Single-Source Shortest Path

Chapter 24

Single-Source Shortest Path

- Problem: given a weighted directed graph G, find the minimum-weight path from a given source vertex s to another vertex v
  - "Shortest-path" = minimum weight
    - $\delta(u,v) = \min\{w(p): u \xrightarrow{p} v\}$, if there is a path from u to v
    - $\delta(u,v) = \infty$, otherwise
  - E.g., a road map: what is the shortest path from Belo Horizonte to Maringá?

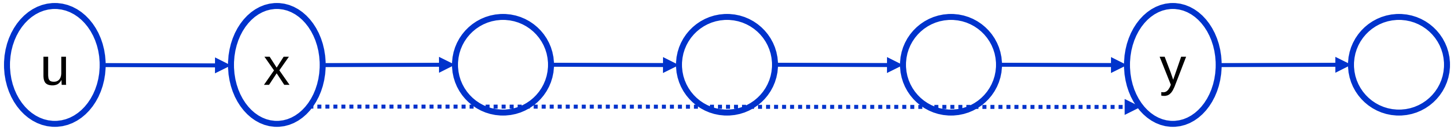
Shortest Path

- Variants of the shortest-path problem:
  - Single-source shortest path
  - Single-pair shortest path
  - All-pairs shortest path

- Satisfies two main properties:
  - Optimal substructure
  - Triangle inequality

Shortest Path Properties

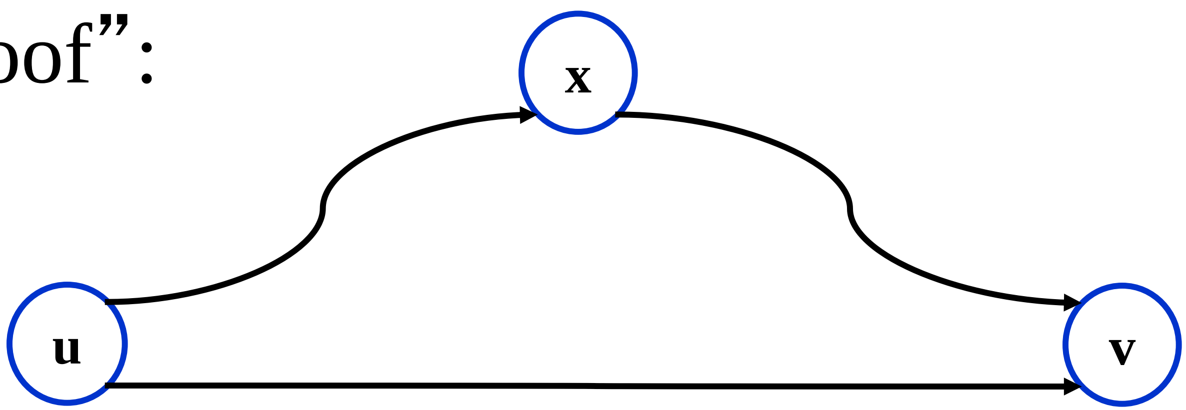
- Optimal substructure: the shortest path consists of shortest subpaths:



  - Proof: Cut and paste
  - Suppose some subpath is not a shortest path
    - There must then exist a shorter subpath
    - Could substitute the shorter subpath for a shorter path
    - But then overall path is not shortest path. Contradiction

Shortest Path Properties

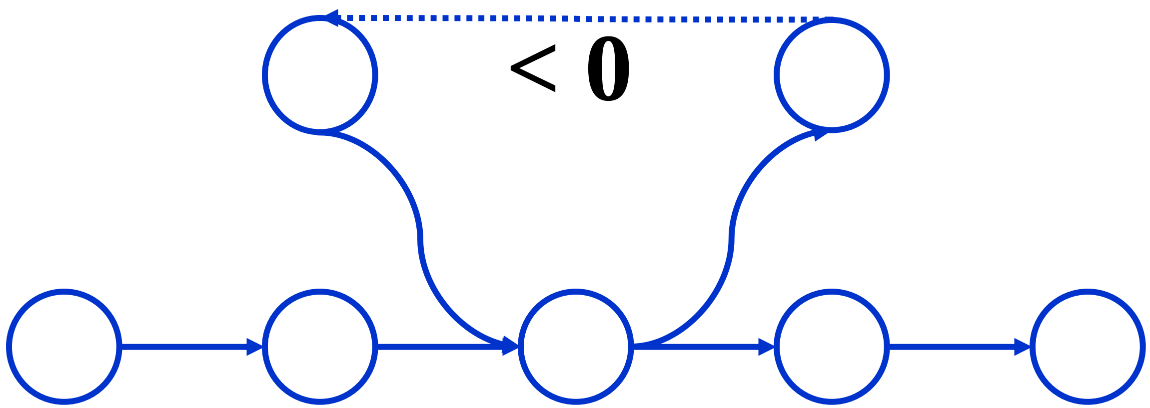
- Define $\delta(u,v)$ to be the weight of the shortest path from u to v
- Shortest paths satisfy the triangle inequality: $\delta(u,v) \le \delta(u,x) + \delta(x,v)$
- "Proof":



This path is no longer than any other path

Negative weight edges

- In graphs with negative weight cycles, some shortest paths will not exist (Why?):



