Refined version of Prim's Minimum Spanning Tree Algorithm

Input: a graph G = (V, E) with non-negative weights

Output: a minimum spanning tree with the starting vertex s as the root

```
MST getMinimumSpanningTree(s) {
  Let T be a set that contains the vertices in the spanning tree;
  Initially T is empty;
  Set cost[s] = 0; and cost[v] = infinity for all other vertices in V;

  while (size of T < n) {
    Find u not in T with the smallest cost[u];
    Add u to T;
    for each v not in T and (u, v) in E
      if (cost[v] > w(u, v)) { cost[v] = w(u, v); parent[v] = u; } // end if
  } // end while
} // end getMinimumSpanningTree
```

```java
public MST getMinimumSpanningTree(int startingVertex) {
  double[] cost = new double[getSize()];
  for (int i = 0; i < cost.length; i++)
    cost[i] = Double.POSITIVE_INFINITY;
  cost[startingVertex] = 0;

  int[] parent = new int[getSize()];
  parent[startingVertex] = -1;
  double totalWeight = 0;

  List<Integer> T = new ArrayList<>();
```

```java
  while (T.size() < getSize()) {
    int u = -1;
    double currentMinCost = Double.POSITIVE_INFINITY;
    for (int i = 0; i < getSize(); i++) {
      if (!T.contains(i) && cost[i] < currentMinCost) {
        currentMinCost = cost[i];
        u = i;
      }
    }

    if (u == -1) break; else T.add(u);
    totalWeight += cost[u];

    for (Edge e : neighbors.get(u)) {
      if (!T.contains(e.v) &&
          cost[e.v] > ((WeightedEdge)e).weight) {
        cost[e.v] = ((WeightedEdge)e).weight;
        parent[e.v] = u;
      }
    }
  }

  return new MST(startingVertex, parent, T, totalWeight);
}
```

Finding shortest paths

- The shortest path between two vertices is a path with the minimum total weights.

- Given a graph with nonnegative weights on the edges, a well-known algorithm for finding a shortest path between two vertices was discovered by Edsger Dijkstra, a Dutch computer scientist.

- In order to find a shortest path from vertex s to vertex v, *Dijkstra's algorithm* finds the shortest path from s to all vertices.
- So Dijkstra's algorithm is known as a *single-source shortest-path* algorithm.

Edsger Wybe Dijkstra

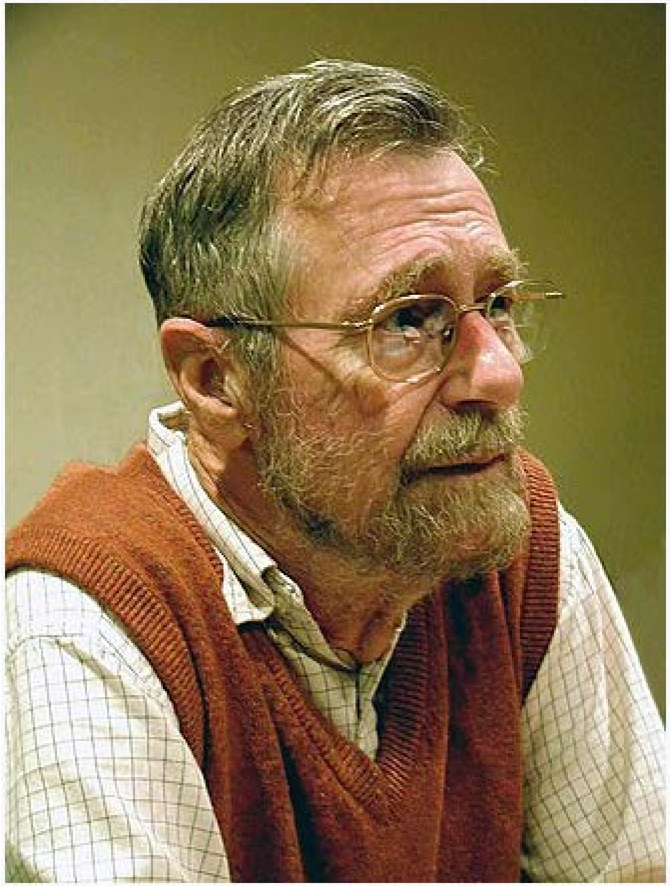


Image Source:
https://en.wikipedia.org/wiki/Edsger_W._Dijkstra

Dijkstra’s Single Source Shortest Path Algorithm

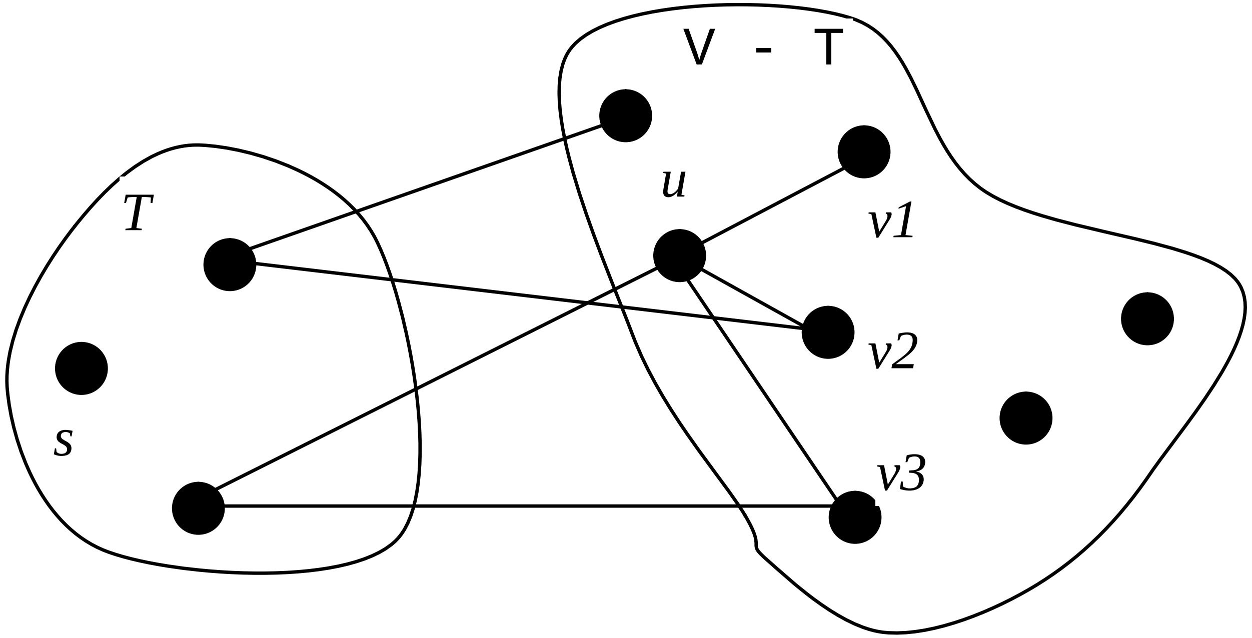
Input: a graph G = (V, E) with nonnegative weights

