CS 340 - Analysis of Algorithms
Greedy Algorithms – Dijkstra's

Dianna Xu

Dijkstra's Algorithm

Definitions

- Given a directed graph $G = (V, E)$
- Each edge $(u, v) \in E$ is associated with an edge weight $w(u, v)$
- The *length* of a path is the sum of weights along the edges of the path
- The *distance* between two vertices u and v is the minimum length of any path between the vertices, denoted $\delta(u, v)$

3

Shortest Path Variations

- Single-source, single-sink
- Collection of source-sink pairs
- Single-source to all
- All-pairs

- Typically assume non-sparse graphs, and connected

4

Single-Source Shortest Path

- Given a directed graph $G = (V, E)$ with edge weights and a source vertex $s \in V$, determine the distance $\delta(s, v), \forall v \in V$
- Negative weights? – Bellman-Ford
- Dijkstra's – simple greedy algorithm that assumes nonnegative weights

5

Description

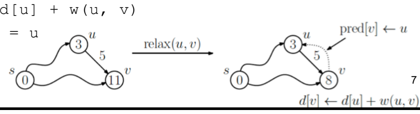
- Maintain an estimate of the shortest path for each vertex, $d[v]$, from s
- $d[v]$ stores the length of the shortest path from s to v that the algorithm currently knows of
- Initially, $d[s] = 0$ and $d[v] = \infty, v \neq s$
- The algorithm updates $d[v]$ as it processes more and more vertices - relaxation

6

Relaxation

- Consider an edge (u, v).
- We have current values for $d[u]$ and $d[v]$
- $d[v]$ should be the smaller of $d[u] + w(u, v)$, or $d[v]$ - do we want to go through u?

```
relax(u, v){
  if (d[u]+w(u, v)<d[v]) {
    d[v] = d[u] + w(u, v)
    pred[v] = u
  }
}
```



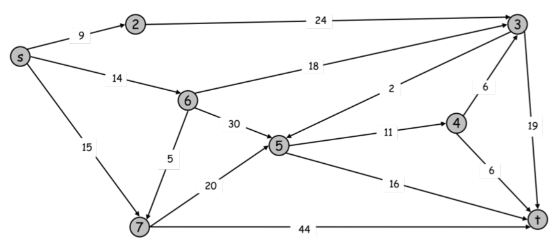
7

Dijkstra's

- A subset of vertices $S \subseteq V$, for which we know the true distance, i.e. $d[v] = \delta(s, v)$
- Initially, $S = \{\}$
- $d[s] = 0, d[v] = \infty, v \neq s$
- Select vertices from $V \backslash S$ to add to S (vertices stored in priority queue)
- Each time select the vertex $u \in V \backslash S$ for which $d[u]$ is minimum and update the d values of u's neighbors