Negative weight edges

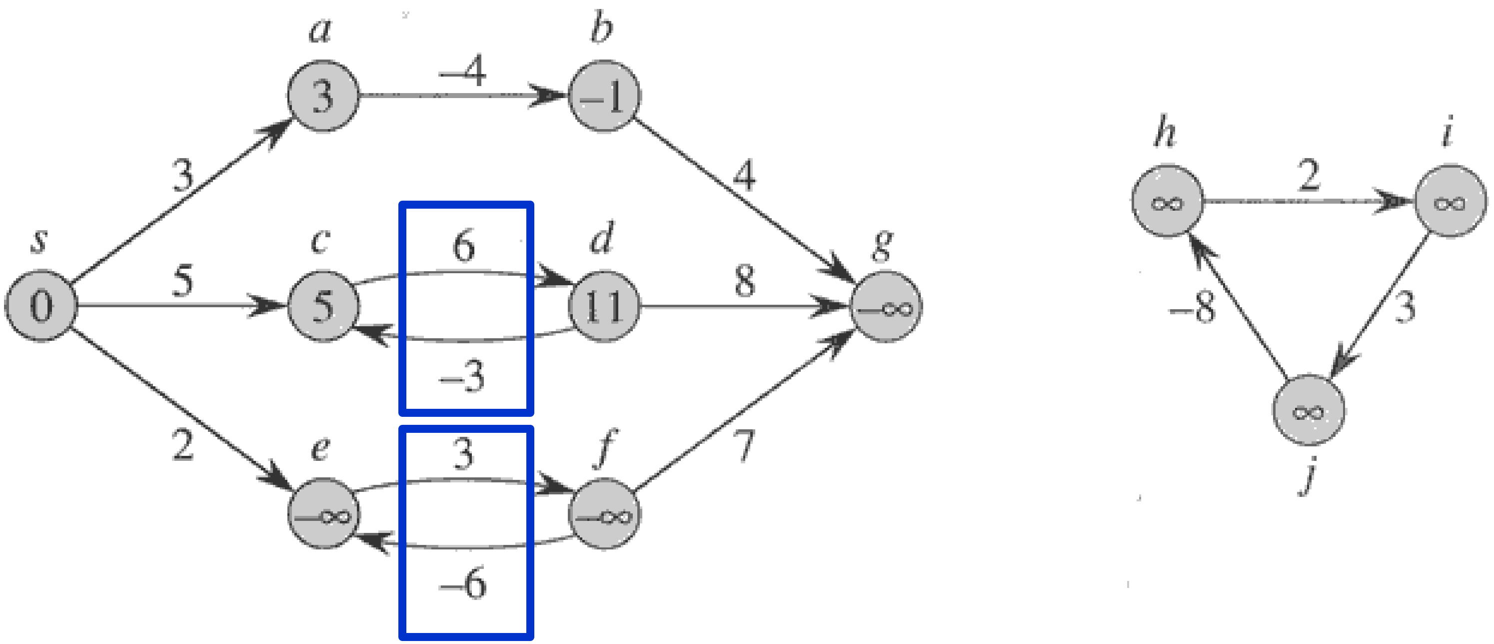
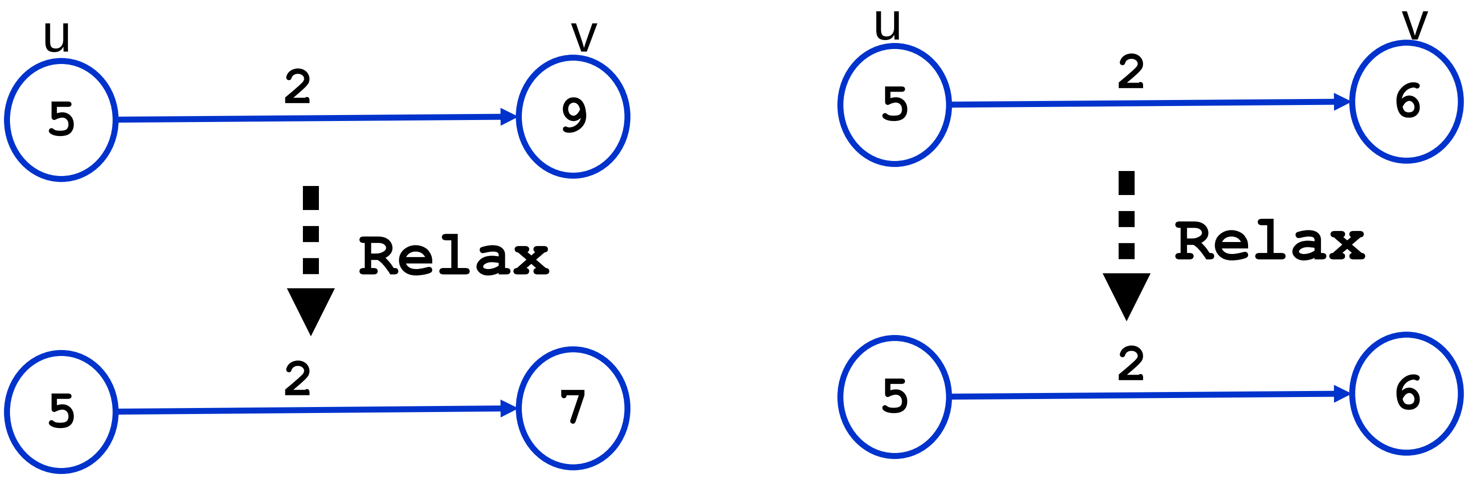


Figure 24.1 Negative edge weights in a directed graph. Shown within each vertex is its shortest-path weight from source s. Because vertices e and f form a negative-weight cycle reachable from s, they have shortest-path weights of $-\infty$. Because vertex g is reachable from a vertex whose shortest-path weight is $-\infty$, it, too, has a shortest-path weight of $-\infty$. Vertices such as h, i, and j are not reachable from s, and so their shortest-path weights are ∞, even though they lie on a negative-weight cycle.

