Introduction to Algorithms

Shortest Paths

Edsger W. Dijkstra (1930-2002)

www.math.bas.bg/…/EWDwww.jpg

- Dutch Computer Scientist
- Received Turing Award for contribution to developing programming languages

Contributed to :

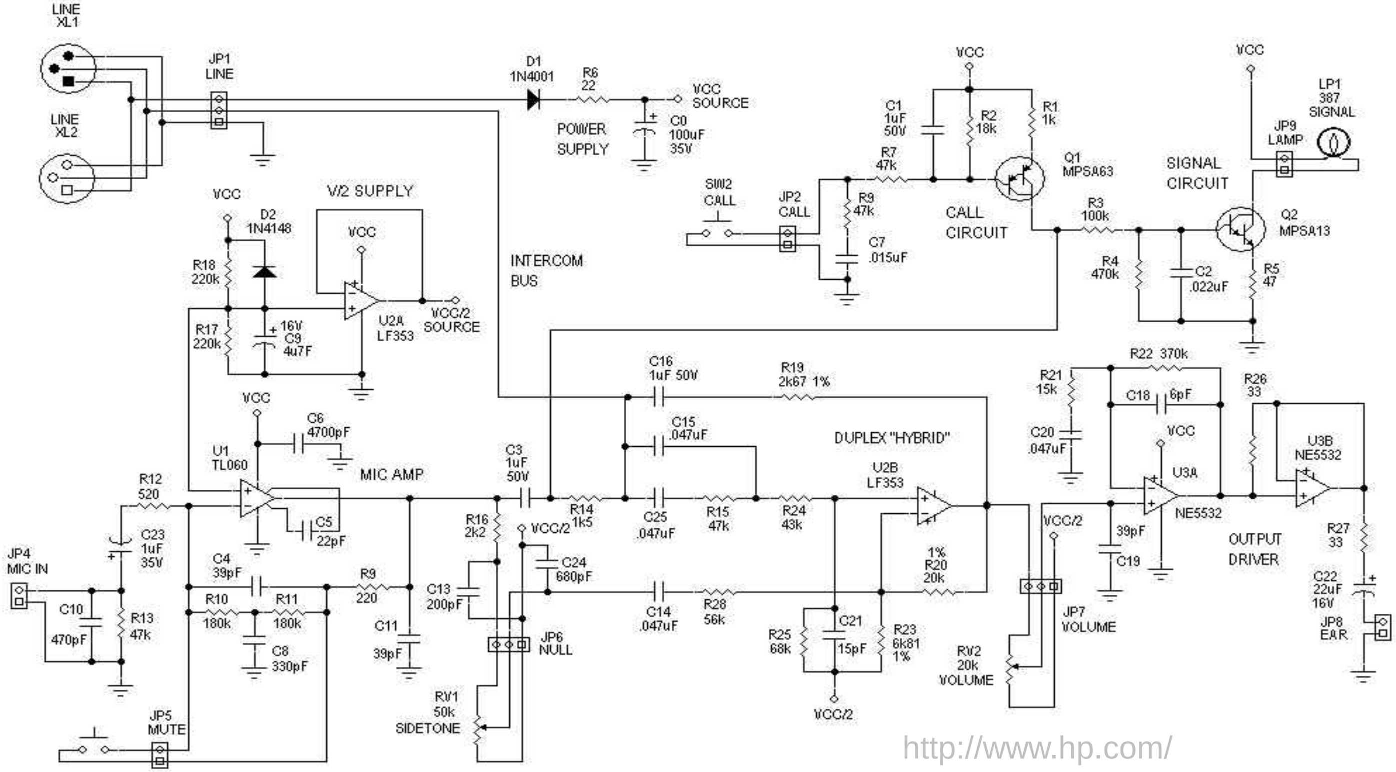
- Shortest path-algorithm, also known as Dijkstra's algorithm;
- Reverse Polish Notation and related Shunting yard algorithm;
- The multiprogramming system;
- Banker's algorithm;
- Self-stabilization – an alternative way to ensure the reliability of the system

Shortest Path

- Given a weighted directed graph, one common problem is finding the shortest path between two given vertices
- Recall that in a weighted graph, the *length* of a path is the sum of the weights of each of the edges in that path

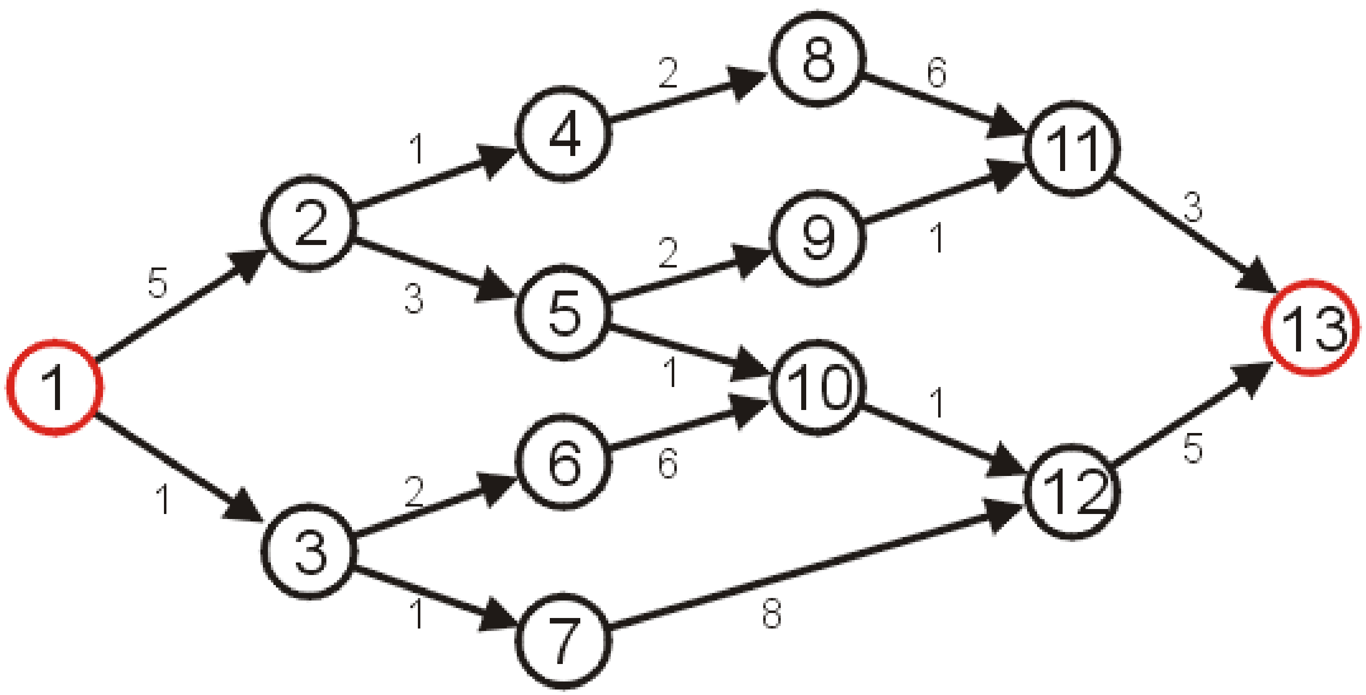
Applications

- One application is circuit design: the time it takes for *a change in input to affect an output* depends on the shortest path



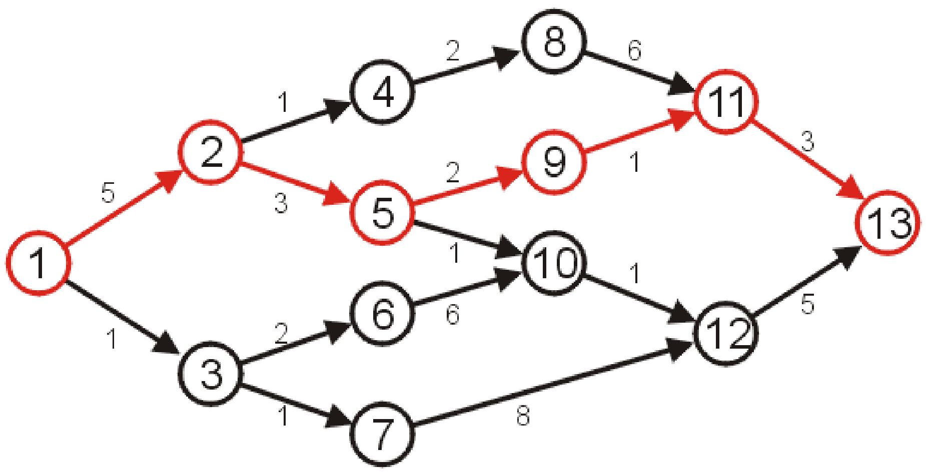
Shortest Path

- Given the graph below, suppose we wish to find the shortest path from vertex 1 to vertex 13



Shortest Path

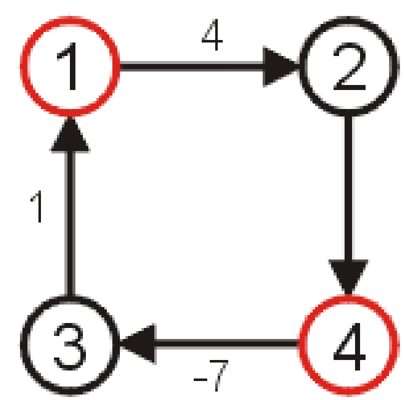
- After some consideration, we may determine that the shortest path is as follows, with length 14



- Other paths exists, but they are longer

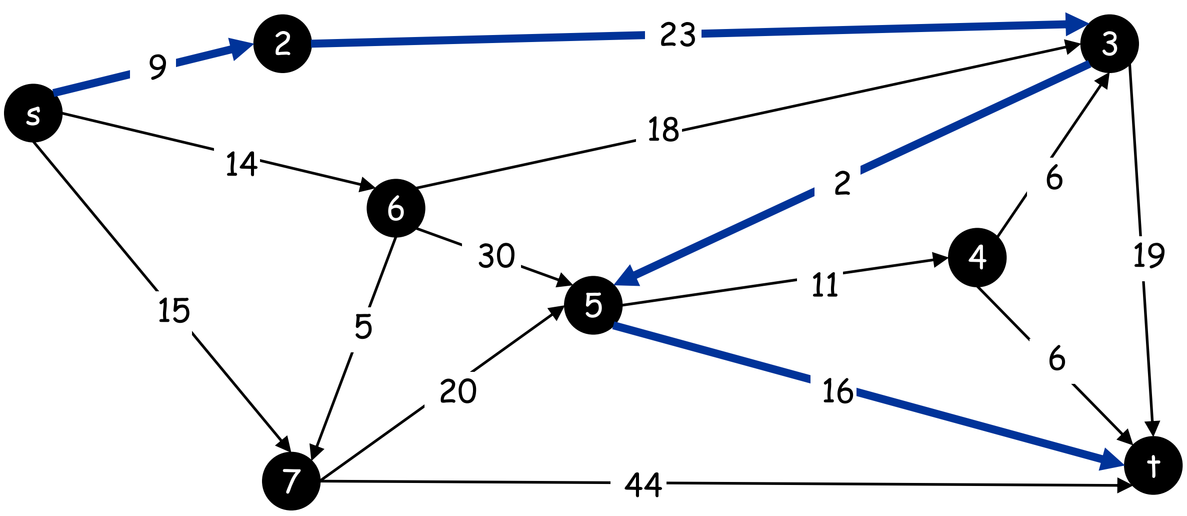
Negative Cycles

- Clearly, if we have negative vertices, it may be possible to end up in a cycle whereby each pass through the cycle decreases the total *length*
- Thus, a shortest length would be undefined for such a graph
- Consider the shortest path from vertex 1 to 4...
- consider non-negative weights only



Shortest Path Example

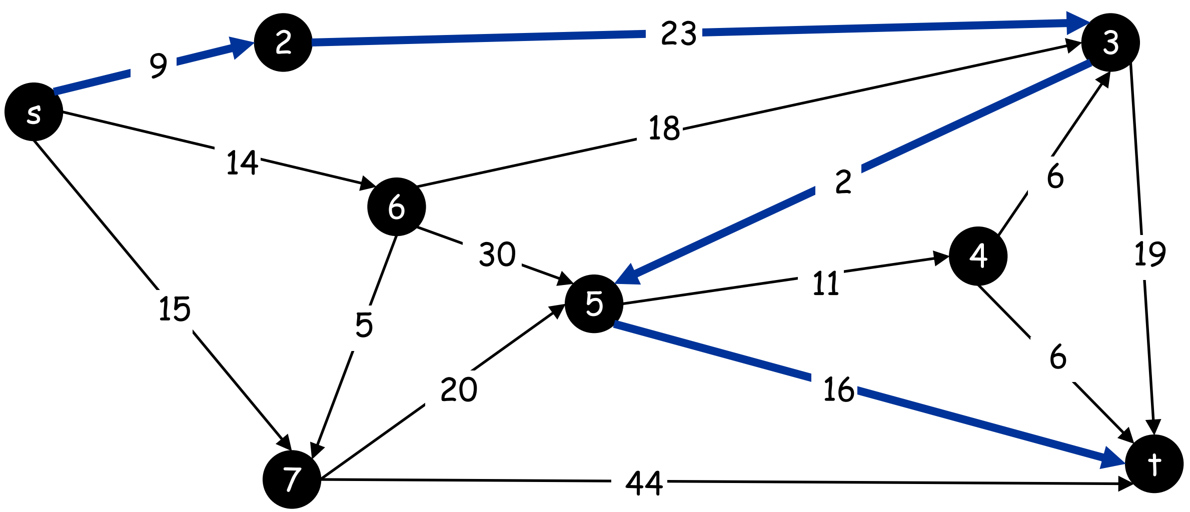
- Given:
  - Weighted Directed graph G = (V, E)
  - Source *s*, destination *t*
- Find shortest directed path from *s* to *t*



Cost of path s-2-3-5-t
= 9 + 23 + 2 + 16
= 48

Discussion…

- How many possible paths are there from *s* to *t*?
- Any suggestions on how to reduce the set of possibilities?
- Can we safely ignore cycles? If so, how?
- Can we determine a lower bound on the complexity?

