Single-Source Shortest Paths

CSE 3318 – Algorithms and Data Structures
Alexandra Stefan
University of Texas at Arlington

Shortest Paths

- 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.

- *NOTE: we want the "shortest path" in terms of path weight, NOT number of edges on the path.*
  - *E.g. cheapest flight, not flight with fewest layovers.*

- We will consider two problems:
  - **Single-source**: find the shortest path from the source vertex s to all other vertices in the graph.
    - These shortest paths will form a tree, with s as the root.
  - **All-pairs**: find the shortest paths for all pairs of vertices in the graph.

- Assumptions:
  - Directed graphs
  - Edges do NOT have negative weights.

Discussing the Assumptions

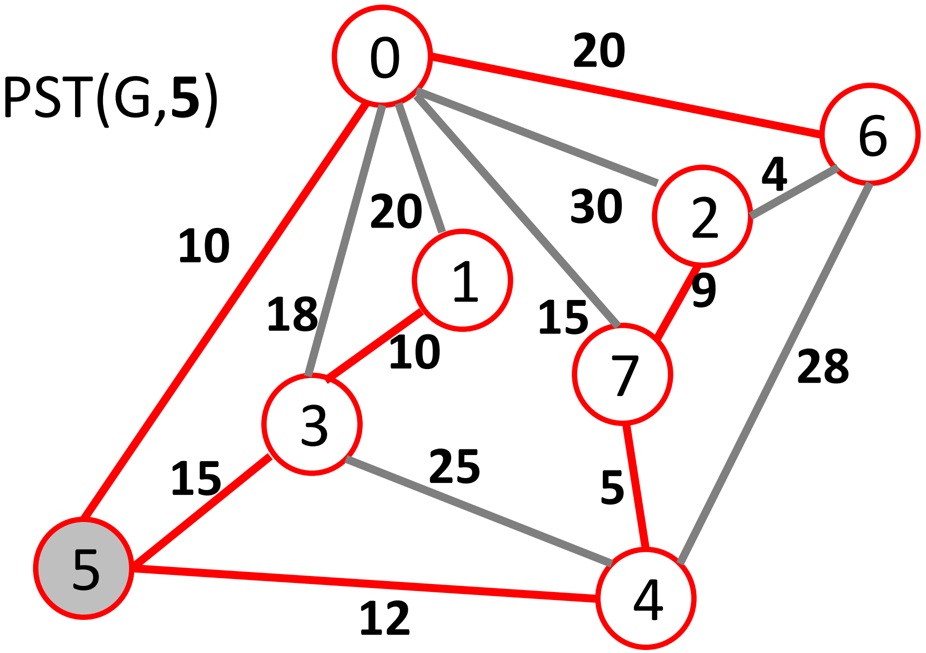
- Can Dijkstra be applied to undirected graphs as well?
  - Yes: Undirected graphs are a special case of directed graphs.

- Negative edge weights are not allowed.
  - The algorithm variation given here will fail to find the shortest path for some

Shortest-Paths Spanning Tree

- Given a directed graph G and a designated vertex s, a **<u>shortest-paths spanning tree</u>** (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. (Here s=5)
  - Each tree path from s to v, is a shortest path in G from s to v.

SPST(G,**5**)



Dijkstra’s Algorithm

```
Dijkstra(G,w,s) // N = |V|
int d[N], p[N]
For v =0 -> N-1
    d[v]=inf    //total weight from s to v
    p[v]=-1     //predecessor of v on path from s to v
d[s]=0
Q = PriorityQueue(d)
While notEmpty(Q)
    u = removeMin(Q,d)
    for each v adjacent to u
        if (d[u]+w(u,v))<d[v]
            p[v]=u
            d[v] = d[u]+w(u,v);  //total weight of path from s to v through u
            decreasedKeyFix(Q,v,d)  //v is neither index nor key
```

Add to the SPST the vertex, u, with the shortest distance.

For each vertex, *v*, record the shortest distance from *s* to it and the edge that connects it (like Prim).

