

Algorithms

Xiaojuan Cai
cxj@sjtu.edu.cn

Fall semester 2015

Course Overview

- Why study Algorithms
- Outline of the course
- Course Policies

What is Algorithm?

The word ‘**algorithm**’ is derived from the name of



http://en.wikipedia.org/wiki/File:Abu_Abdullah_Muhammad_bin_Musa_al-Khwarizmi_edit.png

Muhammad ibn Mūsā al-Khwārizmī (780?-850?)

*a Muslim mathematician whose works introduced Arabic numerals and algebraic concepts to Western mathematics.
(American Heritage Dictionary)*

What is Algorithm?

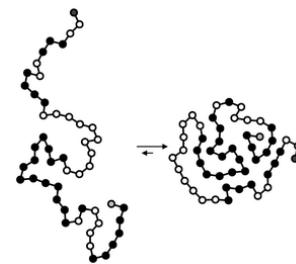
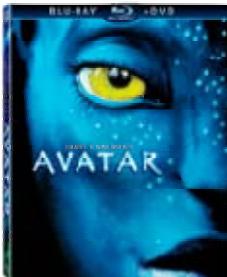
- An **algorithm** is a procedure that consists of
 - a finite set of instructions which,
 - given an **input** from some set of possible inputs,
 - enables us to obtain an **output** through a systematic execution of the instructions
 - that **terminates** in a finite number of steps.

Why study Algorithms?

1. Their impact is broad and far-reaching.

Internet, Biology, Computers, Computer graphics, Security, Multimedia, Physics

Google
YAHOO!
bing



“Computer Science is the study of **algorithms**. ”

-- Donald E. Knuth

Why study Algorithms?

2. to become a proficient programmer



“ Algorithms + Data Structures = Programs. ”

-- *Niklaus Wirth*

“Bad programmers worry about the code. Good programmers worry about **data structures** and their relationships.”

-- *Linus Torvalds*



Why study Algorithms?

3. For intellectual stimulation, be a wise person

“有人天生喜欢“遍历”，踏遍千山万水，遍享万种风情.....

有人一生“贪婪”，眼界不宽，及时行乐；

有人注定适用“穷搜”，辛辛苦苦勤勤恳恳一辈子，付出很多，收获有限；

有人善用“时空权衡”，用最少的时间办最多的事情，的确精明；

有人会“分治”，再多的难题也能迎刃而解；

有人常“回溯”，错的太多，后悔太多；

有的人压根没有算法，于是盲目生活，盲目做事，最后所获无几；.....”

4. For profit

Google™



Apple Computer

facebook.



Morgan Stanley

IBM

Nintendo®



RSA
SECURITY™

D E Shaw & Co

ORACLE™



Akamai

YAHOO!

amazon.com

Microsoft®



Why study Algorithms?

1. *Their impact is broad and far-reaching.*
2. *to become a proficient programmer.*
3. *For intellectual stimulation, be a wise person*
4. *For profit*
5. *Old roots, new opportunities.*
6. *To solve problems that could not otherwise be addressed.*
7. *They may unlock the secrets of life and of the universe.*

Why study anything else?



.....

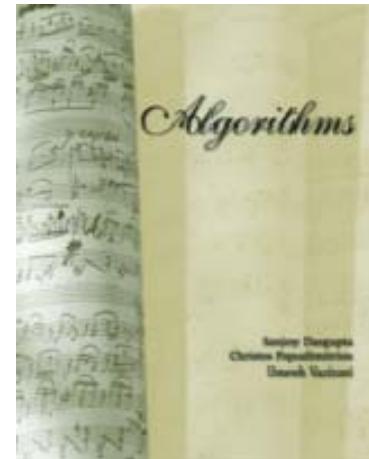
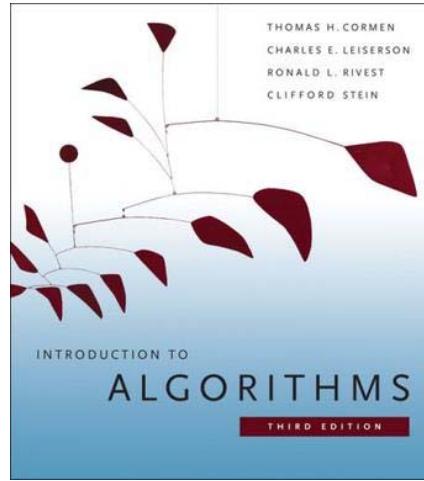
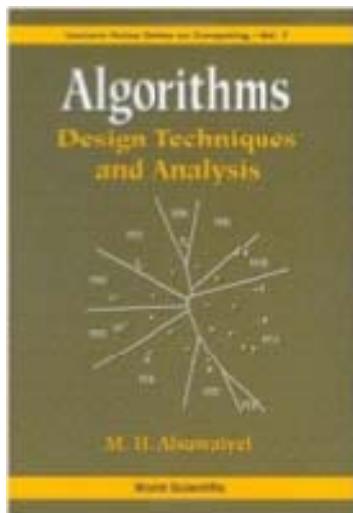
(Source from “Algorithms” by R. Sedgewick and K. Wayne)

Where are we?

- Why study Algorithms
- Outline of the course
- Course Policies

References

1. Algorithms: Design Techniques and Analysis, M.H. Alsuwaiyel
2. Introduction to Algorithms, T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein
3. Algorithms, S. Dasgupta, C. Papadimitriou, U. Vazirani
4. Algorithms, 4th edition, R. Sedgewick, K. Wayne



Course Overview

Weeks	Lectures	Topics	Readings	Dates
#1	Lecture 1 slides1 , slides2	Introduction	[0] ch 4.3, [1] ch 5.1.4, [2] ch 21, [3] ch 1.5	Sep. 15 (Tuesday)
	Lecture 2	Fundamentals	[0] ch 1.8-1.14, [1] ch 0, [2] ch 3, [3] ch 1.4	Sep. 17 (Thursday)
#2	Lecture 3	Induction&Sort(I)	[0] ch 5, [1] ch 1	Sep. 22 (Tuesday)
#3	Lecture 4	Sorting(II)	[0] ch 1.5, 1.7, 5.2, 5.3, 6.3, 6.6 [1] ch 2.3, 2.4, [2] ch 6, 7, 8, [3] ch 2	Sep. 29 (Tuesday)
		Holiday		Oct. 1 (Thursday)
#4	Lecture 5	Sort(III)	[0] ch 1.5, 1.7, 5.2, 5.3, 6.3, 6.6 [1] ch 2.3, 2.4, [2] ch 6, 7, 8, [3] ch 2	Oct. 6 (Tuesday)
#5	Lecture 6	Divide-and-conquer	[0] ch 2, 6, [1] ch 4, [2] ch 9, [3] ch 2.2	Oct. 13 (Tuesday)
	Lecture 7	Graph Traversal, part 1	[0] 9 [1] 3 [2] 22 [3] 4	Oct. 15 (Thursday)
#6	Lecture 7	Graph Traversal, part 2	[0] 9 [1] 3 [2] 22 [3] 4	Oct. 20 (Tuesday)
#7	Lecture 8	Shortest paths	[0] ch 4.2, 8.2, [1] ch 4, [2] ch 24, [3] ch 4	Oct. 27 (Tuesday)
				Oct. 29 (Thursday)
#8	Lecture 9	Greedy approach	[0] ch 8, [1] ch 5, [2] ch 16	Nov. 3 (Tuesday)
#9	Lecture 10	Dynamic programming	[0] ch 7, [1] ch 6, [2] ch 15	Nov. 10 (Tuesday)
				Nov. 12 (Thursday)
#10	Lecture 11	Network flow	[0] ch 16,17, [1] ch 6, [2] ch 26	Nov. 18 (Tuesday)
#11	Lecture 12	P and NP	[0] ch 10, [1] ch 8, [2] ch 34	Nov. 24 (Tuesday)
				Nov. 26 (Thursday)
#12	Lecture 13	Lower bounds	[0] ch 12, [2] ch 8.1	Dec. 1 (Tuesday)
#13	Lecture 14	String		Dec. 8 (Tuesday)
	Lecture 15	Data compression		Dec. 10 (Thursday)
#14	Lecture 16	Randomized algorithms	[0] ch 14, [1] ch 5	Dec. 15 (Tuesday)
#15	Lecture 17	Approximation algorithms	[0] ch 15, [1] ch 9.2, [2] ch 35	Dec. 22 (Tuesday)
	Lecture 18	Computational geometry	[0] ch 18, [2] ch 33	Dec. 24 (Thursday)
#16	Wrap up	Review		Dec. 29 (Tuesday)

Course goals

- To become proficient in the application of fundamental *algorithm design techniques*
- To gain familiarity with the main theoretical tools used in the *analysis of algorithms*
- To study and analyze different algorithms for many of “standard” *algorithmic problems*
- To introduce students to some of the *prominent subfields* of algorithmic study in which they may wish to pursue further study

Where are we?

- Why study Algorithms
- Outline of the course
- Course Policies

Grading Policy

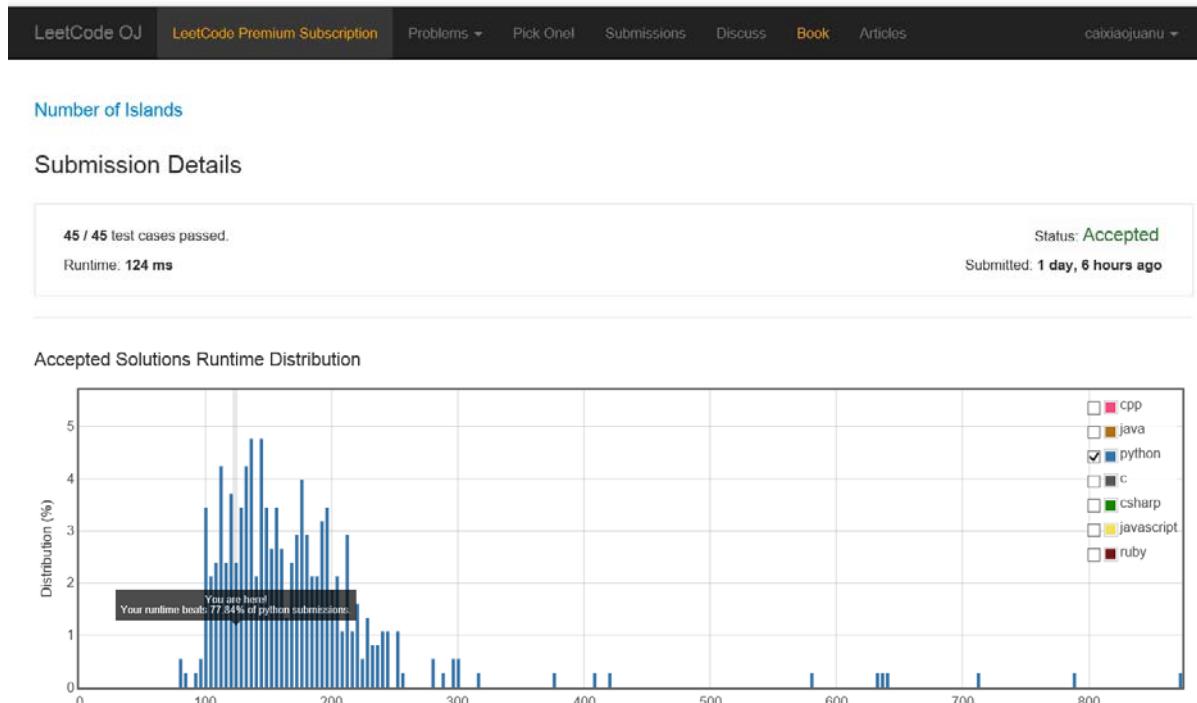
- Homework 25%
 - Distributed on course website after every lecture
 - Submit every Tuesday
- Programming assignment 15%
 - LeetCode (<http://leetcode.com>)
- Final exam 60%

TAs

- TAs
 - Bingbing Fang (方冰冰),
fke_htj@sjtu.edu.cn
 - Wenbo Zhang (张文博),
wbzhang@sjtu.edu.cn
- Homepage:
<http://basics.sjtu.edu.cn/~xiaojuan/algo15/>

More on LeetCode

- Submission:



- Due on every Saturday night: 23:59
- One student will be selected (randomly) to be report your solution (in 5 minutes) on Tuesday.

Teaching techniques

a. One-minute paper



Voices from senior students

- 算法是用来娱乐身心、课余刷题、公司面试、学会使用编程思想来看待各种问题的课程
- 用心学算法，很有意思的，这是程序员的内力修为，别太在意考试面试
- 坚持到期末
- 算法无论从哪个次元来看都是程序员的基础。
- 好好听课，不然会对自己失去信心。
- 这是我在软院上过最有意思最有收获的一门课。
- 算法很重要，但更重要的是思想。或许将来我们不会用到今天学习的算法，但是某个算法的思想会帮我们解决很多实际问题。
- 大神们请低调，给学渣们活路啊！
- 冬天还是从被窝里爬起来吧，对找工作还是很重要的
- ...

Enjoy!

Lecture I Introduction

- **Problem:** Dynamic Connectivity
- **Solutions:**
correctness, complexity, improvements
- **Applications**

ACKNOWLEDGEMENTS: Some slides in this lecture source from COS226
of Princeton University by [Kevin Wayne](#) and [Bob Sedgewick](#)

The problem

Problem: DynamicConnectivity

Input: n sites, m operations (*union* or *connected query*)

Output: the answer of connected queries

union(4, 3)

union(3, 8)

union(6, 5)

union(9, 4)

union(2, 1)

connected(0, 7) ✗

connected(8, 9) ✓

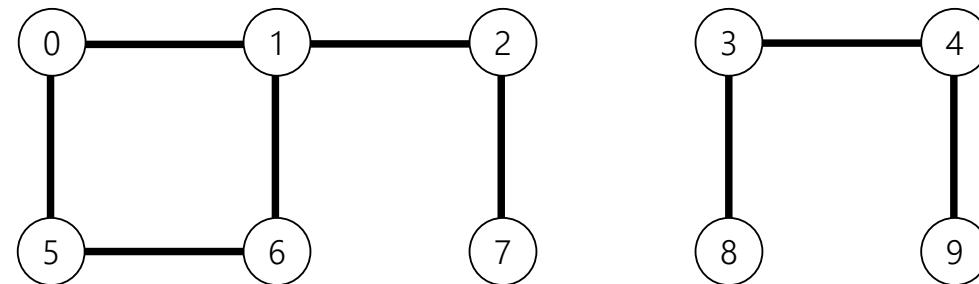
union(5, 0)

union(7, 2)

union(6, 1)

connected(0, 7) ✓

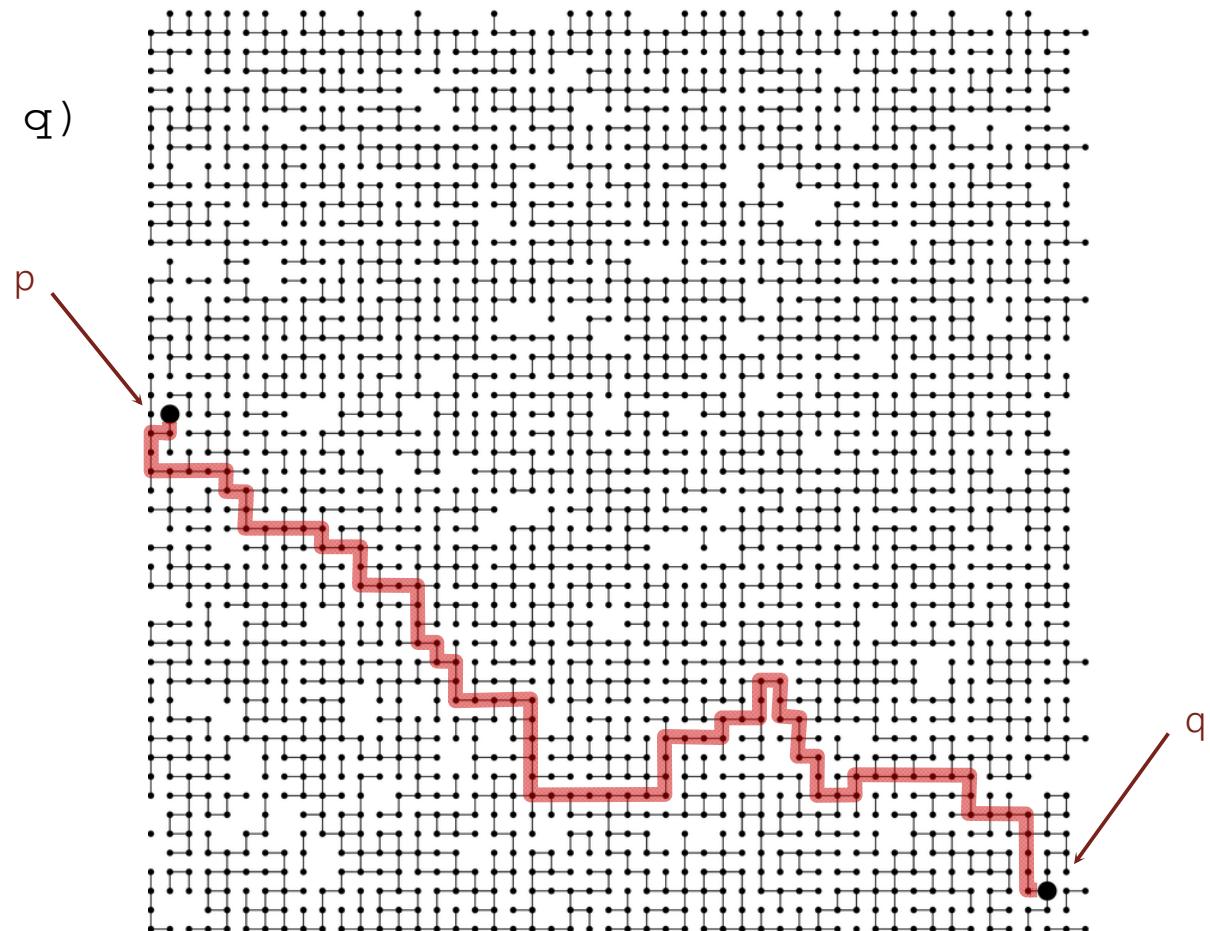
union(1, 0)



More difficult example

Q. $\text{Connected}(p, q)$

A. Yes.

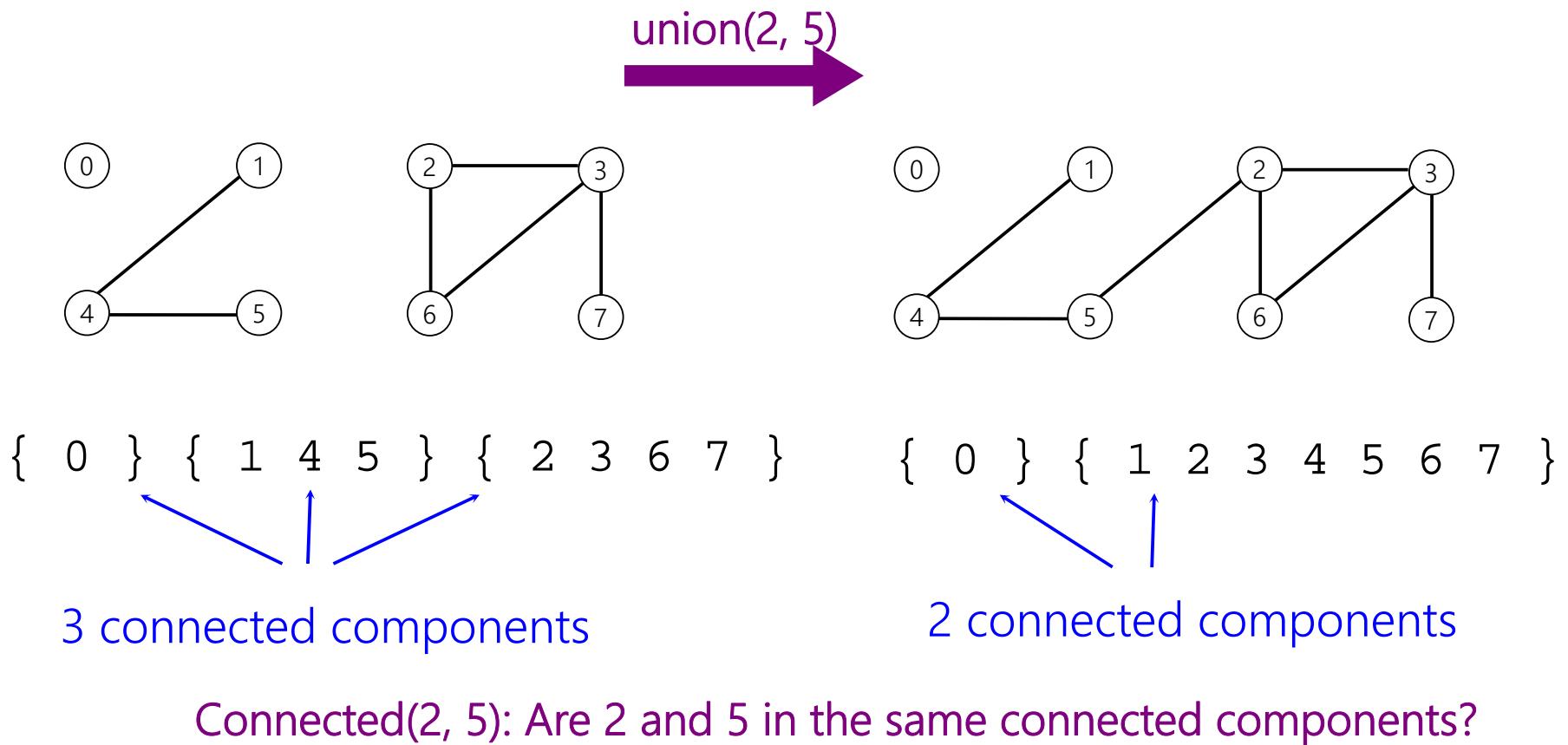


Where are we?

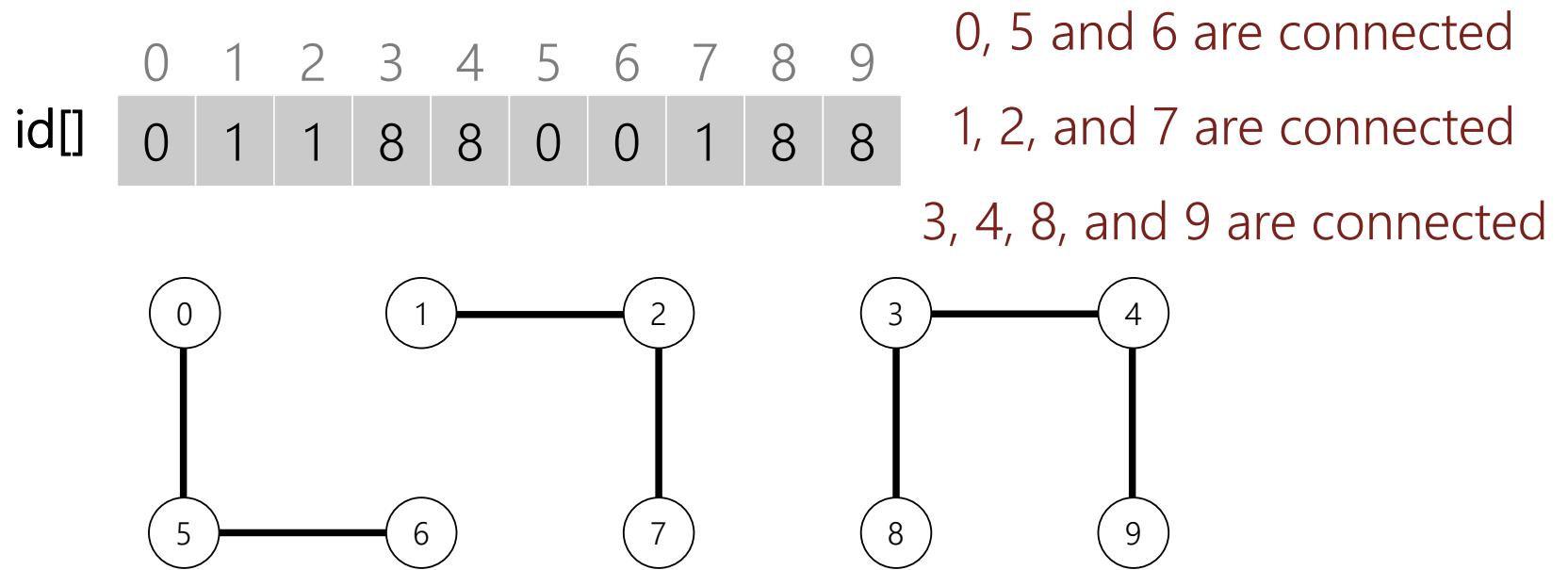
- Problem: Dynamic Connectivity
- **Solutions:**
correctness, complexity, improvements
- Applications

Your ideas

Model the problem



S1: Quick find



Find(p,q). Check if p and q have the same id.

Union(p,q). To merge components containing p and q, change all entries whose id equals $\text{id}[p]$ to $\text{id}[q]$.

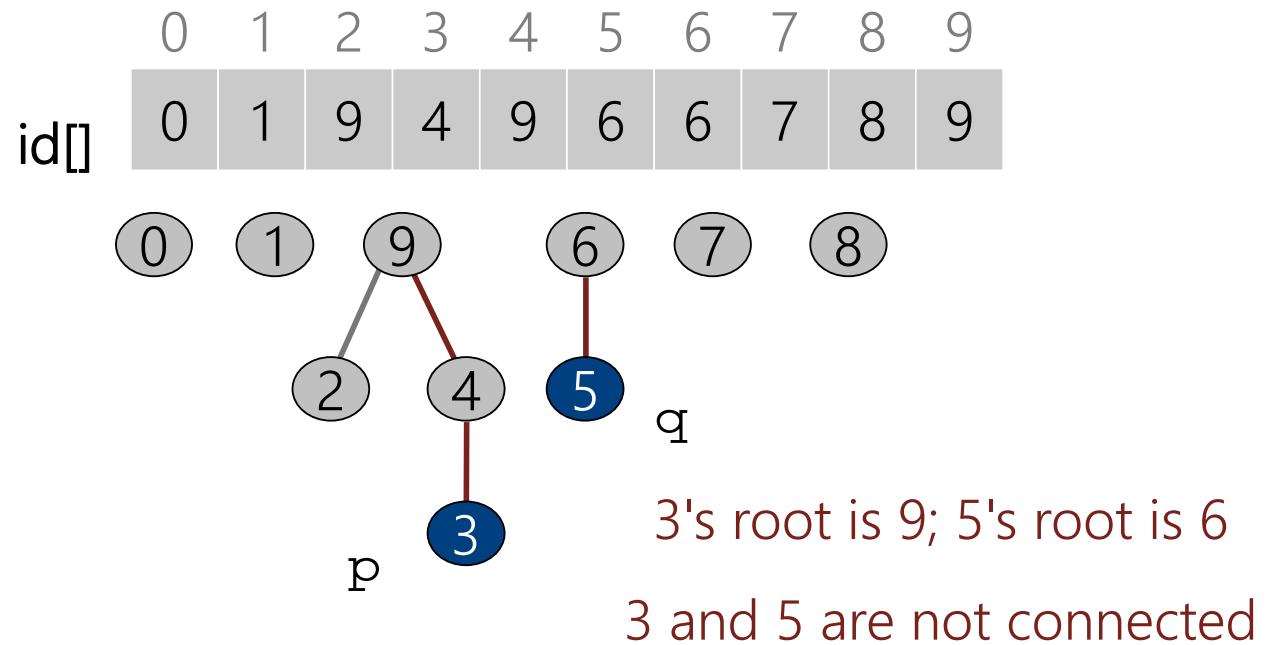
Quiz

Assume there are **N** sites, then in the worst case,

- *quick-find* need () array access for *union*.
- *quick-find* need () array access for *find*.

- A. N B. 1 C. N^2 D. $\log N$

S2: Quick union



Find(p, q). Check if p and q have the same root.

Union(p, q). To merge components containing p and q , set the id of p 's root to the id of q 's root.

Quiz

Assume there are **N** sites, then in the worst case,

- *quick-union* need () array access for *union*.
- *quick-union* need () array access for *find*.

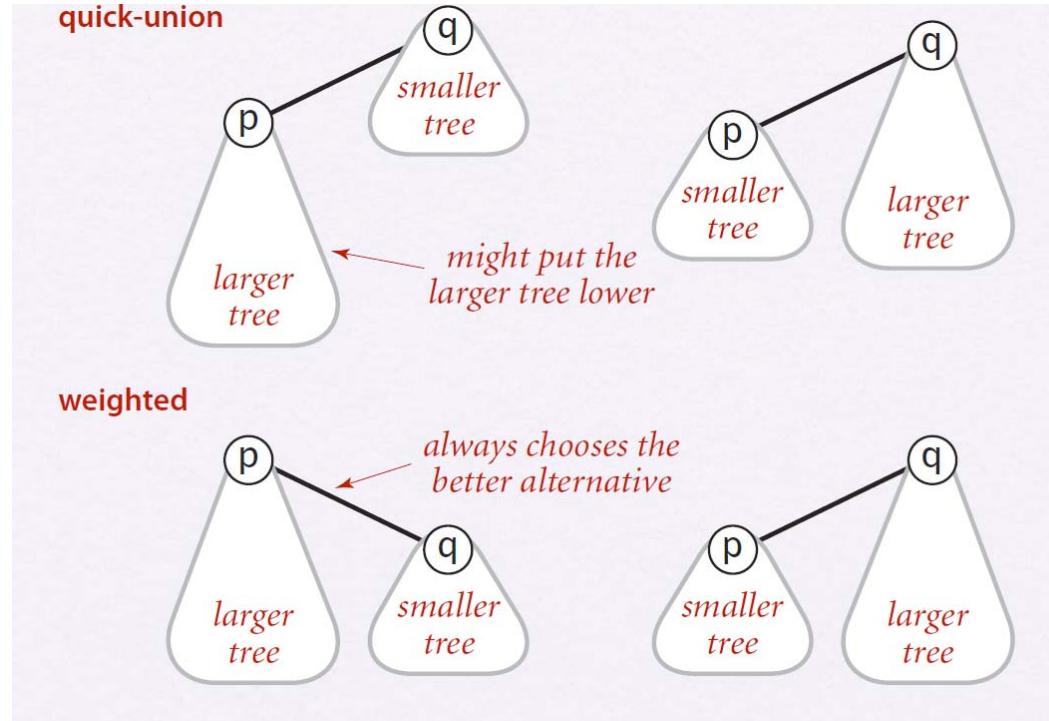
- A. N B. 1 C. N^2 D. $\log N$

Complexity

algorithm	initialize	union	find
quick-find	N	N	1
quick-union	N	N	N

Can we do better?

