Dijkstra's Algorithm

- If no negative edge weights, we can beat Bellman-Ford Algorithm
- Similar to breadth-first search
  - Grow a tree gradually, advancing from vertices taken from a queue
- Also similar to Prim's algorithm for MST
  - Use a priority queue keyed on d[v]

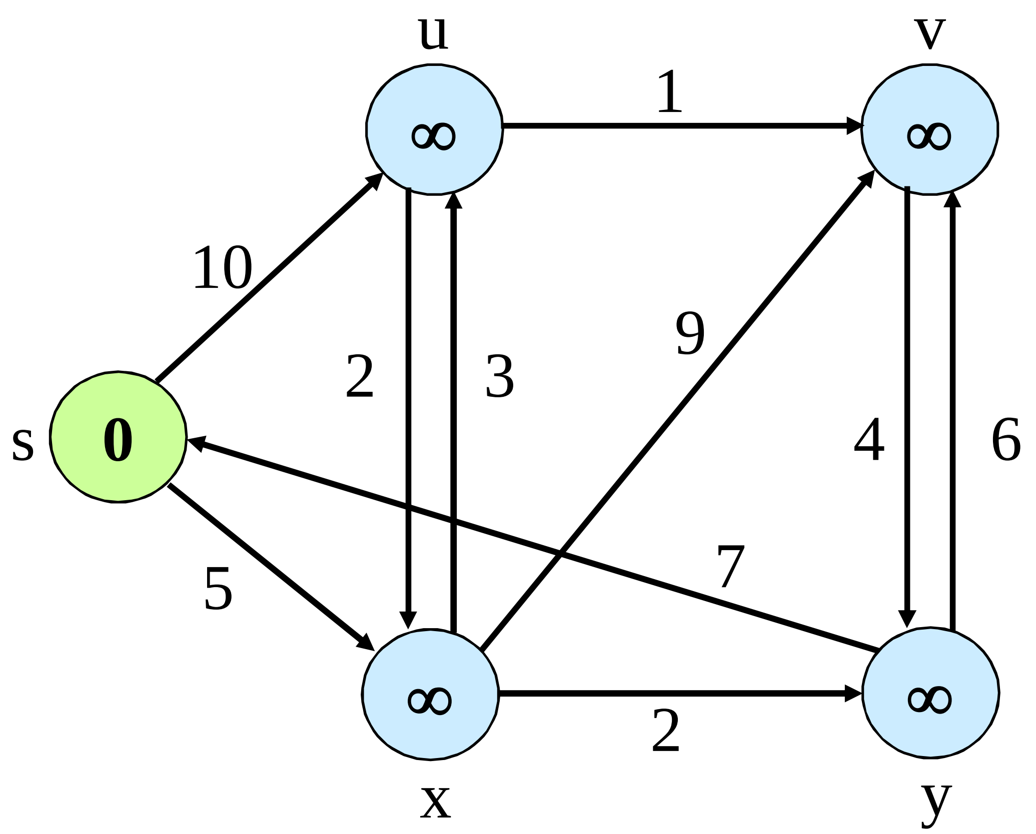
Dijkstra's Algorithm

```
Dijkstra(G)
   for each v ∈ V
      d[v] = ∞;
   d[s] = 0; S = ∅; Q = V;

   while (Q ≠ ∅)
      u = ExtractMin(Q);
      S = S U {u};

      for each v ∈ u->Adj[]
         if (d[v] > d[u]+w(u,v))
            d[v] = d[u]+w(u,v);
```