Output: a shortest-path tree with the source vertex s as the root

```
ShortestPathTree getShortestPath(s) {
  Let T be a set that contains the vertices whose paths to s are known;
  Initially T is empty;
  Set cost[s] = 0; and cost[v] = infinity for all other vertices in V;

  while (size of T < n) {
    Find u not in T with the smallest cost[u];
    Add u to T;
    for each v not in T and (u, v) in E
      if (cost[v] > cost[u] + w(u, v)) { cost[v] = cost[u] + w(u, v); parent[v] = u; } // end if
  } // end while
} // end getShortestPath
```

Dijkstra’s Single Source Shortest Path Algorithm

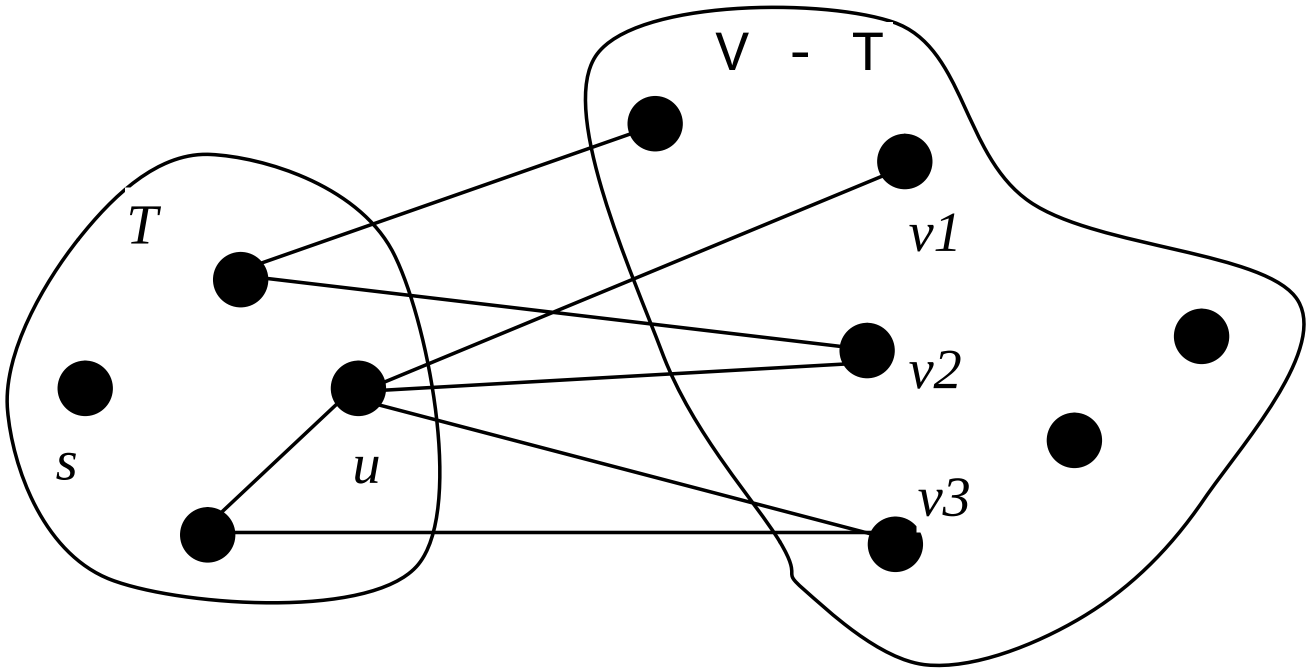