Dijkstra's Algorithm: TC and SC

```
Dijkstra(G,w,s) // N = |V|
1   int d[N], p[N]
2   For v =0 -> N-1        ---------------------> Θ(V)
3       d[v]=inf   //total weight from s to v
4       p[v]=-1    //predecessor of v on path from s to v
5   d[s]=0
6   Q = PriorityQueue(d) ---------------------> Θ(V)
7   While notEmpty(Q) ---------------------> O(V)
8       u = removeMin(Q,d) ---------------------> O(lgV)  --> O(VlgV) (lines 7 and 8)
9       for each v adjacent to u -----------------> O(E) (from lines 7 and 9)
10          if (d[u]+w(u,v))<d[v]
11              p[v]=u
12              d[v] = d[u]+w(u,v);  //total weight of path from s to v through u
13              decreasedKeyFix(Q,v,d)  //v is neither index nor key ---> O(lgV) --> O(ElgV)
                                                  (aggregate from
                                                  both for-loop and
                                                  while-loop
                                                  Lines: 7,9,13)
```

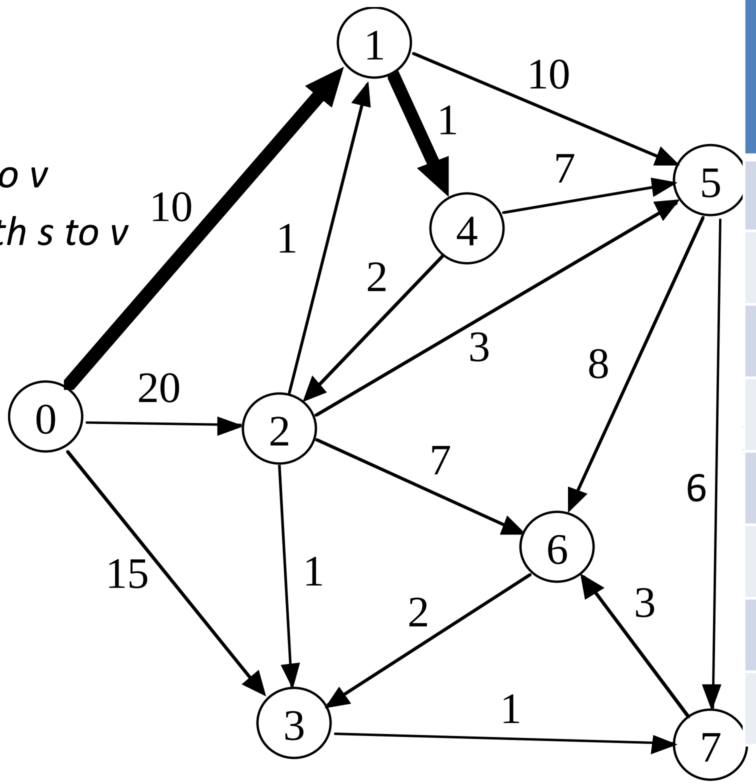
Time complexity: **O(ElgV)**
(for adj list)
O(V + VlgV + E lgV) = O(ElgV)
Assuming V=O(E)
Space complexity: Θ(V)
(for d,p, and Q)

Dijkstra's Algorithm

- Computes an SPST for a graph G and a source s.
- Very similar to Prim's algorithm, but:
  - First vertex to add is the source, s.
  - Works with directed graphs as well, whereas Prim's only works with undirected graphs.
  - Requires edge weights to be non-negative.
  - **It looks at the total path weight, not just the weight of the current edge.**
- Time complexity(same as Prim): O(ElgV) using a heap for the priority-queue and adjacency list for edges.

Dijkstra’s Algorithm: SPST(G,0)

```
Dijkstra(G,w,s) // N = |V|
int d[N], p[N]
For v =0 -> N-1
    d[v]=inf    //total weight from s to v
    p[v]=-1   //v’s predecessor on path s to v
d[s]=0
Q = PriorityQueue(d)
While notEmpty(Q)
    u = removeMin(Q,w)
    for each v adjacent to u
        if (d[u]+w(u,v))<d[v]
            p[v]=u
            d[v] = d[u]+w(u,v);
            decreasedKeyFix(Q,v,d)
```