Relaxation

- A key technique in shortest path algorithms is relaxation
  - Idea: for all v, maintain upper bound d[v] on δ(s,v)

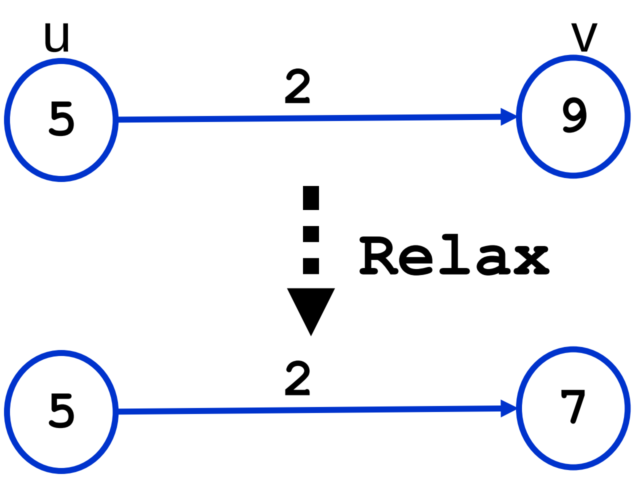
```
Relax(u,v,w) {
    if (d[v] > d[u]+w) then d[v]=d[u]+w;
}
```

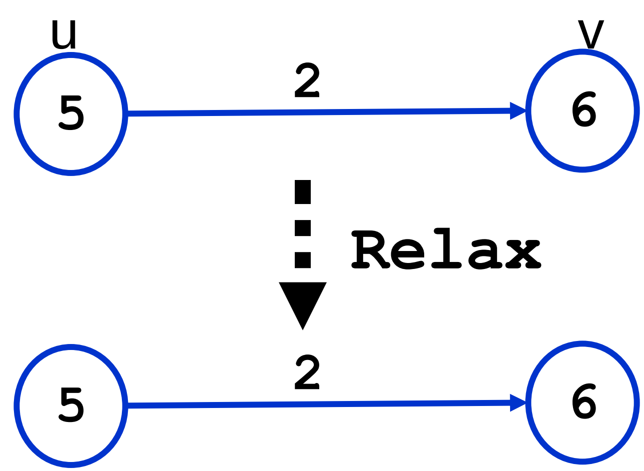


Relaxation

- A key technique in shortest path algorithms is relaxation
  - Idea: for all v, maintain upper bound d[v] on δ(s,v)

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





Algorithms

- Differ on how many times they relax each edge and the order in which they relax edges

- Dijkstra
  - Works only on graphs with non-negative weights
  - Relaxes each edge exactly once
- Bellman-Ford
  - Works on graphs with negative weigths
  - Relaxes each edge |V-1| times

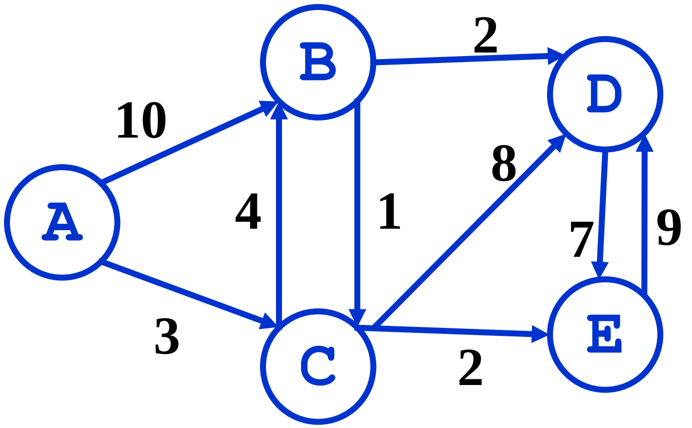
Dijkstra's Algorithm

- Similar to breadth-first search
  - Grows 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

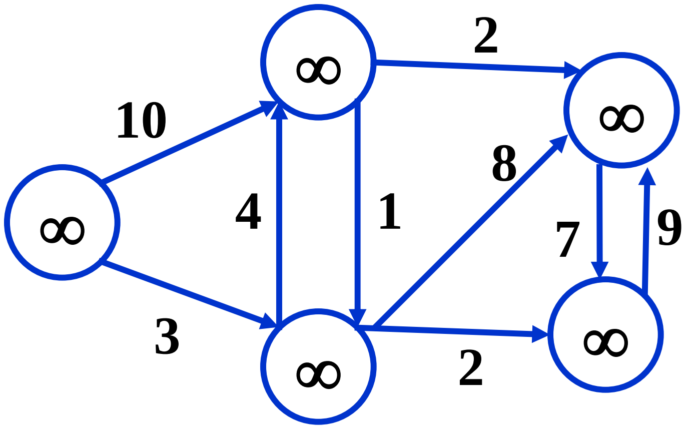
```
Dijkstra(G, s)
    for each v ∈ V
        d[v] = ∞;
    d[s] = 0; S = ∅; Q = V;
    while (Q ≠ ∅)
        u = ExtractMin(Q);
        S = S U {u};
        for each v ∈ u->Adj[]
            if (d[v] > d[u]+w(u,v))
                d[v] = d[u]+w(u,v);
```

Relaxation Step



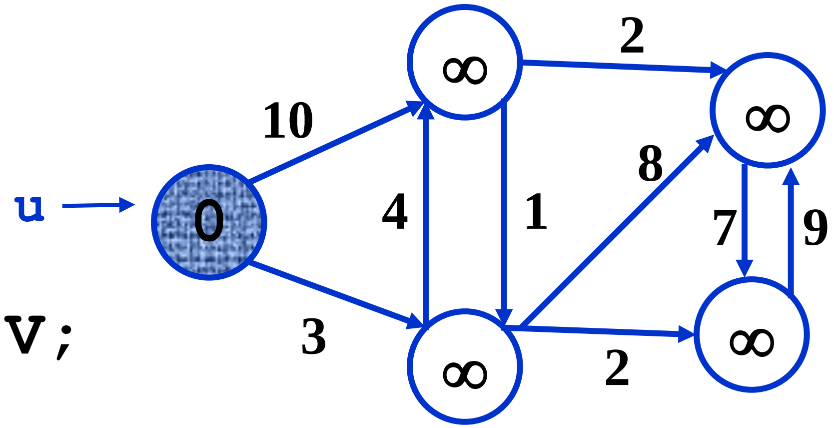
Dijkstra's Algorithm

```
Dijkstra(G, s)
    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);
```



