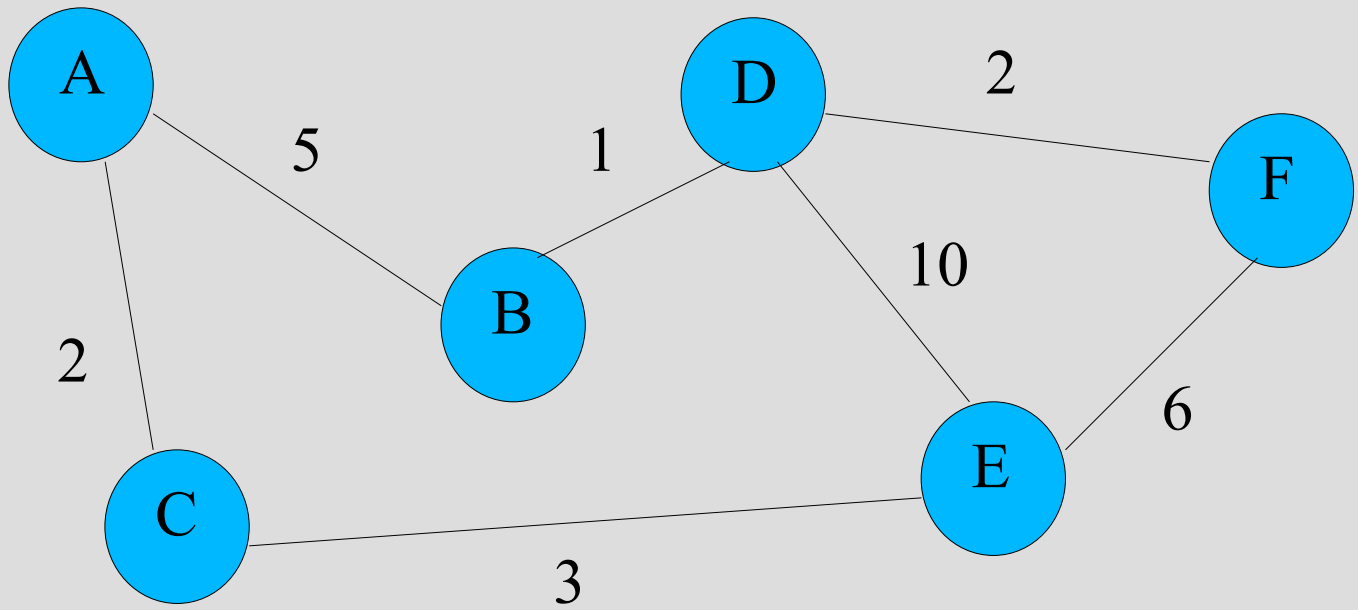
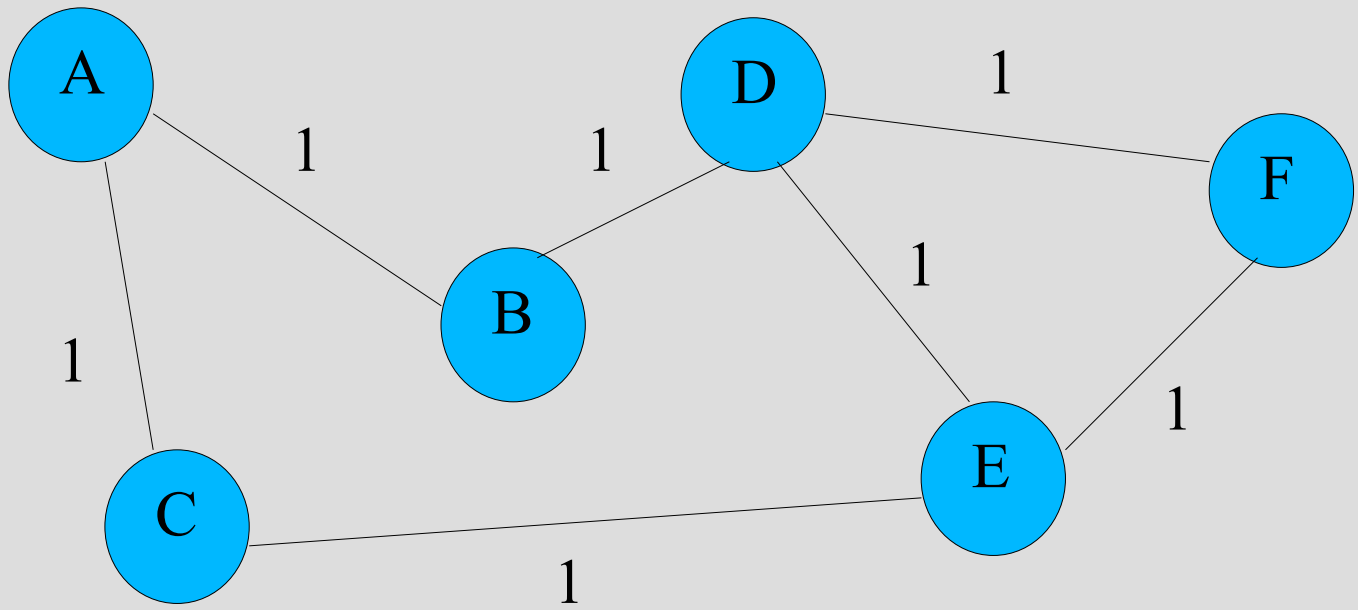
Shortest Paths

