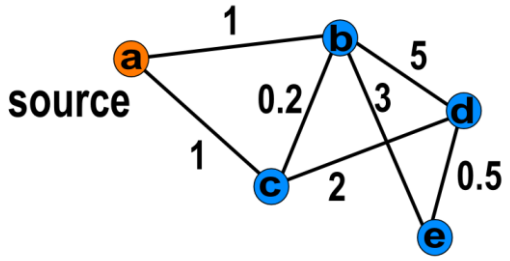
Dijkstra Algorithm

Mustafa Hajij



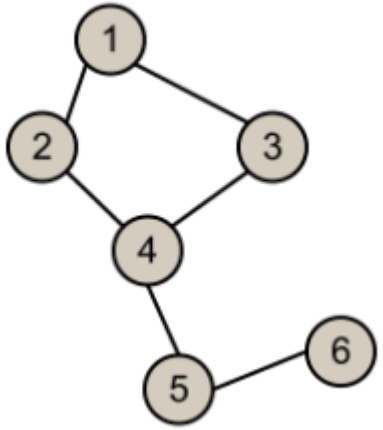
Graphs

A **graph** is an ordered pair (V,E) where,

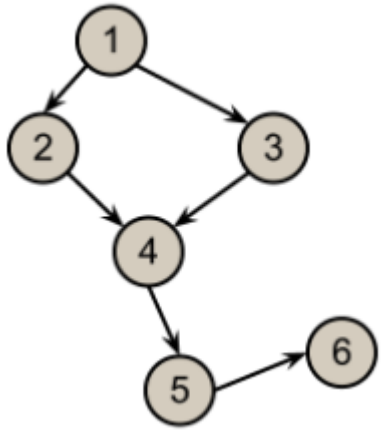
• V is the *vertex set (also node set)* whose elements are the vertices, or *nodes* of the graph.

• E is the *edge set* whose elements are the edges, or connections between vertices, of the graph. If the graph is undirected, individual edges are unordered pairs.

• If the graph is directed, edges are ordered pairs

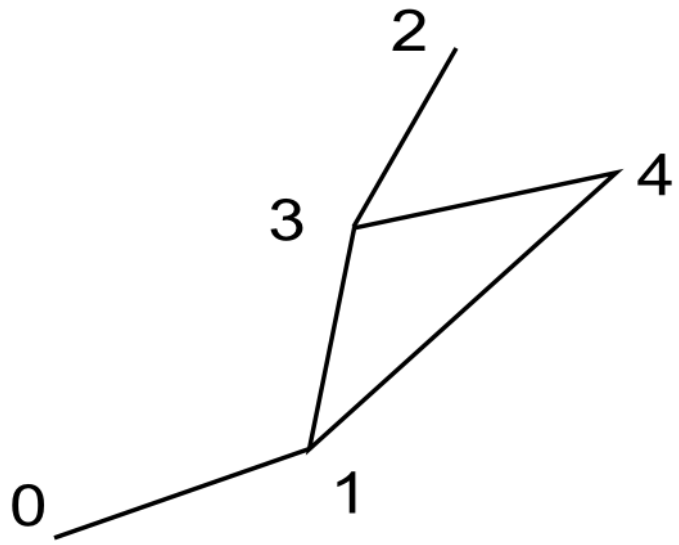


undirected



directed

Graphs representation: list of nodes and edges

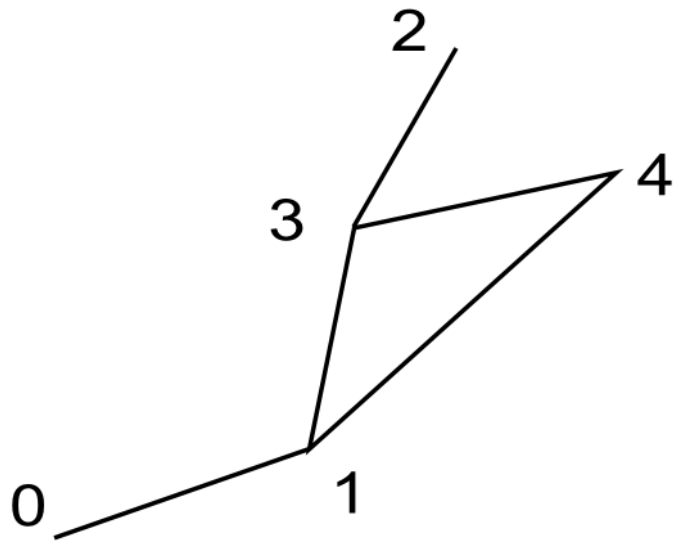


Nodes: [0,1,2,3,4]

Edges: [[0,1], [1,3],[1,4] [3,4], [3,2]]

Note that if the graph is connected, then the list of edges is enough to determine the graph completely.

Graphs representation: list of nodes and edges

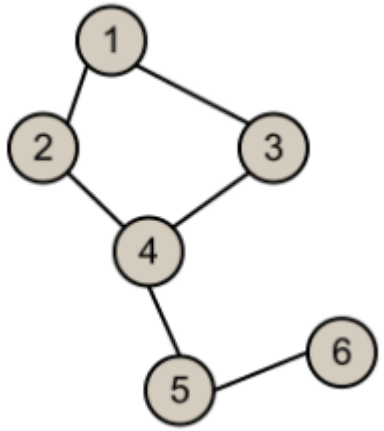


Nodes: [0,1,2,3,4]

Edges: [[0,1], [1,3],[1,4] [3,4], [3,2]]

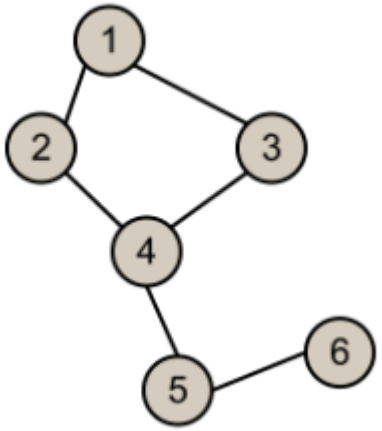
The order of the vertices is important only if the graph is directed.

