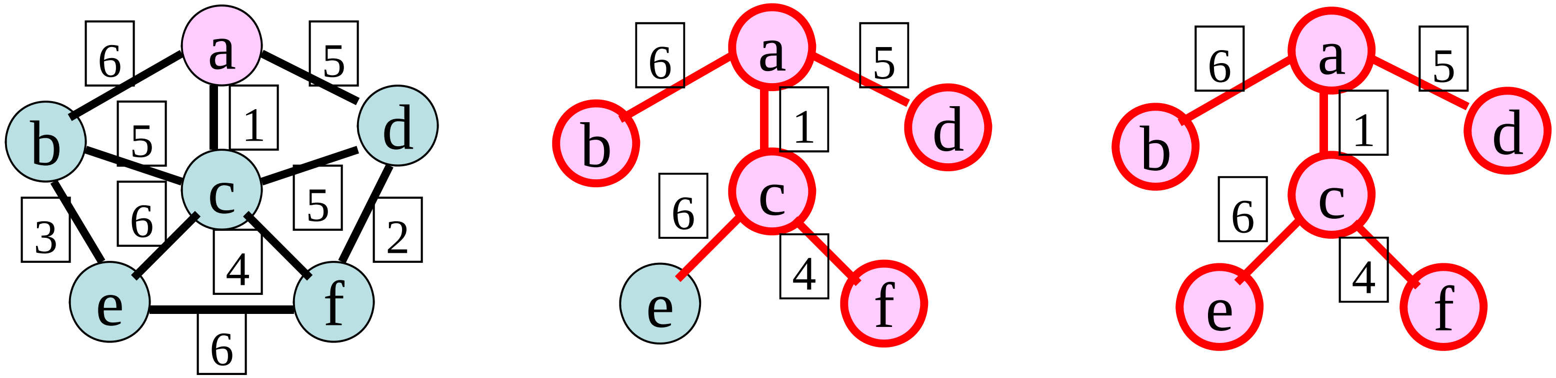




- **a-6-b-3-e (cost 9) – not cheaper than a-1-c-6-e (cost 7)**
- **No change to the SPT**

Dijkstra – worked example

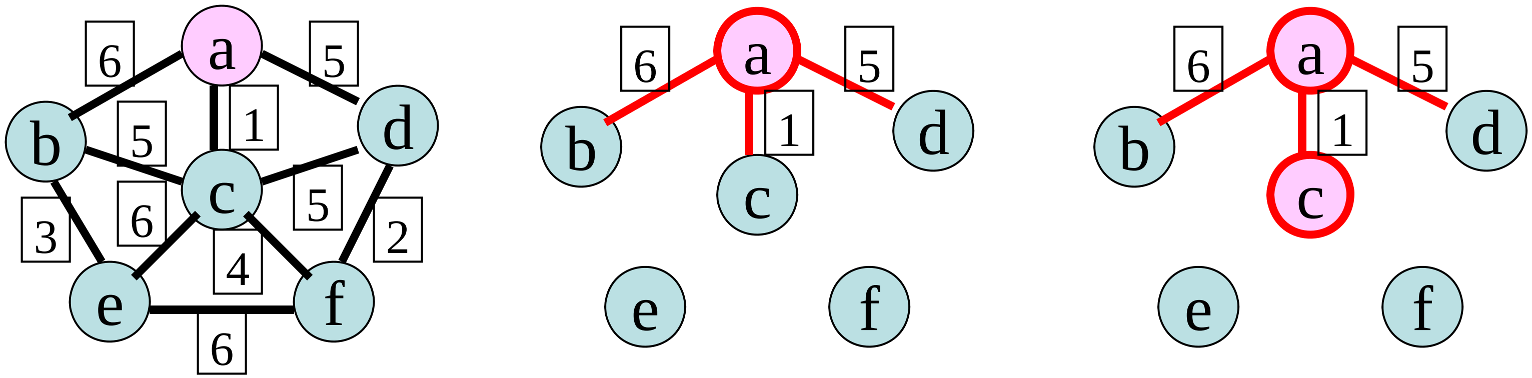
- **Calculate paths via e to unvisited = {¤}, visited = {a, c, d, f, b, e}**
- **a-1-6-e (cost 7) is the cheapest path to an unvisited node**