T contains vertices whose shortest path to s are known

V - T contains vertices whose shortest path to s are not known yet

Before moving u to T

Dijkstra’s Single Source Shortest Path Algorithm

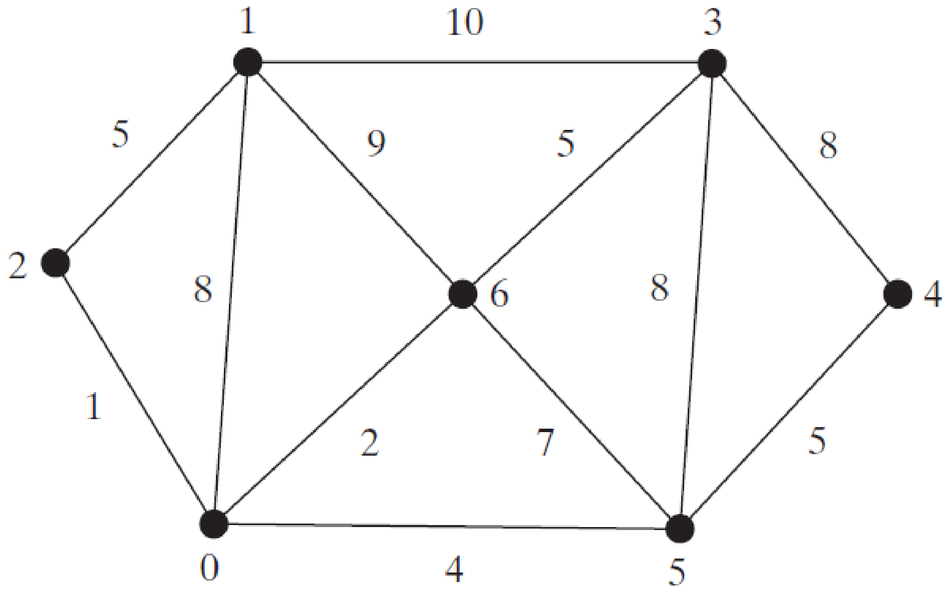


T contains vertices whose shortest path to s are known

V - T contains vertices whose shortest path to s are not known yet

After moving u to T

Example: Step 0



(a)

cost

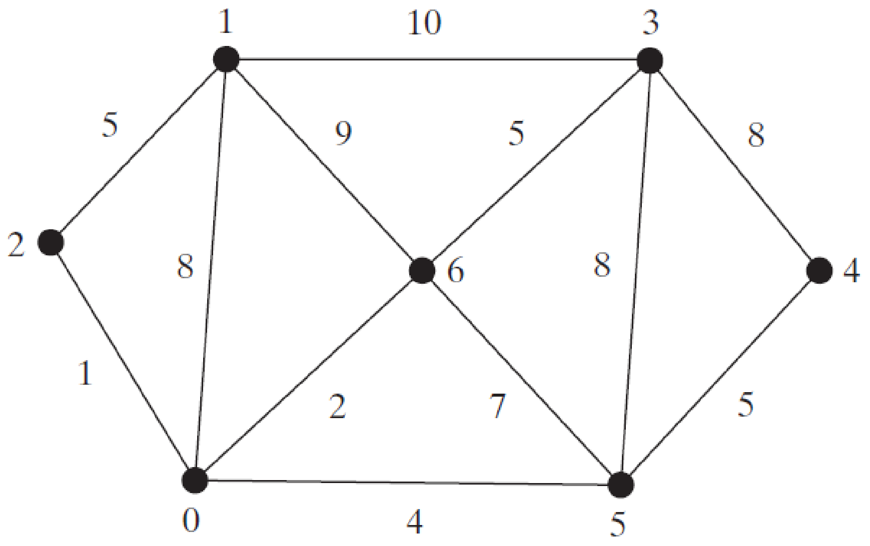
0	1	2	3	4	5	6
∞	0	∞	∞	∞	∞	∞

parent

0	1	2	3	4	5	6
	–1					

(b)

Example: Step 1



(a)

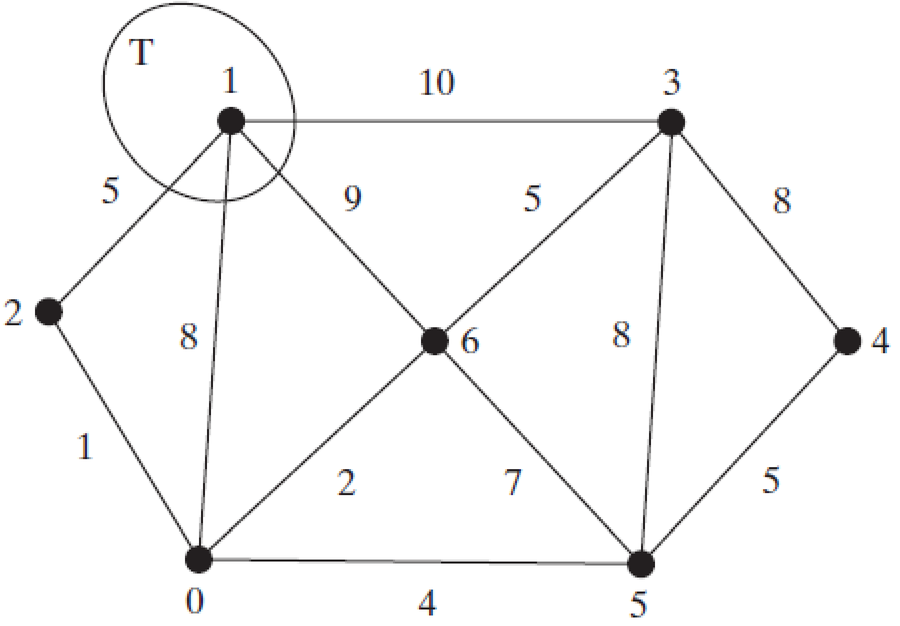
cost

∞	0	∞	∞	∞	∞	∞
0	1	2	3	4	5	6

parent

	–1					
0	1	2	3	4	5	6

(b)



(a)