Graphs representation: adjacency matrix

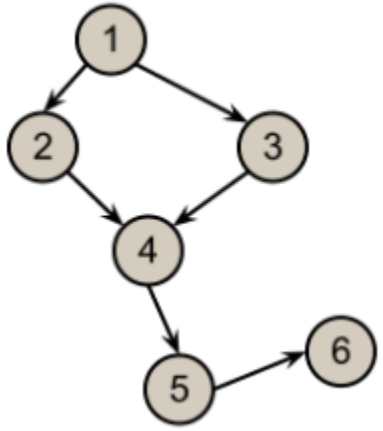


	1	2	3	4	5	6
1	0	1	1	0	0	0
2	1	0	0	1	0	0
3	1	0	0	1	0	0
4	0	1	1	0	1	0
5	0	0	0	1	0	1
6	0	0	0	0	1	0

Graphs representation: adjacency matrix



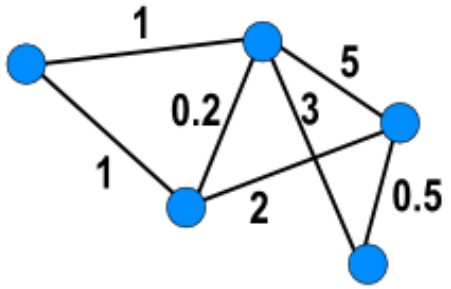
	1	2	3	4	5	6
1	0	1	1	0	0	0
2	1	0	0	1	0	0
3	1	0	0	1	0	0
4	0	1	1	0	1	0
5	0	0	0	1	0	1
6	0	0	0	0	1	0



	1	2	3	4	5	6
1	0	1	1	0	0	0
2	-1	0	0	1	0	0
3	-1	0	0	1	0	0
4	0	-1	-1	0	1	0
5	0	0	0	-1	0	1
6	0	0	0	0	-1	0

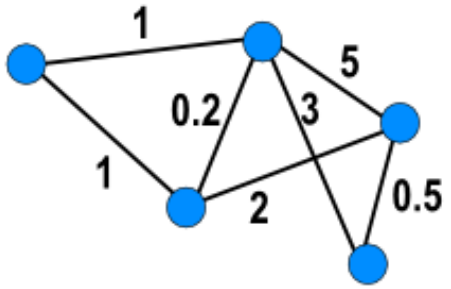
Weighted Graphs

A weighted graph is a graph in which every edge has a weight (non-negative real number)



Weighted Graphs

A weighted graph is a graph in which every edge has a weight (non-negative real number)



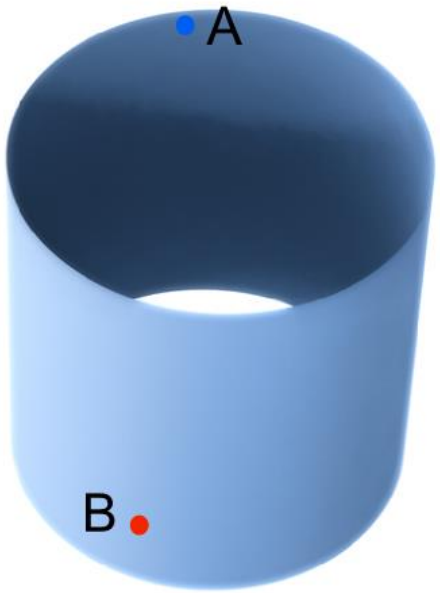
Formally speaking :

A weight function $w: E \rightarrow R^{+}$. In other words, the function w associates to every edge e a positive number (weight) $w(e)$

A weighted graph is a graph G=(V,E) with a weight function $w: E \rightarrow R^{+}$.

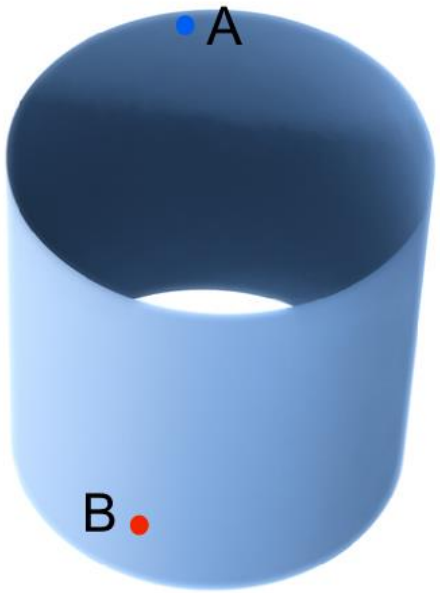
Shortest distance

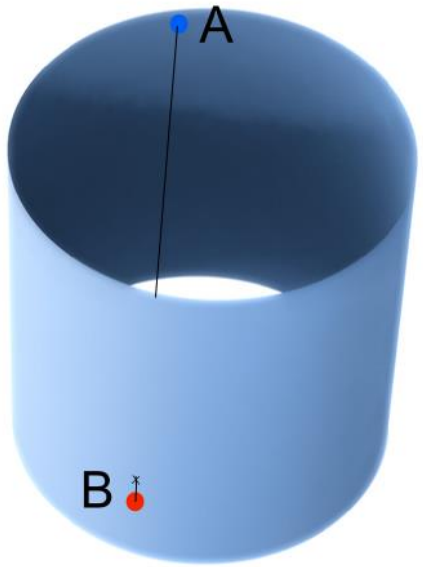
What is the shortest distance between A and B?

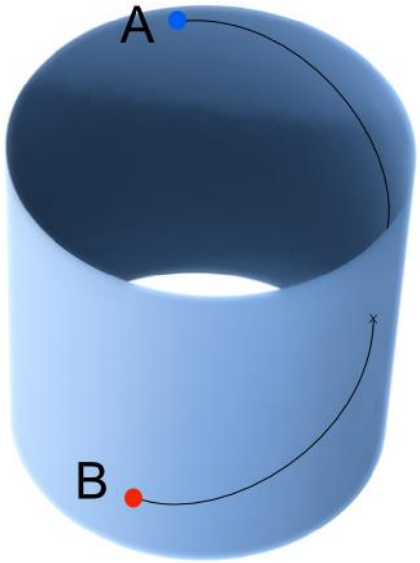


Shortest distance

What is the shortest distance between A and B?





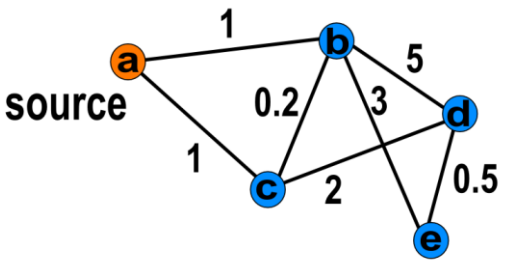


Dijkstra algorithm

The basic Dijkstra algorithm operates on connected, undirected, weighted graph.