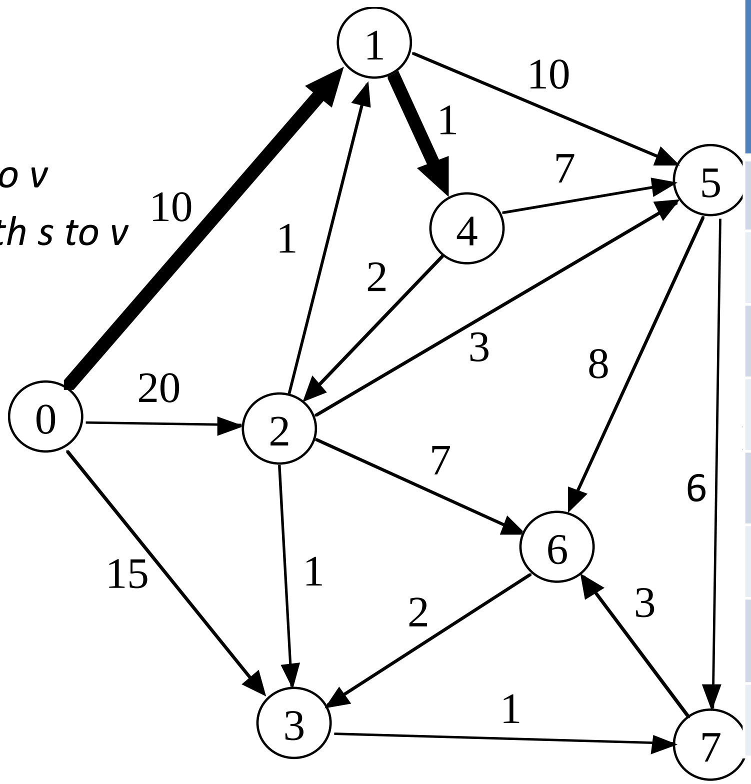
Added Vertex, v	Edge	Dis-tance from s to v

Vertex	0	1	2	3	4	5	6	7
d/p Work (dist and parent updates for nodes)	d[0]/p[0]	d[1]/p[1]	d[2]/p[2]	d[3]/p[3]	d[4]/p[4]	d[5]/p[5]	d[6]/p[6]	d[7]/p[7]

Dist/parent

Dijkstra’s Algorithm: SPST(G,0)

```
Dijkstra(G,w,s) // N = |V|
    int d[N], p[N]
    For v =0 -> N-1
        d[v]=inf    //total weight from s to v
        p[v]=-1   //v’s predecessor on path s to v
    d[s]=0
    Q = PriorityQueue(d)
    While notEmpty(Q)
        u = removeMin(Q,w)
        for each v adjacent to u
            if (d[u]+w(u,v))<d[v]
                p[v]=u
                d[v] = d[u]+w(u,v);
                decreasedKeyFix(Q,v,d)
```



Added Vertex , v	Edge	Dis-tance from s to v
0	-1	0
1	(0,1)	10
4	(1,4)	11
2	(4,2)	13
3	(2,3)	14
7	(3,7)	15
5	(2,5)	16
6	(7,6)	18

Vertex	0	1	2	3	4	5	6	7
d/p Work (dist and parent updates for nodes)	d[0]/p[0] i/-1 0/-1	d[1]/p[1] i/-1 10/0	d[2]/p[2] i/-1 20/0 13/4	d[3]/p[3] i/-1 15/0 14/2	d[4]/p[4] i/-1 11/1	d[5]/p[5] i/-1 20/1 18/4 16/2	d[6]/p[6] i/-1 20/2 18/7	d[7]/p[7] i/-1 15/3

Questions:
- Why not 0->3?

B.c. 14<15
- Why not 0->1->2->3 ?