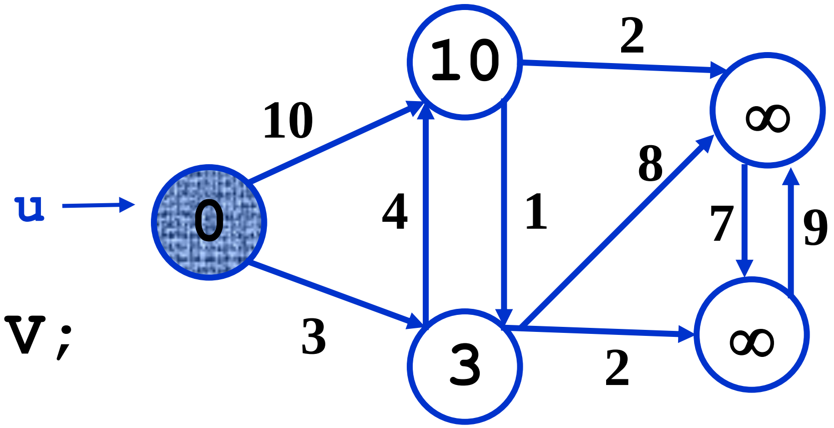
Dijkstra's Algorithm

```
Dijkstra(G, s)
    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);
```



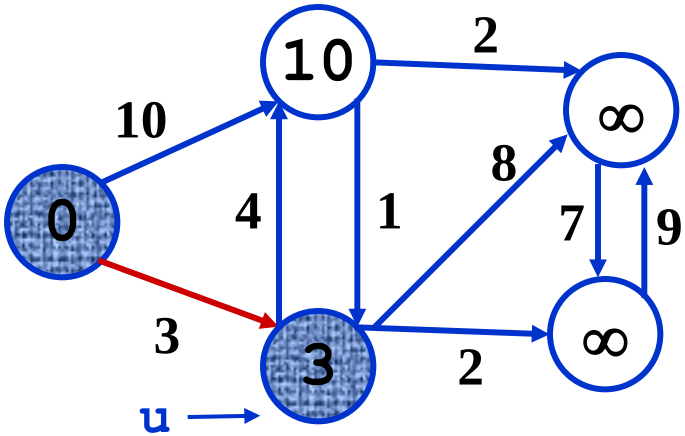
Dijkstra’s Algorithm

```
Dijkstra(G, s)
    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);
```



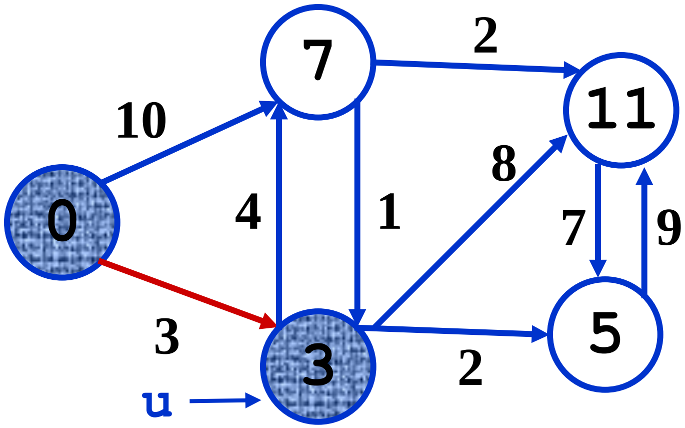
Dijkstra's Algorithm

```
Dijkstra(G, s)
    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);
```



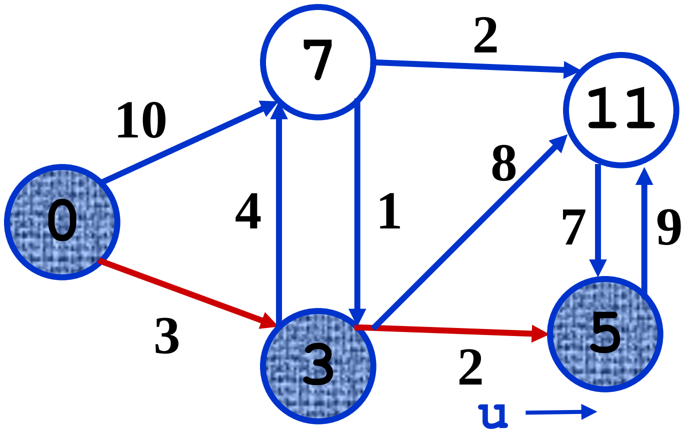
Dijkstra's Algorithm

```
Dijkstra(G, s)
   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);
```



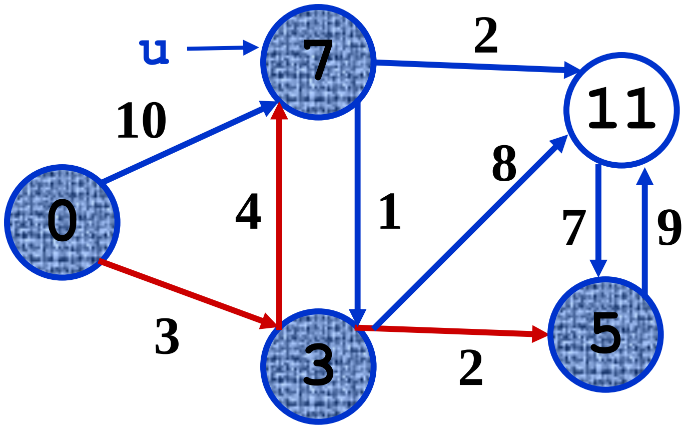
Dijkstra's Algorithm

```
Dijkstra(G, s)
    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);
```