- **The unvisited node set is empty - STOP**
- **No change to the SPT**

Prim – worked example

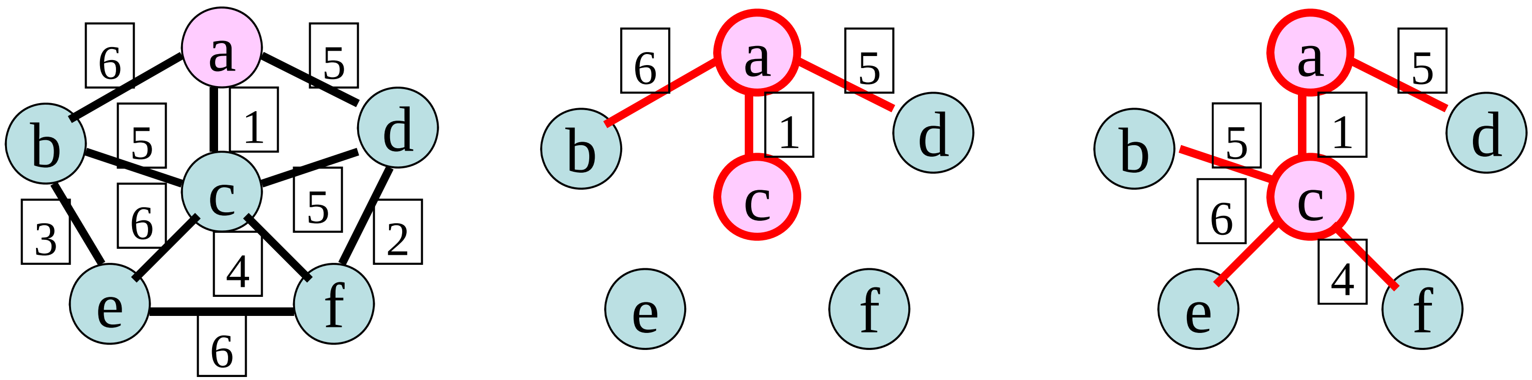
- Graph & initialisation (edges) from node a
- **<u>Edge</u>** a-1-c (cost 1) is the cheapest **<u>edge</u>**



- Now calculate **alternative <u>edges</u> from c**

Prim – worked example

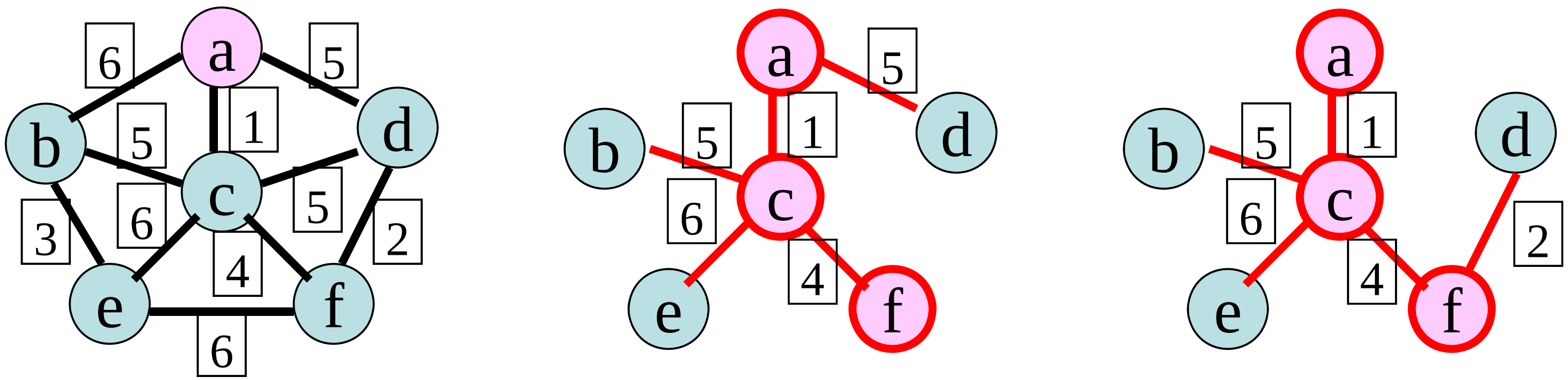
- **Calculate** **edges from c** **to unvisited = {b, d, e, f}, visited = {a, c}**