B.c. no edge 1->2

Shortest path 0 to 7 is recovered in reverse order: 7 <- 3 <- 2 <- 4 <- 1 <- 0 , path: 0,1,4,2,3,7

Applications

- See:

http://www.csl.mtu.edu/cs2321/www/newLectures/30_More_Dijkstra.htm

and the robot navigation

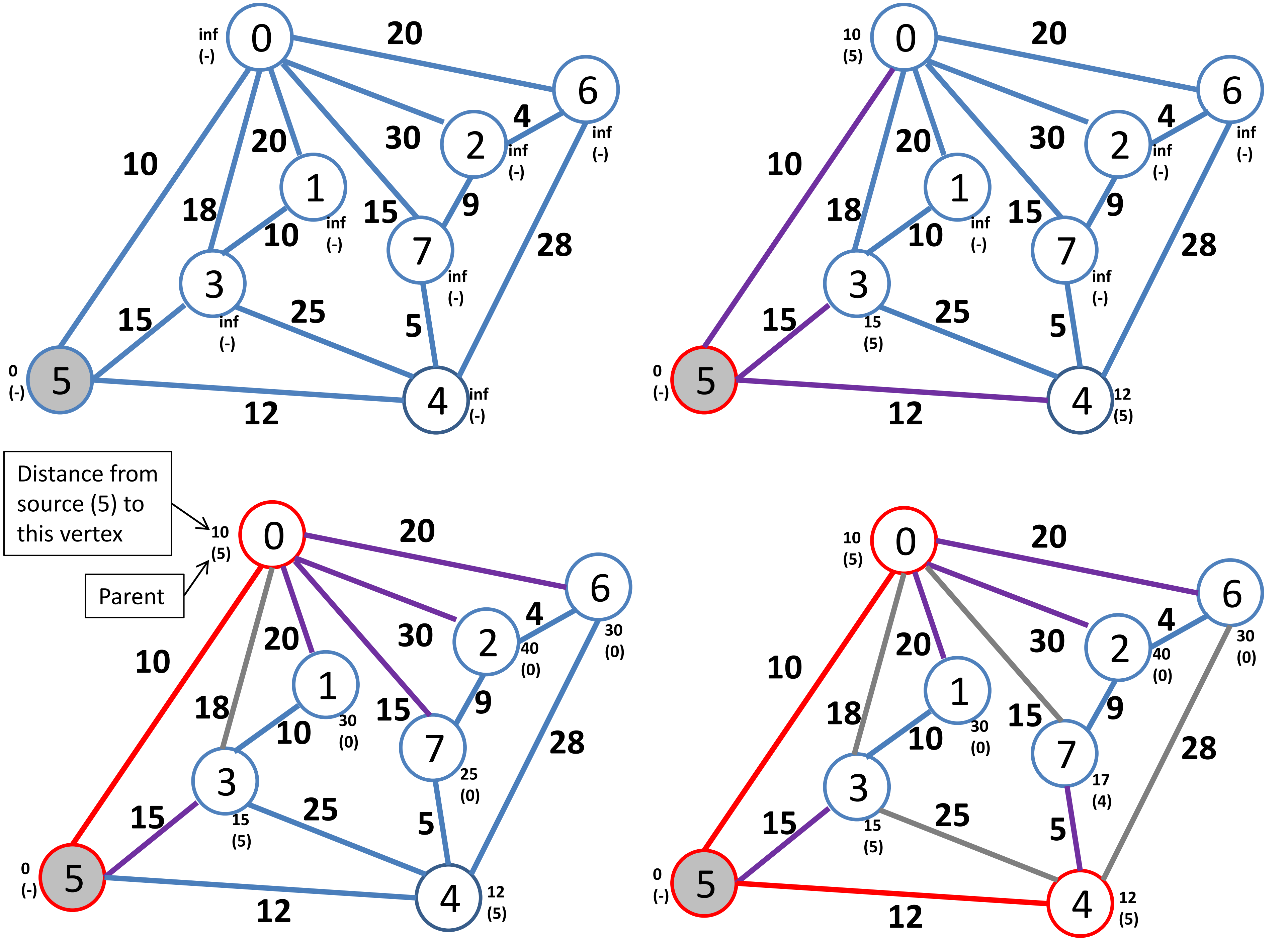
https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm

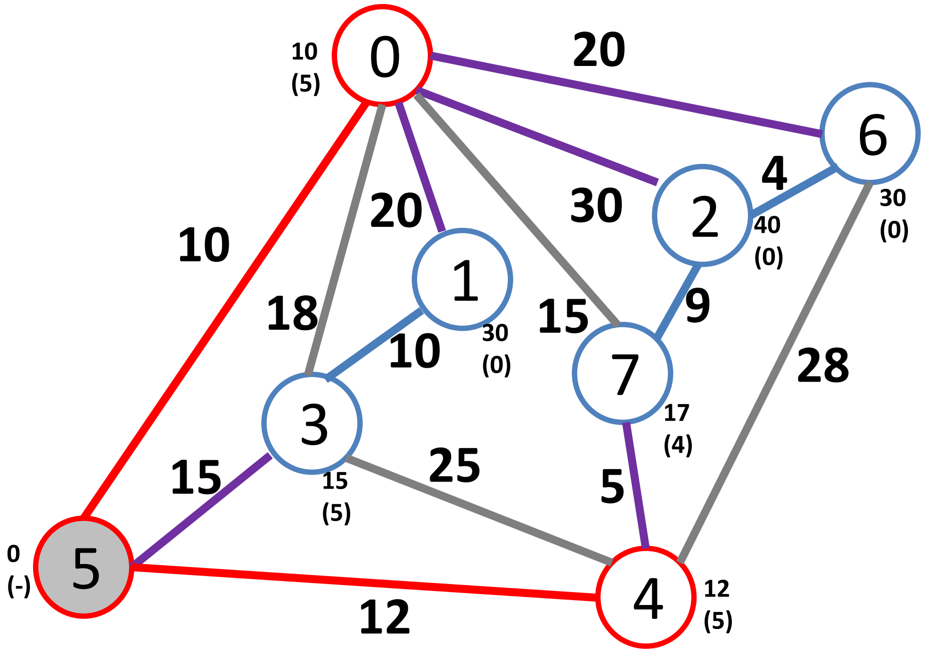
All-Pairs Shortest Paths

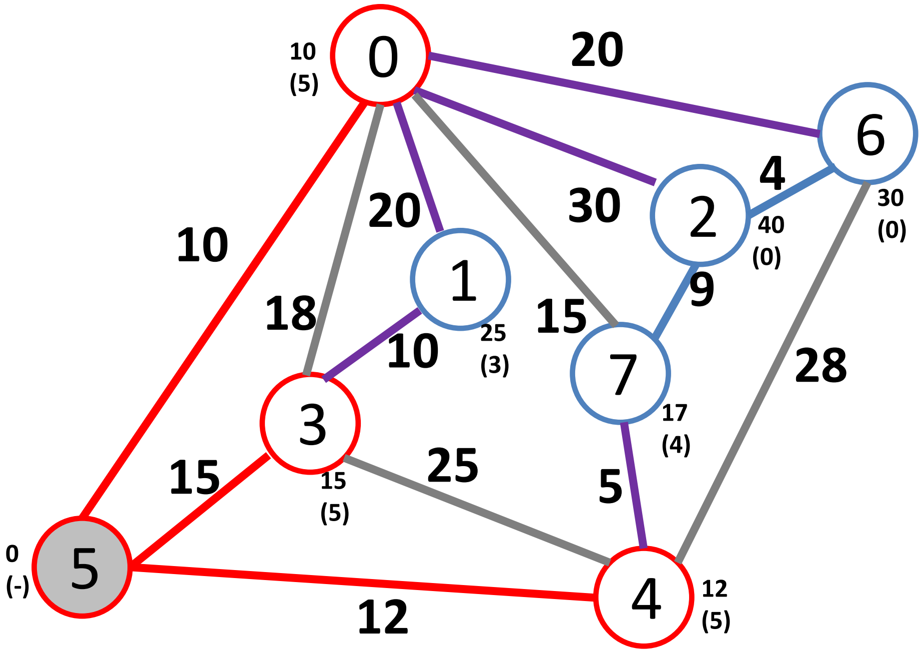
- Run Dijkstra to compute the Shortest Path Spanning Tree (SPST) for each vertex used as source.
  - Note that the array of predecessors completely specifies the SPST.