cost

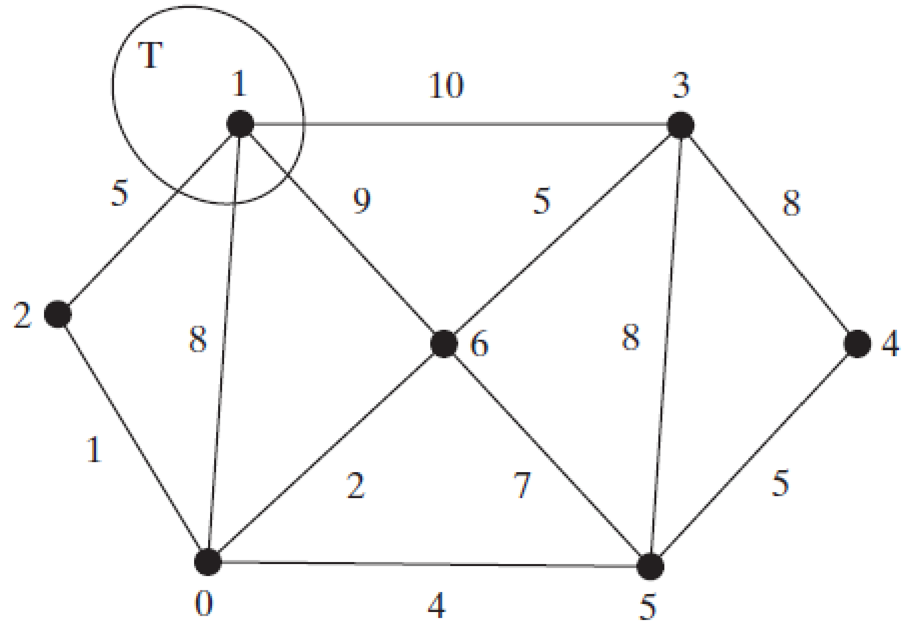
8	0	5	10	∞	∞	9
0	1	2	3	4	5	6

parent

1	–1	1	1			1
0	1	2	3	4	5	6

(b)

Example: Step 2



(a)

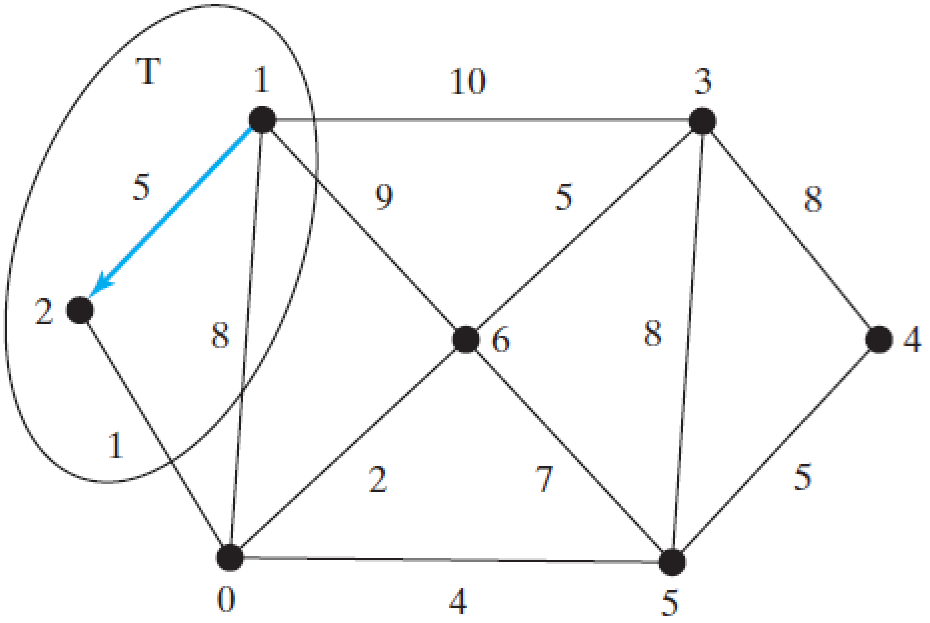
cost

8	0	5	10	∞	∞	9
0	1	2	3	4	5	6

parent

1	–1	1	1			1
0	1	2	3	4	5	6

(b)



(a)

cost

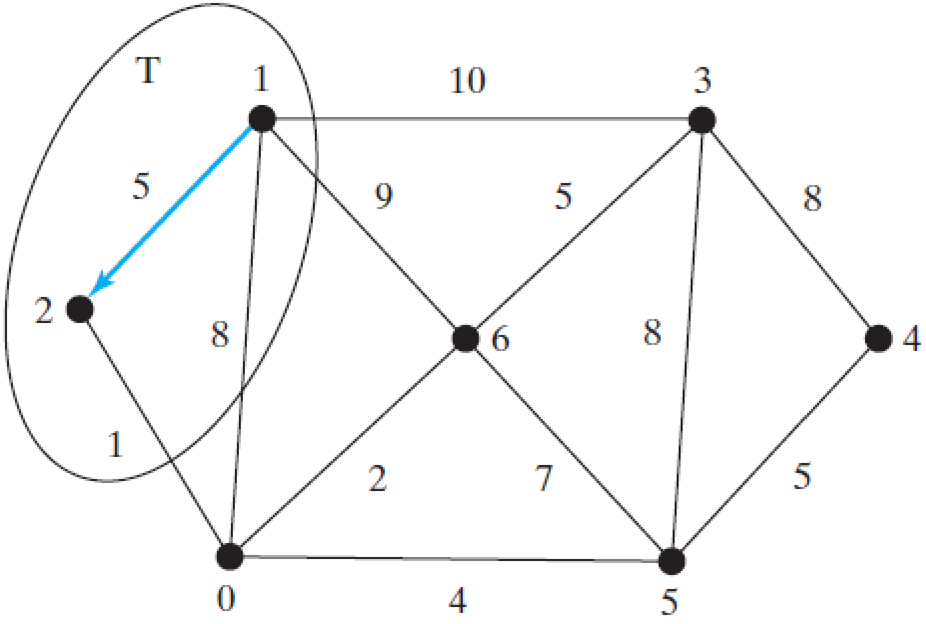
6	0	5	10	∞	∞	9
0	1	2	3	4	5	6

parent

2	–1	1	1			1
0	1	2	3	4	5	6

(b)

Example: Step 3



(a)