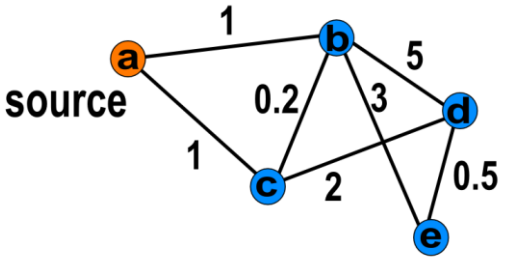
```
1:      Dijkstra(Graph, source):
2:          for each vertex v in Graph:
3:              distance[v] := infinity          // the initial distance from source to any other vertex v is infinity
4:              previous[v] := undefined
5:          distance[source] := 0                // Distance from the source to itself is zero
6:          Q := the set of all nodes in the Graph   // all nodes are going in this container
7:          while Q is not empty:                // main loop
8:              u := the node in Q with smallest distance from the source   (what kind of queue you use here?)
9:              remove u from Q                  //the source will be removed first
10:             for each neighbor v of u:        // v is still in the container Q
11:                 alt := distance[u] + length(u, v)
12:                 if alt < distance[v]                //A shorter path from v to the source has been found
13:                     distance[v] := alt
14:                     previous[v] := u
15:         return distance[], previous[ ]
```

Dijkstra algorithm : Example

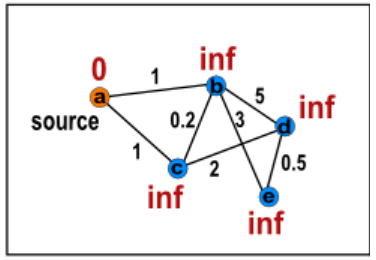


Input : weighted graph
with a source vertex

Dijkstra algorithm : Example

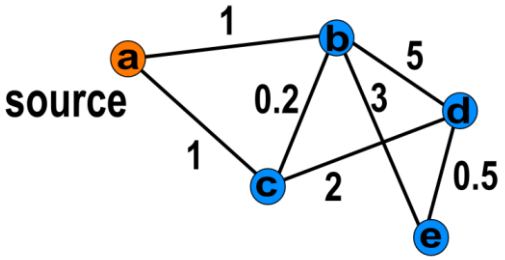


Q={a,b,c,d,e}

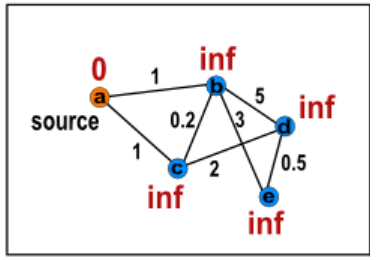


Algorithm starts by initializing the distance to every vertex other than the source to infinity. We also create a queue Q and put in it all vertices of G.

Dijkstra algorithm : Example

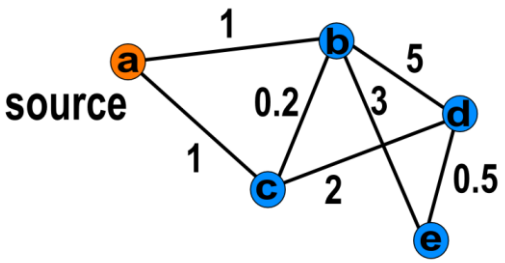


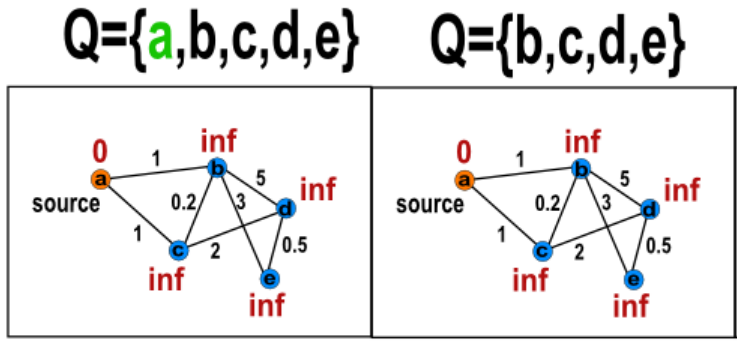
Q={a,b,c,d,e}



When we enter the while loop we dequeue the element in Q with shortest distance to source. In this case it is a.

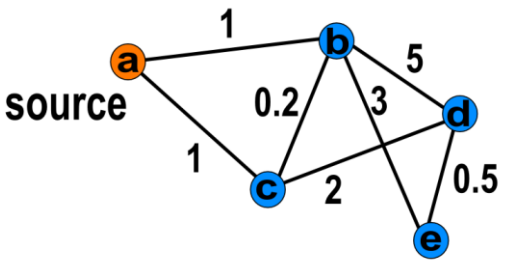
Dijkstra algorithm : Example

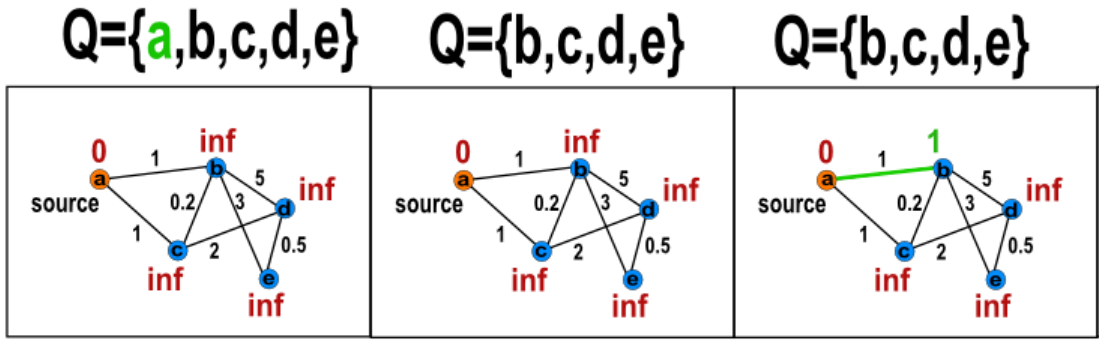




When we enter the while loop we dequeue the element in Q with shortest distance to source. In this case it is a.