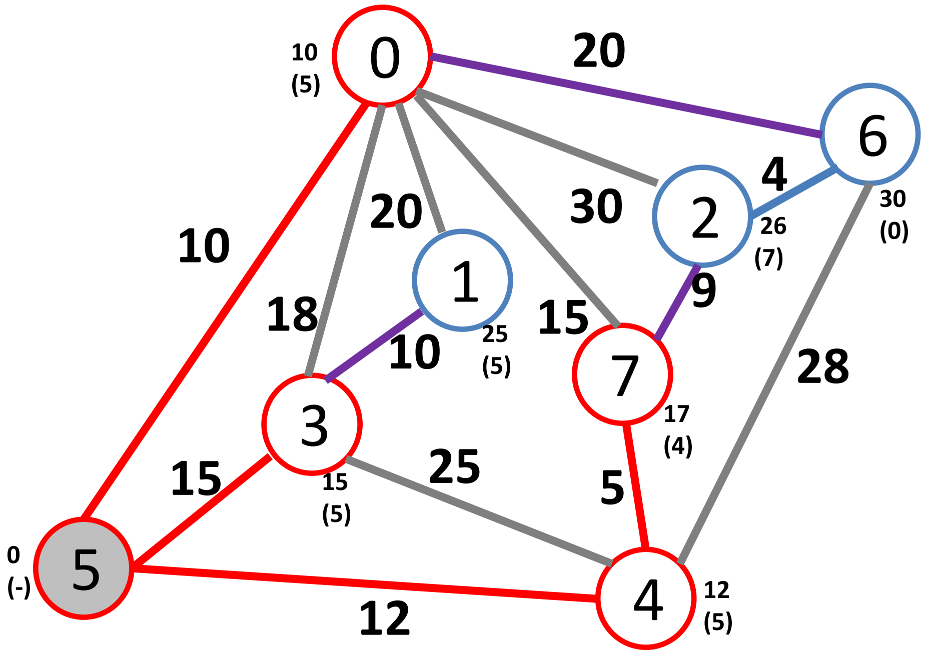
Worked-out (SPST) Dijkstra example

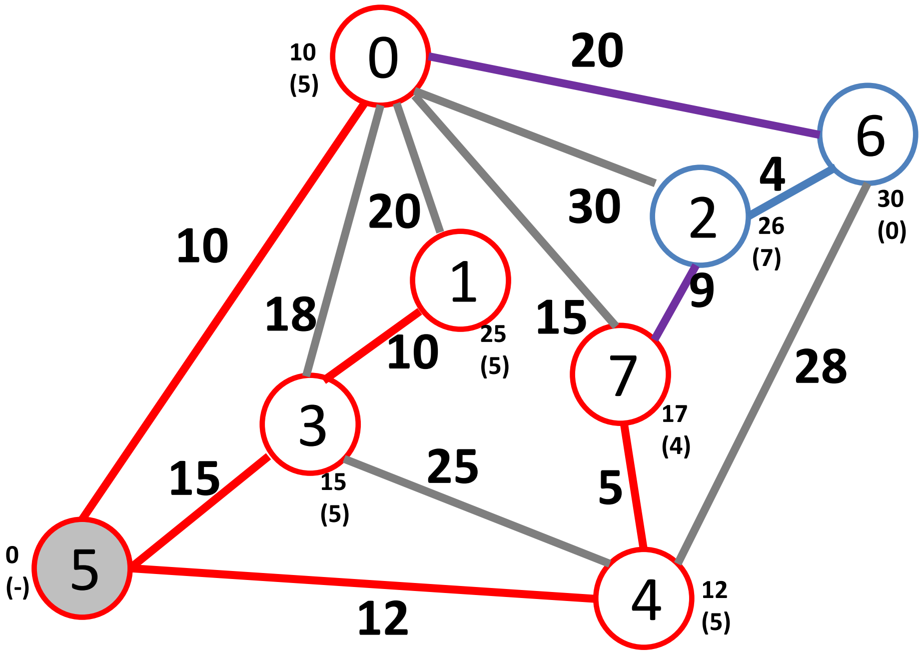
- Note that this example is for an undirected graph. The same algorithm will be applied to a directed graph (going in the direction of the arrows).
- When moving to a new page, the last state of the graph (bottom right) is copied first (top left).
- Purple edges – (u,v) best edge discovered so far to connect v to the tree (u is already in the tree). Edges between nodes in current SPST and nodes outside of it.
- Gray edges – edges discovered that do not provide a shorter path to the vertex (discovered, but not used).
- Red edges and vertices – shortest path spanning tree (SPST) built with Dijkstra.