S3: Weighted union



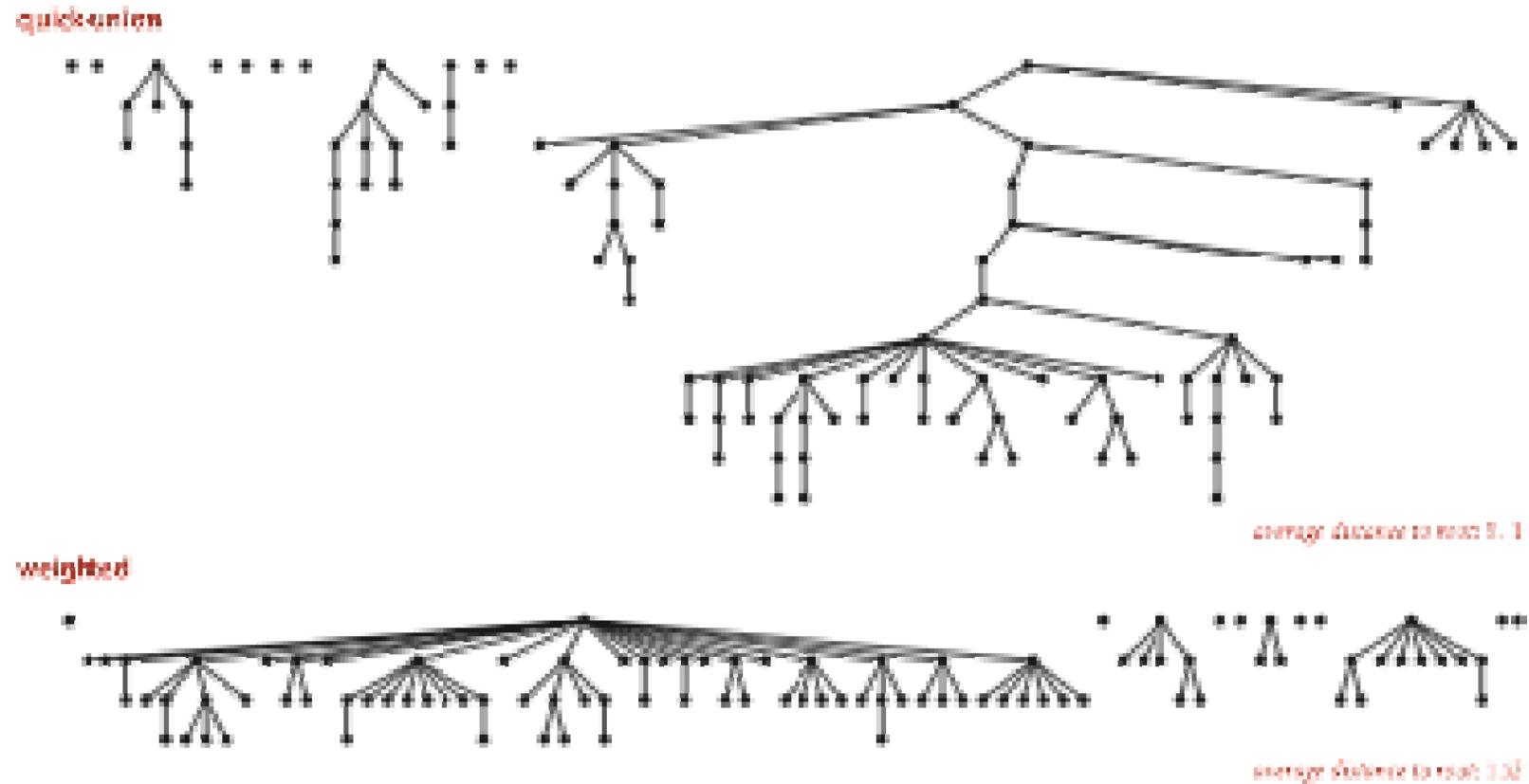
Find(p, q). Check if p and q have the same root.

Union(p, q). if $\text{size}[\text{root}(p)] \leq \text{size}[\text{root}(q)]$

then $\text{id}[\text{root}(p)] = \text{id}[\text{root}(q)]$

else $\text{id}[\text{root}(q)] = \text{id}[\text{root}(p)]$

S3: Weighted union



Quick-union and weighted quick-union (100 sites, 88 union() operations)

Quiz

Assume there are **N** sites, then in the worst case,

- *weighted-QU* need () array access for *union*.
- *weighted-QU* need () array access for *find*.

- A. N B. 1 C. N^2 D. $\log N$

Complexity

Proposition. Depth of any node x is at most $\log N$.

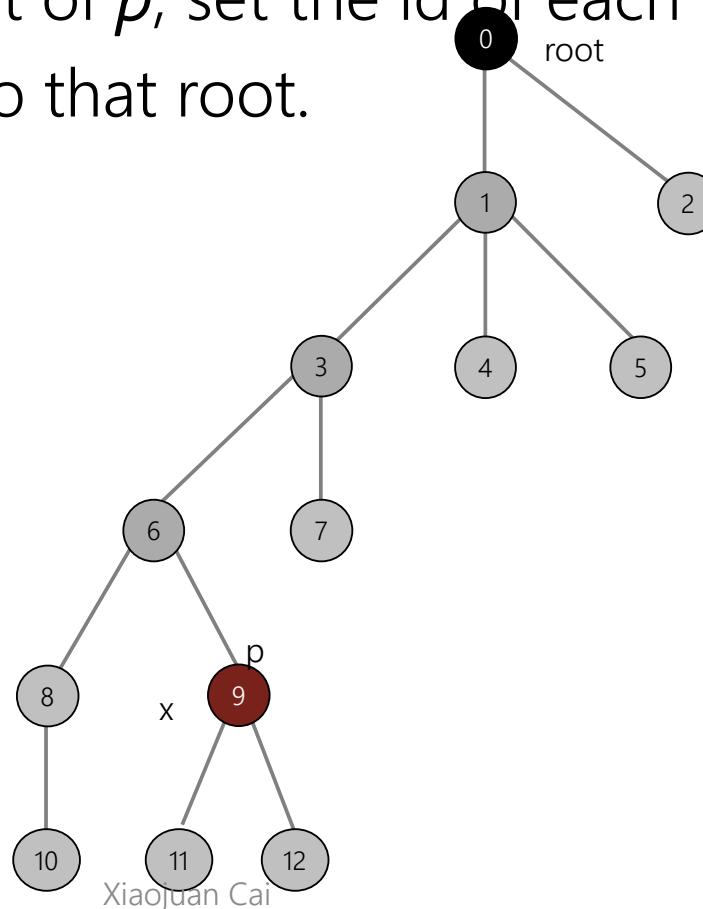
Complexity

Proposition. Depth of any node x is at most $\log N$.

algorithm	initialize	union	connect ed
quick-find	N	N	1
quick-union	N	N	N
weighted QU	N	$\log N$	$\log N$

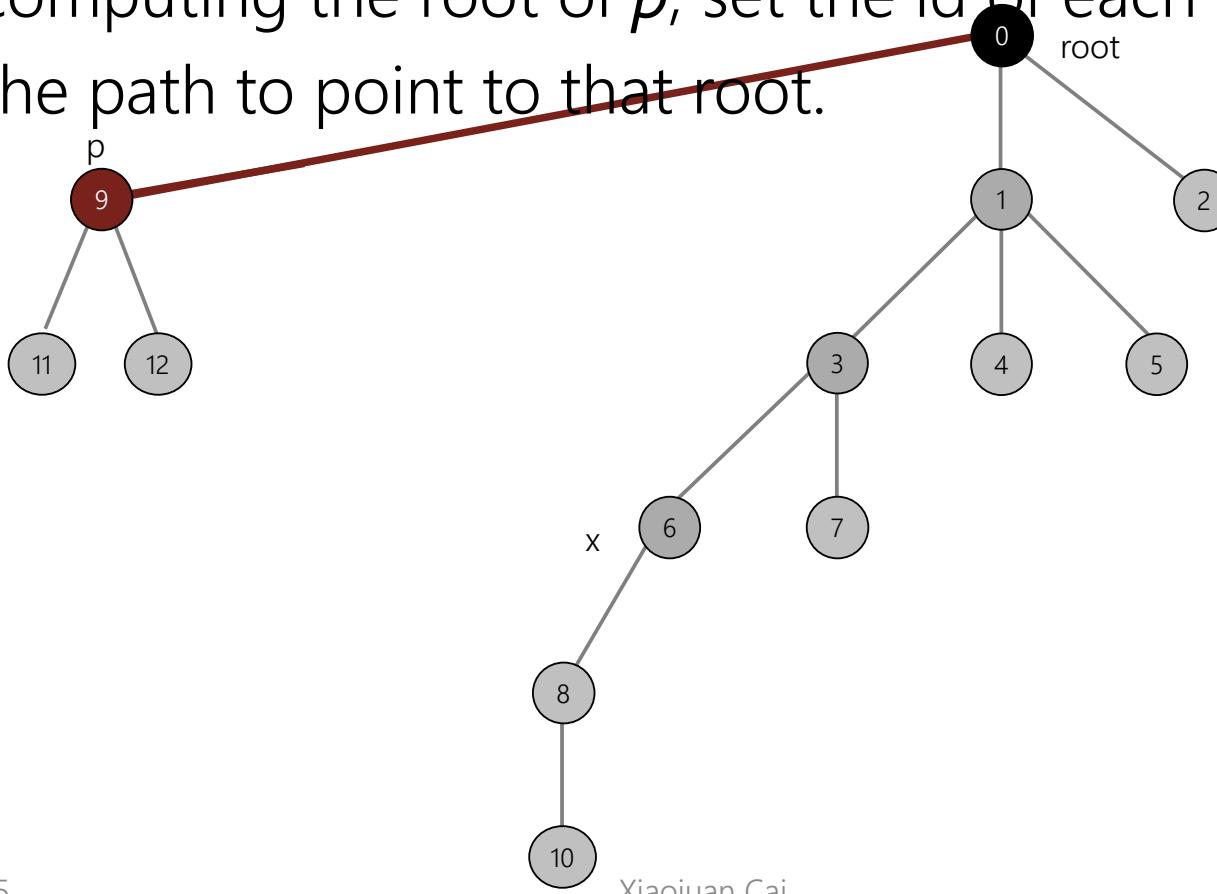
S4: Path compression

Quick union with path compression. Just after computing the root of p , set the id of each node on the path to point to that root.



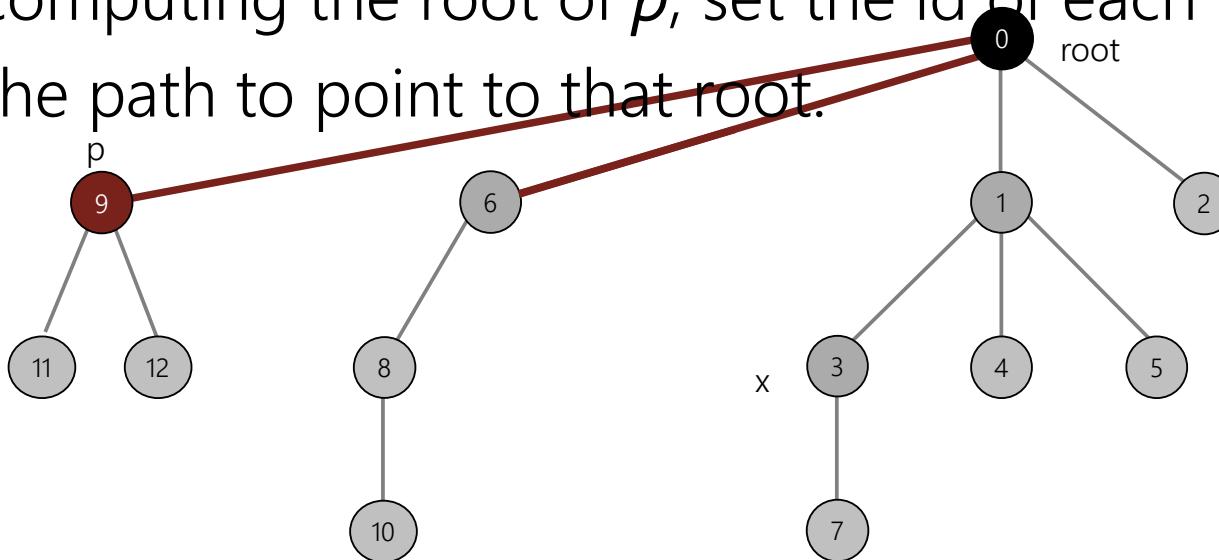
S4: Path compression

Quick union with path compression. Just after computing the root of p , set the id of each node on the path to point to that root.



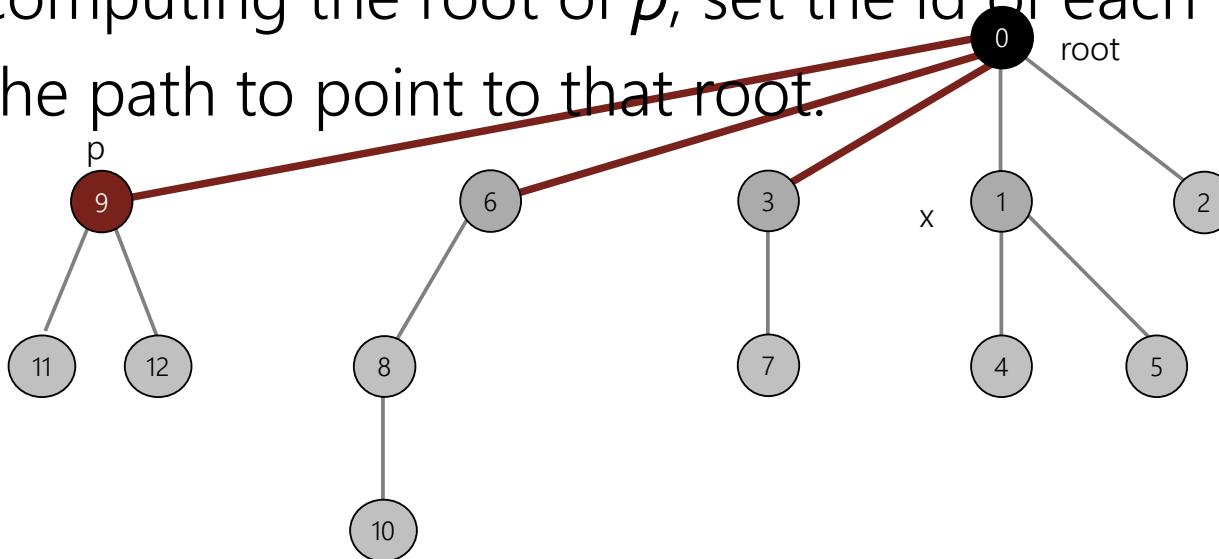
S4: Path compression

Quick union with path compression. Just after computing the root of p , set the id of each node on the path to point to that root.



S4: Path compression

Quick union with path compression. Just after computing the root of p , set the id of each node on the path to point to that root.



Complexity

Proposition. Starting from an empty data structure, any sequence of M union-find operations on N objects makes at most proportional to $N + M \log^* N$ array accesses.



Bob Tarjan
(Turing Award '86)

N	$\log^* N$
1	0
2	1
4	2
16	3
65536	4
2^{65536}	5

Proofs are refer to [textbook] 4.3 and [DPV]5.1.4.

Xiaojuan Cai

Complexity

Proposition. Starting from an empty data structure, any sequence of M union-find operations on N objects makes at most proportional to $N + M \log^* N$ array accesses.



Bob Tarjan
(Turing Award '86)

N	$\log^* N$
1	0
2	1
4	2
16	3
65536	4
2^{65536}	5

Can we do better?

Amazing fact. No linear-time algorithm exists.

Xiaojuan Cai

Where are we?

- Problem: Dynamic Connectivity
- Solutions:
correctness, complexity, improvements
- **Applications**

Application

- Percolation. [Coursera. AlgoPartl. PA1]
- Dynamic connectivity.
- Hinley-Milner polymorphic type inference.
- Kruskal's minimum spanning tree algorithm.
- Compiling equivalence statements in Fortran.
-

Percolation

Conclusion

- Model a problem
- Solve it, then ask three questions:
 - Is it correct?
 - How much time does it take?
 - Can we do better?
- Applications

Lecture 2 Analysis of Algorithms

- How to estimate time complexity?
- Analysis of algorithms
- Techniques based on Recursions

ACKNOWLEDGEMENTS: Some contents in this lecture source from slides of [Yuxi Fu](#)

ROADMAP

- How to estimate time complexity?
- Analysis of algorithms
 - worst case, best case?
 - Big-O notation

Observation

Problem: 3-Sum

Input: N distinct integers

Output: the number of triples sum to exactly zero?

30 -40 -20 -10 40 0 10 5

a[i]	a[j]	a[k]	sum
30	-40	10	0
30	-20	-10	0
-40	40	0	0
-10	0	10	0



Observation

System dependent effects.

- Hardware: CPU, memory, cache, ...
- Software: compiler, interpreter, garbage collector, ...
- System: operating system, network, other applications, ...

Easy, but difficult to get precise measurements.

Mathematical model

Total running time = sum of cost × frequency for all operations.

- Cost depends on machine, compiler.
- Frequency depends on algorithm, input data.

```
% 1-sum  
int count = 0;  
for (int i = 0; i < N; i++)  
    if (a[i] == 0)  
        count++;
```

Mathematical model

- Relative rather than Absolute.
- Machine independent.
- About large input instance
 - **Time complexity** of an algorithm is a function of the size of the input n .

Input size

Algorithm 1.9 FIRST

Input: A positive integer n and an array $A[1..n]$ with $A[j] = j$ for $1 \leq j \leq n$.

Output: $\sum_{j=1}^n A[j]$.

1. $sum \leftarrow 0;$
2. **for** $j \leftarrow 1$ **to** n
3. $sum \leftarrow sum + A[j]$
4. **end for**
5. **return** sum

Algorithm 1.10 SECOND

Input: A positive integer n .

Output: $\sum_{j=1}^n j$.

1. $sum \leftarrow 0;$
2. **for** $j \leftarrow 1$ **to** n
3. $sum \leftarrow sum + j$
4. **end for**
5. **return** sum

	Algo 1.9	Algo 1.10
input size	n	$\lceil \log n \rceil + 1$
time	n	n
relation	Linear	Exponential

Input size -- convention

- Sorting and searching: the size of the array
- Graph: the number of the vertices and edges
- Computational geometry: the number points or line segments
- Matrix operation: the dimensions of the input matrices
- Number theory and Cryptography: the number of bits of the input.

Where are we?

- How to estimate time complexity?
- Analysis of algorithms
 - worst case, best case?
 - Big-O notation

Searching

Problem: Searching

Input: An integer array $A[1\dots n]$, and an integer x

Output: Does x exist in A ?

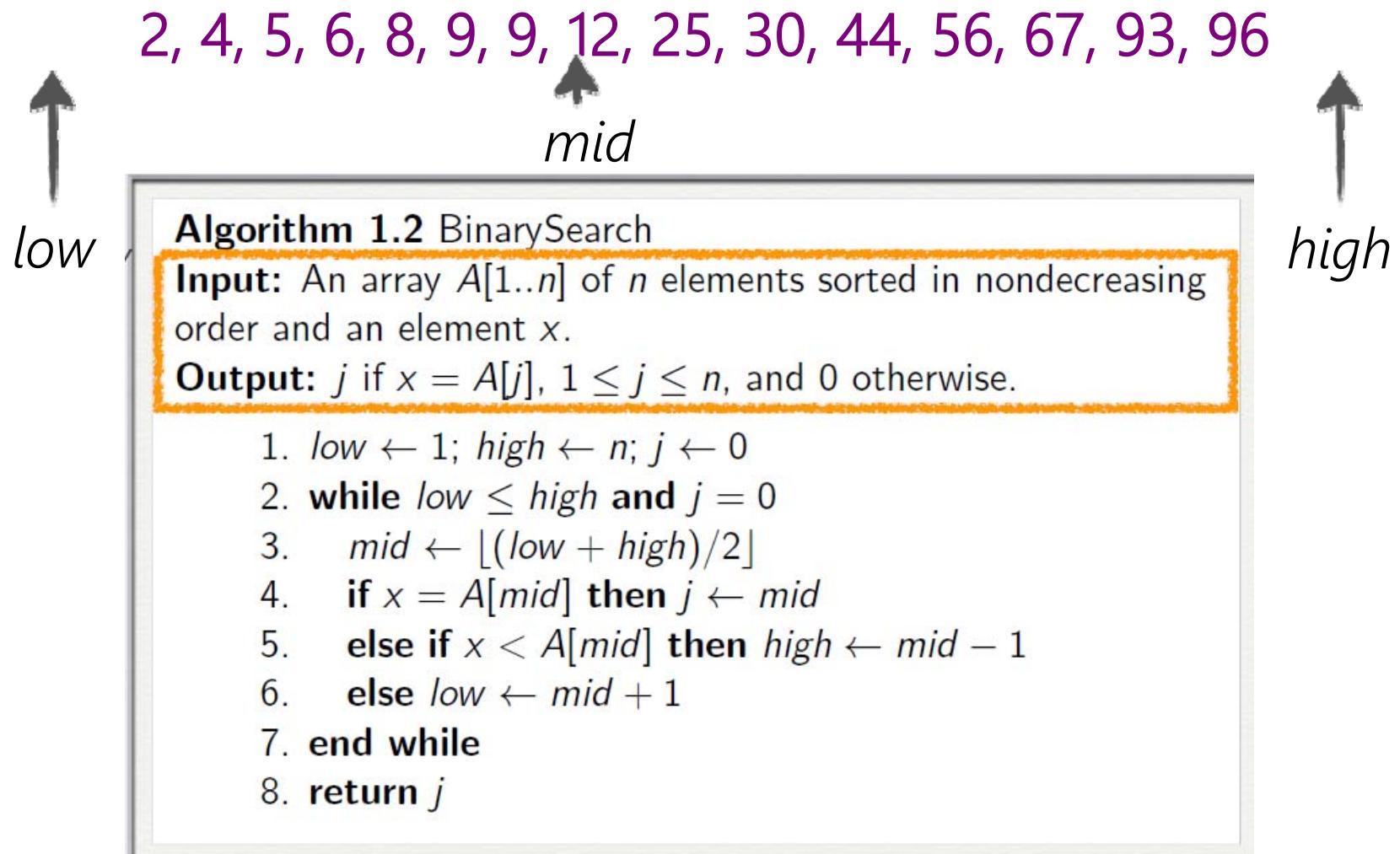
Does 45 exist in:

9, 8, 9, 6, 2, 56, 12, 5, 4, 30, 67, 93, 25, 44, 96

2, 4, 5, 6, 8, 9, 9, 12, 25, 30, 44, 56, 67, 93, 96

Linear searching v.s. Binary searching

Binary searching



Quiz

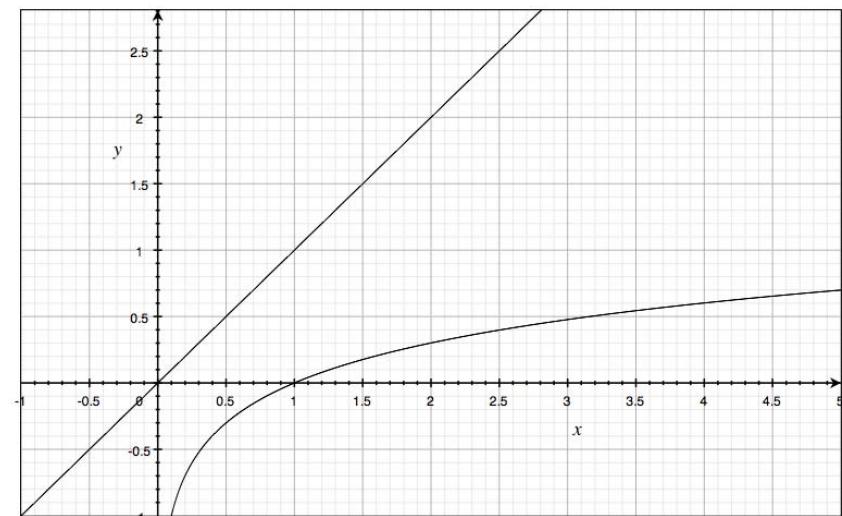
	Linear Searching	Binary Searching
Best case		
Worst case		

- A. 1 B. logn. C. n D. *none of above*

Computational Complexity

	Linear Searching	Binary Searching
Worst case	n	$\log n$

10^{-6} s per instruction	Linear Searching	Binary Searching
$n=1024$	0.001s	0.00001s
$n=2^{40}$	12.7day	0.00004s



Where are we?

- How to estimate time complexity?
- Analysis of algorithms
 - worst case, best case?
 - Big-O notation

Big-O Notation

Definition (O -notation)

Let $f(n)$ and $g(n) : \mathbb{N} \rightarrow \mathbb{R}^+ \cup \{0\}$

$f(n)$ is said to be $O(g(n))$, written $f(n) = O(g(n))$,
if $\exists c > 0. \forall n_0. \forall n \geq n_0. f(n) \leq cg(n)$

- The O -notation provides an upper bound of the running time;
- f grows no faster than some constant times g .

Examples

- $10n^2 + 20n = O(n^2)$.
- $\log n^2 = O(\log n)$.
- $\log(n!) = O(n \log n)$.
- Hanoi: $T(n) = O(2^n)$, n is the input size.
- Searching: $T(n) = O(n)$

Big- Ω Notation

Definition (Ω -notation)

Let $f(n)$ and $g(n) : \mathbb{N} \rightarrow \mathbb{R}^+ \cup \{0\}$

$f(n)$ is said to be $\Omega(g(n))$, written $f(n) = \Omega(g(n))$, if $\exists c > 0. \forall n \geq n_0. f(n) \geq cg(n)$

- The Ω -notation provides an **lower bound** of the running time;
- $f(n) = O(g(n))$ iff $g(n) = \Omega(f(n))$.

Examples

- $\log n^k = \Omega(\log n)$.
- $\log n! = \Omega(n \log n)$.
- $n! = \Omega(2^n)$.
- Searching: $T(n) = \Omega(1)$



Does there exist f, g , such that $f = O(g)$ and $f = \Omega(g)$?

Big- Θ notation

Definition (Θ -notation)

$f(n) = \Theta(g(n))$, if both $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$.

- $\log n! = \Theta(n \log n)$.
- $10n^2 + 20n = \Theta(n^2)$

Small o -notation

Definition (o -notation)

Let $f(n)$ and $g(n) : \mathbb{N} \rightarrow \mathbb{R}^+ \cup \{0\}$

$f(n)$ is said to be $o(g(n))$, written $f(n) = o(g(n))$, if $\forall c. \exists n_0. \forall n \geq n_0. f(n) < cg(n)$

- $\log n! = o(n \log n)?$
- $10n^2 + 20n = o(n^2)?$
- $n \log n = o(n^2)?$

Small ω -notation

Definition (ω -notation)

Let $f(n)$ and $g(n) : \mathbb{N} \rightarrow \mathbb{R}^+ \cup \{0\}$

$f(n)$ is said to be $\omega(g(n))$, written $f(n) = \omega(g(n))$,
if $\forall c. \exists n_0. \forall n \geq n_0. f(n) > cg(n)$

Quiz

$f(n)$	$g(n)$
$n^{1.01}$	$n \log^2 n$
$n \log n$	$\log(n!)$
$1 + 1/2 + 1/3 + \dots + 1/n$	$\log n$
$n 2^n$	3^n

- A. O B. Ω . C. Θ D. *none of above*

Logarithms

$$\log_b xy = \log_b x + \log_b y$$

$$\log_b c^y = y \log_b c$$

$$\log_b x = \frac{\log_a x}{\log_a b}$$

$$x^{\log_a y} = y^{\log_a x}$$

Pigeonhole principle



(source from wikipedia)



Johann Dirichlet
(source from wikipedia)



Prove there exists a number consisting of 7 and 0 that can divide 13.

Summation

$$\sum_{j=1}^n j = \frac{n(n+1)}{2} = \Theta(n^2)$$

$$\sum_{j=0}^n c^j = \frac{c^{n+1} - 1}{c - 1} = \Theta(c^n)$$

Approximation by Integration

$$\int_{m-1}^n f(x)dx \leq \sum_{j=m}^n f(j) \leq \int_m^{n+1} f(x)dx$$

$$\int_m^{n+1} f(x)dx \leq \sum_{j=m}^n f(j) \leq \int_{m-1}^n f(x)dx$$

$$\sum_{j=1}^n j^k = \Theta(n^{k+1})$$

$$H_n = \sum_{j=1}^n \frac{1}{j} = \Theta(\ln n) = \Theta(\log n)$$

$$\sum_{j=1}^n \log j = \Theta(n \log n)$$

In terms of limits

Suppose $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)}$ exists

$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} \neq \infty$ implies $f(n) = O(g(n))$

$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} \neq 0$ implies $f(n) = \Omega(g(n))$

$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = c$ implies $f(n) = \Theta(g(n))$

$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 0$ implies $f(n) = o(g(n))$

$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = \infty$ implies $f(n) = \omega(g(n))$

Complexity class

Definition (*Complexity class*)

An equivalence relation R on the set of complexity functions is defined as follows:

$$f R g \text{ if and only if } f(n) = \Theta(g(n)).$$

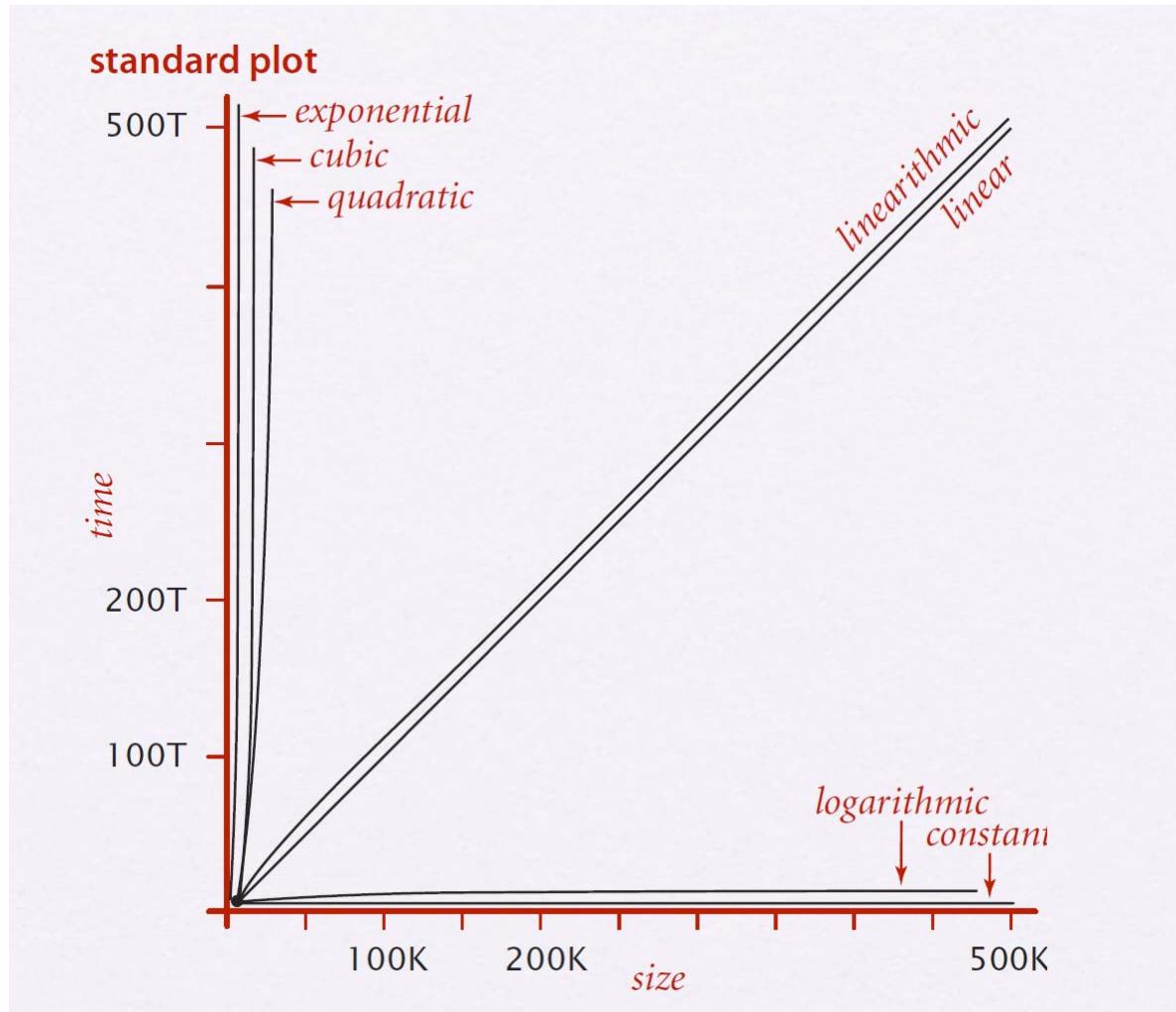
$$1 < \log \log n < \log n < \sqrt{n} < n^{3/4} < n < n \log n < n^2 < 2^n < n! < 2^{n^2}$$

Order of growth

order	name	example	N = 1000	N = 2000
1	constant	add two numbers	instant	instant
$\log N$	logarithmic	binary search	instant	instant
N	linear	find the maximum	instant	instant
$N \log N$	linearithmic	mergesort	instant	instant
N^2	quadratic	2-sum	~1s	~2s
2^N	exponential	Hanoi	forever	forever

Assume the computer consumes 10^{-6} s per instruction

Order of growth



Quiz

Algorithm 1.7 Count1

Input: $n = 2^k$, for some positive integer k .