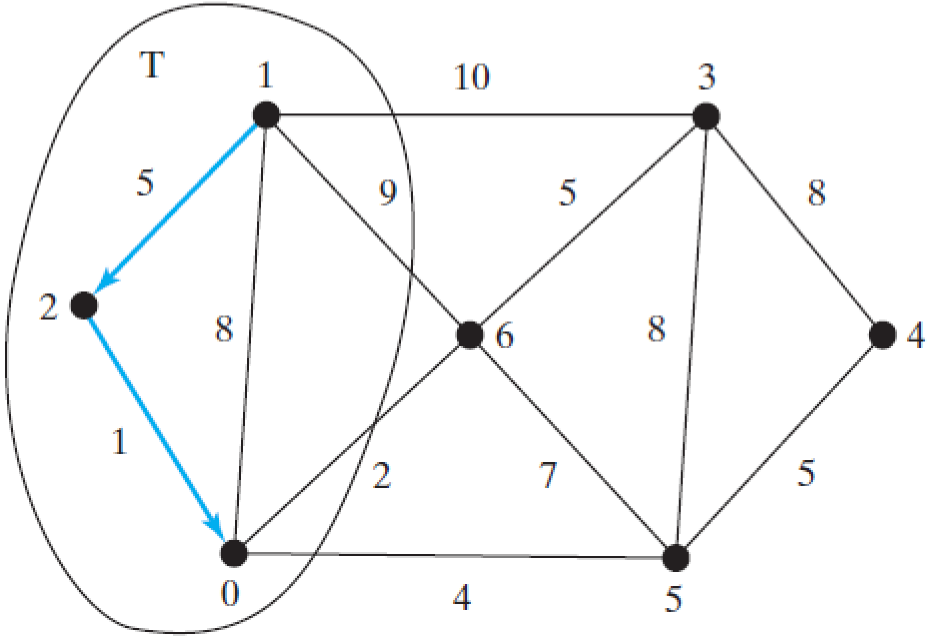
cost

6	0	5	10	∞	∞	9
0	1	2	3	4	5	6

parent

2	–1	1	1			1
0	1	2	3	4	5	6

(b)



(a)

cost

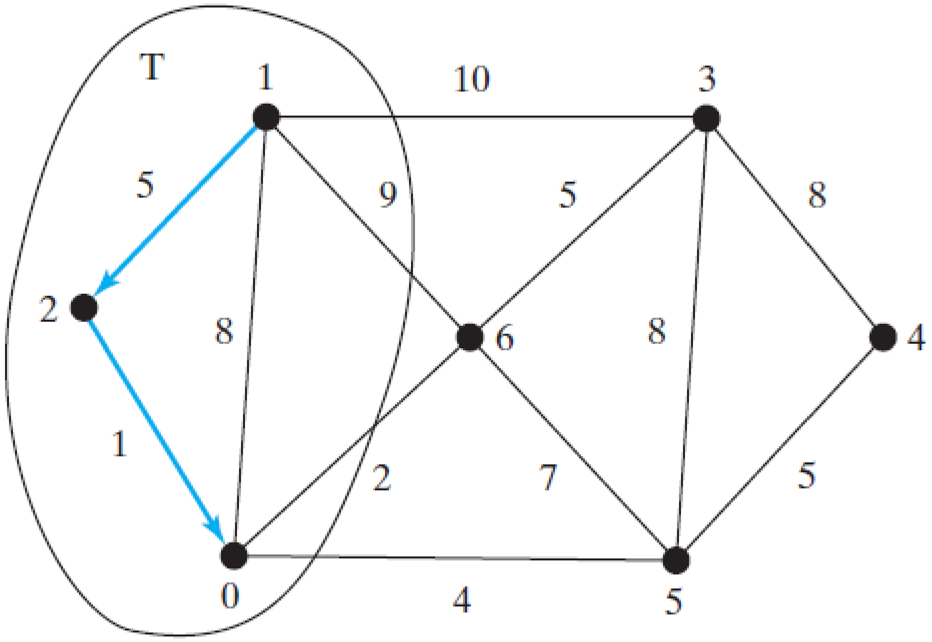
6	0	5	10	∞	10	8
0	1	2	3	4	5	6

parent

2	–1	1	1		0	0
0	1	2	3	4	5	6

(b)

Example: Step 4



(a)

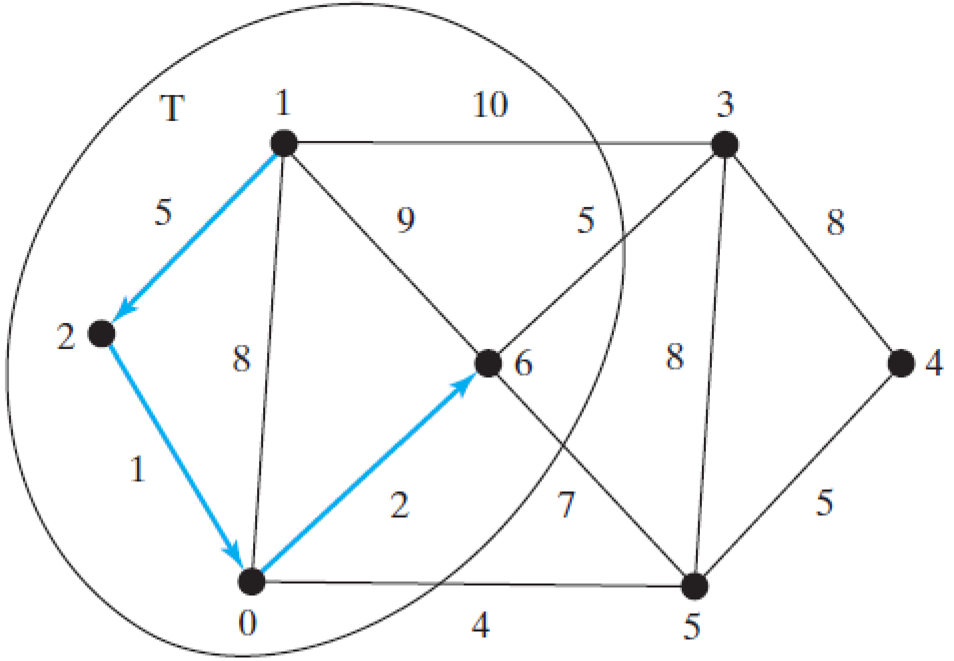
cost

6	0	5	10	∞	10	8
0	1	2	3	4	5	6

parent

2	–1	1	1		0	0
0	1	2	3	4	5	6

(b)