- **c-5-b – is cheaper than a-6-b – replace a-6-b with c-5-b**
- **c-6-e – is cheaper than a-§-e (no edge)**
- **c-4-f – is cheaper than a-§-f (no edge)**

Prim – worked example

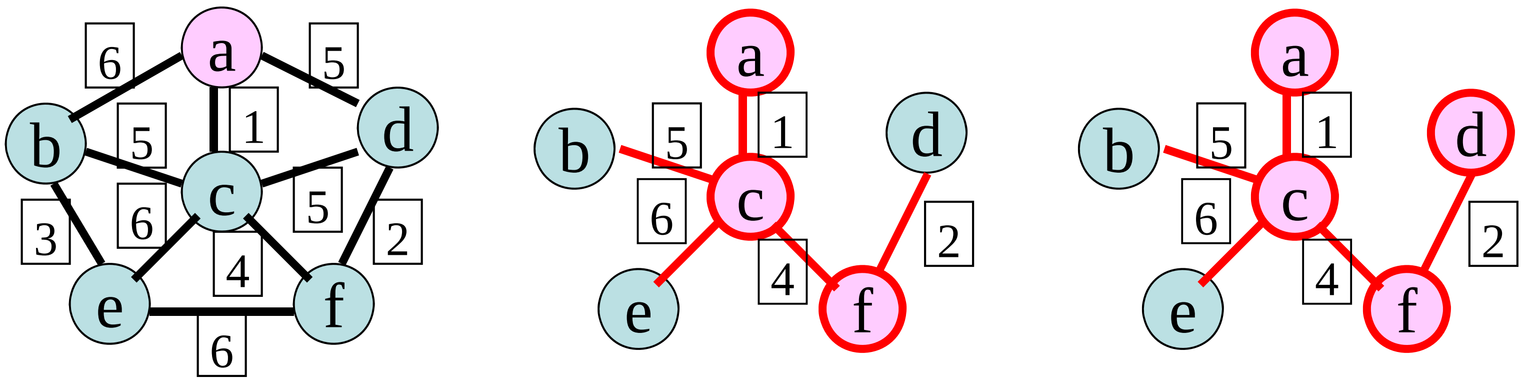
- **Calculate edges from f to unvisited = {b, d, e}, visited = {a, c, f}**
- **c-4-f is the cheapest edge from component a-c**



- **f-§-b – is not cheaper than c-5-b**
- **f-2-d – is cheaper than a-5-e – replace a-5-d with f-2-d**
- **f-6-e – is not cheaper than c-6-e**