The problem

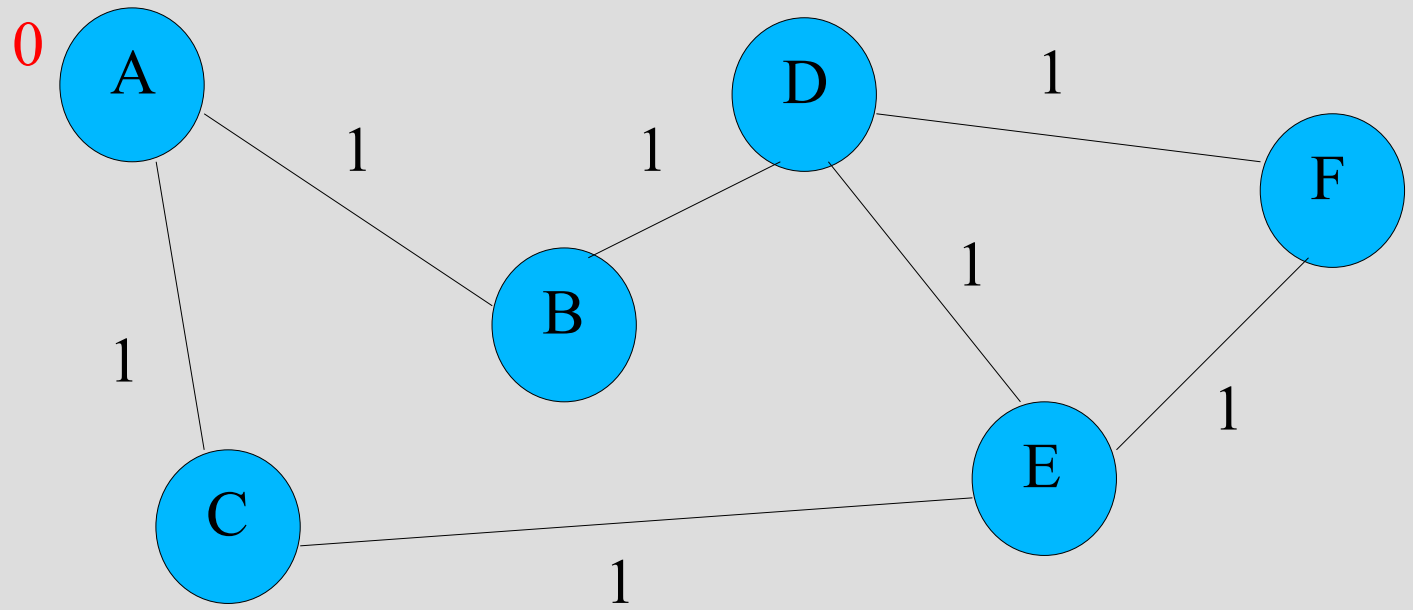


Breadth-first search



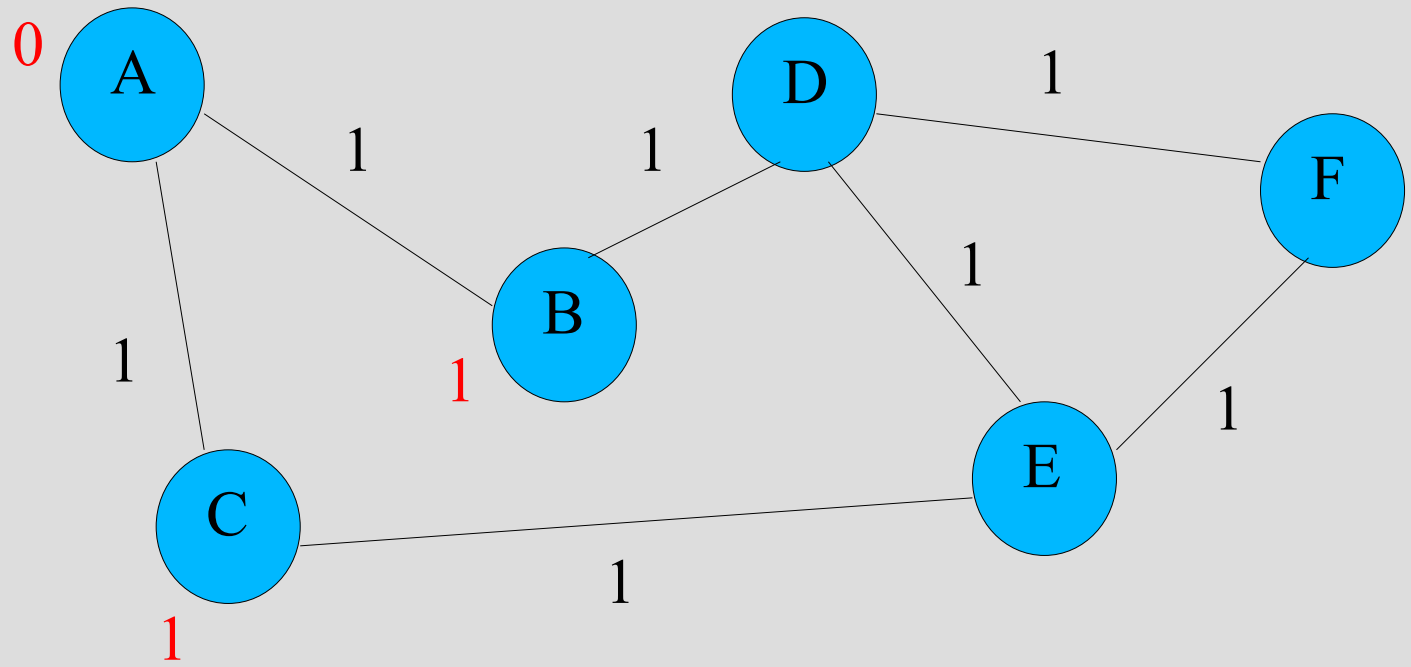
Breadth-first search

Queue: A