(a)

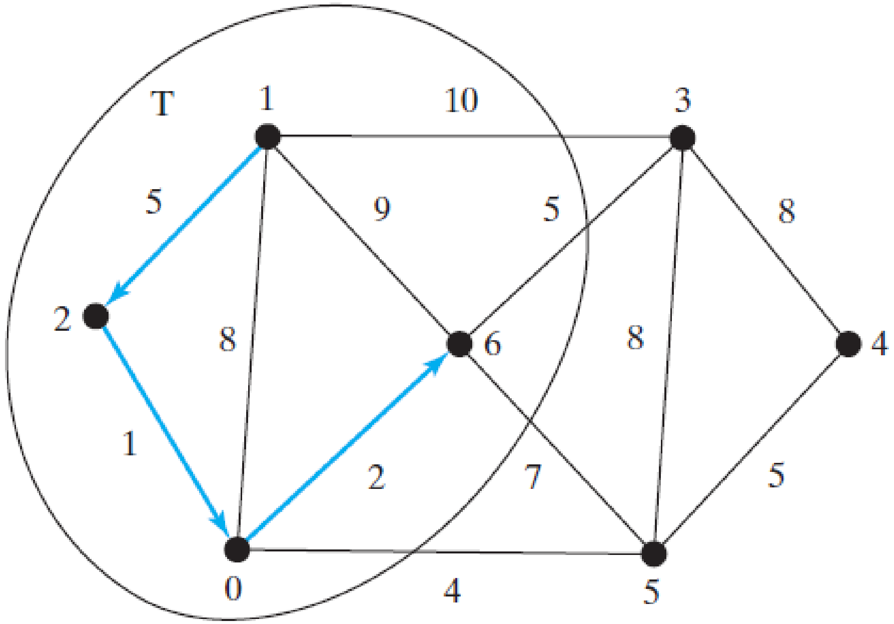
cost

6	0	5	10	∞	10	8
0	1	2	3	4	5	6

parent

2	–1	1	1		0	0
0	1	2	3	4	5	6

(b)



(a)

cost

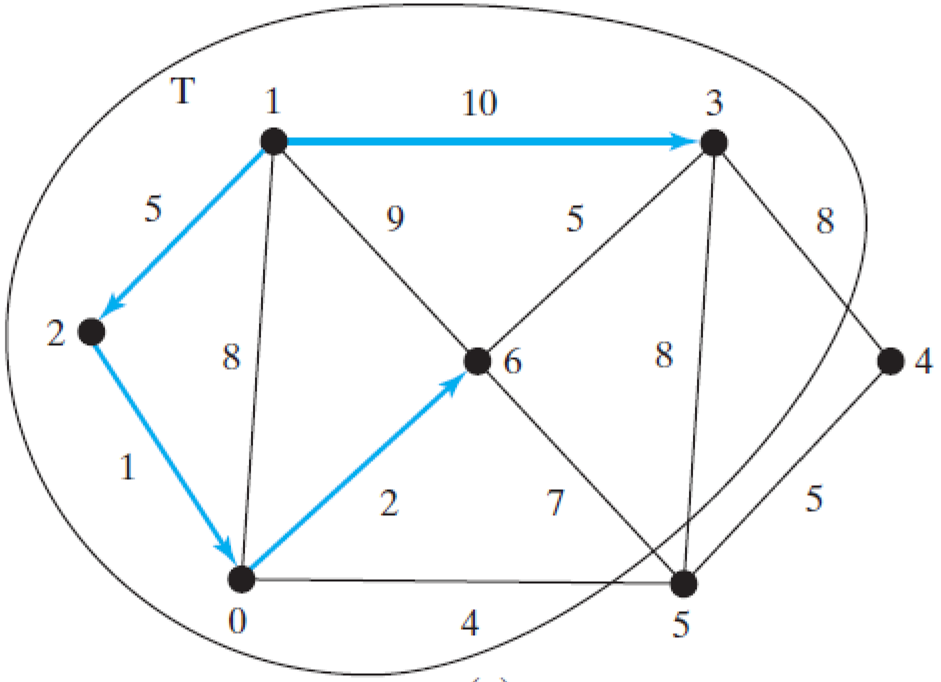
0	1	2	3	4	5	6
6	0	5	10	∞	10	8

parent

0	1	2	3	4	5	6
2	–1	1	1		0	0

(b)

Example: Step 5



(a)

cost

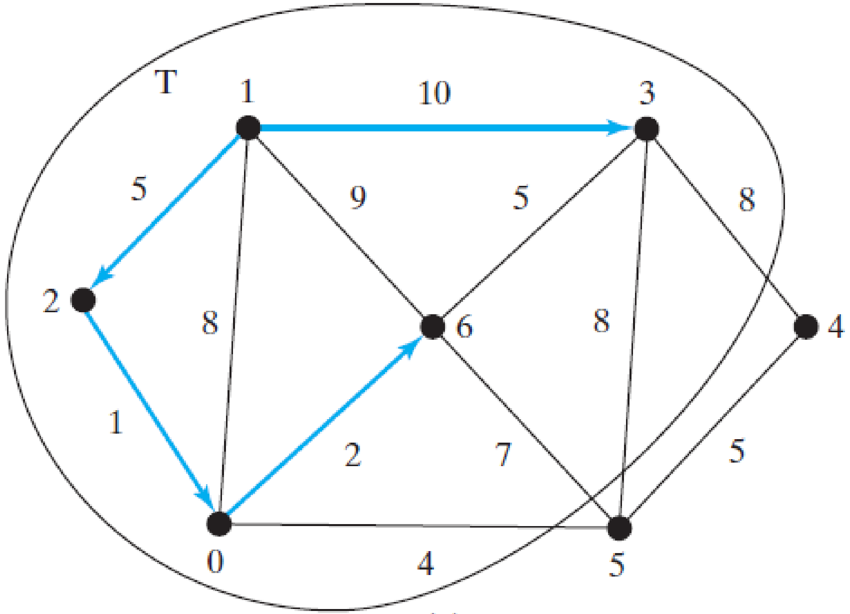
0	1	2	3	4	5	6
6	0	5	10	18	10	8

parent

0	1	2	3	4	5	6
2	–1	1	1	3	0	0

(b)

