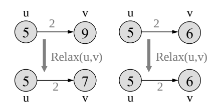
Dijkstra’s Algorithm

1

Relaxation

- For each vertex *v* in the graph, we maintain *v*.**d**(), the estimate of the shortest path from *s*, initialized to ∞ at the start
- Relaxing an edge (*u*,*v*) means testing whether we can improve the shortest path to *v* found so far by going through *u*




```
Relax (u,v,G)
if v.d() > u.d()+G.w(u,v) then
   v.setd(u.d()+G.w(u,v))
   v.setparent(u)
```

2

Dijkstra's Algorithm

- Non-negative edge weights
- Greedy, similar to Prim's algorithm for MST
- Like breadth-first search (if all weights = 1, one can simply use BFS)
- Use *Q*, a priority queue ADT keyed by *v*.**d**() (BFS used FIFO queue, here we use a PQ, which is re-organized whenever some **d** decreases)
- Basic idea
  - maintain a set *S* of solved vertices
  - at each step select "closest" vertex *u*, add it to *S*, and relax all edges from *u*

3

Dijkstra’s ALgorithm

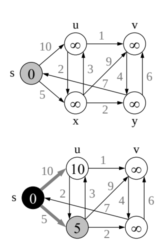
Solution to **Single-source (multiple-destination).**

DIJKSTRA(G, w, s)
INITIALIZE-SINGLE-SOURCE(G, s)
$S = \emptyset$
$Q = G.V$
while $Q \neq \emptyset$
  $u =$ EXTRACT-MIN(Q)
  $S = S \cup \{u\}$
  for each vertex $v \in G.Adj[u]$
    RELAX(u, v, w)

5

Dijkstra’s Example

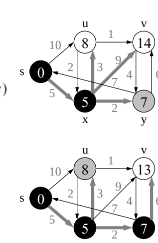
DIJKSTRA(G, w, s)
INITIALIZE-SINGLE-SOURCE(G, s)
$S = \emptyset$
$Q = G.V$
while $Q \neq \emptyset$
  $u =$ EXTRACT-MIN(Q)
  $S = S \cup \{u\}$
  for each vertex $v \in G.Adj[u]$
    RELAX(u, v, w)



6

Dijkstra’s Example

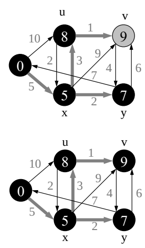
DIJKSTRA(G, w, s)
INITIALIZE-SINGLE-SOURCE(G, s)
$S = \emptyset$
$Q = G.V$
while $Q \neq \emptyset$
  $u =$ EXTRACT-MIN(Q)
  $S = S \cup \{u\}$
  for each vertex $v \in G.Adj[u]$
    RELAX(u, v, w)



7

Dijkstra's Example

```
DIJKSTRA(G, w, s)
INITIALIZE-SINGLE-SOURCE(G, s)
S = ∅
Q = G.V
while Q ≠ ∅
    u = EXTRACT-MIN(Q)
    S = S ∪ {u}
    for each vertex v ∈ G.Adj[u]
        RELAX(u, v, w)
```



8

Running time?

```
DIJKSTRA(G, w, s)
INITIALIZE-SINGLE-SOURCE(G, s)
S = ∅
Q = G.V
while Q ≠ ∅
    u = EXTRACT-MIN(Q)
    S = S ∪ {u}
    for each vertex v ∈ G.Adj[u]
        RELAX(u, v, w)
```

- Depends on the heap implementation

	1 MakeHeap	|V| ExtractMin	|E| DecreaseKey	Total
Array	O(|V|)	O(|V|²)	O(|E|)	O(|V|²)
Bin heap	O(|V|)	O(|V| log |V|)	O(|E| log |V|)	O((|V|+|E|) log |V|) O(|E| log |V|)
Fib heap	O(|V|)	O(|V| log |V|)	O(|E|)	O(|V| log |V| + |E|)

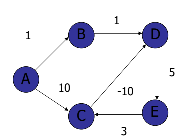
9

Bellman Ford's Algorithm

10

Negative cycles

What is the shortest path from a to e?



11

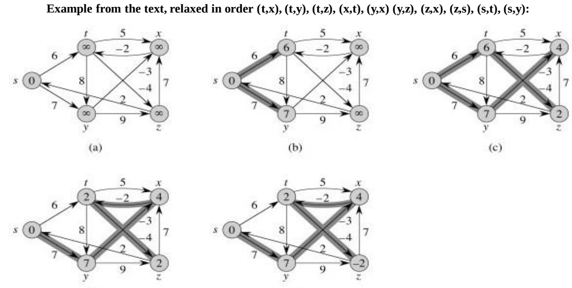
Bellman-Ford Algorithm

```
BELLMAN-FORD(G, w, s)
INITIALIZE-SINGLE-SOURCE(G, s)
for i = 1 to |G.V| - 1
    for each edge (u, v) ∈ G.E
        RELAX(u, v, w)
for each edge (u, v) ∈ G.E
    if v.d > u.d + w(u, v)
        return FALSE
return TRUE
```

14

12

Example from the text, relaxed in order (t,x), (t,y), (t,z), (x,t), (y,x) (y,z), (z,x), (z,s), (s,t), (s,y):



13

Analysis

```
BELLMAN-FORD(G, w, s)
INITIALIZE-SINGLE-SOURCE(G, s)
for i = 1 to |G.V| - 1
    for each edge (u, v) ∈ G.E
        RELAX(u, v, w)
for each edge (u, v) ∈ G.E
    if v.d > u.d + w(u, v)
        return FALSE
return TRUE
```

14