8

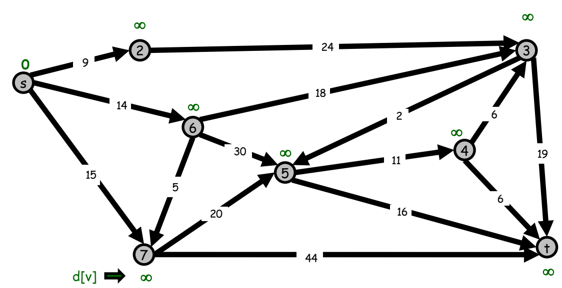
Input



9

Dijkstra's Shortest Path Algorithm

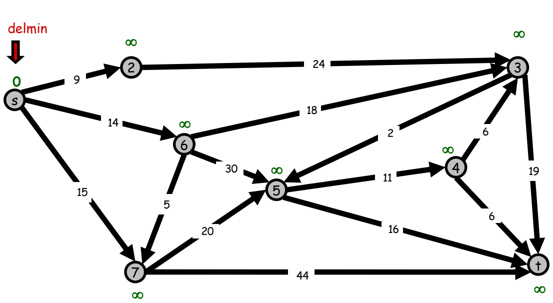
S = { }
V = { s, 2, 3, 4, 5, 6, 7, t }



10

Dijkstra's Shortest Path Algorithm

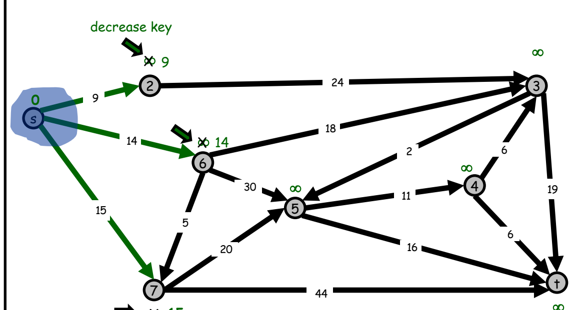
S = { }
V = { s, 2, 3, 4, 5, 6, 7, t }



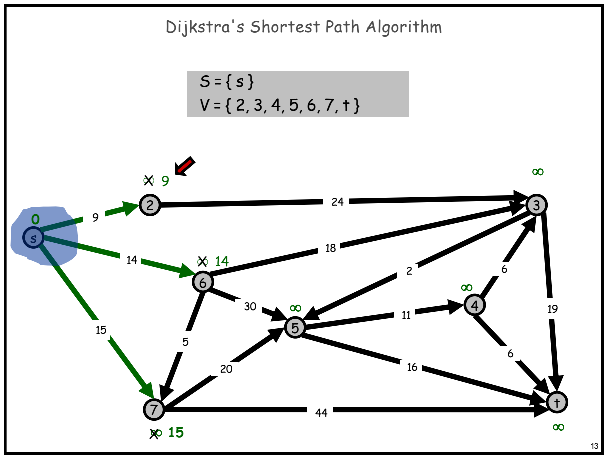
11

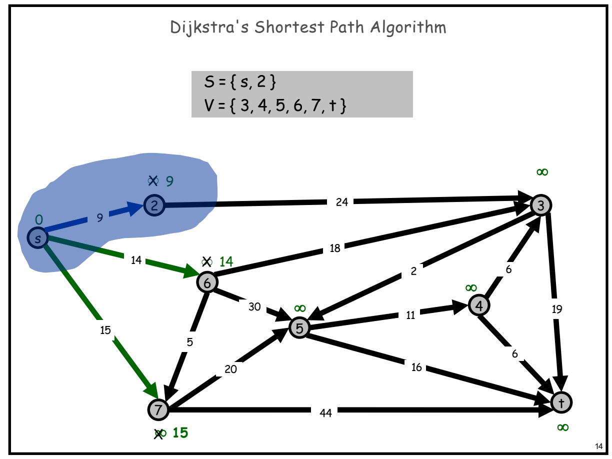
Dijkstra's Shortest Path Algorithm

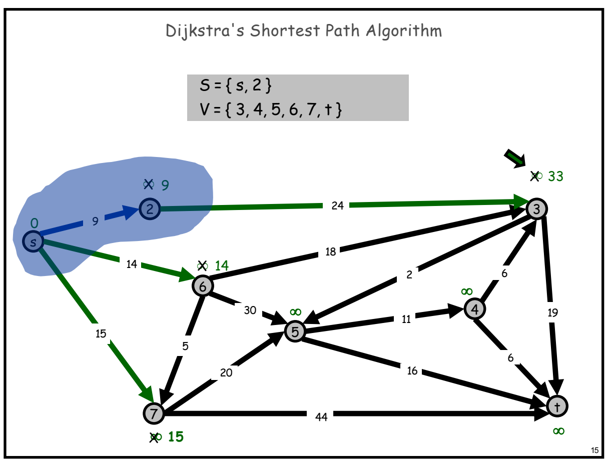
S = { s }
V = { 2, 3, 4, 5, 6, 7, t }

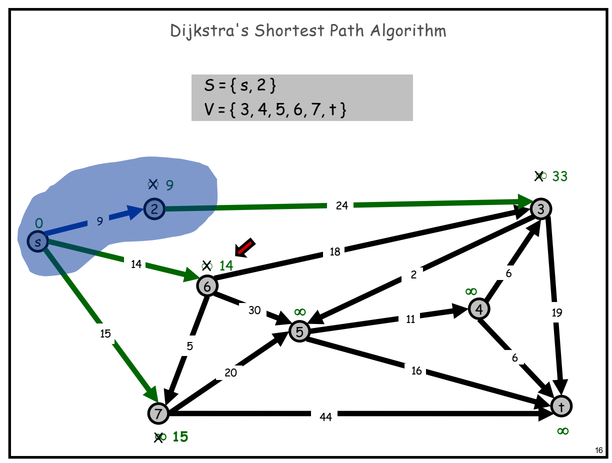


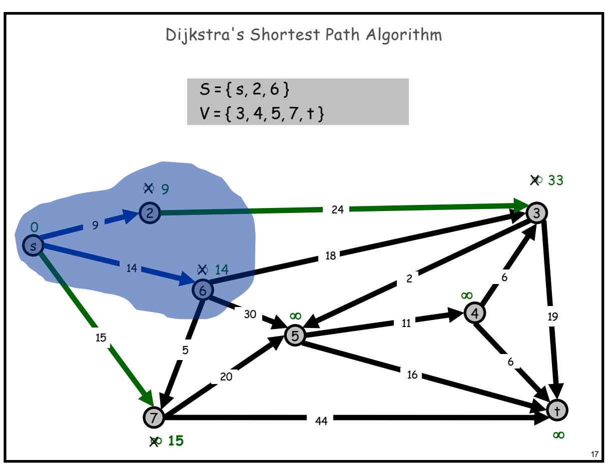
12

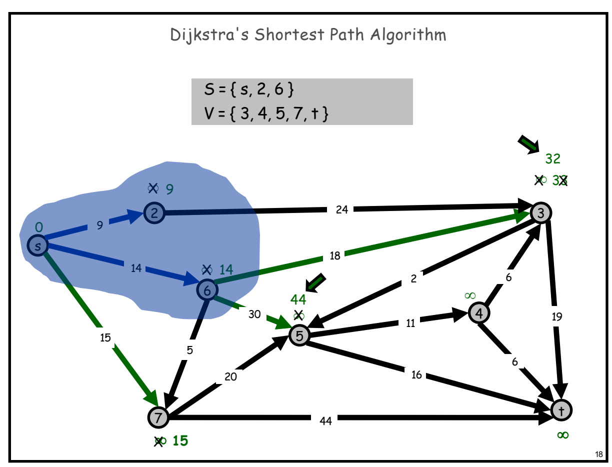

Dijkstra's Shortest Path Algorithm
S = { s }
V = { 2, 3, 4, 5, 6, 7, t }
13


Dijkstra's Shortest Path Algorithm
S = { s, 2 }
V = { 3, 4, 5, 6, 7, t }
14


Dijkstra's Shortest Path Algorithm
S = { s, 2 }
V = { 3, 4, 5, 6, 7, t }
15

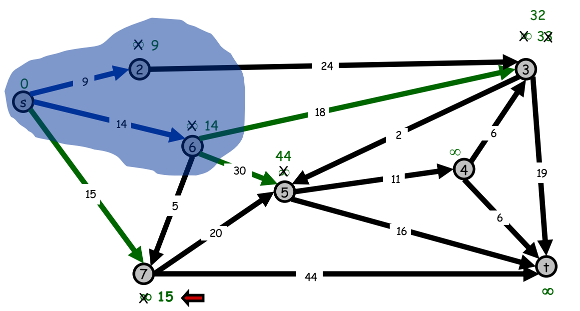

Dijkstra's Shortest Path Algorithm
S = { s, 2 }