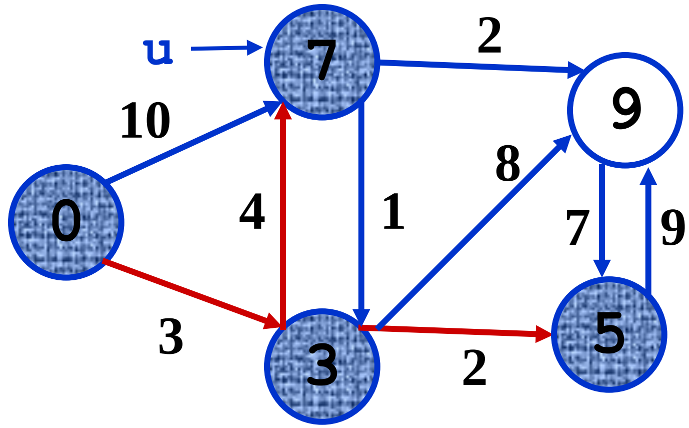
Dijkstra's Algorithm

```
Dijkstra(G, s)
    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);
```



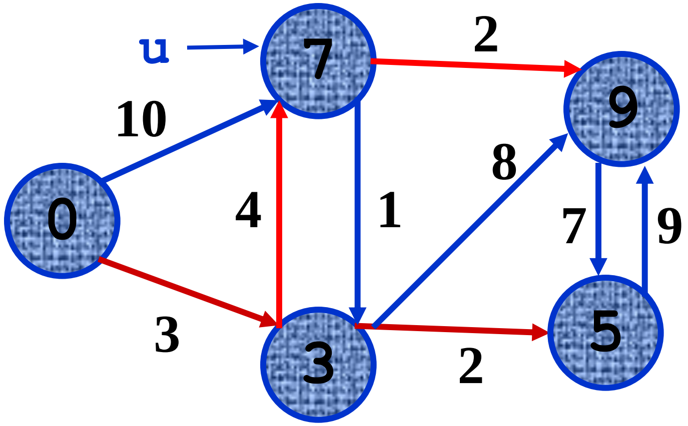
Dijkstra's Algorithm

```
Dijkstra(G, s)
   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);
```



Dijkstra's Algorithm

```
Dijkstra(G, s)
    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);
```



Dijkstra's Algorithm

```
Dijkstra(G, s)
    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);
```

Implicit DecreaseKey()

Dijkstra's Algorithm

```
Dijkstra(G, s)
    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?

Running time

- Time = |V| Textract + |E| Tdecrease
- Depends on data structure
- O(E lg V) using binary heap for Q
- O(V lg V + E) with Fibonacci heaps

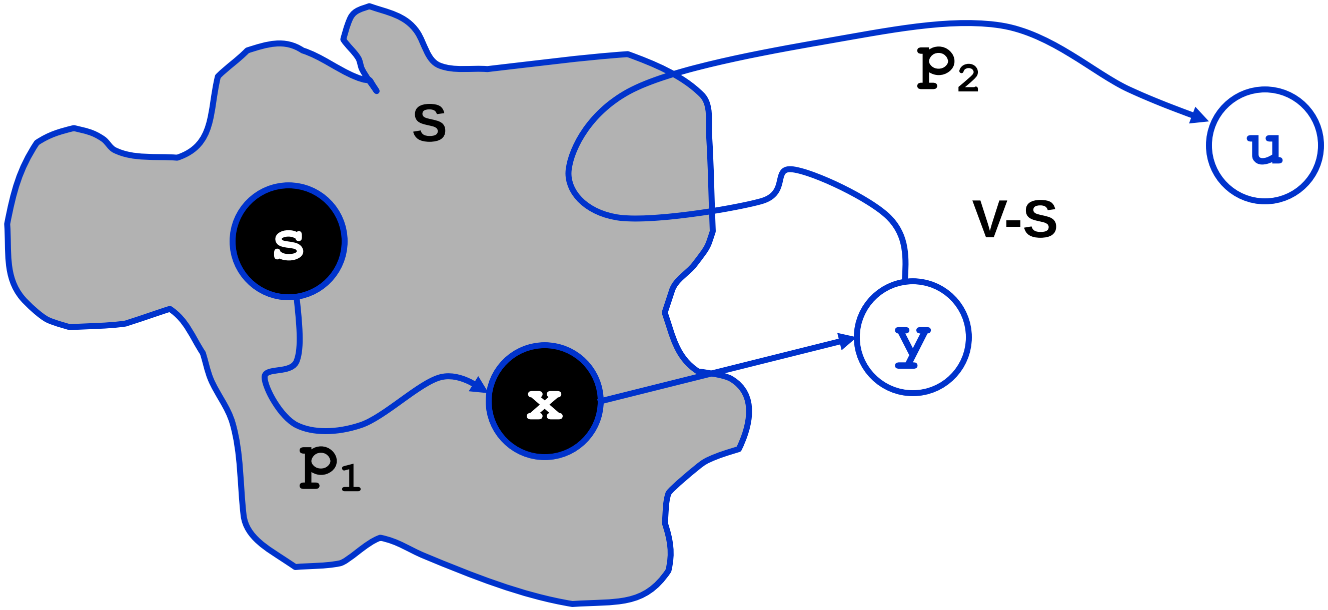
Q	Textract	Tdecrease	Total
Binary Heap	O(lg V)	O(lg V)	O((V+E) lg V)
Fib Heap	O(lg V)amort	O(1)amort	O(E +Vlg V)

Correctness Of Dijkstra's Algorithm

- Show that when Dijkstra terminates, u.d = δ(s,u) ∀ u in V
- Show that, at each iteration,

  d[u] = δ(s,u) for the vertex added to S

  d[u] = δ'(s,u) for vertexes outside S (δ' best path using only vertexes in S



Dijkstra's Algorithm

- Suppose the directed graph is unweighted
  - W(u,v) = 1
- Can I do better than O(E+ V lg V)?
- Yes! Breadth-first search -> same as Dijkstra
- 2 main differences:
  - Uses a queue (FIFO)
  - Relaxation slightly different