Output: $count =$ number of times Step 4 is executed.

1. $count \leftarrow 0;$
2. **while** $n \geq 1$
3. **for** $j \leftarrow 1$ **to** n
4. $count \leftarrow count + 1$
5. **end for**
6. $n \leftarrow n/2$
7. **end while**
8. **return** $count$

- A. $\Theta(n^2)$ B. $\Theta(n)$. C. $\Theta(n \log n)$ D. *none of above*

Quiz

Algorithm 1.7 Count1

Input: $n = 2^k$, for some positive integer k .

Output: $count =$ number of times Step 4 is executed.

1. $count \leftarrow 0;$
2. **while** $n \geq 1$
3. **for** $j \leftarrow 1$ **to** n
4. $count \leftarrow count + 1$
5. **end for**
6. $n \leftarrow n/2$
7. **end while**
8. **return** $count$

$$n + \frac{n}{2} + \frac{n}{2^2} + \cdots + \frac{n}{2^j} + \cdots + \frac{n}{2^{\log n}} = 2n - 1 = \theta(n)$$

Quiz

Algorithm 1.8 Count2

Input: A positive integer n .

Output: $count = \text{number of times Step 5 is executed.}$

1. $count \leftarrow 0;$
2. **for** $i \leftarrow 1$ **to** n
3. $m \leftarrow \lfloor n/i \rfloor$
4. **for** $j \leftarrow 1$ **to** m
5. $count \leftarrow count + 1$
6. **end for**
7. **end for**
8. **return** $count$

- A. $\Theta(n^2)$ B. $\Theta(n)$. C. $\Theta(n\log n)$ D. *none of above*

Quiz

Algorithm 1.8 Count2

Input: A positive integer n .

Output: $count = \text{number of times Step 5 is executed.}$

1. $count \leftarrow 0;$
2. **for** $i \leftarrow 1$ **to** n
3. $m \leftarrow \lfloor n/i \rfloor$
4. **for** $j \leftarrow 1$ **to** m
5. $count \leftarrow count + 1$
6. **end for**
7. **end for**
8. **return** $count$

- A. $\Theta(n^2)$ B. $\Theta(n)$.

$$\sum_{i=1}^n \left(\frac{n}{i} - 1 \right) \leq \sum_{i=1}^n \left\lfloor \frac{n}{i} \right\rfloor \leq \sum_{i=1}^n \frac{n}{i}$$

Quiz

Algorithm 1.9 Count3

Input: $n = 2^{2^k}$, k is a positive integer.

Output: $count =$ number of times Step 6 is executed.

1. $count \leftarrow 0;$
2. **for** $i \leftarrow 1$ **to** n
3. $j \leftarrow 2;$
4. **while** $j \leq n$
5. $j \leftarrow j^2;$
6. $count \leftarrow count + 1$
7. **end while**
8. **end for**
9. **return** $count$

- A. $\Theta(n^2)$ B. $\Theta(n)$. C. $\Theta(n \log n)$ D. *none of above*

Quiz

Algorithm 1.9 Count3

Input: $n = 2^{2^k}$, k is a positive integer.

Output: $count =$ number of times Step 6 is executed.

1. $count \leftarrow 0;$
2. **for** $i \leftarrow 1$ **to** n
3. $j \leftarrow 2;$
4. **while** $j \leq n$
5. $j \leftarrow j^2;$
6. $count \leftarrow count + 1$
7. **end while**
8. **end for**
9. **return** $count$



$\theta(n \log \log n)$

- A. $\Theta(n^2)$ B. $\Theta(n)$. C. $\Theta(n \log n)$ D. *none of above*

Amortized analysis

1. **for** $j \leftarrow 1$ **to** n
2. $x \leftarrow A[j]$
3. Append x to the list
4. **if** x is even **then**
5. **while** $\text{pred}(x)$ is odd **do** delete $\text{pred}(x)$
6. **end if**
7. **end for**

The total number of **append** is n .

The total number of **delete** is between 0 and $n-1$.
So the time complexity is $O(n)$.

Memory

- Time: Time complexity
 - faster, better.
- Memory: Space complexity
 - less, better.
- Space complexity \leq Time complexity

Conclusion

- How to estimate time complexity?
- Analysis of algorithms
 - worst case, best case?
 - Big-O notation

3-Sum, better solution

Problem: 3-Sum

Input: N distinct integers

Output: the number of triples sum to exactly zero?

30 -40 -20 -10 40 0 10 5

a[i]	a[j]	a[k]	sum
30	-40	10	0
30	-20	-10	0
-40	40	0	0
-10	0	10	0

- $O(N^2)$

Lecture 3 Induction & Sort(1)

- Algorithm design techniques: induction
- Selection sort, Insertion sort, Shell sort ...

ROADMAP

- Algorithm design techniques: induction
 - Permutation
 - Majority element
 - Radix sort
- Selection sort, Insertion sort, Shell sort ...
 - Proofs and comparisons

Techniques based on Recursions

- Induction or Tail recursion
 - Iteration,
 - Proof: invariants by induction
- Non-overlapping subproblems
 - Divide-and-conquer
- Overlapping subproblems
 - Dynamic programming

Techniques based on Recursions

- Induction or Tail recursion
 - Iteration,
 - Proof: invariants by induction
- Non-overlapping subproblems
 - Divide-and-conquer
- Overlapping subproblems
 - Dynamic programming

Induction example

Selection sort

Problem size: 6 9, 8, 9, 6, 2, 56

Problem size: 5 2, 8, 9, 6, 9, 56

Algorithm Design: problem size $n \rightarrow n-1$

Correctness Proof: by *mathematical induction.*

Permutation

Problem: Permuuation

Input: an array of n elements

Output: the permutations of n
elements

1 2 3
1 3 2
2 1 3
2 3 1
3 1 2
3 2 1

Permutation1

Permutation2

Permutation

Permutation 1:

1 2 3

1 3 2

2 1 3

2 3 1

3 1 2

3 2 1

$a_1 \ a_2 \ a_3 \ a_4$

Permutation 2:

$a_1 \ a_2 \ a_3 \ a_4$



Permutation 1

Algorithm 5.7 Permutations1

Input: A positive integer n .

Output: All permutations of the numbers $1, 2, \dots, n$.

1. **for** $j \leftarrow 1$ **to** n
2. $P[j] \leftarrow j$
3. **end for**
4. $perm1(1)$

 $\Omega(n \square n!)$ **Procedure** $perm1(m)$

1. **if** $m = n$ **then output** $P[1..n]$
2. **else**
3. **for** $j \leftarrow m$ **to** n
4. interchange $P[j]$ **and** $P[m]$
5. $perm1(m + 1)$
6. interchange $P[j]$ **and** $P[m]$
7. **comment:** At this point $P[m..n] = m, m + 1, \dots, n$.
8. **end for**
9. **end if**

Permutation 2

Algorithm 5.8 Permutations2

Input: A positive integer n .

Output: All permutations of the numbers $1, 2, \dots, n$.

1. **for** $j \leftarrow 1$ **to** n
2. $P[j] \leftarrow 0$
3. **end for**
4. $\text{perm2}(n)$

Procedure $\text{perm2}(m)$

1. **if** $m = 0$ **then output** $P[1..n]$
2. **else**
3. **for** $j \leftarrow 1$ **to** n
4. **if** $P[j] = 0$ **then**
5. $P[j] \leftarrow m$
6. $\text{perm2}(m - 1)$
7. $P[j] \leftarrow 0$
8. **end if**
9. **end for**
10. **end if**

$\Omega(n \square n!)$

Quiz

Which output is in alphabetical order?

- A. permutation1
- B. permutation2
- C. both
- D. *none of above*

Polynomial

$$p(x) = \sum_{i=0}^n a_i x^i = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + \cdots + a_n x^n,$$

Problem: PolynomialEval

Input: $a[0...n]$, and x

Output: the value of $p(x)$



William George Horner
(source from *Wikipedia*)

Horner's rule

$$p(x) = a_0 + x(a_1 + x(a_2 + \cdots + x(a_{n-1} + a_n x) \cdots)).$$

Horner's rule

Algorithm 5.6 Horner

Input: A sequence of $n + 2$ real number a_0, a_1, \dots, a_n and x .

Output: $P_n(x) = a_nx^n + a_{n-1}x^{n-1} + \dots + a_1x + a_0$.

1. $p \leftarrow a_n$
2. **for** $j \leftarrow 1$ **to** n
3. $p \leftarrow xp + a_{n-j}$
4. **end for**
5. **return** p

Finding majority element

Problem: MajorityElement

Input: a sequence of integers $A[1\dots n]$

Output: An integer a in A that appears more than $n/2$ times, otherwise, output *none*.

1 2 3 3 4 2 2 2 3 2 *none*

1 5 1 1 4 2 1 3 1 1 1

Observation

If two different elements in the original sequence are removed, then the majority in the original sequence remains the majority in the new sequence.

Finding majority element

Algorithm 5.9 Majority

Input: An array $A[1..n]$ of n elements.

Output: The majority element if it exists; otherwise `none`.

```
1.  $c \leftarrow \text{candidate}(1)$ 
2.  $\text{count} \leftarrow 0$ 
3. for  $j \leftarrow 1$  to  $n$ 
4.   if  $A[j] = c$  then  $\text{count} \leftarrow \text{count} + 1$ 
5. end for
6. if  $\text{count} > \lfloor n/2 \rfloor$  then return  $c$ 
7. else return none
```



$\Theta(n)$

Procedure $\text{candidate}(m)$

```
1.  $j \leftarrow m$ ;  $c \leftarrow A[m]$ ;  $\text{count} \leftarrow 1$ 
2. while  $j < n$  and  $\text{count} > 0$ 
3.    $j \leftarrow j + 1$ 
4.   if  $A[j] = c$  then  $\text{count} \leftarrow \text{count} + 1$ 
5.   else  $\text{count} \leftarrow \text{count} - 1$ 
6. end while
7. if  $j = n$  then return  $c$ 
8. else return  $\text{candidate}(j + 1)$ 
```

Proof of correctness

Lemma (Invariants)

Given $a[1..N]$, Candidates(i) returns the majority element of $a[i...N]$ if there exists a majority element.

Proof. (Inductive proof)

Base step. Lemma holds for $i = N$

Inductive step.

Induction Hypothesis. Assume for $j > i$, the lemma holds.

Prove it holds for i .

Radix sort

All numbers in the array consists
of EXACTLY k digits.

7467	6792	9134	9134	1239
1247	9134	1239	9187	1247
3275	3275	1247	1239	3275
6792	4675	7467	1247	4675
9187	7467	3275	3275	6792
9134	1247	4675	7467	7467
4675	9187	9187	4675	9134
1239	1239	6792	6792	9187

Proof of correctness

Lemma (Invariants)

In Radix sort, if the i -th digits are sorted, then the numbers consisting of $i, i-1, \dots, 1$ digits are sorted..

Proof. (Inductive proof)

Base step. Lemma holds for $i = 1$

Inductive step.

Induction Hypothesis. Assume for $j < i$, the lemma holds.

Prove it holds for i .

Where are we?

- Algorithm design techniques: induction
 - Permutation
 - Majority element
 - Radix sort
- Selection sort, Insertion sort, Shell sort ...
 - Proofs and comparisons

?



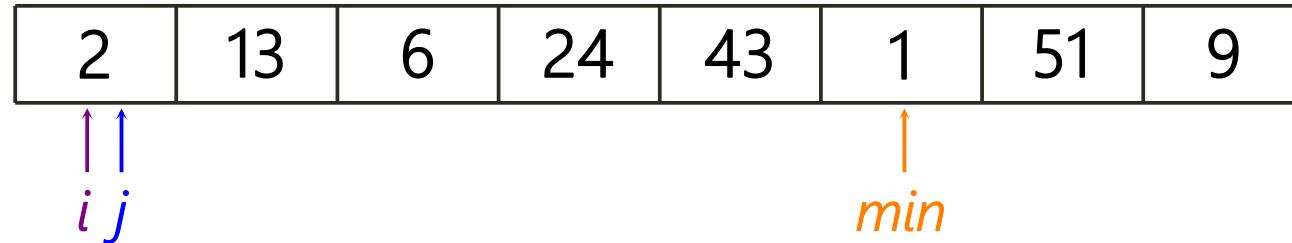
?



?



Selection sort

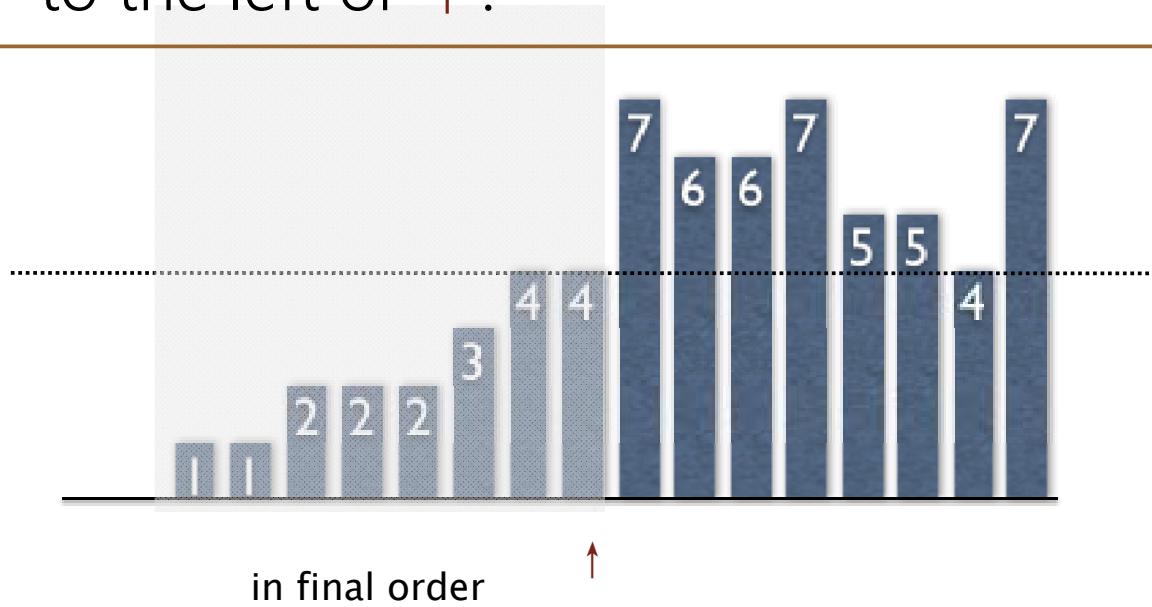


	Best case	Worst case
#Comparison:	$\frac{n(n-1)}{2}$	$\frac{n(n-1)}{2}$
#Interchange:	0	$n - 1$

Selection sort

Lemma (Invariants)

- Entries to the left of \uparrow (including \uparrow) fixed and in ascending order.
- No entry to right of \uparrow is smaller than any entry to the left of \uparrow .



Insertion sort

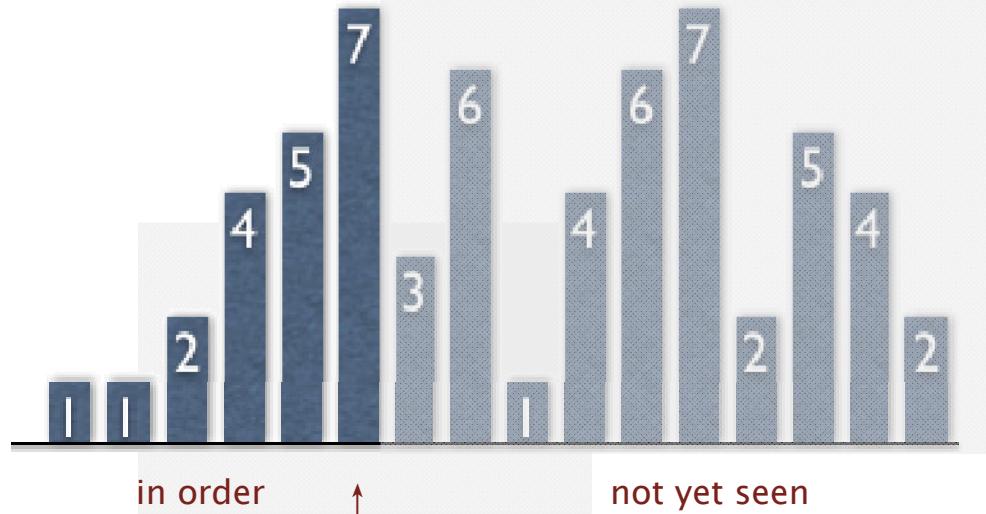
2	13	6	24	43	1	51	9
---	----	---	----	----	---	----	---

	Best case	Worst case
#Comparison:	$n - 1$	$\frac{n(n-1)}{2}$
#Move:	0	$\frac{(n+2)(n-1)}{2}$

Insertion sort

Lemma (Invariants)

- Entries to the left of \uparrow (including \uparrow) are in ascending order.
- Entries to the right of \uparrow have not yet been seen.



Shell sort (3-1)

2	13	6	24	43	1	51	9	10
---	----	---	----	----	---	----	---	----

Correctness: The last round of *shell sort* is *insertion sort*.

Comparison

<http://www.sorting-algorithms.com/>

	worst	average	best	remarks
selection	$N^2 / 2$	$N^2 / 2$	$N^2 / 2$	N exchanges
insertion	$N^2 / 2$	$N^2 / 4$	N	for small N or partially ordered
shell	?	?	N	tight code, subquadratic

Stability and in-place

Definition: Stability

A stable sort preserves the relative order of items with equal keys.

Definition: In-place

A sorting algorithm is in-place if the extra memory it uses $\leq c \log N$.

	in-place?	stable?
selection	?	?
insertion	?	?
shell	?	?

Comparison

	in-place?	stable?	worst	average	best	remarks
selection	x		$N^2 / 2$	$N^2 / 2$	$N^2 / 2$	N exchanges
insertion	x	x	$N^2 / 2$	$N^2 / 4$	N	for small N or partially ordered
shell	x		?	?	N	tight code, subquadratic

Conclusion

Algorithm design techniques: induction

Permutation

Majority element

Radix sort

Selection sort, Insertion sort, Shell sort ...

▫ Proofs and comparisons

Next permutation

1 3 2
Next_perm 2 1 3

1 4 3 2 5
Next_perm 1 4 3 5 2

Exercises

- Download [hw3.pdf](#) from our course homepage
- Due on next Tuesday

Lecture 4 Sort(2)

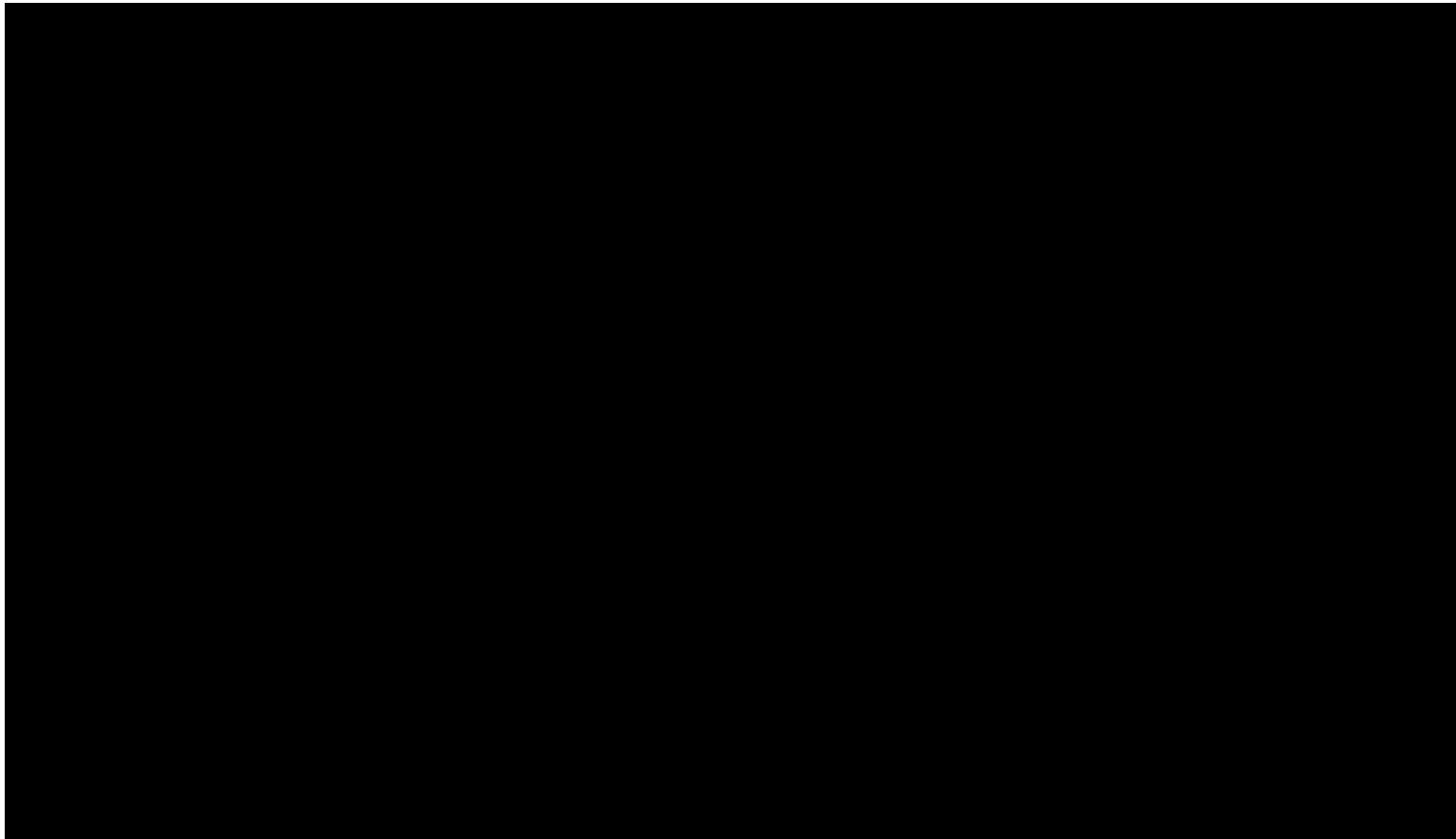
- Merge sort
- Quick sort

ACKNOWLEDGEMENTS: Some contents in this lecture source from COS226 of Princeton University
by [Kevin Wayne](#) and [Bob Sedgewick](#)

Roadmap

- Merge sort
- Quick sort

Merge sort



Merge sort

- Divide array into two halves.
- *Recursively* sort each half.
- Merge two halves.

2		13		6		24		43		1		51		9		10
---	--	----	--	---	--	----	--	----	--	---	--	----	--	---	--	----

2	13	6	24	1	43	9	10
---	----	---	----	---	----	---	----

2	6	13	24	9	10	43	51
---	---	----	----	---	----	----	----

1	2	6	9	10	13	24	43	51
---	---	---	---	----	----	----	----	----

1	2	6	9	10	13	24	43	51
---	---	---	---	----	----	----	----	----

Merging

2, 13, 43, 45, 89

6, 24, 51, 90, 93



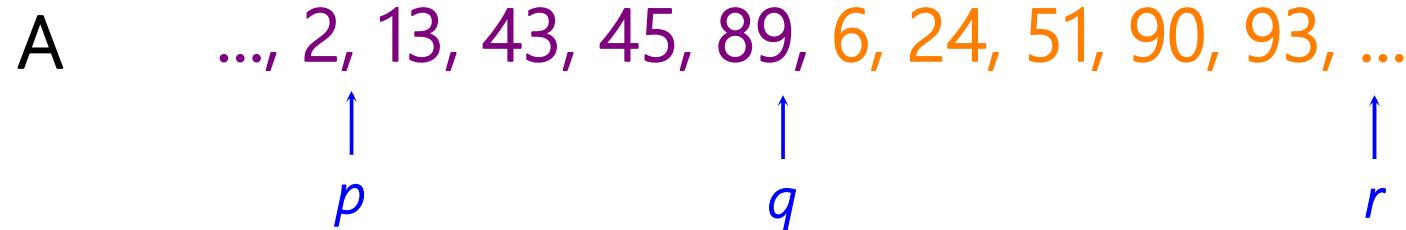
2, 6, 13, 24, 43, 45, 51, 89, 90, 93

Assume we need to merge two sorted arrays, with M , N elements respectively. Then

- How many comparisons in *best* case?
- How many comparisons in *worst* case?

- A. $M+N$ B. $\max\{M,N\}$ C. $\min\{M,N\}$ D. $M+N-1$

MERGE(A, p, q, r)

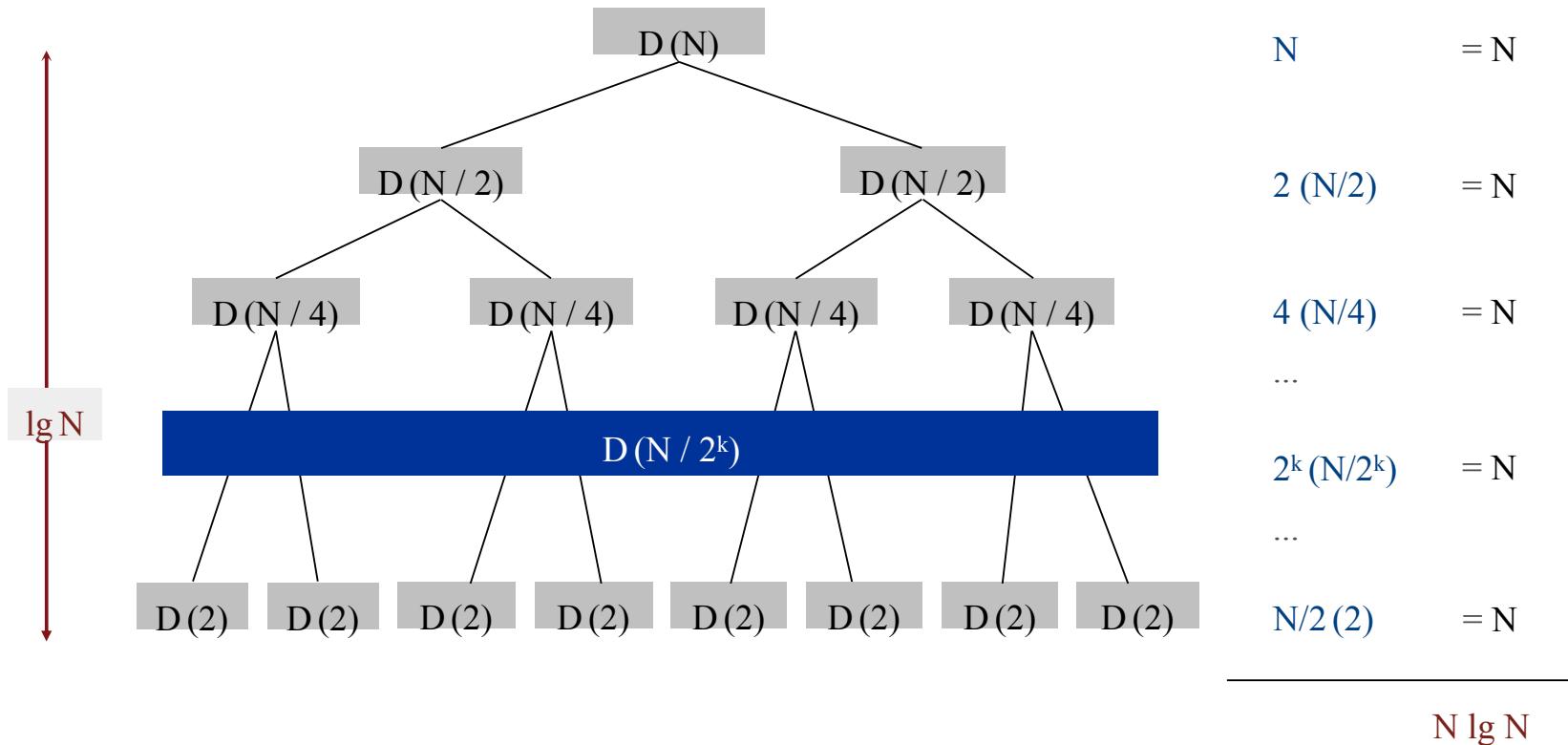


```
1.  $s \leftarrow p; t \leftarrow q + 1; k \leftarrow p$ 
2. while  $s \leq q$  and  $t \leq r$ 
3.   if  $A[s] \leq A[t]$  then
4.      $B[k] \leftarrow A[s]$ 
5.      $s \leftarrow s + 1$ 
6.   else
7.      $B[k] \leftarrow A[t]$ 
8.      $t \leftarrow t + 1$ 
9.   end if
10.   $k \leftarrow k + 1$ 
11. end while
12. if  $s = q + 1$  then  $B[k..r] \leftarrow A[t..r]$ 
13. else  $B[k..r] \leftarrow A[s..q]$ 
14. end if
15.  $A[p..r] \leftarrow B[p..r]$ 
```

Complexity

$$D(N) = 2 D(N/2) + N$$

Compute by picture.



Complexity

$$D(N) = 2 D(N/2) + N$$

Compute by expansion.

$$D(N) = 2 D(N/2) + N$$

$$D(N)/N = 2 D(N/2)/N + 1$$

$$= D(N/2)/(N/2) + 1$$

$$= D(N/4)/(N/4) + 1 + 1$$

$$= D(N/8)/(N/8) + 1 + 1 + 1$$

...

$$= D(N/N)/(N/N) + 1 + 1 + \dots + 1$$

$$= \lg N$$

Complexity

- Laptop executes 10^8 compares/second.
- Supercomputer executes 10^{12} compares/second.

