Shortest Paths

CSE 2320 – Algorithms and Data Structures
Vassilis Athitsos
University of Texas at Arlington

Terminology

- A **network** is a **directed graph**. We will use both terms interchangeably.
- The **weight of a path** is the sum of weights of the edges that make up the path.
- The **shortest path** between two vertices s and t in a directed graph is a directed path from s to t with the property that no other such path has a lower weight.

Shortest Paths

- Finding shortest paths is not a single problem, but rather a family of problems.
- We will consider three of these problems, each of which is a generalization of the previous one:
  - Source-sink: find the shortest path from a source vertex v to a sink vertex w.
  - Single-source: find the shortest path from the source vertex v to all other vertices in the graph.
    - It turns out that these shortest paths form a tree, with v as the root.
  - All-pairs: find the shortest paths for all pairs of vertices in the graph.

Assumptions

- Allow directed graphs.
  - In all our algorithms, we will allow graphs to be directed.
  - Obviously, any algorithm that works on directed graphs will also work on undirected graphs. Why?

- Negative edge weights are not allowed. Why?

Assumptions

- Allow directed graphs.
  - In all our algorithms, we will allow graphs to be directed.
  - Obviously, any algorithm that works on directed graphs will also work on undirected graphs. Why?
    - Undirected graphs are a special case of directed graphs.
- Negative edge weights are not allowed. Why?
  - If we have negative weights, then "shortest paths" may not be defined.
  - If we can find a cyclic path with negative weight, then repeating that path infinitely will lead to "shorter" and "shorter" paths.
  - If all weights are nonnegative, a shortest path never needs to include a cycle.

Shortest-Paths Spanning Tree

- Given a network G and a designated vertex s, a **shortest-paths spanning tree** (SPST) for s is a tree that contains s and all vertices reachable from s, such that:
  - Vertex s is the root of this tree.
  - Each tree path is a shortest path in G.

Computing SPSTs

- To compute an SPST, given a graph G and a vertex s, we will design an algorithm that maintains and updates the following two arrays:
  - Array wt: wt[v] is the weight of the shortest path we have found so far from s to v.
    - At the beginning, wt[v] = infinity, except for s, where wt[s] = 0.
  - Array st: st[v] is the parent vertex of v on the shortest path found so far from s to v.
    - At the beginning, st[v] = -1, except for s, where st[s] = s.
  - Array in: in[v] is 1 if v has been already added to the SPST, 0 otherwise.
    - At the beginning, in[v] = 0, except for s, where in[s] = 1.

Dijkstra's Algorithm

- Computes an SPST for a graph G and a source s.
- Like Prim's algorithm, but:
  - First vertex to add is the source.
  - Works with directed graphs, whereas Prim's only works with undirected graphs.
  - Requires edge weights to be non-negative.
- Time: $O(V^2)$, similar analysis to that of Prim's algorithm.
- Time $O(E \lg V)$ using a priority-queue implementation.

Dijkstra's Algorithm

Input: number of vertices V, VxV array weight, source vertex s.

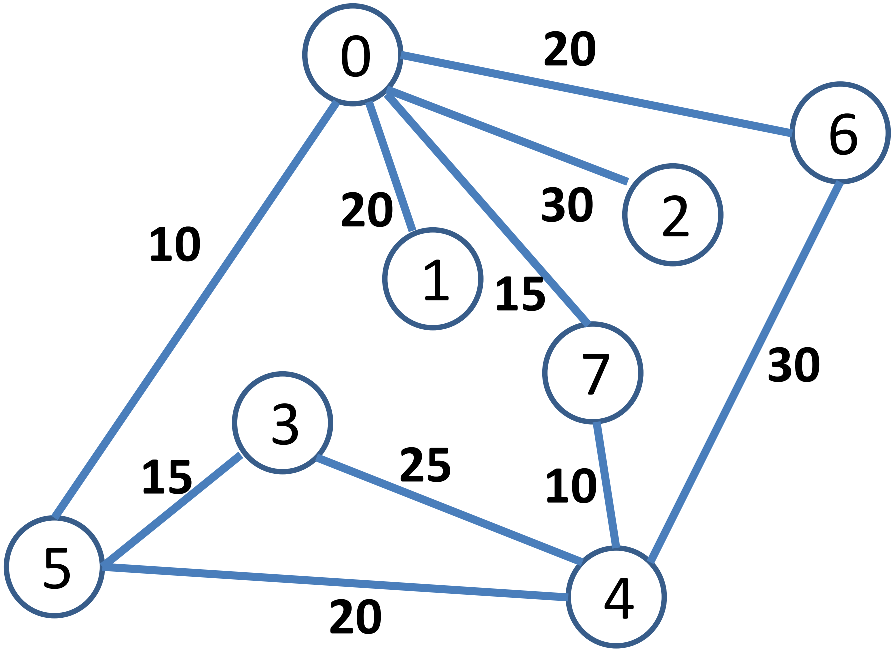
1. For all v:
    2. wt[v] = infinity.
    3. st[v] = -1.
    4. in[v] = 0.
5. wt[s] = 0, st[s] = 0.
6. Repeat until all vertices have been added to the tree:
    7. Find the v with the smallest wt[v], among all v such that in[v] = 0.
    8. Add to the SPST vertex v and edge from st[v] to v.
    9. in[v] = 1.
    10. For each neighbor w of v, such that in[w] = 0:
        11. If wt[w] > wt[v] + weight[v, w]:
            12. wt[w] = wt[v] + weight[v, w],
            13. st[w] = v.

Edge Relaxation

```
if (wt[w] > wt[v] + e.wt)
{
  wt[w] = wt[v] + e.wt;
  st[w] = v;
}
```

- wt[w]: current estimate of shortest distance from source to w.
- st[w]: parent vertex of w on shortest found path from source to w.

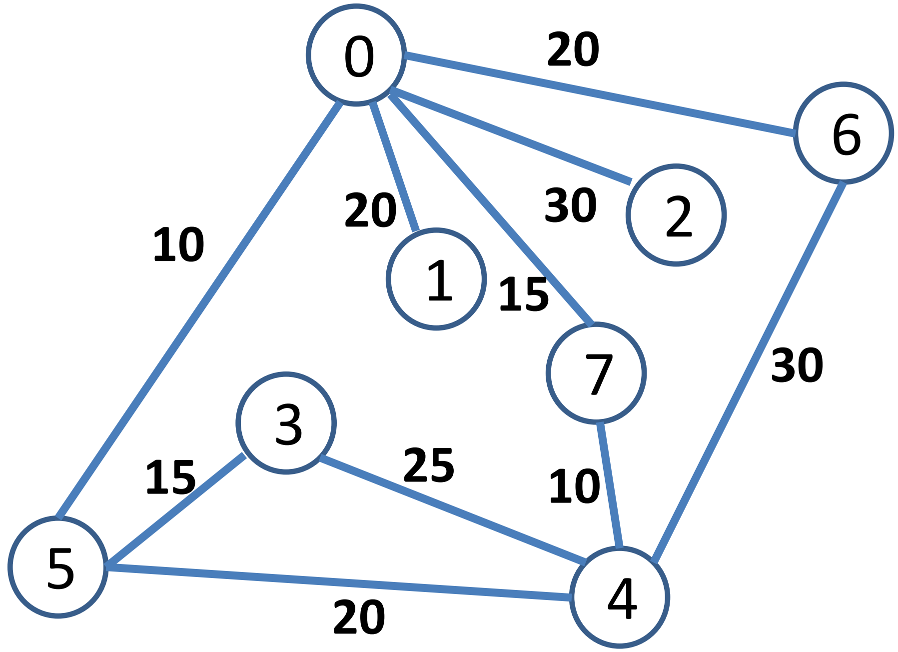
Dijkstra Example



- Suppose we want to compute the SPST for vertex 7.
- First, we initialize arrays wt, st, in (steps 2, 3, 4).

vertex	0	1	2	3	4	5	6	7
wt	inf	inf	inf	inf	inf	inf	inf	inf
st	-1	-1	-1	-1	-1	-1	-1	-1
in	0	0	0	0	0	0	0	0

Dijkstra Example

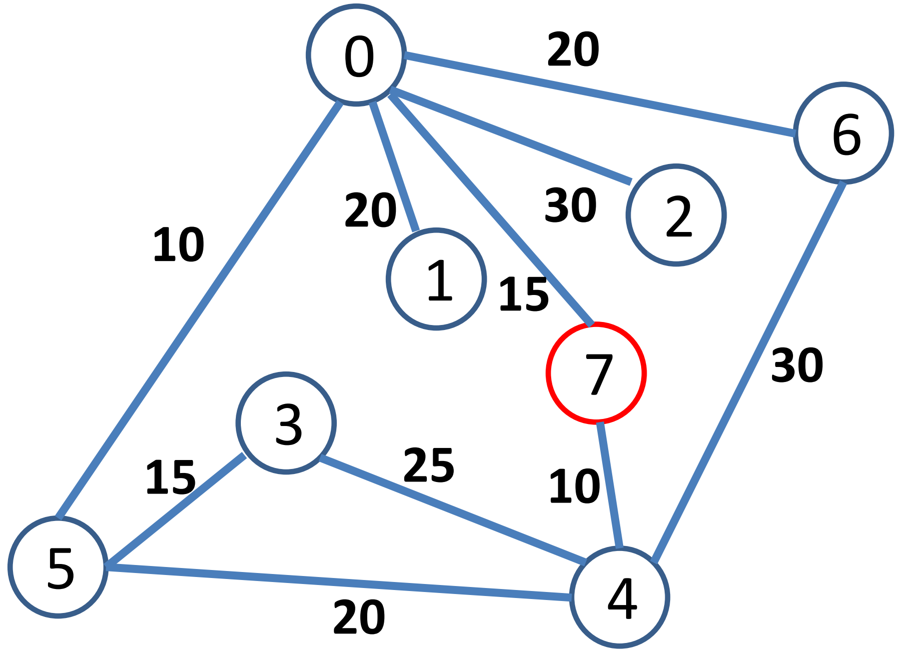


- Suppose we want to compute the SPST for vertex 7.
- Step 5.

vertex	0	1	2	3	4	5	6	7
wt	inf	inf	inf	inf	inf	inf	inf	0
st	-1	-1	-1	-1	-1	-1	-1	7
in	0	0	0	0	0	0	0	0

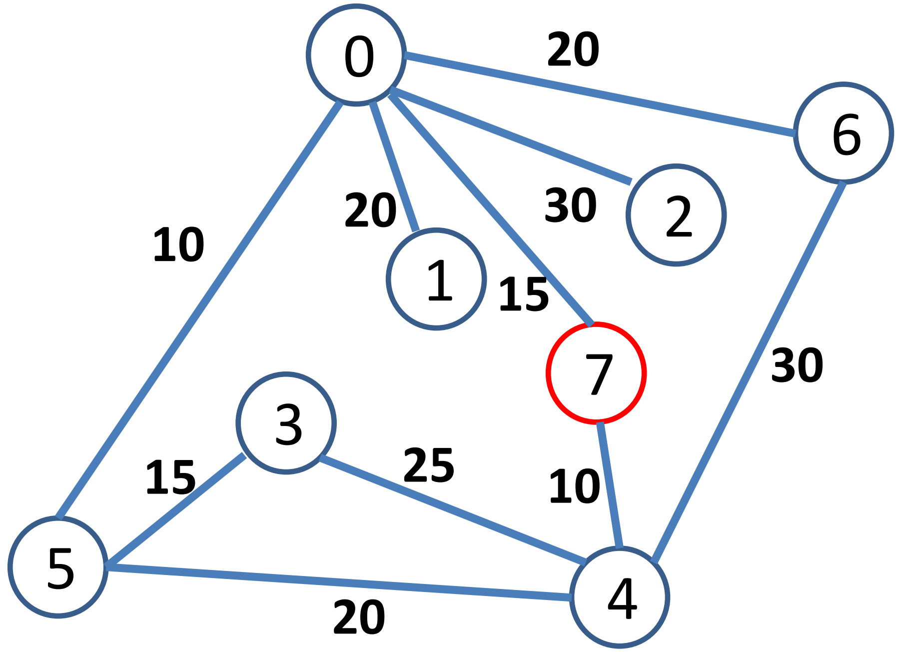
Dijkstra Example

- Suppose we want to compute the SPST for vertex 7.
- Steps 7, 8, 9: v = 7



vertex	0	1	2	3	4	5	6	7
wt	inf	inf	inf	inf	inf	inf	inf	0
st	-1	-1	-1	-1	-1	-1	-1	7
in	0	0	0	0	0	0	0	1

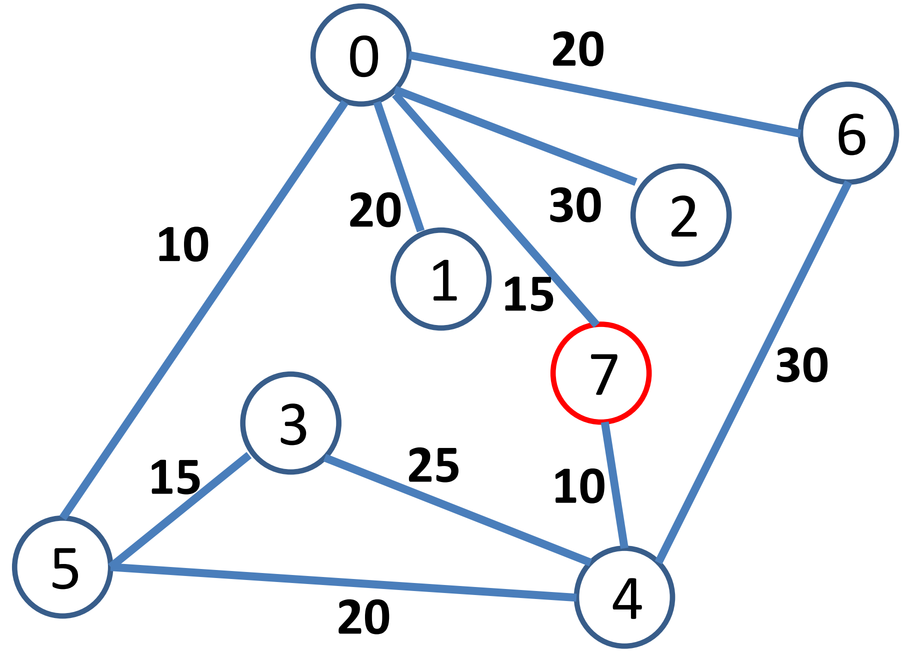
Dijkstra Example



- Suppose we want to compute the SPST for vertex 7.
- Steps 7, 8, 9: v = 7
- Step 10: For w = {0, 4}
  - Step 11: Compare inf with 15
  - Steps 12, 13: wt[0] = wt[7] + 15, st[0] = 7.