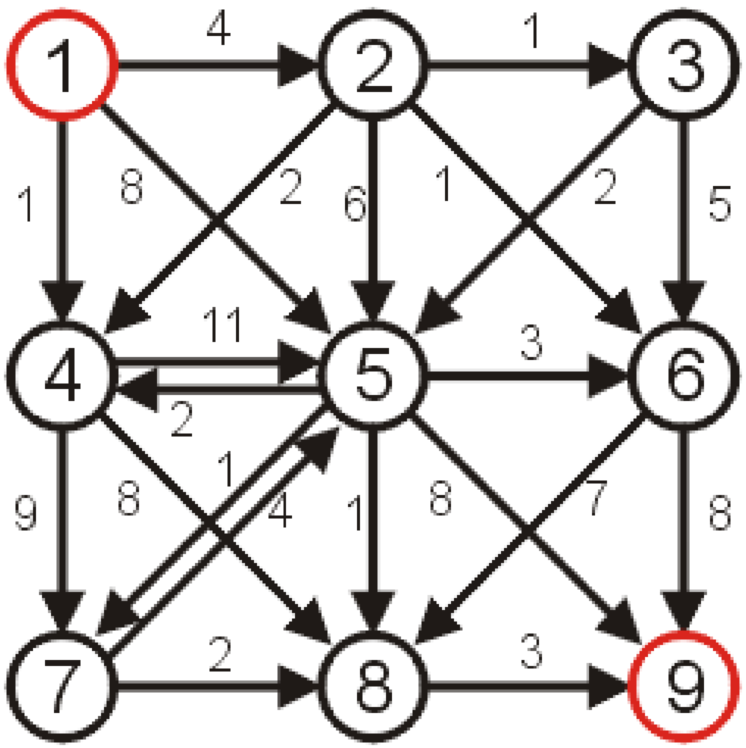


Dijkstra's Algorithm

- Works when all of the weights are positive.
- Provides the shortest paths from a source to **all** other vertices in the graph.
  - Can be terminated early once the shortest path to *t* is found if desired

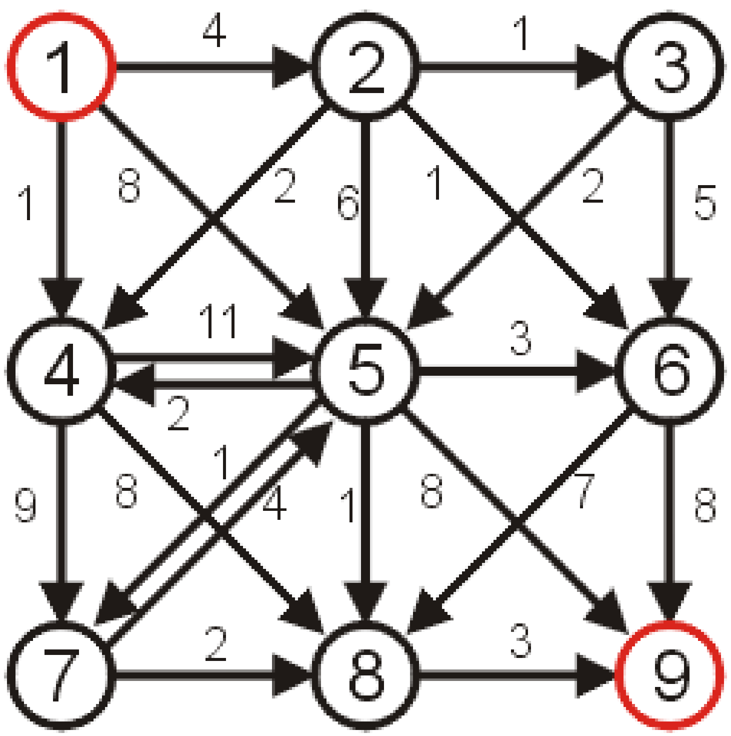
Shortest Path

- Consider the following graph with positive weights and cycles.



Dijkstra's Algorithm

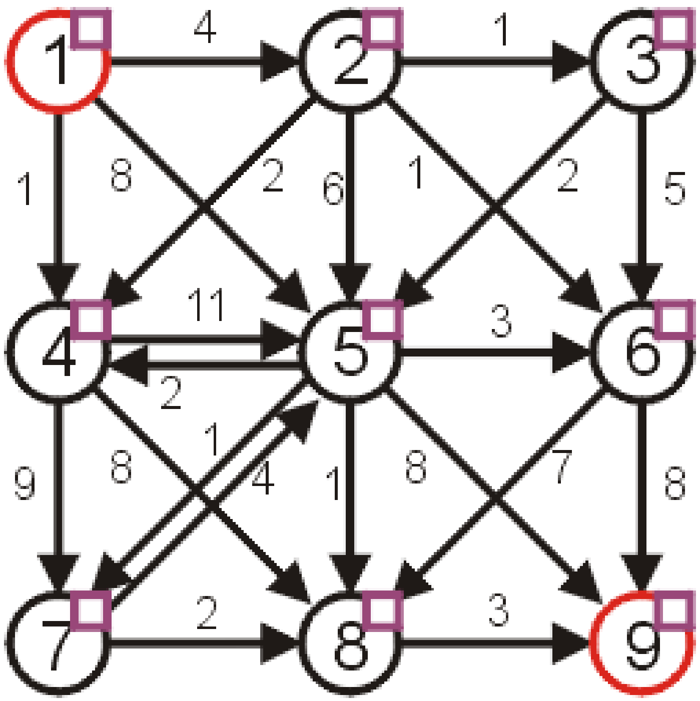
- A first attempt at solving this problem might require an array of Boolean values, all initially false, that indicate whether we have found a path from the source.



1	F
2	F
3	F
4	F
5	F
6	F
7	F
8	F
9	F

Dijkstra's Algorithm

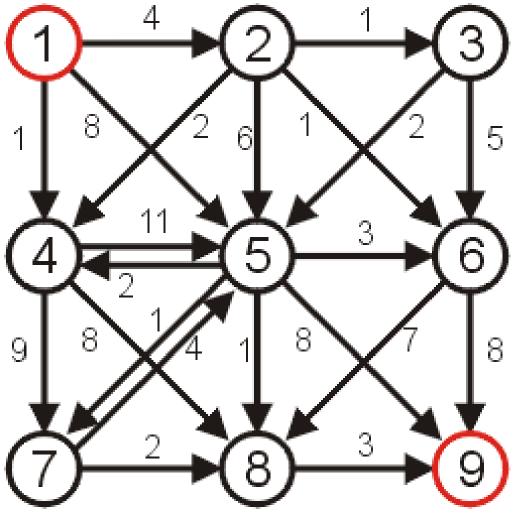
- Graphically, we will denote this with check boxes next to each of the vertices (initially unchecked)



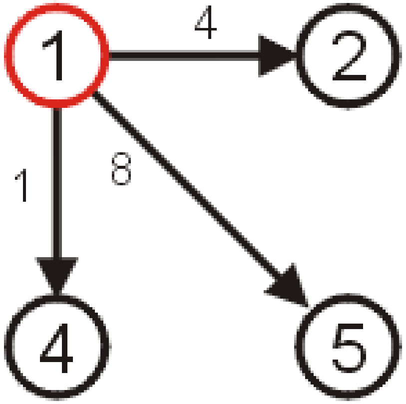
Dijkstra's Algorithm

- work bottom up:
  - Note that if the starting vertex has any adjacent edges, then there will be one vertex that is the shortest distance from the starting vertex. This is the shortest reachable vertex of the graph.
- then try to extend any ***existing*** paths to new vertices.
- Initially, start with the path of length 0
  - this is the trivial path from vertex 1 to itself

Dijkstra's Algorithm

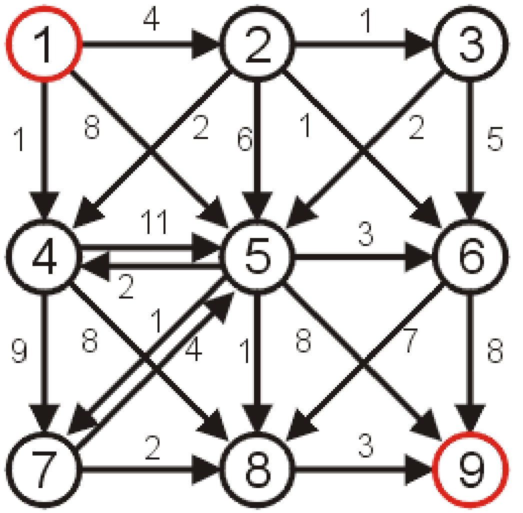


- extending the paths,
  - (1, 2)   length 4
  - (1, 4)   length 1
  - (1, 5)   length 8



The *shortest* path so far is (1, 4) which is of length 1

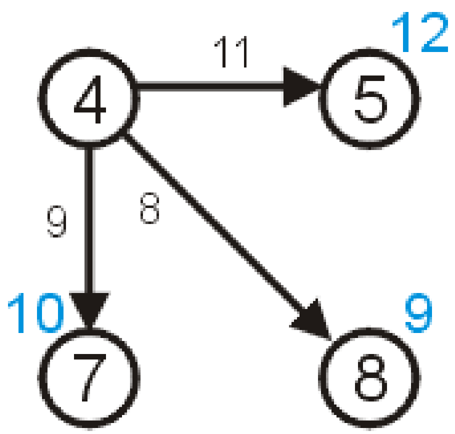
Dijkstra’s Algorithm

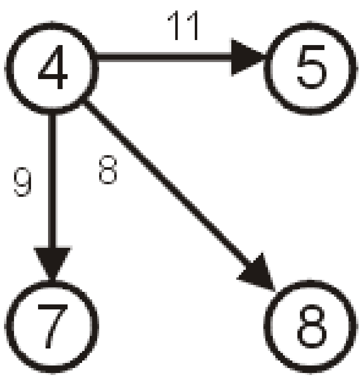


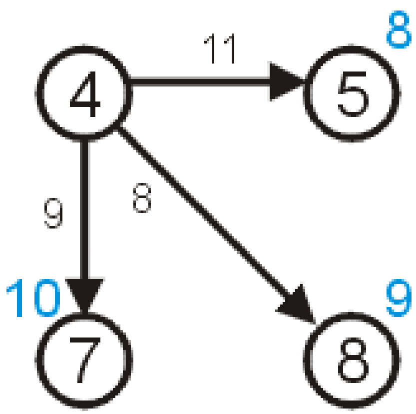
- Remembering length(1, 4)= 1

→ examine vertex 4; to get :

  – (1, 4, 5) length 12
  – (1, 4, 7) length 10
  – (1, 4, 8) length 9

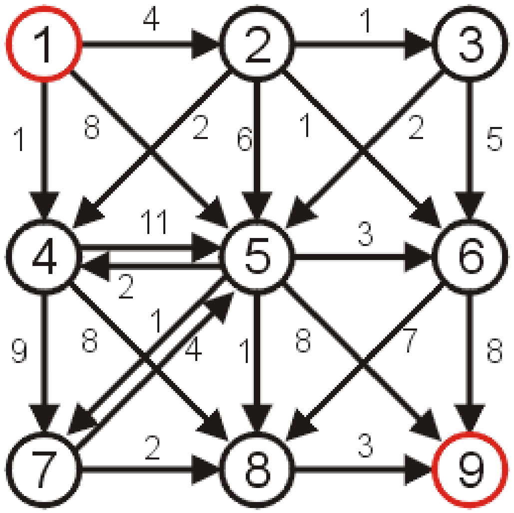






However, recalling length(1, 5)= 8

Dijkstra’s Algorithm



- knowing that:
  - There exist paths from vertex 1 to vertices {2,4,5,7,8}
  - shortest path (1,4) is of length 1
  - shortest path from 1 to the other vertices {2,5,7,8} is at most the length listed in the table

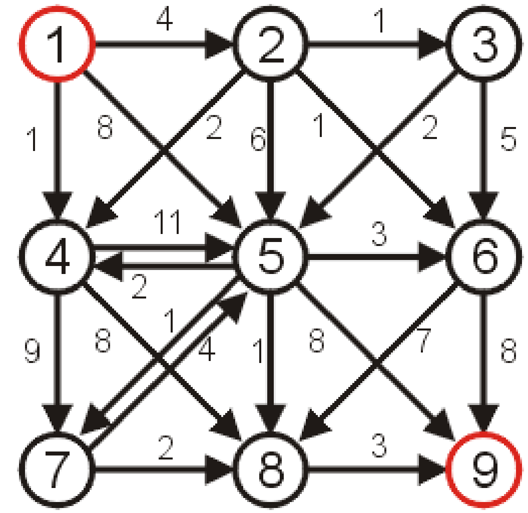
Vertex	Length
1	0
2	4
4	1
5	8
7	10
8	9

Relaxation

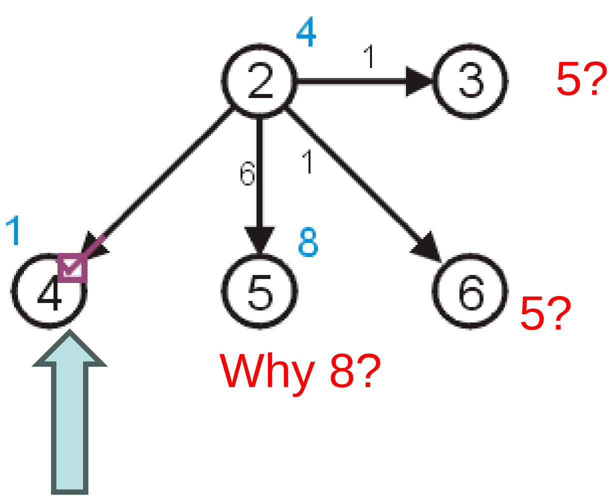
- Maintaining this shortest discovered distance d[*v*] is called **relaxation**:

```
Relax(u,v,w) {
  if (d[v] > d[u]+w) then
    d[v]=d[u]+w;
                }
```