Dijkstra algorithm : Example

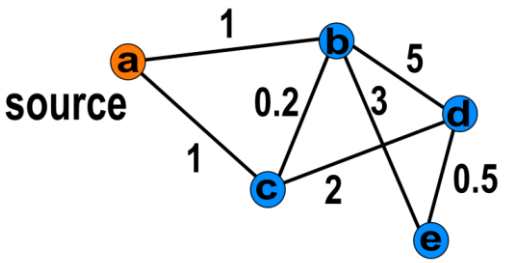




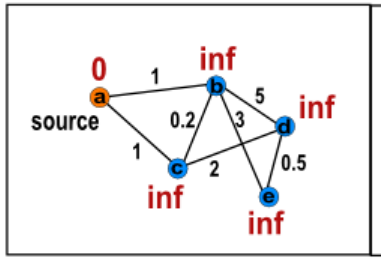
Now we visit all neighbors of a and update the distance to them:

```
for each neighbor v of u:
                alt := distance[u] + length(u, v)
                if alt < distance[v]
                        distance[v] := alt
```

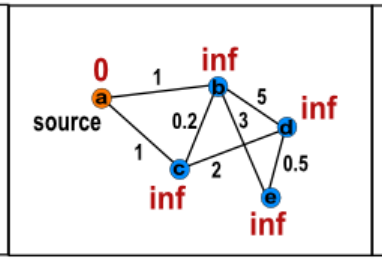
Dijkstra algorithm : Example



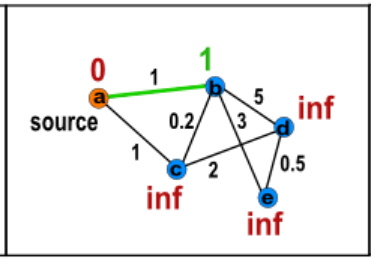
Q={a,b,c,d,e}



Q={b,c,d,e}



Q={b,c,d,e}

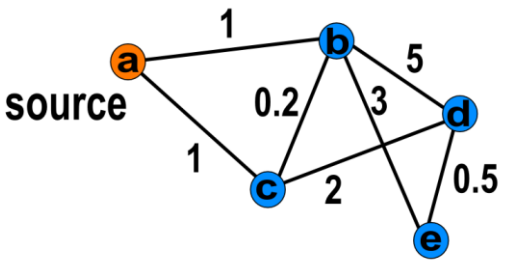


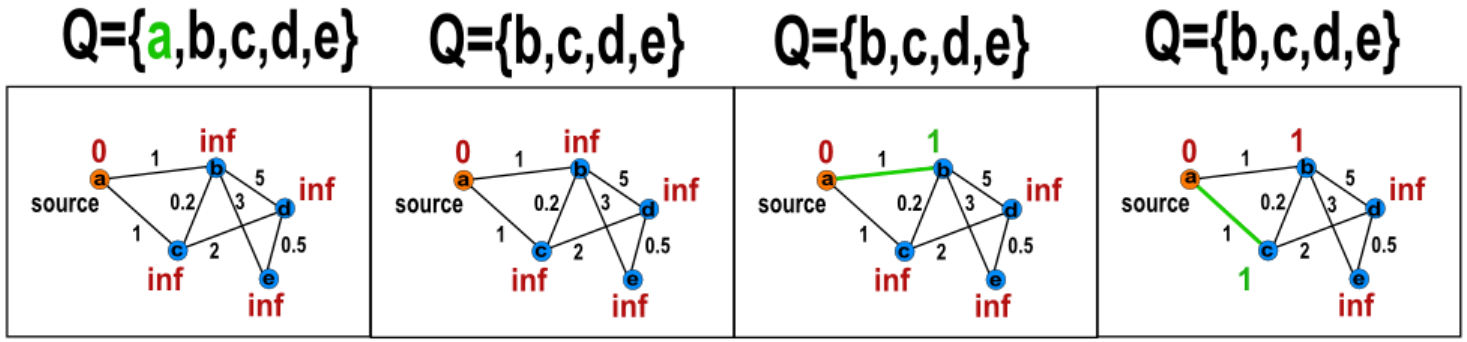
In this case we update the distance to b to 1.

Now we visit all neighbors of a and update the distance to them:

```
for each neighbor v of u:
                alt := distance[u] + length(u, v)
                if alt < distance[v]
                        distance[v] := alt
```

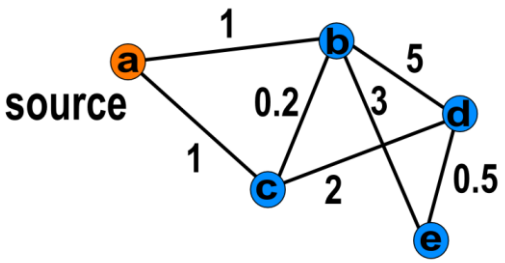
Dijkstra algorithm : Example

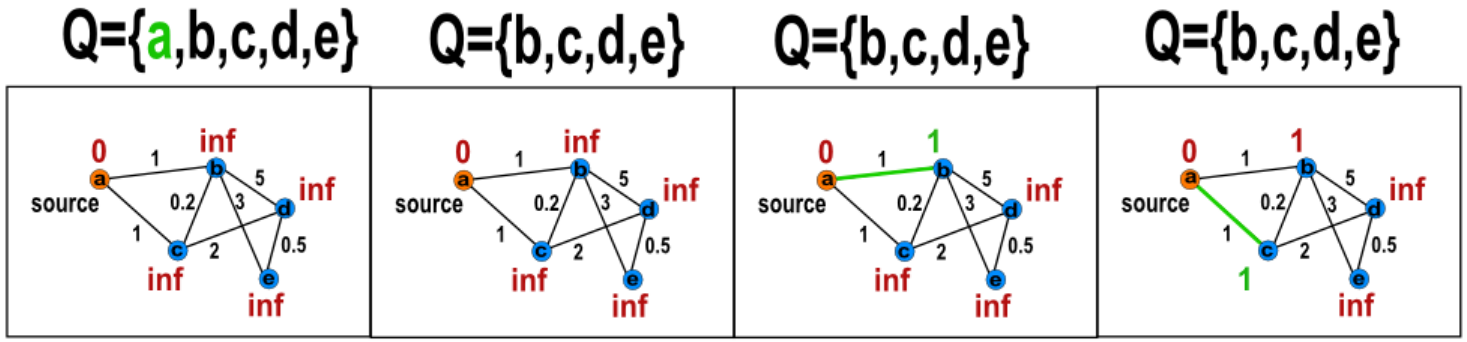




Here we update the distance to c to be 1 as well

Dijkstra algorithm : Example

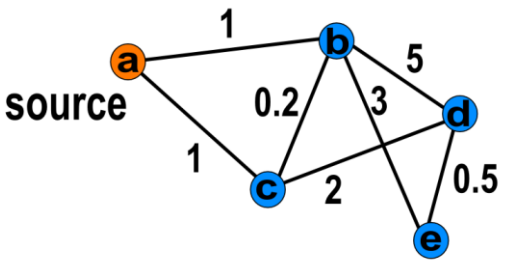




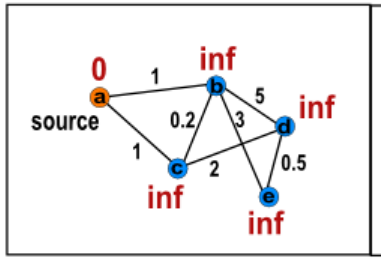
Here we update the distance to c to be 1 as well

At this stage all neighbors of a have been visited so we check the queue again: if is not empty we start the process again.