vertex	0	1	2	3	4	5	6	7
wt	15	inf	inf	inf	inf	inf	inf	0
st	7	-1	-1	-1	-1	-1	-1	7
in	0	0	0	0	0	0	0	1

Dijkstra Example

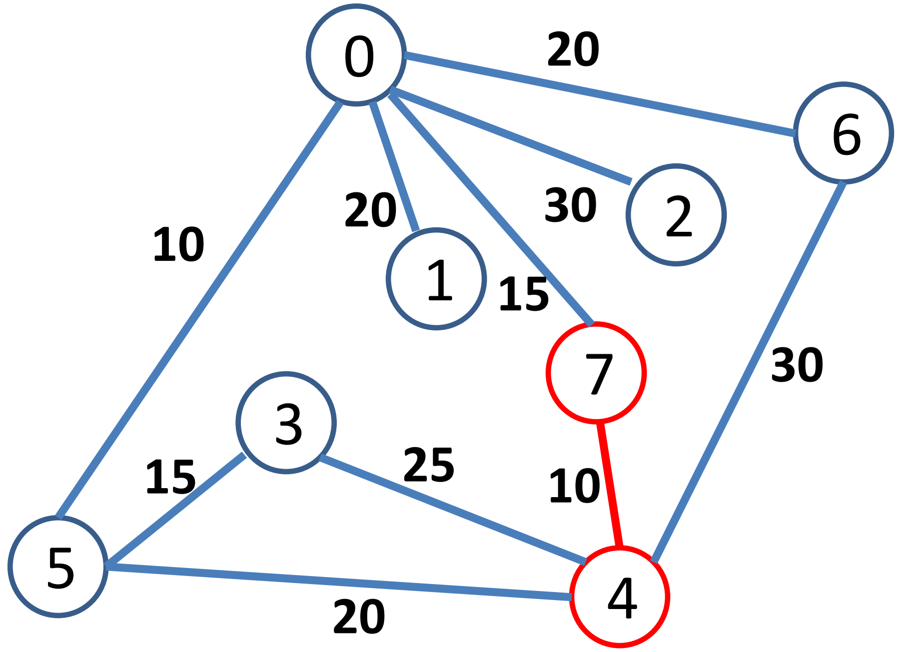


- Suppose we want to compute the SPST for vertex 7.
- Steps 7, 8, 9: v = 7
- Step 10: For w = {0, 4}
  - Step 11: Compare inf with 10
  - Steps 12, 13: wt[4] = wt[7] + 10, st[4] = 7.

vertex	0	1	2	3	4	5	6	7
wt	15	inf	inf	inf	10	inf	inf	0
st	7	-1	-1	-1	7	-1	-1	7
in	0	0	0	0	0	0	0	1

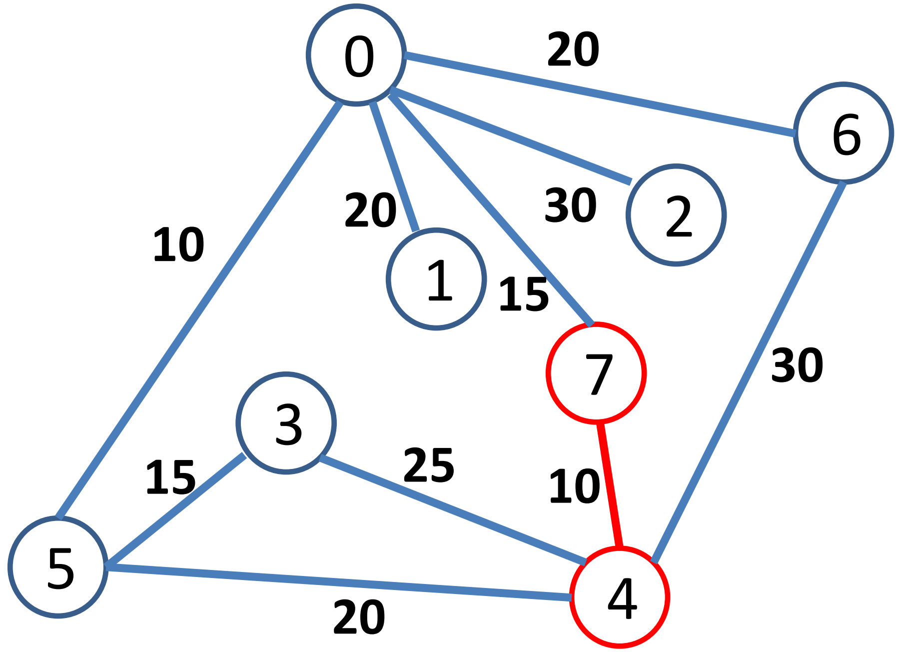
Dijkstra Example

- Suppose we want to compute the SPST for vertex 7.
- Steps 7, 8, 9: v = 4



vertex	0	1	2	3	4	5	6	7
wt	15	inf	inf	inf	10	inf	inf	0
st	7	-1	-1	-1	7	-1	-1	7
in	0	0	0	0	1	0	0	1

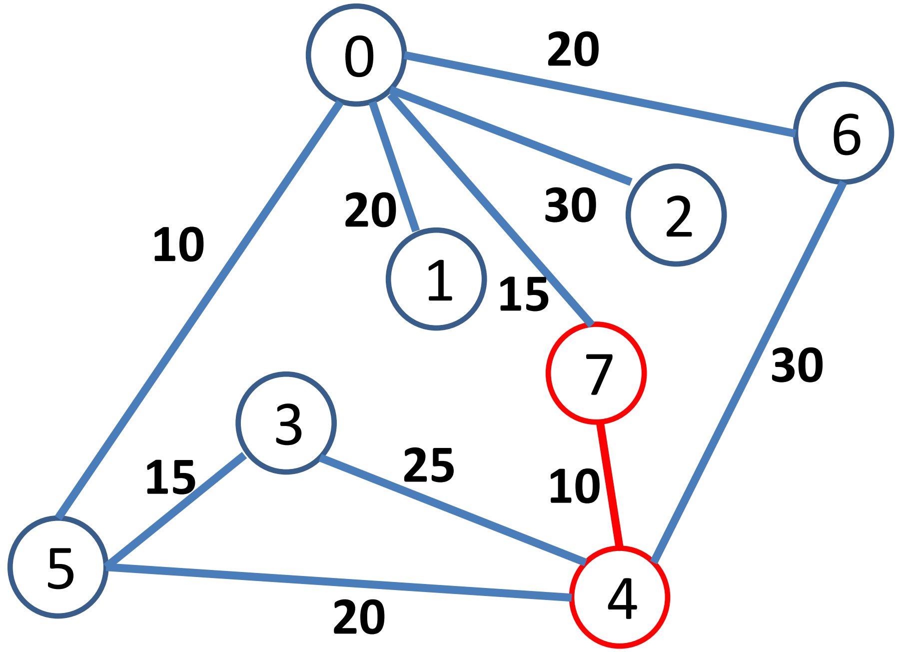
Dijkstra Example



- Suppose we want to compute the SPST for vertex 7.
- Steps 7, 8, 9: v = 4
- Step 10: For w = {3, 5, 6}
  - Step 11: Compare inf with 10+25=35
  - Steps 12, 13: wt[3] = wt[4] + 25, st[3] = 4.

vertex	0	1	2	3	4	5	6	7
wt	15	inf	inf	35	10	inf	inf	0
st	7	-1	-1	4	7	-1	-1	7
in	0	0	0	0	1	0	0	1

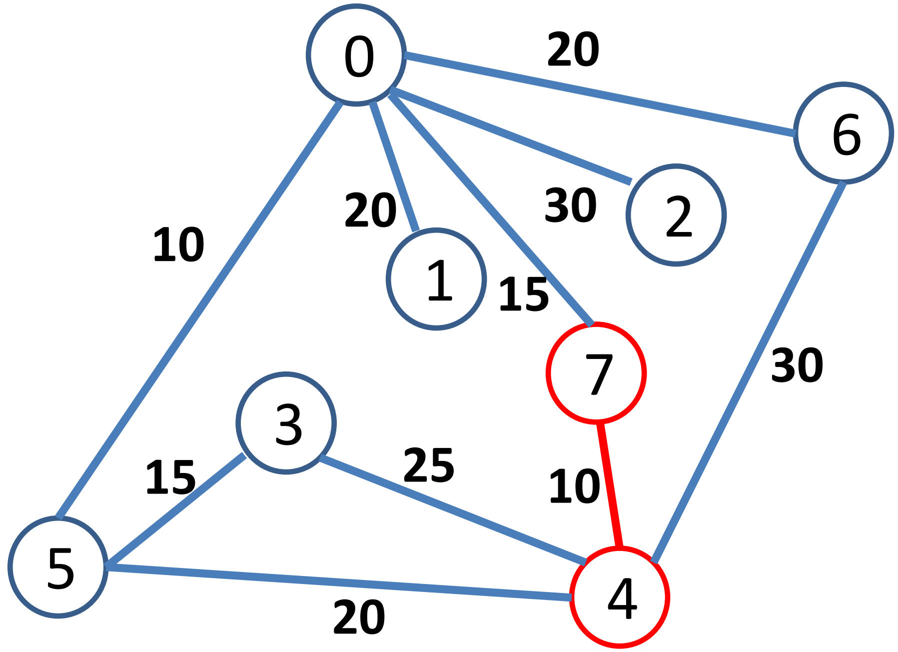
Dijkstra Example



- Suppose we want to compute the SPST for vertex 7.
- Steps 7, 8, 9: v = 4
- Step 10: For w = {3, 5, 6}
  - Step 11: Compare inf with 10+20=30
  - Steps 12, 13: wt[5] = wt[4] + 20, st[5] = 4.

vertex	0	1	2	3	4	5	6	7
wt	15	inf	inf	35	10	30	inf	0
st	7	-1	-1	4	7	4	-1	7
in	0	0	0	0	1	0	0	1

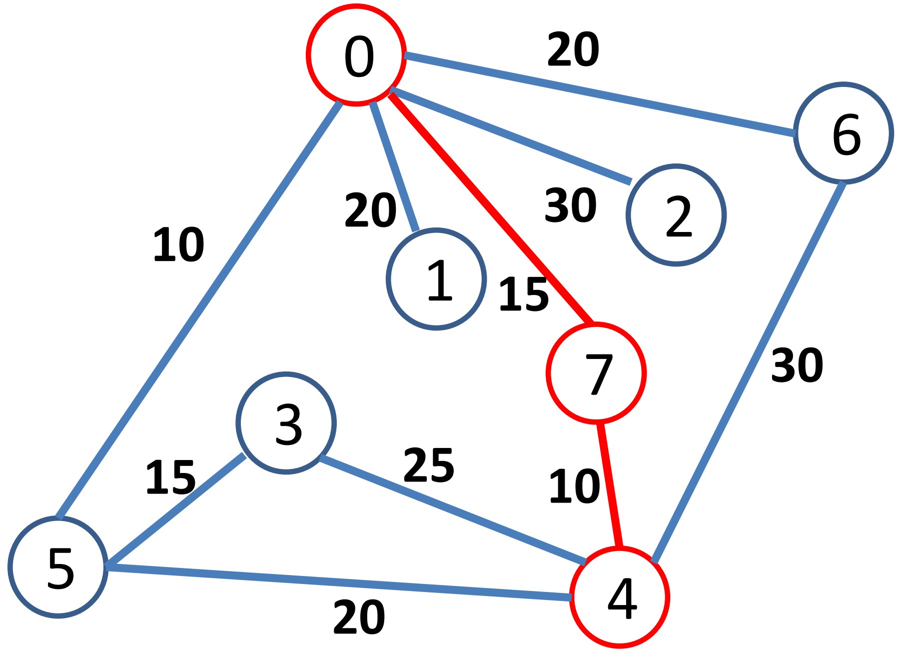
Dijkstra Example



- Suppose we want to compute the SPST for vertex 7.
- Steps 7, 8, 9: v = 4
- Step 10: For w = {3, 5, 6}
  - Step 11: Compare inf with 10+30=40
  - Steps 12, 13: wt[6] = wt[4] + 30, st[6] = 4.

vertex	0	1	2	3	4	5	6	7
wt	15	inf	inf	35	10	30	40	0
st	7	-1	-1	4	7	4	4	7
in	0	0	0	0	1	0	0	1

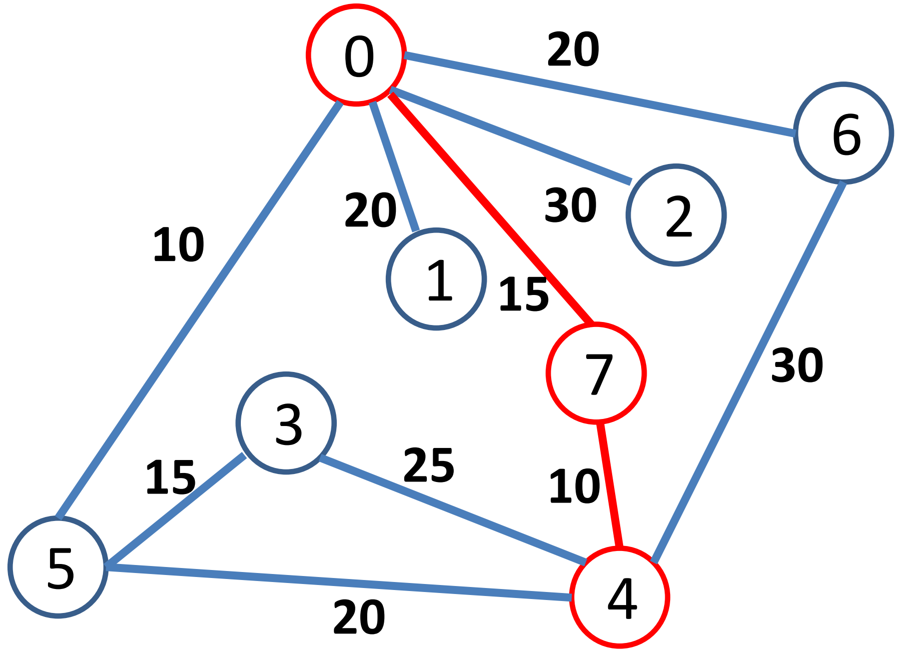
Dijkstra Example



- Suppose we want to compute the SPST for vertex 7.
- Steps 7, 8, 9: v = 0

vertex	0	1	2	3	4	5	6	7
wt	15	inf	inf	35	10	30	40	0
st	7	-1	-1	4	7	4	4	7
in	1	0	0	0	1	0	0	1