Prim – worked example

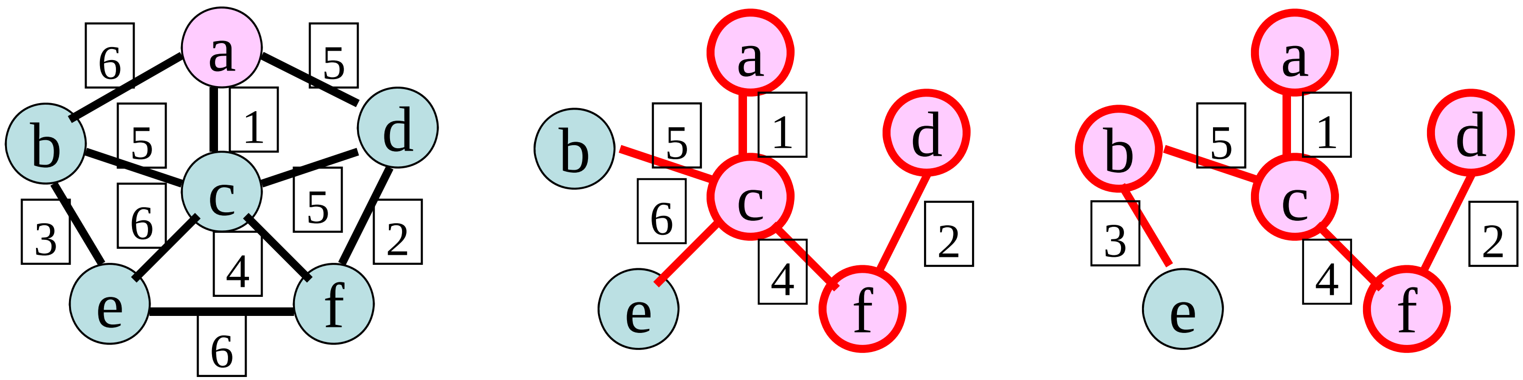
- **Calculate edges from d to unvisited = {b, e}, visited = {a, c, f, d}**
- **f-2-d is the cheapest edge from component a-c-f**



- **d-§-b – is not cheaper than c-5-b**
- **d-§-e – is not cheaper than c-6-e**
- **No change**

Prim – worked example

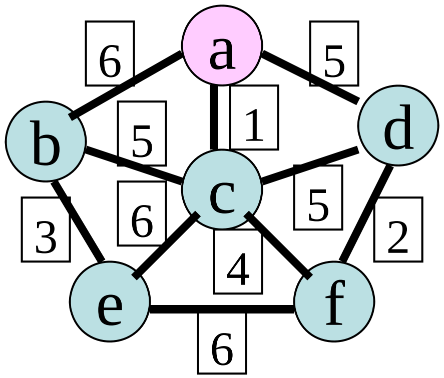
- **Calculate edges from b to unvisited = {e}, visited = {a, c, f, d, b}**
- **c-5-b is the cheapest edge from component a-c-f-d**

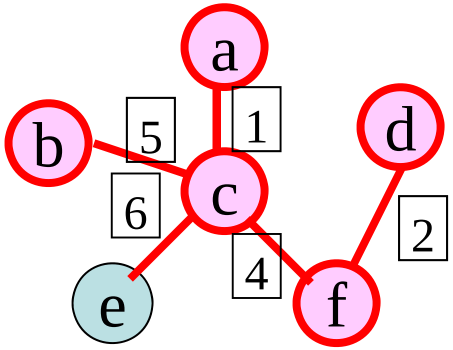


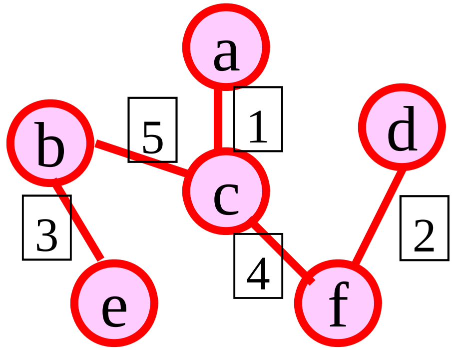
- **b-3-e – is cheaper than c-6-e – replace c-6-e with b-3-e**