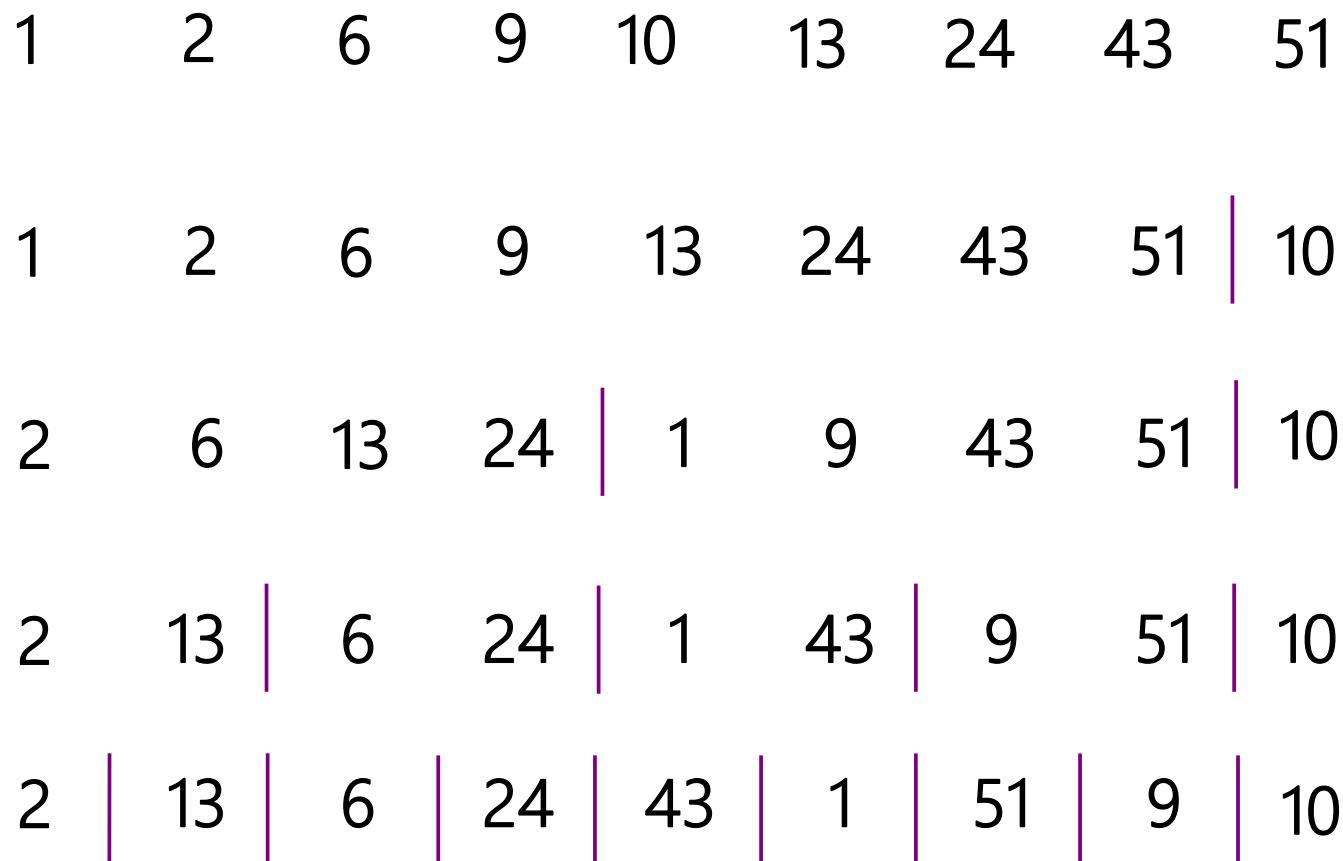
computer	insertion sort (N^2)			mergesort ($N \log N$)		
	thousand	million	billion	thousand	million	billion
home	instant	2.8 hours	317 years	instant	1 second	18 min
super	instant	1 second	1 week	instant	instant	instant

💡 *Good algorithms are better than supercomputers.*

Discussion

- Mergesort is
 - stable?
 - in place?
- Recursions cost spaces.

Bottom-up merge sort



Bottom-up merge sort

Input: An array $A[1..n]$ of n elements.

Output: $A[1..n]$ sorted in nondecreasing order.

1. $t \leftarrow 1$
2. **while** $t < n$
3. $s \leftarrow t$; $t \leftarrow 2s$; $i \leftarrow 0$
4. **while** $i + t \leq n$
5. $\text{Merge}(A, i + 1, i + s, i + t)$
6. $i \leftarrow i + t$
7. **end while**
8. **if** $i + s < n$ **then** $\text{Merge}(A, i + 1, i + s, n)$
9. **end while**

Quiz

Give the array that results immediately after the 7th call to merge() by using:

a) top-down mergesort

b) bottom-up merge sort

M B X V Z Y H U N S I K

- A. B M V X Y Z H N U S I K
- B. B M V X Y Z H N U S K I
- C. B M V X Y Z H U N S I K
- D. B M V X Y Z H N U S K I

Quiz

Mergesort, BottomupMergesort,

which one is more efficient?

which one is easier for understanding and
debugging?

which one you prefer?

- A. *Mergesort*
- B. *BottomupMergesort*

$O(n \log n)$ sorting algorithms

$n=2^k$	Best case	Worst case
Bottom-up-merge	$\frac{n \log n}{2}$	$n \log n - n + 1$
Merge	$\frac{n \log n}{2}$	 Space $\Theta(n)$ $n \log n - n + 1$

Quicksort: worst $O(n^2)$, average $O(n \log n)$, space $O(\log n)$
Heapsort: worst $O(n \log n)$, space $O(1)$

Sorting

n	selectionsort	insertionsort	bottomupsort	mergesort	quicksort
500	124750	62747	3852	3852	6291
1000	499500	261260	8682	8704	15693
1500	1124250	566627	14085	13984	28172
2000	1999000	1000488	19393	19426	34020
2500	3123750	1564522	25951	25111	52513
3000	4498500	2251112	31241	30930	55397
3500	6123250	3088971	37102	36762	67131
4000	7998000	4042842	42882	42859	79432
4500	10122750	5103513	51615	49071	98635
5000	12497500	6180358	56888	55280	106178

Where are we?

- Merge sort
- Quick sort

Quick sort

Quicksort honored as one of top 10 algorithms of 20th century in science and engineering.

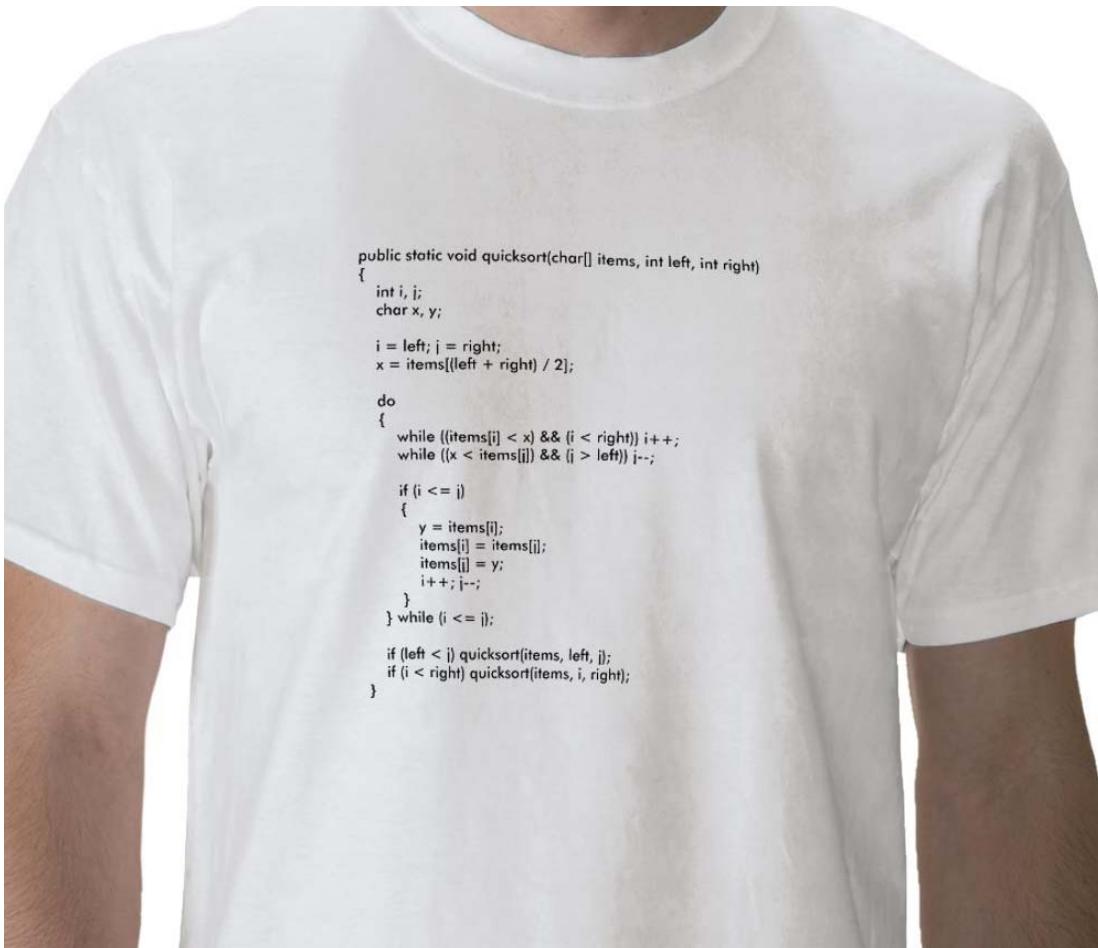
Mergesort.

- Java sort for objects.
- Perl, C++ stable sort, Python stable sort, Firefox
JavaScript, ...

Quicksort.

- Java sort for primitive types.
- C qsort, Unix, Visual C++, Python, Matlab, Chrome
JavaScript, ...

Quick sort



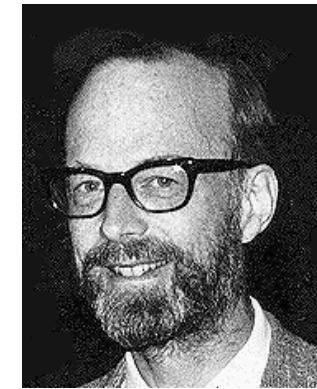
```
public static void quicksort(char[] items, int left, int right)
{
    int i, j;
    char x, y;

    i = left; j = right;
    x = items[(left + right) / 2];

    do
    {
        while ((items[i] < x) && (i < right)) i++;
        while ((x < items[j]) && (j > left)) j--;

        if (i <= j)
        {
            y = items[i];
            items[i] = items[j];
            items[j] = y;
            i++; j--;
        }
    } while (i <= j);

    if (left < j) quicksort(items, left, j);
    if (i < right) quicksort(items, i, right);
}
```



Sir Charles Antony
Richard Hoare
1980 Turing Award

Quick sort

- **Shuffle** the array.
- **Partition** so that, for some j
 - entry $a[j]$ is in place
 - no larger entry to the left of j
 - no smaller entry to the right of j
- **Sort** each piece recursively.

Partition#1

Algorithm 6.5 Split

Input: An array of elements $A[low..high]$.

Output: A with its elements rearranged and w the new position of the splitting element $A[low]$.

1. $i \leftarrow low$
2. $x \leftarrow A[low]$
3. **for** $j \leftarrow low + 1$ **to** $high$
4. **if** $A[j] \leq x$ **then**
5. $i \leftarrow i + 1$
6. **if** $i \neq j$ **then** interchange $A[i]$ and $A[j]$
7. **end if**
8. **end for**
9. interchange $A[low]$ and $A[i]$
10. $w \leftarrow i$
11. **return** A and w

Partition#1

10 13 6 24 43 1 51 9
 i j

Partition #2

- by Sedgewick



```
private static int partition(Comparable[] a, int lo, int hi)
{
    int i = lo, j = hi+1;
    while (true)
    {
        while (less(a[++i], a[lo]))
            if (i == hi) break;

        while (less(a[lo], a[--j]))
            if (j == lo) break;

        if (i >= j) break;
        exch(a, i, j);
    }

    exch(a, lo, j);
    return j;
}
```

Complexity

Best case: array always split into two equal-sized subarrays.
similar to mergesort, $O(N \log N)$

Worst case: array always split into a 0-sized and an $N-1$ -sized subarrays
similar to selection sort, $O(N^2)$

Average case:

$$C_N = (N+1) + \left(\frac{C_0 + C_{N-1}}{N} \right) + \left(\frac{C_1 + C_{N-2}}{N} \right) + \dots + \left(\frac{C_{N-1} + C_0}{N} \right)$$

$$C_N \sim 2(N+1) \ln N \approx 1.39N \lg N$$

Complexity

- Laptop executes 10^8 compares/second.
- Supercomputer executes 10^{12} compares/second.

	insertion sort (N^2)			mergesort ($N \log N$)			quicksort ($N \log N$)		
computer	thousand	million	billion	thousand	million	billion	thousand	million	billion
home	instant	2.8 hours	317 years	instant	1 second	18 min	instant	0.6 sec	12 min
super	instant	1 second	1 week	instant	instant	instant	instant	instant	instant



- Good algorithms are better than supercomputers.
- Great algorithms are better than good algorithms.

Duplicate keys

```
Chicago 09:25:52
Chicago 09:03:13
Chicago 09:21:05
Chicago 09:19:46
Chicago 09:19:32
Chicago 09:00:00
Chicago 09:35:21
Chicago 09:00:59
Houston 09:01:10
Houston 09:00:13
Phoenix 09:37:44
Phoenix 09:00:03
Phoenix 09:14:25
Seattle 09:10:25
Seattle 09:36:14
Seattle 09:22:43
Seattle 09:10:11
Seattle 09:22:54
```

B A A B A B B **B** C C C
A A A A A A A A A A **A**

Dijkstra's 3-way partition

Quiz

Give the array that results after applying quicksort partitioning to the following array by using:

- a) partition #1
- b) partition #2
- c) Dijkstra's 3-way partition

H J B R H R H B C V S

- A. C B H H B H R J R V S
- B. H C B B H H R R J V S
- C. C B B H H H R R V S J
- D. none of above

Sort summary

	inplace?	stable?	worst	average	best	remarks
selection	x		$N^2 / 2$	$N^2 / 2$	$N^2 / 2$	N exchanges
insertion	x	x	$N^2 / 2$	$N^2 / 4$	N	use for small N or partially ordered
shell	x		?	?	N	tight code, subquadratic
quick	x		$N^2 / 2$	$2N \ln N$	$N \lg N$	$N \log N$ probabilistic guarantee fastest in practice
3-way quick	x		$N^2 / 2$	$2N \ln N$	N	improves quicksort in presence of duplicate keys
merge		x	$N \lg N$	$N \lg N$	$N \lg N$	$N \log N$ guarantee, stable

Exercise

- Download [hw4.pdf](#) from our course homepage
- Due on next Tuesday.

Quicksort Partitioning Demo



- Sedgewick 2-way
- Dijkstra 3-way
- Bentley-McIlroy 3-way

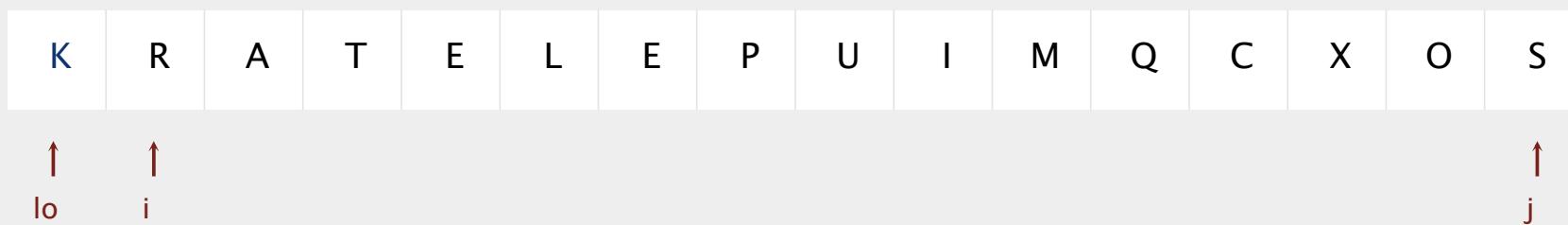
Sedgewick 2-way Partitioning



Quicksort partitioning

Repeat until i and j pointers cross.

- Scan i from left to right so long as $a[i] < a[lo]$.
- Scan j from right to left so long as $a[j] > a[lo]$.
- Exchange $a[i]$ with $a[j]$.

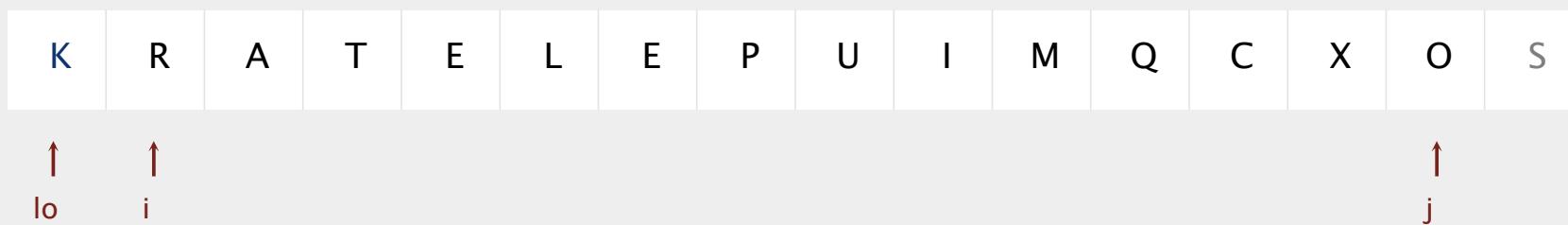


stop i scan because $a[i] \geq a[lo]$

Quicksort partitioning

Repeat until i and j pointers cross.

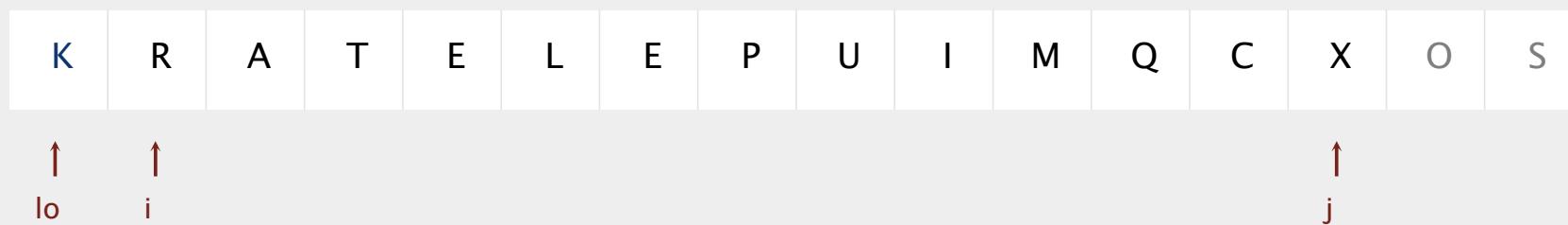
- Scan i from left to right so long as $a[i] < a[lo]$.
- Scan j from right to left so long as $a[j] > a[lo]$.
- Exchange $a[i]$ with $a[j]$.



Quicksort partitioning

Repeat until i and j pointers cross.

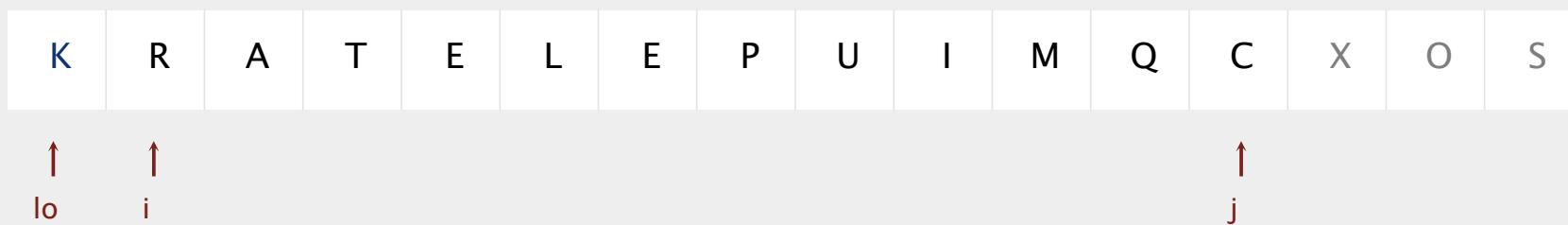
- Scan i from left to right so long as $a[i] < a[lo]$.
- Scan j from right to left so long as $a[j] > a[lo]$.
- Exchange $a[i]$ with $a[j]$.



Quicksort partitioning

Repeat until i and j pointers cross.

- Scan i from left to right so long as $a[i] < a[lo]$.
- Scan j from right to left so long as $a[j] > a[lo]$.
- Exchange $a[i]$ with $a[j]$.

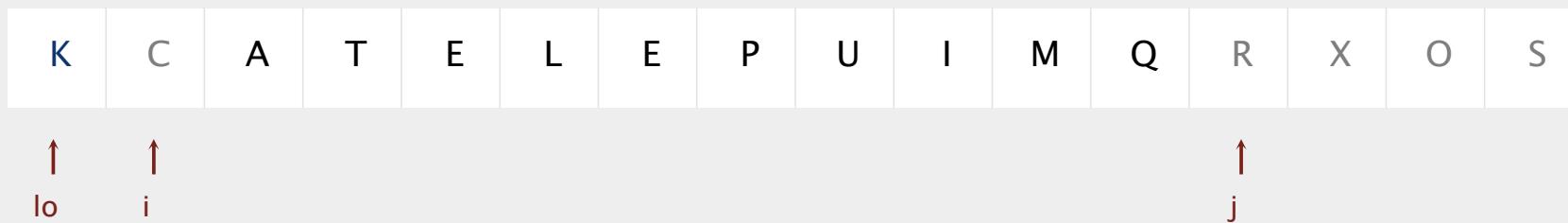


stop j scan and exchange $a[i]$ with $a[j]$

Quicksort partitioning

Repeat until i and j pointers cross.

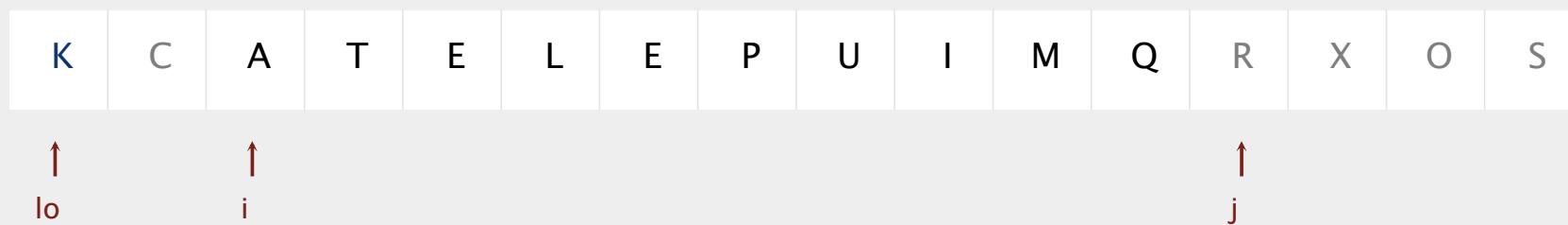
- Scan i from left to right so long as $a[i] < a[lo]$.
- Scan j from right to left so long as $a[j] > a[lo]$.
- Exchange $a[i]$ with $a[j]$.



Quicksort partitioning

Repeat until i and j pointers cross.

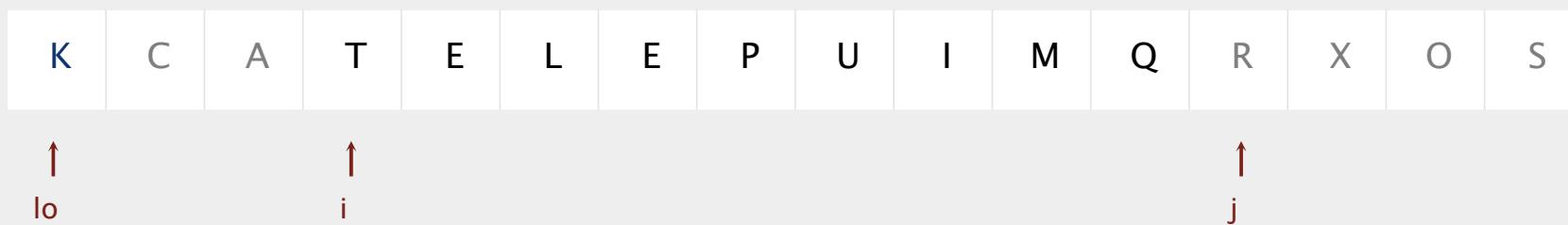
- Scan i from left to right so long as $a[i] < a[lo]$.
- Scan j from right to left so long as $a[j] > a[lo]$.
- Exchange $a[i]$ with $a[j]$.



Quicksort partitioning

Repeat until i and j pointers cross.

- Scan i from left to right so long as $a[i] < a[lo]$.
- Scan j from right to left so long as $a[j] > a[lo]$.
- Exchange $a[i]$ with $a[j]$.

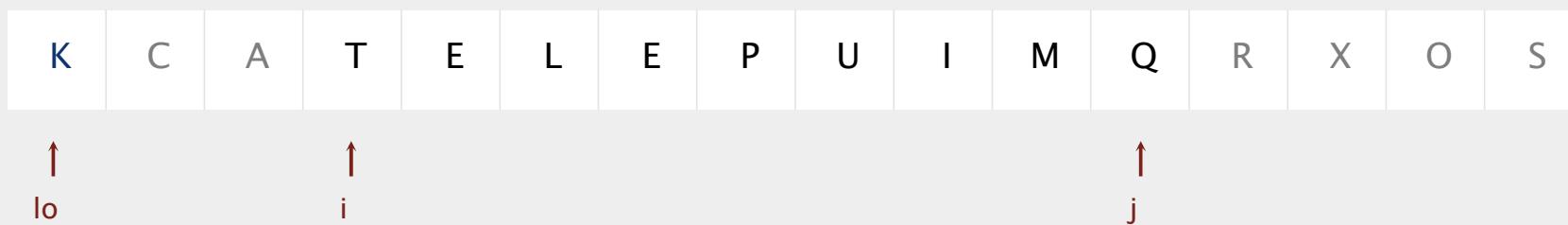


stop i scan because $a[i] \geq a[lo]$

Quicksort partitioning

Repeat until i and j pointers cross.

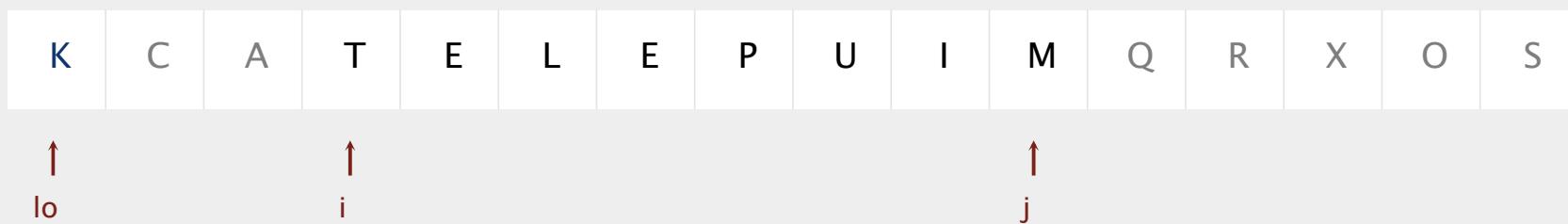
- Scan i from left to right so long as $a[i] < a[lo]$.
- Scan j from right to left so long as $a[j] > a[lo]$.
- Exchange $a[i]$ with $a[j]$.



Quicksort partitioning

Repeat until i and j pointers cross.

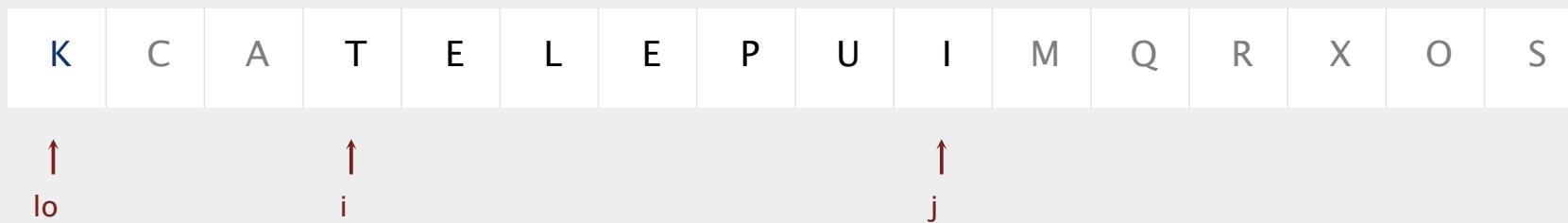
- Scan i from left to right so long as $a[i] < a[lo]$.
- Scan j from right to left so long as $a[j] > a[lo]$.
- Exchange $a[i]$ with $a[j]$.



Quicksort partitioning

Repeat until i and j pointers cross.

- Scan i from left to right so long as $a[i] < a[lo]$.
- Scan j from right to left so long as $a[j] > a[lo]$.
- Exchange $a[i]$ with $a[j]$.

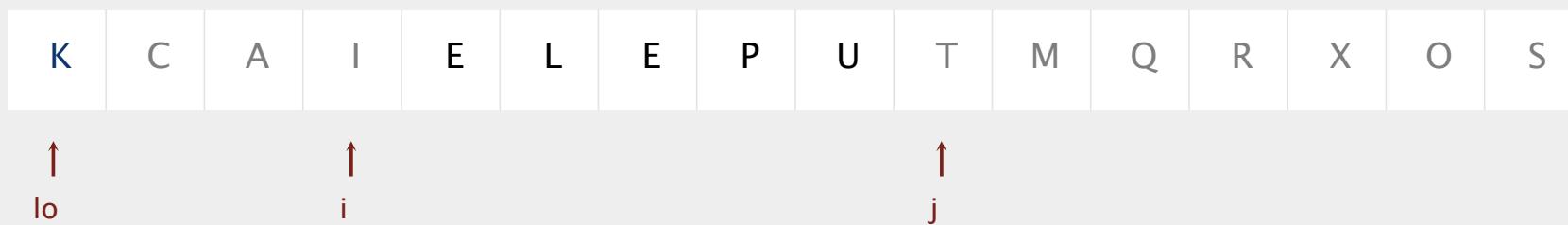


stop j scan and exchange $a[i]$ with $a[j]$

Quicksort partitioning

Repeat until i and j pointers cross.

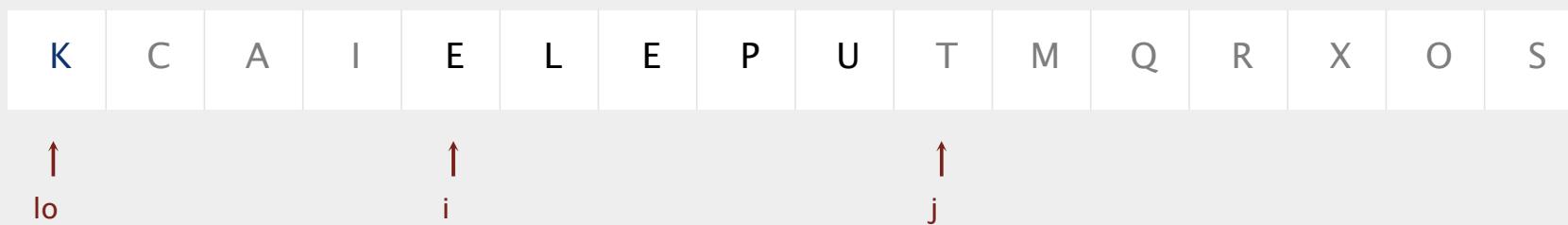
- Scan i from left to right so long as $a[i] < a[lo]$.
- Scan j from right to left so long as $a[j] > a[lo]$.
- Exchange $a[i]$ with $a[j]$.