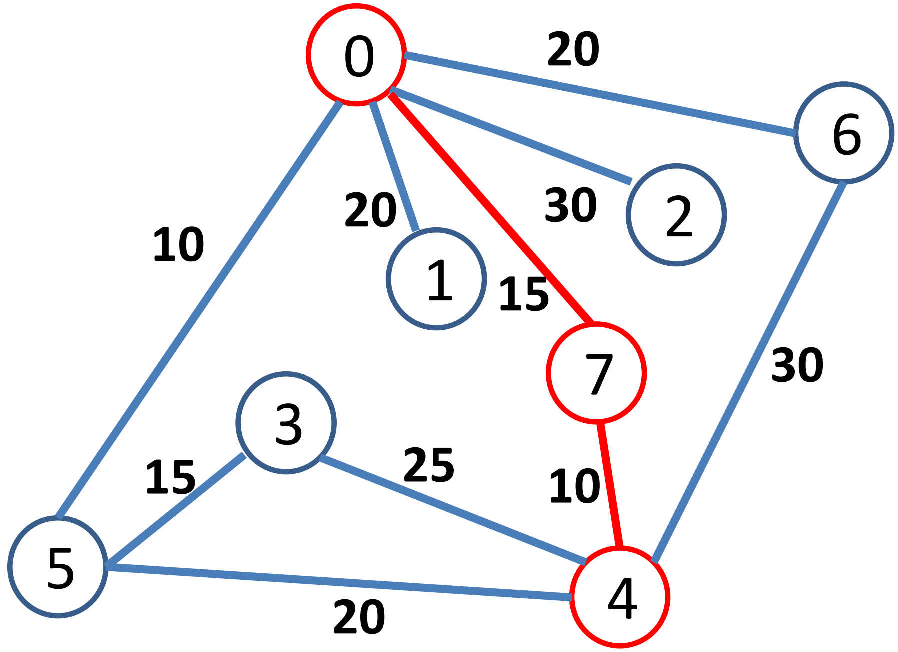
Dijkstra Example



- Suppose we want to compute the SPST for vertex 7.
- Steps 7, 8, 9: v = 0
- Step 10: For w = {1, 2, 5, 6}
  - Step 11: Compare inf with 15+20=35
  - Steps 12, 13: wt[1] = wt[0] + 20, st[1] = 0.

vertex	0	1	2	3	4	5	6	7
wt	15	35	inf	35	10	30	40	0
st	7	0	-1	4	7	4	4	7
in	1	0	0	0	1	0	0	1

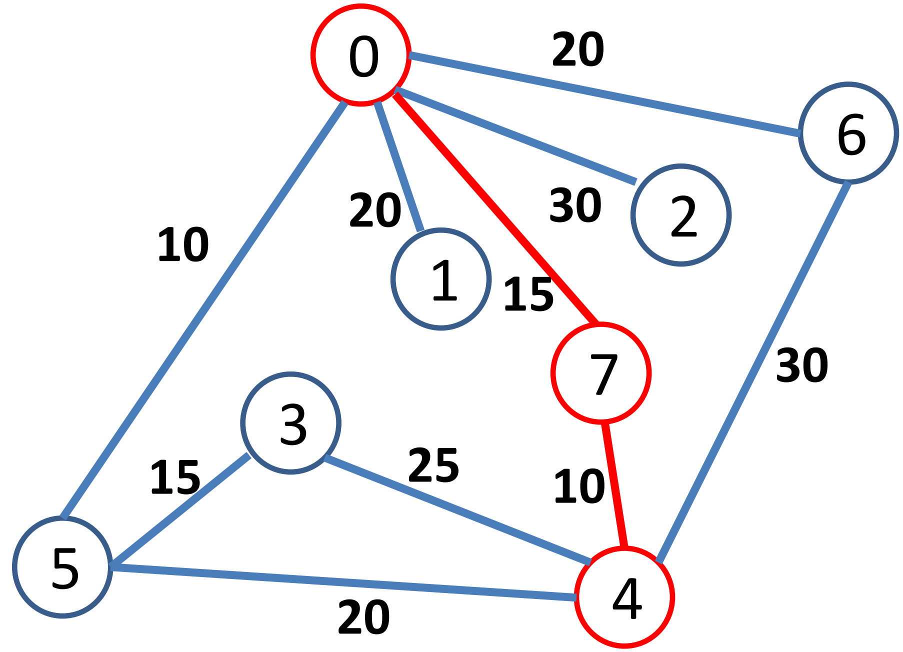
Dijkstra Example



- Suppose we want to compute the SPST for vertex 7.
- Steps 7, 8, 9: v = 0
- Step 10: For w = {1, 2, 5, 6}
  - Step 11: Compare inf with 15+30=45
  - Steps 12, 13: wt[2] = wt[0] + 30, st[2] = 0.

vertex	0	1	2	3	4	5	6	7
wt	15	35	45	35	10	30	40	0
st	7	0	0	4	7	4	4	7
in	1	0	0	0	1	0	0	1

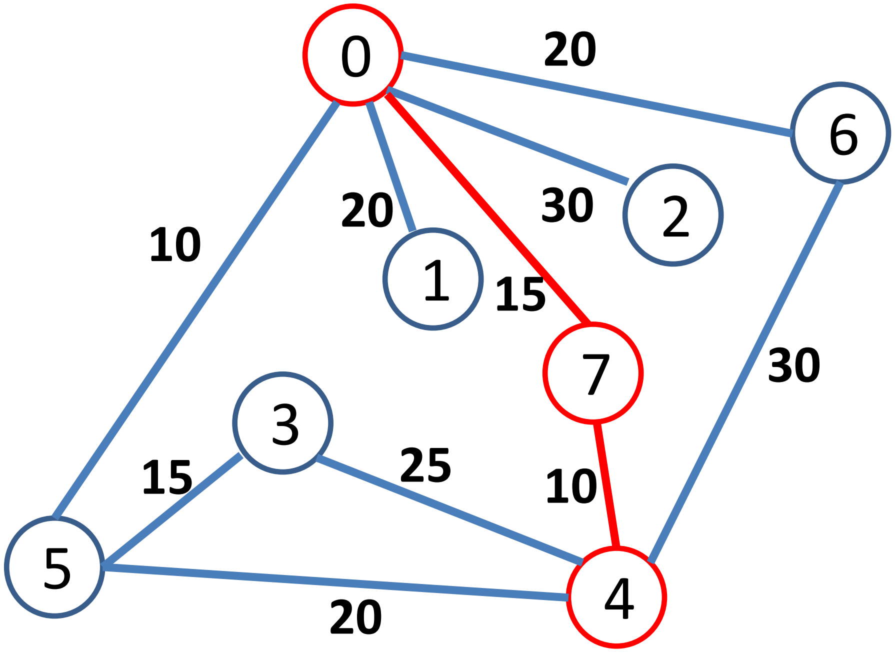
Dijkstra Example



- Suppose we want to compute the SPST for vertex 7.
- Steps 7, 8, 9: v = 0
- Step 10: For w = {1, 2, 5, 6}
  - Step 11: Compare 30 with 15+10=25
  - Steps 12, 13: wt[5] = wt[0] + 10, st[5] = 0.

vertex	0	1	2	3	4	5	6	7
wt	15	35	45	35	10	25	40	0
st	7	0	0	4	7	0	4	7
in	1	0	0	0	1	0	0	1

Dijkstra Example

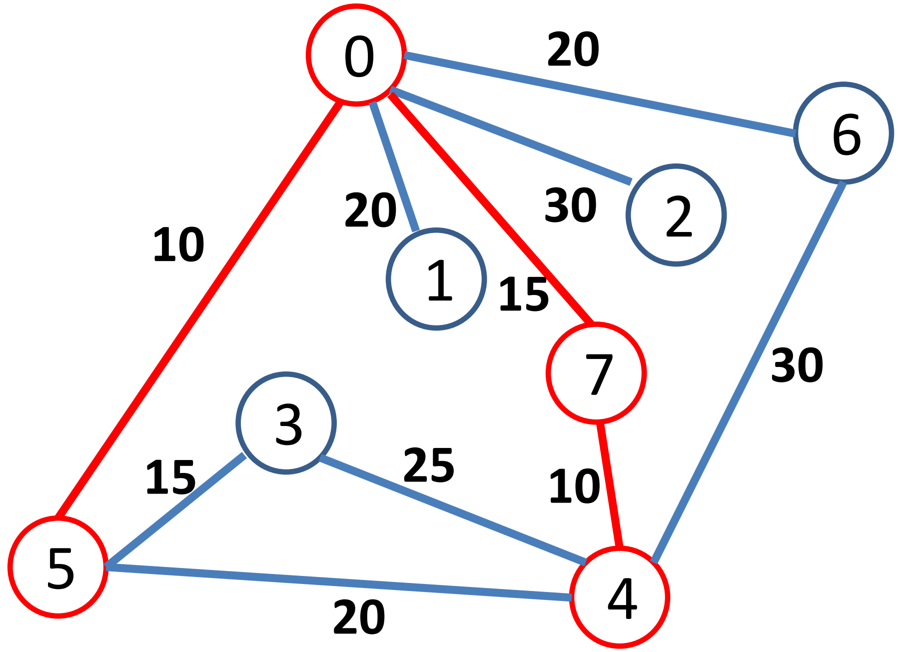


- Suppose we want to compute the SPST for vertex 7.
- Steps 7, 8, 9: v = 0
- Step 10: For w = {1, 2, 5, 6}
  - Step 11: Compare 40 with 15+20=35
  - Steps 12, 13: wt[6] = wt[0] + 20, st[6] = 0.

vertex	0	1	2	3	4	5	6	7
wt	15	35	45	35	10	25	35	0
st	7	0	0	4	7	0	0	7
in	1	0	0	0	1	0	0	1

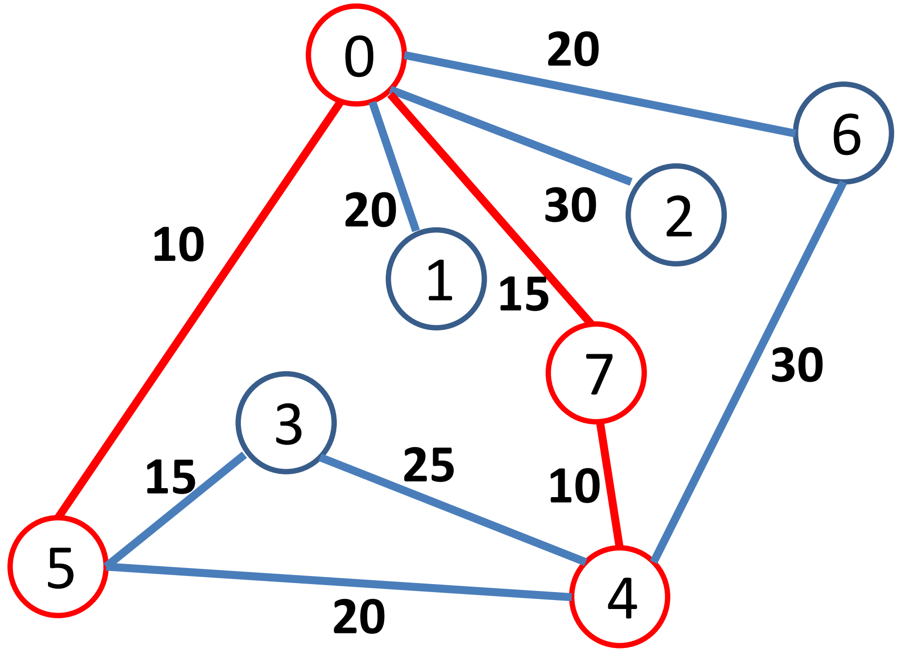
Dijkstra Example

- Suppose we want to compute the SPST for vertex 7.
- Steps 7, 8, 9: v = 5



vertex	0	1	2	3	4	5	6	7
wt	15	35	45	35	10	25	35	0
st	7	0	0	4	7	0	0	7
in	1	0	0	0	1	1	0	1

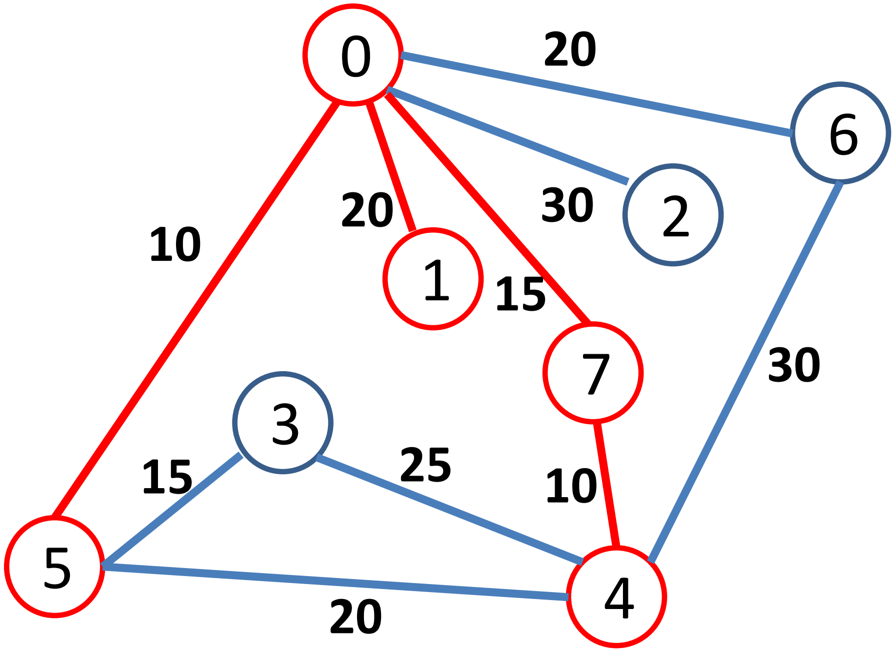
Dijkstra Example



- Suppose we want to compute the SPST for vertex 7.
- Steps 7, 8, 9: v = 5
- Step 10: For w = {3}
  - Step 11: Compare 35 with 25+15=40
    NO UPDATE

vertex	0	1	2	3	4	5	6	7
wt	15	35	45	35	10	25	35	0
st	7	0	0	4	7	0	0	7
in	1	0	0	0	1	1	0	1