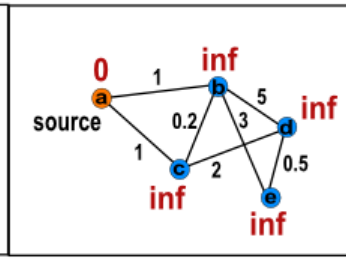
Dijkstra algorithm : Example



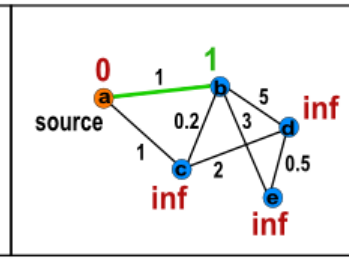
Q={a,b,c,d,e}



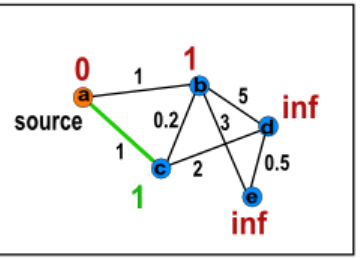
Q={b,c,d,e}



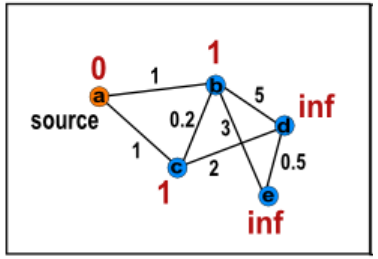
Q={b,c,d,e}



Q={b,c,d,e}

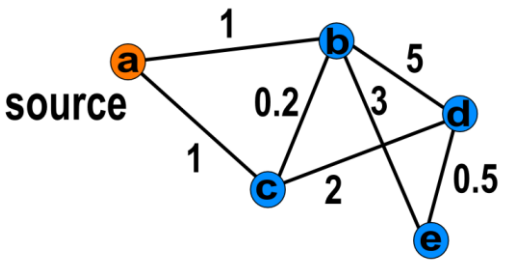


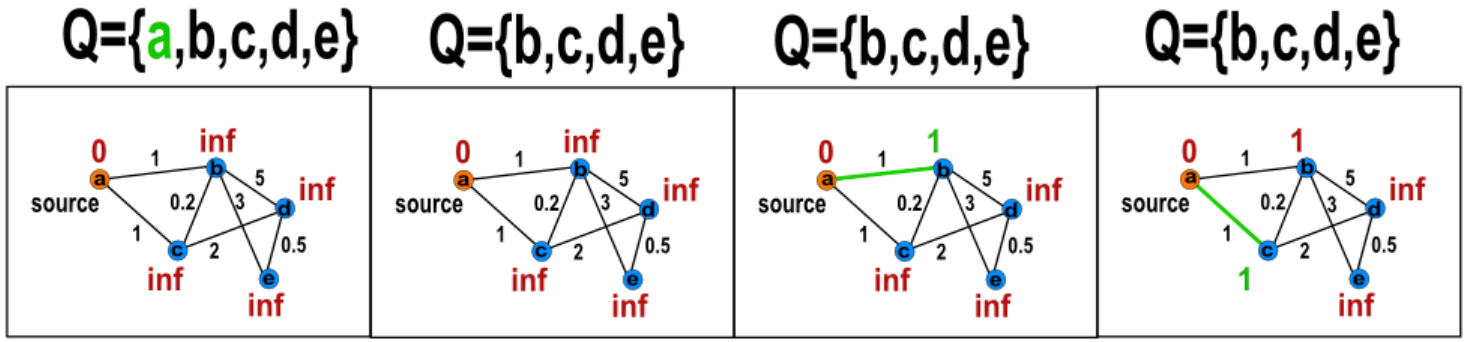
Q={b,c,d,e}

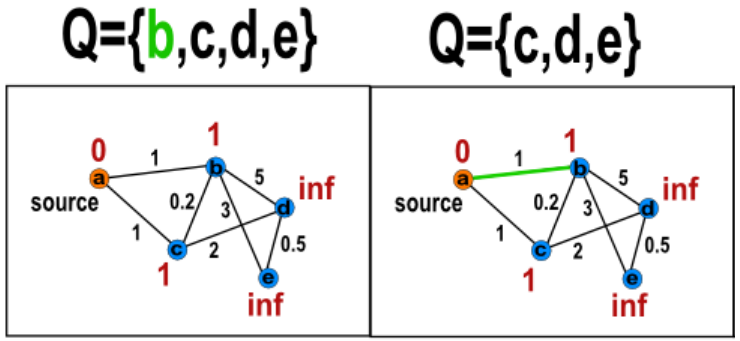


We select b (the closest element to a so far)