V = { 3, 4, 5, 6, 7, t }
16


Dijkstra's Shortest Path Algorithm
S = { s, 2, 6 }
V = { 3, 4, 5, 7, t }
17


Dijkstra's Shortest Path Algorithm
S = { s, 2, 6 }
V = { 3, 4, 5, 7, t }
18

Dijkstra's Shortest Path Algorithm

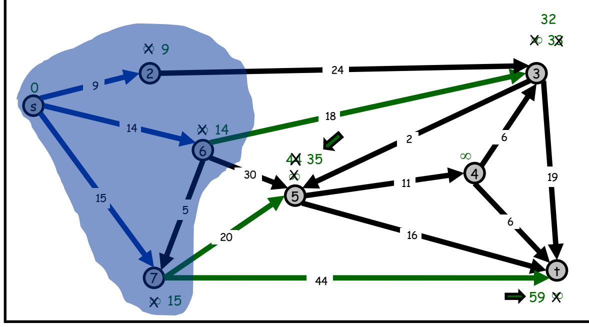
S = { s, 2, 6 }
V = { 3, 4, 5, 7, t }



19

Dijkstra's Shortest Path Algorithm

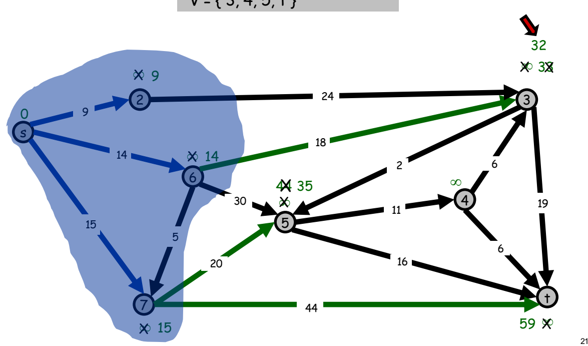
S = { s, 2, 6, 7 }
V = { 3, 4, 5, t }



20

Dijkstra's Shortest Path Algorithm

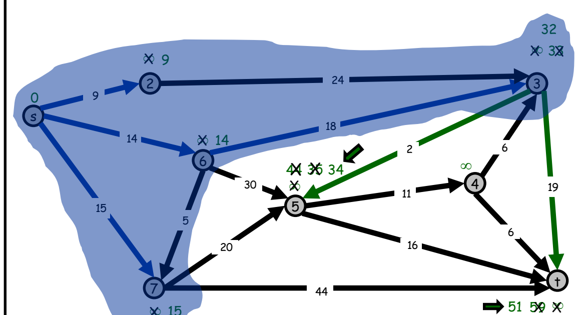
S = { s, 2, 6, 7 }
V = { 3, 4, 5, t }



21

Dijkstra's Shortest Path Algorithm

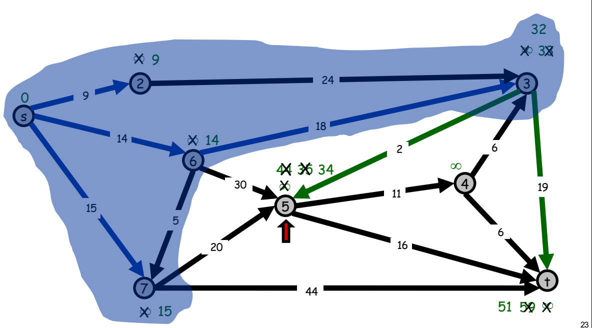
S = { s, 2, 3, 6, 7 }
V = { 4, 5, t }



22

Dijkstra's Shortest Path Algorithm

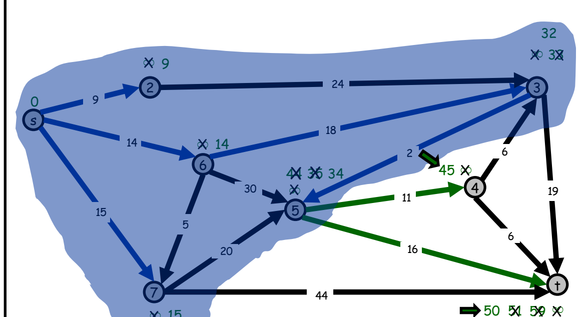
S = { s, 2, 3, 6, 7 }
V = { 4, 5, t }



23

Dijkstra's Shortest Path Algorithm

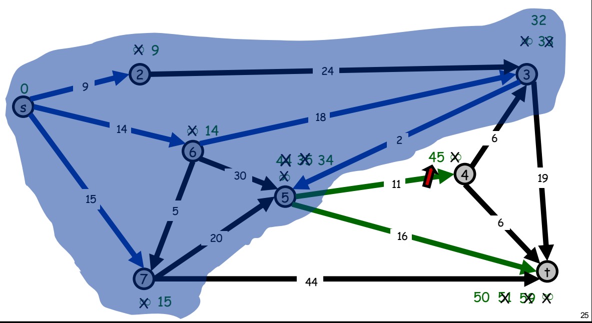
S = { s, 2, 3, 5, 6, 7 }
V = { 4, t }



24

Dijkstra's Shortest Path Algorithm

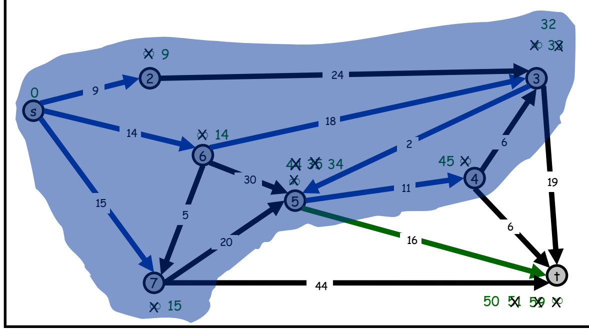