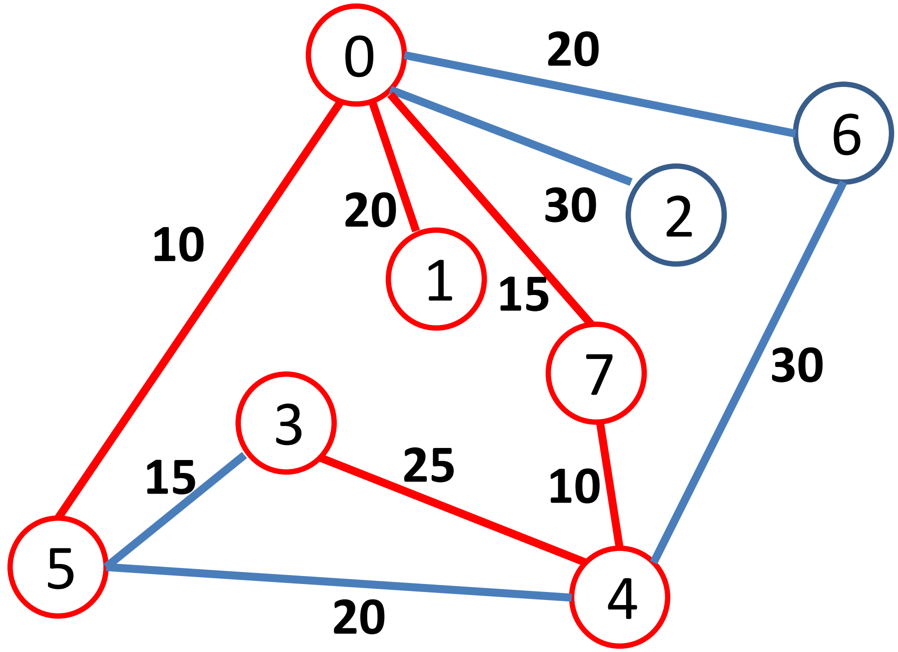
Dijkstra Example



- Suppose we want to compute the SPST for vertex 7.
- Steps 7, 8, 9: v = 1
- Step 10: For w = empty list

vertex	0	1	2	3	4	5	6	7
wt	15	35	45	35	10	25	35	0
st	7	0	0	4	7	0	0	7
in	1	1	0	0	1	1	0	1

Dijkstra Example

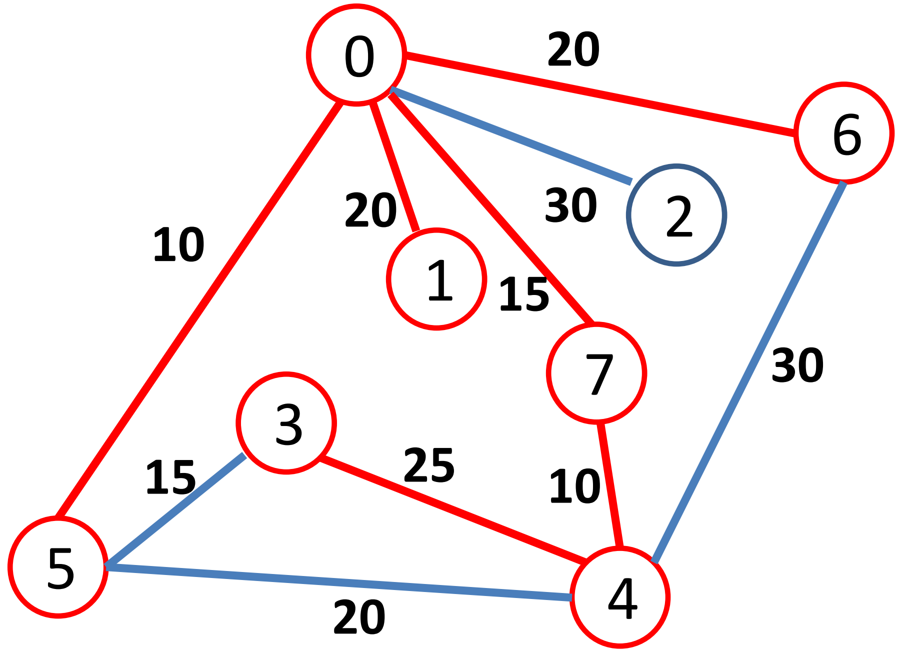


- Suppose we want to compute the SPST for vertex 7.
- Steps 7, 8, 9: v = 3
- Step 10: For w = empty list

vertex	0	1	2	3	4	5	6	7
wt	15	35	45	35	10	25	35	0
st	7	0	0	4	7	0	0	7
in	1	1	0	1	1	1	0	1

Dijkstra Example

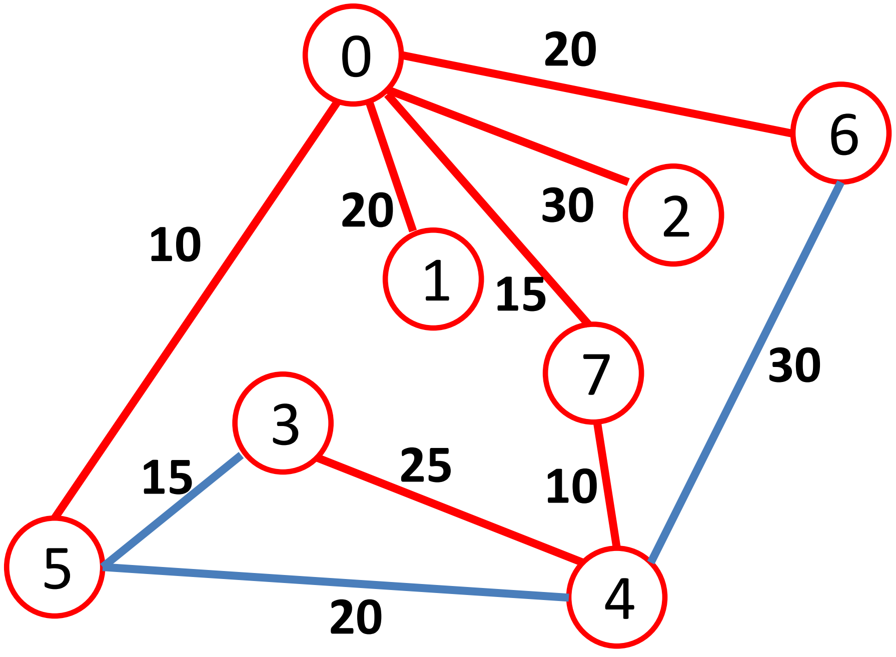
- Suppose we want to compute the SPST for vertex 7.
- Steps 7, 8, 9: v = 6
- Step 10: For w = empty list



vertex	0	1	2	3	4	5	6	7
wt	15	35	45	35	10	25	35	0
st	7	0	0	4	7	0	0	7
in	1	1	0	1	1	1	1	1

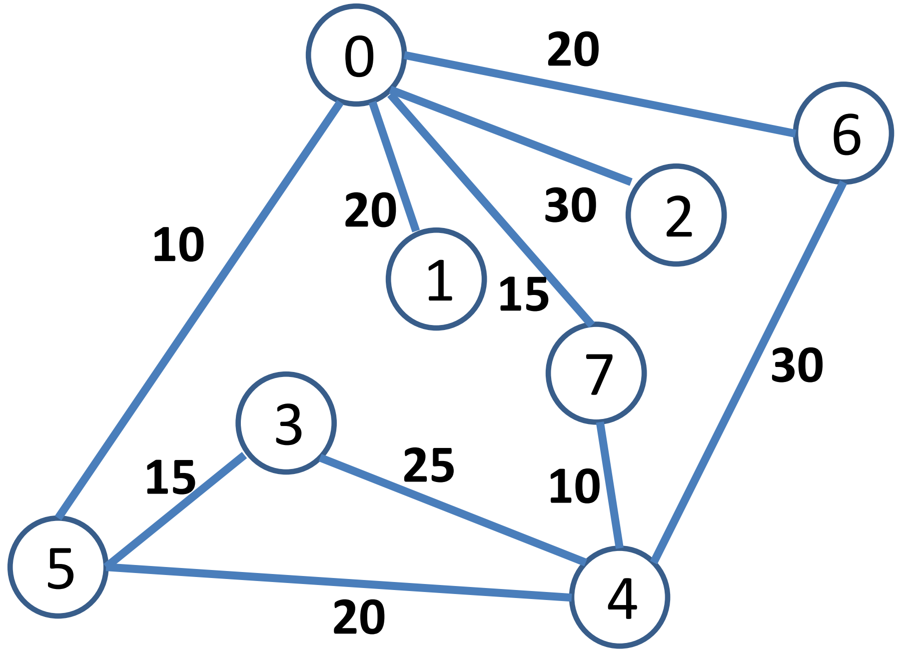
Dijkstra Example

- Suppose we want to compute the SPST for vertex 7.
- Steps 7, 8, 9: v = 6
- Step 10: For w = empty list



vertex	0	1	2	3	4	5	6	7
wt	15	35	45	35	10	25	35	0
st	7	0	0	4	7	0	0	7
in	1	1	1	1	1	1	1	1

Dijkstra Example

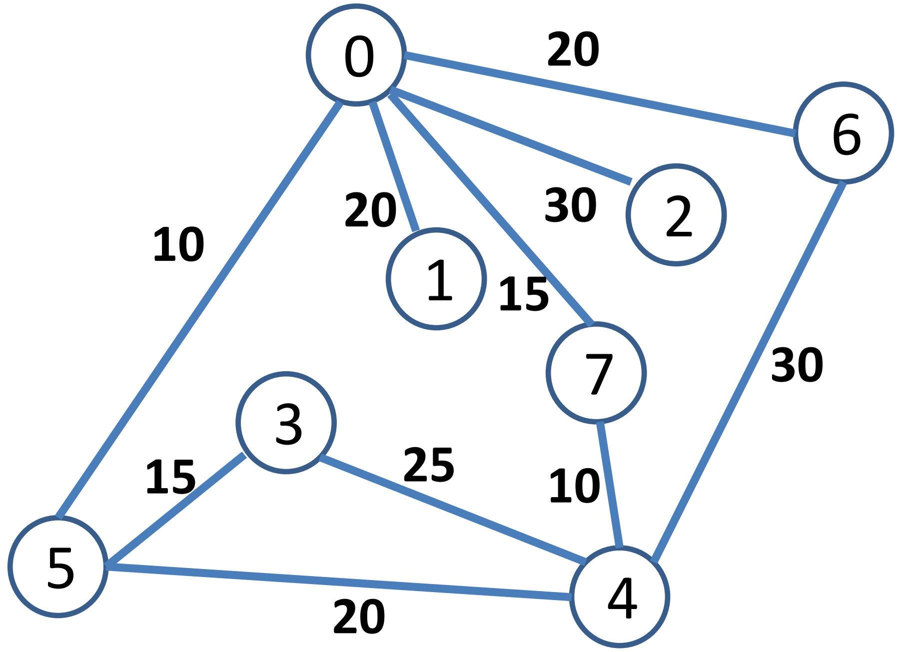


- Suppose we want to compute the SPST for vertex 4.
- First, we initialize arrays wt, st, in (steps 2, 3, 4).

vertex	0	1	2	3	4	5	6	7
wt	inf	inf	inf	inf	inf	inf	inf	inf
st	-1	-1	-1	-1	-1	-1	-1	-1
in	0	0	0	0	0	0	0	0

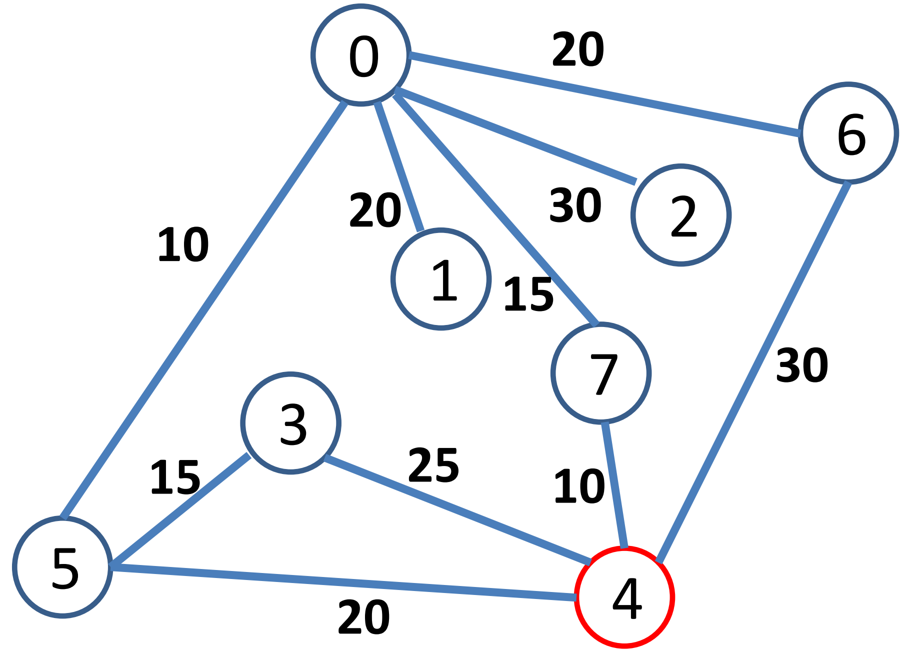
Dijkstra Example

- Suppose we want to compute the SPST for vertex 4.
- Step 5.



vertex	0	1	2	3	4	5	6	7
wt	inf	inf	inf	inf	0	inf	inf	inf
st	-1	-1	-1	-1	4	-1	-1	-1
in	0	0	0	0	0	0	0	0