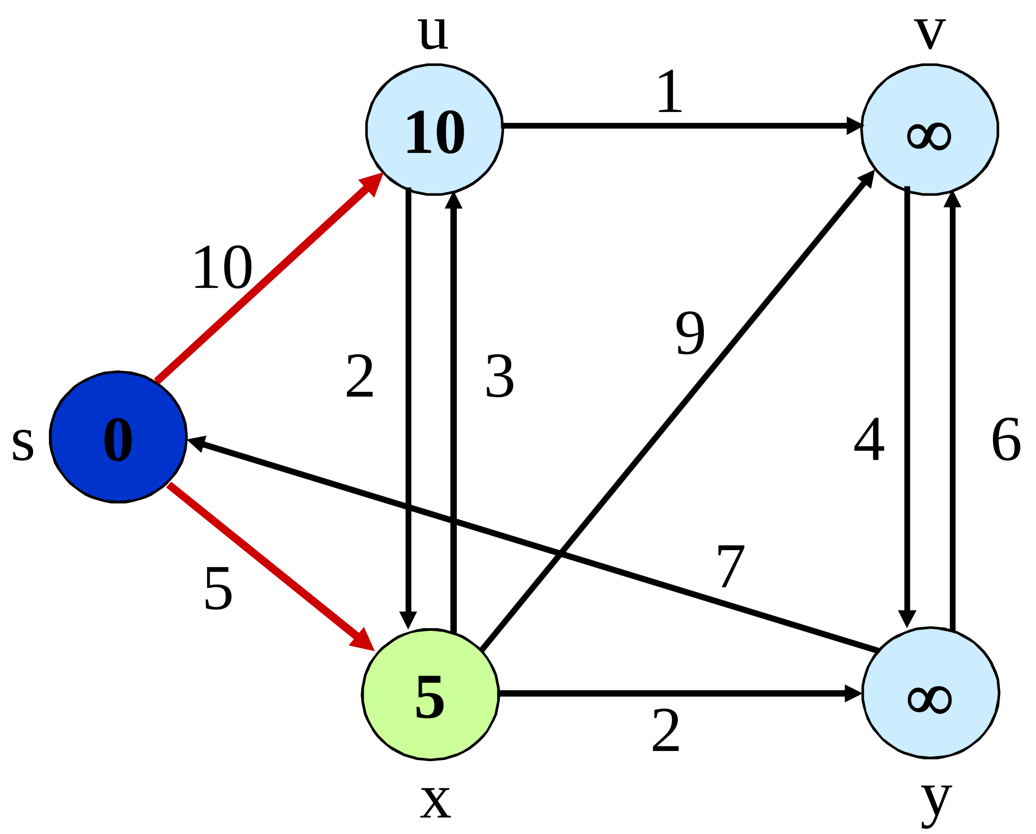
Relaxation Step

Note: this is really a call to Q->DecreaseKey()