Dijkstra’s Algorithm

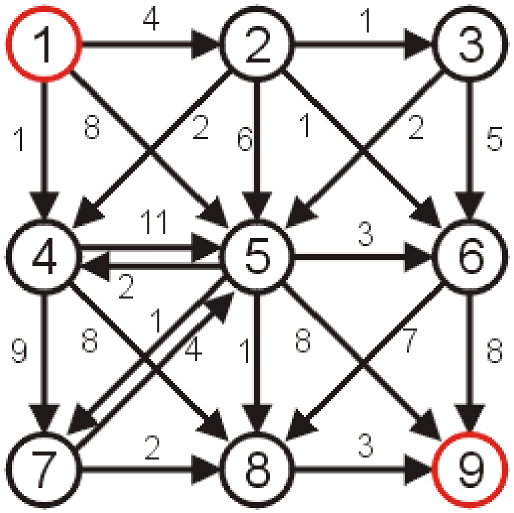


- always take the next unvisited vertex which has the current shortest path from the starting vertex in the table
- This is vertex 2

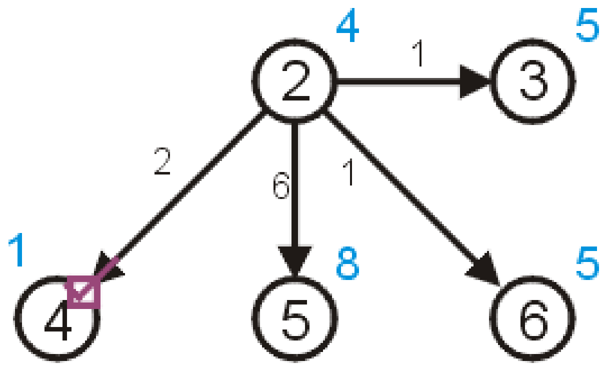


Vertex	Length
1	0
2	4
4	1
5	8
7	10
8	9

Dijkstra’s Algorithm



- can try to update the shortest paths to vertices 3 and 6 (both of length 5)
  However:
  – there already exists a path of length 8 < 10 to vertex 5 (10 = 4 + 6)
  – we already know the shortest path to 4 is 1



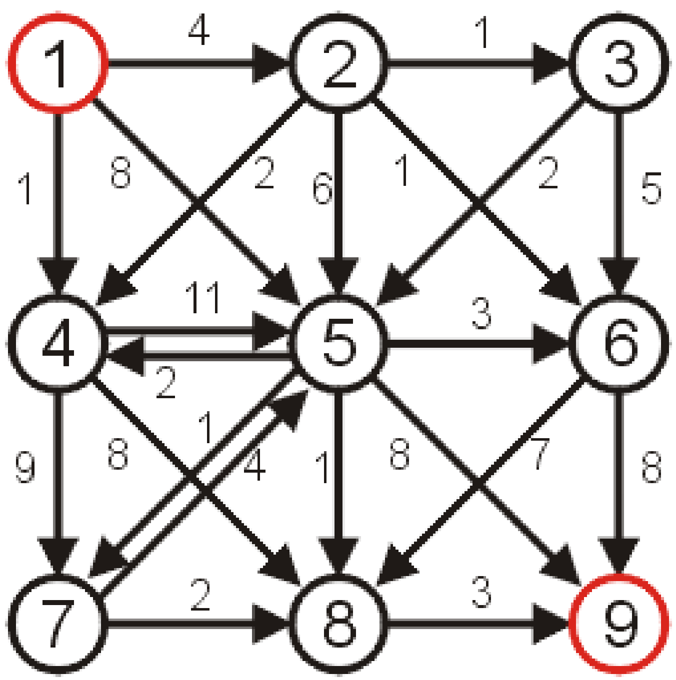
→ keep track of the predecessor used to reach the vertex on the shortest path

Dijkstra’s Algorithm

- To keep track of those vertices to which no path has reached, we can assign those vertices an initial distance of either
  - infinity (∞),
  - a number larger than any possible path, or
  - a negative number
- For demonstration purposes, we will use ∞

Dijkstra's Algorithm

- store a table of pointers, each initially 0
- update as per distance



1	0
2	0
3	0
4	0
5	0
6	0
7	0
8	0
9	0

Dijkstra's Algorithm

- Assumptions:

  - display the reference to the preceding vertex by a red arrow
    - if the distance to a vertex is ∞, there will be no preceding vertex
    - otherwise, there will be exactly one preceding vertex

Dijkstra’s Algorithm

- Initialization:
  - set the current distance to the initial vertex as 0
  - for all other vertices, set the current distance to ∞
  - all vertices are initially marked as unvisited
  - set the previous pointer for all vertices to null

Dijkstra’s Algorithm

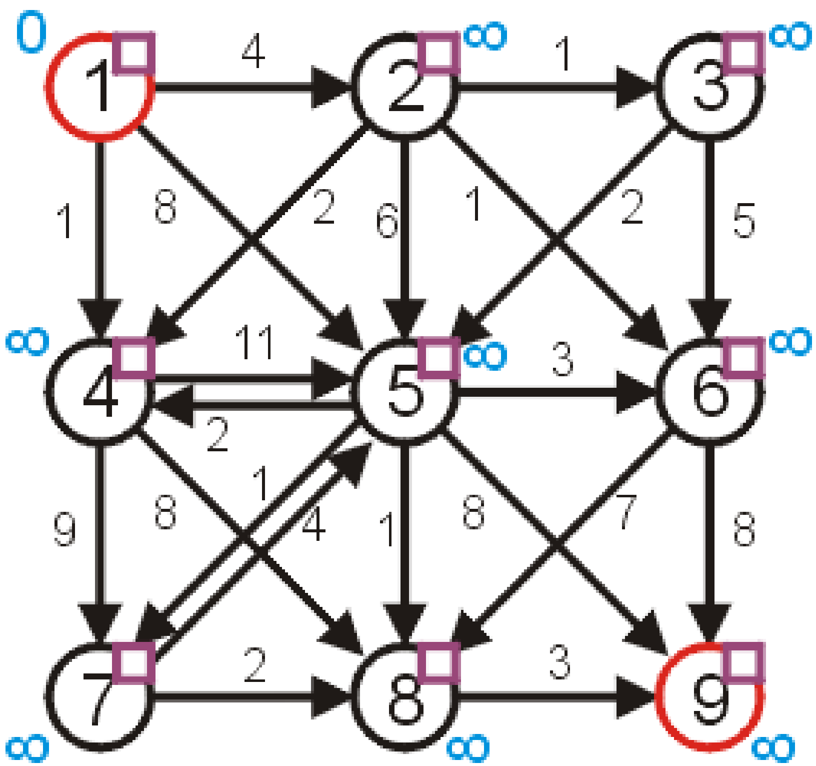
- Steps:
  - find an unvisited vertex which has the shortest distance to it
  - mark it as visited
  - for each unvisited vertex which is adjacent to the current vertex:
    - add the distance to the current vertex to the weight of the connecting edge
    - if this is less than the current distance to that vertex, update the distance and set the parent vertex of the adjacent vertex to be the current vertex

Dijkstra's Algorithm

- Halting condition:
  - successfully halt when the vertex visiting the target vertex
  - if at some point, all remaining unvisited vertices have distance ∞, then no path from the starting vertex to the end vertex exits

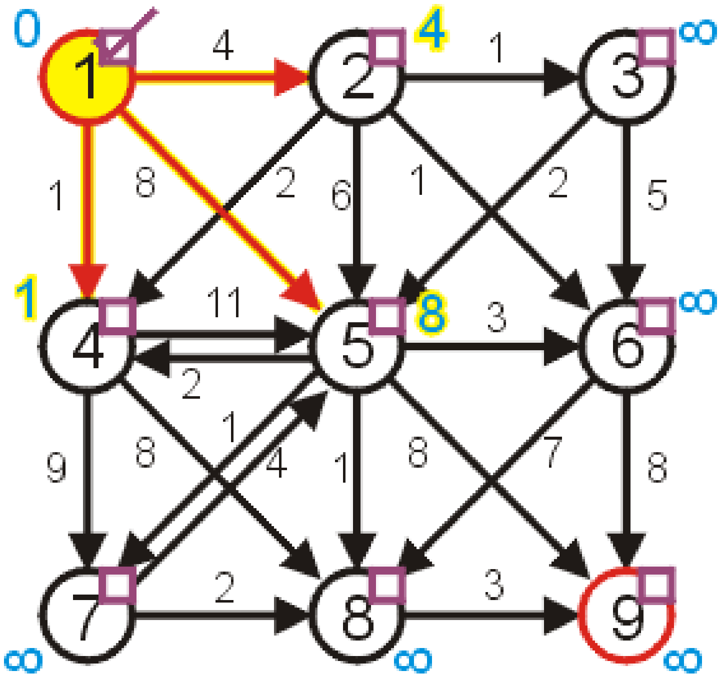
Example 1

- Consider the graph:
  - the distances are appropriately initialized
  - all vertices are marked as being unvisited



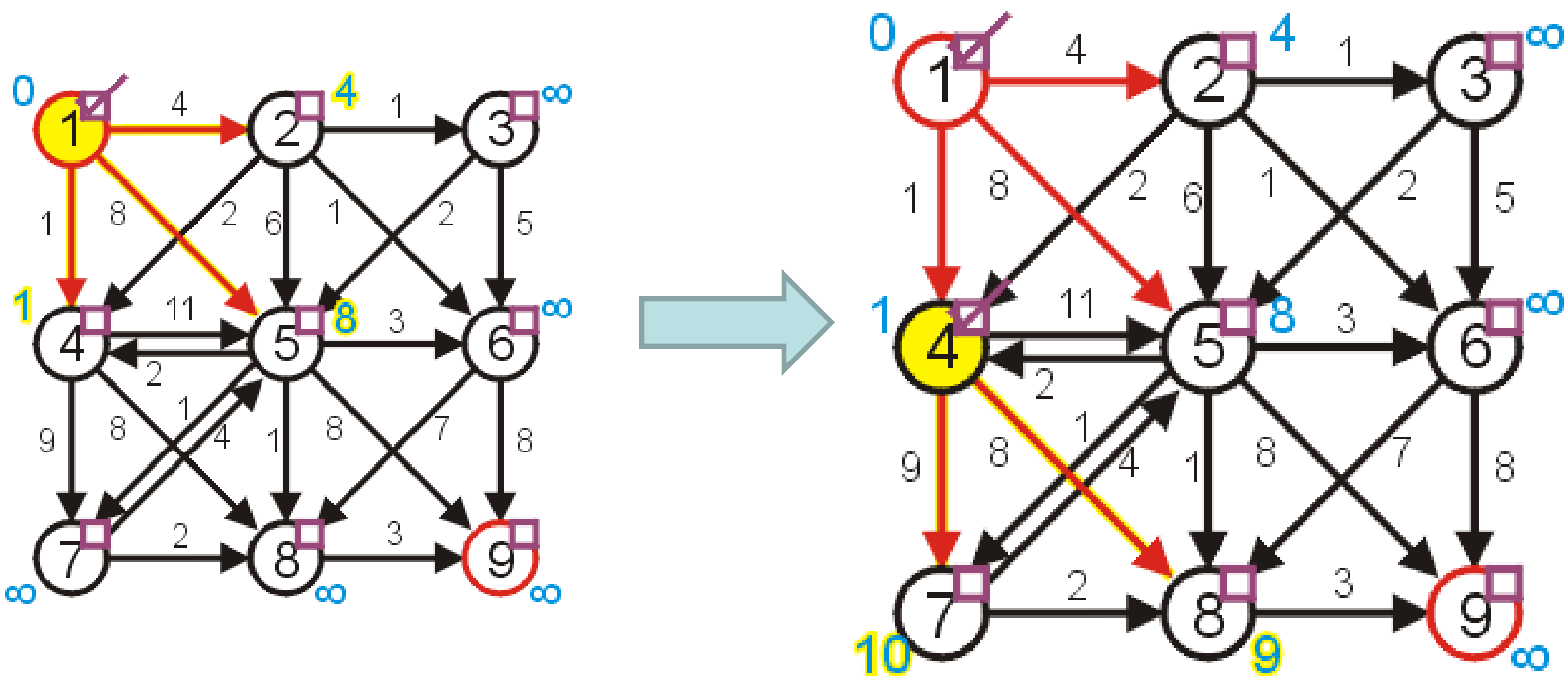
Example 1

- Visit vertex 1 and update its neighbours, marking it as visited
  - the shortest paths to 2, 4, and 5 are updated



