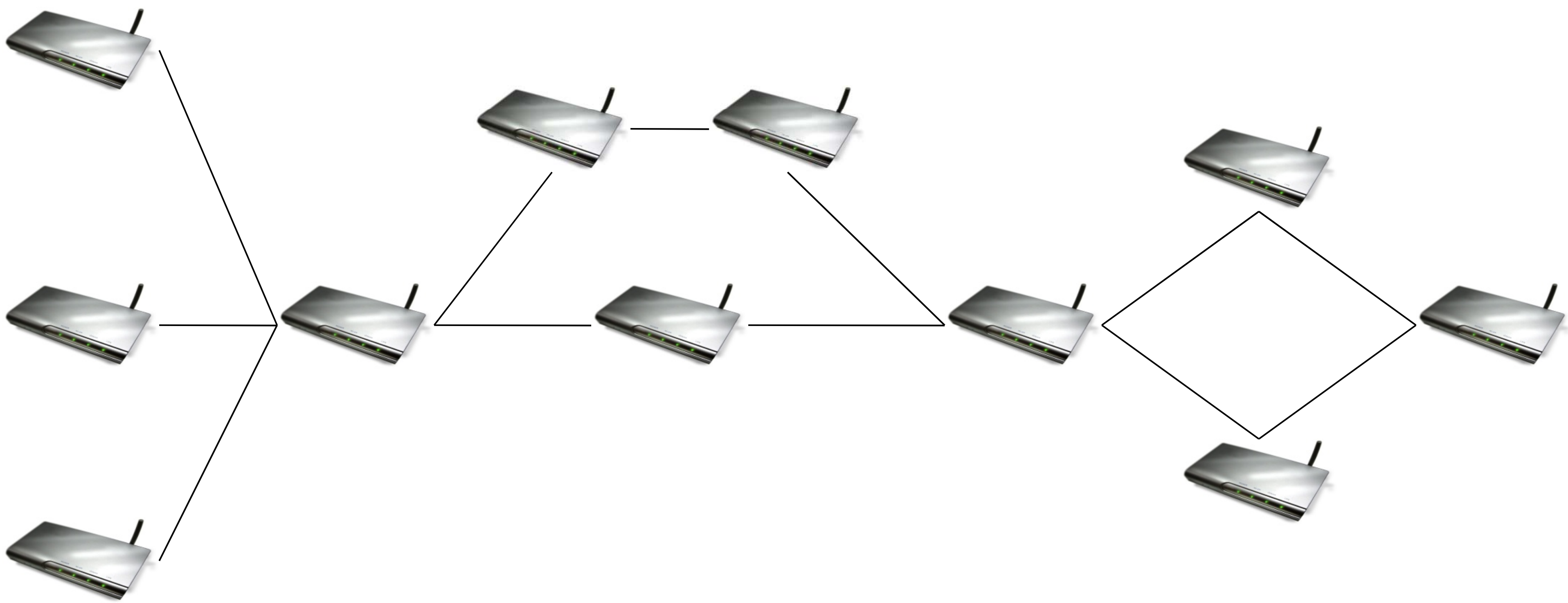
Array Based Betweenness Centrality

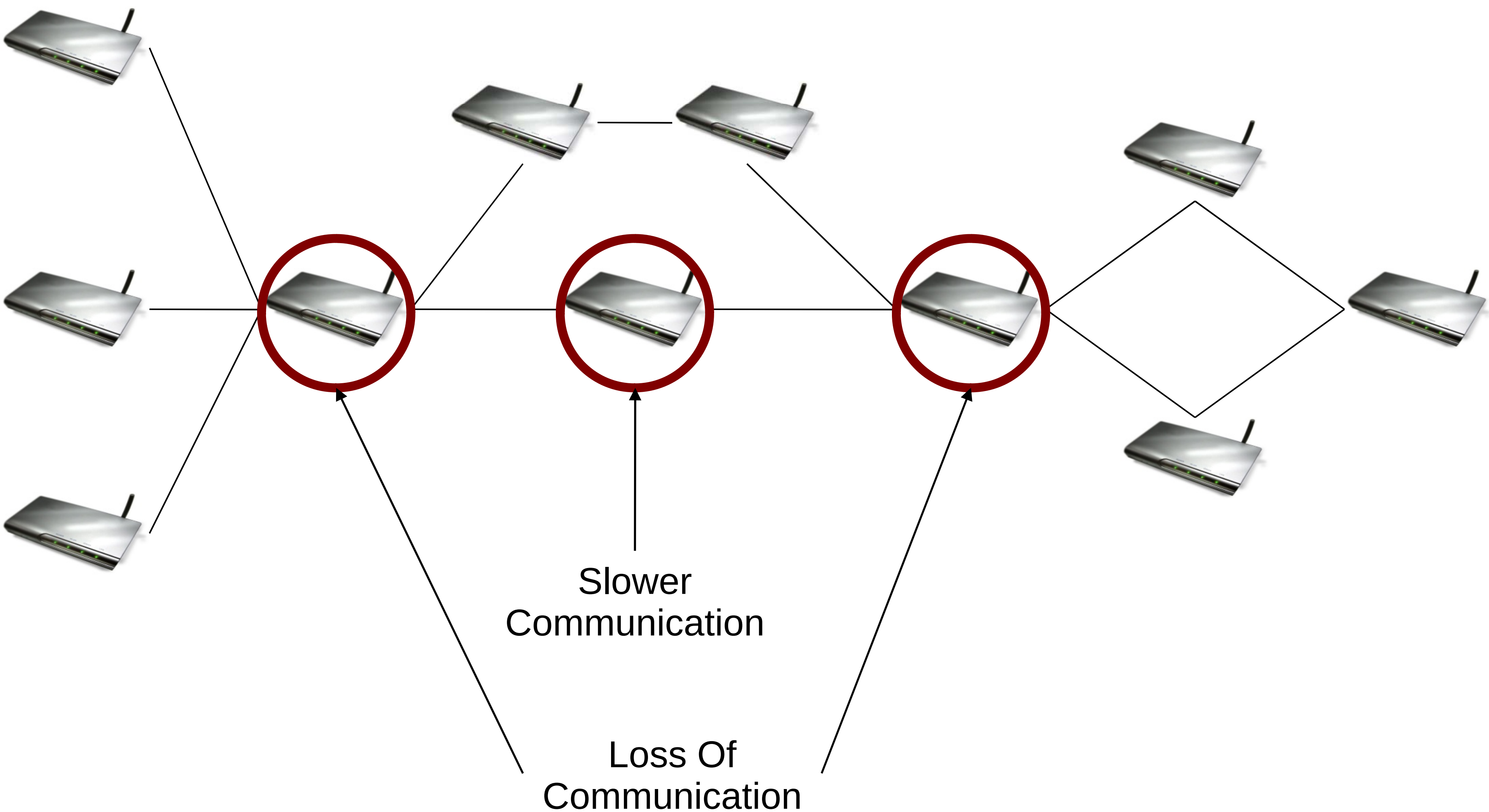
Eric Robinson
Northeastern University
MIT Lincoln Labs

Jeremy Kepner
MIT Lincoln Labs

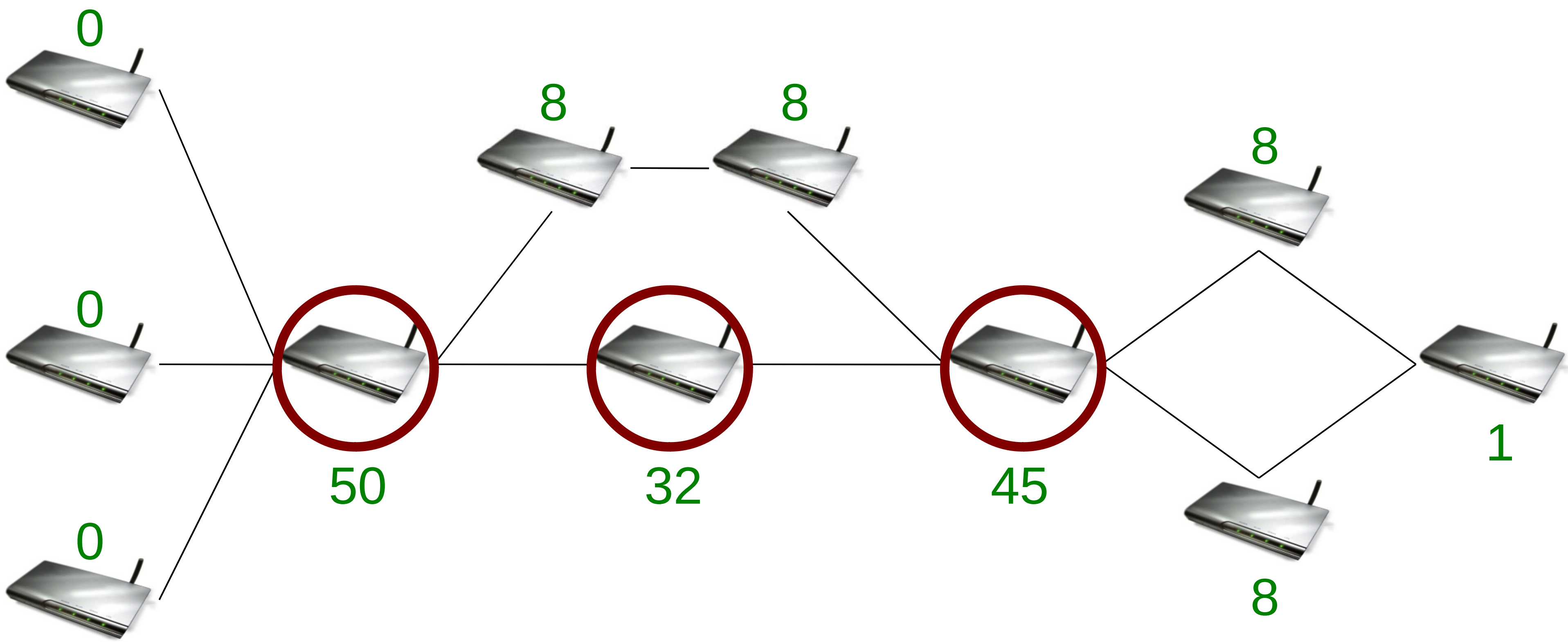
Vertex Betweenness Centrality
Which Vertices are Important?