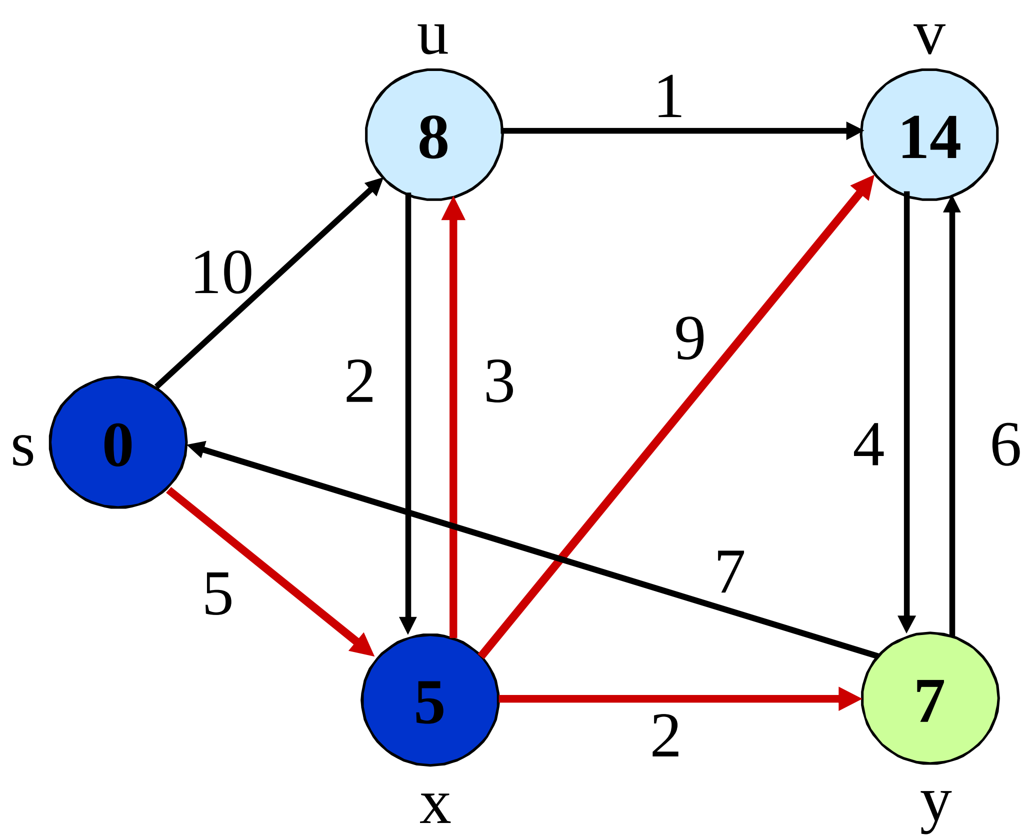
Dijkstra’s Algorithm: Example



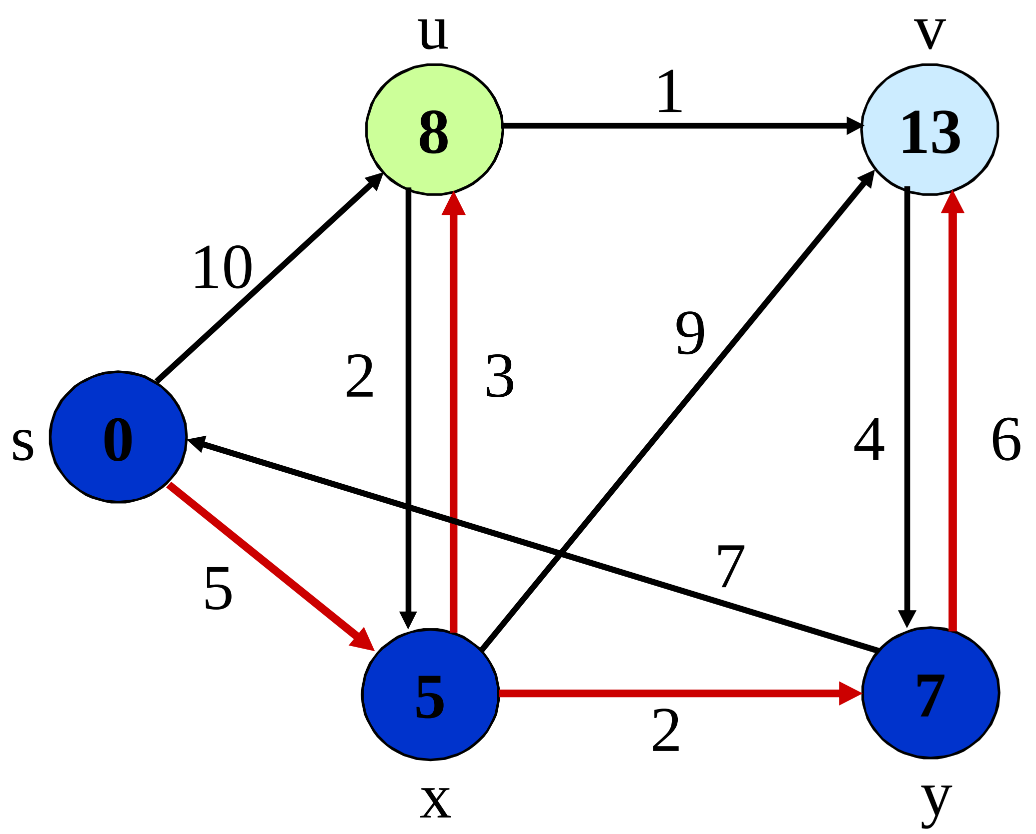
Dijkstra’s Algorithm: Example



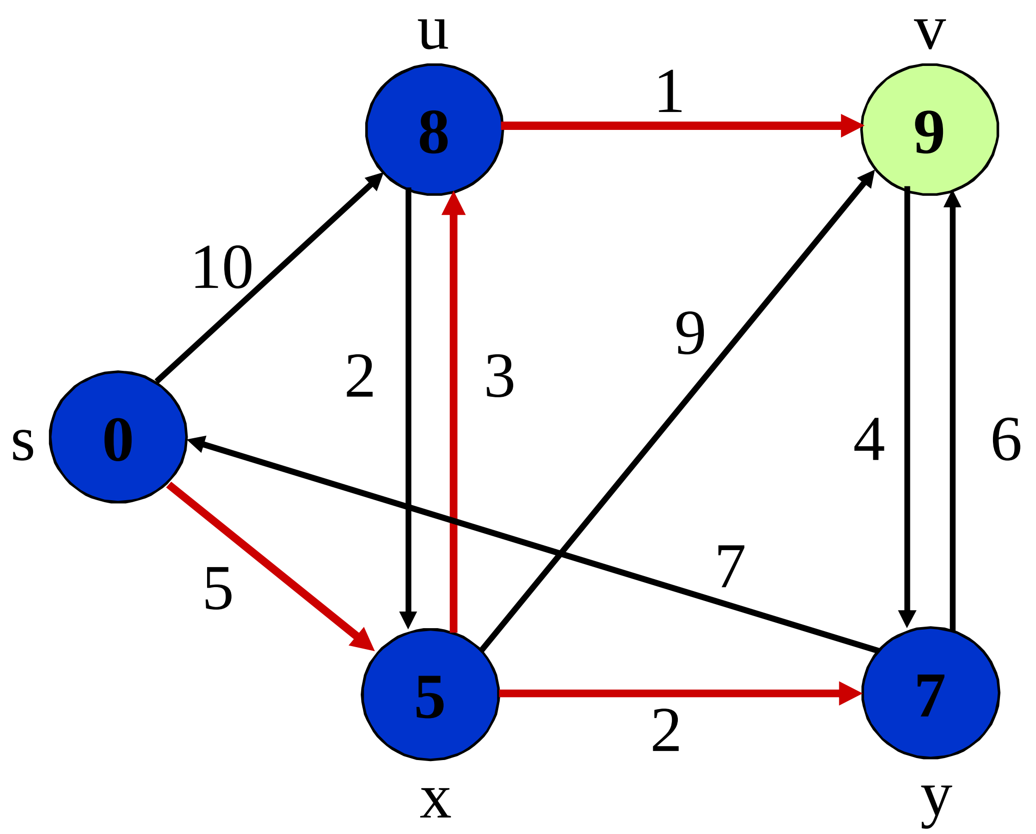
Dijkstra's Algorithm: Example



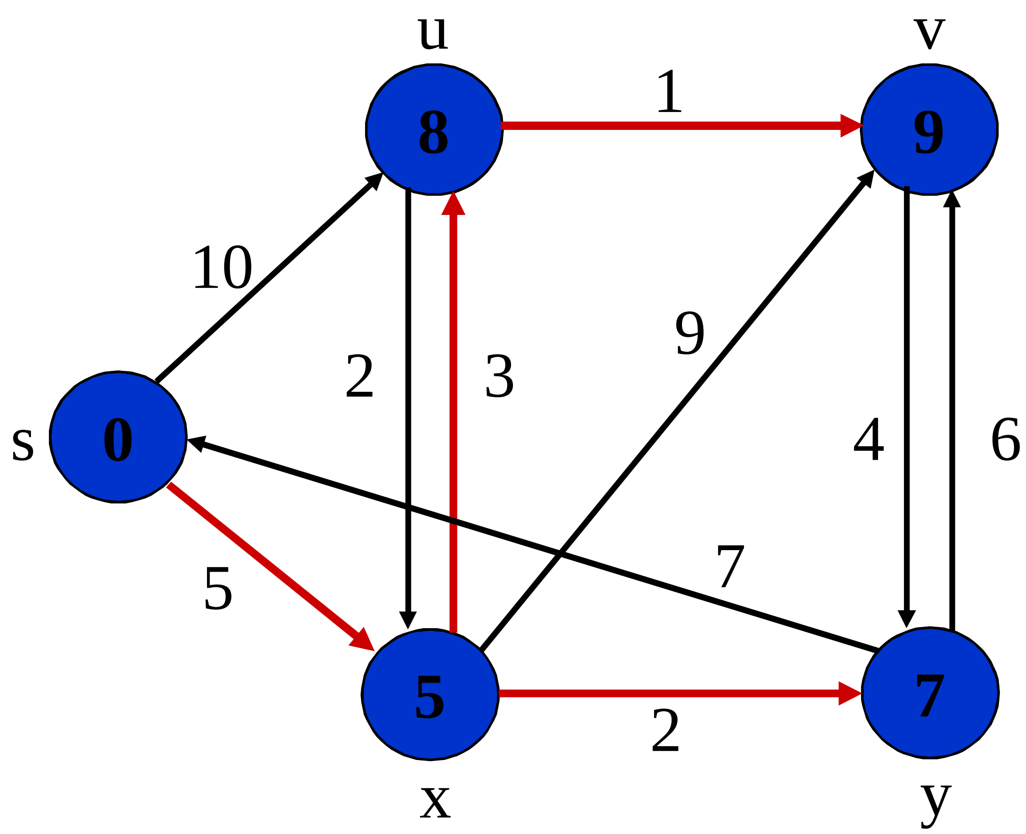
Dijkstra’s Algorithm: Example



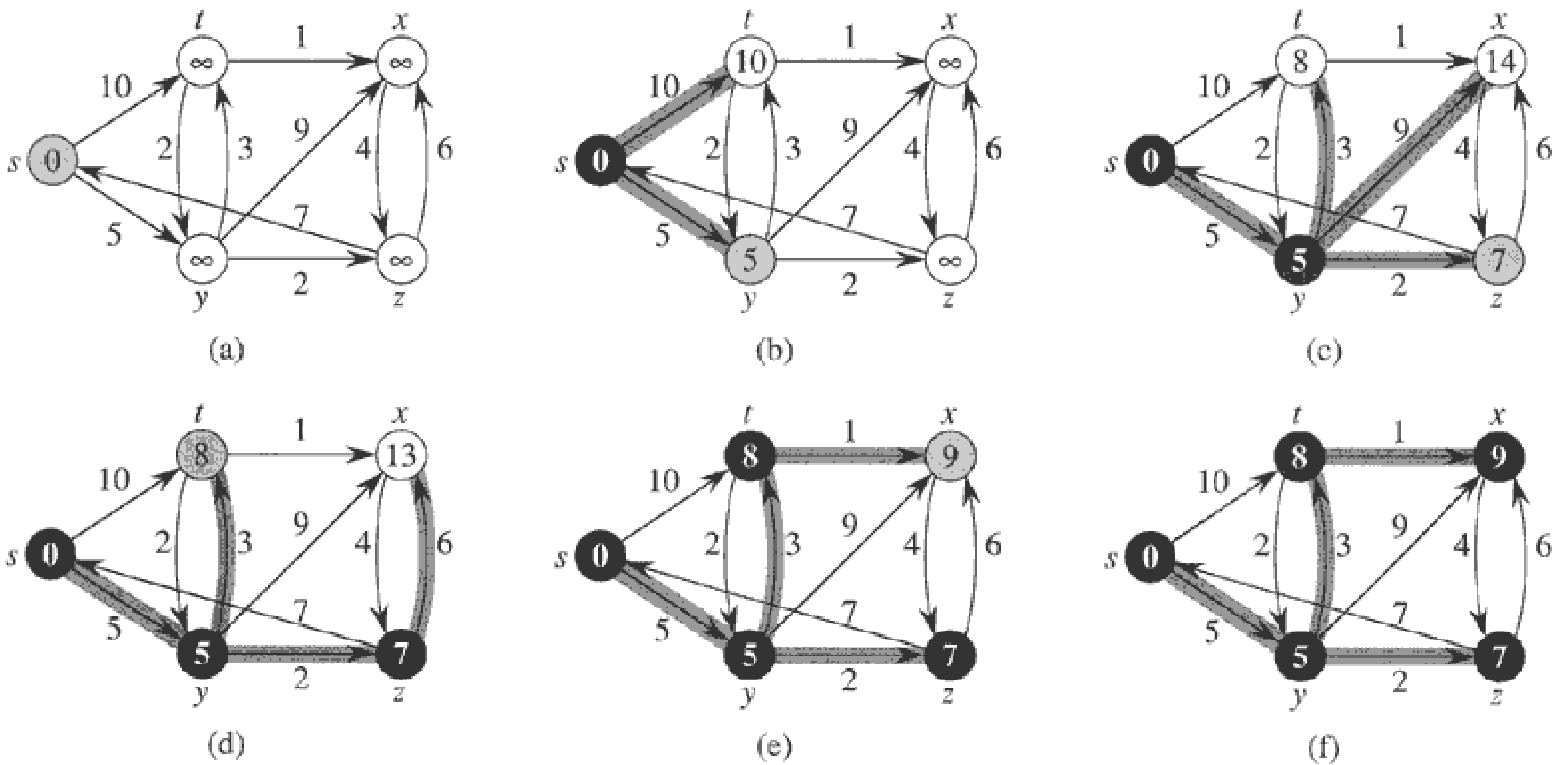
Dijkstra’s Algorithm: Example