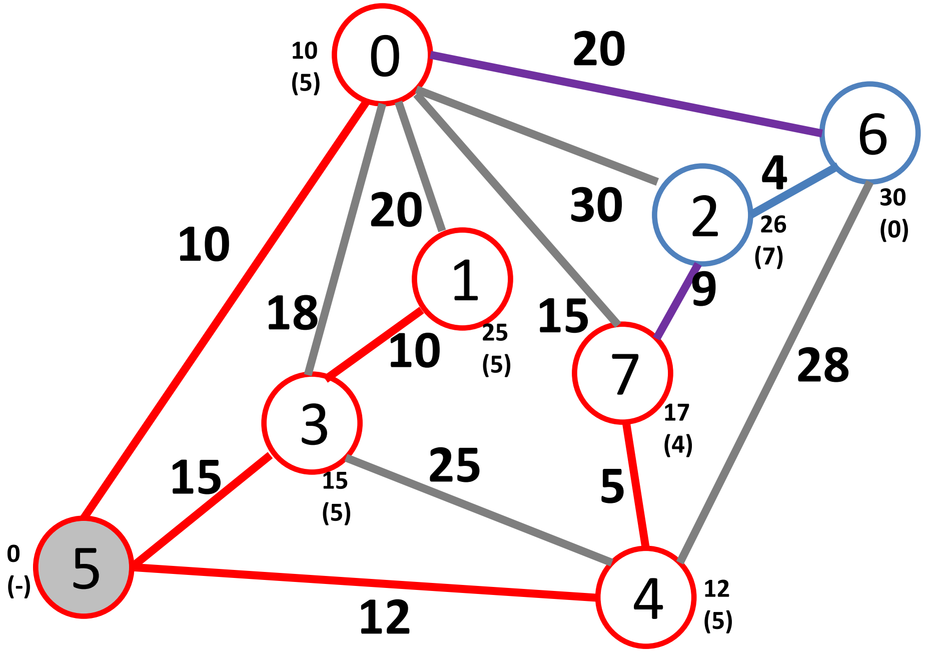

Distance from source (5) to this vertex
Parent

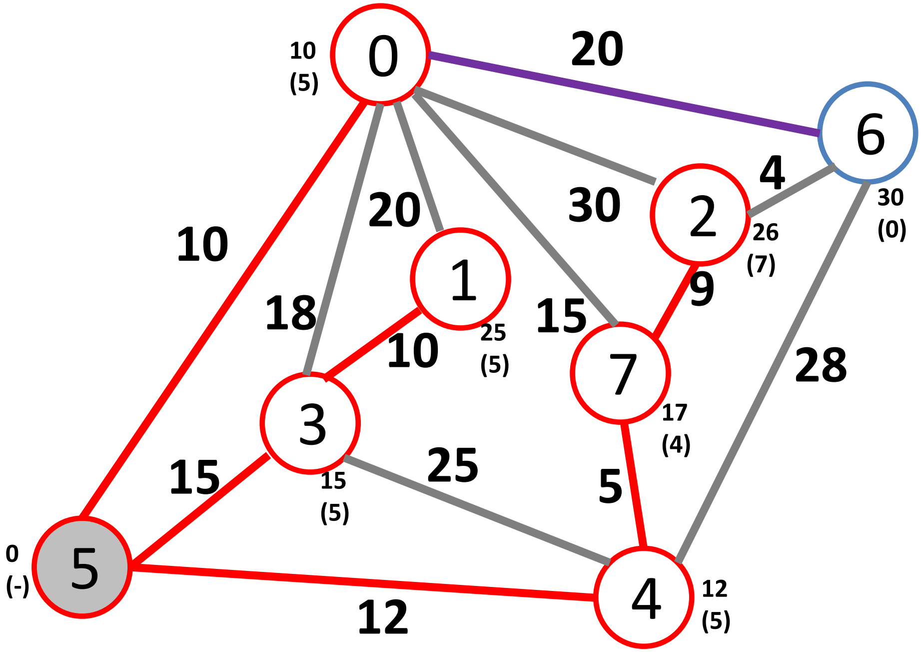

0
10
(5)
20
6
30
(0)
2
4
40
(0)
10
20
30
1
30
(0)
18
10
15
9
7
17
(4)
28
3
15
(5)
25
15
5
5
0
(-)
4
12
(5)
12