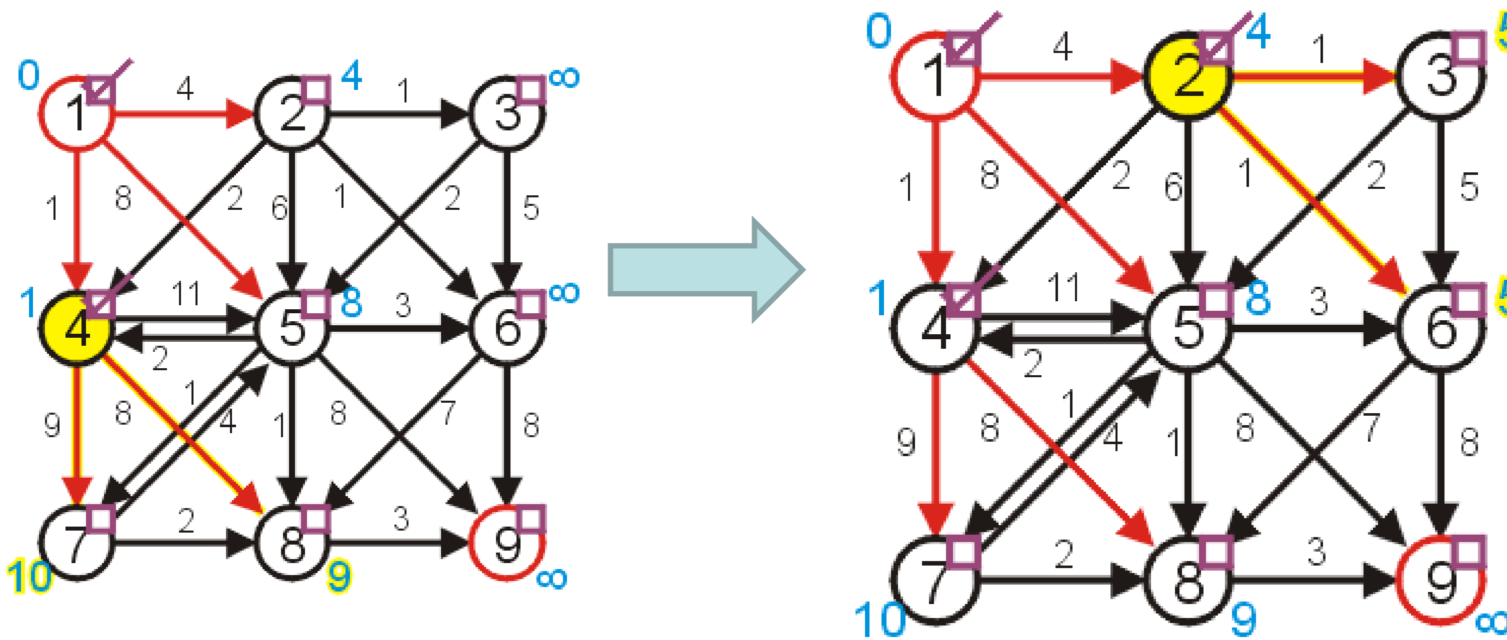
Example 1

- The next vertex to visit is 4
  - vertex 5 $1 + 11 \geq 8$ don't update
  - vertex 7 $1 + 9 < \infty$ update
  - vertex 8 $1 + 8 < \infty$ update



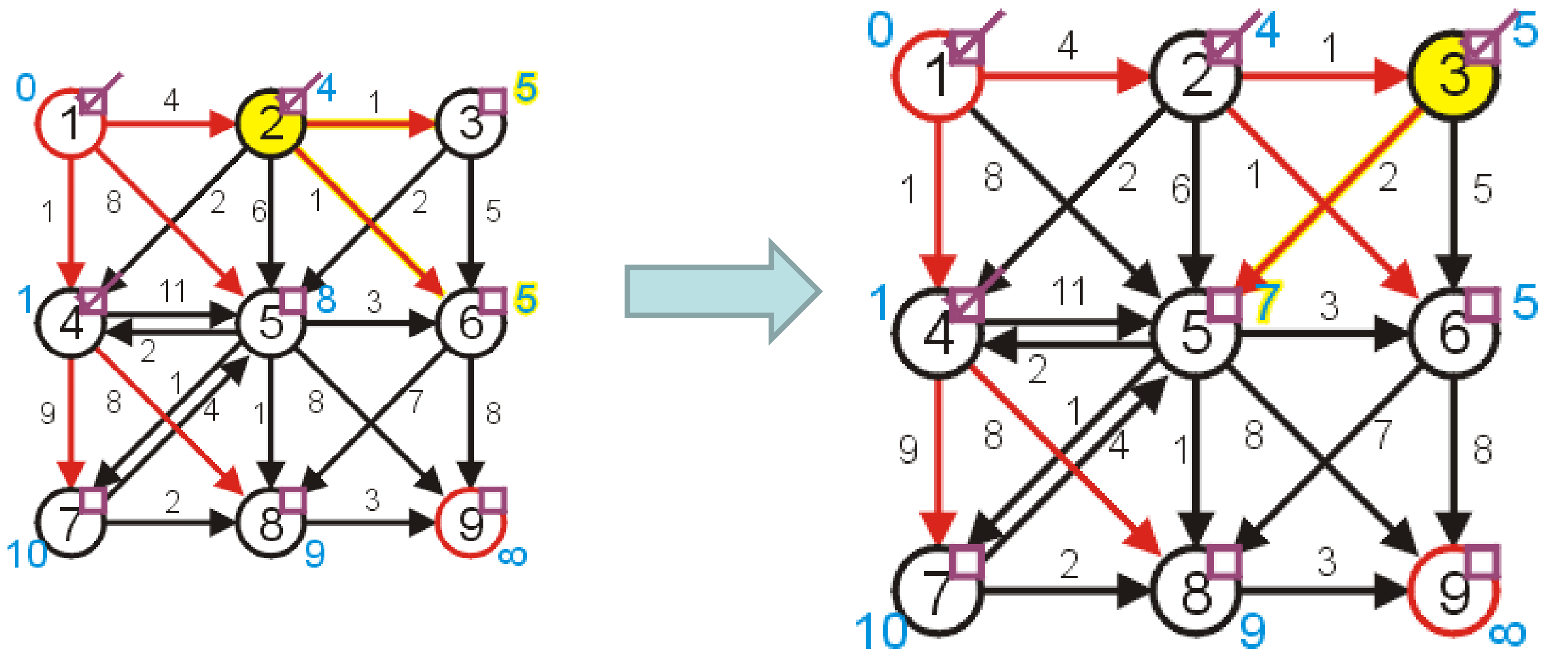
Example 1

- Next, visit vertex 2
  - vertex 3 $4 + 1 < \infty$ update
  - vertex 4 already visited
  - vertex 5 $4 + 6 \geq 8$ don't update
  - vertex 6 $4 + 1 < \infty$ update



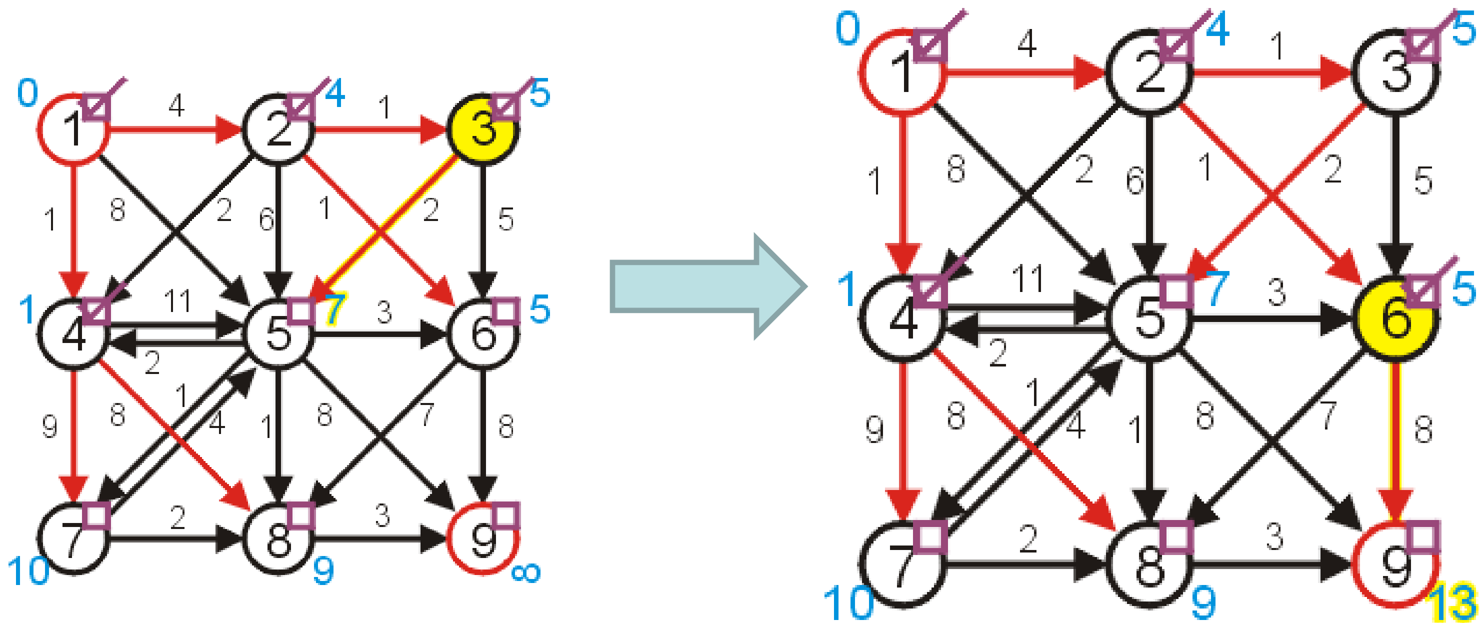
Example 1

- Next, a vertex visit choice of either 3 or 6
- If to visit 3
  - vertex 5 $5 + 2 < 8$ update
  - vertex 6 $5 + 5 \geq 5$ don't update



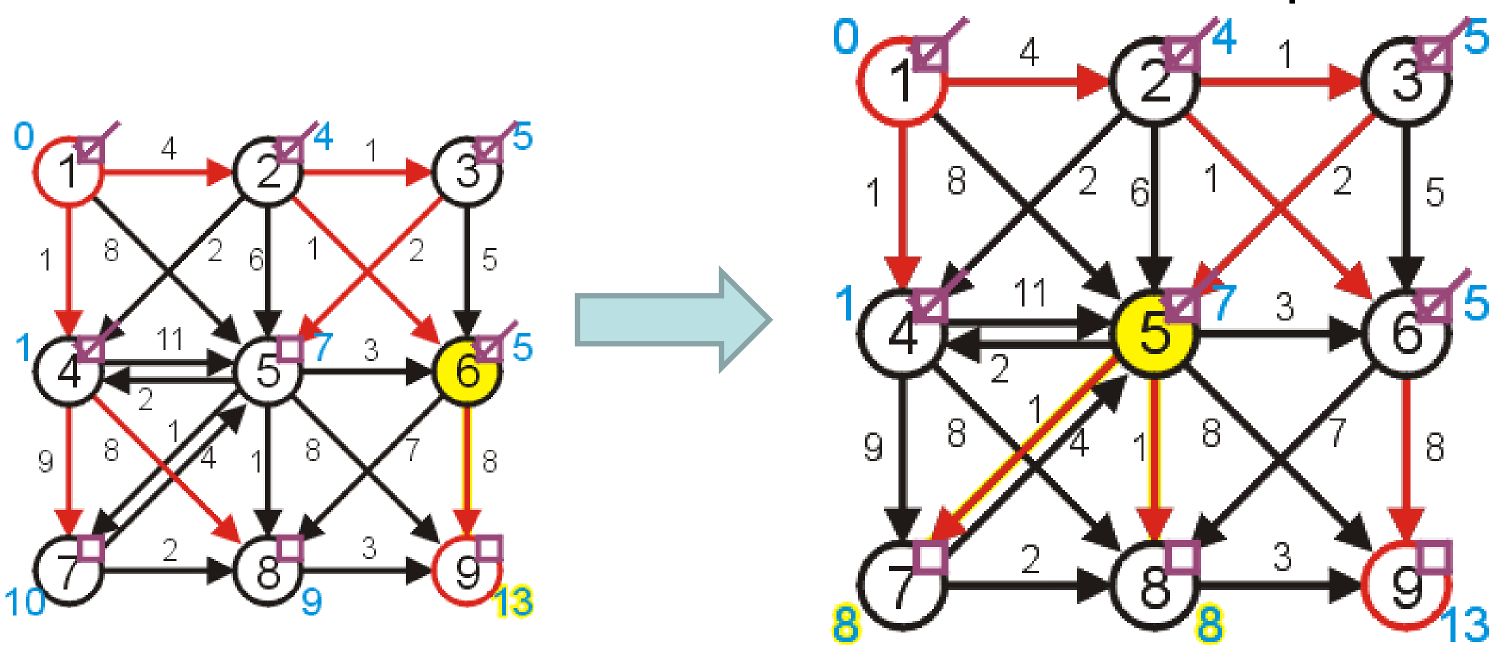
Example 1

- visit 6
  - vertex 8 $5 + 7 \geq 9$ don't update
  - vertex 9 $5 + 8 < \infty$ update



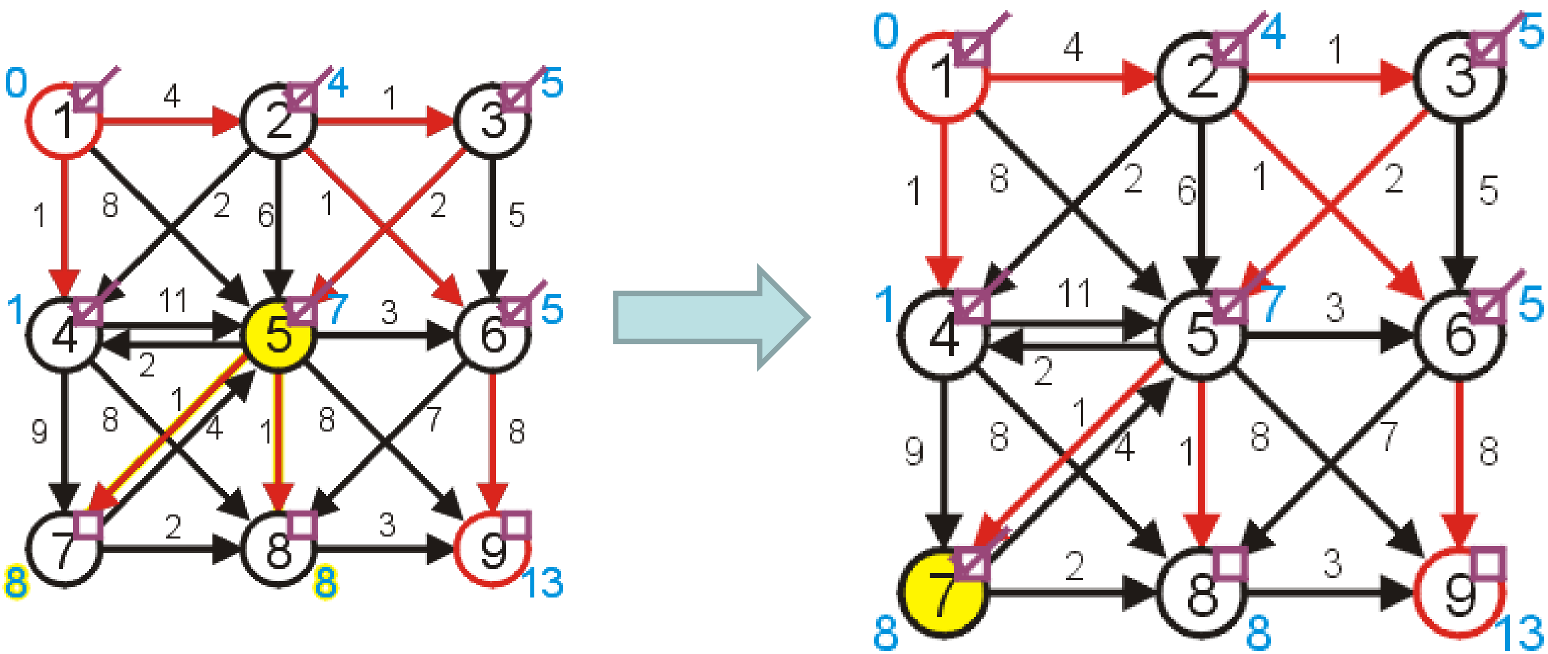
Example 1

- visit vertex 5:
  – vertices 4 and 6 have already been visited
  – vertex 7 $7 + 1 < 10$ update
  – vertex 8 $7 + 1 < 9$ update
  – vertex 9 $7 + 8 \geq 13$ don’t update