Vertex Betweenness Centrality
Which Vertices are Important?

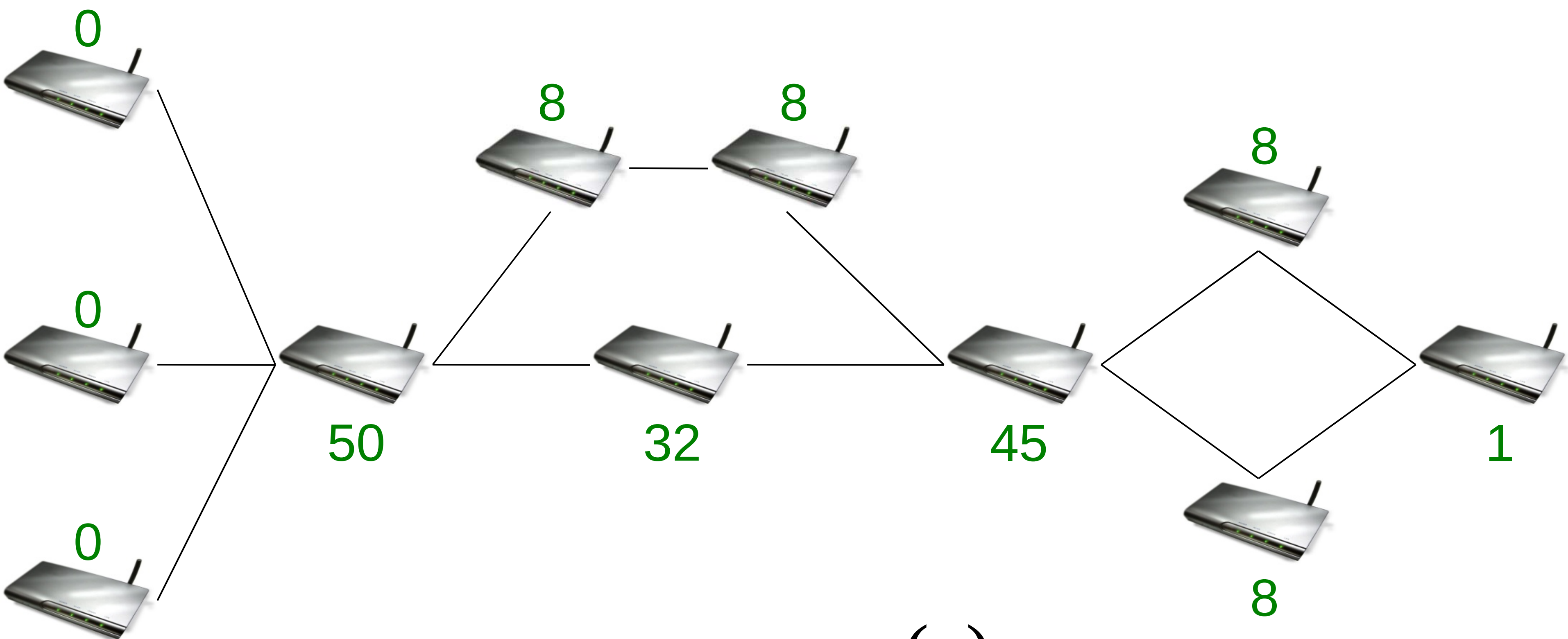


Vertex Betweenness Centrality
How do we Measure Importance?