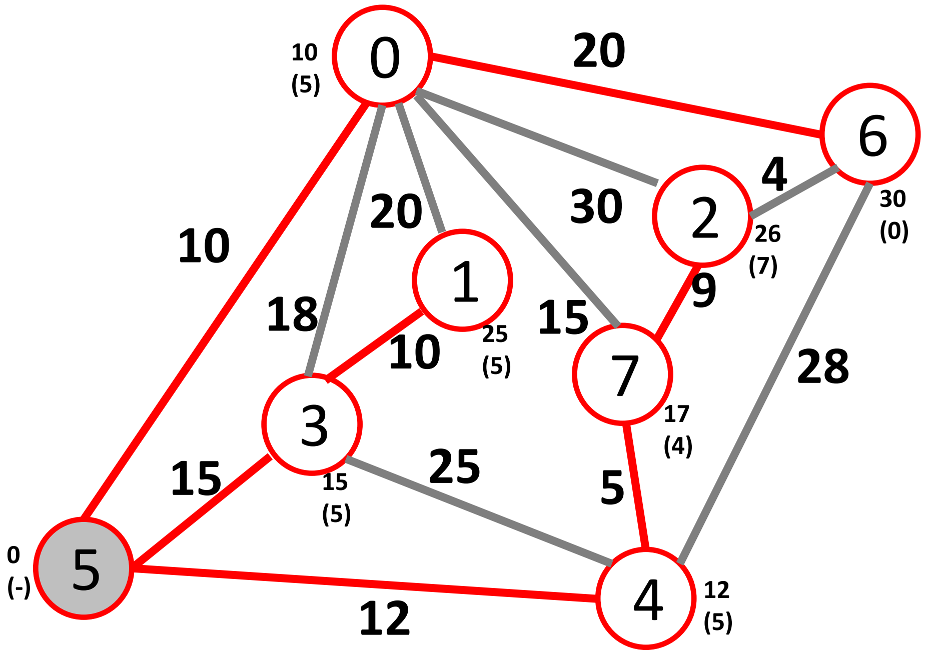

0
10
(5)
20
6
30
(0)
2
4
40
(0)
10
20
30
1
25
(3)
18
10
15
9
7
17
(4)
28
3
15
(5)
25
15
5
5
0
(-)
4
12
(5)
12


0
10
(5)
20
6
30
(0)
2
4
26
(7)
10
20
30
1
25
(5)
18
10
15
9
7
17
(4)
28
3
15
(5)
25
15
5
5
0
(-)
4
12
(5)
12


0
10
(5)
20
6
30
(0)
2
4
26
(7)
10
20
30
1
25
(5)
18
10
15
9
7
17
(4)
28
3
15
(5)
25
15
5
5
0
(-)
4
12
(5)
12







Vertex, as added	Edge (parent,vertex)	Distance from source to vertex
5	-	0
0	(5,0)	10
4	(5,4)	12
3	(5,3)	15
7	(4,7)	17
1	(3,1)	25
2	(7,2)	26
6	(0,6)	30

Dijkstra’s Algorithm

```
Dijkstra(G,w,s) // N = |V|
int d[N], p[N]
For v =0 -> N-1
    d[v]=inf    //total weight from s to v
    p[v]=-1   //v’s predecessor on path s to v
d[s]=0
Q = PriorityQueue(d)
While notEmpty(Q)
    u = removeMin(Q,w)
    for each v adjacent to u
        if (d[u]+w(u,v))<d[v]
            p[v]=u
            d[v] = d[u]+w(u,v);
            decreasedKeyFix(Q,v,d)
```

- Find the SPST(G,5).

Show the distance and the parent for each vertex.