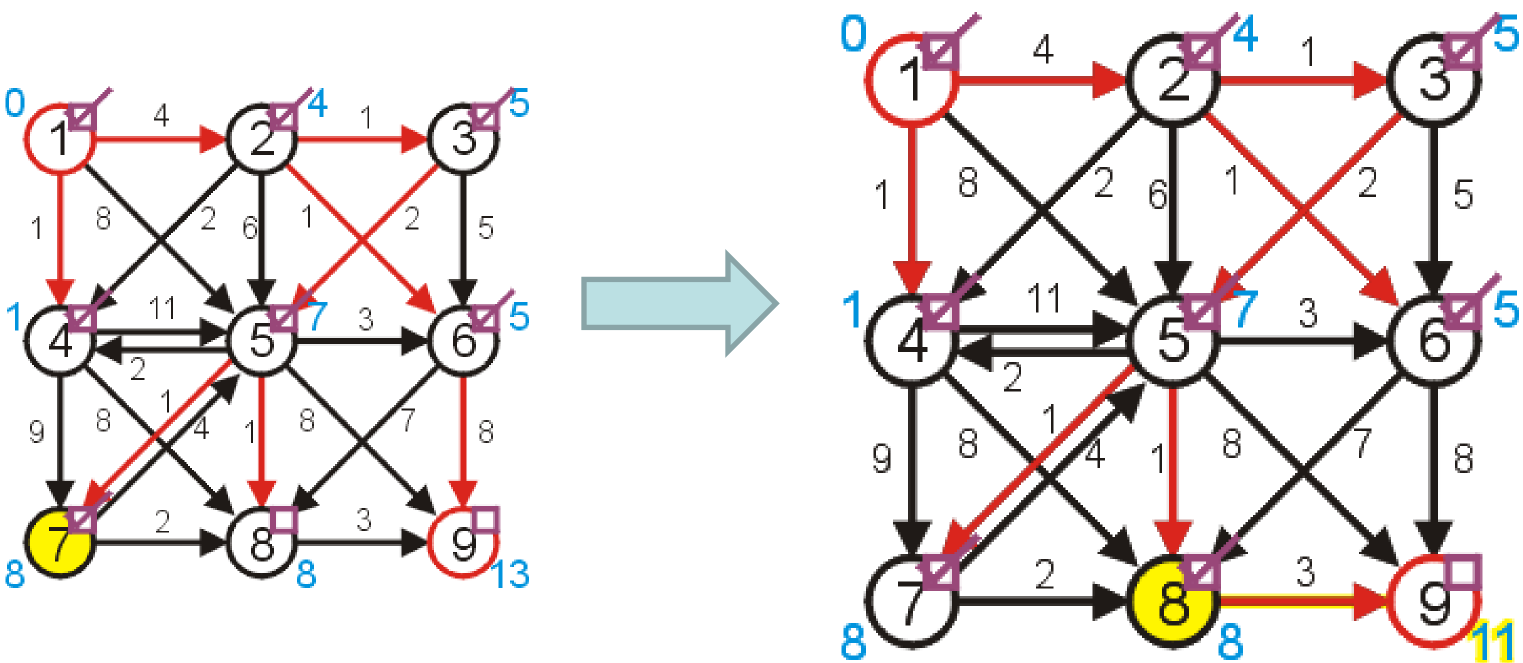
Example 1

- Given a choice between vertices 7 and 8; choose vertex 7
  – vertices 5 has already been visited
  – vertex 8 $8 + 2 \geq 8$ don’t update



Example 1

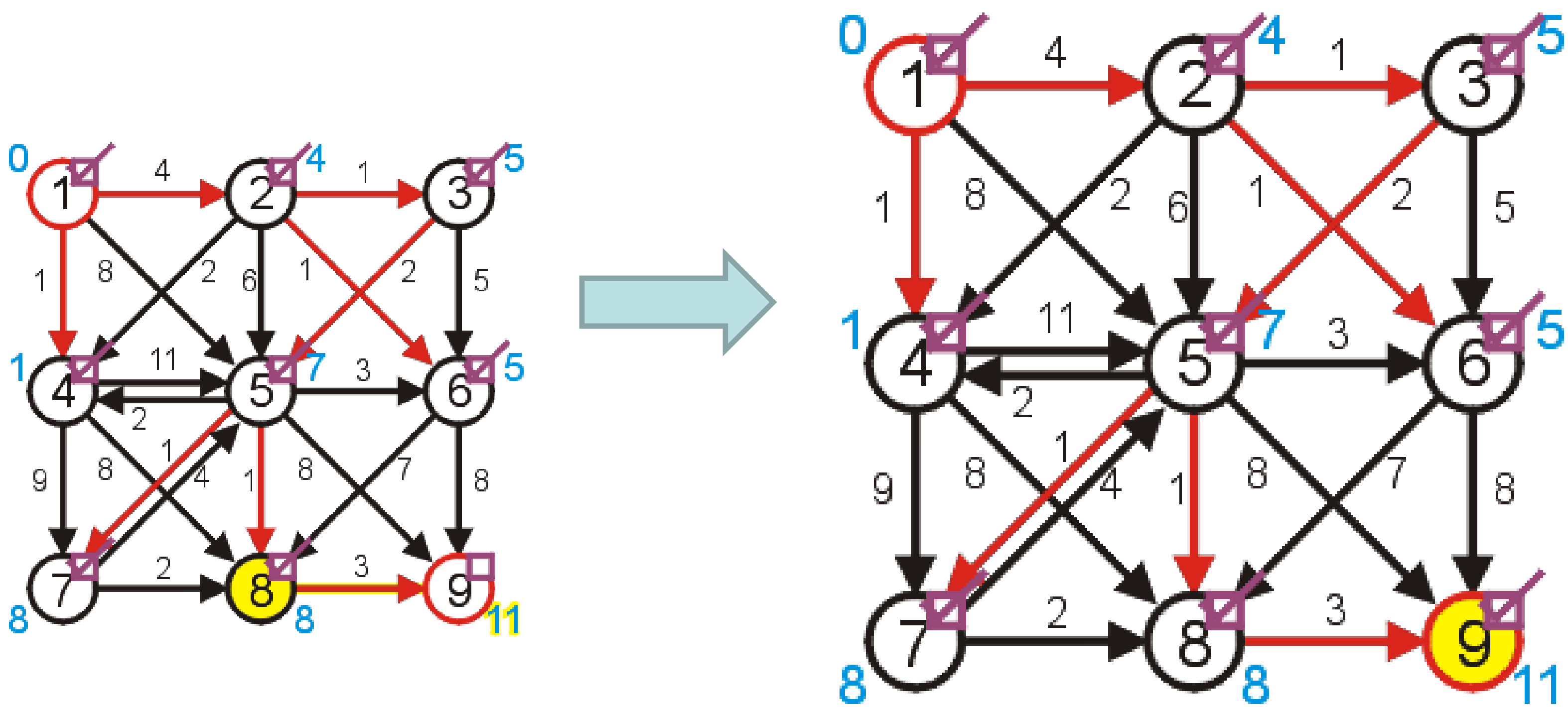
- Next, visit vertex 8:
  - vertex 9   8 + 3 < 13   update



Example 1

- Finally, visit the end vertex

→ shortest path from 1 to 9 has length 11

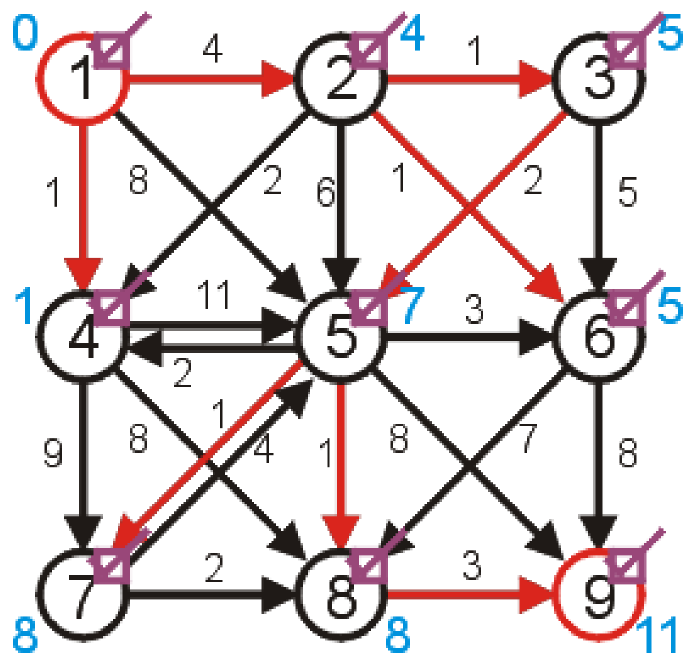


Example 1

- can find the shortest path by working back from the final vertex:

  → 9, 8, 5, 3, 2, 1

- Thus, the shortest path is (1, 2, 3, 5, 8, 9)



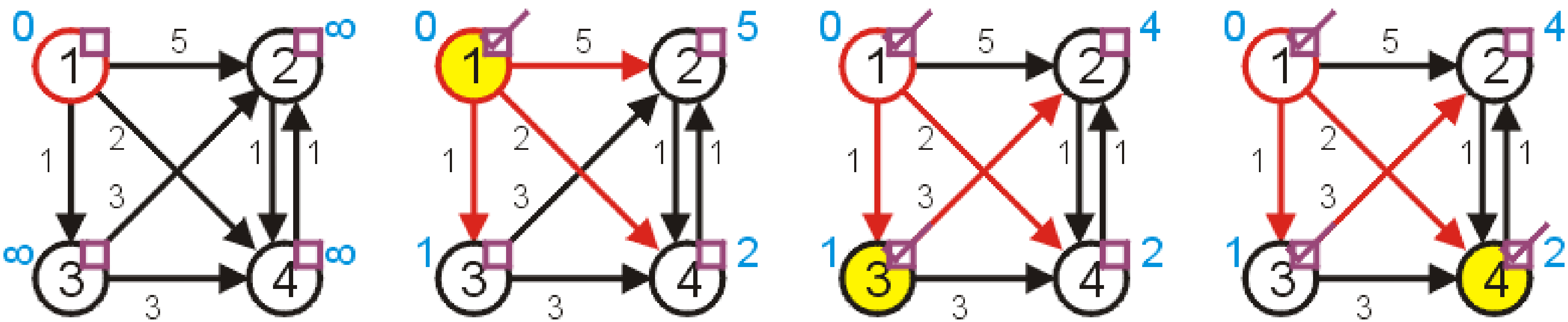
Note 1

- In the example, visited all vertices in the graph during the process of finding the shortest path
- However, such case may not always be

considered…

Example 2

- Find the shortest path from 1 to 4:
  - the shortest path is found by visiting only three vertices
  - algorithm terminates as soon as reaches to vertex 4
  - have useful information about 1, 3, 4
  - don't have the shortest path to vertex 2



Dijkstra's algorithm

$d[s] \leftarrow 0$
for each $v \in V - \{s\}$
  do $d[v] \leftarrow \infty$
$S \leftarrow \varnothing$
$Q \leftarrow V$ ⊳ Q is a priority queue maintaining $V - S$
while $Q \neq \varnothing$
  do $u \leftarrow$ EXTRACT-MIN(Q)
    $S \leftarrow S \cup \{u\}$
    for each $v \in Adj[u]$
      do if $d[v] > d[u] + w(u, v)$
        then $d[v] \leftarrow d[u] + w(u, v)$
        $p[v] \leftarrow u$

Dijkstra's Algorithm

$d[s] \leftarrow 0$
for each $v \in V - \{s\}$
    do $d[v] \leftarrow \infty$
$S \leftarrow \varnothing$
$Q \leftarrow V$ ⊳ Q is a priority queue maintaining $V - S$
while $Q \neq \varnothing$
    do $u \leftarrow$ EXTRACT-MIN(Q)
        $S \leftarrow S \cup \{u\}$
        for each $v \in Adj[u]$
            do if $d[v] > d[u] + w(u, v)$
                then $d[v] \leftarrow d[u] + w(u, v)$
                $p[v] \leftarrow u$
        ………