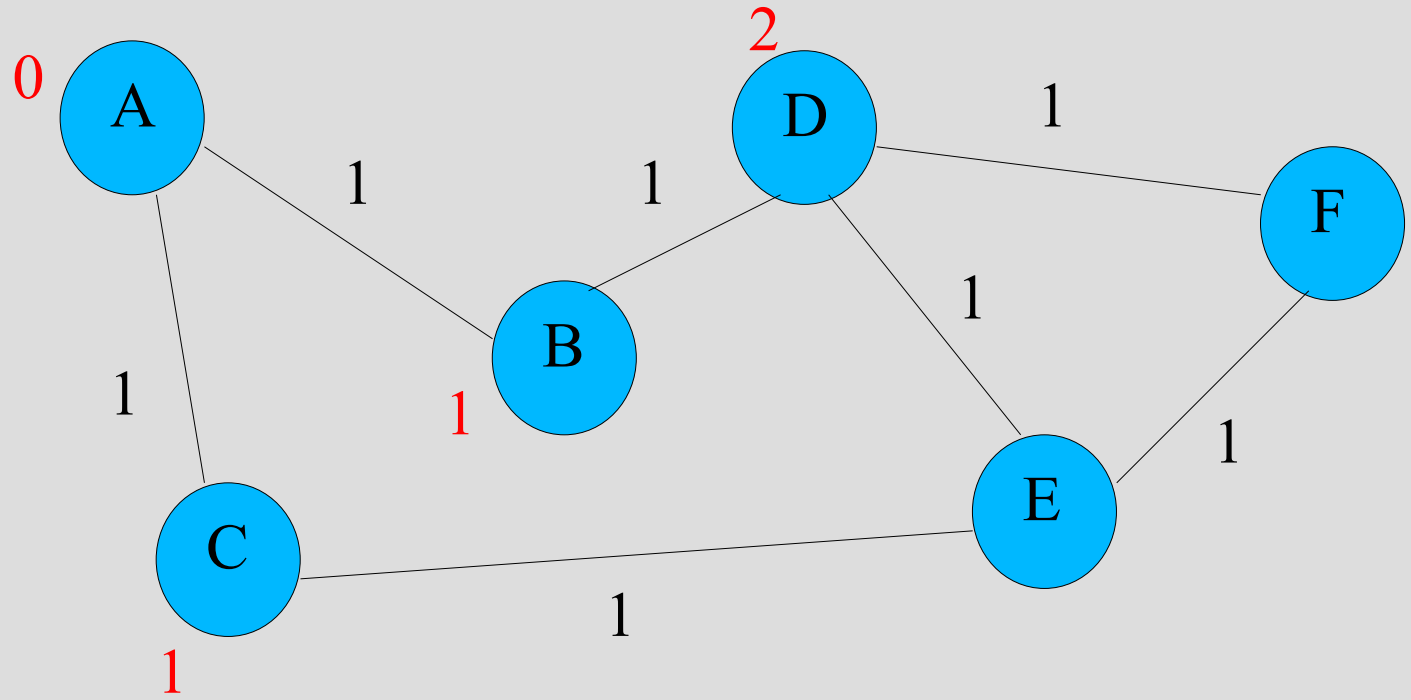
Breadth-first search

Queue: B, C



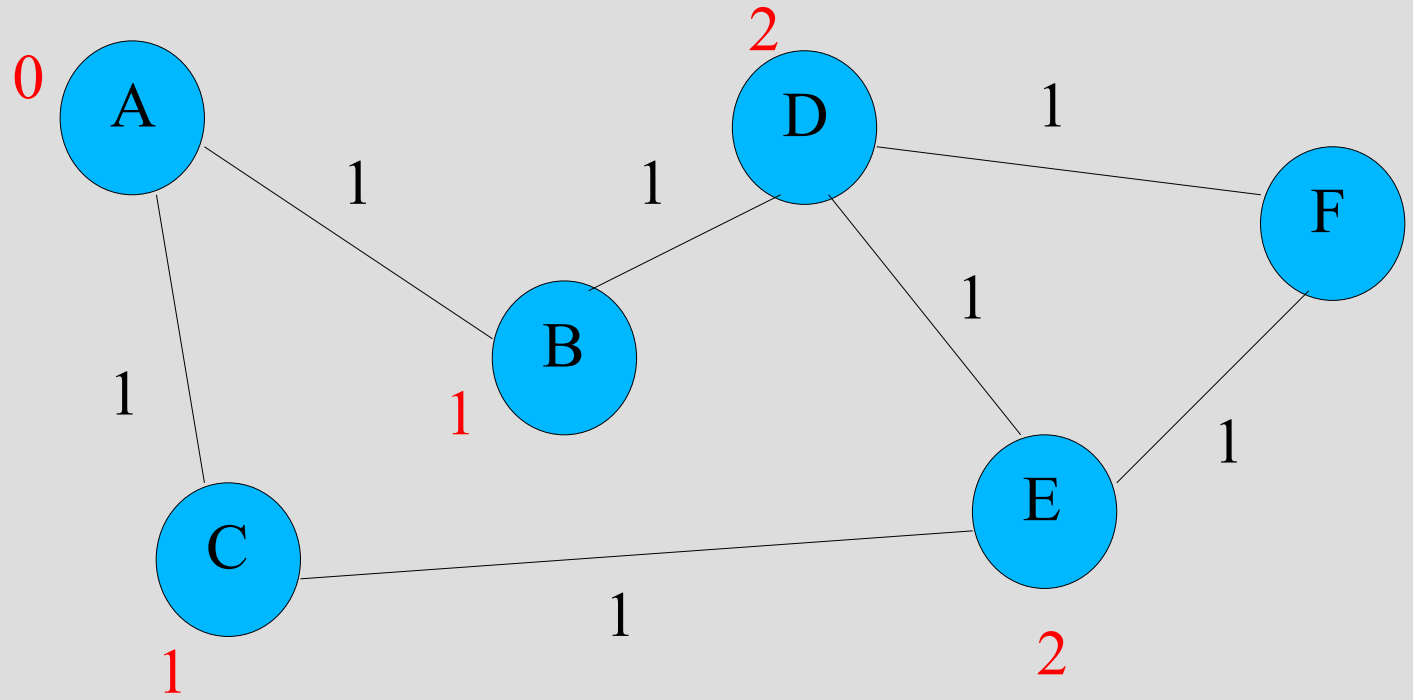
Breadth-first search

Queue: C, D



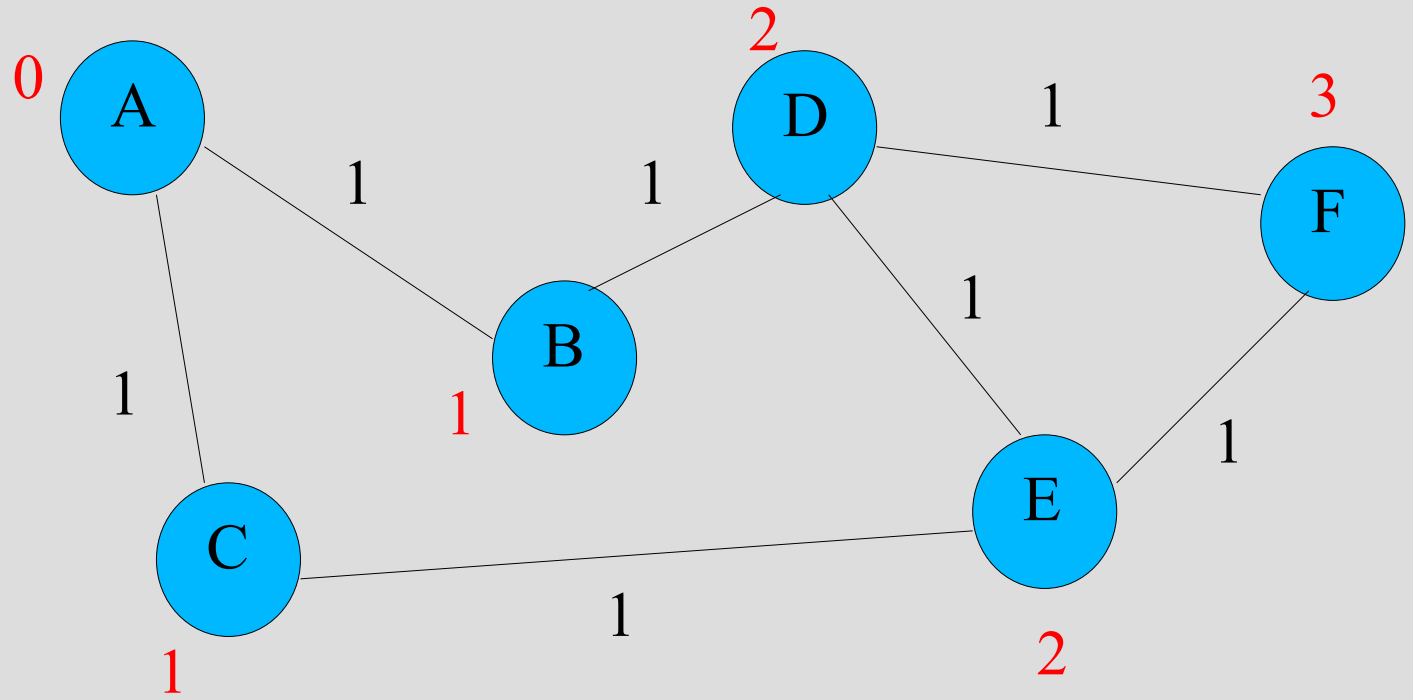