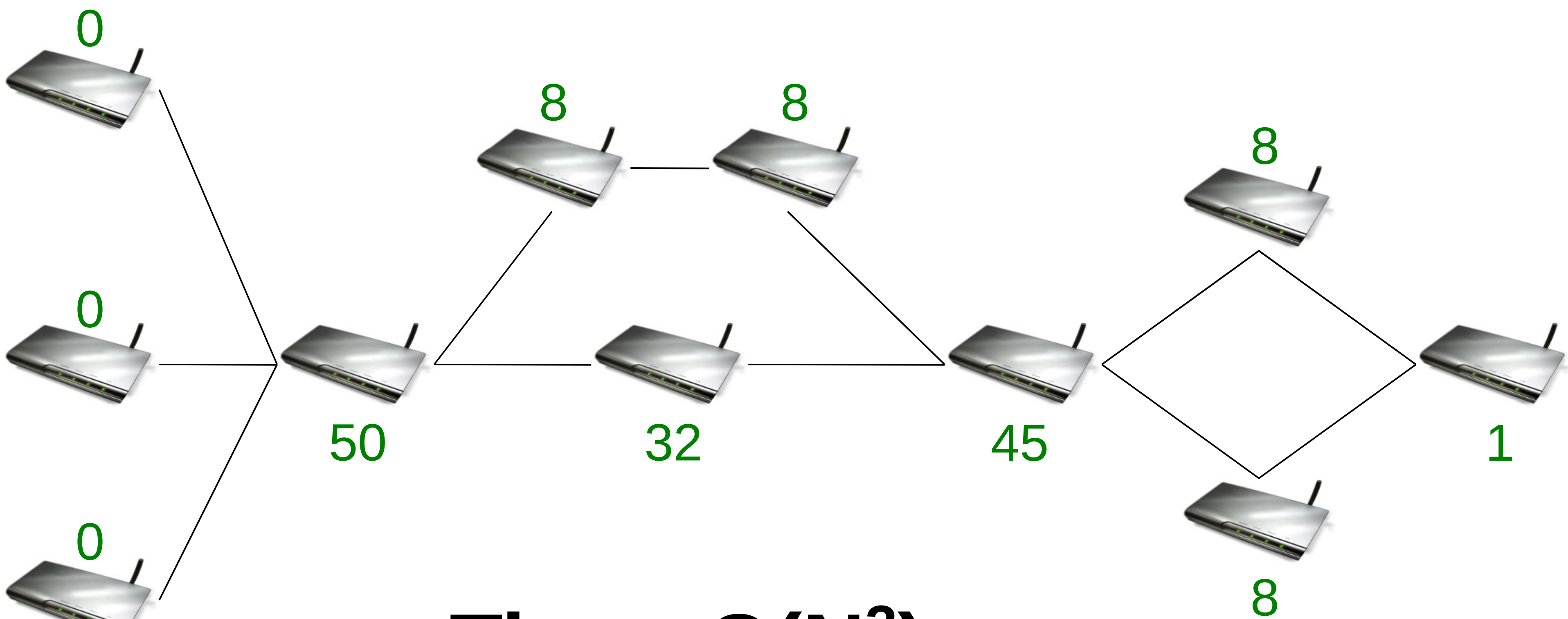
Number of Shortest Paths
Through Node

Vertex Betweenness Centrality Traditional Algorithm



$$C_B(v) = \sum_{s \neq t \neq v \in V} \frac{\sigma_{st}(v)}{\sigma_{st}}$$

Traditional Algorithm Theoretical Time and Space



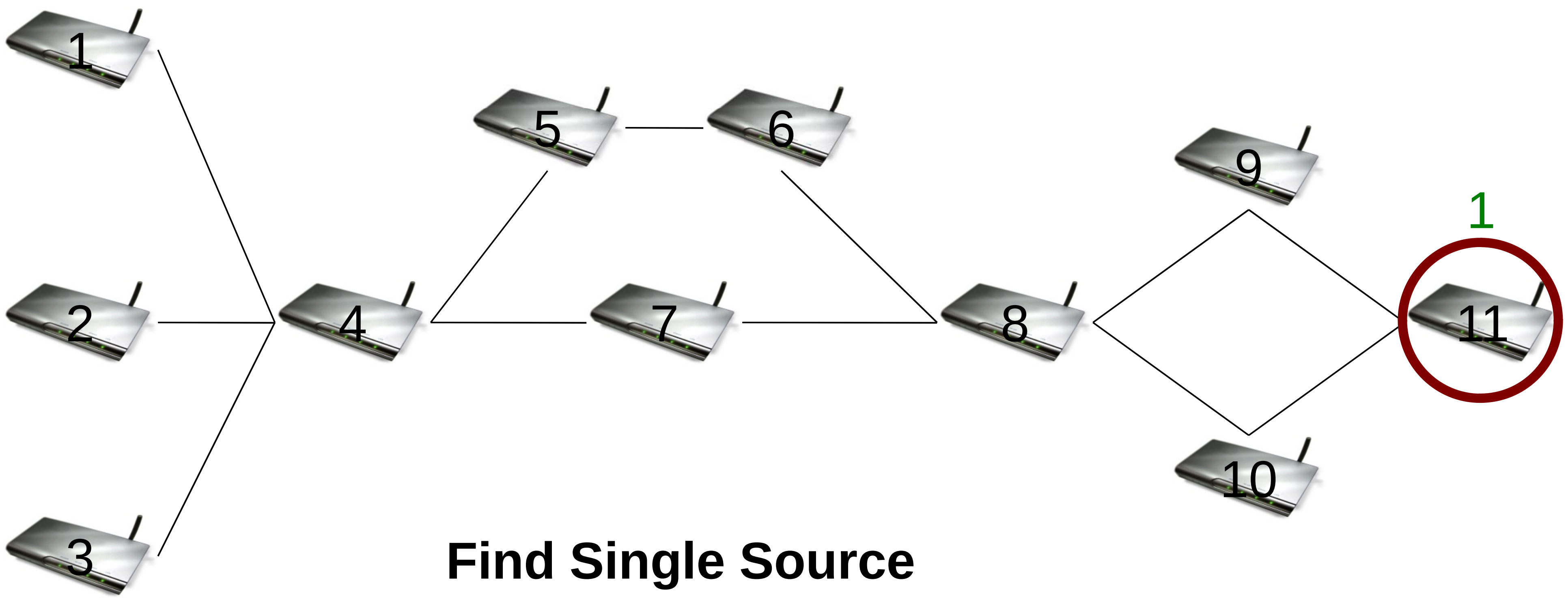
Time: $O(N^3)$
Storage: $O(N^2)$

Vertex Betweenness Centrality Updating Algorithm

- For each starting node:
  - Once you know:
    - The depth of each node in the BFS, D
    - The centrality updates for nodes at depth d, u
    - The shortest path counts from the root, s
  - Can determine centrality of nodes at depth d-1:
    - For each node, v, at depth d-1, it's update is the sum:

$$u_v = \sum_{\substack{(v,w)\in E \\ w\in D(d)}} (1+u_d)\times\frac{s_v}{s_w}$$

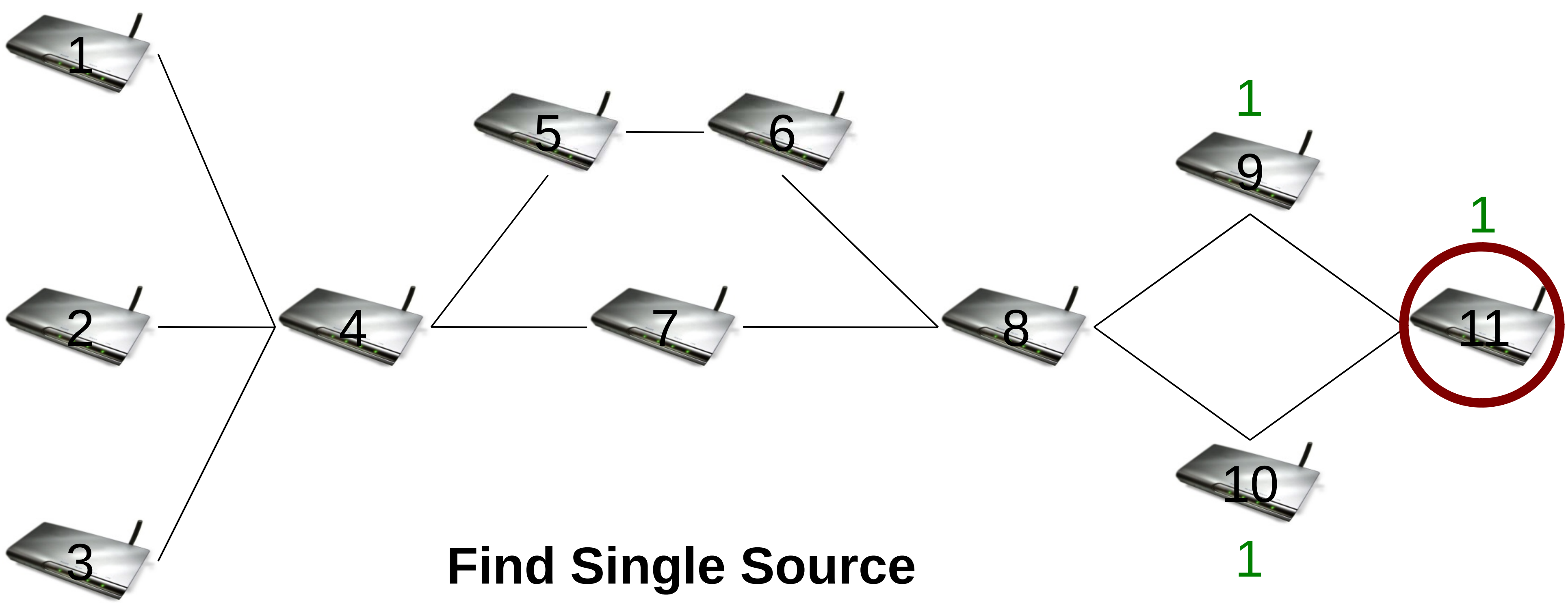
Updating Algorithm
An Example



Find Single Source
Shortest Path Counts

O(N + M)

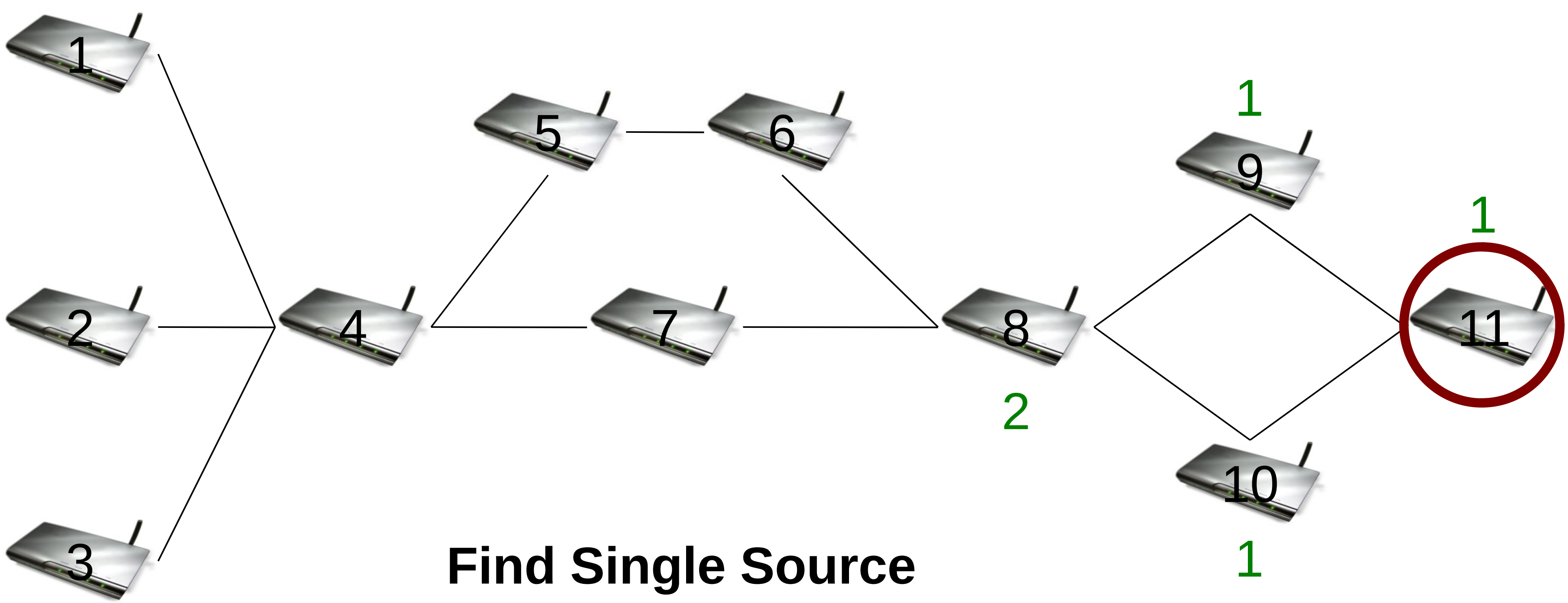
Updating Algorithm
An Example



Find Single Source Shortest Path Counts

O(N + M)