```
If (d[v] = ∞){
    d[v] = d[v]+1
    Enqueue(Q,v)
}
```

- O(V+E)

Bellman-Ford Algorithm

- Edges can have negative weights
- Capable of detecting negative cycles

Bellman-Ford Algorithm

```
BellmanFord(G, s)
    for each v ∈ V
        d[v] = ∞;
    d[s] = 0;
    for i=1 to |V|-1
        for each edge (u,v) ∈ E
         if (d[v] > d[u]+w)
          then d[v]=d[u]+w;
    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 δ

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

Test for solution
Under what condition do we get a solution?

Bellman-Ford Algorithm

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

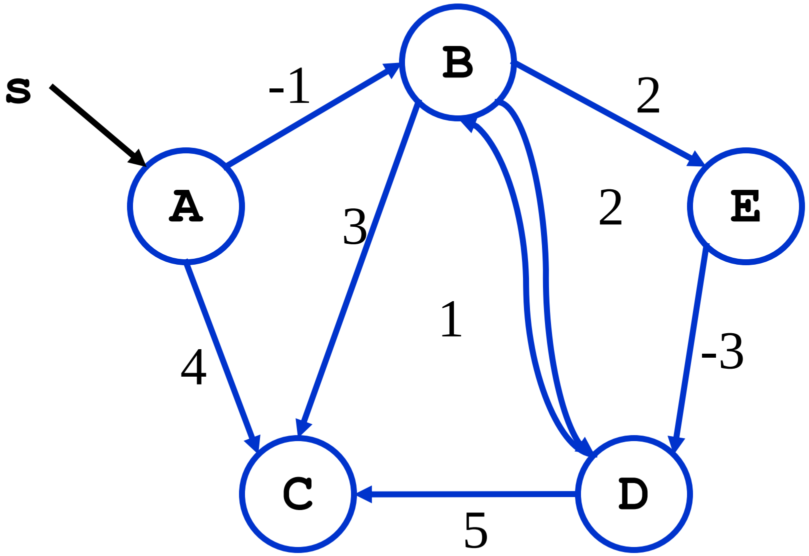
Initialize d[], which will converge to shortest-path value δ

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

Test for solution
Under what condition do we get a solution?

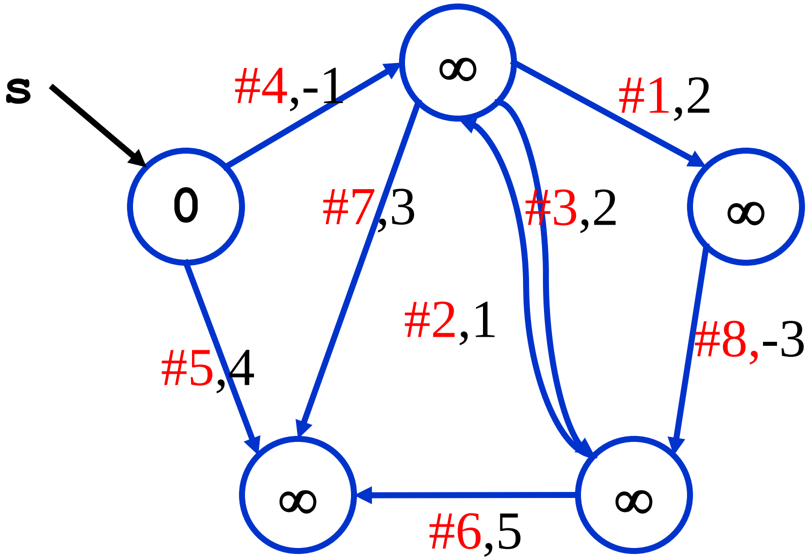
Bellman-Ford Algorithm

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



Bellman-Ford Algorithm

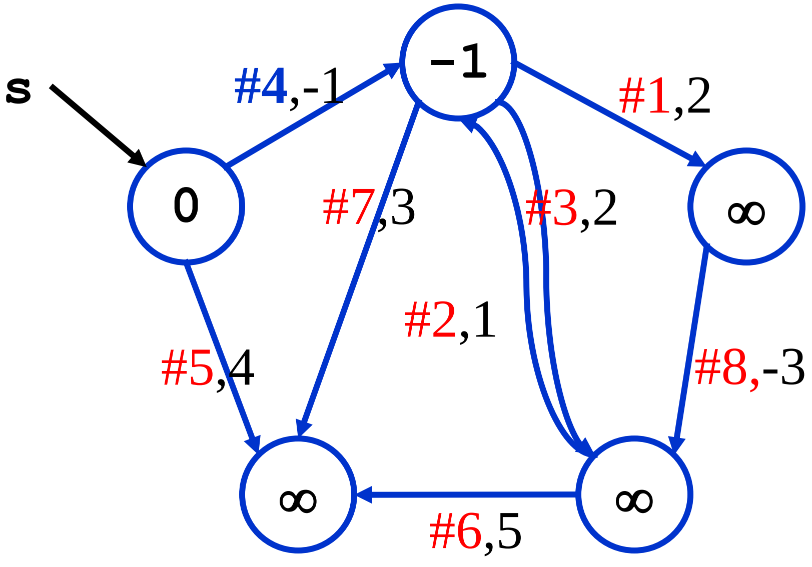
```
BellmanFord(G, s)
   for each v ∈ V
      d[v] = ∞;
   d[s] = 0;
   for i=1 to |V|-1
      for each edge (u,v) ∈ E
       if (d[v] > d[u]+w)
        then d[v]=d[u]+w;
   for each edge (u,v) ∈ E
      if (d[v] > d[u] + w(u,v))
           return "no solution";
```