Breadth-first search

Queue: D, E



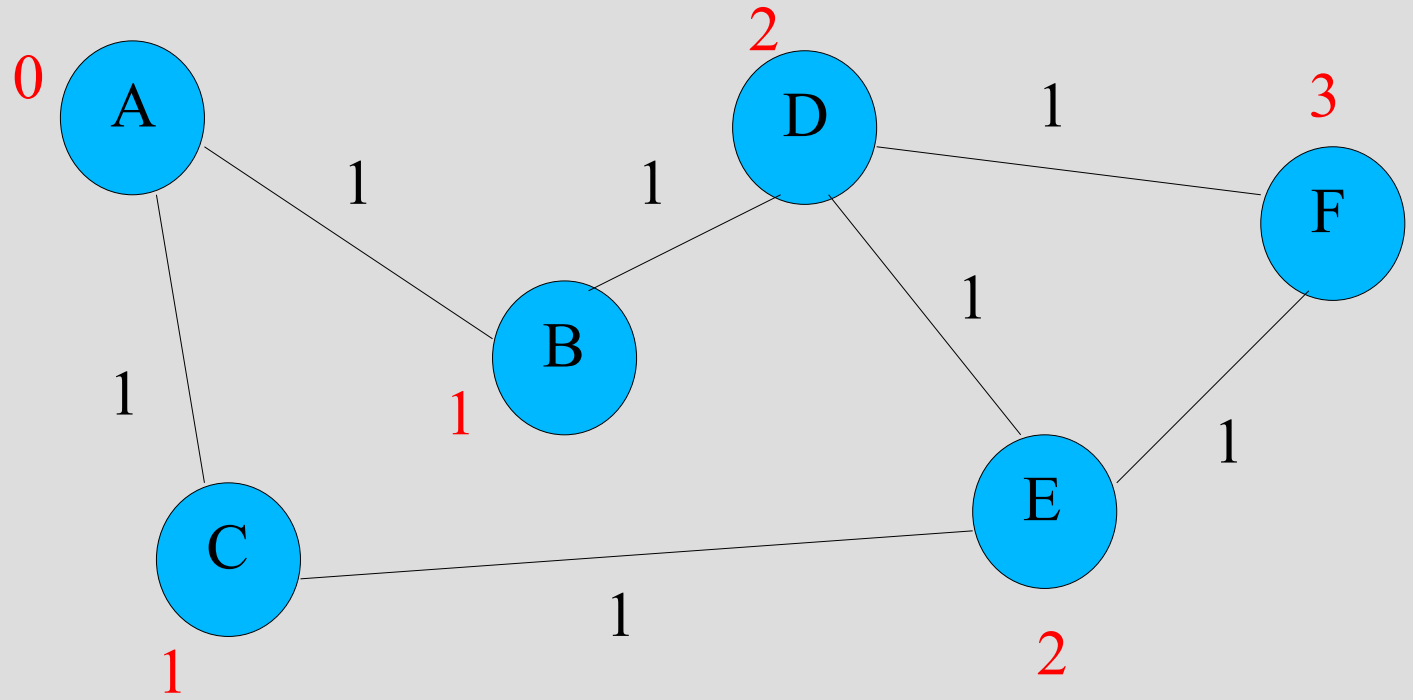
Breadth-first search

Queue: E, F



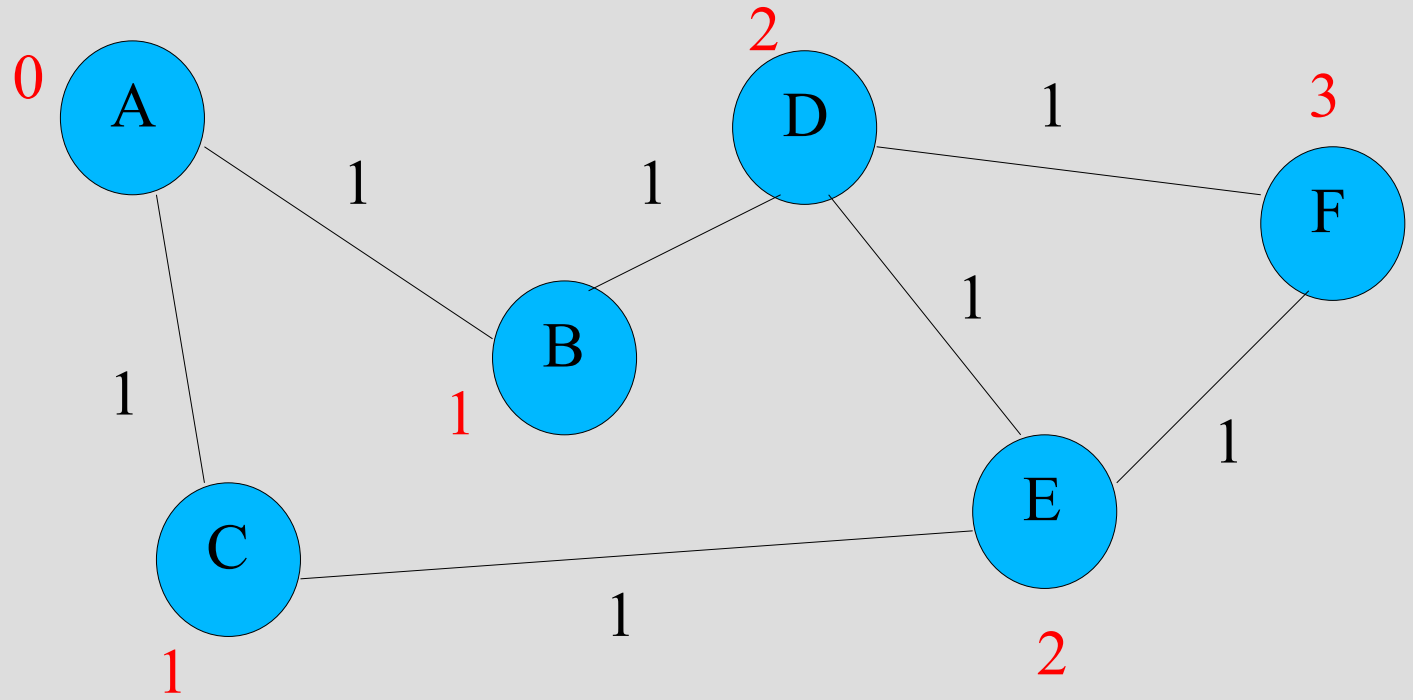
Breadth-first search