Quicksort partitioning

Repeat until i and j pointers cross.

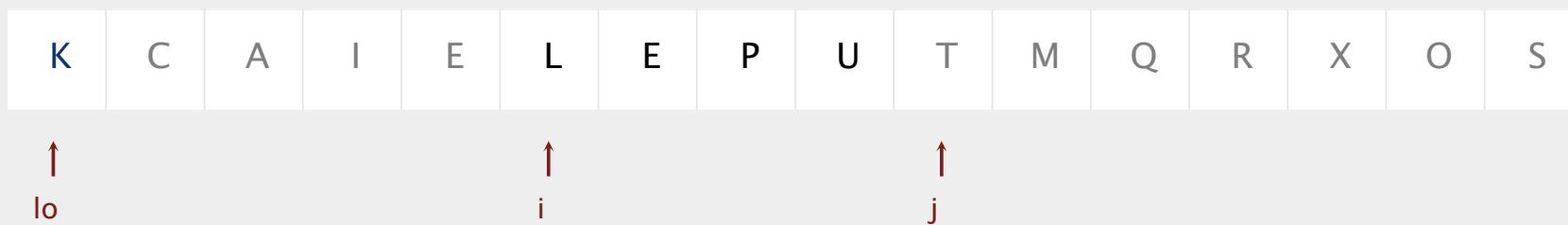
- Scan i from left to right so long as $a[i] < a[lo]$.
- Scan j from right to left so long as $a[j] > a[lo]$.
- Exchange $a[i]$ with $a[j]$.



Quicksort partitioning

Repeat until i and j pointers cross.

- Scan i from left to right so long as $a[i] < a[lo]$.
- Scan j from right to left so long as $a[j] > a[lo]$.
- Exchange $a[i]$ with $a[j]$.



stop i scan because $a[i] \geq a[lo]$

Quicksort partitioning

Repeat until i and j pointers cross.

- Scan i from left to right so long as $a[i] < a[lo]$.
- Scan j from right to left so long as $a[j] > a[lo]$.
- Exchange $a[i]$ with $a[j]$.



Quicksort partitioning

Repeat until i and j pointers cross.

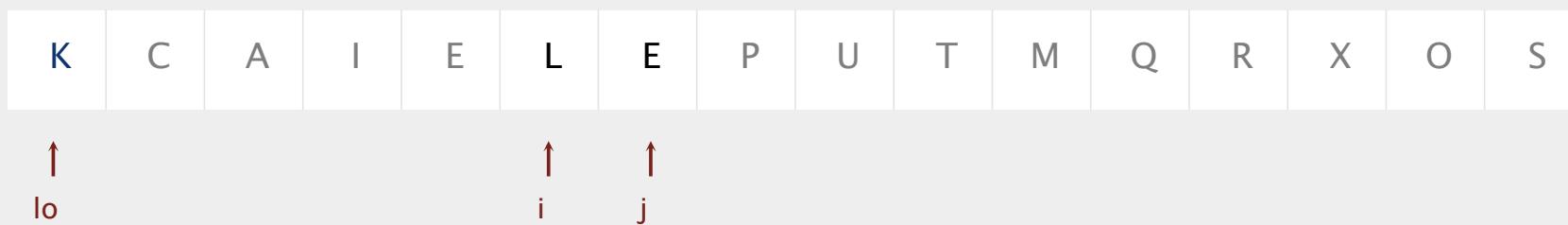
- Scan i from left to right so long as $a[i] < a[lo]$.
- Scan j from right to left so long as $a[j] > a[lo]$.
- Exchange $a[i]$ with $a[j]$.



Quicksort partitioning

Repeat until i and j pointers cross.

- Scan i from left to right so long as $a[i] < a[lo]$.
- Scan j from right to left so long as $a[j] > a[lo]$.
- Exchange $a[i]$ with $a[j]$.

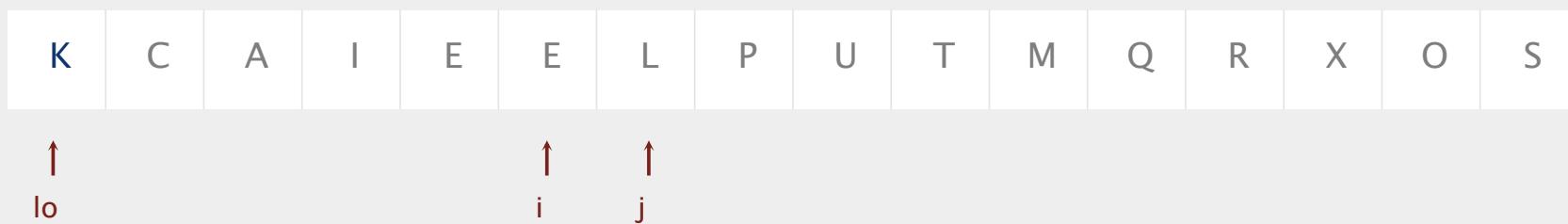


stop j scan and exchange $a[i]$ with $a[j]$

Quicksort partitioning

Repeat until i and j pointers cross.

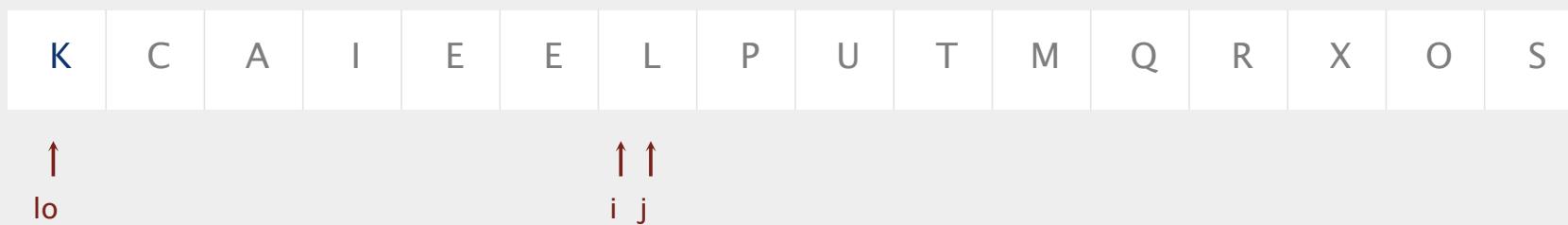
- Scan i from left to right so long as $a[i] < a[lo]$.
- Scan j from right to left so long as $a[j] > a[lo]$.
- Exchange $a[i]$ with $a[j]$.



Quicksort partitioning

Repeat until i and j pointers cross.

- Scan i from left to right so long as $a[i] < a[lo]$.
- Scan j from right to left so long as $a[j] > a[lo]$.
- Exchange $a[i]$ with $a[j]$.

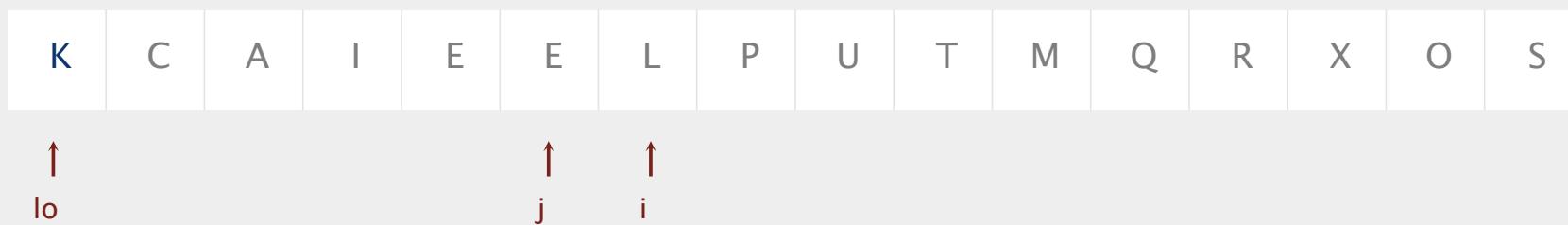


stop i scan because $a[i] \geq a[lo]$

Quicksort partitioning

Repeat until i and j pointers cross.

- Scan i from left to right so long as $a[i] < a[lo]$.
- Scan j from right to left so long as $a[j] > a[lo]$.
- Exchange $a[i]$ with $a[j]$.



stop j scan because $a[j] \leq a[lo]$

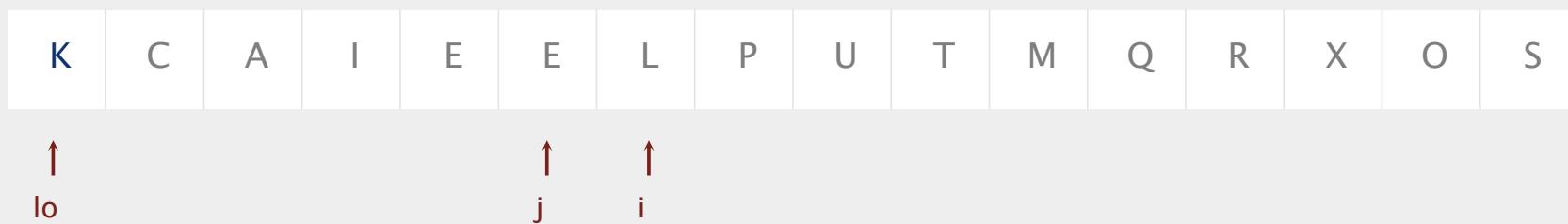
Quicksort partitioning

Repeat until i and j pointers cross.

- Scan i from left to right so long as $a[i] < a[lo]$.
- Scan j from right to left so long as $a[j] > a[lo]$.
- Exchange $a[i]$ with $a[j]$.

When pointers cross.

- Exchange $a[lo]$ with $a[j]$.



pointers cross: exchange $a[lo]$ with $a[j]$

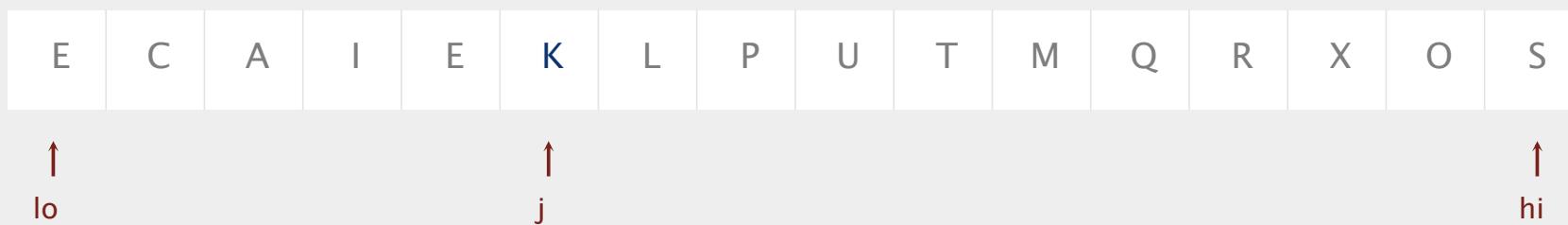
Quicksort partitioning

Repeat until i and j pointers cross.

- Scan i from left to right so long as $a[i] < a[lo]$.
- Scan j from right to left so long as $a[j] > a[lo]$.
- Exchange $a[i]$ with $a[j]$.

When pointers cross.

- Exchange $a[lo]$ with $a[j]$.



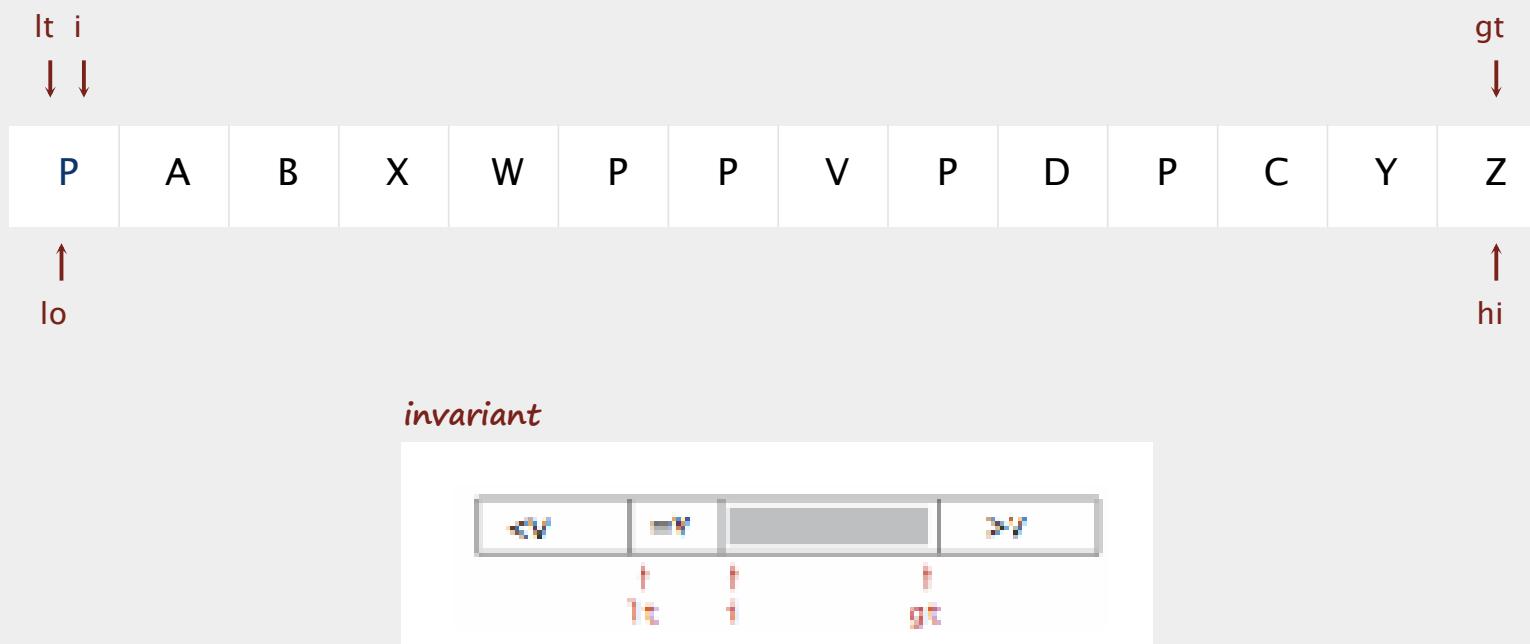
partitioned!

Dijkstra 3-Way Partitioning



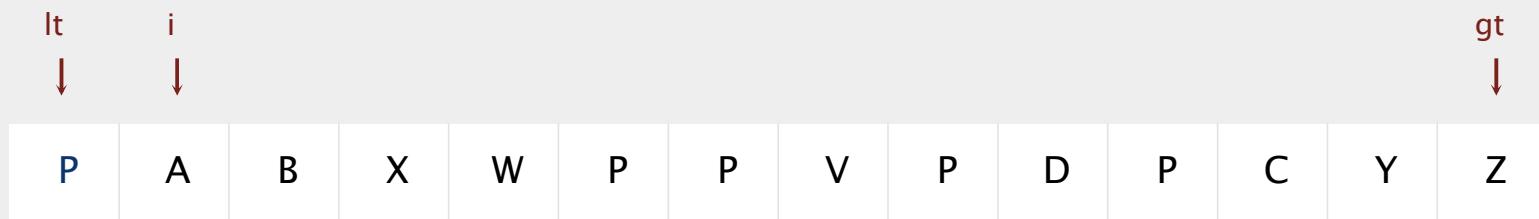
Dijkstra 3-way partitioning

- Let v be partitioning item $a[lo]$.
- Scan i from left to right.
 - $(a[i] < v)$: exchange $a[lt]$ with $a[i]$ and increment both lt and i
 - $(a[i] > v)$: exchange $a[gt]$ with $a[i]$ and decrement gt
 - $(a[i] == v)$: increment i

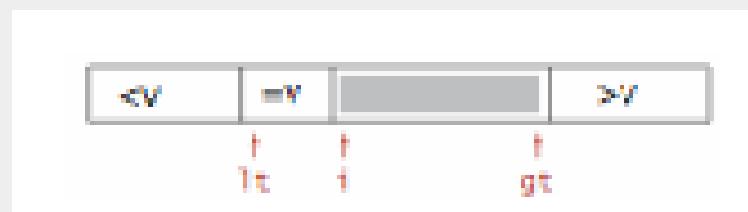


Dijkstra 3-way partitioning

- Let v be partitioning item $a[lo]$.
- Scan i from left to right.
 - $(a[i] < v)$: exchange $a[lt]$ with $a[i]$ and increment both lt and i
 - $(a[i] > v)$: exchange $a[gt]$ with $a[i]$ and decrement gt
 - $(a[i] == v)$: increment i

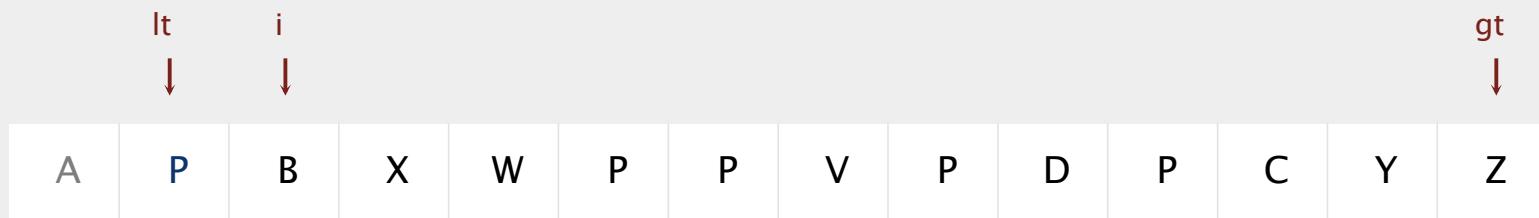


invariant

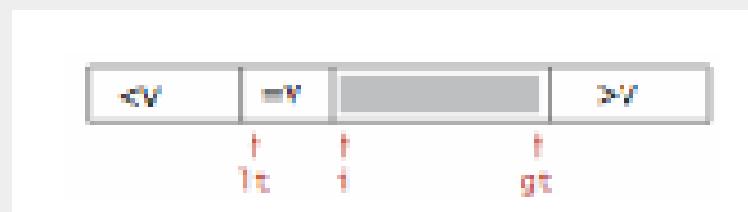


Dijkstra 3-way partitioning

- Let v be partitioning item $a[lo]$.
- Scan i from left to right.
 - $(a[i] < v)$: exchange $a[lt]$ with $a[i]$ and increment both lt and i
 - $(a[i] > v)$: exchange $a[gt]$ with $a[i]$ and decrement gt
 - $(a[i] == v)$: increment i

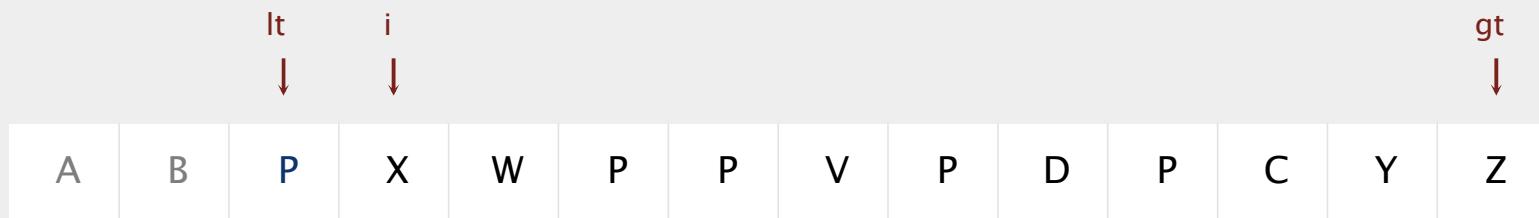


invariant

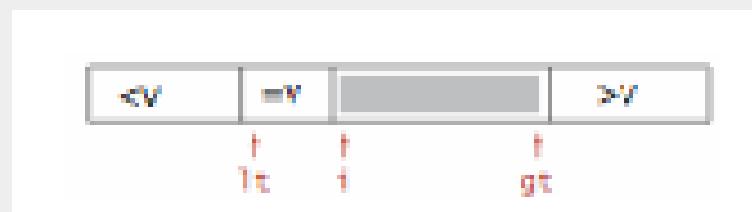


Dijkstra 3-way partitioning

- Let v be partitioning item $a[lo]$.
- Scan i from left to right.
 - $(a[i] < v)$: exchange $a[lt]$ with $a[i]$ and increment both lt and i
 - $(a[i] > v)$: exchange $a[gt]$ with $a[i]$ and decrement gt
 - $(a[i] == v)$: increment i

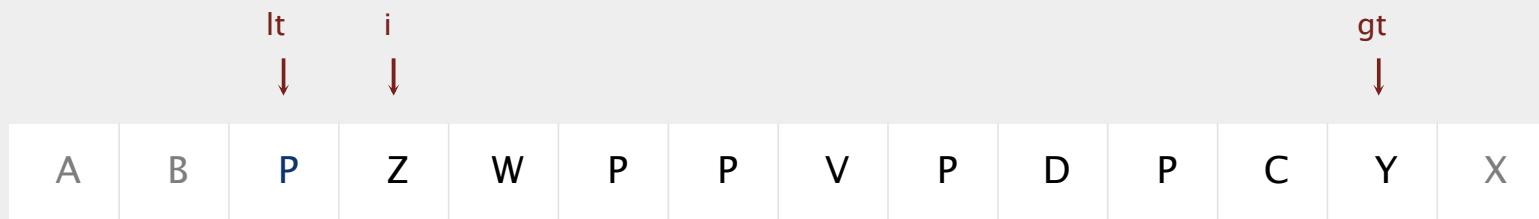


invariant

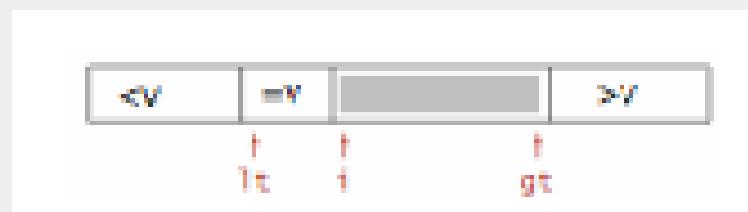


Dijkstra 3-way partitioning

- Let v be partitioning item $a[lo]$.
- Scan i from left to right.
 - $(a[i] < v)$: exchange $a[lt]$ with $a[i]$ and increment both lt and i
 - $(a[i] > v)$: exchange $a[gt]$ with $a[i]$ and decrement gt
 - $(a[i] == v)$: increment i

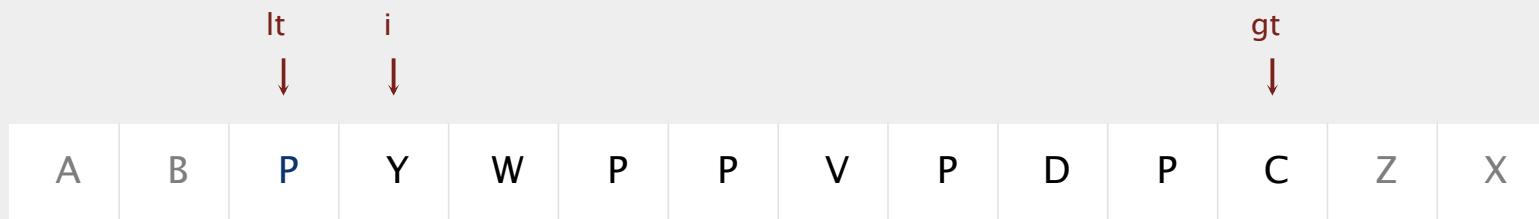


invariant

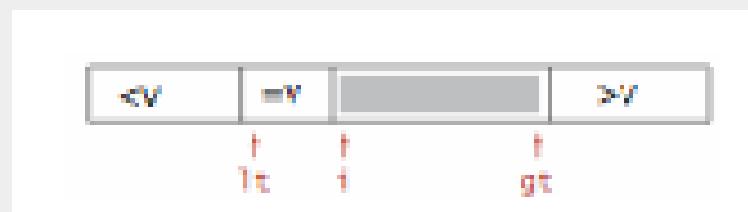


Dijkstra 3-way partitioning

- Let v be partitioning item $a[lo]$.
- Scan i from left to right.
 - $(a[i] < v)$: exchange $a[lt]$ with $a[i]$ and increment both lt and i
 - $(a[i] > v)$: exchange $a[gt]$ with $a[i]$ and decrement gt
 - $(a[i] == v)$: increment i

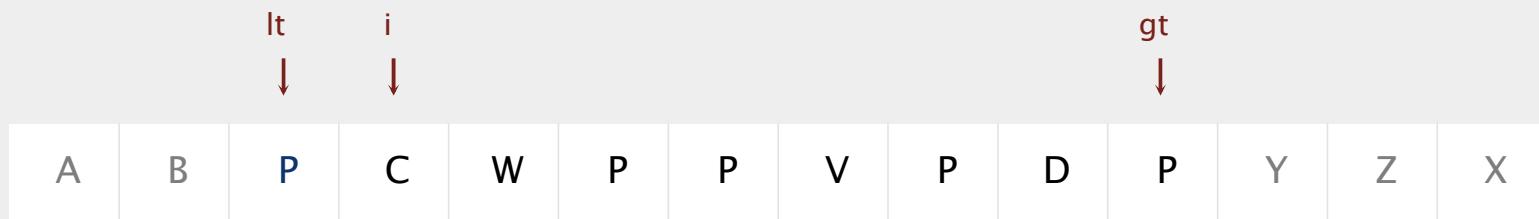


invariant

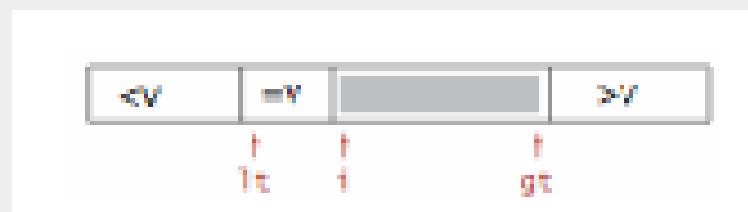


Dijkstra 3-way partitioning

- Let v be partitioning item $a[lo]$.
- Scan i from left to right.
 - $(a[i] < v)$: exchange $a[lt]$ with $a[i]$ and increment both lt and i
 - $(a[i] > v)$: exchange $a[gt]$ with $a[i]$ and decrement gt
 - $(a[i] == v)$: increment i

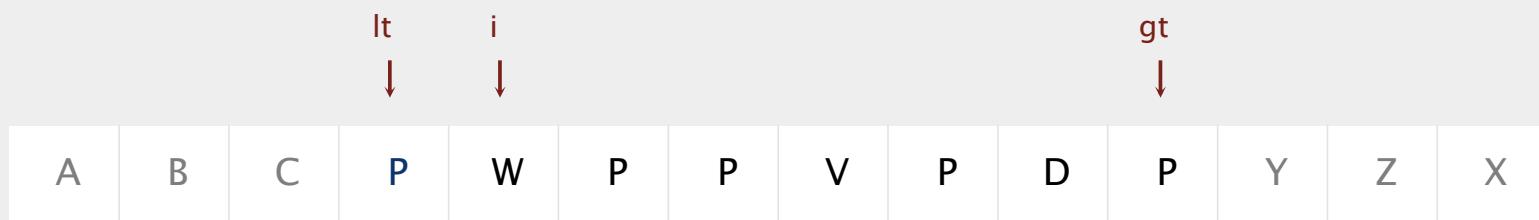


invariant

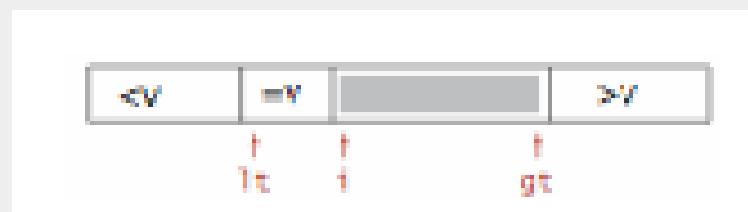


Dijkstra 3-way partitioning

- Let v be partitioning item $a[lo]$.
- Scan i from left to right.
 - $(a[i] < v)$: exchange $a[lt]$ with $a[i]$ and increment both lt and i
 - $(a[i] > v)$: exchange $a[gt]$ with $a[i]$ and decrement gt
 - $(a[i] == v)$: increment i

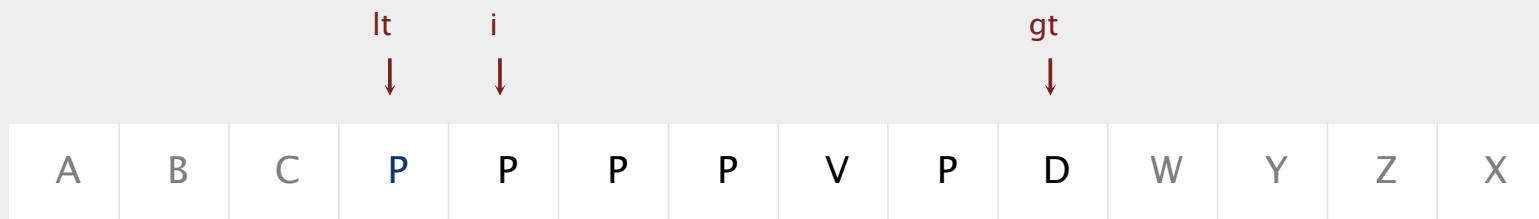


invariant

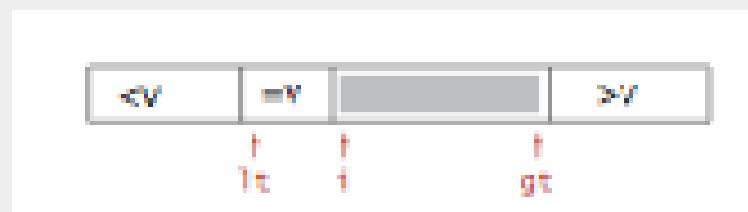


Dijkstra 3-way partitioning

- Let v be partitioning item $a[lo]$.
- Scan i from left to right.
 - $(a[i] < v)$: exchange $a[lt]$ with $a[i]$ and increment both lt and i
 - $(a[i] > v)$: exchange $a[gt]$ with $a[i]$ and decrement gt
 - $(a[i] == v)$: increment i