Example: Step 6



(a)

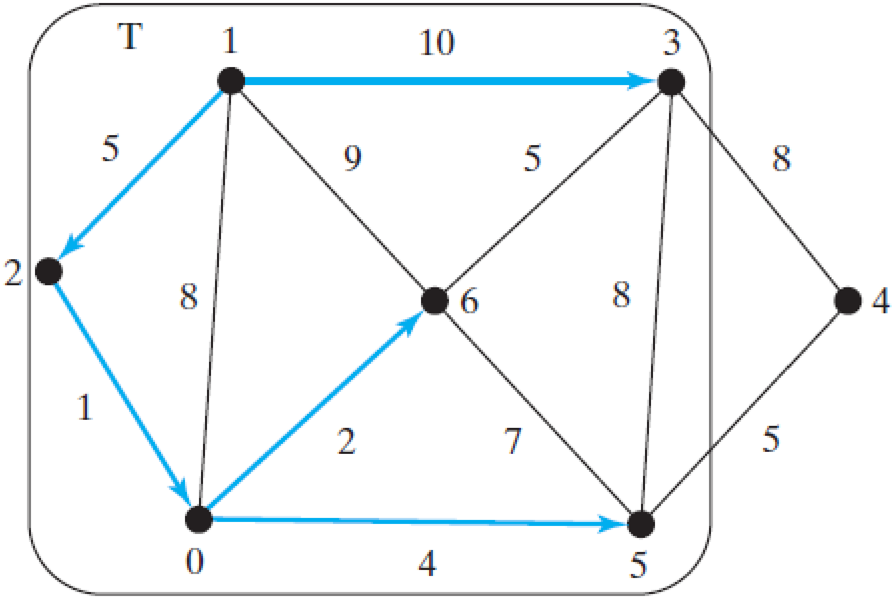
cost

6	0	5	10	18	10	8
0	1	2	3	4	5	6

parent

2	–1	1	1	3	0	0
0	1	2	3	4	5	6

(b)



(a)

cost

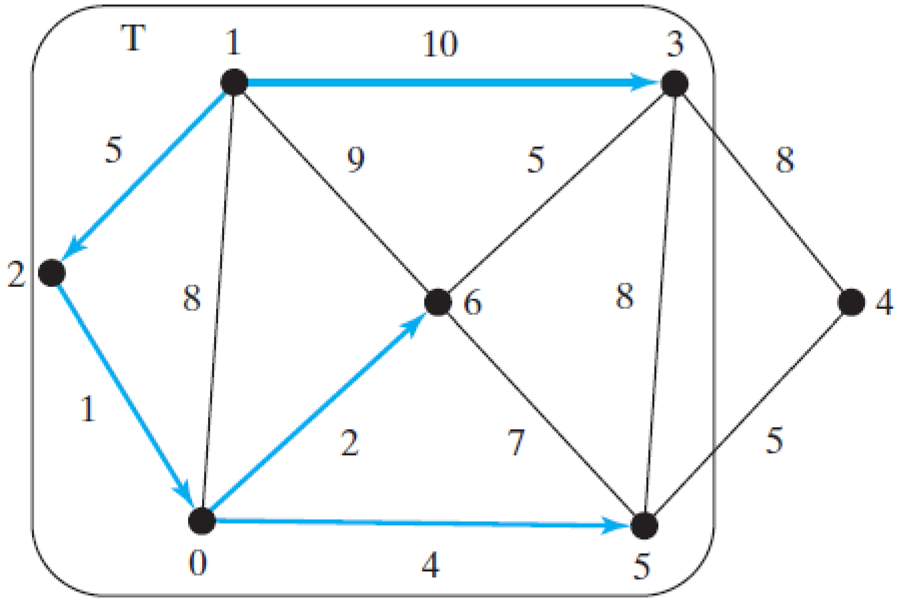
6	0	5	10	15	10	8
0	1	2	3	4	5	6

parent

2	–1	1	1	5	0	0
0	1	2	3	4	5	6

(b)

Example: Step 7



(a)

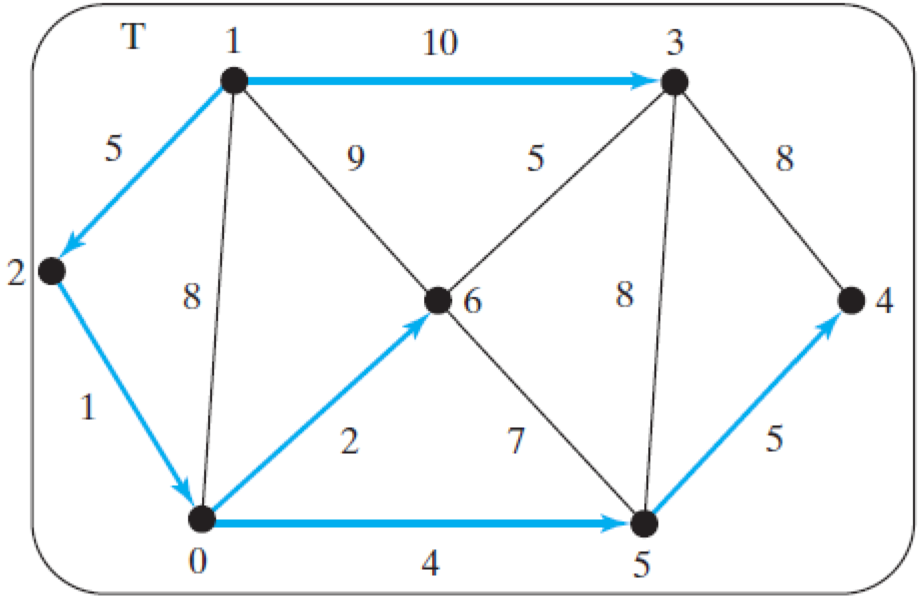
cost

6	0	5	10	15	10	8
0	1	2	3	4	5	6

parent

2	–1	1	1	5	0	0
0	1	2	3	4	5	6

(b)