Dijkstra algorithm : Example



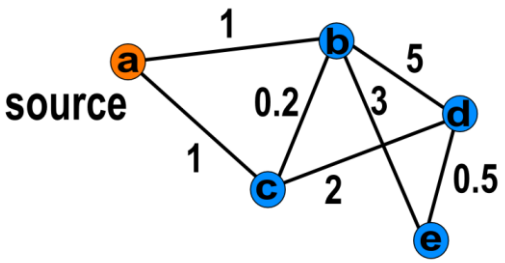


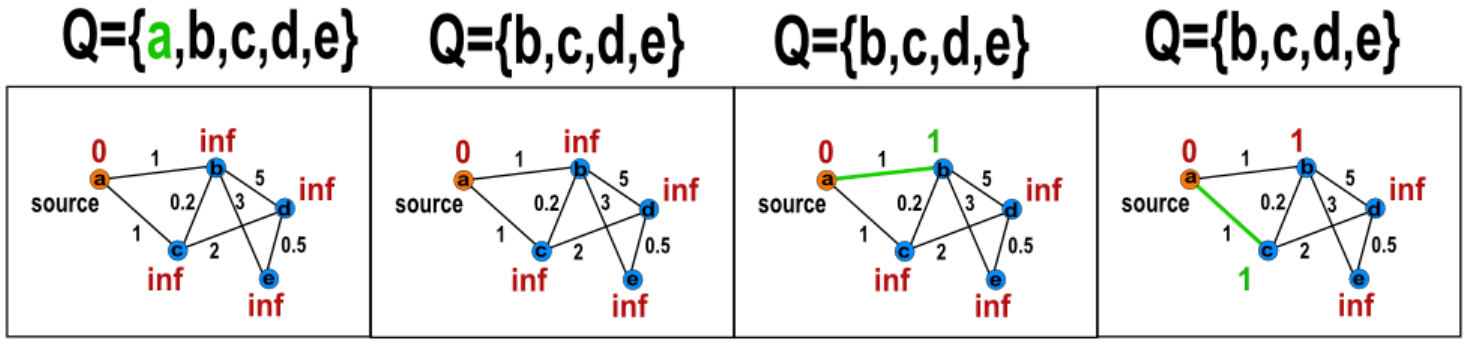


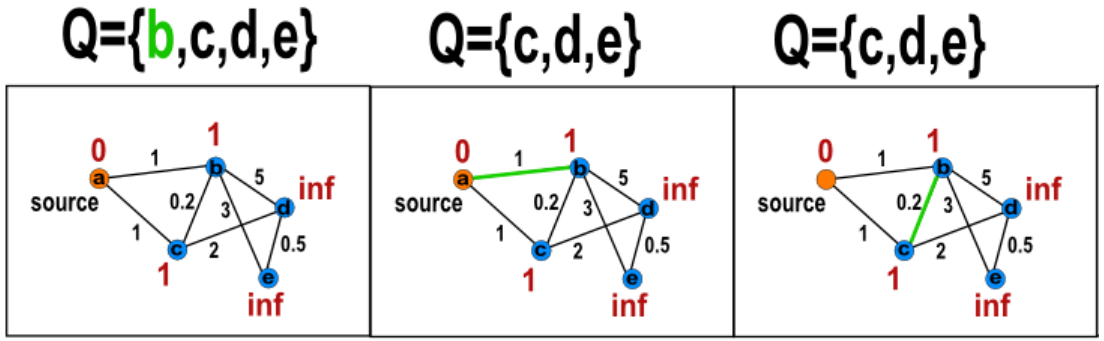
Remove b from the queue and start visiting all its neighbors and update the distance.

In this case the distance does not update—why ?