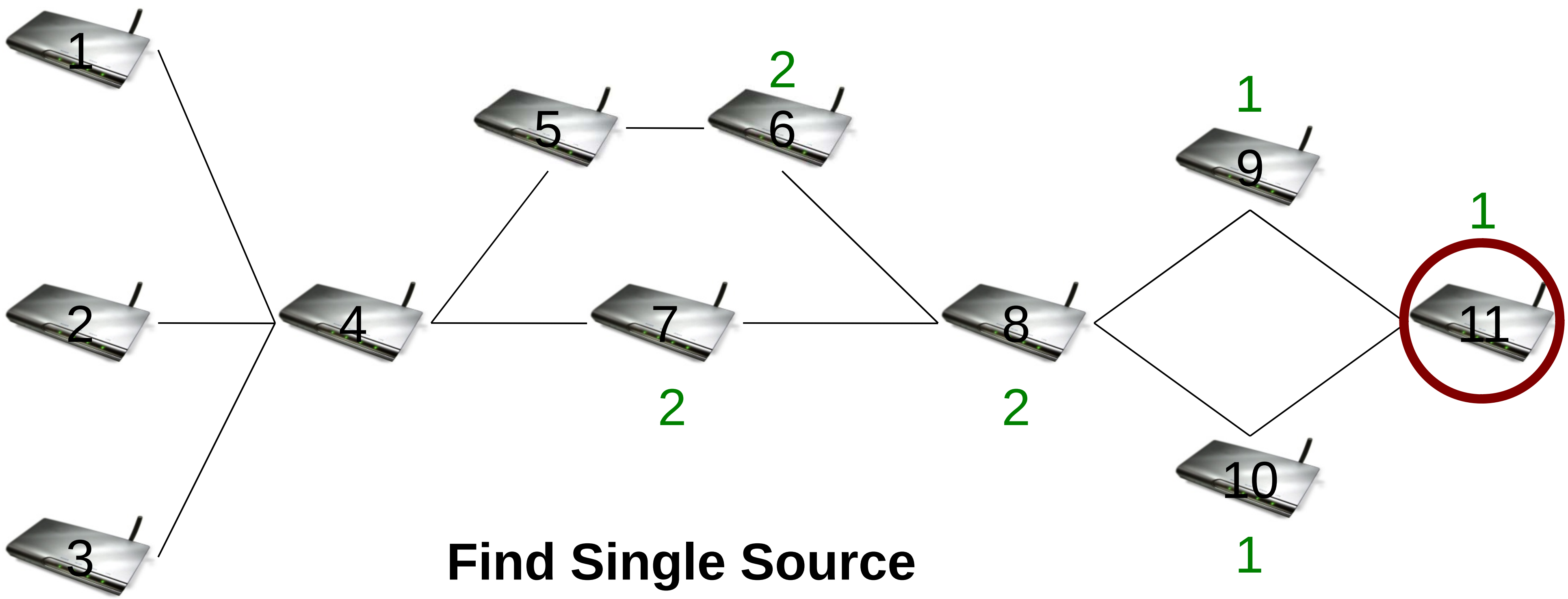
Updating Algorithm
An Example



Find Single Source Shortest Path Counts

O(N + M)

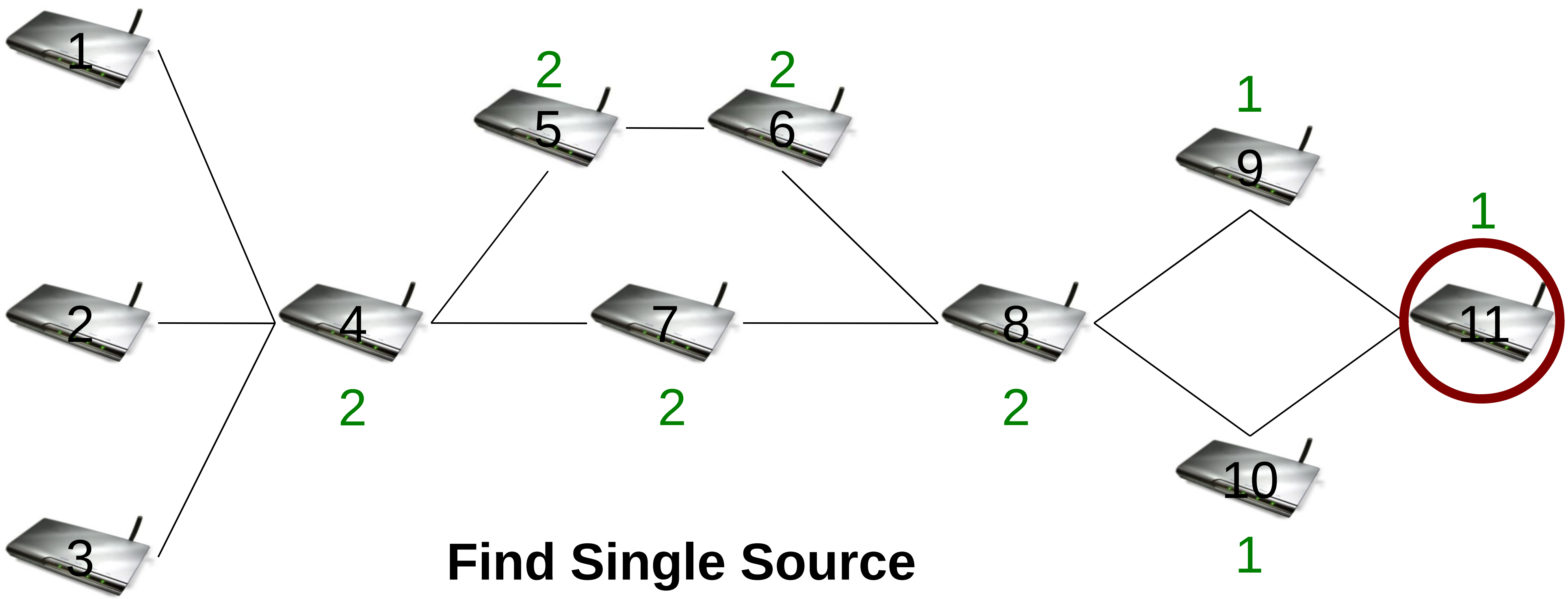
Updating Algorithm
An Example



Find Single Source
Shortest Path Counts

O(N + M)

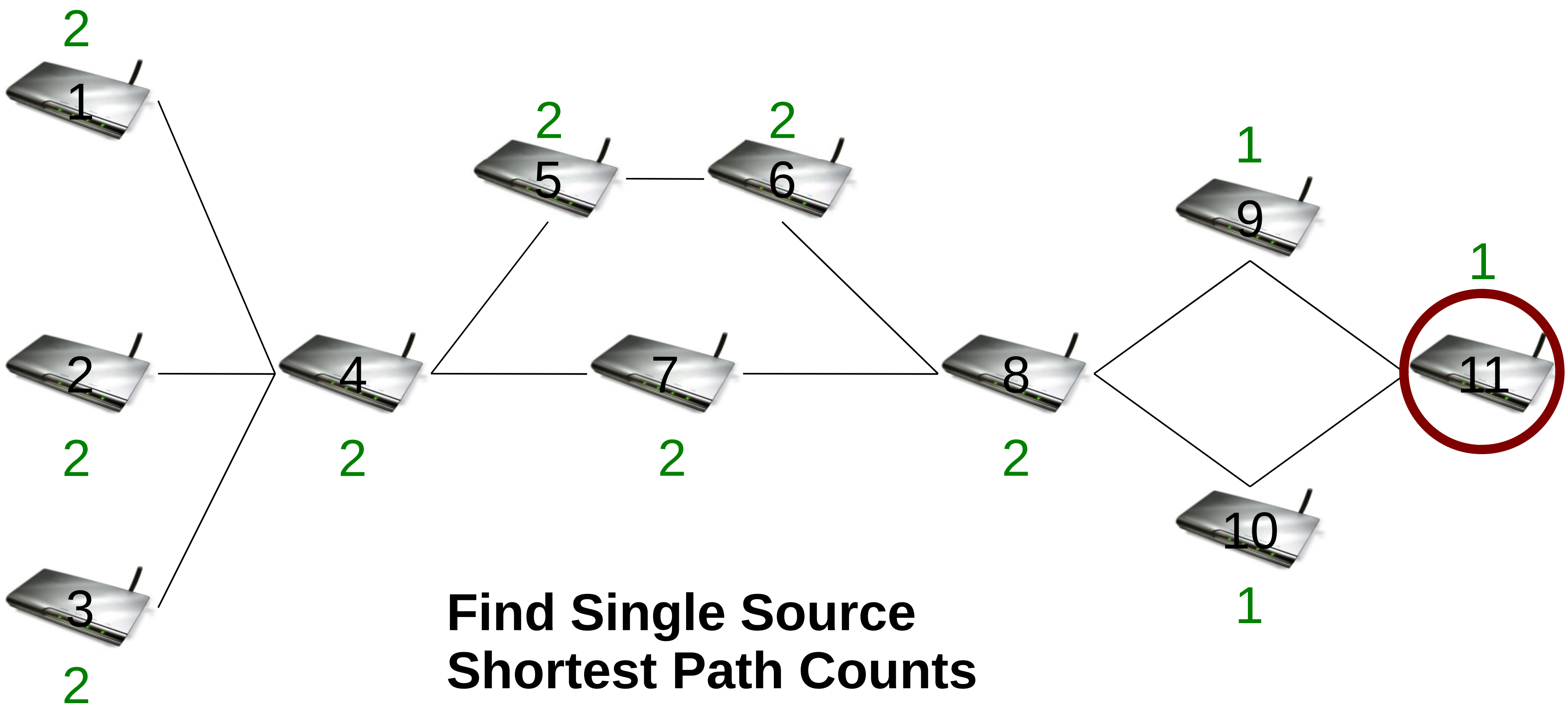
Updating Algorithm
An Example



Find Single Source Shortest Path Counts

O(N + M)