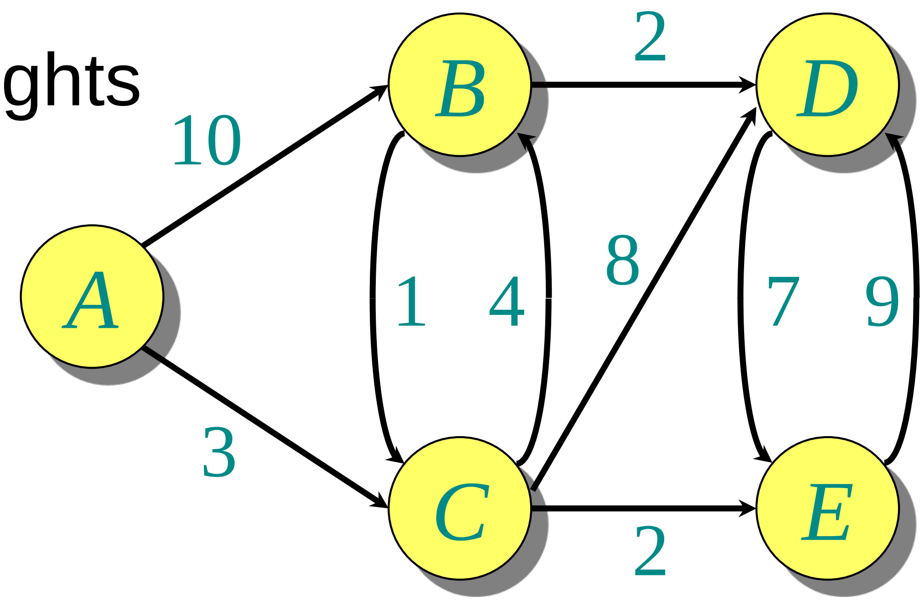
relaxation step

Implicit DECREASE-KEY

Example 3
Dijkstra's Algorithm

Example of Dijkstra's algorithm

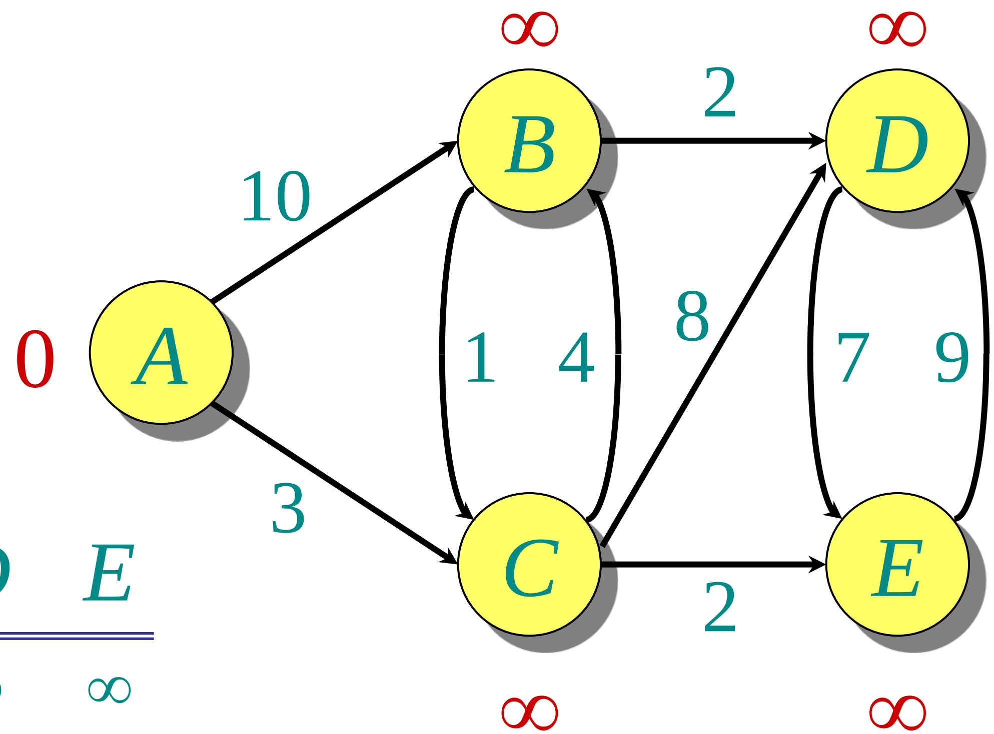
- Graph with nonnegative edge weights



Q. Find shortest distance to all other vertices from Source A

Example of Dijkstra's algorithm

Initialize:

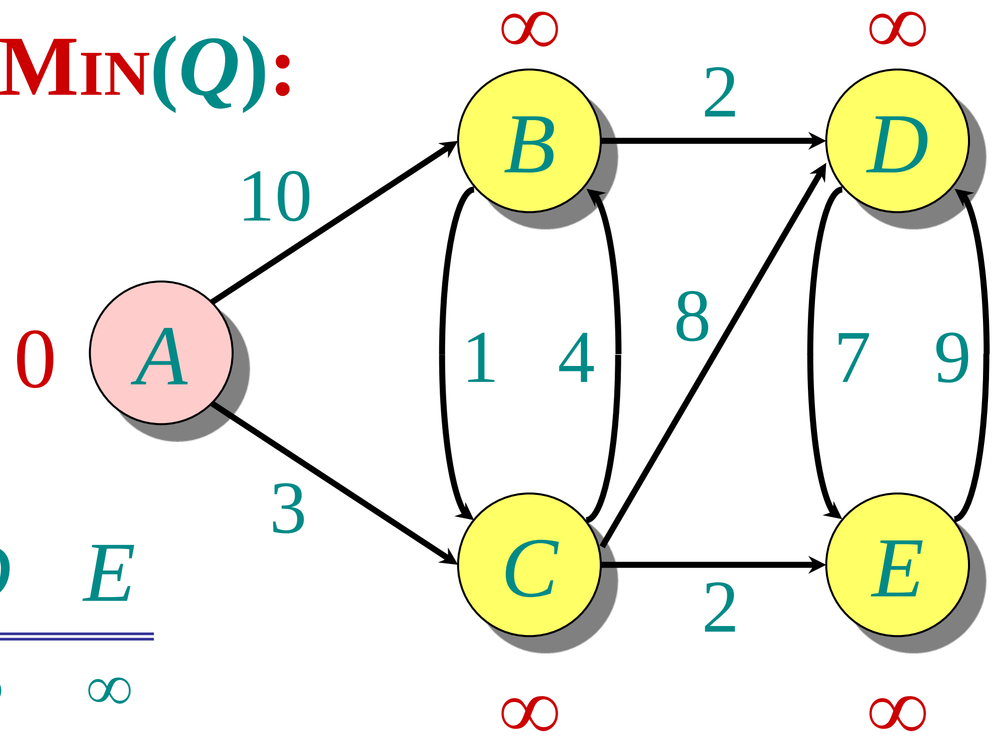


Q:	A	B	C	D	E
	0	∞	∞	∞	∞

S: { }

Example of Dijkstra's algorithm

"A" ← **EXTRACT-MIN(Q):**

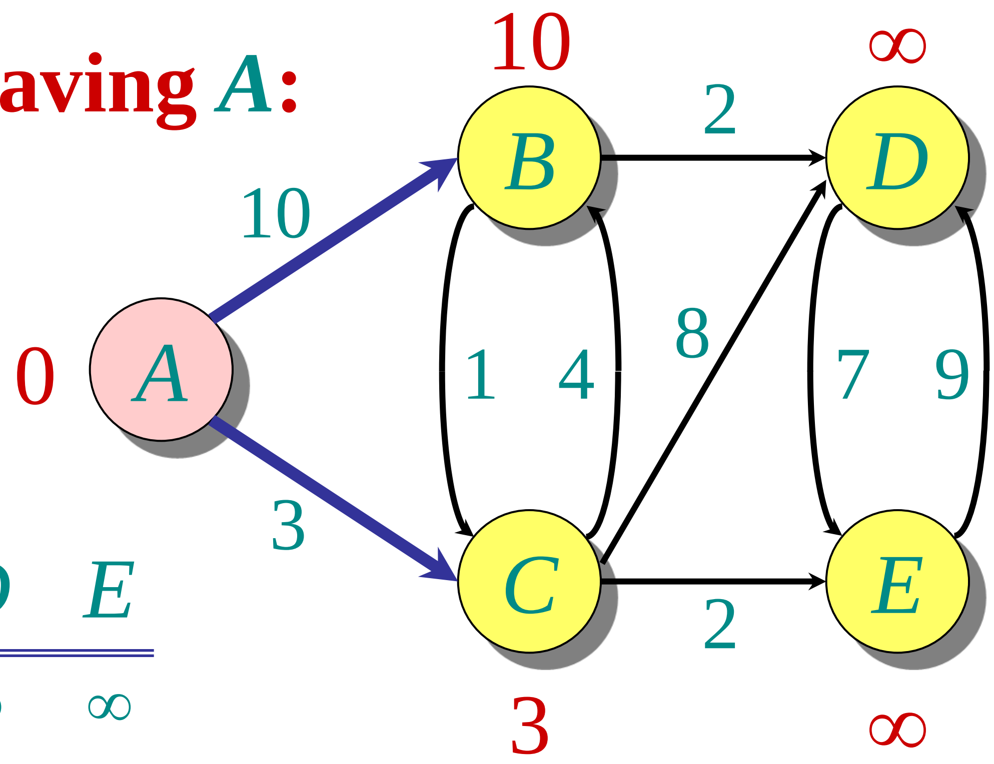


Q:	A	B	C	D	E
	0	∞	∞	∞	∞

S: { A }

Example of Dijkstra's algorithm

Relax all edges leaving *A*:

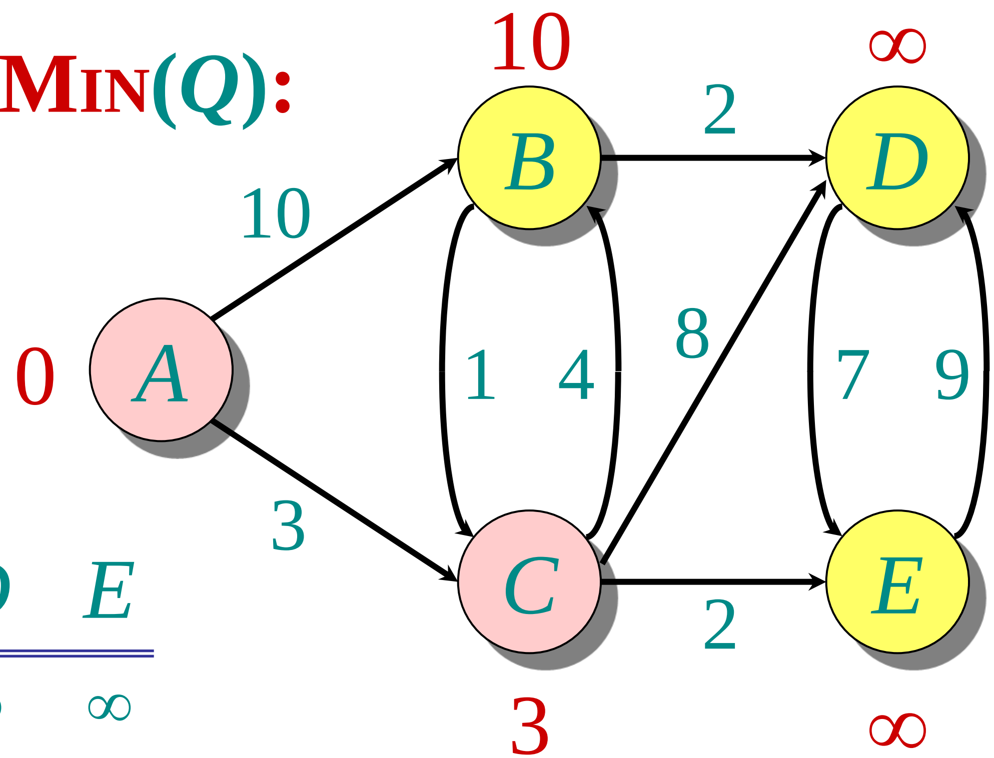


Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞

S: { *A* }

Example of Dijkstra's algorithm

"C" ← EXTRACT-MIN(Q):

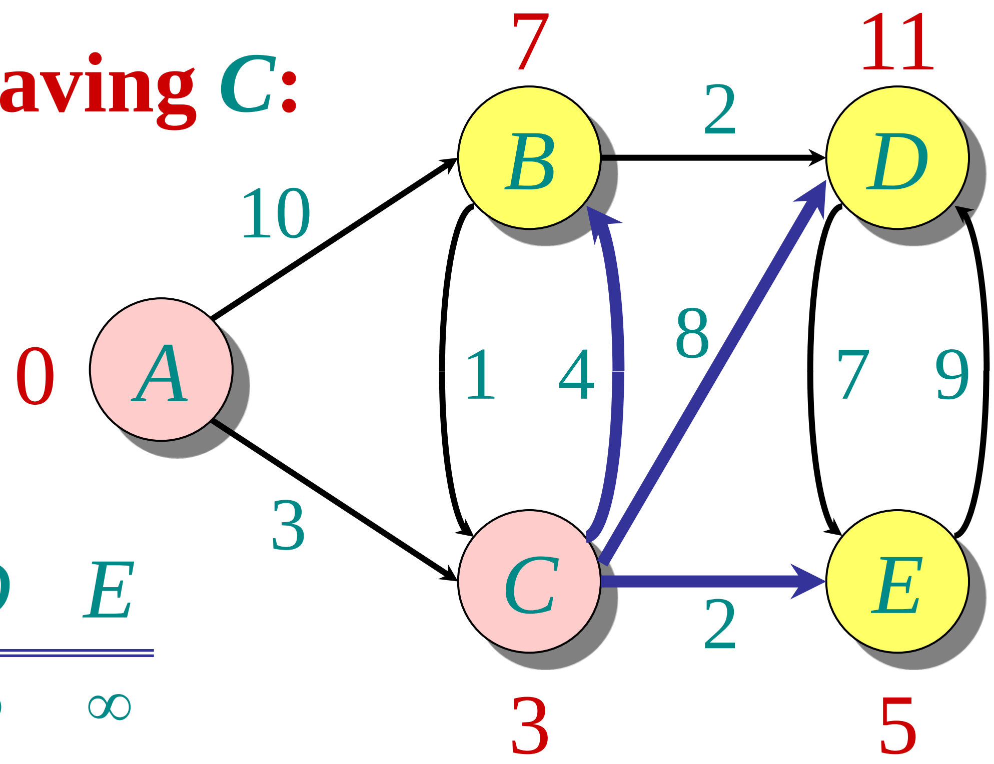


Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞

S: { A, C }

Example of Dijkstra's algorithm

Relax all edges leaving *C*:

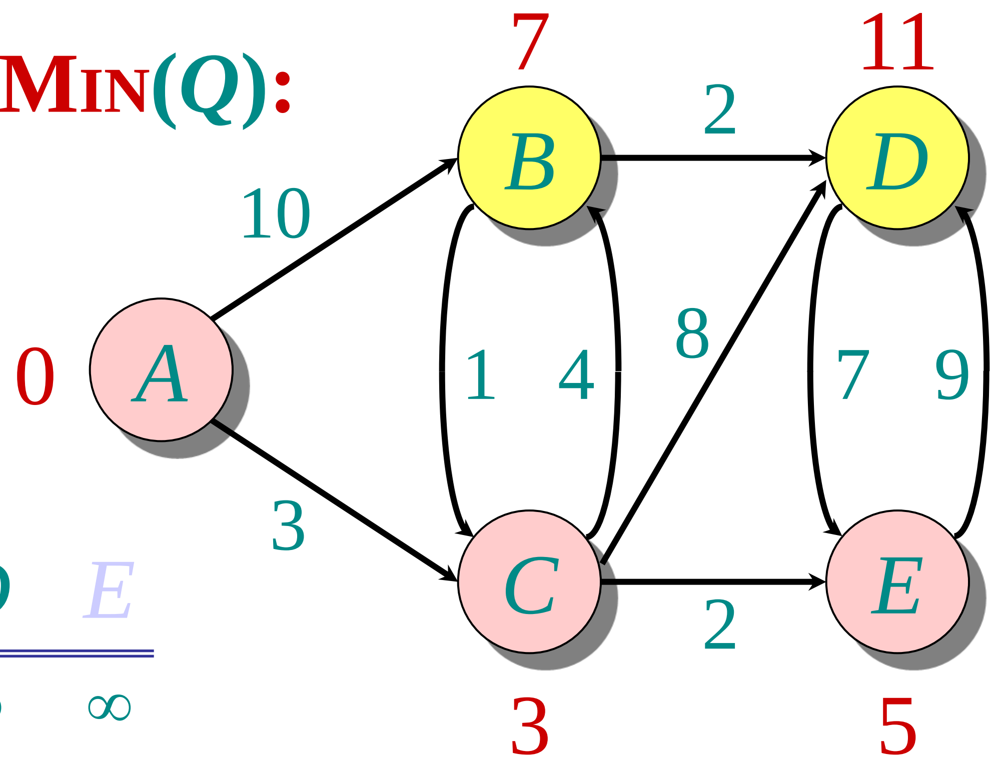


Q:	*A*	*B*	*C*	*D*	*E*
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5

S: { *A*, *C* }

Example of Dijkstra's algorithm

"E" ← **EXTRACT-MIN(*Q*):**

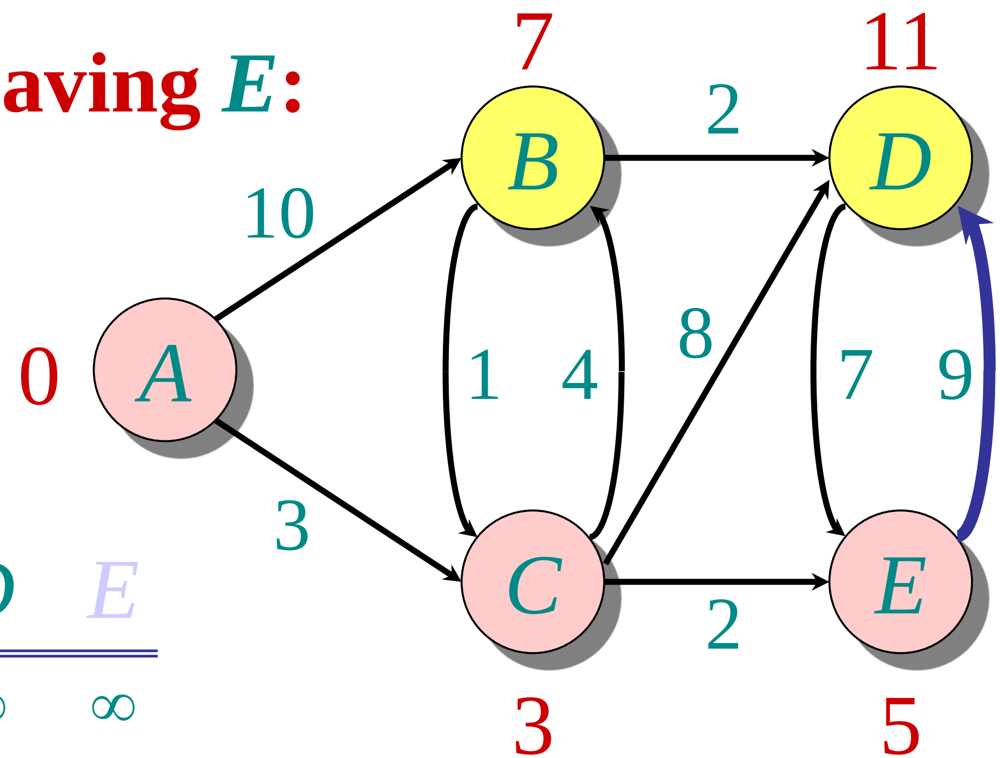


Q:	*A*	*B*	*C*	*D*	*E*
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5

S: { *A*, *C*, *E* }

Example of Dijkstra's algorithm

Relax all edges leaving *E*:

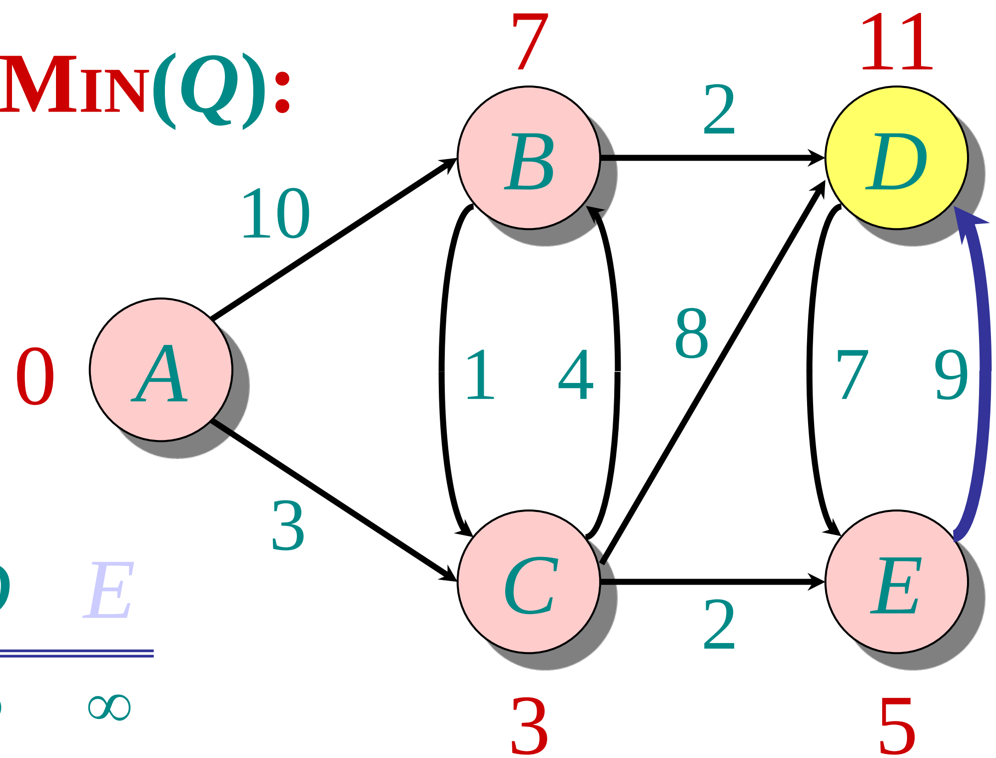


Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5
		7		11	

S: { A, C, E }

Example of Dijkstra's algorithm

"B" ← EXTRACT-MIN(*Q*):

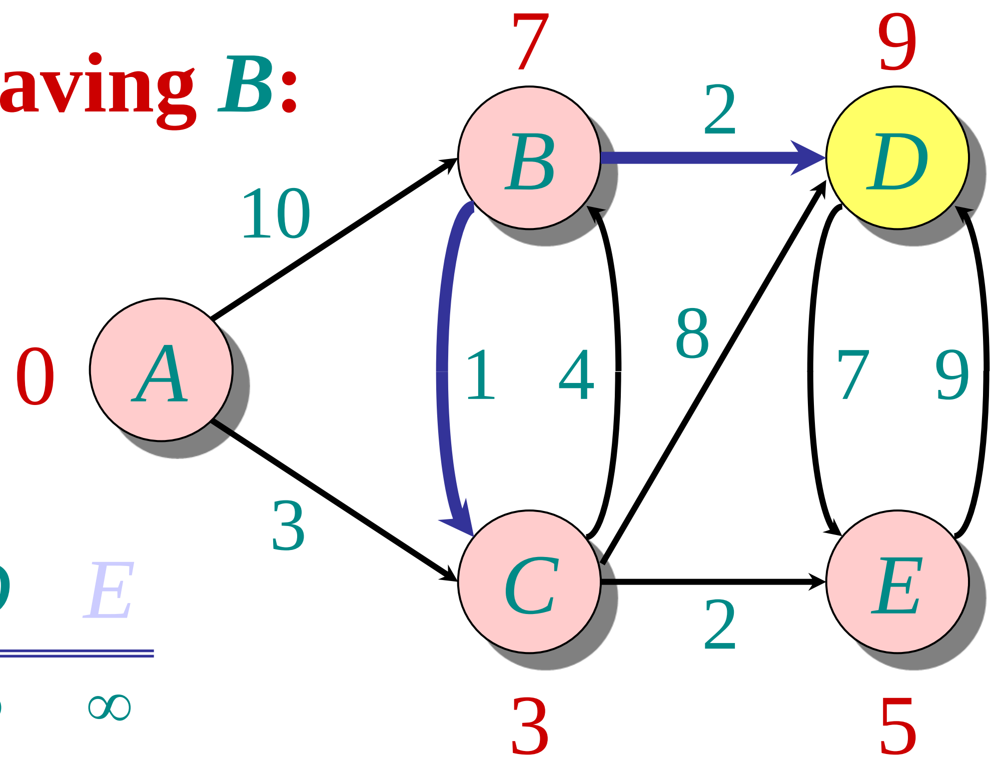


Q:	*A*	*B*	*C*	*D*	*E*
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5
		7		11	

S: { *A*, *C*, *E*, *B* }

Example of Dijkstra's algorithm

Relax all edges leaving *B*:

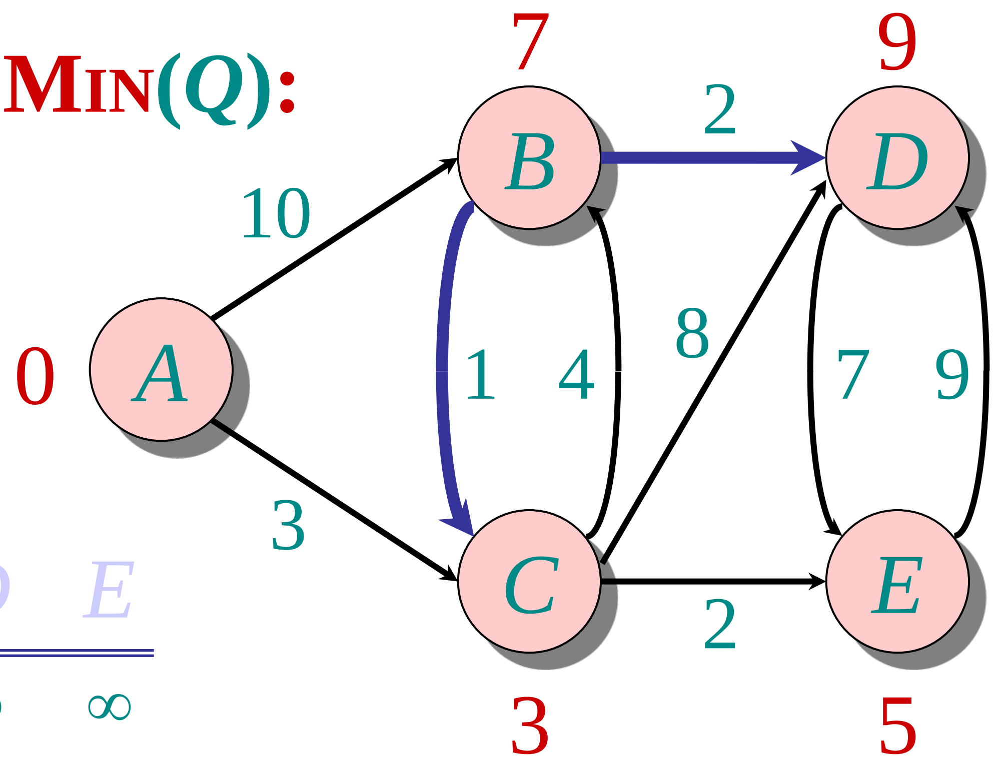


Q:	*A*	*B*	*C*	*D*	*E*
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5
		7		11	
				9	

S: { *A*, *C*, *E*, *B* }

Example of Dijkstra's algorithm

"D" ← **EXTRACT-MIN(*Q*):**



Q:	*A*	*B*	*C*	*D*	*E*
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5
		7		11	
				9	