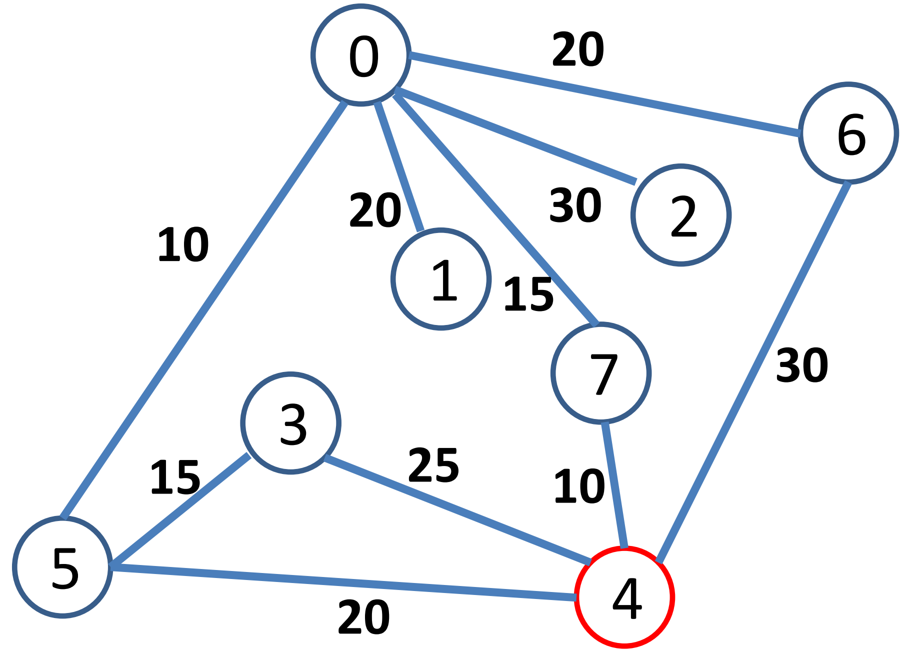
Dijkstra Example



- Suppose we want to compute the SPST for vertex 4.
- Steps 7, 8, 9: v = 4

vertex	0	1	2	3	4	5	6	7
wt	inf	inf	inf	inf	0	inf	inf	inf
st	-1	-1	-1	-1	4	-1	-1	-1
in	0	0	0	0	1	0	0	0

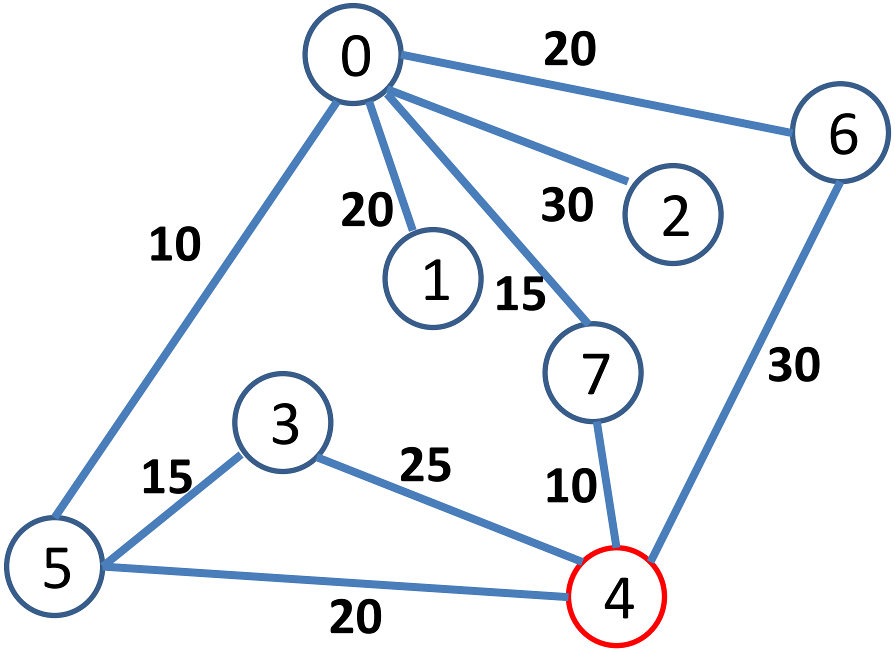
Dijkstra Example



- Suppose we want to compute the SPST for vertex 4.
- Steps 7, 8, 9: v = 4
- Step 10: For w = {3, 5, 6, 7}
  - Step 11: Compare inf with 25
  - Steps 12, 13: wt[3] = wt[4] + 25, st[3] = 4.

vertex	0	1	2	3	4	5	6	7
wt	inf	inf	inf	25	0	inf	inf	inf
st	-1	-1	-1	4	4	-1	-1	-1
in	0	0	0	0	1	0	0	0

Dijkstra Example

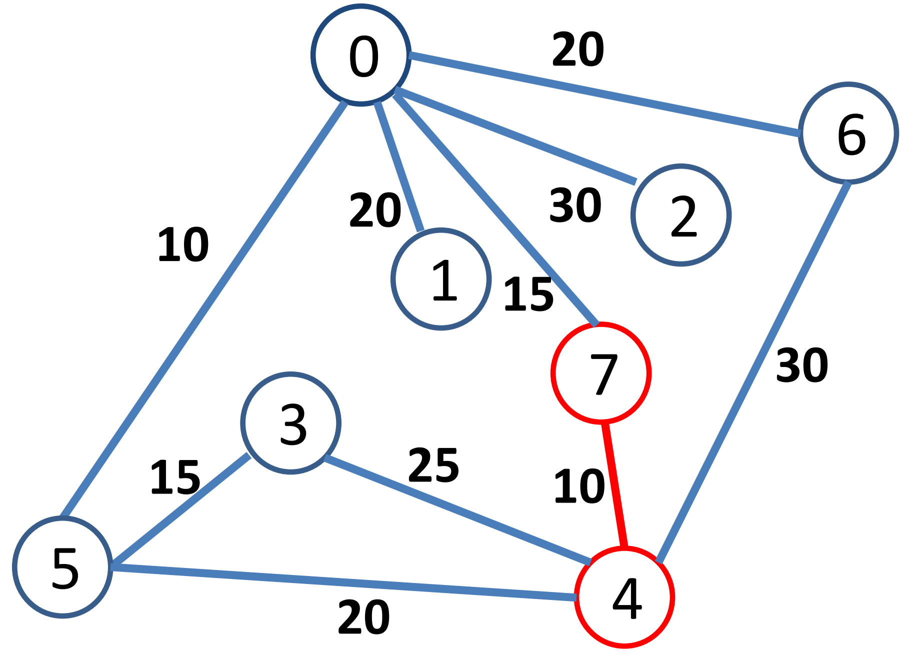


- Suppose we want to compute the SPST for vertex 4.
- Steps 7, 8, 9: v = 4
- Step 10: For w = {3, 5, 6, 7}
  - Steps 12, 13: update wt[w], st[w]

vertex	0	1	2	3	4	5	6	7
wt	inf	inf	inf	25	0	20	30	10
st	-1	-1	-1	4	4	4	4	4
in	0	0	0	0	1	0	0	0

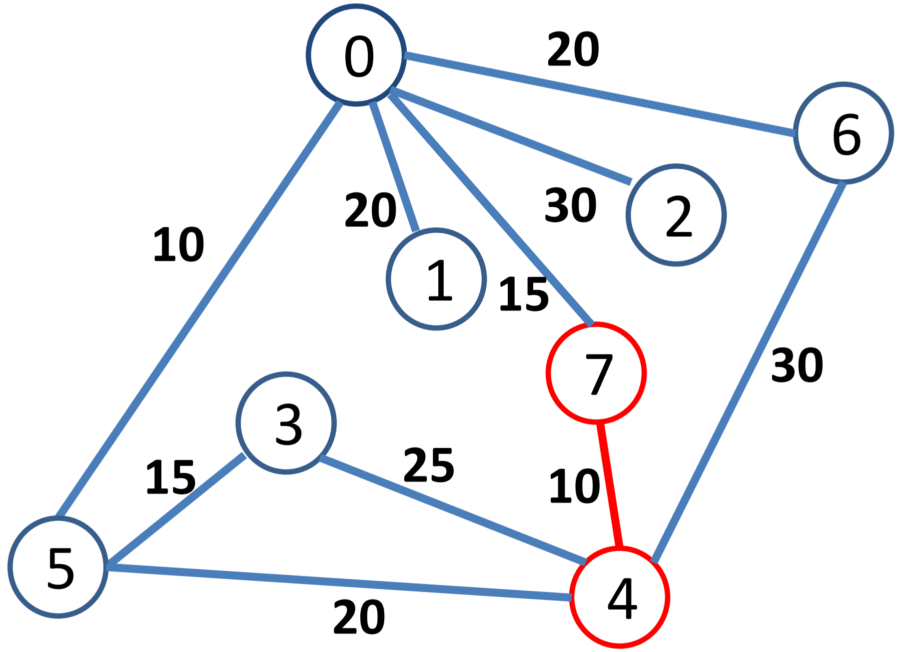
Dijkstra Example

- Suppose we want to compute the SPST for vertex 4.
- Steps 7, 8, 9: v = 7



vertex	0	1	2	3	4	5	6	7
wt	inf	inf	inf	25	0	20	30	10
st	-1	-1	-1	4	4	4	4	4
in	0	0	0	0	1	0	0	1

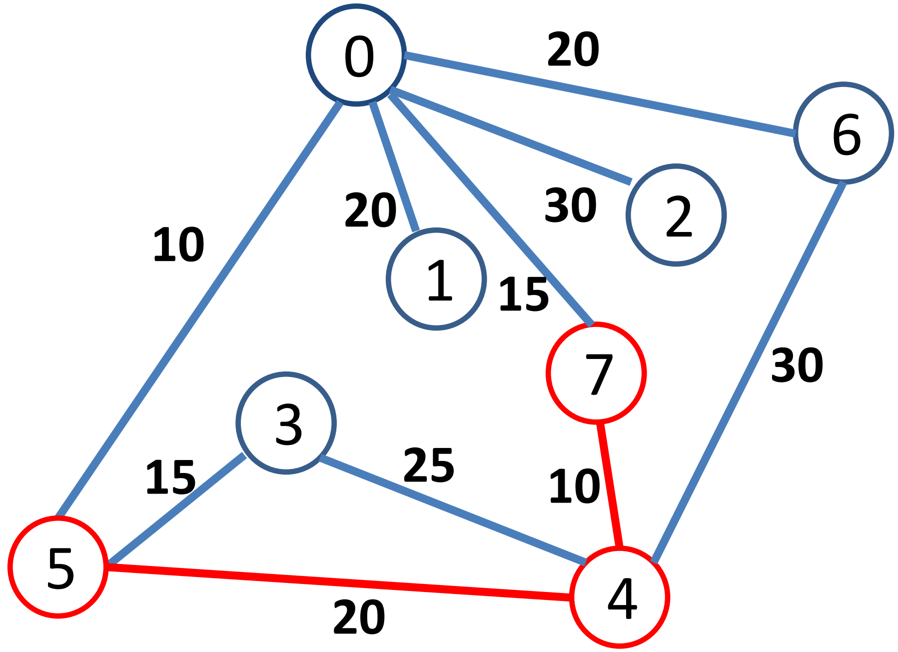
Dijkstra Example



- Suppose we want to compute the SPST for vertex 4.
- Steps 7, 8, 9: v = 7
- Step 10: For w = {0}
  - Step 11: Compare inf with 10+15 = 25.
  - Steps 12, 13: wt[0] = wt[7] + 15, st[0] = 7.

vertex	0	1	2	3	4	5	6	7
wt	25	inf	inf	25	0	20	30	10
st	7	-1	-1	4	4	4	4	4
in	0	0	0	0	1	0	0	1

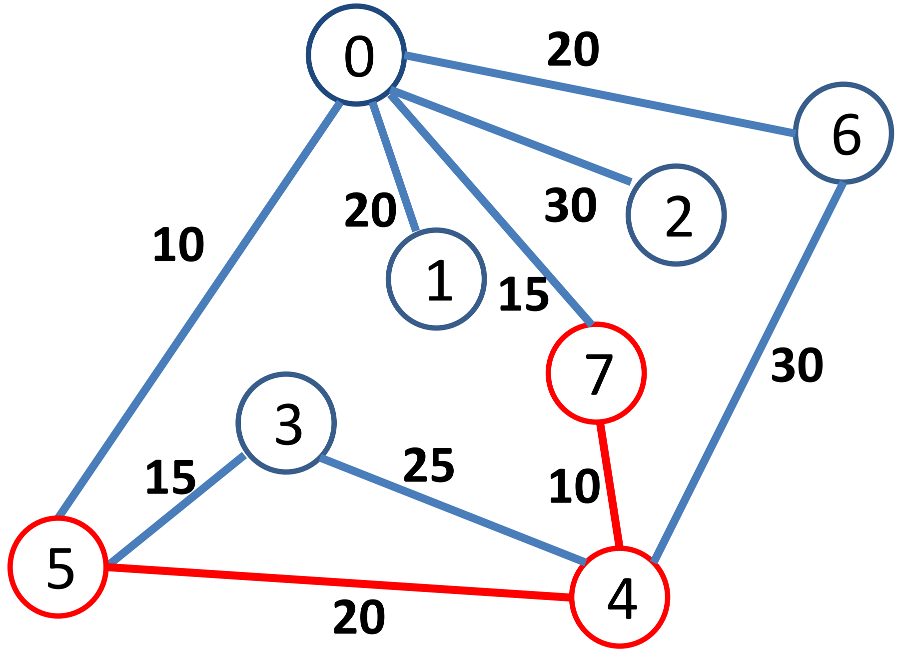
Dijkstra Example



- Suppose we want to compute the SPST for vertex 4.
- Steps 7, 8, 9: v = 5

vertex	0	1	2	3	4	5	6	7
wt	25	inf	inf	25	0	20	30	10
st	7	-1	-1	4	4	4	4	4
in	0	0	0	0	1	1	0	1