Updating Algorithm
An Example



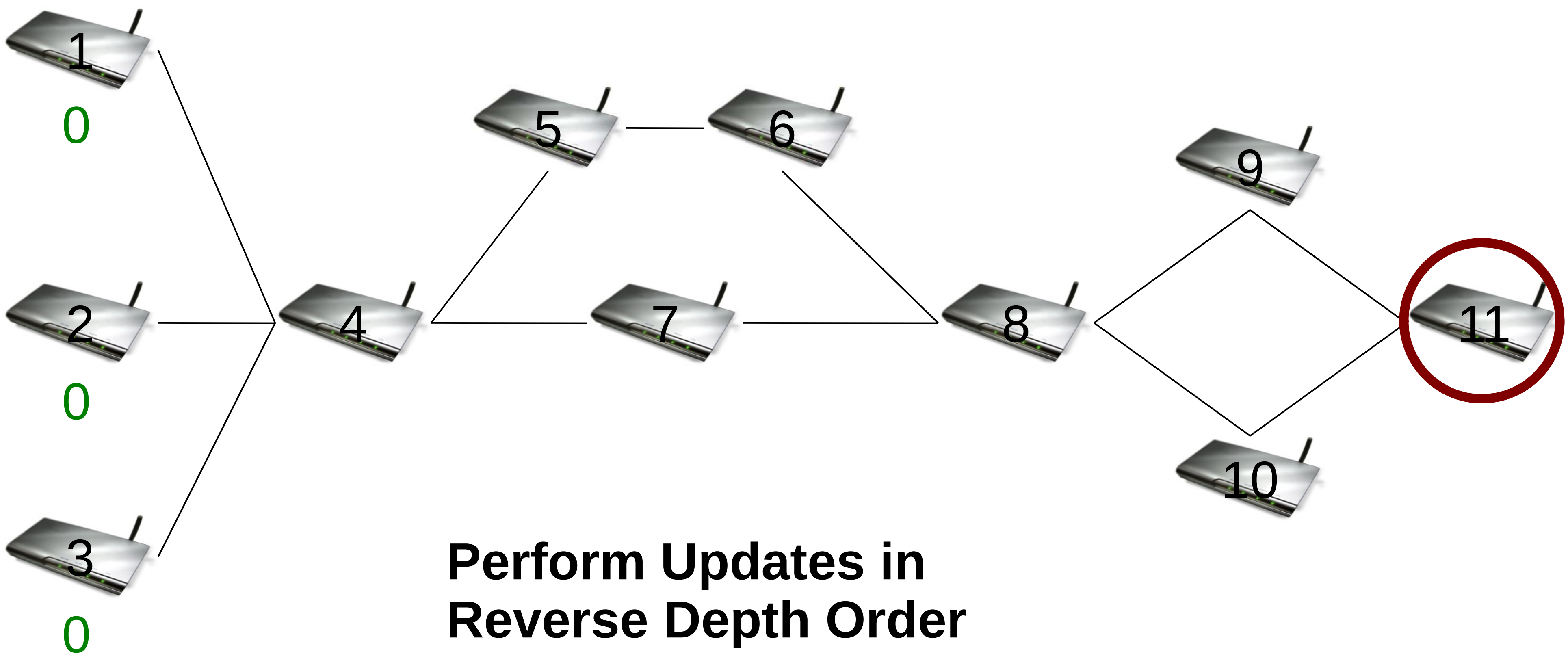
Find Single Source Shortest Path Counts

O(N + M)

Updating Algorithm
An Example

Shortest Paths:

1	2	3	4	5	6	7	8	9	10	11
2	2	2	2	2	2	2	2	1	1	1



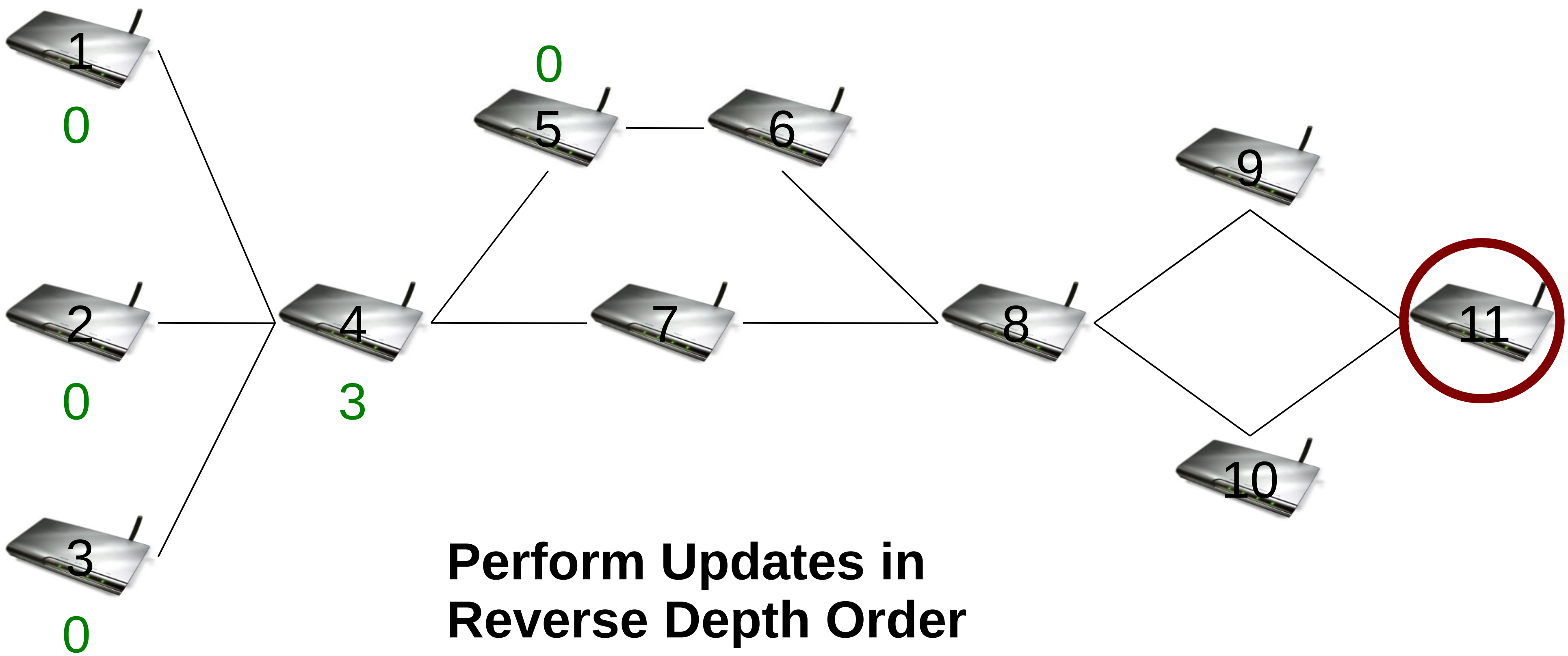
Perform Updates in Reverse Depth Order

O(N + M)

Updating Algorithm
An Example

Shortest Paths:

1	2	3	4	5	6	7	8	9	10	11
2	2	2	2	2	2	2	2	1	1	1



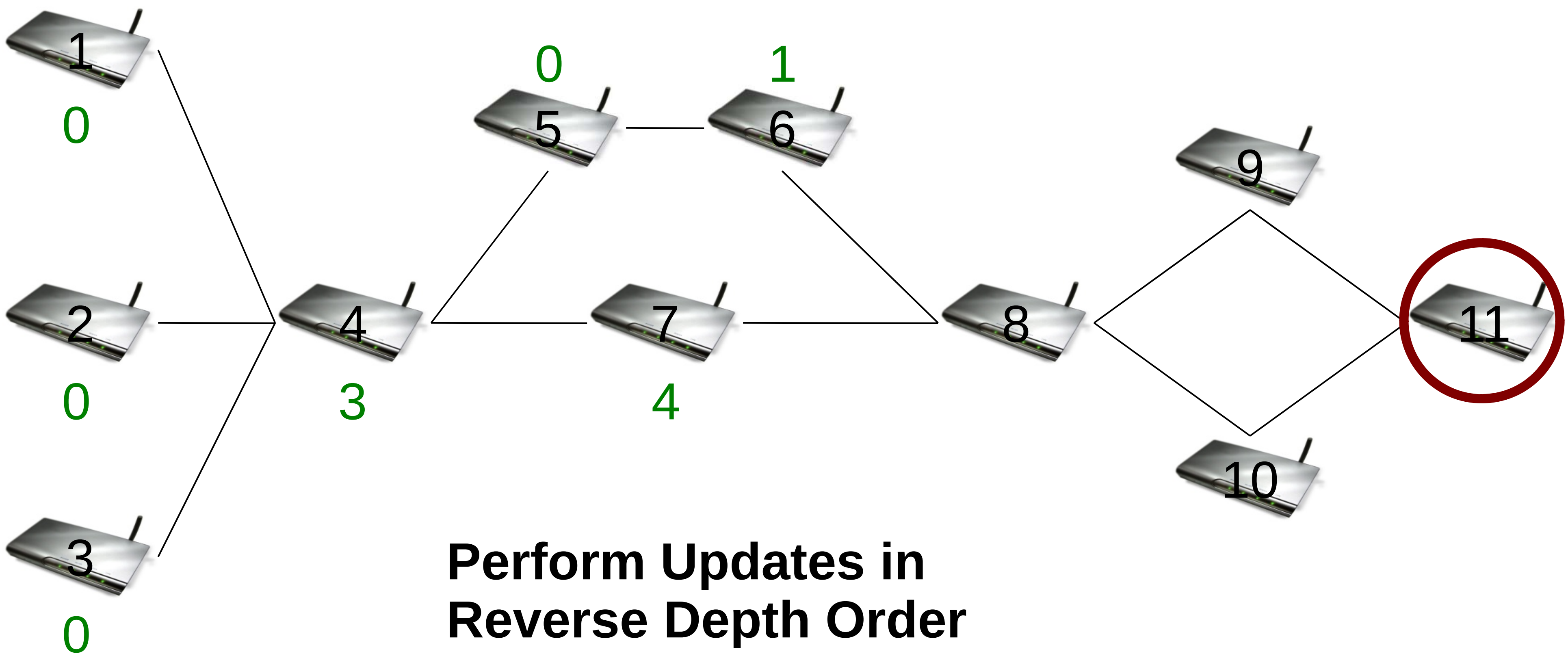
Perform Updates in Reverse Depth Order

O(N + M)