Queue: F



Breadth-first search

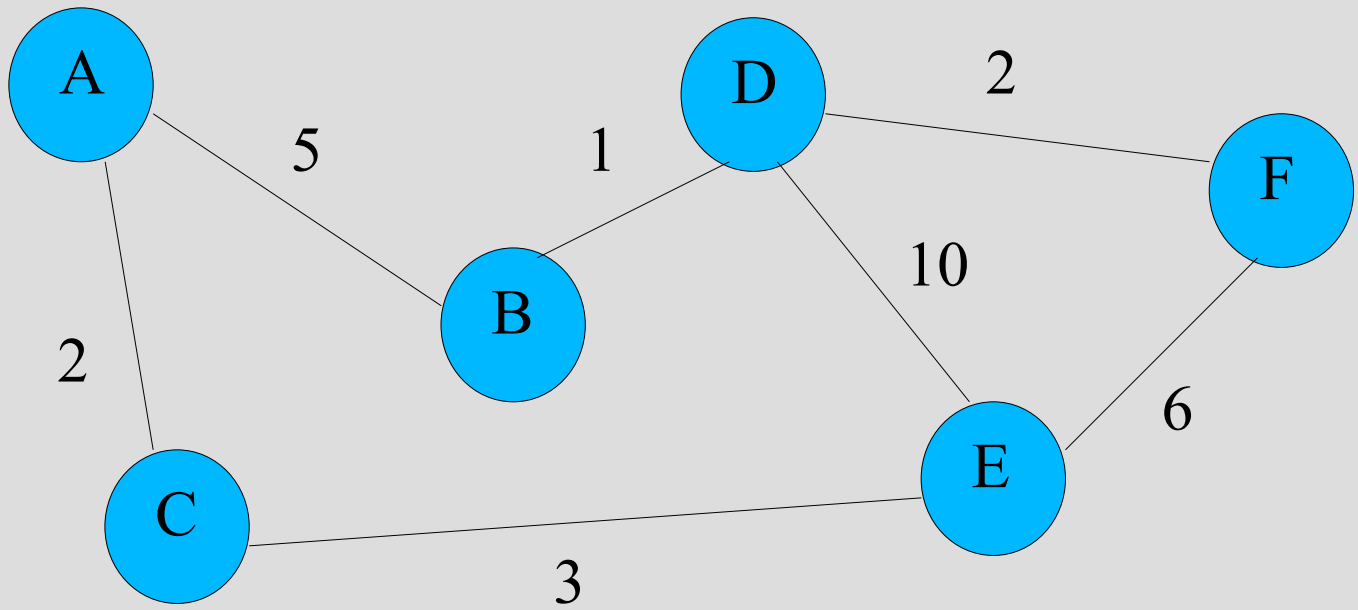
Queue:



Breadth-first search

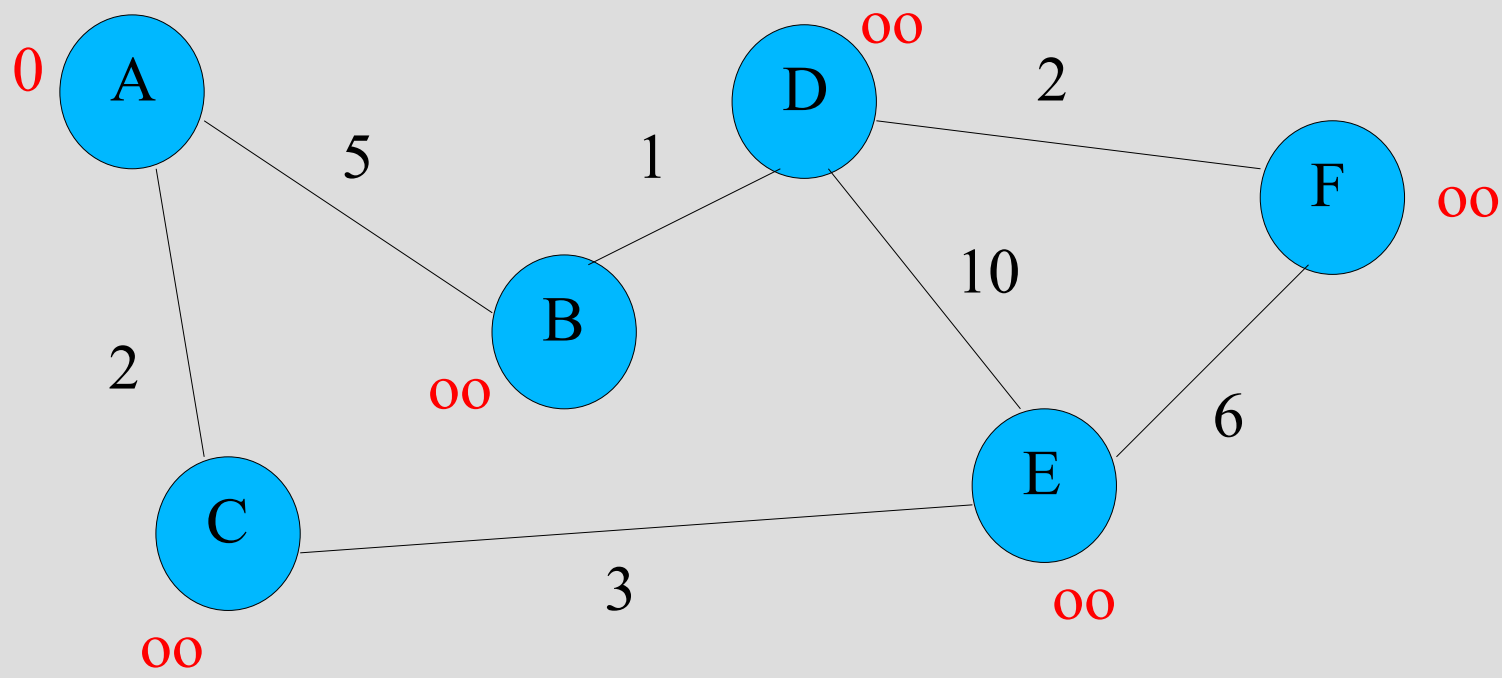
- Time: O(n+e)