(a)

cost

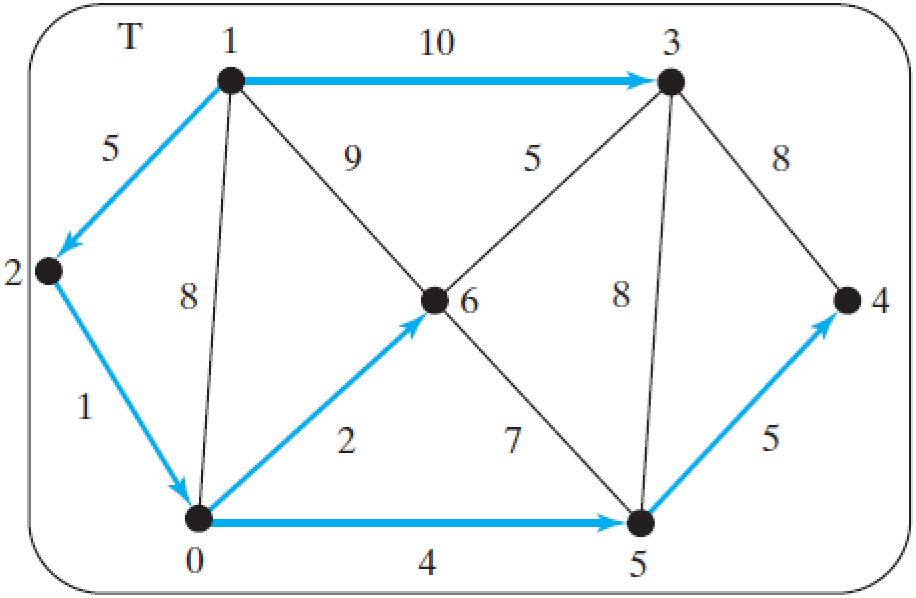
6	0	5	10	15	10	8
0	1	2	3	4	5	6

parent

2	–1	1	1	5	0	0
0	1	2	3	4	5	6

(b)

Example: Step 8



(a)

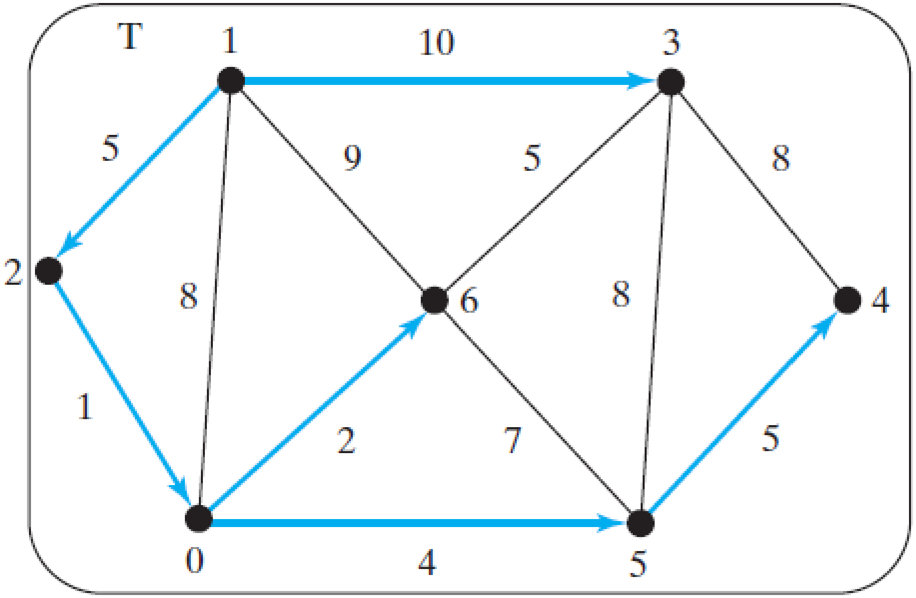
cost

6	0	5	10	15	10	8
0	1	2	3	4	5	6

parent

2	–1	1	1	5	0	0
0	1	2	3	4	5	6

(b)


(a)

cost

6	0	5	10	15	10	8
0	1	2	3	4	5	6

parent

2	–1	1	1	5	0	0
0	1	2	3	4	5	6

(b)

```java
public ShortestPathTree getShortestPath(int sourceVertex) {
  double[] cost = new double[getSize()];
  for (int i = 0; i < cost.length; i++) {
    cost[i] = Double.POSITIVE_INFINITY;
  }
  cost[sourceVertex] = 0;

  int[] parent = new int[getSize()];
  parent[sourceVertex] = -1;

  List<Integer> T = new ArrayList<>();

  while (T.size() < getSize()) {
    int u = -1;
    double currentMinCost = Double.POSITIVE_INFINITY;
    for (int i = 0; i < getSize(); i++) {
      if (!T.contains(i) && cost[i] < currentMinCost) {
        currentMinCost = cost[i];
        u = i;
      }
    }

    if (u == -1) break; else T.add(u);

    for (Edge e : neighbors.get(u)) {
      if (!T.contains(e.v)
          && cost[e.v] > cost[u] + ((WeightedEdge)e).weight) {
        cost[e.v] = cost[u] + ((WeightedEdge)e).weight;
        parent[e.v] = u;
      }
    }
  }

  return new ShortestPathTree(sourceVertex, parent, T, cost);
}
```

References

- Y. D. Liang, "Introduction to Java Programming and Data Structures," Comprehensive version, 11th ed. Pearson Education, Inc., 2018.