Updating Algorithm
An Example

Shortest Paths:

1	2	3	4	5	6	7	8	9	10	11
2	2	2	2	2	2	2	2	1	1	1



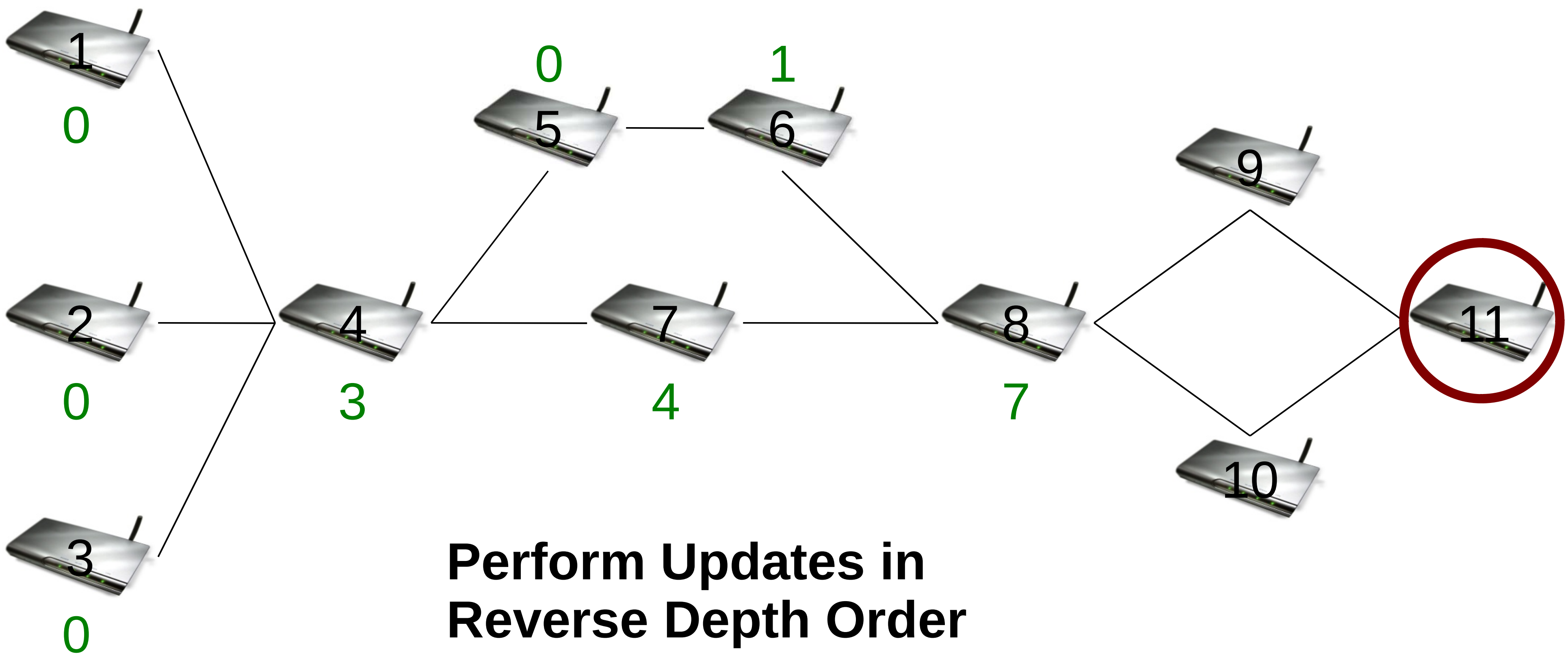
Perform Updates in Reverse Depth Order

O(N + M)

Updating Algorithm
An Example

Shortest Paths:

1	2	3	4	5	6	7	8	9	10	11
2	2	2	2	2	2	2	2	1	1	1



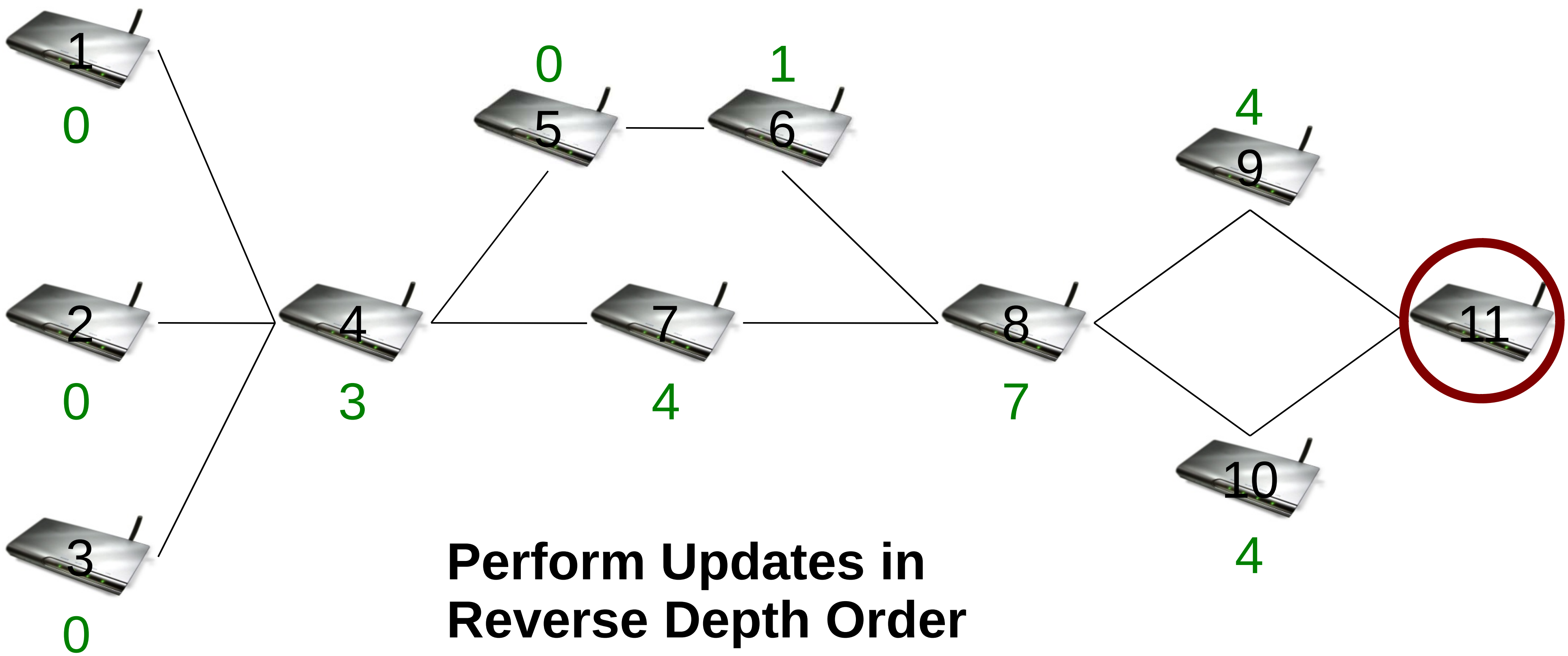
Perform Updates in Reverse Depth Order

O(N + M)

Updating Algorithm
An Example

Shortest Paths:

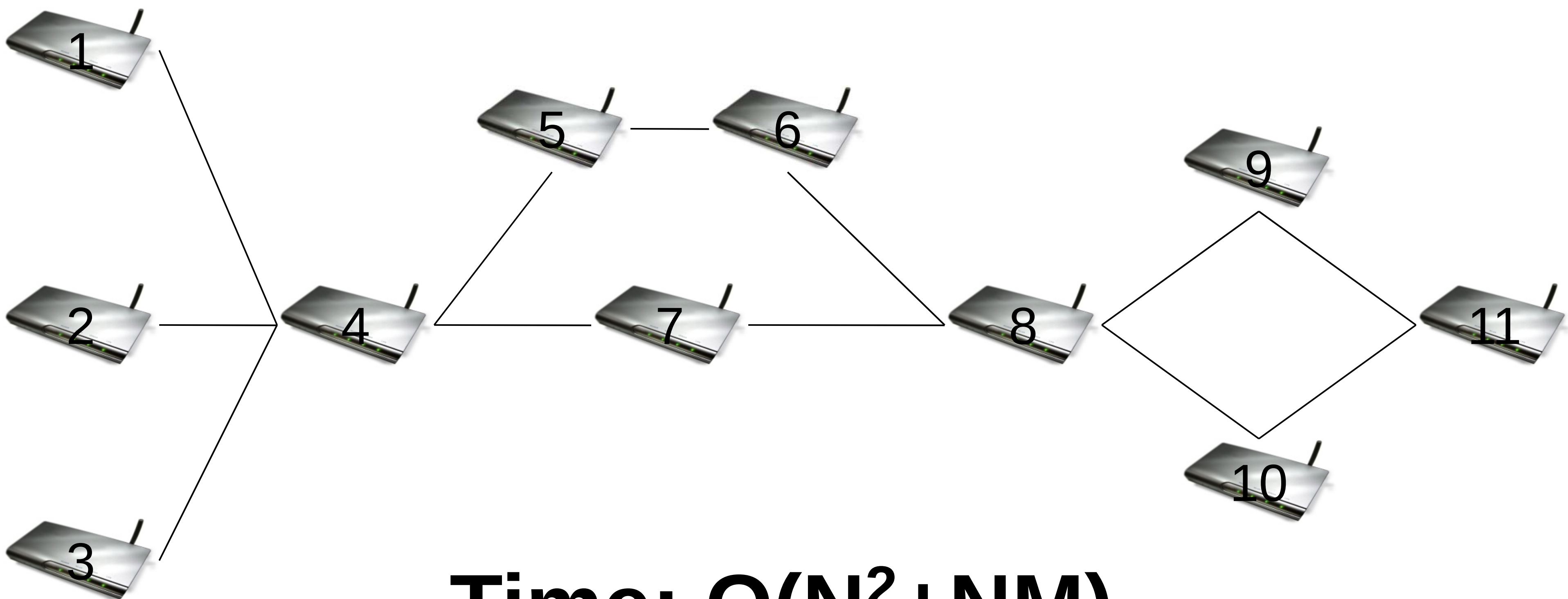
1	2	3	4	5	6	7	8	9	10	11
2	2	2	2	2	2	2	2	1	1	1