S = { s, 2, 3, 5, 6, 7 }
V = { 4, t }



25

Dijkstra's Shortest Path Algorithm

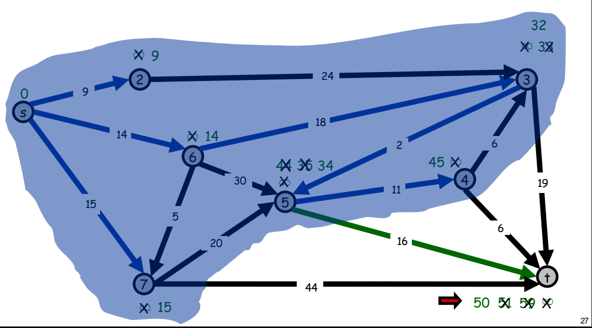
S = { s, 2, 3, 4, 5, 6, 7 }
V = { t }



26

Dijkstra's Shortest Path Algorithm

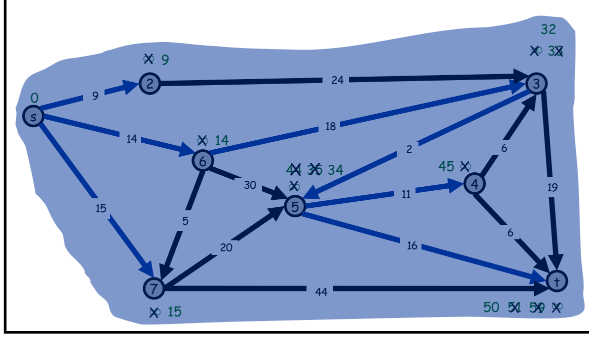
S = { s, 2, 3, 4, 5, 6, 7 }
V = { t }



27

Dijkstra's Shortest Path Algorithm

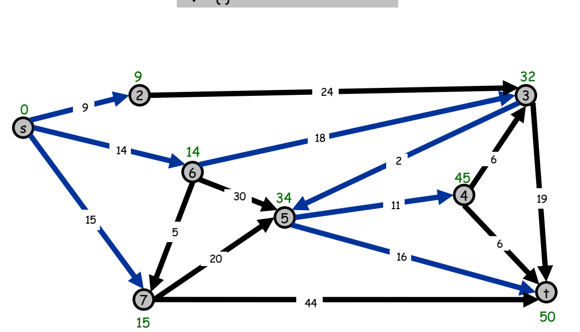
S = { s, 2, 3, 4, 5, 6, 7, t }
V = { }



28

Dijkstra's Shortest Path Algorithm

S = { s, 2, 3, 4, 5, 6, 7, t }
V = { }



29

Legends

- blue blob: S 
- green edges: updates, current `pred[v]`
- blue edges: SSP tree edges
- red arrows: delmin
- green arrows: decrease key
- green numbers: current `d[v]` 34

30

Dijkstra's

```
dijkstra(G, s){
  for each (u in V) { d[u] = infinity }
  d[s] = 0 pred[s] = null
  Q = priority queue of all vertices u keyed by d[u]
  while (Q is not empty) {
    u = extractMin from Q
    for each (v in Adj[u]) {
      if (d[u] + w(u, v) < d[v]) {