Dijkstra algorithm : Example

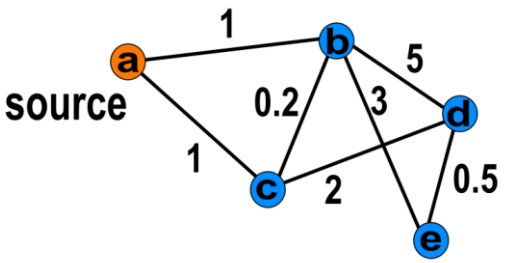




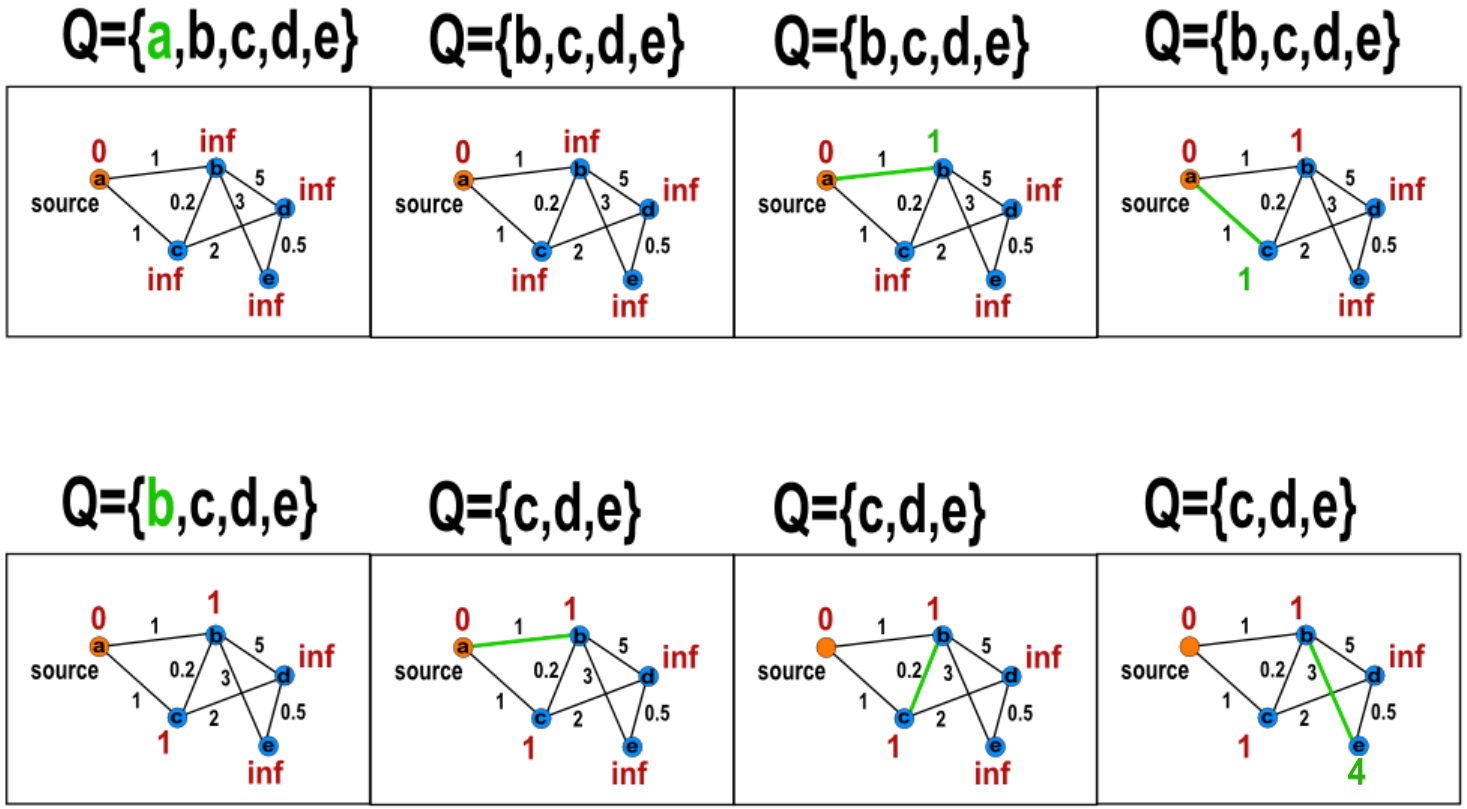


The distance here also does not update

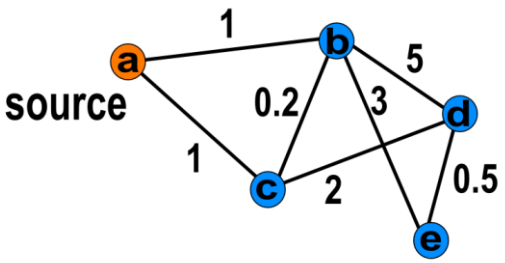
Dijkstra algorithm : Example



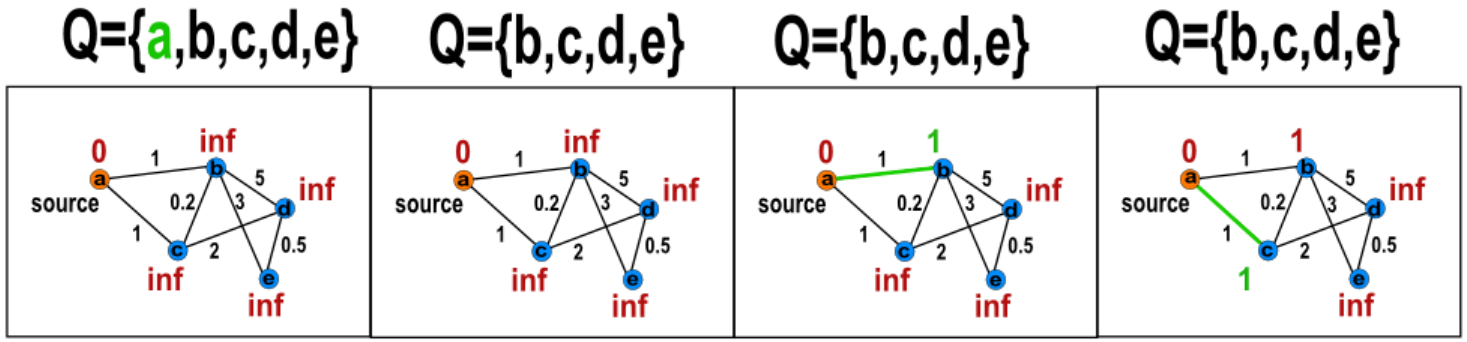
Here we update the distance from a to e to be 4