Dijkstra's algorithm



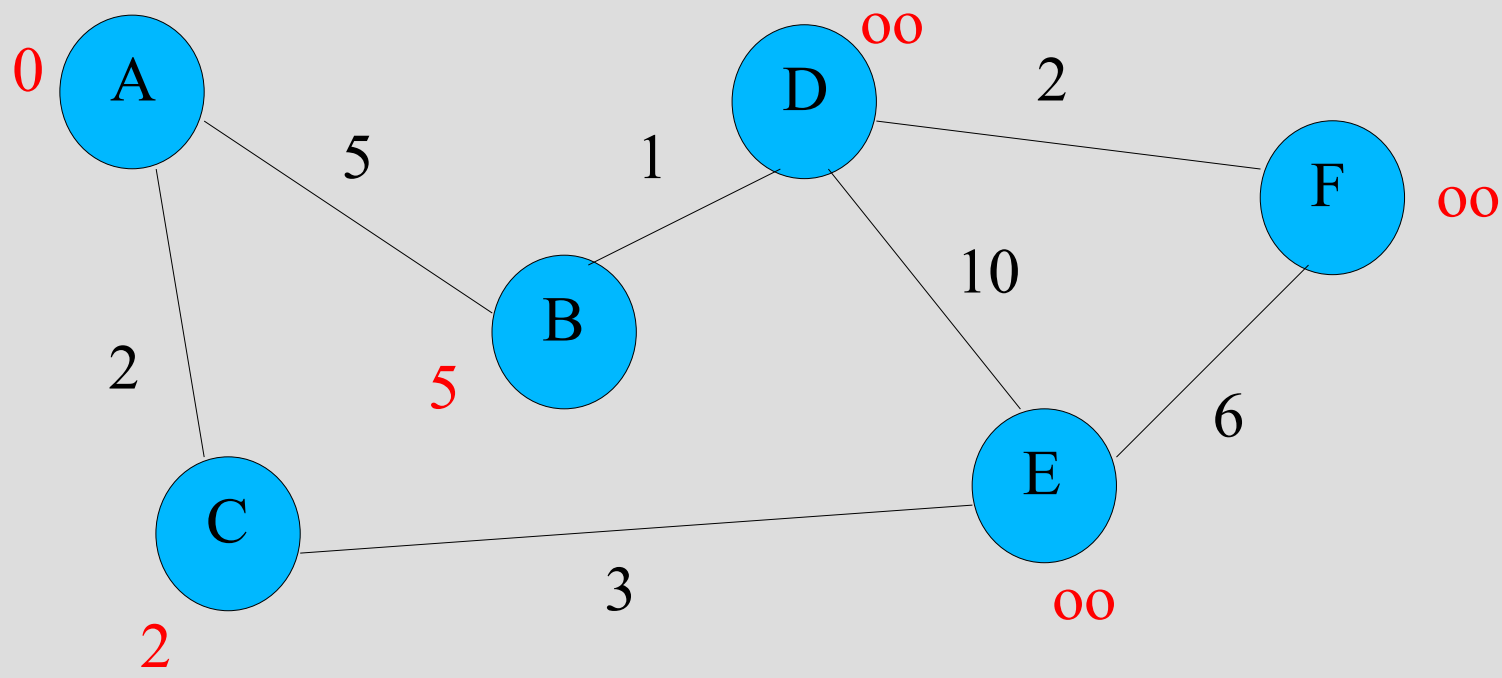
Dijkstra's algorithm

Priority queue: A, B, C, D, E, F



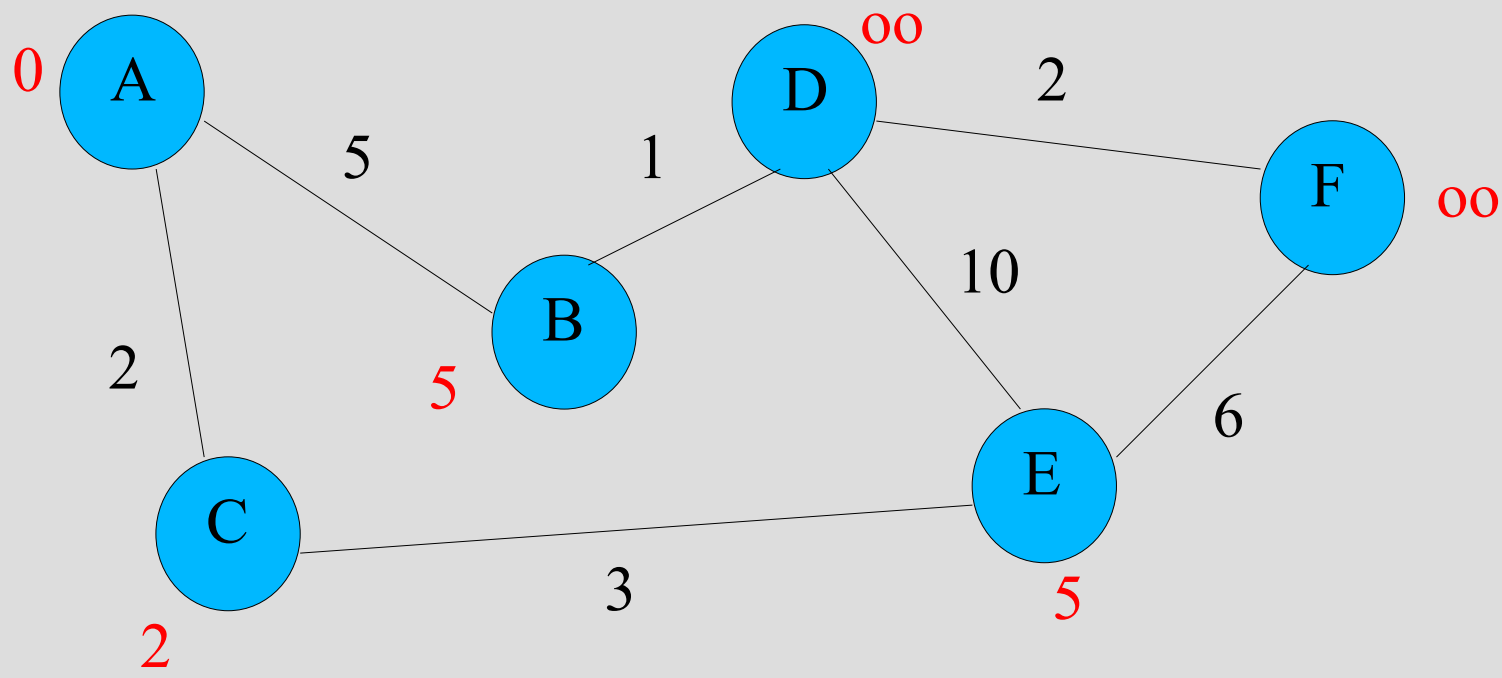
Dijkstra's algorithm

Priority queue: C, B, D, E, F



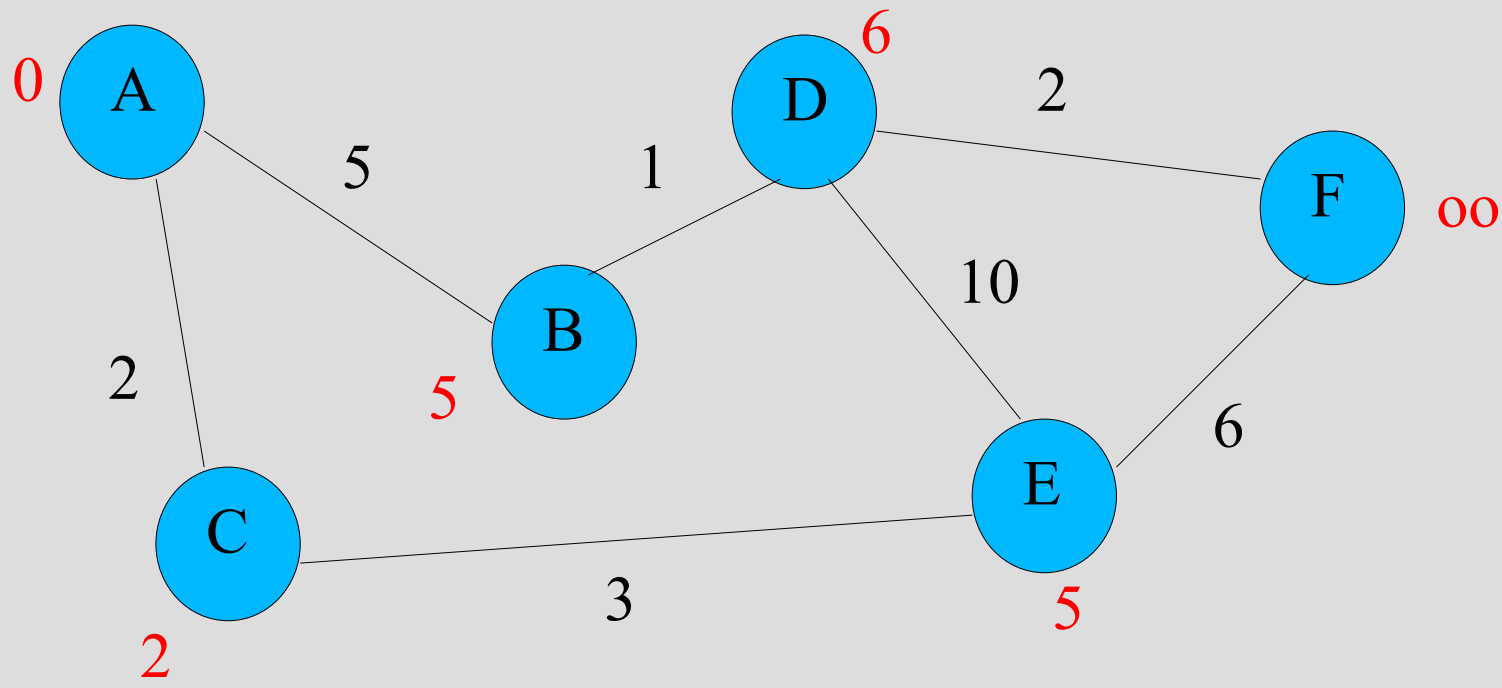
Dijkstra's algorithm

Priority queue: B, E, D, F



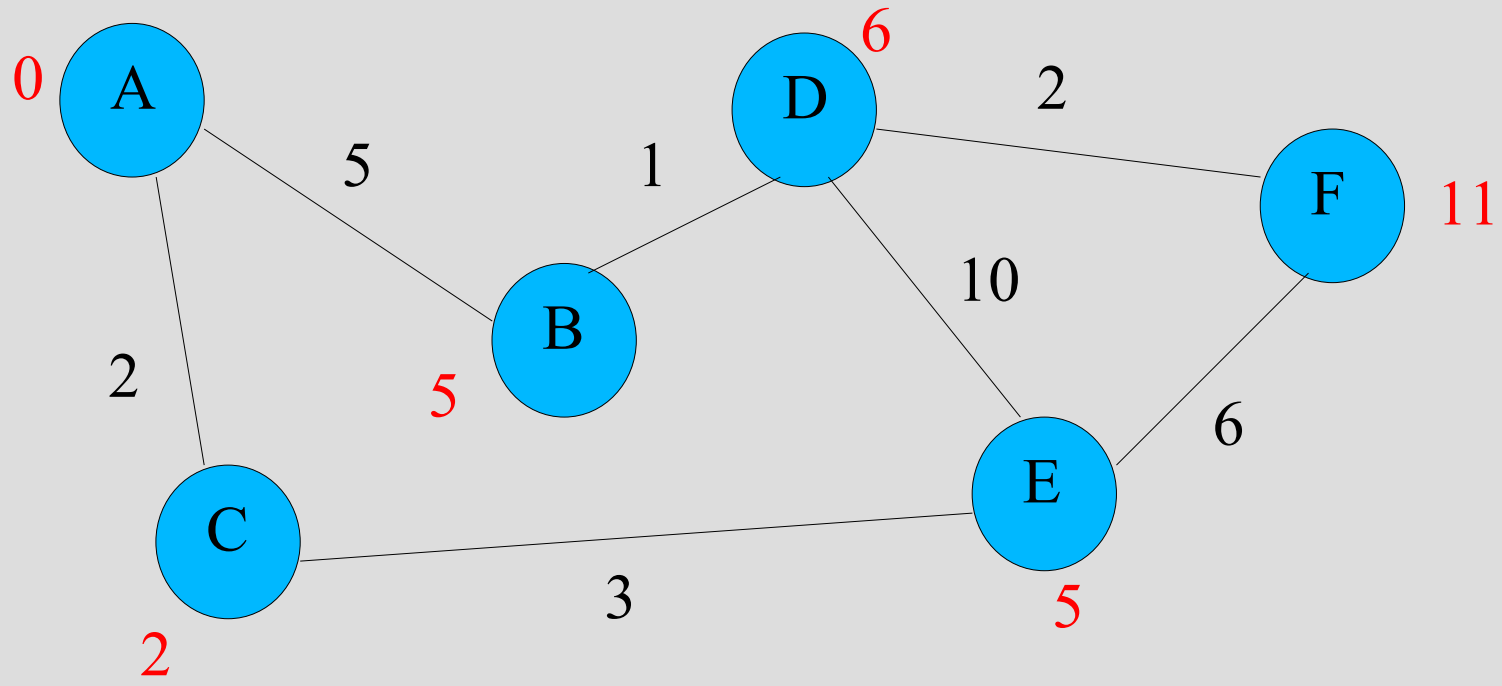