        d[v] = d[u] + w(u, v)
        decrease v's key value in Q to d[v]
        pred[v] = u //keeps track of the tree
      }
    }
  }
}
```

31

Time Analysis

- Vertices $V \setminus S$ are stored in a priority queue via key value $d[u]$
- Priority queue operations (binary heap)
  - build $O(n)$
  - delmin (extract min) ⇨ $O(logn)$
  - decrease key ⇨ $O(logn)$
- $T(n,m) = n + n + \sum_{u \in V} (\log n + \deg(u) \cdot \log n)$
- $= 2n + \log n \sum_{u \in V} (1 + \deg(u))$
- $= 2n + \log n \,(n + 2m)$
- $= 2n + n \log n + 2mlogn = O(m \log n)$

32

Data Structures and Cost

	insert	search	delete	findMin	deleteMin	changeKey
Ordered Array	n	$logn$	1^*	1	1^*	n
Ordered List	n	n	1	1	1	n
Unordered Array	1^*	n	1^*	n	n	1
Unordered List	1	n	1	n	n	n
BST	$logn$	$logn$	$logn$	$logn$	$logn$	$logn$
Binary Heap	$logn$	n	$logn$	1	$logn$	$logn$

33

Data Structures and Run Times

Dijkstra's PQ Op	Naïve	Array	Binary Heap	d-way Heap	Fibonacci Heap