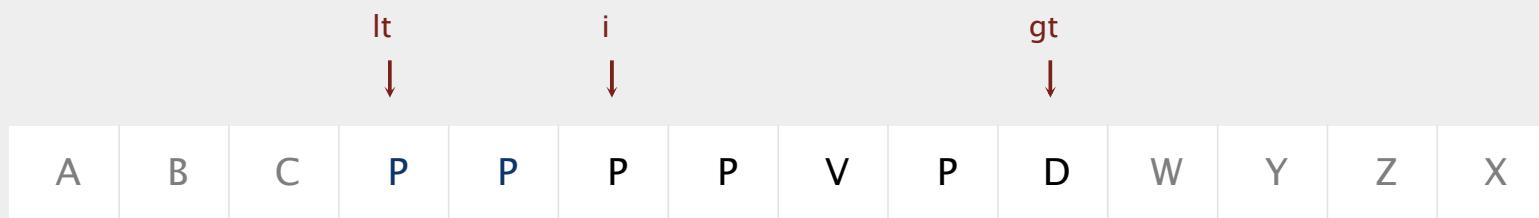


invariant

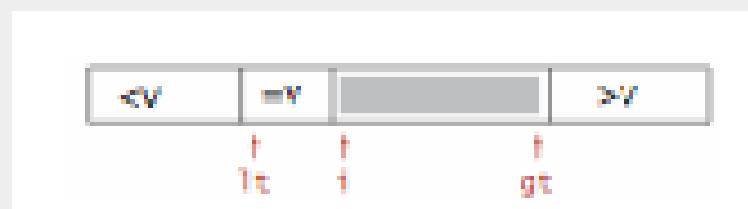


Dijkstra 3-way partitioning

- Let v be partitioning item $a[lo]$.
- Scan i from left to right.
 - $(a[i] < v)$: exchange $a[lt]$ with $a[i]$ and increment both lt and i
 - $(a[i] > v)$: exchange $a[gt]$ with $a[i]$ and decrement gt
 - $(a[i] == v)$: increment i



invariant

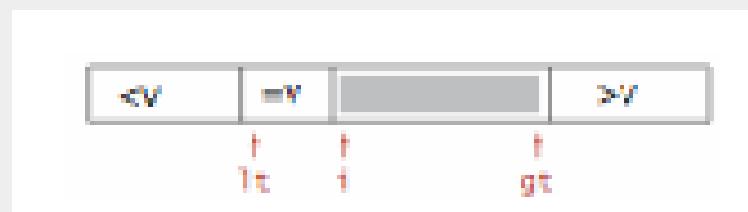


Dijkstra 3-way partitioning

- Let v be partitioning item $a[lo]$.
- Scan i from left to right.
 - $(a[i] < v)$: exchange $a[lt]$ with $a[i]$ and increment both lt and i
 - $(a[i] > v)$: exchange $a[gt]$ with $a[i]$ and decrement gt
 - $(a[i] == v)$: increment i



invariant

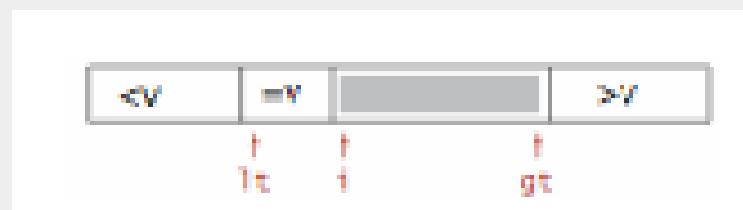


Dijkstra 3-way partitioning

- Let v be partitioning item $a[lo]$.
- Scan i from left to right.
 - $(a[i] < v)$: exchange $a[lt]$ with $a[i]$ and increment both lt and i
 - $(a[i] > v)$: exchange $a[gt]$ with $a[i]$ and decrement gt
 - $(a[i] == v)$: increment i



invariant

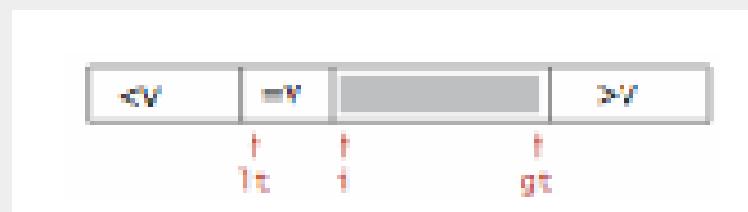


Dijkstra 3-way partitioning

- Let v be partitioning item $a[lo]$.
- Scan i from left to right.
 - $(a[i] < v)$: exchange $a[lt]$ with $a[i]$ and increment both lt and i
 - $(a[i] > v)$: exchange $a[gt]$ with $a[i]$ and decrement gt
 - $(a[i] == v)$: increment i

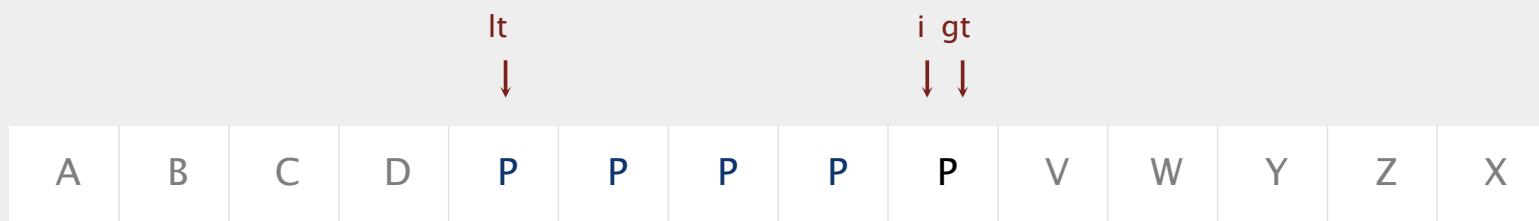


invariant

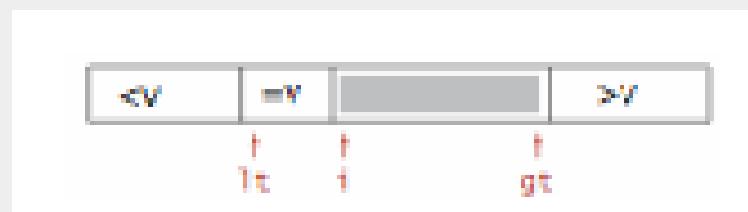


Dijkstra 3-way partitioning

- Let v be partitioning item $a[lo]$.
- Scan i from left to right.
 - $(a[i] < v)$: exchange $a[lt]$ with $a[i]$ and increment both lt and i
 - $(a[i] > v)$: exchange $a[gt]$ with $a[i]$ and decrement gt
 - $(a[i] == v)$: increment i

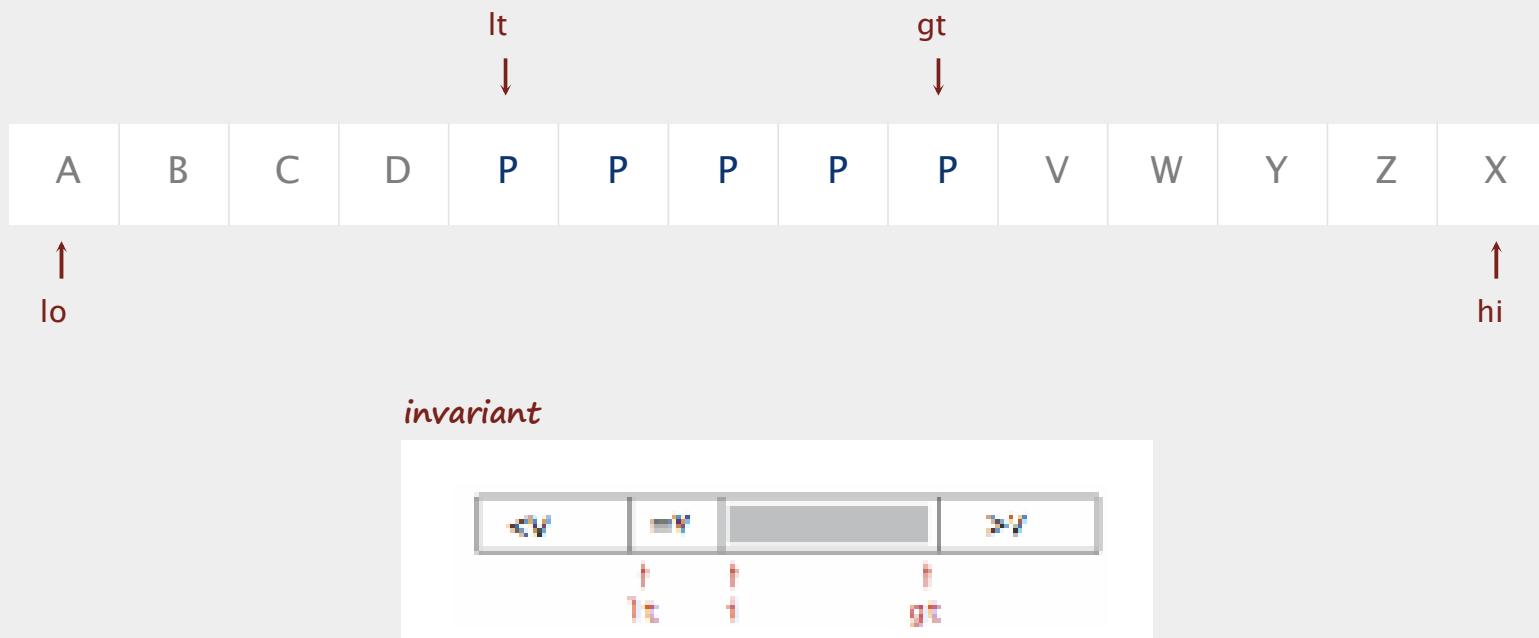


invariant



Dijkstra 3-way partitioning

- Let v be partitioning item $a[lo]$.
- Scan i from left to right.
 - $(a[i] < v)$: exchange $a[lt]$ with $a[i]$ and increment both lt and i
 - $(a[i] > v)$: exchange $a[gt]$ with $a[i]$ and decrement gt
 - $(a[i] == v)$: increment i



Bentley-McIlroy 3-Way Partitioning



Bentley-McIlroy 3-way partitioning

Repeat until i and j pointers cross.

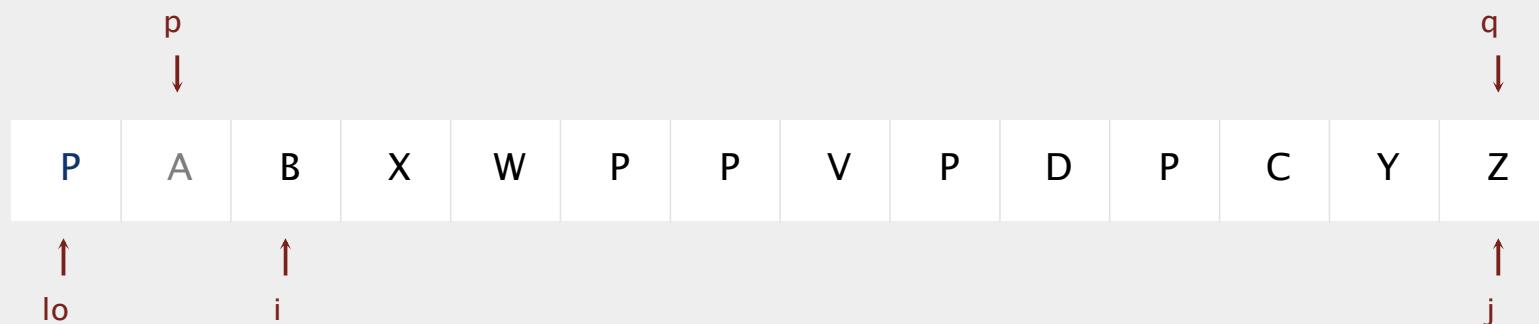
- Scan i from left to right so long as $a[i] < a[lo]$.
- Scan j from right to left so long as $a[j] > a[lo]$.
- Exchange $a[i]$ with $a[j]$.
- If $a[i] == a[lo]$, exchange $a[i]$ with $a[p]$ and increment p .
- If $a[j] == a[lo]$, exchange $a[j]$ with $a[q]$ and decrement q .



Bentley-McIlroy 3-way partitioning

Repeat until i and j pointers cross.

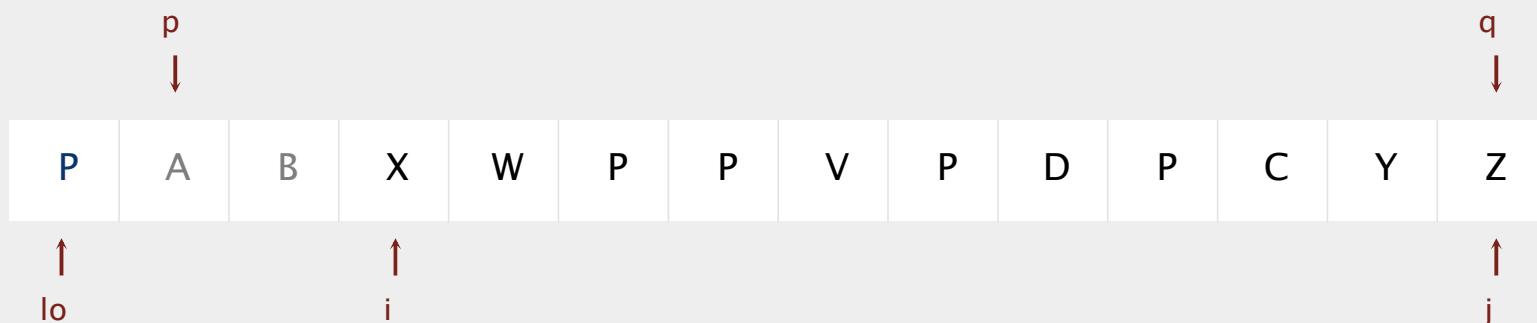
- Scan i from left to right so long as $a[i] < a[lo]$.
- Scan j from right to left so long as $a[j] > a[lo]$.
- Exchange $a[i]$ with $a[j]$.
- If $a[i] == a[lo]$, exchange $a[i]$ with $a[p]$ and increment p .
- If $a[j] == a[lo]$, exchange $a[j]$ with $a[q]$ and decrement q .



Bentley-McIlroy 3-way partitioning

Repeat until i and j pointers cross.

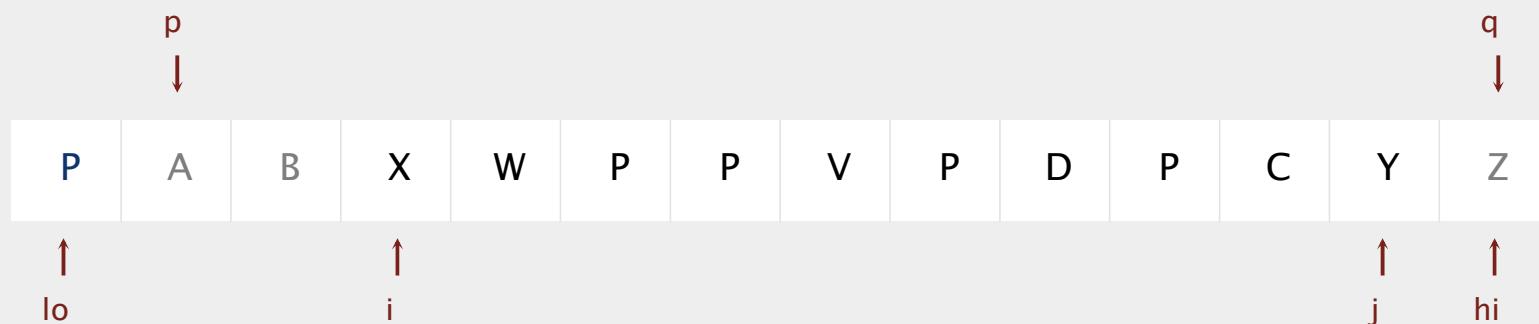
- Scan i from left to right so long as $a[i] < a[lo]$.
- Scan j from right to left so long as $a[j] > a[lo]$.
- Exchange $a[i]$ with $a[j]$.
- If $a[i] == a[lo]$, exchange $a[i]$ with $a[p]$ and increment p .
- If $a[j] == a[lo]$, exchange $a[j]$ with $a[q]$ and decrement q .



Bentley-McIlroy 3-way partitioning

Repeat until i and j pointers cross.

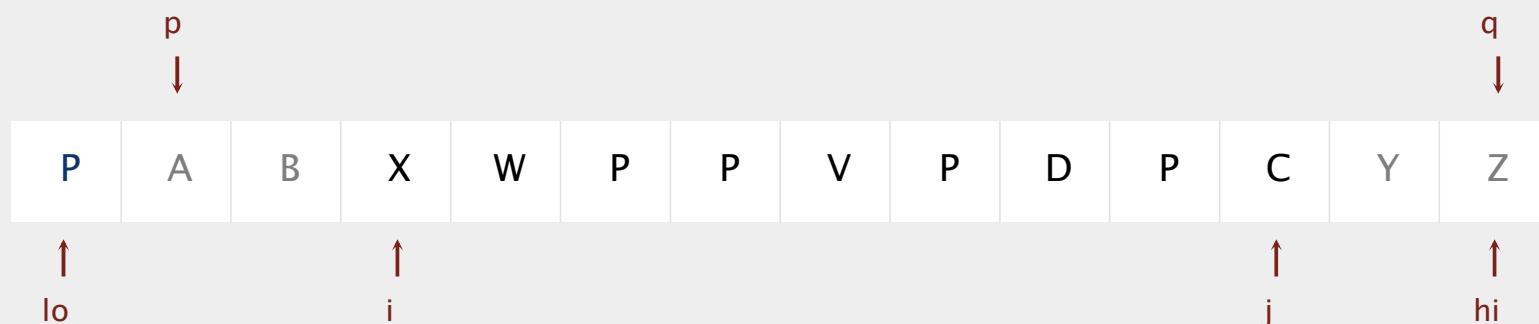
- Scan i from left to right so long as $a[i] < a[lo]$.
- Scan j from right to left so long as $a[j] > a[lo]$.
- Exchange $a[i]$ with $a[j]$.
- If $a[i] == a[lo]$, exchange $a[i]$ with $a[p]$ and increment p .
- If $a[j] == a[lo]$, exchange $a[j]$ with $a[q]$ and decrement q .



Bentley-McIlroy 3-way partitioning

Repeat until i and j pointers cross.

- Scan i from left to right so long as $a[i] < a[lo]$.
- Scan j from right to left so long as $a[j] > a[lo]$.
- Exchange $a[i]$ with $a[j]$.
- If $a[i] == a[lo]$, exchange $a[i]$ with $a[p]$ and increment p .
- If $a[j] == a[lo]$, exchange $a[j]$ with $a[q]$ and decrement q .

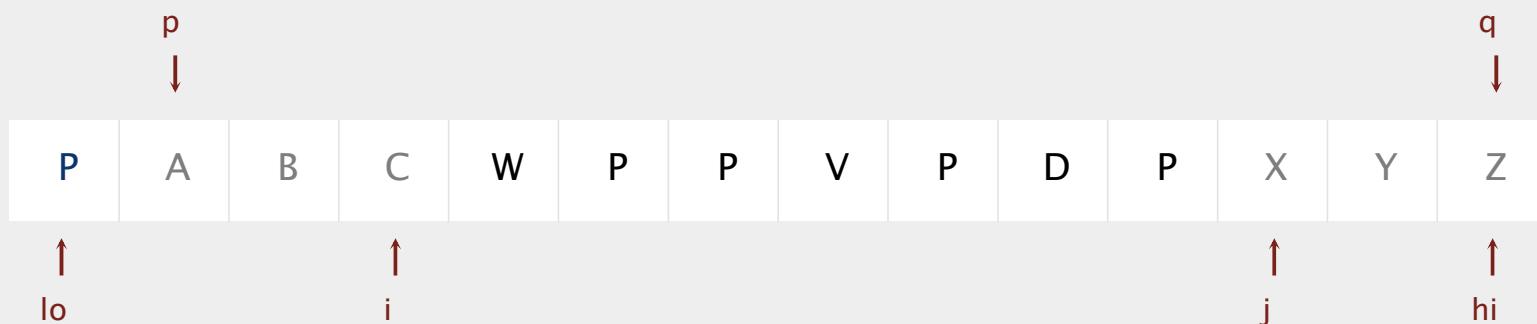


exchange $a[i]$ with $a[j]$

Bentley-McIlroy 3-way partitioning

Repeat until i and j pointers cross.

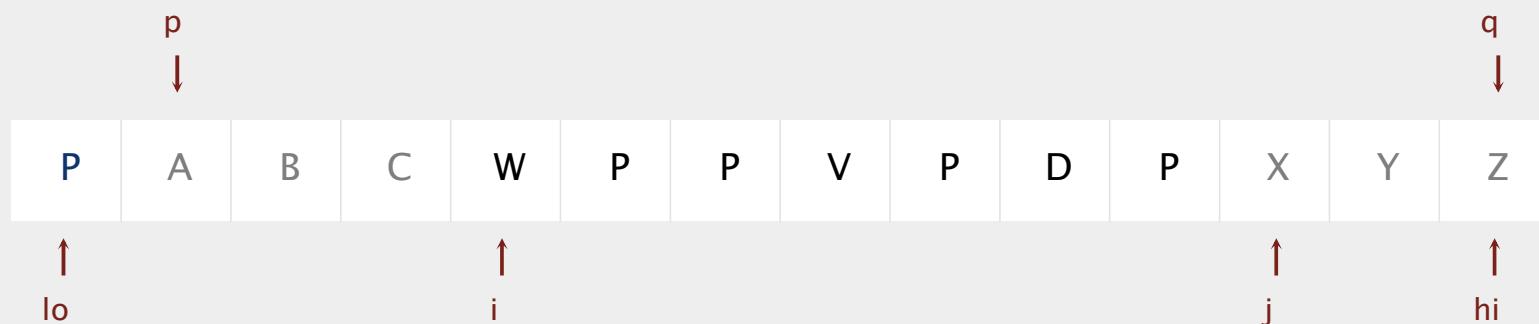
- Scan i from left to right so long as $a[i] < a[lo]$.
- Scan j from right to left so long as $a[j] > a[lo]$.
- Exchange $a[i]$ with $a[j]$.
- If $a[i] == a[lo]$, exchange $a[i]$ with $a[p]$ and increment p .
- If $a[j] == a[lo]$, exchange $a[j]$ with $a[q]$ and decrement q .



Bentley-McIlroy 3-way partitioning

Repeat until i and j pointers cross.

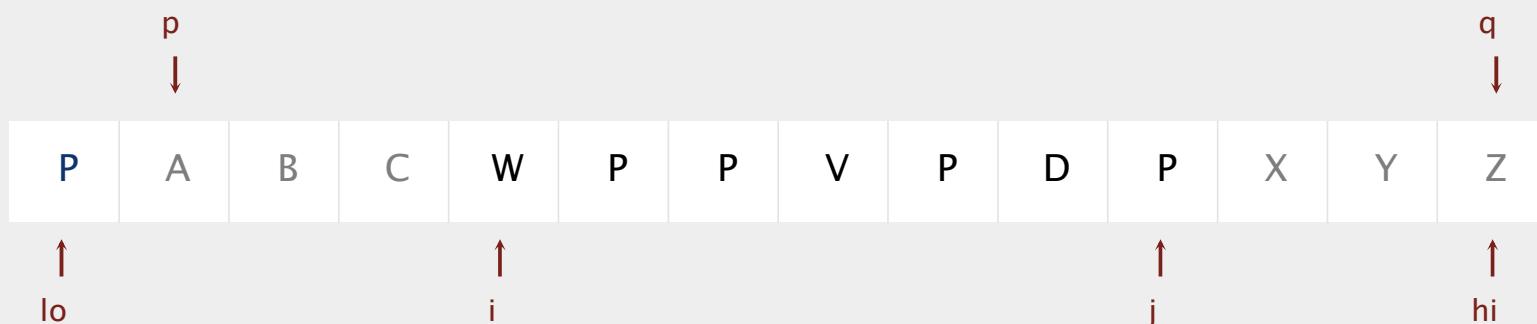
- Scan i from left to right so long as $a[i] < a[lo]$.
- Scan j from right to left so long as $a[j] > a[lo]$.
- Exchange $a[i]$ with $a[j]$.
- If $a[i] == a[lo]$, exchange $a[i]$ with $a[p]$ and increment p .
- If $a[j] == a[lo]$, exchange $a[j]$ with $a[q]$ and decrement q .



Bentley-McIlroy 3-way partitioning

Repeat until i and j pointers cross.

- Scan i from left to right so long as $a[i] < a[lo]$.
- Scan j from right to left so long as $a[j] > a[lo]$.
- Exchange $a[i]$ with $a[j]$.
- If $a[i] == a[lo]$, exchange $a[i]$ with $a[p]$ and increment p .
- If $a[j] == a[lo]$, exchange $a[j]$ with $a[q]$ and decrement q .

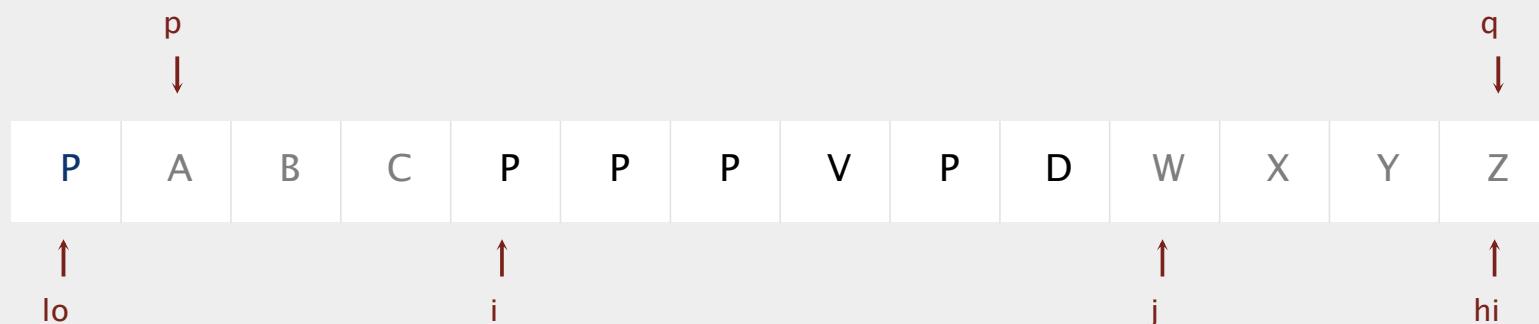


exchange $a[i]$ with $a[j]$

Bentley-McIlroy 3-way partitioning

Repeat until i and j pointers cross.

- Scan i from left to right so long as $a[i] < a[lo]$.
- Scan j from right to left so long as $a[j] > a[lo]$.
- Exchange $a[i]$ with $a[j]$.
- If $a[i] == a[lo]$, exchange $a[i]$ with $a[p]$ and increment p .
- If $a[j] == a[lo]$, exchange $a[j]$ with $a[q]$ and decrement q .

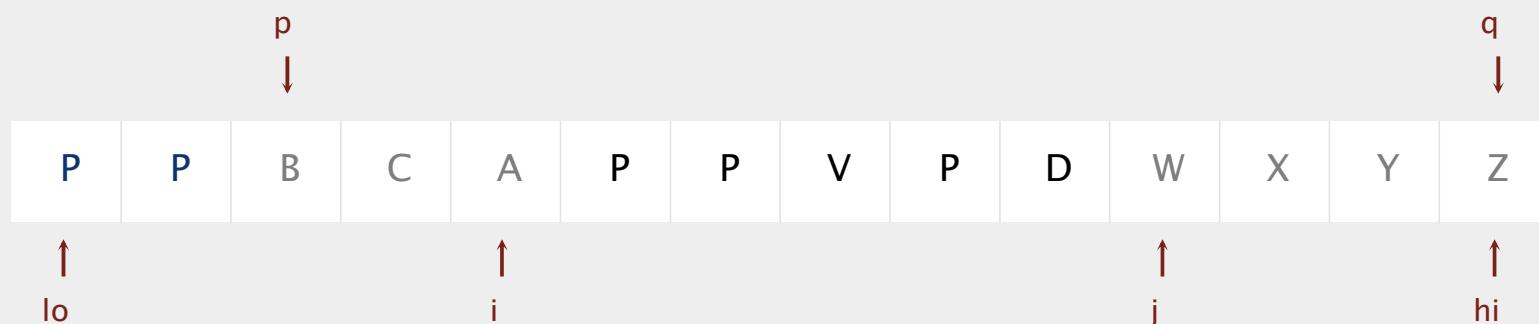


exchange $a[i]$ with $a[p]$ and increment p

Bentley-McIlroy 3-way partitioning

Repeat until i and j pointers cross.

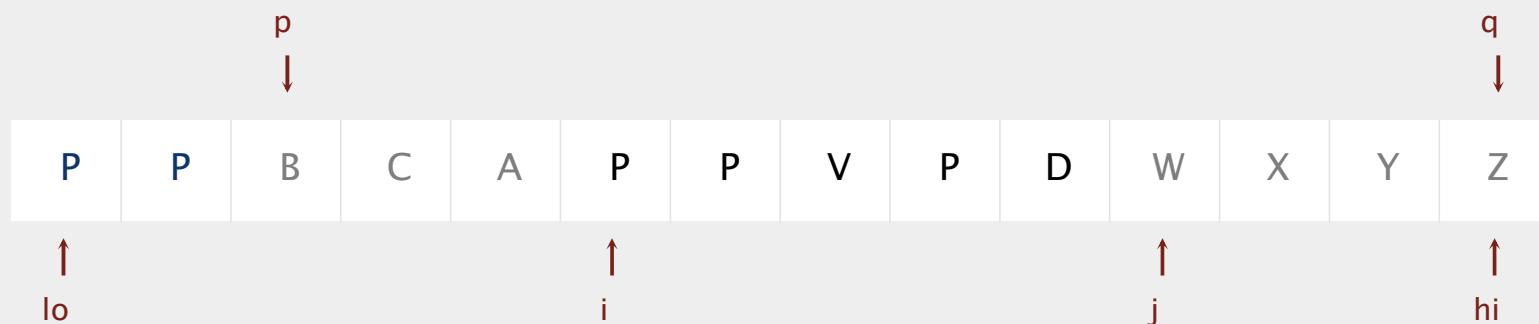
- Scan i from left to right so long as $a[i] < a[lo]$.
- Scan j from right to left so long as $a[j] > a[lo]$.
- Exchange $a[i]$ with $a[j]$.
- If $a[i] == a[lo]$, exchange $a[i]$ with $a[p]$ and increment p .
- If $a[j] == a[lo]$, exchange $a[j]$ with $a[q]$ and decrement q .



Bentley-McIlroy 3-way partitioning

Repeat until i and j pointers cross.

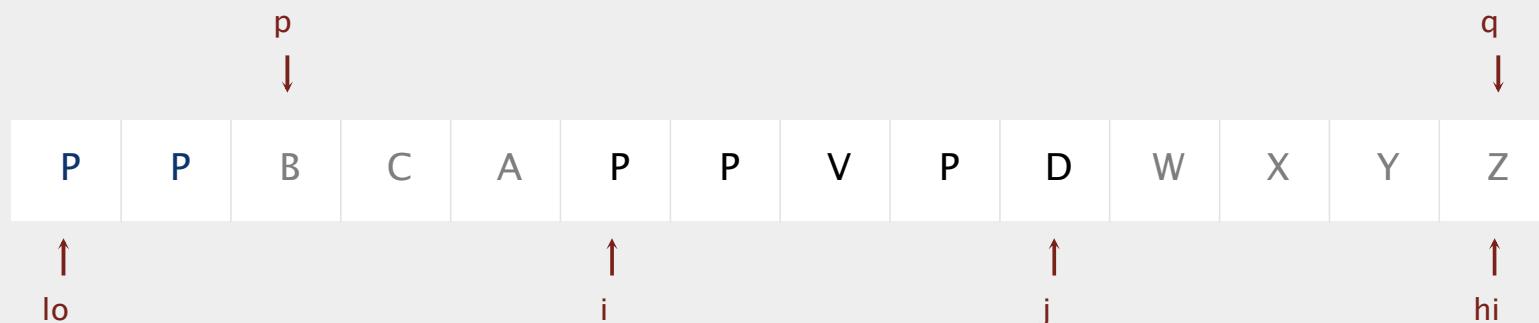
- Scan i from left to right so long as $a[i] < a[lo]$.
- Scan j from right to left so long as $a[j] > a[lo]$.
- Exchange $a[i]$ with $a[j]$.
- If $a[i] == a[lo]$, exchange $a[i]$ with $a[p]$ and increment p .
- If $a[j] == a[lo]$, exchange $a[j]$ with $a[q]$ and decrement q .



Bentley-McIlroy 3-way partitioning

Repeat until i and j pointers cross.