Dijkstra’s Algorithm: Example



Dijkstra's Algorithm: Example



Dijkstra's Algorithm

```
Dijkstra(G)
   for each v ∈ V
      d[v] = ∞;
   d[s] = 0; S = ∅; Q = V;
   while (Q ≠ ∅)
      u = ExtractMin(Q);
      S = S U {u};
      for each v ∈ u->Adj[]
         if (d[v] > d[u]+w(u,v))
            d[v] = d[u]+w(u,v);
```

How many times is ExtractMin() called?

How many times is DecraseKey() called?

What will be the total running time?

Dijkstra's Algorithm

```
Dijkstra(G)
   for each v ∈ V
      d[v] = ∞;
   d[s] = 0; S = ∅; Q = V;

   while (Q ≠ ∅)
      u = ExtractMin(Q);
      S = S U {u};

      for each v ∈ u->Adj[]
         if (d[v] > d[u]+w(u,v))
            d[v] = d[u]+w(u,v);
```

A: *O*(E lg V) using binary heap for Q
Can achieve *O*(V lg V + E) with Fibonacci heaps

Dijkstra's Algorithm

```
Dijkstra(G)
   for each v ∈ V
      d[v] = ∞;
   d[s] = 0; S = ∅; Q = V;

   while (Q ≠ ∅)
      u = ExtractMin(Q);
      S = S U {u};

      for each v ∈ u->Adj[]
         if (d[v] > d[u]+w(u,v))
            d[v] = d[u]+w(u,v);
```

Correctness: we must show that when u is removed from Q, it has already converged