i = 1

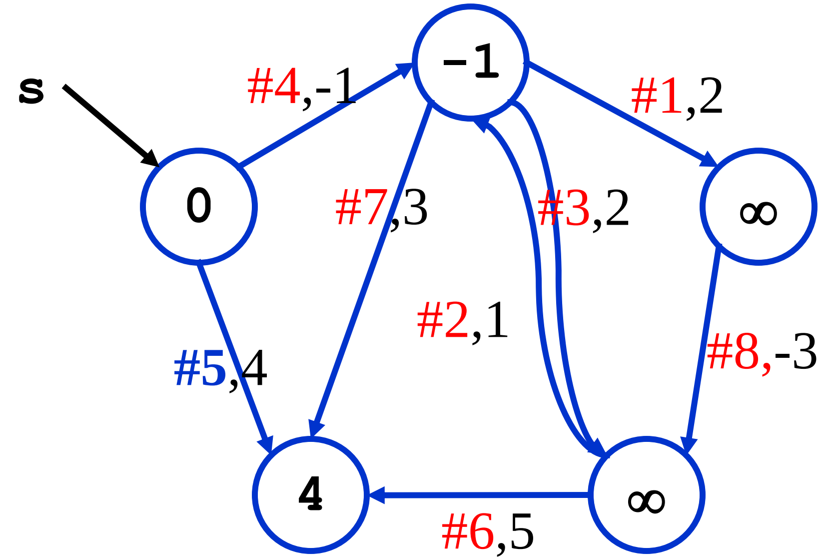
Bellman-Ford Algorithm

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



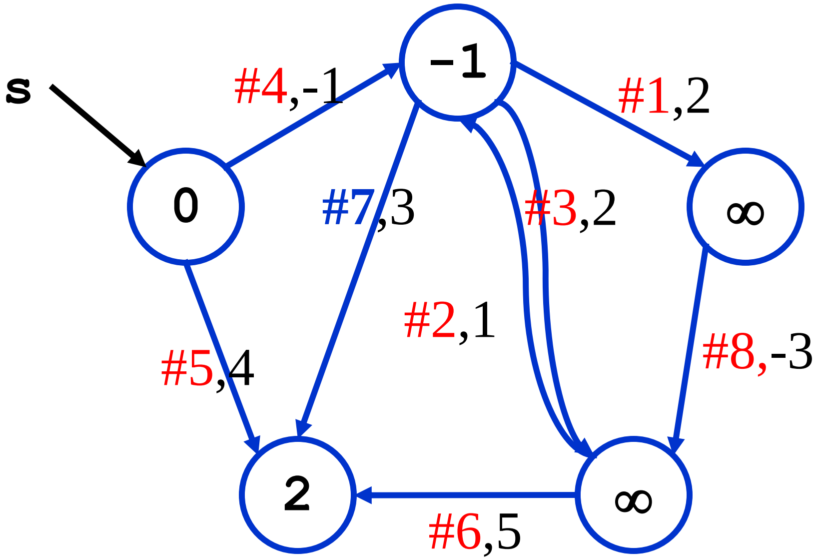
Bellman-Ford Algorithm

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



Bellman-Ford Algorithm

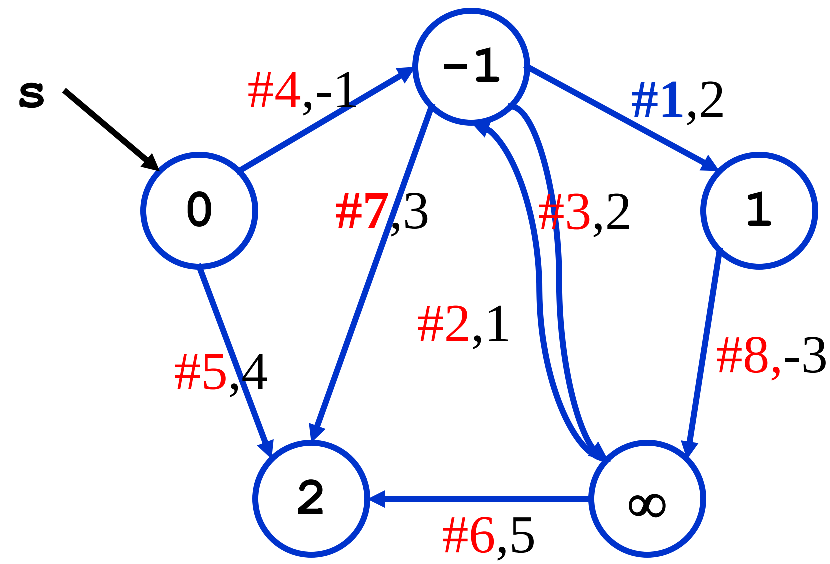
```
BellmanFord(G, s)
    for each v ∈ V
        d[v] = ∞;
    d[s] = 0;
    for i=1 to |V|-1
        for each edge (u,v) ∈ E
         if (d[v] > d[u]+w)
          then d[v]=d[u]+w;
    for each edge (u,v) ∈ E
        if (d[v] > d[u] + w(u,v))
             return "no solution";
```