Perform Updates in Reverse Depth Order

O(N + M)

Updating Algorithm
Theoretical Time and Space



Time: $O(N^2+NM)$

Storage: $O(N+M)$

Updating Algorithm
Single Processor

Algorithm 1: Betweenness centrality in unweighted graphs

```
C_B[v] ← 0, v ∈ V;
for s ∈ V do
    S ← empty stack;
    P[w] ← empty list, w ∈ V;
    σ[t] ← 0, t ∈ V;   σ[s] ← 1;
    d[t] ← −1, t ∈ V;   d[s] ← 0;
    Q ← empty queue;
    enqueue s → Q;
    while Q not empty do
        dequeue v ← Q;
        push v → S;
        foreach neighbor w of v do
            // w found for the first time?
            if d[w] < 0 then
                enqueue w → Q;
                d[w] ← d[v] + 1;
            end
            // shortest path to w via v?
            if d[w] = d[v] + 1 then
                σ[w] ← σ[w] + σ[v];
                append v → P[w];
            end
        end
    end
    δ[v] ← 0, v ∈ V;
    // S returns vertices in order of non-increasing distance from s
    while S not empty do
        pop w ← S;
        for v ∈ P[w] do δ[v] ← δ[v] + σ[v]/σ[w] · (1 + δ[w]);
        if w ≠ s then C_B[w] ← C_B[w] + δ[w];
    end
end
```

Variables:		Storage:
V :	set of vertices	O(M+N)
d :	depth of vertices	O(N)
Q :	BFS queue	O(N)
P :	shortest path parents	O(M+N)
σ :	number of paths	O(N)
S :	order seen	O(N)
δ :	centrality update	O(N)

Storage: O(M+N)
Time: O(MN + N^2)

Updating Algorithm
P processors

Algorithm 1: Betweenness centrality in unweighted graphs

```
C_B[v] ← 0, v ∈ V;
for s ∈ V do
    S ← empty stack;
    P[w] ← empty list, w ∈ V;
    σ[t] ← 0, t ∈ V;   σ[s] ← 1;
    d[t] ← −1, t ∈ V;   d[s] ← 0;
    Q ← empty queue;
    enqueue s → Q;
    while Q not empty do
        dequeue v ← Q;
        push v → S;
        foreach neighbor w of v do
            // w found for the first time?
            if d[w] < 0 then
                enqueue w → Q;
                d[w] ← d[v] + 1;
            end
            // shortest path to w via v?
            if d[w] = d[v] + 1 then
                σ[w] ← σ[w] + σ[v];
                append v → P[w];
            end
        end
    end
    δ[v] ← 0, v ∈ V;
    // S returns vertices in order of non-increasing distance from s
    while S not empty do
        pop w ← S;
        for v ∈ P[w] do δ[v] ← δ[v] + σ[v]/σ[w] · (1 + δ[w]);
        if w ≠ s then C_B[w] ← C_B[w] + δ[w];
    end
end
```

For each vertex, in parallel

Variables:		**Storage:**
V :	set of vertices	O(PM+PN)
d :	depth of vertices	O(PN)
Q :	BFS queue	O(PN)
P :	shortest path parents	O(PM+PN)
σ :	number of paths	O(PN)
S :	order seen	O(PN)
δ :	centrality update	O(PN)

Storage: O(PM+PN)
Time: $O((MN + N^2)/P)$

Updating Algorithm
Array Based Version

$b = \textsc{BetweennessCentrality}(G = A : \mathbb{B}^{N_V \times N_V})$

$\mathbf{b} = 0$
for $1 \leq r \leq N_V$
  do
    $d = 0$
    $\mathbf{S} = 0$
    $\mathbf{p} = 0,\ p_r = 1$
    $\mathbf{f} = \mathbf{a}_{r,:}$
    while $\mathbf{f} \neq 0$
      do
        $d = d + 1$
        $\mathbf{p} = \mathbf{p} + \mathbf{f}$
        $\mathbf{s}_{d,:} = \mathbf{f}$
        $\mathbf{f} = \mathbf{f}\mathbf{A} \times \neg\mathbf{p}$
    while $d \geq 2$
      do
        $\mathbf{w} = \mathbf{s}_{d,:} \times (1 + \mathbf{u}) \div \mathbf{p}$
        $\mathbf{w} = \mathbf{A}\mathbf{w}$
        $\mathbf{w} = \mathbf{w} \times \mathbf{s}_{d-1,:} \times \mathbf{p}$
        $\mathbf{u} = \mathbf{u} + \mathbf{w}$
        $d = d - 1$
    $\mathbf{b} = \mathbf{b} + \mathbf{u}$

Variables:		**Storage:**	
A :	sparse adjacency matrix	$B^{S(NxN)}$	O(M+N)
f :	sparse fringe vector	$Z^{S(N)}$	O(N)
p :	shortest path vector	Z^{N}	O(N)
S :	sparse depth matrix	$B^{S(NxN)}$	O(N)
u :	centrality update vector	R^{N}	O(N)

Storage: O(M+N)
Time: O(MN + N^2)

Updating Algorithm
Array Based Version

$b = \textsc{BetweennessCentrality}(G = A : \mathbb{B}^{N_V \times N_V})$

$\mathbf{b} = 0$
for $1 \le r \le N_V$
    do
        $d = 0$
        $\mathbf{S} = 0$
        $\mathbf{p} = 0,\ p_r = 1$
        $\mathbf{f} = \mathbf{a}_{r,:}$
        while $\mathbf{f} \neq 0$
            do
                $d = d + 1$
                $\mathbf{p} = \mathbf{p} + \mathbf{f}$
                $\mathbf{s}_{d,:} = \mathbf{f}$
                $\mathbf{f} = \mathbf{fA} \times \neg\mathbf{p}$
        while $d \ge 2$
            do
                $\mathbf{w} = \mathbf{s}_{d,:} \times (1 + \mathbf{u}) \div \mathbf{p}$
                $\mathbf{w} = \mathbf{Aw}$
                $\mathbf{w} = \mathbf{w} \times \mathbf{s}_{d-1,:} \times \mathbf{p}$
                $\mathbf{u} = \mathbf{u} + \mathbf{w}$
                $d = d - 1$
        $\mathbf{b} = \mathbf{b} + \mathbf{u}$

Variables:

- A : sparse adjacency matrix
- f : sparse fringe vector
- p : shortest path vector
- S : sparse depth matrix
- u : centrality update vector

Discover Paths:

f = fA
f = f .* ¬p
p = p + f
S_d = boolean(f)

Updating Algorithm
Array Based Version

$b = \textsc{BetweennessCentrality}(G = A : \mathbb{B}^{N_V \times N_V})$

$\mathbf{b} = 0$
for $1 \leq r \leq N_V$
  do
    $d = 0$
    $\mathbf{S} = 0$
    $\mathbf{p} = 0,\ p_r = 1$
    $\mathbf{f} = \mathbf{a}_{r,:}$
    while $\mathbf{f} \neq 0$
      do
        $d = d + 1$
        $\mathbf{p} = \mathbf{p} + \mathbf{f}$
        $\mathbf{s}_{d,:} = \mathbf{f}$
        $\mathbf{f} = \mathbf{f}\mathbf{A} \times \neg\mathbf{p}$
    while $d \geq 2$
      do
        $\mathbf{w} = \mathbf{s}_{d,:} \times (1 + \mathbf{u}) \div \mathbf{p}$
        $\mathbf{w} = \mathbf{A}\mathbf{w}$
        $\mathbf{w} = \mathbf{w} \times \mathbf{s}_{d-1,:} \times \mathbf{p}$
        $\mathbf{u} = \mathbf{u} + \mathbf{w}$
        $d = d - 1$
    $\mathbf{b} = \mathbf{b} + \mathbf{u}$

Variables:

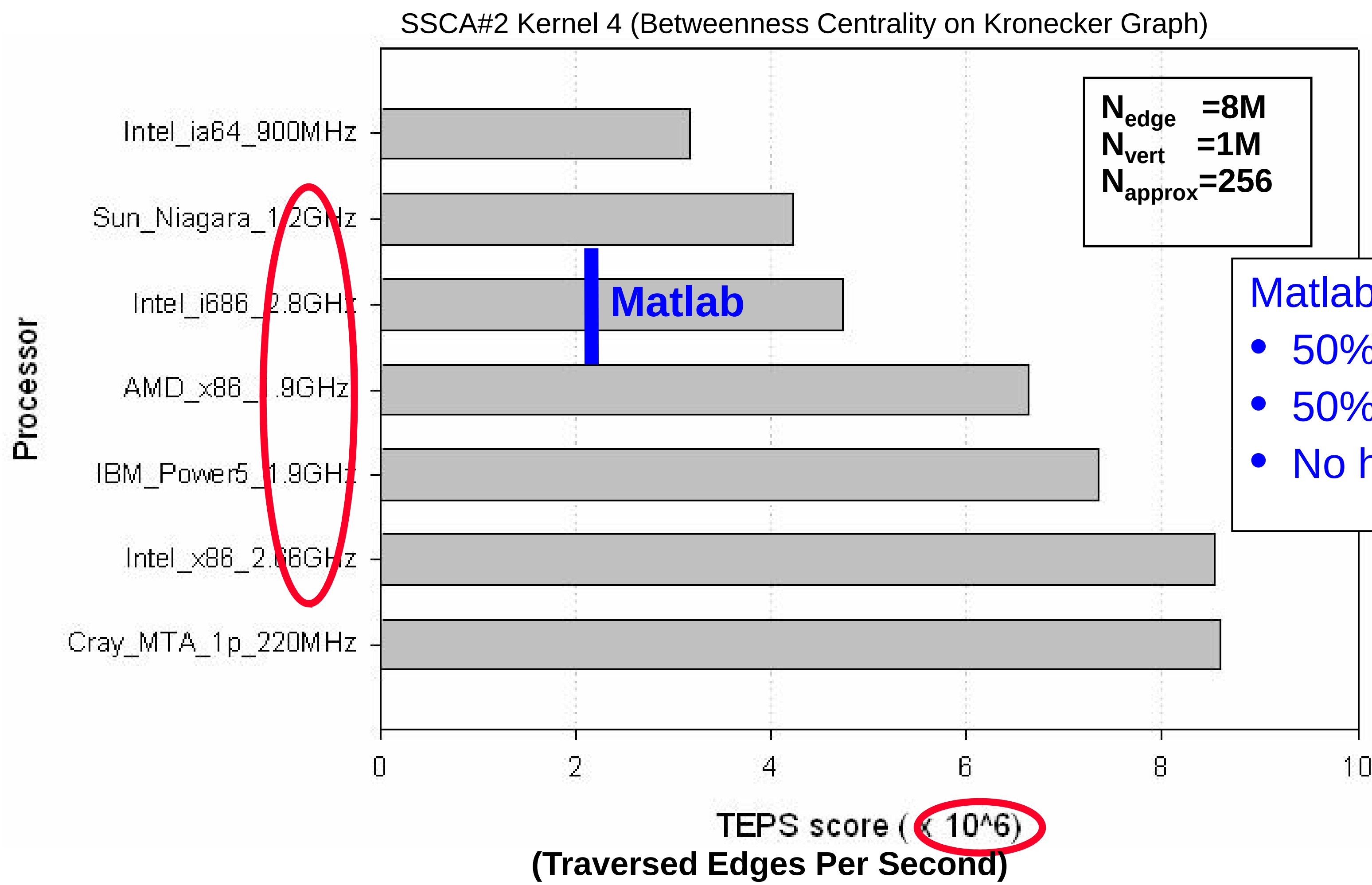
- A : sparse adjacency matrix
- f : sparse fringe vector
- p : shortest path vector
- S : sparse depth matrix
- u : centrality update vector

Update Centralities:

$w = S_d .* (1+u) ./ p$
$w = Aw$
$w = w .* S_{d-1} .* p$
$u = u + w$

Array Based Version
Single Processor Performance

Data Courtesy of Prof. David Bader & Kamesh Madduri (Georgia Tech)



Matlab achieves

- 50% of C
- 50% of sparse matmul
- No hidden gotchas

Array Based Version
Why is it Useful?

$b = \textsc{BetweennessCentrality}(G = A : \mathbb{B}^{N_V \times N_V})$

$\mathbf{b} = 0$
for $1 \leq r \leq N_V$
 do
  $d = 0$
  $\mathbf{S} = 0$
  $\mathbf{p} = 0,\ p_r = 1$
  $\mathbf{f} = \mathbf{a}_{r,:}$
  while $\mathbf{f} \neq 0$
   do
    $d = d + 1$
    $\mathbf{p} = \mathbf{p} + \mathbf{f}$
    $\mathbf{s}_{d,:} = \mathbf{f}$
    $\mathbf{f} = \mathbf{f}\mathbf{A} \times \neg\mathbf{p}$
  while $d \geq 2$
   do
    $\mathbf{w} = \mathbf{s}_{d,:} \times (1 + \mathbf{u}) \div \mathbf{p}$
    $\mathbf{w} = \mathbf{A}\mathbf{w}$
    $\mathbf{w} = \mathbf{w} \times \mathbf{s}_{d-1,:} \times \mathbf{p}$
    $\mathbf{u} = \mathbf{u} + \mathbf{w}$
    $d = d - 1$
  $\mathbf{b} = \mathbf{b} + \mathbf{u}$

- Linear Performance within:
  Factor of 2 of C code
- **Fewer Lines of Code**
  (More work behind-the-scenes)
- Natural Implementation in:
  - **Matlab**
  - Maple
  - ...
- Processes full depth at a time:
  - **Low-level parallelism**

Array Based Version
P Processors

$b = \textsc{BetweennessCentrality}(G = A : \mathbb{B}^{N_V \times N_V})$

$\mathbf{b} = 0$
for $1 \leq r \leq N_V$
  do
    $d = 0$
    $\mathbf{S} = 0$
    $\mathbf{p} = 0,\ p_r = 1$
    $\mathbf{f} = \mathbf{a}_{r,:}$
    while $\mathbf{f} \neq 0$
      do
        $d = d + 1$
        $\mathbf{p} = \mathbf{p} + \mathbf{f}$
        $\mathbf{s}_{d,:} = \mathbf{f}$
        $\mathbf{f} = \mathbf{f}\mathbf{A} \times \neg\mathbf{p}$
    while $d \geq 2$
      do
        $\mathbf{w} = \mathbf{s}_{d,:} \times (1 + \mathbf{u}) \div \mathbf{p}$
        $\mathbf{w} = \mathbf{A}\mathbf{w}$
        $\mathbf{w} = \mathbf{w} \times \mathbf{s}_{d-1,:} \times \mathbf{p}$
        $\mathbf{u} = \mathbf{u} + \mathbf{w}$
        $d = d - 1$
    $\mathbf{b} = \mathbf{b} + \mathbf{u}$

Storage: O(M+N)
Time: $O((MN + N^2)/P)$

Discover Paths in Parallel

Update Centralities in Parallel

Array Based P Processor Version Why is it Useful?

$b = \textsc{BetweennessCentrality}(G = A : \mathbb{B}^{N_V \times N_V})$

$\mathbf{b} = 0$
for $1 \leq r \leq N_V$
  do
    $d = 0$
    $\mathbf{S} = 0$
    $\mathbf{p} = 0,\ p_r = 1$
    $\mathbf{f} = \mathbf{a}_{r,:}$
    while $\mathbf{f} \neq 0$
      do
        $d = d + 1$
        $\mathbf{p} = \mathbf{p} + \mathbf{f}$
        $\mathbf{s}_{d,:} = \mathbf{f}$
        $\mathbf{f} = \mathbf{f}\mathbf{A} \times \neg\mathbf{p}$
    while $d \geq 2$
      do
        $\mathbf{w} = \mathbf{s}_{d,:} \times (1 + \mathbf{u}) \div \mathbf{p}$
        $\mathbf{w} = \mathbf{A}\mathbf{w}$
        $\mathbf{w} = \mathbf{w} \times \mathbf{s}_{d-1,:} \times \mathbf{p}$
        $\mathbf{u} = \mathbf{u} + \mathbf{w}$
        $d = d - 1$
    $\mathbf{b} = \mathbf{b} + \mathbf{u}$

- **Performance Currently Untested**
- **Memory per machine Scales as Expected**
- **Fewer Lines of Code**
  (More work behind-the-scenes)
- Natural Implementation in:
  - **PMatlab**
  - StarP
  - ...

Matrix Based Version
How does it work?

Choose a vertex block size V
(Optimal Size in tests, V = 16)

Variables:		**Storage:**	
A :	sparse adjacency matrix	$B^{S(NxN)}$	O(M+N)
f :	sparse fringe vector	$Z^{S(VxN)}$	O(VN)
p :	shortest path vector	$Z^{(VxN)}$	O(VN)
S :	sparse depth matrix	$B^{S(VxNxN)}$	O(VN)
u :	centrality update vector	$R^{(VxN)}$	O(VN)

Time: **$O(N^2+MN)$**

Storage: **O(VN+M)**

Acknowledgements

- Original Updating Algorithm: Ulrik Brandes
- Parallel Updating Algorithm: David Bader
Kamesh Madduri
- Collaboration at Lincoln Labs: Jeremy Kepner
- LA Graph Algorithms: Jeremy Fineman
Crystal Kahn