Insert	-	1	$logn$	$dlog_d n$	1
ExtractMin	m	n	$logn$	$dlog_d n$	$logn$
ChangeKey	-	1	$logn$	$dlog_d n$	1
IsEmpty	n	1	1	1	1
Total	mn	n^2	$mlogn$	$mlog_{m/n} n$	$m + nlogn$

- $m = 10^{10}$ edges connecting $n = 10^9$ vertices
- Difference of 6 minutes and 3000 years

34

Termination

- Loops
  - outer `while` – V is finite
  - inner `for` – G is finite

35

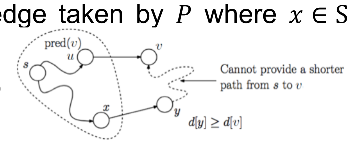
Correctness

- Need to show that $d[v] = \delta(s,v), \forall v \in V$
- Invariant: $d[v] = \delta(s,v), \forall v \in S$
- Proof by induction on $|S|$
  - Base case: $|S| = 1, \delta(s,s) = 0$
  - Assume true for $|S| = k > 1$
  - $|S| = k + 1$

36

Proof

- Let v be the next vertex added to S, along with (u, v)
- $d[v] = \delta(s, u) + w(u, v)$
- Consider any other $s - v$ path P. let (x, y) be the first edge taken by P where $x \in S$ and $y \in V \setminus S$
- $d[x] = \delta(s, x)$
- $d[y] \geq d[v]$
- $\text{len}(P) > \delta(s, x) + w(x, y) = d[y] \geq d[v]$



37