end for i = 1
i = 2

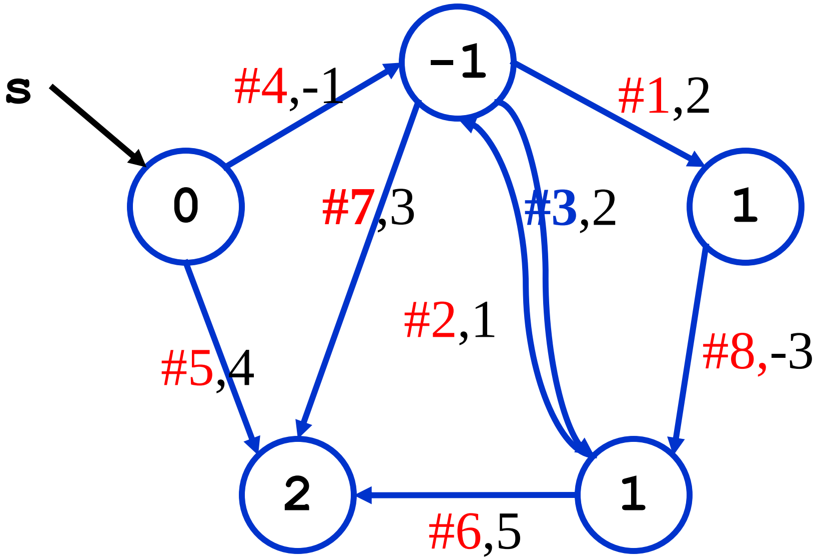
Bellman-Ford Algorithm

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



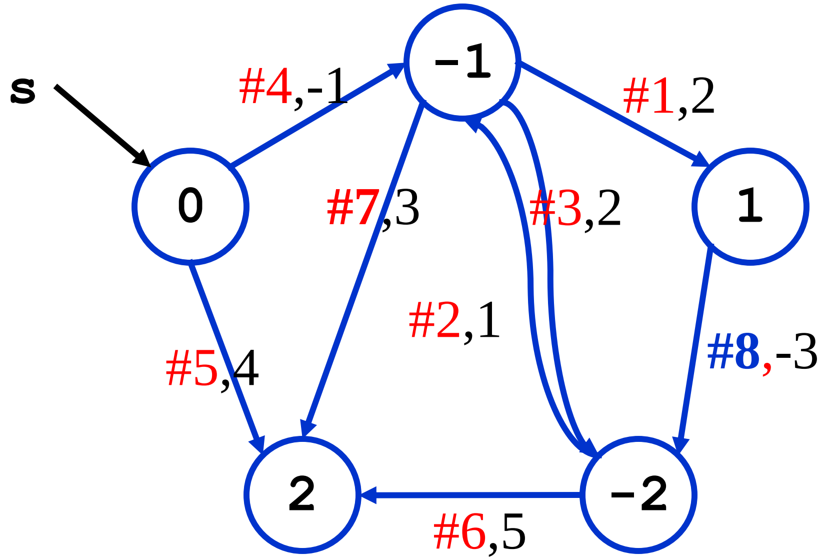
Bellman-Ford Algorithm

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



Bellman-Ford Algorithm

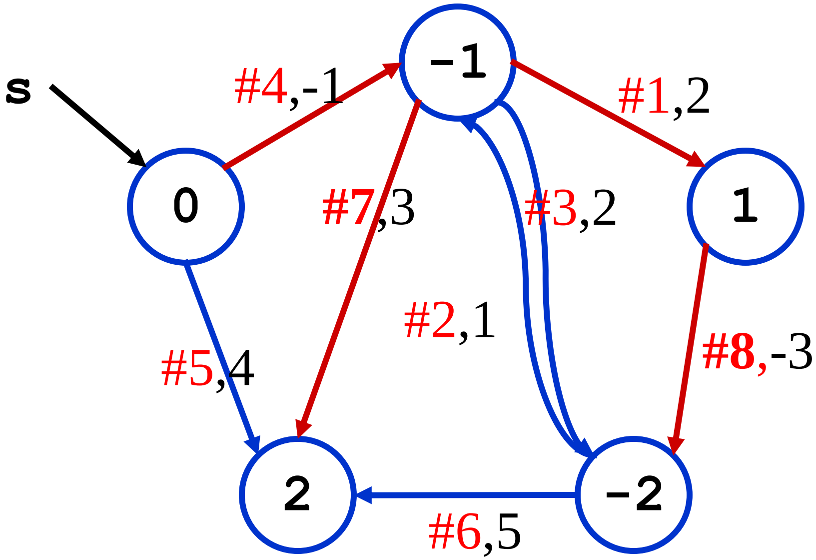
```
BellmanFord(G, s)
    for each v ∈ V
        d[v] = ∞;
    d[s] = 0;
    for i=1 to |V|-1
        for each edge (u,v) ∈ E
         if (d[v] > d[u]+w)
          then d[v]=d[u]+w;
    for each edge (u,v) ∈ E
        if (d[v] > d[u] + w(u,v))
             return "no solution";
```



end for i = 2
No changes for i > 2

Bellman-Ford Algorithm

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



Bellman-Ford Algorithm

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

What will be the running time?

Bellman-Ford Algorithm

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

What will be the running time?

A: O(VE)

Bellman-Ford

- Prove: after |V|-1 passes, all d values correct
  - Consider shortest path from s to v:
    $s \rightarrow v_1 \rightarrow v_2 \rightarrow v_3 \rightarrow v_4 \rightarrow v$
    - Initially, d[s] = 0 is correct, and doesn't change
    - After 1 pass through edges, d[v_1] is correct and doesn't change
    - After 2 passes, d[v_2] is correct and doesn't change
    - …
    - Terminates in |V| - 1 passes: (Why?)
    - What if it doesn't?

SSSP Algorithms

- Dijkstra: O((V+E) lg V) – binary heap
- Bellman-Ford: O(VE)
  - Detects negative cycles