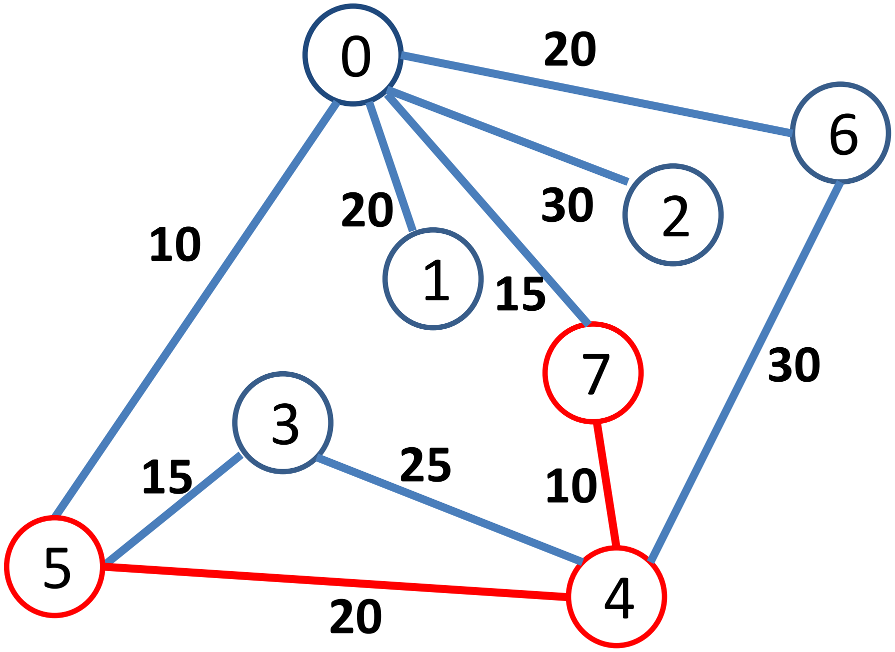
Dijkstra Example



- Suppose we want to compute the SPST for vertex 4.
- Steps 7, 8, 9: v = 5
- Step 10: For w = {0, 3}
  - Step 11: Compare 25 with 20+10 = 25.
    NO UPDATE

vertex	0	1	2	3	4	5	6	7
wt	25	inf	inf	25	0	20	30	10
st	7	-1	-1	4	4	4	4	4
in	0	0	0	0	1	1	0	1

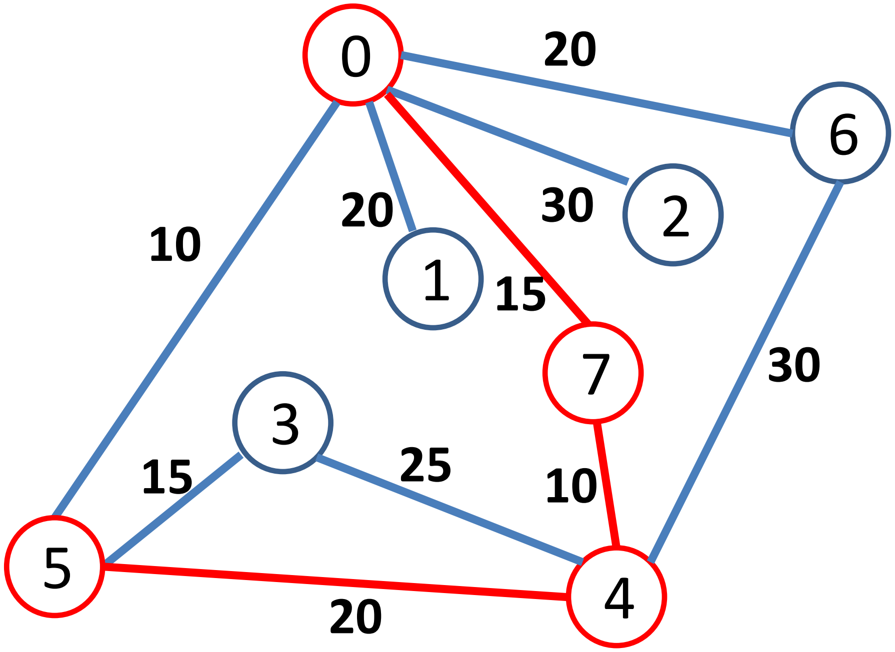
Dijkstra Example



- Suppose we want to compute the SPST for vertex 4.
- Steps 7, 8, 9: v = 5
- Step 10: For w = {0, 3}
  - Step 11: Compare 25 with 20+15 = 35.
    NO UPDATE

vertex	0	1	2	3	4	5	6	7
wt	25	inf	inf	25	0	20	30	10
st	7	-1	-1	4	4	4	4	4
in	0	0	0	0	1	1	0	1

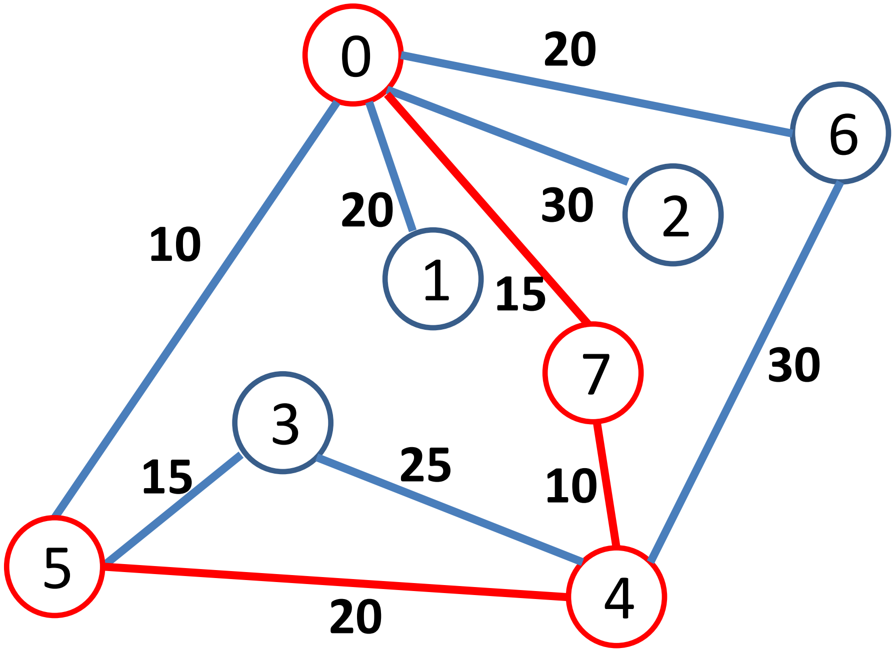
Dijkstra Example



- Suppose we want to compute the SPST for vertex 4.
- Steps 7, 8, 9: v = 0

vertex	0	1	2	3	4	5	6	7
wt	25	inf	inf	25	0	20	30	10
st	7	-1	-1	4	4	4	4	4
in	1	0	0	0	1	1	0	1

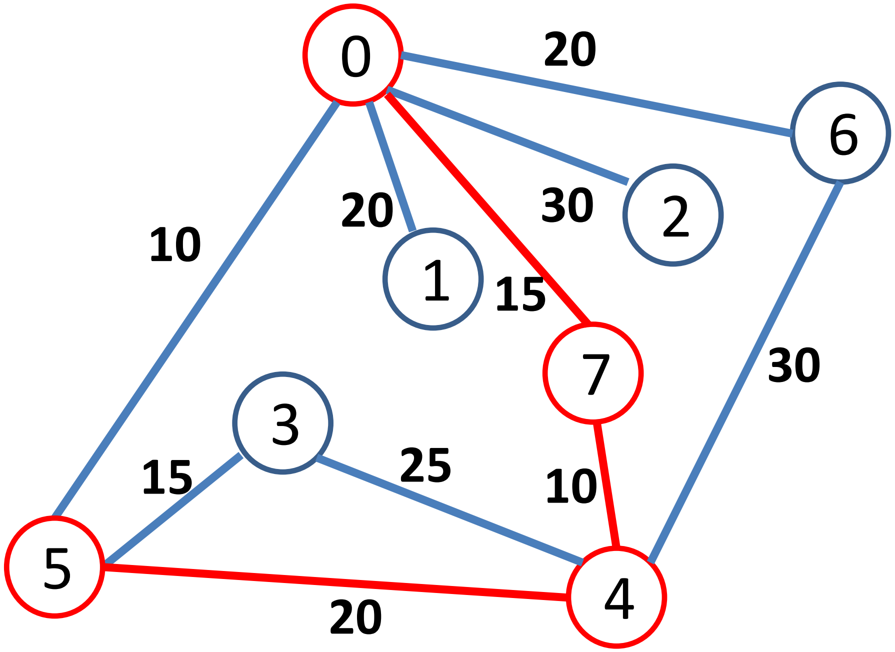
Dijkstra Example



- Suppose we want to compute the SPST for vertex 4.
- Steps 7, 8, 9: v = 0
- Step 10: For w = {1, 2, 6}
  - Step 11: Compare inf with 25+20 = 45.
  - Steps 12, 13: wt[1] = wt[0] + 20, st[1] = 0.

vertex	0	1	2	3	4	5	6	7
wt	25	45	inf	25	0	20	30	10
st	7	0	-1	4	4	4	4	4
in	1	0	0	0	1	1	0	1

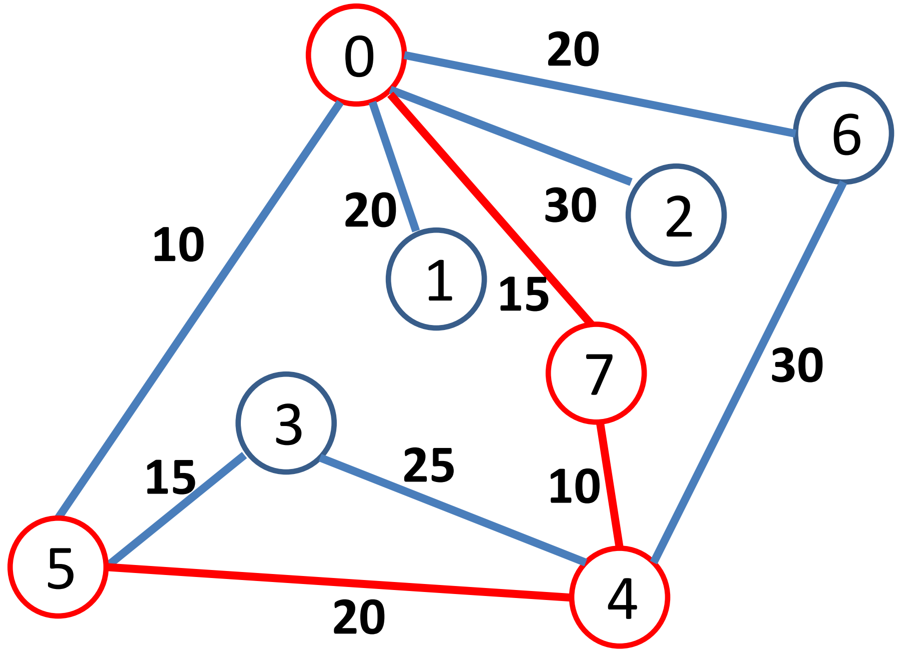
Dijkstra Example



- Suppose we want to compute the SPST for vertex 4.
- Steps 7, 8, 9: v = 0
- Step 10: For w = {1, 2, 6}
  - Step 11: Compare inf with 25+30 = 55.
  - Steps 12, 13: wt[2] = wt[0] + 30, st[2] = 0.

vertex	0	1	2	3	4	5	6	7
wt	25	45	55	25	0	20	30	10
st	7	0	0	4	4	4	4	4
in	1	0	0	0	1	1	0	1

Dijkstra Example

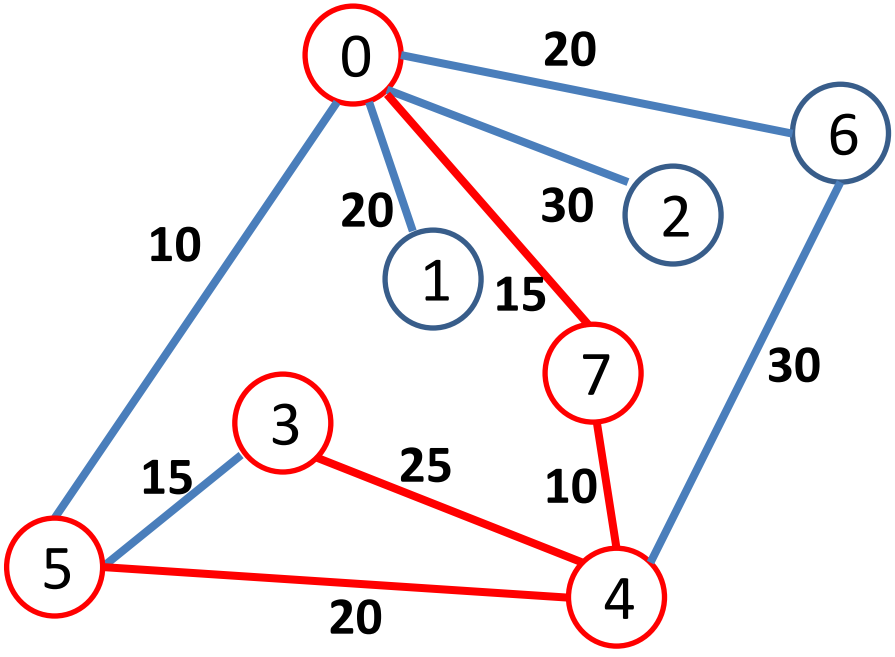


- Suppose we want to compute the SPST for vertex 4.
- Steps 7, 8, 9: v = 0
- Step 10: For w = {1, 2, 6}
  - Step 11: Compare 30 with 25+20 = 45. NO UPDATE

vertex	0	1	2	3	4	5	6	7
wt	25	45	55	25	0	20	30	10
st	7	0	0	4	4	4	4	4
in	1	0	0	0	1	1	0	1