Prim – worked example

- **unvisited = {¤} i.e. is empty, visited = {a, c, f, d, b, e}**



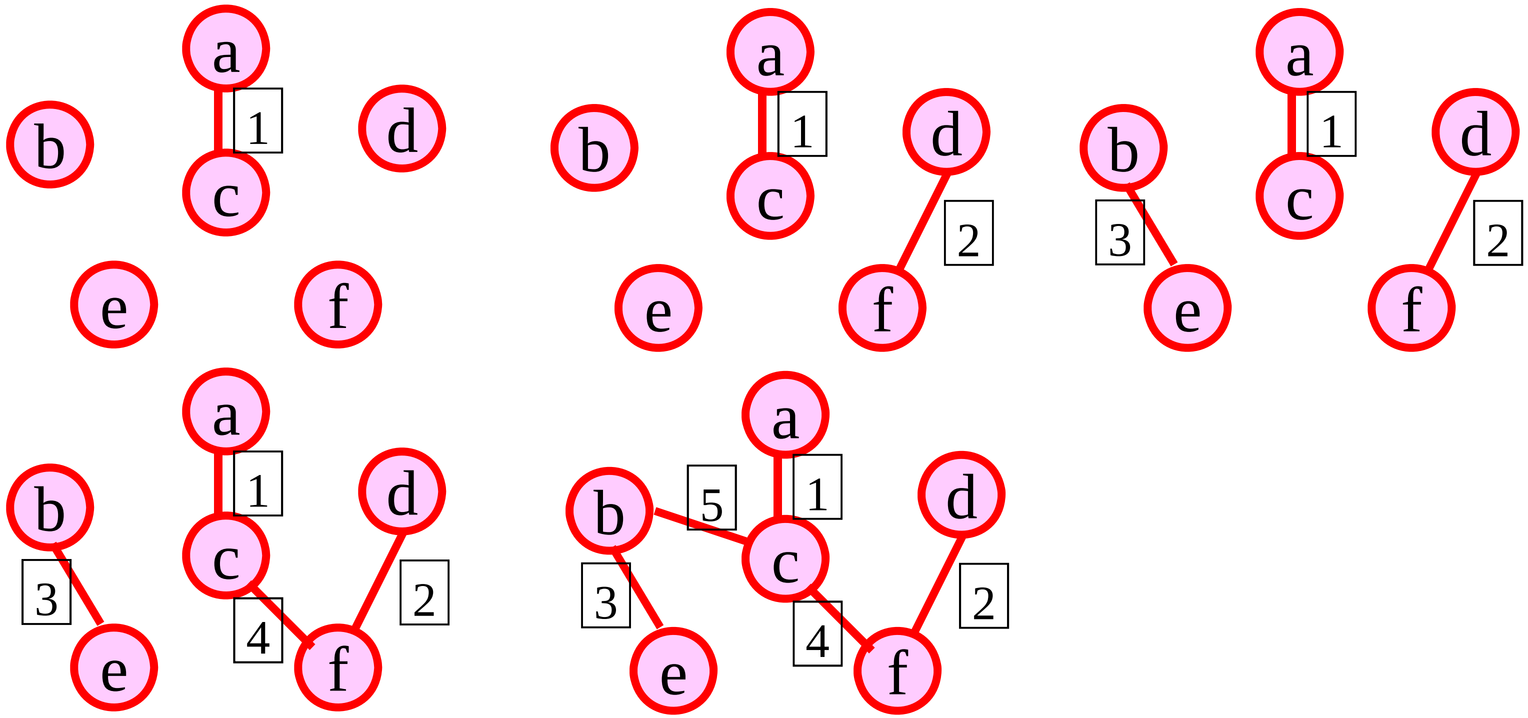




- **Prims has finished**
- **The result may be confirmed using Kruskal (see below)**
- **PQ: a-1-c, d-2-f, b-3-e, c-4-f, c-5-b,**
  a-5-d, c-5-d, a-6-b, c-6-e, e-6-f

Kruskal – worked example

- **PQ:** **a-1-c, d-2-f, b-3-e, c-4-f, c-5-b,**
  a-5-d, c-5-d, a-6-b, c-6-e, e-6-f



Comments: Dijkstra & Prim

- **Dijkstra** uses **path lengths** **- remember this!!!**
- **Prim** uses **edges** **- remember this!!!**

- Both Dijkstra & Prim "grow" a single component
- **Kruskals** "grows" several components which merge
- **Dijkstra** yields an SPT – Shortest Path Tree
- **Prim** yields an MST – Minimal Spanning Tree
- **Kruskal** yields an MST – Minimal Spanning Tree

- Dijkstra & Prim are frequently confused in the exam!!!