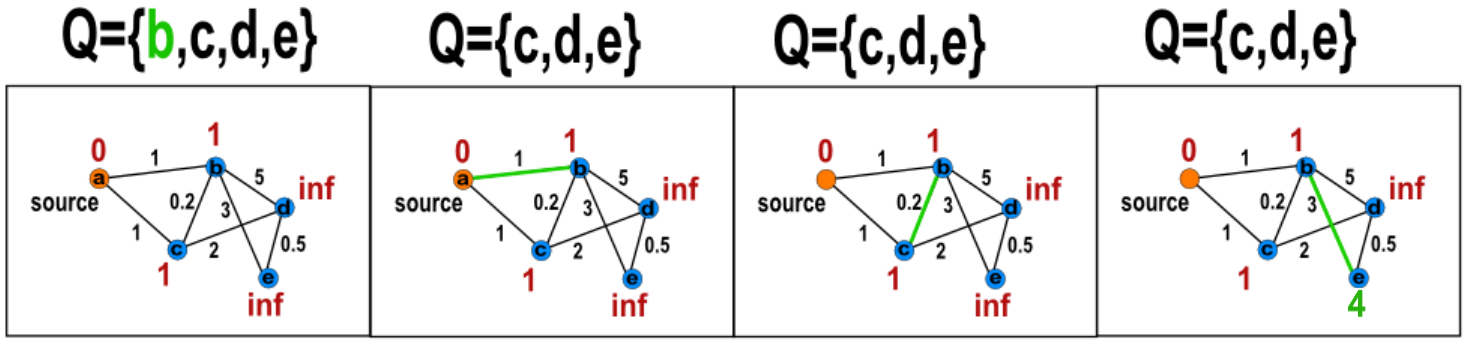


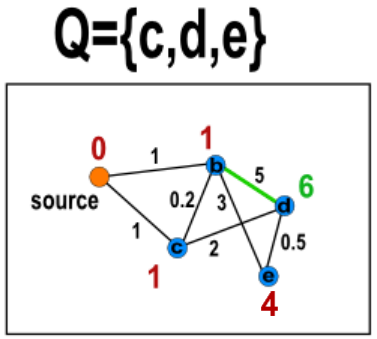
Dijkstra algorithm : Example



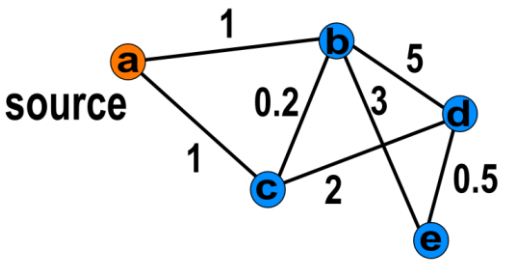
Update the distance from a to d



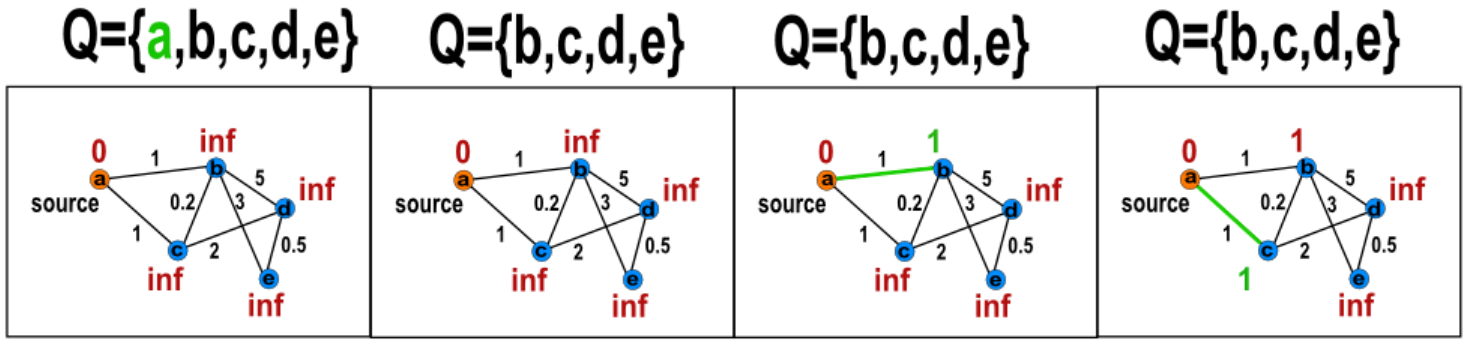




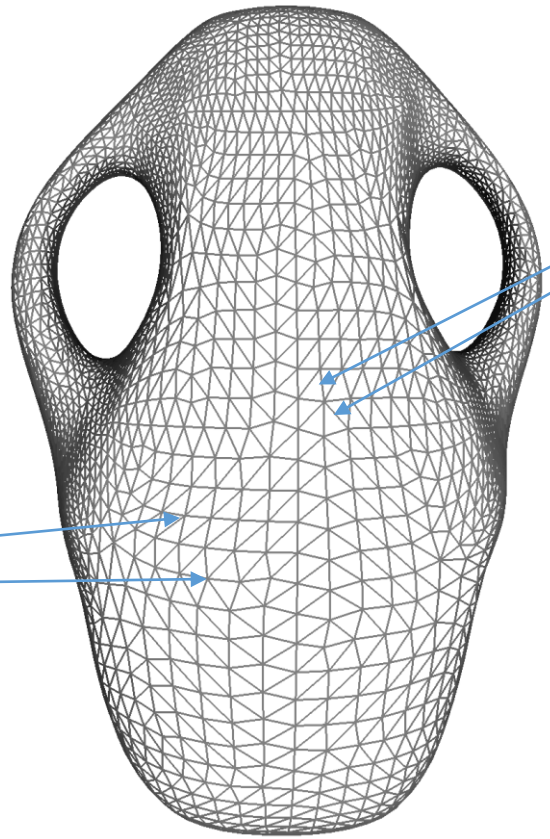
Dijkstra algorithm : Example



And so on

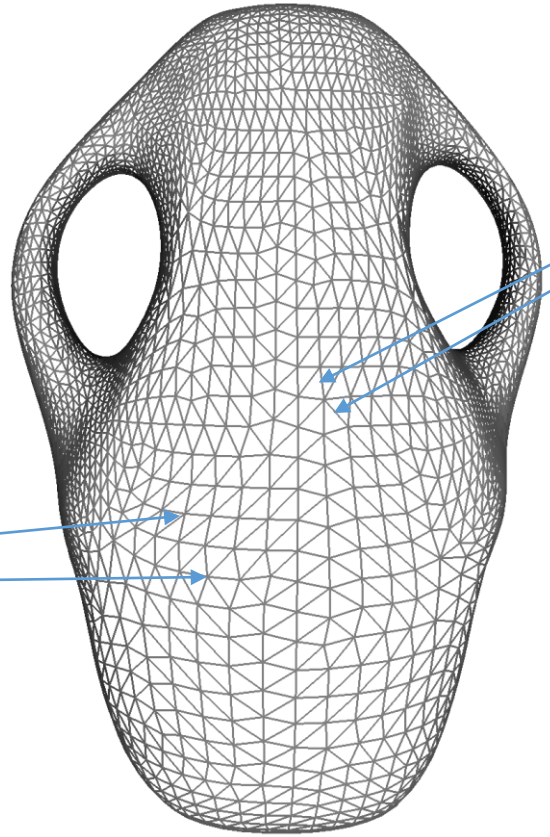


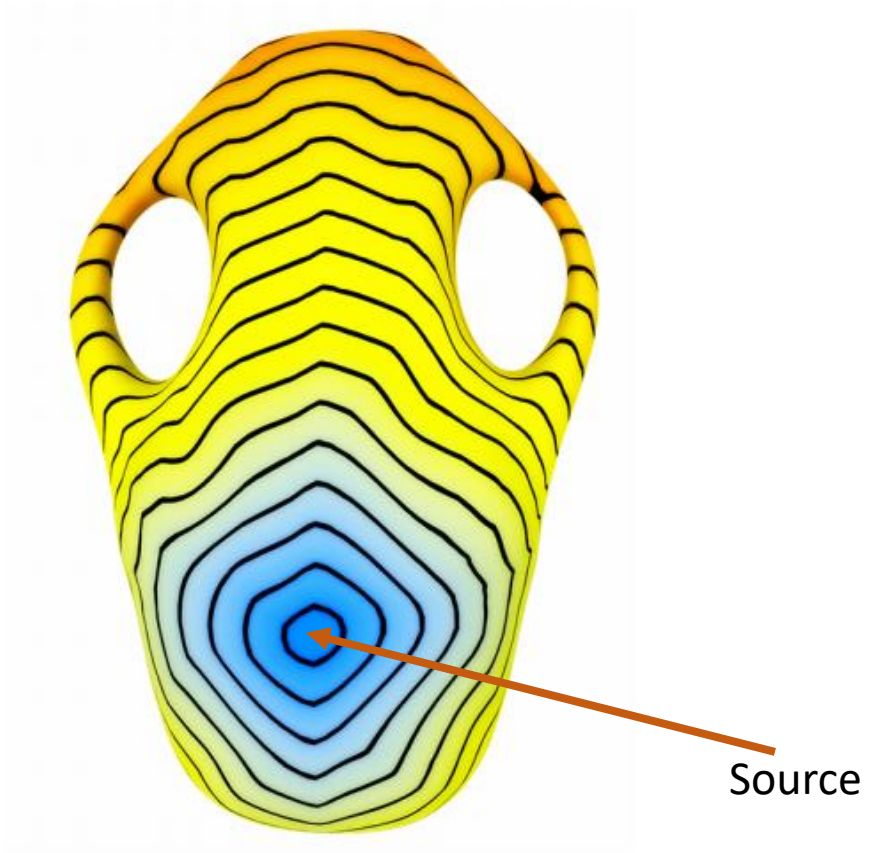
Example



We can view a mesh as a graph and apply Dijkstra algorithm on it.

Example





We can view a mesh as a graph and apply Dijkstra algorithm on it.

Blue indicates the regions closest to the source