S: { *A*, *C*, *E*, *B*, *D* }

Note 2

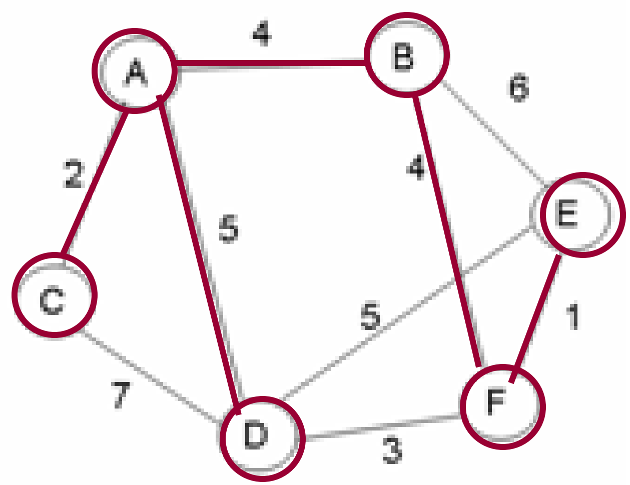
- Given a weighted directed graph, we can find the shortest distance between two vertices by:
  - starting with a trivial path containing the initial vertex
  - growing this path by always going to the next vertex which has the shortest current path

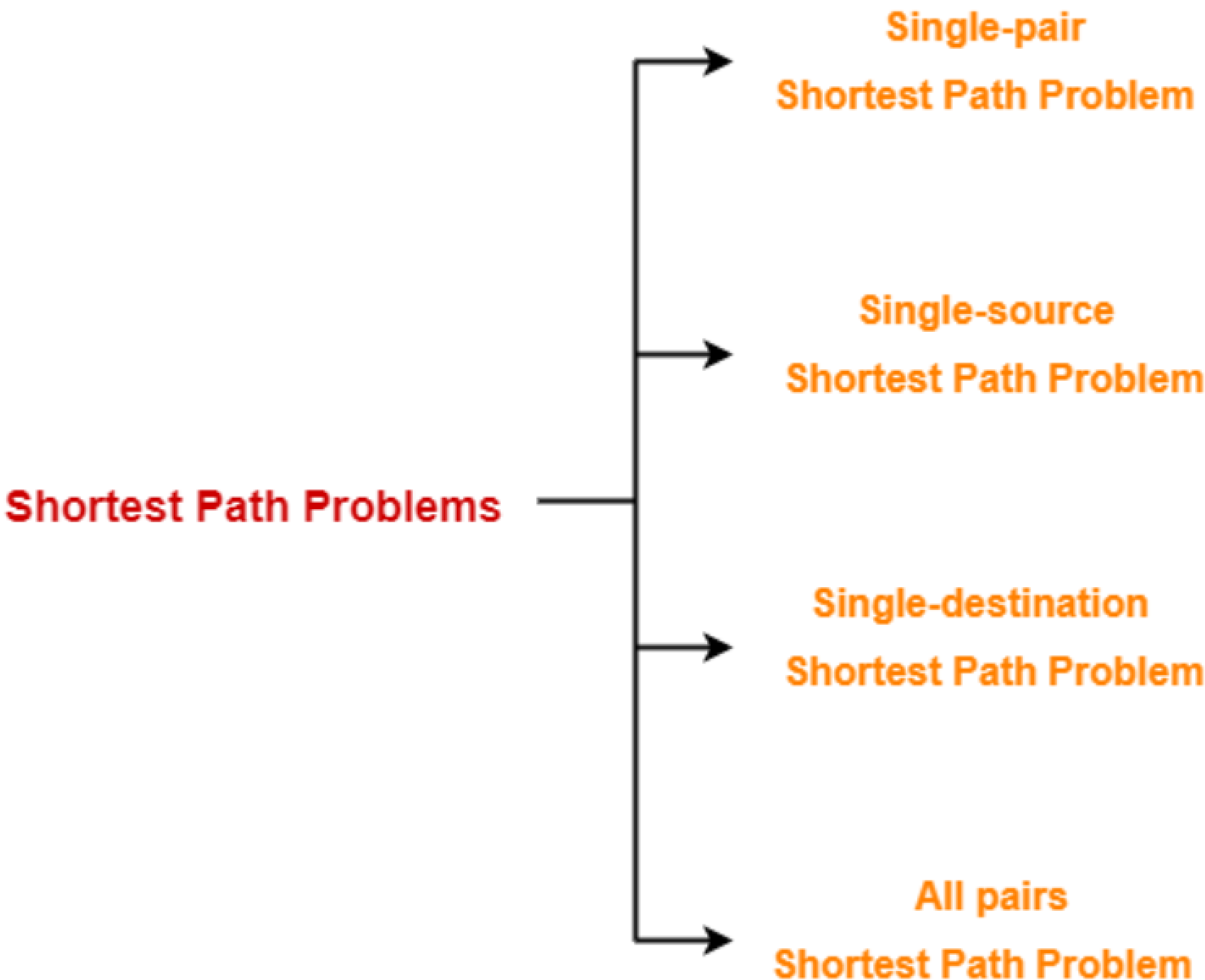
Practice

Example 4

Node	Included	Distance	Path
A	t	-	-
B	~~f~~ t	4	A
C	~~f~~ t	2	A
D	~~f~~ t	5	A
E	~~f~~ t	~~∞~~ ~~10~~ 9	~~-~~ ~~B~~ F
F	~~f~~ t	~~∞~~ 8	~~-~~ B

- Give the <u>shortest path tree</u> for node A for this graph using Dijkstra’s shortest path algorithm.
- Show your work with 3 arrays given and draw the resultant shortest path tree with edge weights included.




Shortest Path Problems
Single-pair
Shortest Path Problem
Single-source
Shortest Path Problem
Single-destination
Shortest Path Problem
All pairs
Shortest Path Problem

shortest path problem is the problem of finding a path between two vertices (or nodes) in a graph such that the sum of the weights of its constituent edges is minimized.

Popular Shortest Path Algorithms :

- Dijkstra's algorithm solves the single-source shortest path problem with non-negative edge weight
- Bellman–Ford algorithm solves the single-source problem if edge weights may be negative
- A* search algorithm solves for single-pair shortest path using heuristics to try to speed up the search
- Floyd–Warshall algorithm solves all pairs shortest paths
- Johnson's algorithm solves all pairs shortest paths, and may be faster than Floyd–Warshall on sparse graphs
- Viterbi algorithm solves the shortest stochastic path problem with an additional probabilistic weight on each node

//Ref:https://en.wikipedia.org/wiki/Shortest_path_problem//

Dijkstra AlgorithmComplexity Analysis-

Case-01:

- graph G is represented as an adjacency matrix
- Priority queue Q is represented as an unordered list
- A[i,j] stores the information about edge (i,j)
- Time taken for selecting i with the smallest dist is O(V)
- For each neighbor of i, time taken for updating dist[j] is O(1) and there will be maximum V neighbors
- Time taken for each iteration of the loop is O(V) and one vertex is deleted from Q
- Thus**, total time complexity becomes $O(V^2)$**

Case-02:

- graph G is represented as an adjacency list
- Priority queue Q is represented as a binary heap
- With adjacency list representation, all vertices of the graph can be traversed using BFS in O(V+E) time
- In min heap, operations like extract-min and decrease-key value takes O(logV) time
- So, **overall time complexity** becomes O(E+V) x O(logV) which is

  O((E + V) x logV) = **O(ElogV)**
- This time complexity can be reduced to **O(E+VlogV)** using Fibonacci heap

Comparison of Dijkstra’s and Floyd–Warshall algorithms:

Main Purposes:

Dijkstra’s Algorithm is one example of a single-source shortest or SSSP algorithm, i.e., given a source vertex it finds shortest path from source to all other vertices.

Floyd Warshall Algorithm is an example of all-pairs shortest path algorithm, meaning it computes the shortest path between all pair of nodes.

- Time Complexity of Dijkstra’s Algorithm: O(E log V) //blind search//
- Time Complexity of Floyd Warshall: $O(V^3)$

Other Points:

- can use Dijskstra’s shortest path algorithm for finding all pair shortest paths by running it for every vertex. But time complexity of this would be O(VE Log V) which can go (V^3 Log V) in worst case
- Unlike Dijkstra’s algorithm, Floyd Warshall can be implemented in a distributed system, making it suitable for data structures such as Graph of Graphs (Used in Maps)
- Floyd Warshall works for negative edge but no negative cycle, whereas Dijkstra’s algorithm don’t work for negative edges
- Floyd Warshall Algorithm is best suited for dense graphs

Bellman-Ford Algorithm

```
BellmanFord()
   for each v ∈ V
      d[v] = ∞;
   d[s] = 0;
   for i=1 to |V|-1
      for each edge (u,v) ∈ E
         Relax(u,v, w(u,v));
   for each edge (u,v) ∈ E
      if (d[v] > d[u] + w(u,v))
         return "no solution";
```

Initialize d[] which will converge to shortest-path value

Relaxation: Make |V|-1 passes, relaxing each edge

Test for solution: have we converged yet? i.e, ∃ negative cycle?

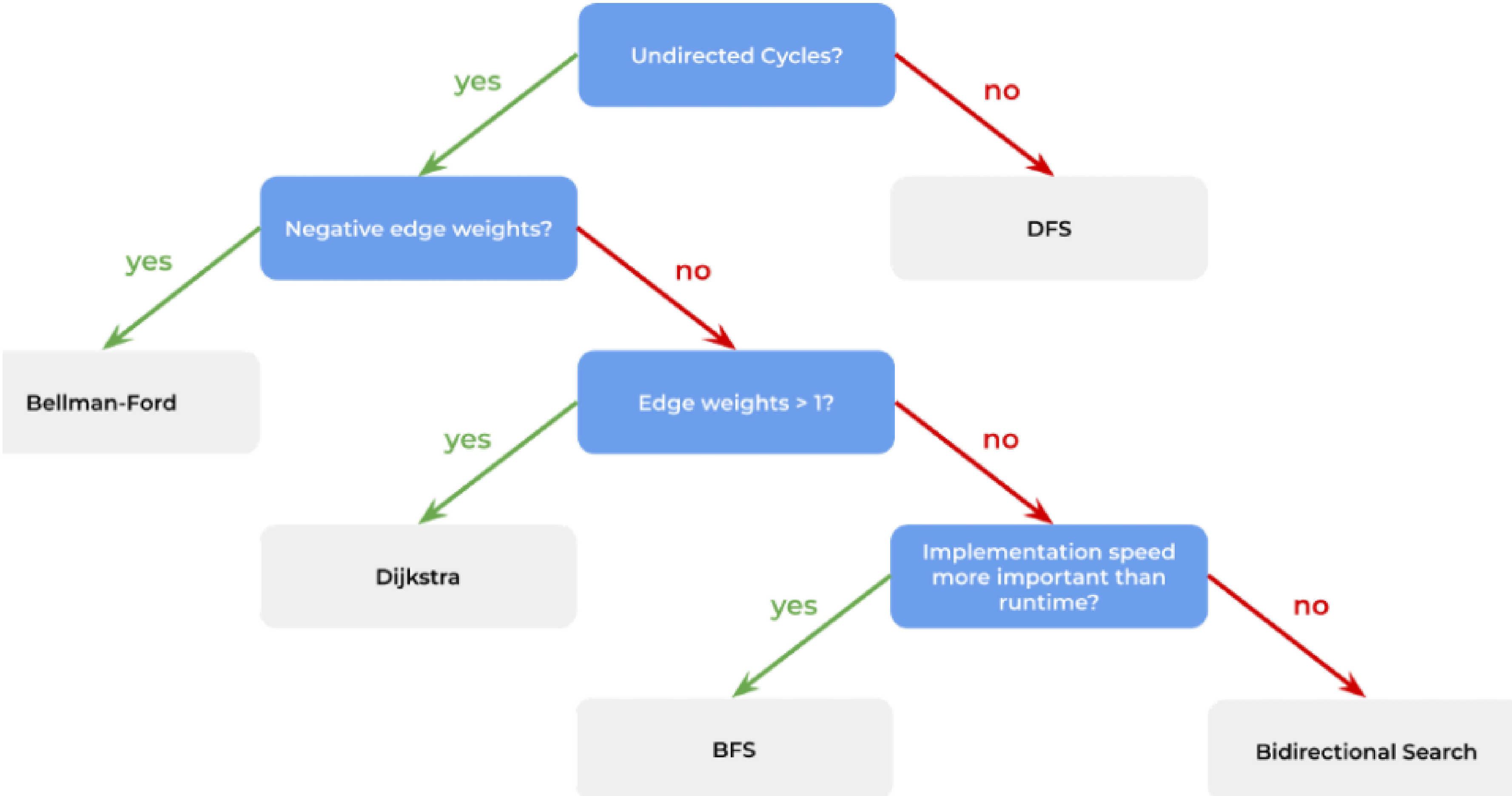
```
Relax(u,v,w): if (d[v] > d[u]+w) then d[v]=d[u]+w
```

Bellman Ford's Algorithm	Dijkstra's Algorithm
Bellman Ford's Algorithm works when there is negative weight edge, it also detects the negative weight cycle	Dijkstra's Algorithm doesn't work when there is negative weight edge
The result contains the vertices that also contains the information about the other vertices they are connected to	The result contains the vertices containing whole information about the network, not only the vertices they are connected to
It can easily be implemented in a distributed way	It can not be implemented easily in a distributed way
It is more time consuming than Dijkstra's algorithm. Its time complexity is O(VE)	It is less time consuming. The time complexity is O(E logV)
Dynamic Programming approach is taken to implement the algorithm	Greedy approach is taken to implement the algorithm

Shortest Path in a Graph

Algorithm	Negative Edge Weights	Positive Edge Weights > 1	Undirected Cycles	Runtime
DFS	✔	✔	✘	$O(n + e)$
BFS	✘	✘	✔	$O(n + e)$ or $O(g^d)$
Bidirectional Search	✘	✘	✔	$O(n + e)$ or $O(g^{(d/2)})$
Dijkstra	✘	✔	✔	$O(e + n \log(n))$
Bellman-Ford	✔	✔	✔	$O(n * e)$

n = number of nodes, e = number of edges, g = largest number of adjacent nodes for any node, d = length of the shortest path



Decision tree to determine the most appropriate shortest-path algorithm

Thank You…