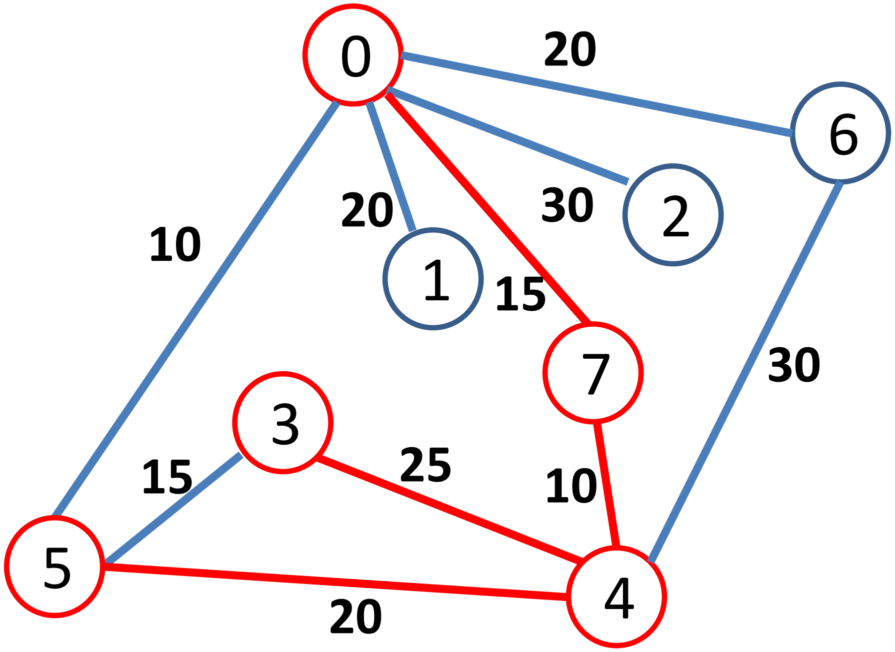
Dijkstra Example

- Suppose we want to compute the SPST for vertex 4.
- Steps 7, 8, 9: v = 3



vertex	0	1	2	3	4	5	6	7
wt	25	45	55	25	0	20	30	10
st	7	0	0	4	4	4	4	4
in	1	0	0	1	1	1	0	1

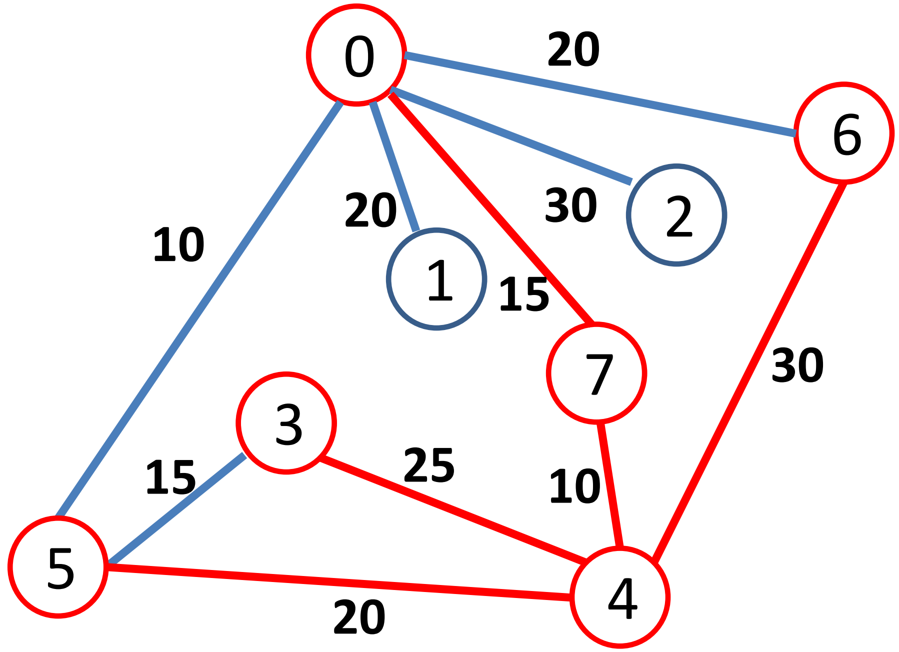
Dijkstra Example



- Suppose we want to compute the SPST for vertex 4.
- Steps 7, 8, 9: v = 3
- Step 10: empty list

vertex	0	1	2	3	4	5	6	7
wt	25	45	55	25	0	20	30	10
st	7	0	0	4	4	4	4	4
in	1	0	0	1	1	1	0	1

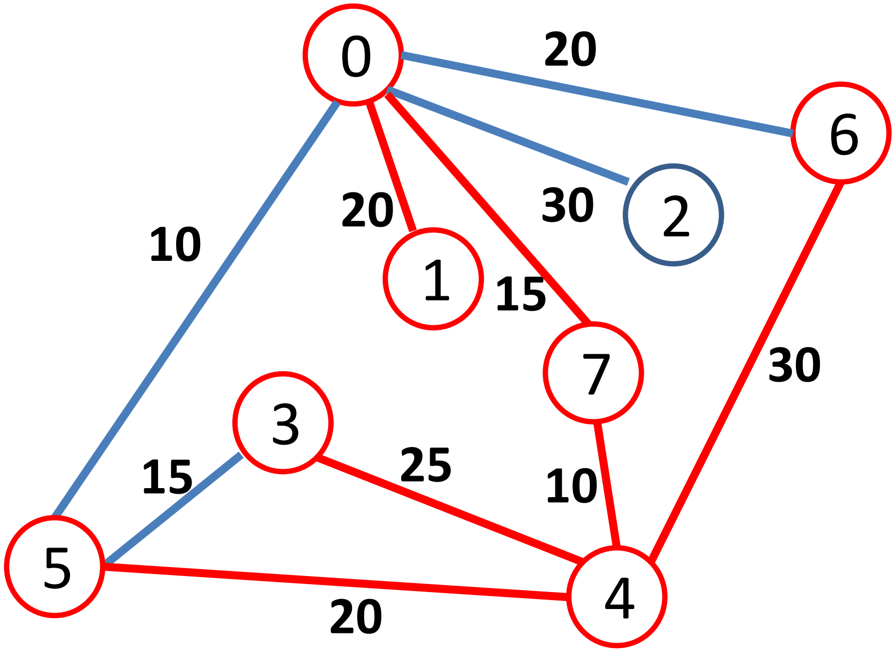
Dijkstra Example



- Suppose we want to compute the SPST for vertex 4.
- Steps 7, 8, 9: v = 6
- Step 10: empty list

vertex	0	1	2	3	4	5	6	7
wt	25	45	55	25	0	20	30	10
st	7	0	0	4	4	4	4	4
in	1	0	0	1	1	1	1	1

Dijkstra Example

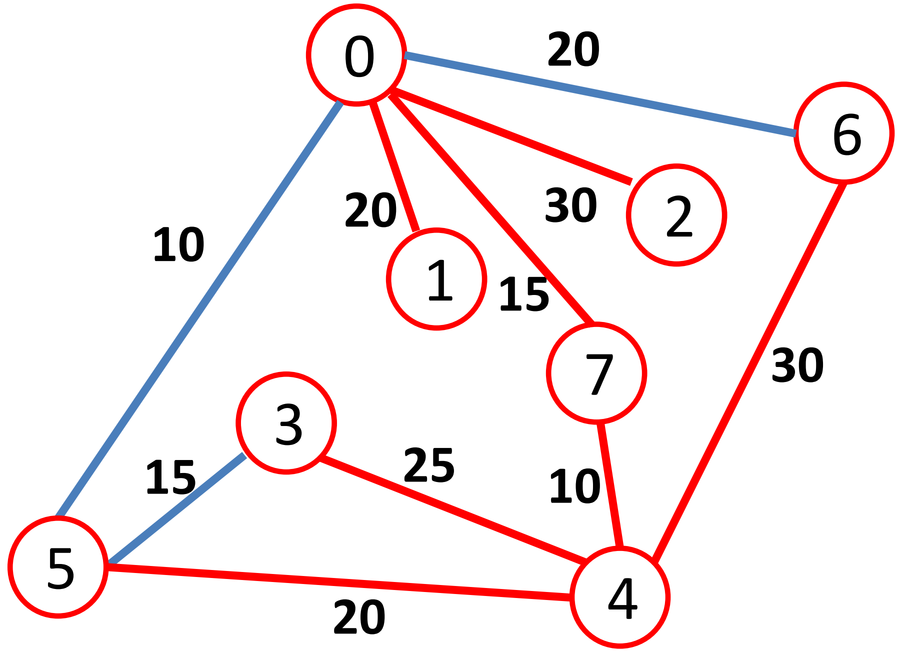


- Suppose we want to compute the SPST for vertex 4.
- Steps 7, 8, 9: v = 1
- Step 10: empty list

vertex	0	1	2	3	4	5	6	7
wt	25	45	55	25	0	20	30	10
st	7	0	0	4	4	4	4	4
in	1	1	0	1	1	1	1	1

Dijkstra Example

- Suppose we want to compute the SPST for vertex 4.
- Steps 7, 8, 9: v = 2
- Step 10: empty list



vertex	0	1	2	3	4	5	6	7
wt	25	45	55	25	0	20	30	10
st	7	0	0	4	4	4	4	4
in	1	1	1	1	1	1	1	1

All-Pairs Shortest Paths

- Before we describe an algorithm for computing the shortest paths among all pairs of vertices, we should agree on what this algorithm should return.
- We need to compute two V x V arrays:
  - dist[v][w] is the distance of the shortest path from v to w.
  - path[v][w] is the vertex following v, on the shortest path from v to w.
- Given these two arrays (after our algorithm has completed), how can we recover the shortest path between some v and w?

All-Pairs Shortest Paths

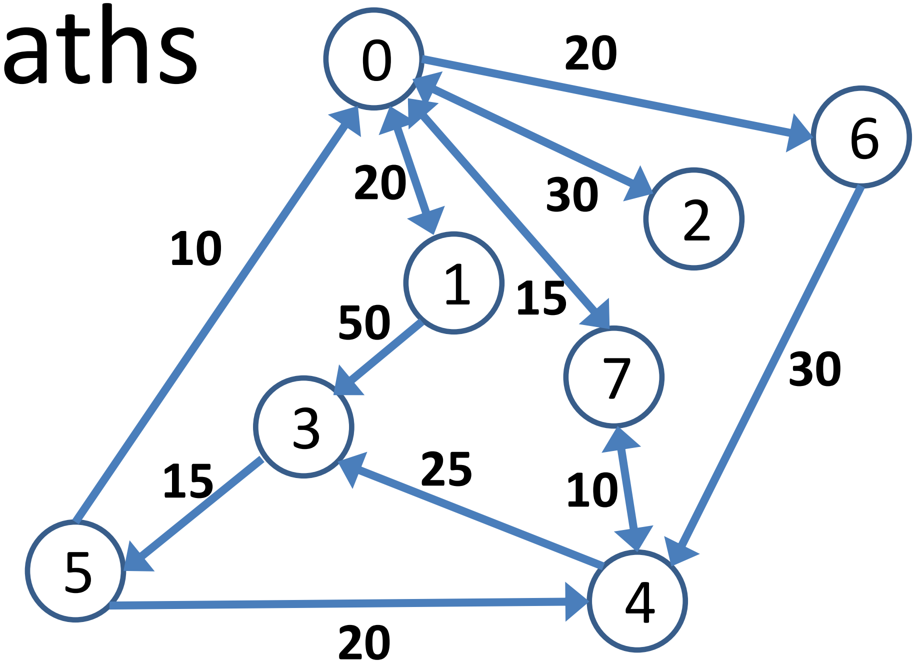
- We need to compute two V x V arrays:
  - dist[v][w] is the distance of the shortest path from v to w.
  - path[v][w] is the vertex following v, on the shortest path from v to w.
- Given these two arrays (after our algorithm has completed), how can we recover the shortest path between some v and w?

- path = empty list
- c = v
- while(true)
  - insert_to_end(path, c)
  - if (c == w) break
  - c = path[c][w]

Computing Shortest Paths

- Overview: we can simply call Dijkstra's algorithm on each vertex.
- Time: V times the time of running Dijkstra's algorithm once.
  - O(E lg V) for one vertex.
  - O(VE lg V) for all vertices.
  - $O(V^3 \lg V)$ for dense graphs.
- There is a better algorithm for dense graphs, Floyd's algorithm, with $O(V^3)$ complexity, but we will not cover it.

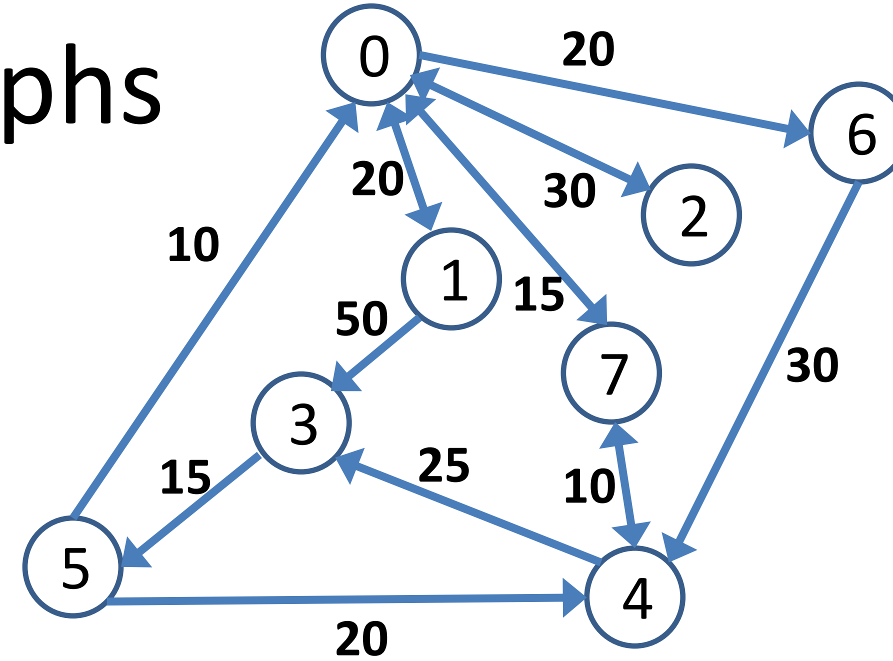
All-Pairs Shortest Paths Using Dijkstra



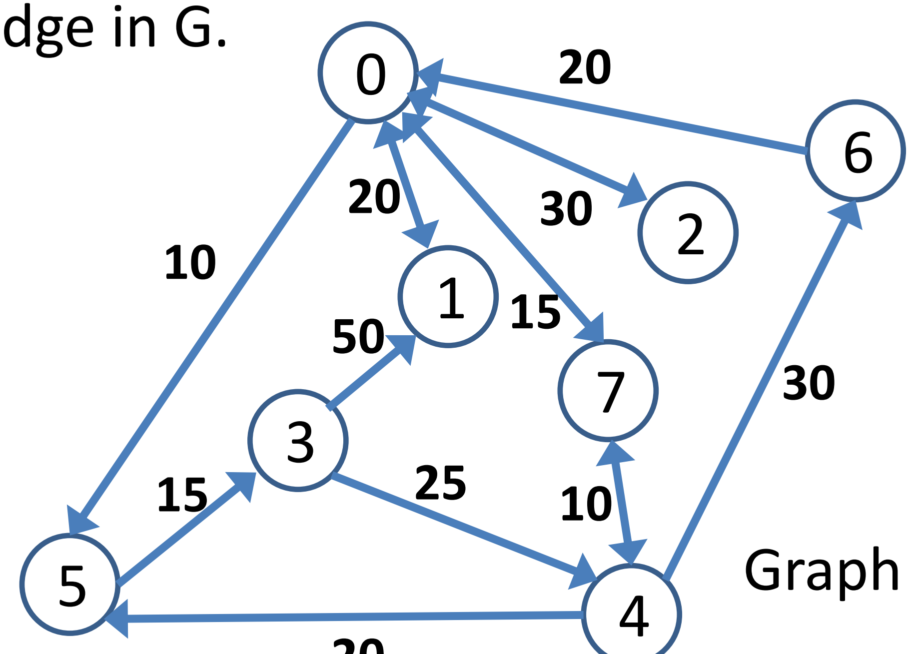
- The complete all-pairs algorithm is more complicated than simply calling Dijkstra's algorithm V times.
- Here is why:
- Suppose we call Dijkstra's algorithm on vertex 1.
- The algorithm computes arrays wt and st:
  - wt[v]: weight of shortest path from source to v.
  - st[v]: parent vertex of v on shortest path from source to v.
- How do arrays wt and st correspond to arrays dist and path?
  - dist[v][w] is the distance of the shortest path from v to w.
  - path[v][w] is the vertex following v, on the shortest path from v to w.
- No useful correspondence!!!