- Scan i from left to right so long as $a[i] < a[lo]$.
- Scan j from right to left so long as $a[j] > a[lo]$.
- Exchange $a[i]$ with $a[j]$.
- If $a[i] == a[lo]$, exchange $a[i]$ with $a[p]$ and increment p .
- If $a[j] == a[lo]$, exchange $a[j]$ with $a[q]$ and decrement q .

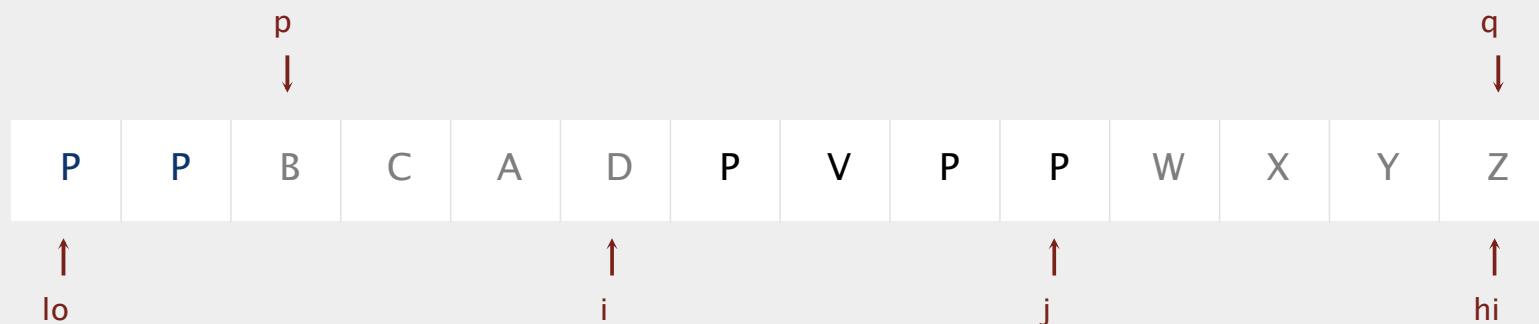


exchange $a[i]$ with $a[j]$

Bentley-McIlroy 3-way partitioning

Repeat until i and j pointers cross.

- Scan i from left to right so long as $a[i] < a[lo]$.
- Scan j from right to left so long as $a[j] > a[lo]$.
- Exchange $a[i]$ with $a[j]$.
- If $a[i] == a[lo]$, exchange $a[i]$ with $a[p]$ and increment p .
- If $a[j] == a[lo]$, exchange $a[j]$ with $a[q]$ and decrement q .

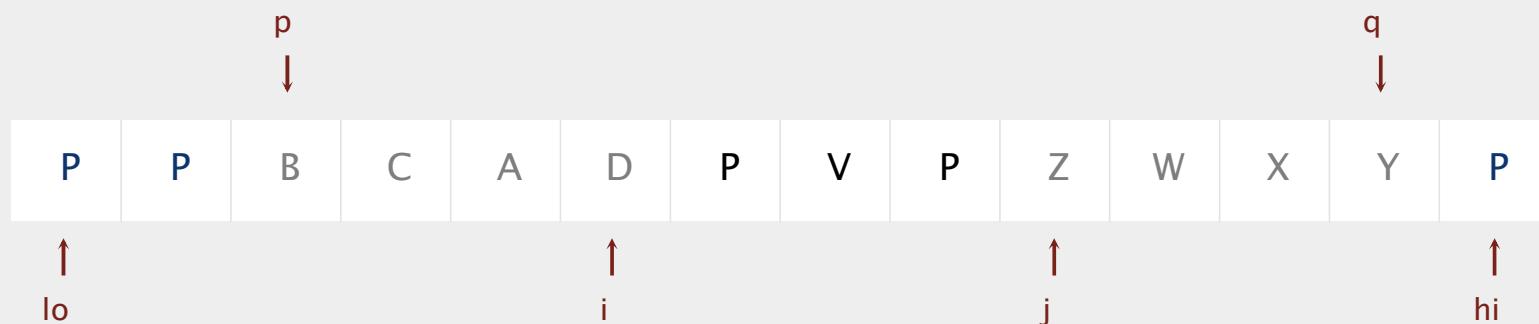


exchange $a[j]$ with $a[q]$ and decrement q

Bentley-McIlroy 3-way partitioning

Repeat until i and j pointers cross.

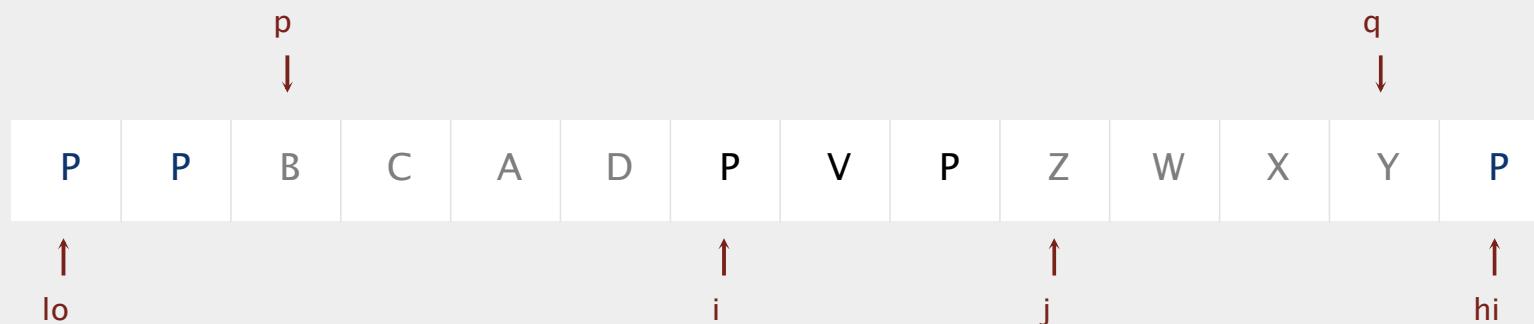
- Scan i from left to right so long as $a[i] < a[lo]$.
- Scan j from right to left so long as $a[j] > a[lo]$.
- Exchange $a[i]$ with $a[j]$.
- If $a[i] == a[lo]$, exchange $a[i]$ with $a[p]$ and increment p .
- If $a[j] == a[lo]$, exchange $a[j]$ with $a[q]$ and decrement q .



Bentley-McIlroy 3-way partitioning

Repeat until i and j pointers cross.

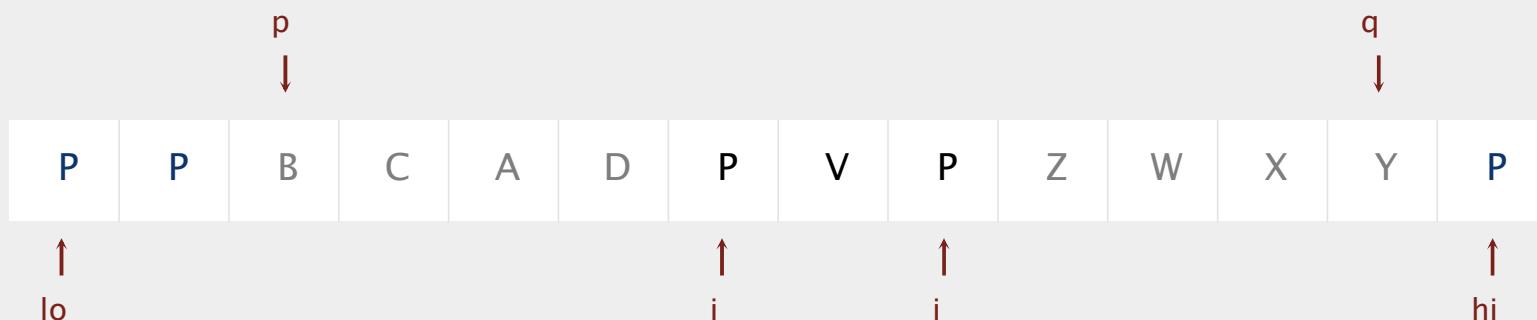
- Scan i from left to right so long as $a[i] < a[lo]$.
- Scan j from right to left so long as $a[j] > a[lo]$.
- Exchange $a[i]$ with $a[j]$.
- If $a[i] == a[lo]$, exchange $a[i]$ with $a[p]$ and increment p .
- If $a[j] == a[lo]$, exchange $a[j]$ with $a[q]$ and decrement q .



Bentley-McIlroy 3-way partitioning

Repeat until i and j pointers cross.

- Scan i from left to right so long as $a[i] < a[lo]$.
- Scan j from right to left so long as $a[j] > a[lo]$.
- Exchange $a[i]$ with $a[j]$.
- If $a[i] == a[lo]$, exchange $a[i]$ with $a[p]$ and increment p .
- If $a[j] == a[lo]$, exchange $a[j]$ with $a[q]$ and decrement q .

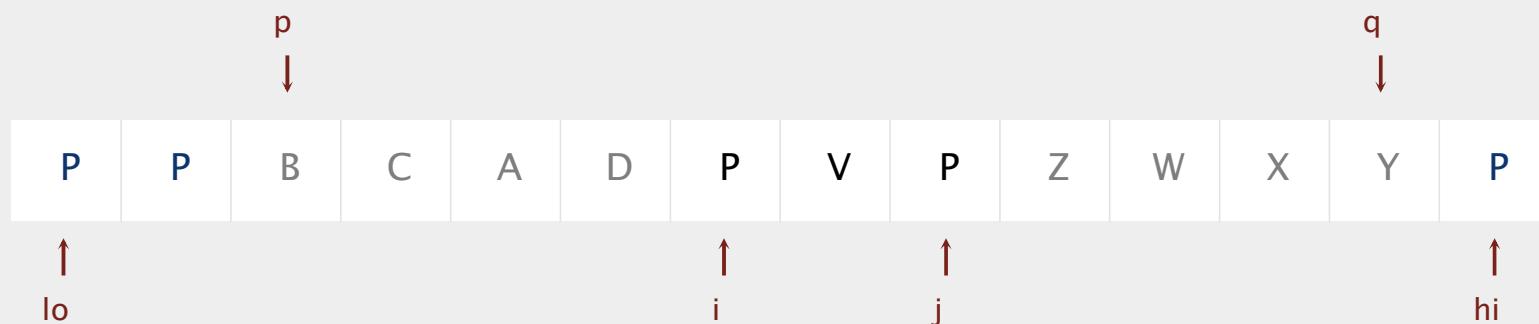


exchange $a[i]$ with $a[j]$

Bentley-McIlroy 3-way partitioning

Repeat until i and j pointers cross.

- Scan i from left to right so long as $a[i] < a[lo]$.
- Scan j from right to left so long as $a[j] > a[lo]$.
- Exchange $a[i]$ with $a[j]$.
- If $a[i] == a[lo]$, exchange $a[i]$ with $a[p]$ and increment p .
- If $a[j] == a[lo]$, exchange $a[j]$ with $a[q]$ and decrement q .

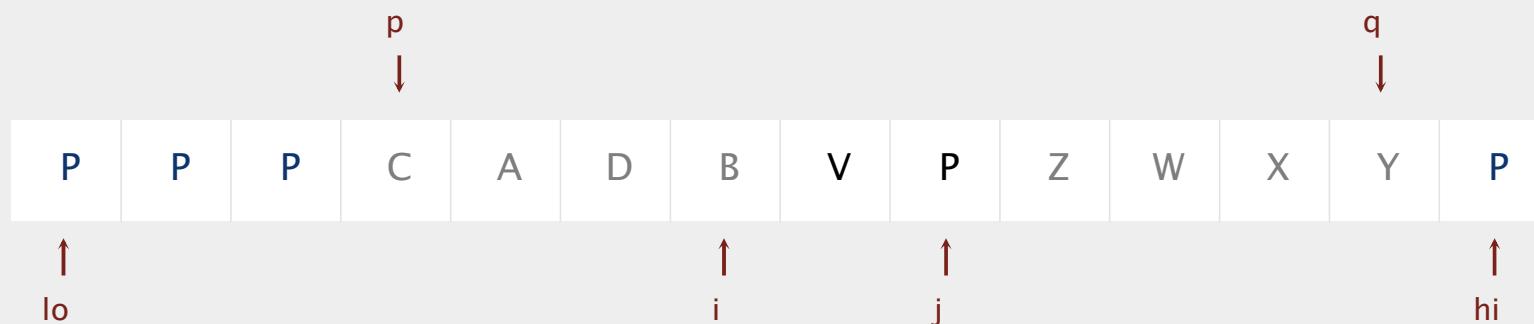


exchange $a[i]$ with $a[p]$ and increment p

Bentley-McIlroy 3-way partitioning

Repeat until i and j pointers cross.

- Scan i from left to right so long as $a[i] < a[lo]$.
- Scan j from right to left so long as $a[j] > a[lo]$.
- Exchange $a[i]$ with $a[j]$.
- If $a[i] == a[lo]$, exchange $a[i]$ with $a[p]$ and increment p .
- If $a[j] == a[lo]$, exchange $a[j]$ with $a[q]$ and decrement q .

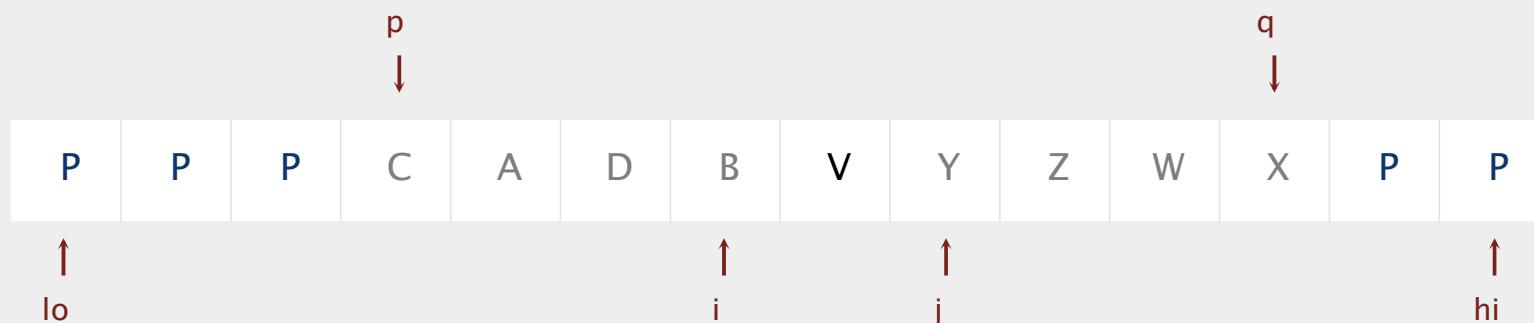


exchange $a[j]$ with $a[q]$ and decrement q

Bentley-McIlroy 3-way partitioning

Repeat until i and j pointers cross.

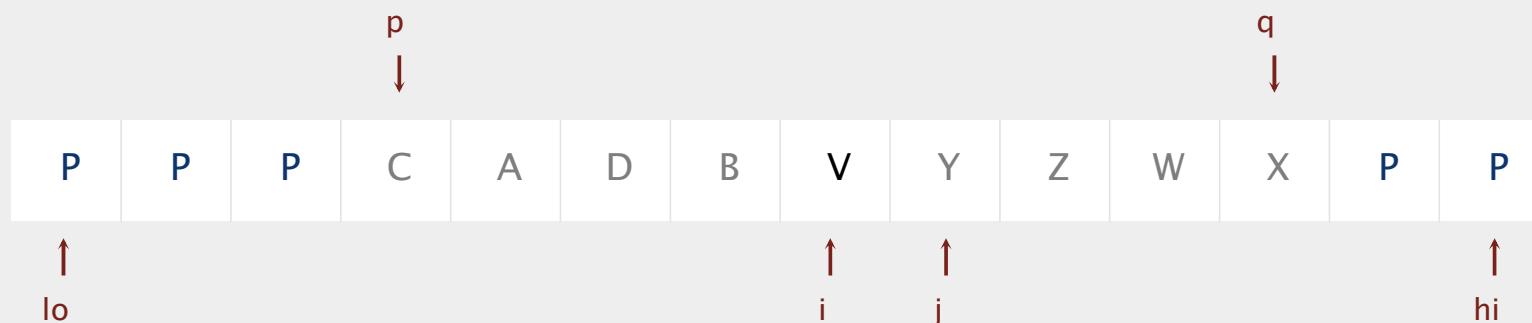
- Scan i from left to right so long as $a[i] < a[lo]$.
- Scan j from right to left so long as $a[j] > a[lo]$.
- Exchange $a[i]$ with $a[j]$.
- If $a[i] == a[lo]$, exchange $a[i]$ with $a[p]$ and increment p .
- If $a[j] == a[lo]$, exchange $a[j]$ with $a[q]$ and decrement q .



Bentley-McIlroy 3-way partitioning

Repeat until i and j pointers cross.

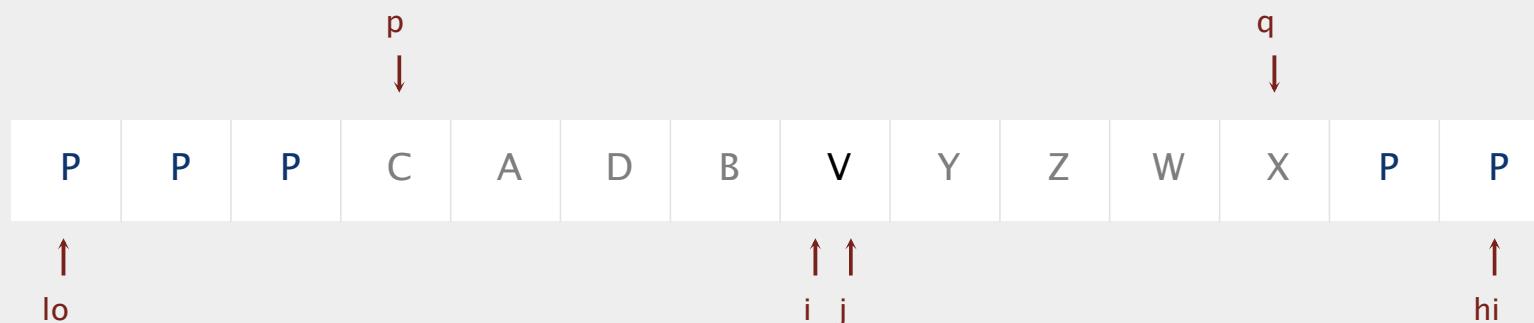
- Scan i from left to right so long as $a[i] < a[lo]$.
- Scan j from right to left so long as $a[j] > a[lo]$.
- Exchange $a[i]$ with $a[j]$.
- If $a[i] == a[lo]$, exchange $a[i]$ with $a[p]$ and increment p .
- If $a[j] == a[lo]$, exchange $a[j]$ with $a[q]$ and decrement q .



Bentley-McIlroy 3-way partitioning

Repeat until i and j pointers cross.

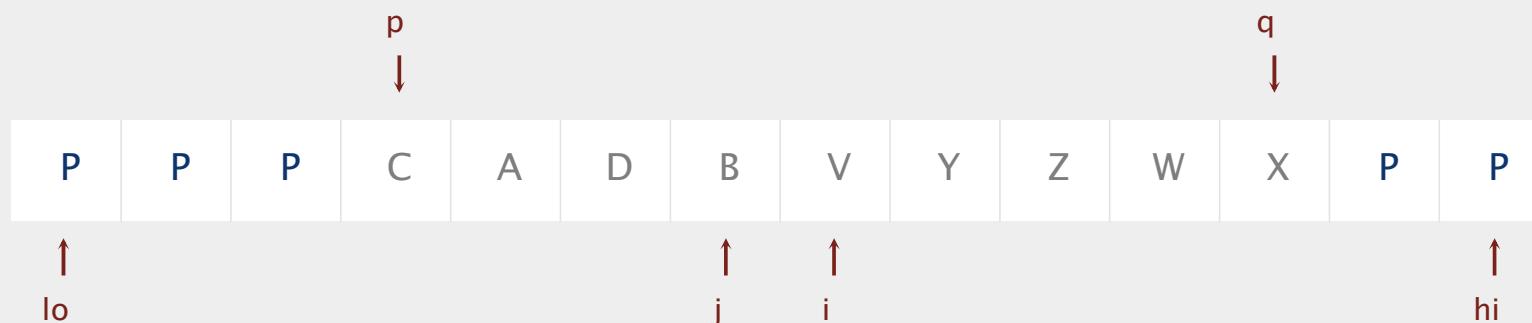
- Scan i from left to right so long as $a[i] < a[lo]$.
- Scan j from right to left so long as $a[j] > a[lo]$.
- Exchange $a[i]$ with $a[j]$.
- If $a[i] == a[lo]$, exchange $a[i]$ with $a[p]$ and increment p .
- If $a[j] == a[lo]$, exchange $a[j]$ with $a[q]$ and decrement q .



Bentley-McIlroy 3-way partitioning

Repeat until i and j pointers cross.

- Scan i from left to right so long as $a[i] < a[lo]$.
- Scan j from right to left so long as $a[j] > a[lo]$.
- Exchange $a[i]$ with $a[j]$.
- If $a[i] == a[lo]$, exchange $a[i]$ with $a[p]$ and increment p .
- If $a[j] == a[lo]$, exchange $a[j]$ with $a[q]$ and decrement q .

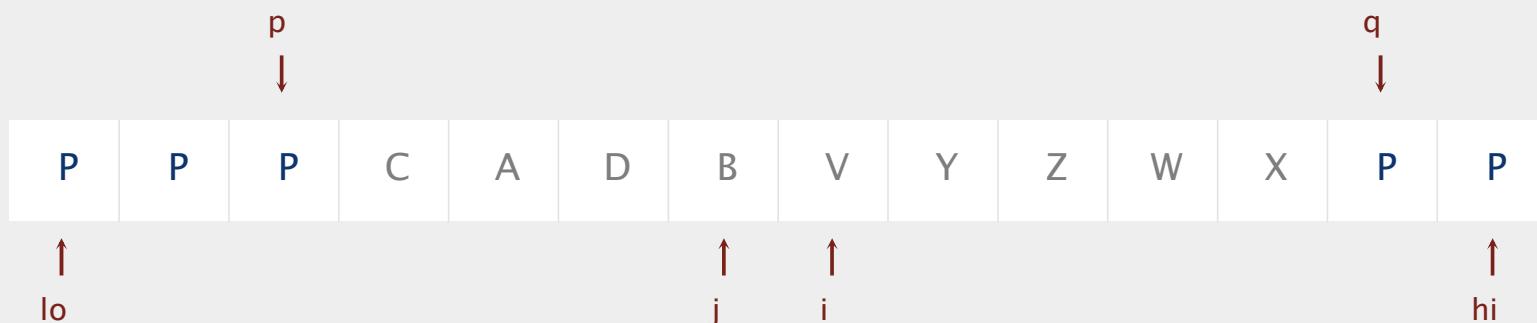


pointers cross

Bentley-McIlroy 3-way partitioning

Afterwards, swap equal keys to the center.

- Scan j and p from right to left and exchange $a[j]$ with $a[p]$.
- Scan i and q from left to right and exchange $a[i]$ with $a[q]$.

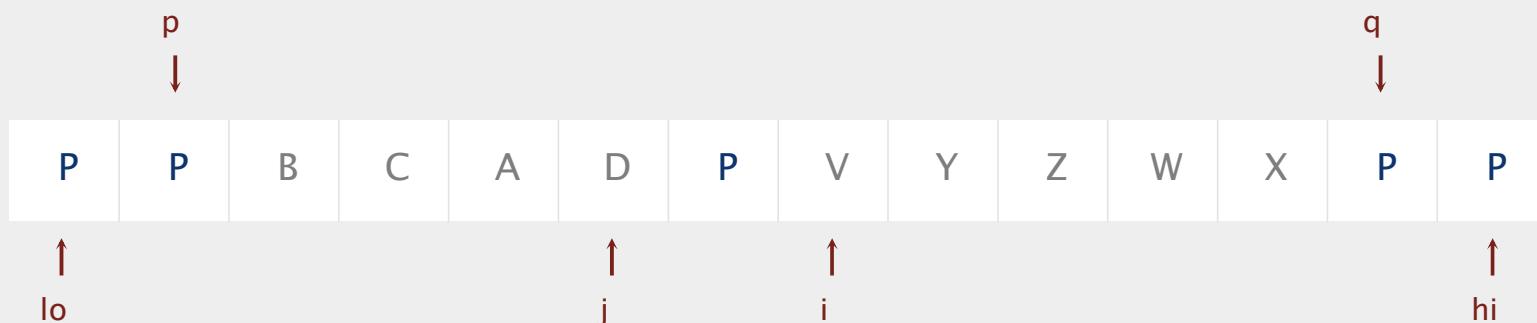


exchange $a[j]$ with $a[p]$

Bentley-McIlroy 3-way partitioning

Afterwards, swap equal keys to the center.

- Scan j and p from right to left and exchange $a[j]$ with $a[p]$.
- Scan i and q from left to right and exchange $a[i]$ with $a[q]$.

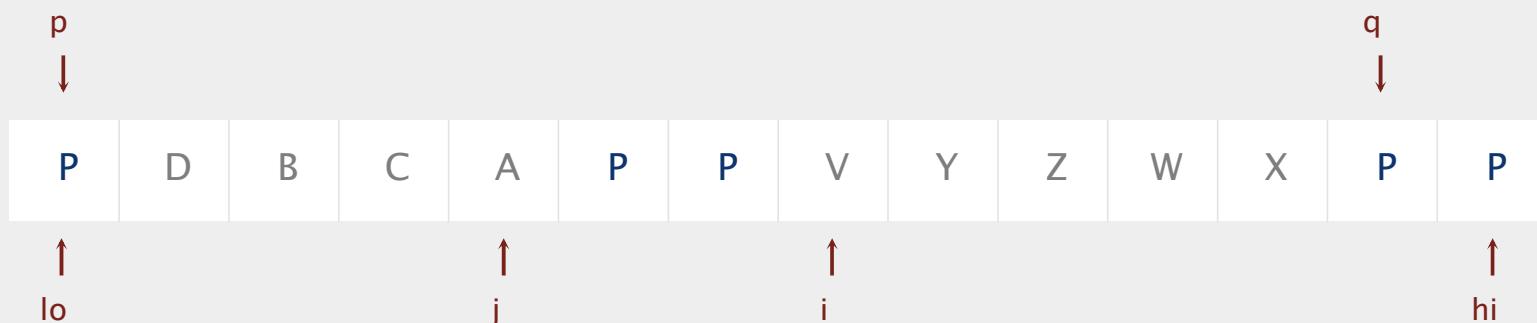


exchange $a[j]$ with $a[p]$

Bentley-McIlroy 3-way partitioning

Afterwards, swap equal keys to the center.

- Scan j and p from right to left and exchange $a[j]$ with $a[p]$.
- Scan i and q from left to right and exchange $a[i]$ with $a[q]$.

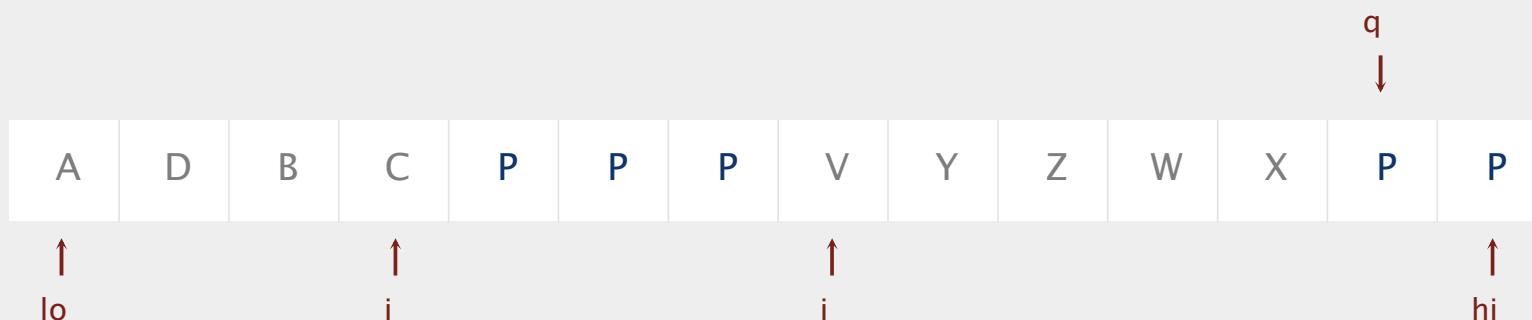


exchange $a[j]$ with $a[p]$

Bentley-McIlroy 3-way partitioning

Afterwards, swap equal keys to the center.

- Scan j and p from right to left and exchange $a[j]$ with $a[p]$.
- Scan i and q from left to right and exchange $a[i]$ with $a[q]$.

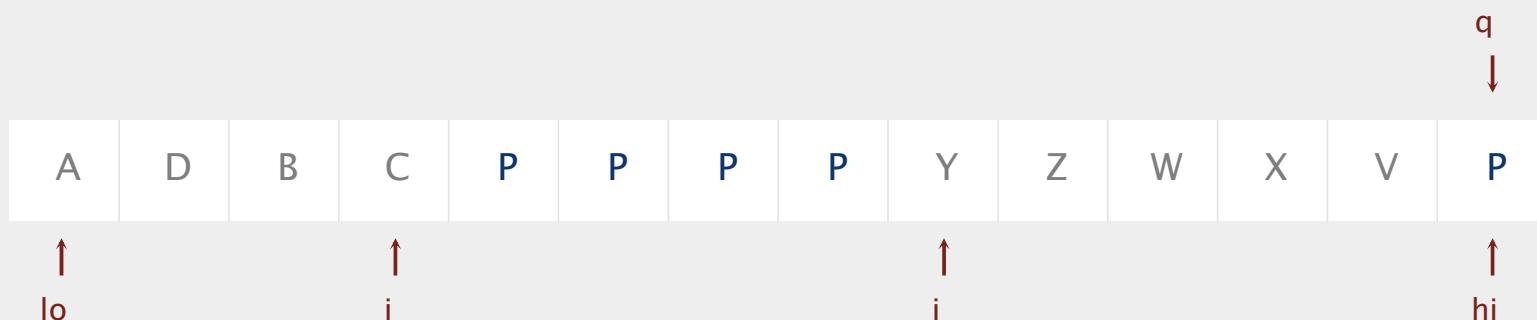


exchange $a[i]$ with $a[q]$

Bentley-McIlroy 3-way partitioning

Afterwards, swap equal keys to the center.

- Scan j and p from right to left and exchange $a[j]$ with $a[p]$.
- Scan i and q from left to right and exchange $a[i]$ with $a[q]$.

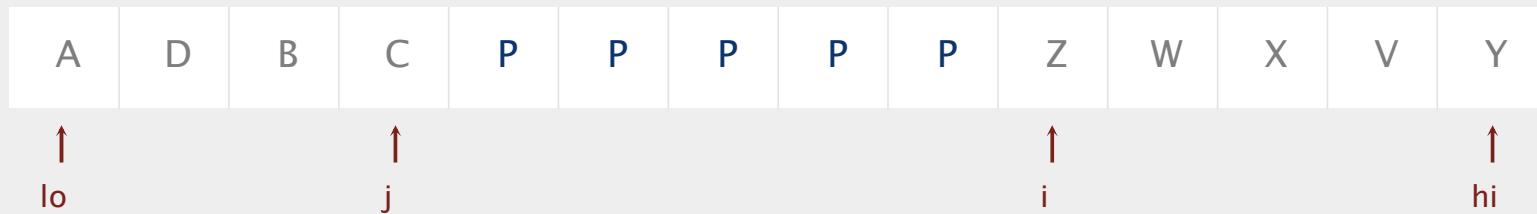


exchange $a[i]$ with $a[q]$

Bentley-McIlroy 3-way partitioning

Afterwards, swap equal keys to the center.