Dijkstra's algorithm

Priority queue: E, D, F



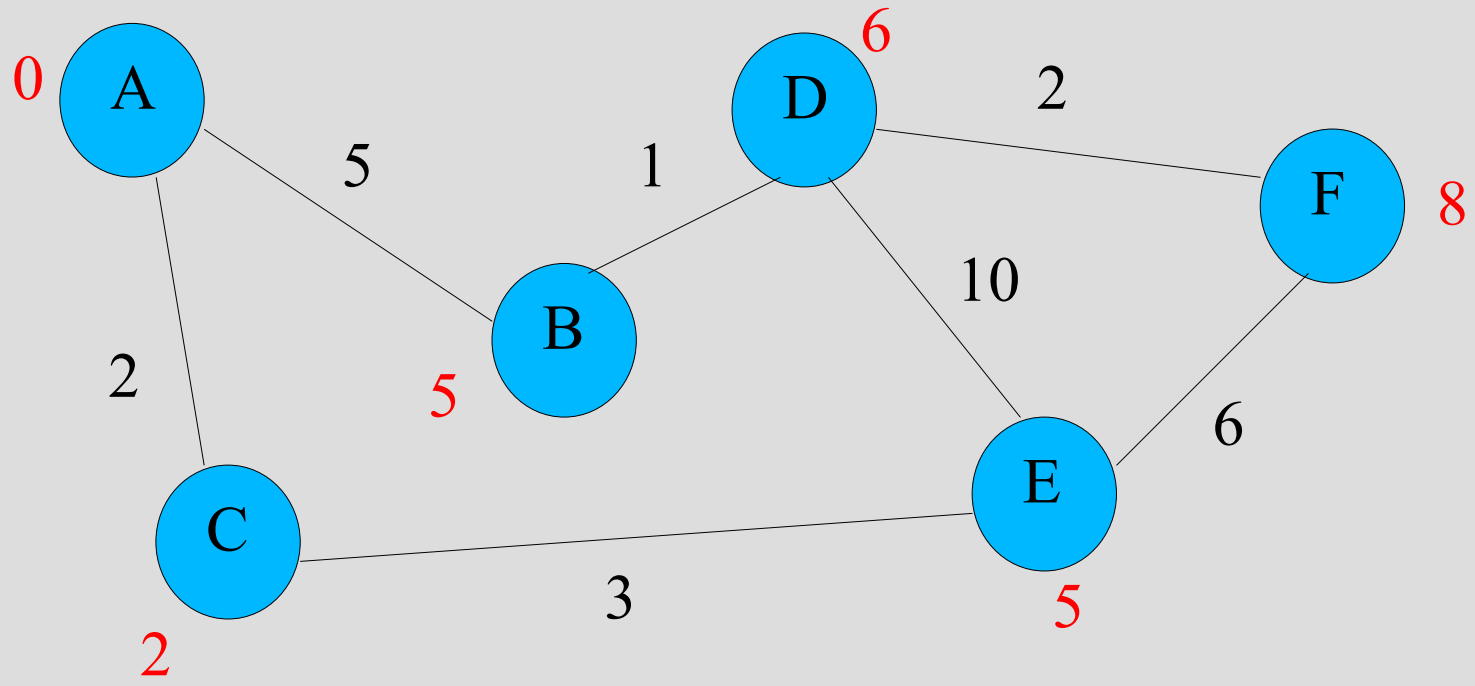
Dijkstra's algorithm

Priority queue: D, F



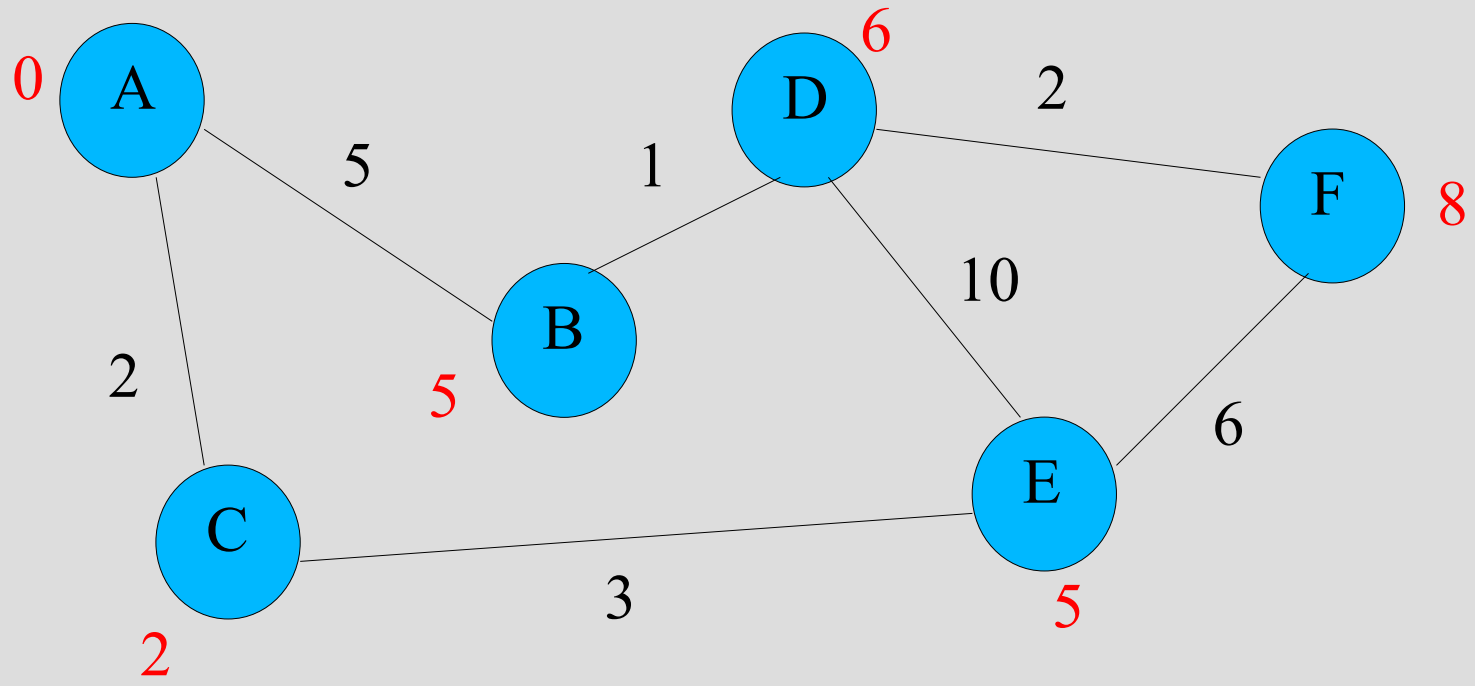
Dijkstra's algorithm

Priority queue: F



Dijkstra's algorithm

Priority queue:



Dijkstra's algorithm

- Time: O((n + e)log n)

Floyd-Warshall Algorithm

Start: d[i][j] = weight of edge from i to j or infinity if no edge
End: d[i][j] = shortest distance from i to j or infinity if no edge

```
for j = 1 to n
    for i = 1 to n
        for k = 1 to n
            d[i][k] = min(d[i][k], d[i][j] + d[j][k])
```

NB: Loop order is very important!

Time: O(n^3)
Space: O(n^2)