Using Reverse Graphs

- Suppose that G is the graph you see on the right.
- Suppose that H is the reverse graph, obtained by switching the direction of every single edge in G.

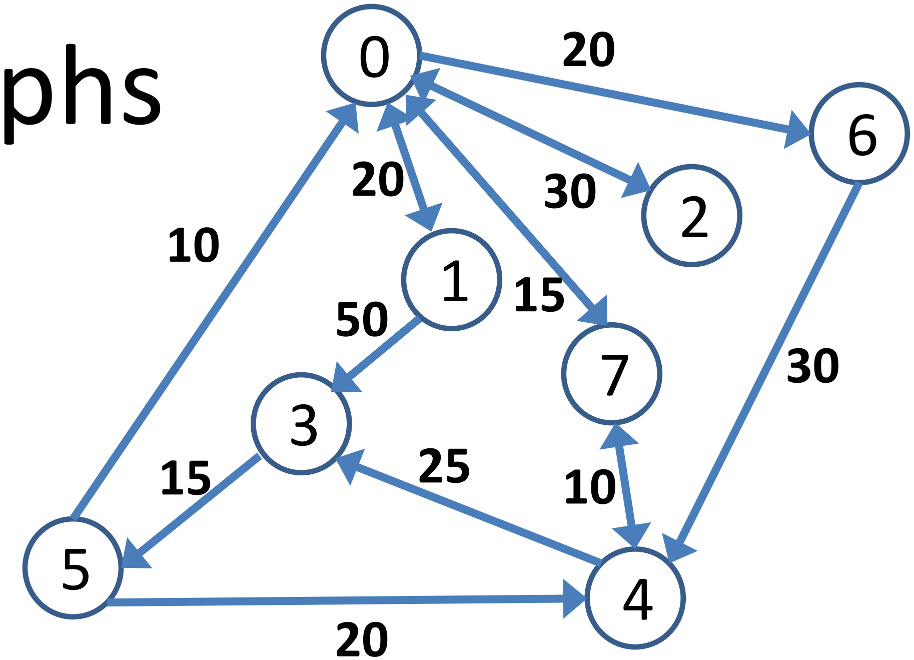


Graph G



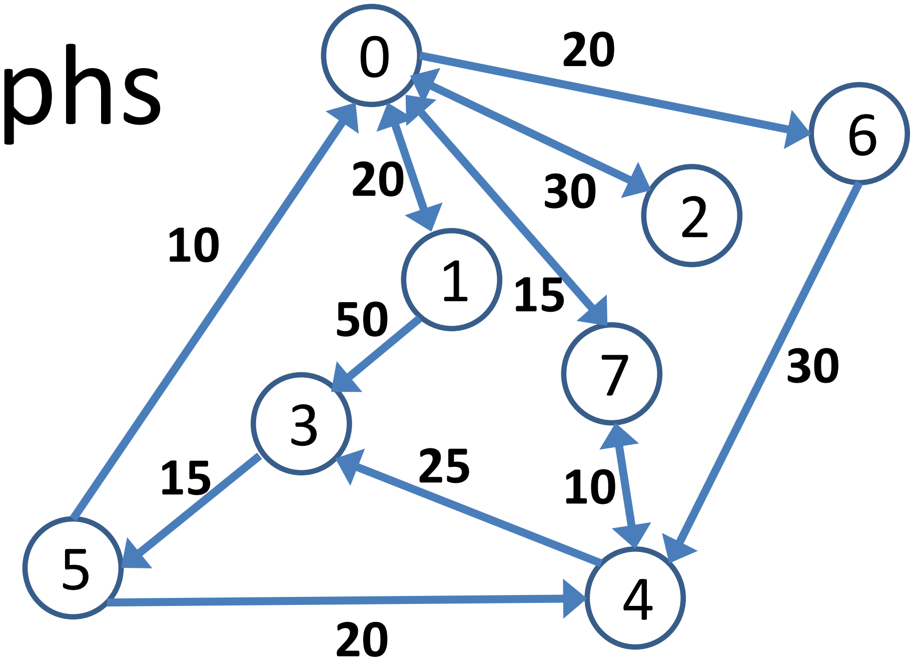
Graph H

Using Reverse Graphs



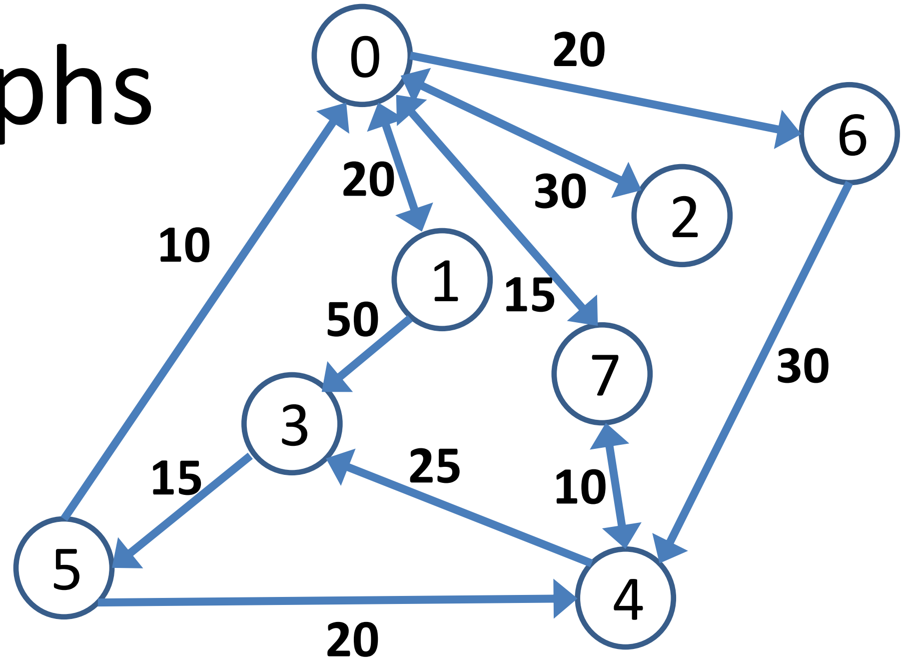
- Suppose that G is the graph you see on the right.
- Suppose that H is the reverse graph, obtained by switching the direction of every single edge in G.
- Then, for any vertices v and w, the shortest path from w to v in H is simply the reverse of the shortest path from v to w in G.
- For example:
  - Shortest path from 1 to 7 in G:
  - Shortest path from 7 to 1 in H:

Using Reverse Graphs



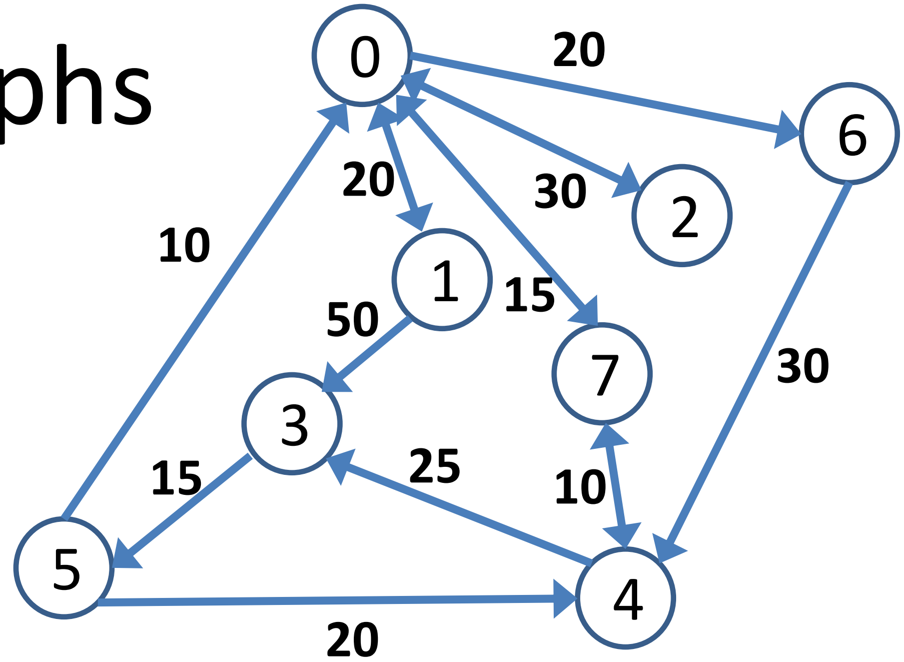
- Suppose that G is the graph you see on the right.
- Suppose that H is the reverse graph, obtained by switching the direction of every single edge in G.
- Then, for any vertices v and w, the shortest path from w to v in H is simply the reverse of the shortest path from v to w in G.
- For example:
  - Shortest path from 1 to 7 in G: 1, 0, 7, 4
  - Shortest path from 7 to 1 in H: 4, 7, 0, 1.
  - These two paths are just reversed forms of each other, and they have the same weights.

Using Reverse Graphs



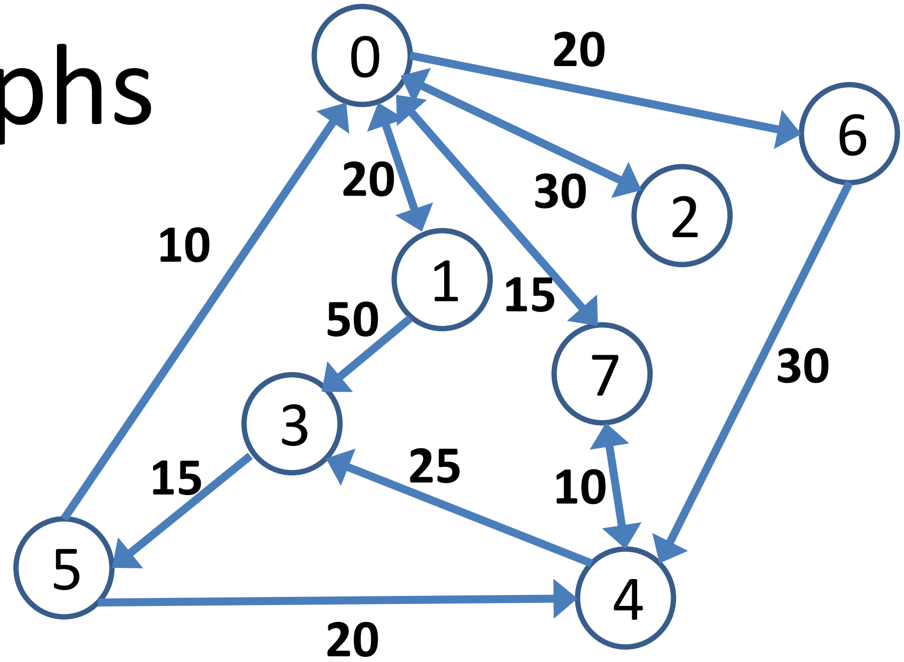
- Suppose that we call Dijkstra's algorithm with source = vertex 1, on graph H (the **reverse graph** of what you see on the right).
- Consider the arrays wt and st we get as a result of that.
- These arrays are related to arrays dist and path on the **original graph G** (what you actually see on the right) as follows:
  - dist[v][1] = wt[v].
  - path[v][1] = st[v].
- Why?

Using Reverse Graphs



- Suppose that we call Dijkstra's algorithm with source = vertex 1, on graph H (the **reverse graph** of what you see on the right).
- Consider the arrays wt and st we get as a result of that.
- wt[v] is the weight of the shortest path from 1 to v in H.
  - Therefore, wt[v] is the weight of the shortest path from v to 1 in G.
  - Therefore, dist[v][1] = wt[v].
- st[v] is the parent of v on the shortest path from 1 to v in H.
  - Therefore, st[v] is the vertex following v on the shortest path from v to 1 in G.
  - Therefore, path[v][1] = st[v].

Using Reverse Graphs



- Suppose that we call Dijkstra's algorithm with source = vertex 1, on graph H (the **reverse graph** of what you see on the right).
- Consider the arrays wt and st we get as a result of that.
- wt[v] is the weight of the shortest path from 1 to v in H.
  - Therefore, wt[v] is the weight of the shortest path from v to 1 in G.
  - Therefore, dist[v][1] = wt[v].
- st[v] is the parent of v on the shortest path from 1 to v in H.
  - Therefore, st[v] is the vertex following v on the shortest path from v to 1 in G.
  - Therefore, path[v][1] = st[v].

Using Dijkstra's Algorithm for All-Pairs Shortest Paths

Input: graph G.

1. Construct reverse graph H.
2. For each s in {0, ..., V-1}:
    3. Call Dijkstra's algorithm on graph H, with source = s.
    4. For each v in {0, ..., V-1}:
        5. dist[v][s] = wt[v].
        6. path[v][s] = st[v].