- Scan j and p from right to left and exchange $a[j]$ with $a[p]$.
- Scan i and q from left to right and exchange $a[i]$ with $a[q]$.



3-way partitioned

Lecture 5 Sort(III)

- Priority queue and Heapsort
- Sort in linear time

ACKNOWLEDGEMENTS: Some contents in this lecture source from COS226 of Princeton University by [Kevin Wayne](#) and [Bob Sedgewick](#)

Priority queues

A *priority queue* is a data structure for maintaining a set S of elements, each with an associated value called a *key*.

A *max-priority queue* supports the following operations:

insert(S , x)

max(S)

delete-max(S)

decrease-key(x , key)

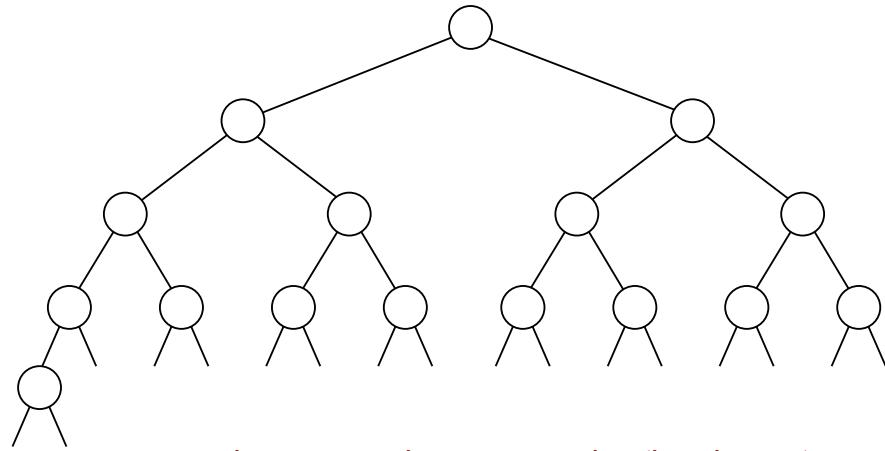
Array and sorted array

	Maximum	Insert	DeleteMax
Array	$O(n)$	$O(1)$	$O(n)$
Sorted array	$O(1)$	$O(n)$	$O(1)$

Data structure matters.

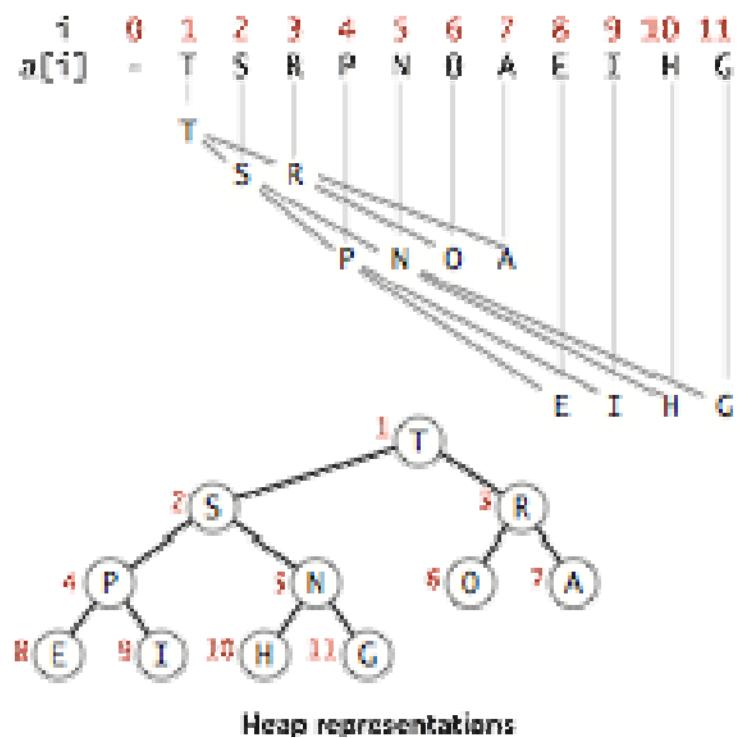
Binary tree

Almost complete binary tree



Property. Height of an almost complete tree with N nodes is $\lceil \log N \rceil$.

Array representation



Binary tree



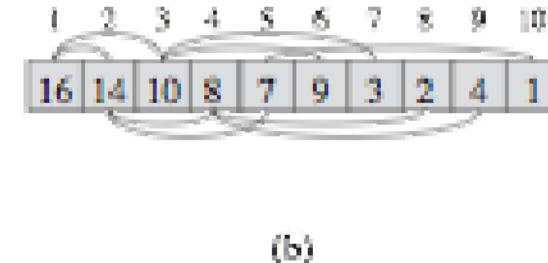
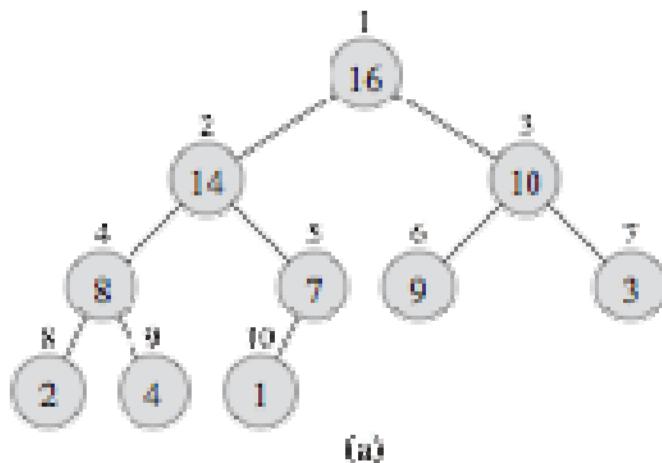
Hyphaene Compressa - Doum Palm

© Shlomit Pinter

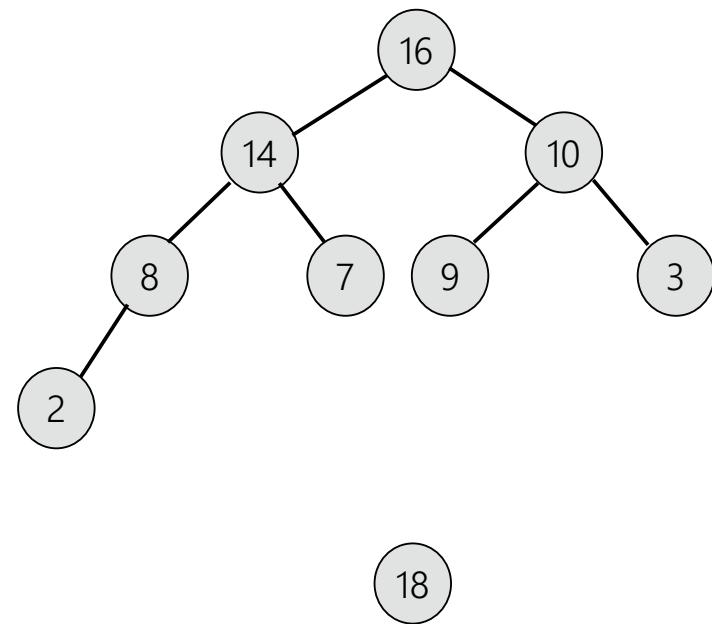
Binary Heap

A *max-heap (min-heap)* is an **almost complete binary tree** with vertices satisfying the max-heap (min-heap) property:

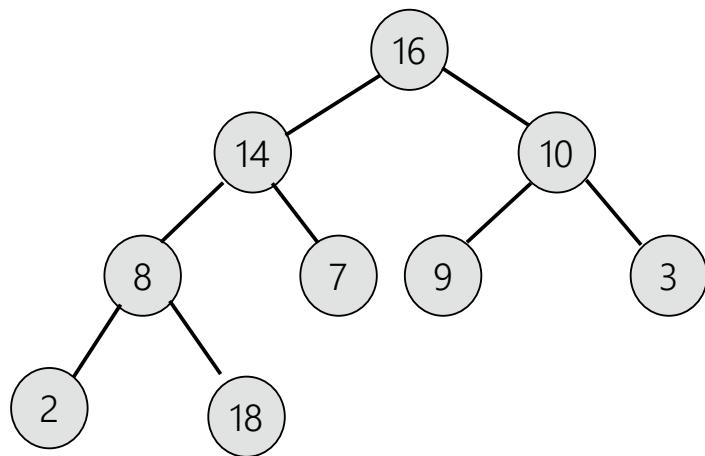
If v and $p(v)$ are a vertex and its parent respectively, then $p(v) \geq v$ ($p(v) \leq v$).



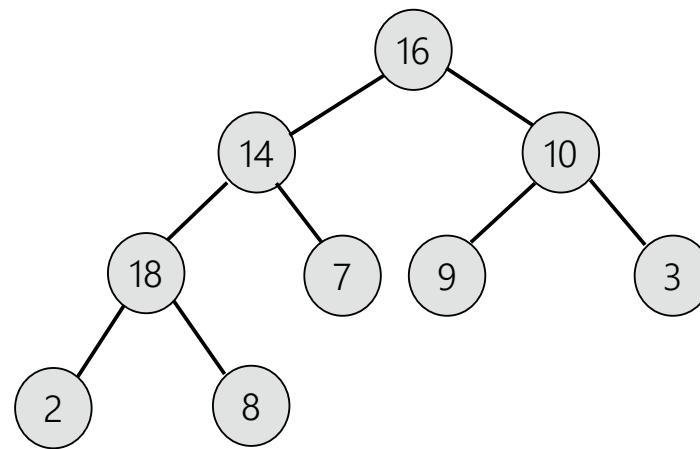
Insert: SiftUp



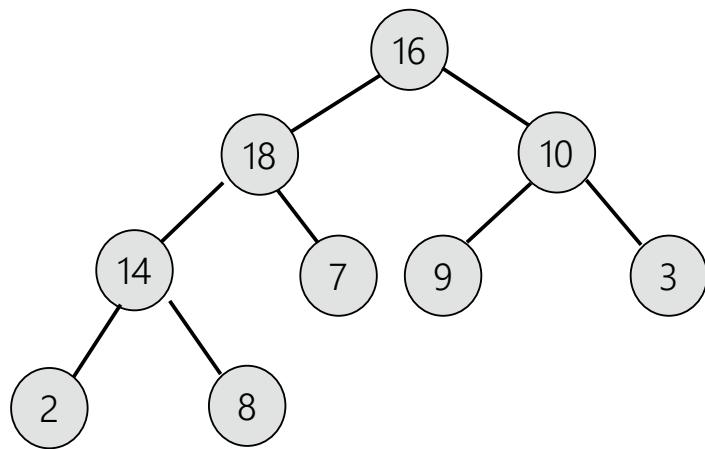
Insert: SiftUp



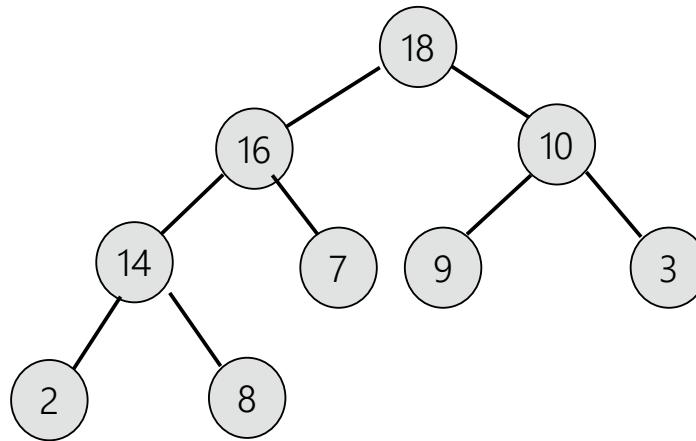
Insert: SiftUp



Insert: SiftUp

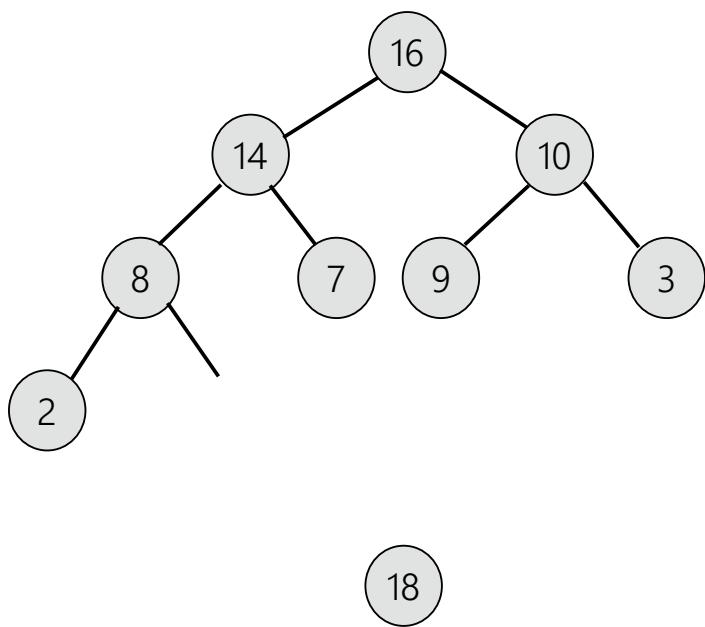


Insert: SiftUp



$O(\log n)$

Insert: SiftUp



Algorithm 3.3 Insert

Input: A heap $H[1..n]$ and a heap element x .

Output: A new heap $H[1..n+1]$ with x being one of its elements.

1. $n \leftarrow n + 1$
2. $H[n] \leftarrow x$
3. $SiftUp(H, n)$

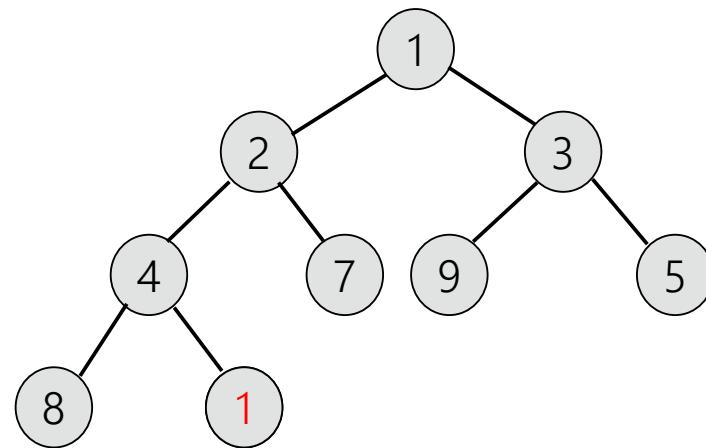
$SiftUp(H, i)$

1. $done \leftarrow \text{false}$
2. if $i = 1$ then exit
3. repeat
4. if $\text{key}(H[i]) > \text{key}(H[\lfloor i/2 \rfloor])$
5. then interchange $H[i]$ and $H[\lfloor i/2 \rfloor]$
6. else $done \leftarrow \text{true}$
7. $i \leftarrow \lfloor i/2 \rfloor$
8. until $i = 1$ or $done$



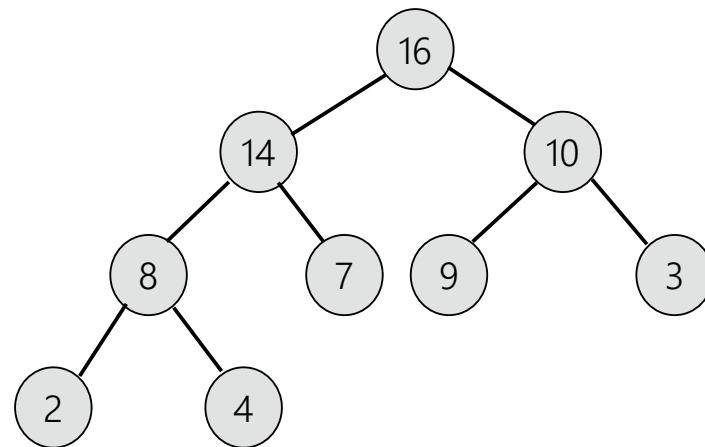
$O(\log n)$

Decrease-key: SiftUp



O(log n)

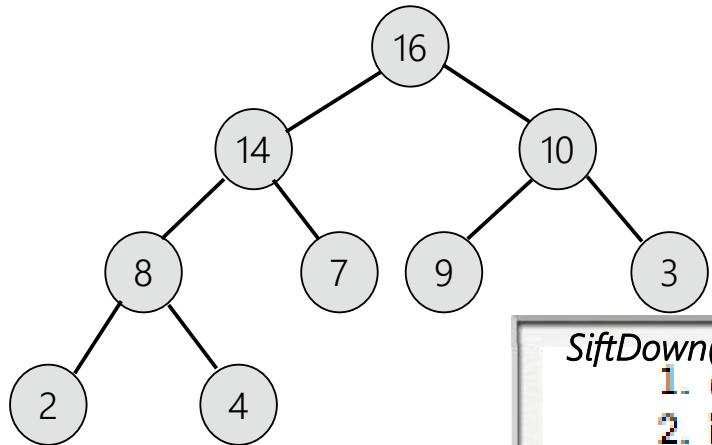
Maximum



return H[1];



Delete-maximum: SiftDown



O(2logn)

SiftDown(H, i)

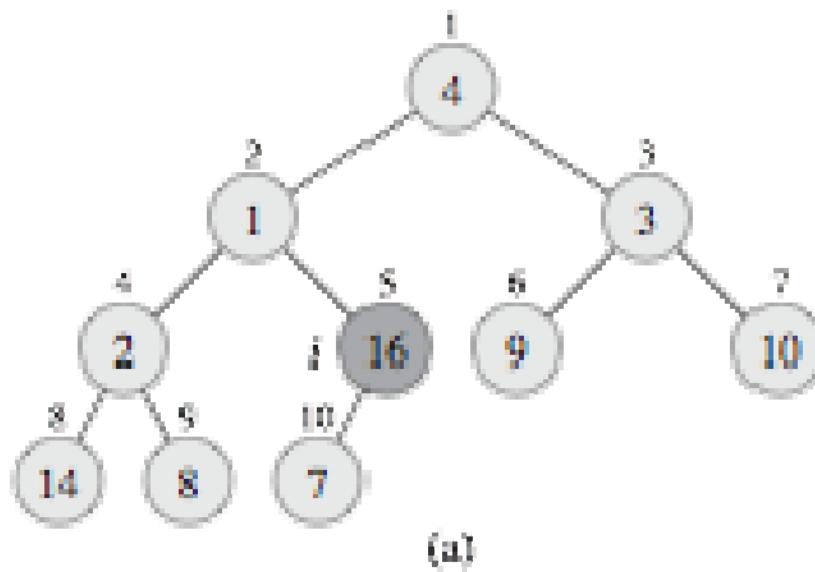
1. $\text{done} \leftarrow \text{false}$
2. if $2i > n$ then exit
3. repeat
4. $i \leftarrow 2i$
5. if $i + 1 \leq n$ and $\text{key}(H[i + 1]) > \text{key}(H[i])$
6. then $i \leftarrow i + 1$
7. if $\text{key}(H[\lfloor i/2 \rfloor]) < \text{key}(H[i])$
8. then interchange $H[i]$ and $H[\lfloor i/2 \rfloor]$
9. else $\text{done} \leftarrow \text{true}$
10. until $2i > n$ or done

Heap and Array

	Maximum	Insert	DeleteMax
Array	$O(n)$	$O(1)$	$O(n)$
Sorted array	$O(1)$	$O(n)$	$O(1)$
Heap	$O(1)$	$O(\log n)$	$O(\log n)$

How to make a heap

A [4 | 1 | 3 | 2 | 16 | 9 | 10 | 14 | 8 | 7]



Heap sort

- Create max-heap with all N keys.
- Repeatedly remove the maximum key.

Question: How to make a heap?



How to make a heap

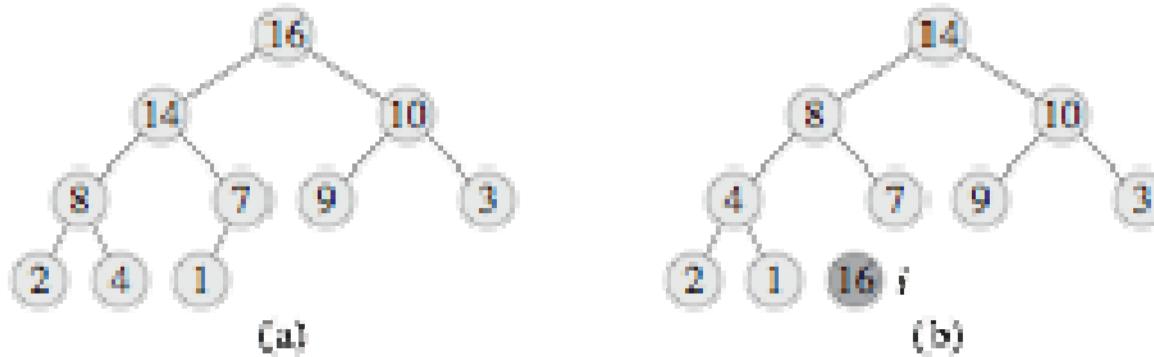
Algorithm 3.6 MakeHeap

Input: An array $A[1..n]$ of n elements.

Output: $A[1..n]$ is transformed into a heap.

1. for $i \leftarrow \lfloor n/2 \rfloor$ downto 1
2. $SiftDown(A, i)$
3. end for

Heap sort



Algorithm 3.7 Heapsort

Input: An array $A[1..n]$ of n elements.

Output: Array $A[1..n]$ sorted in nondecreasing order.

1. $\text{MakeHeap}(A)$
2. $\text{for } j \leftarrow n \text{ downto } 2$
3. $\text{interchange } A[1] \text{ and } A[j]$
4. $\text{SiftDown}(A[1..j - 1], 1)$
5. end for

Quiz

Give the array that results after heap construction on the following array of 10 keys:

X U M V Y T A J I N

Give the array that results after *performing 3 successive delete-the-max* on above constructed max heap.

- A. Y X T V M U A N I J
- B. Y X T V U M A J I N
- C. U N T J I M A
- D. U N T I J M A

Quiz

What's the time complexity of making a heap?
What's the time complexity of heap sort?

- A. $\Theta(n)$
- B. $\Theta(n \log n)$
- C. $\Theta(n^2)$
- D. none of above

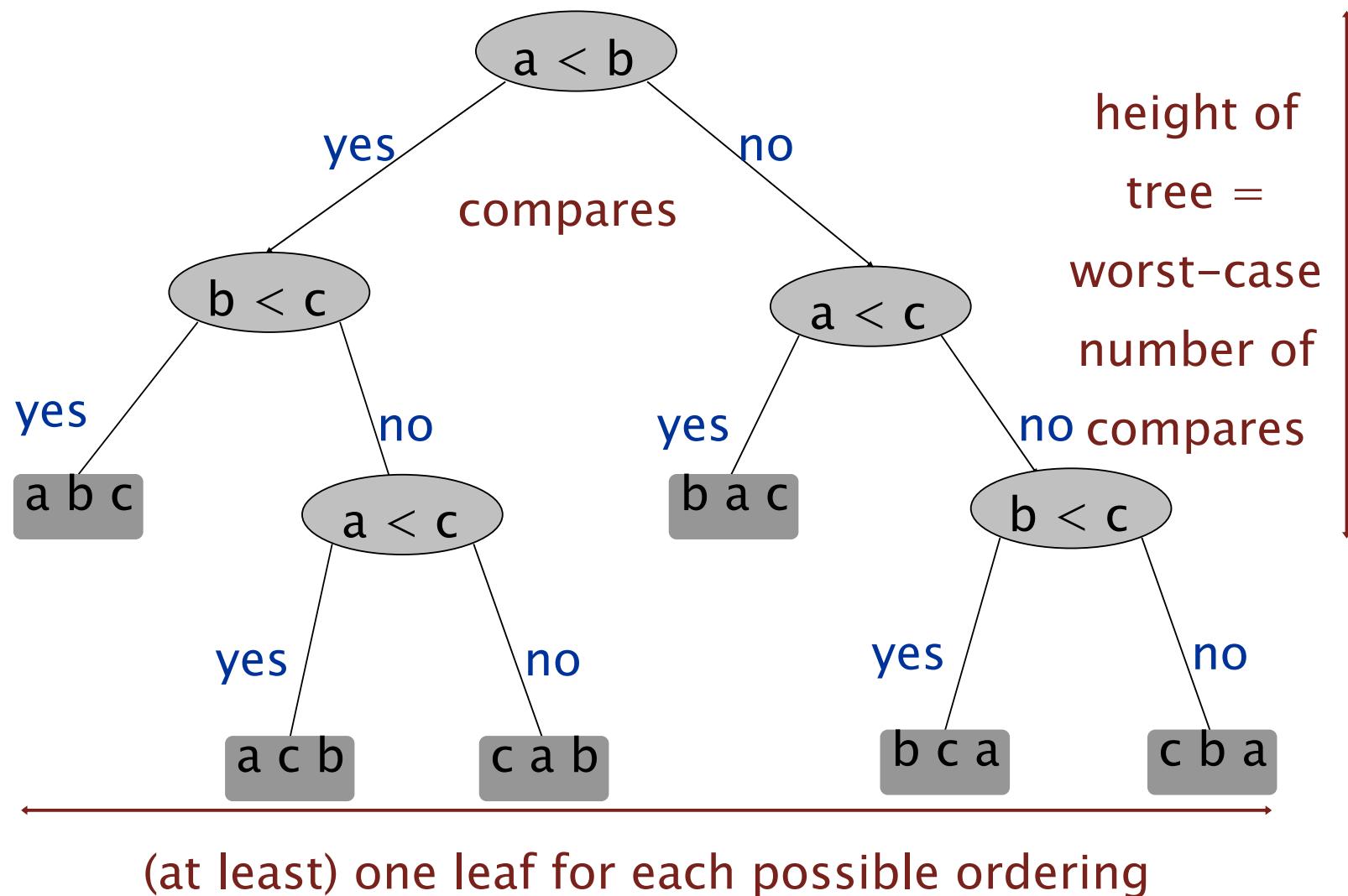
Applications

- Event-driven simulation. [customers in a line]
- Data compression. [Huffman codes]
- Graph searching.[Dijkstra's algorithm, Prim's algorithm]
- Statistics. [maintain largest M values in a sequence]
- Operating systems.[load balancing, interrupt handling]
- Discrete optimization. [bin packing, scheduling]
-

Sort summary

	inplace?	stable?	worst	average	best	remarks
selection	x		$N^2 / 2$	$N^2 / 2$	$N^2 / 2$	N exchanges
insertion	x	x	$N^2 / 2$	$N^2 / 4$	N	use for small N or partially ordered
shell	x		?	?	N	tight code, subquadratic
quick	x		$N^2 / 2$	$2N \ln N$	$N \lg N$	$N \log N$ probabilistic guarantee fastest in practice
3-way quick	x		$N^2 / 2$	$2N \ln N$	N	improves quicksort in presence of duplicate keys
merge		x	$N \lg N$	$N \lg N$	$N \lg N$	$N \log N$ guarantee, stable
heap	x		$2N \lg N$	$2N \lg N$	$N \lg N$	$N \log N$ guarantee, in-place
???	x	x	$N \lg N$	$N \lg N$	$N \lg N$	holy sorting grail

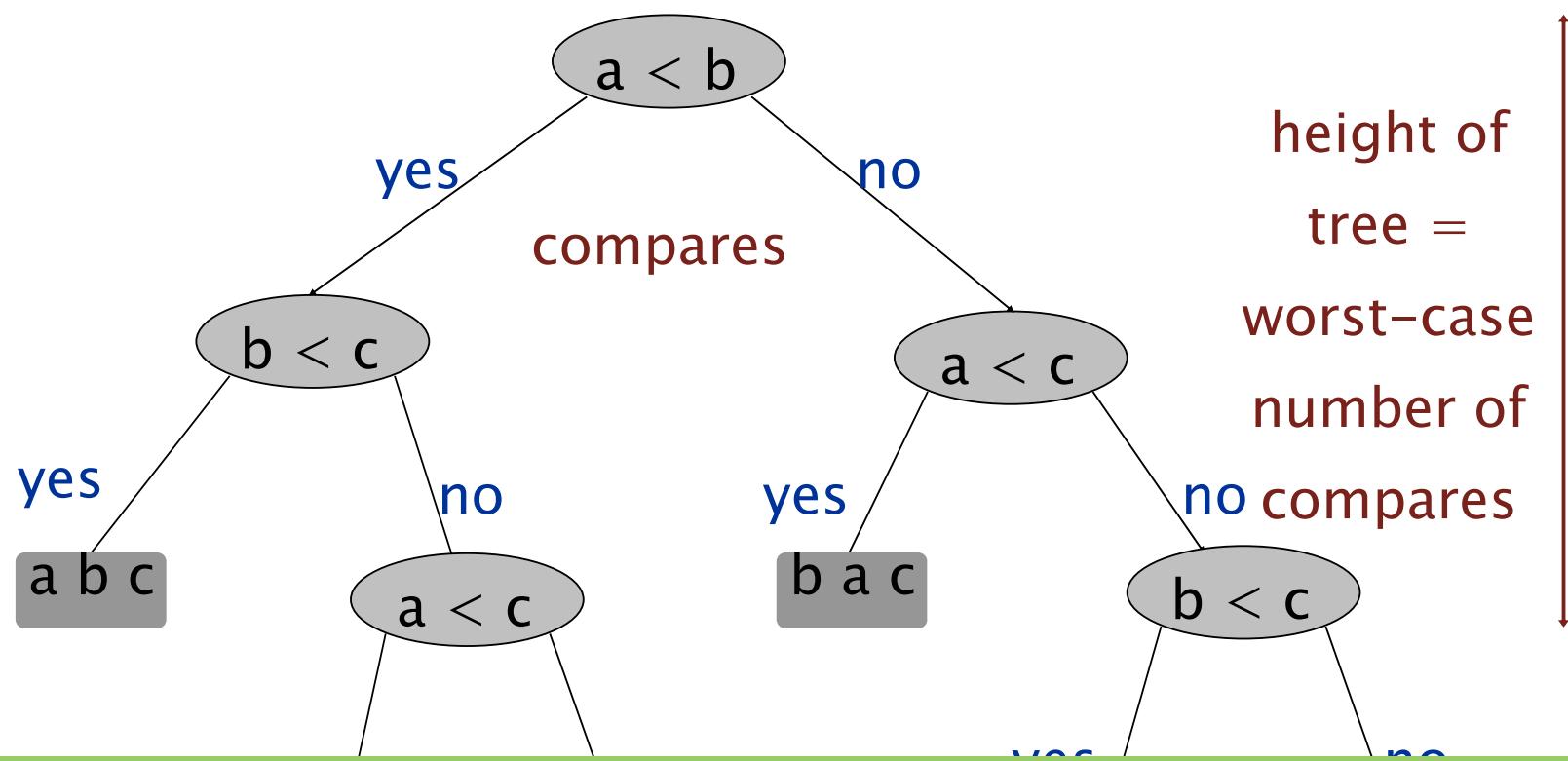
Complexity of Sort



height of
tree =
worst-case
number of
no compares

(at least) one leaf for each possible ordering

Complexity of Sort



Proposition. Any compare-based sorting algorithm must use at least $\log(n!)$ ~ $n \log n$ compares in the worst-case.

(at least) one leaf for each possible ordering

Sort summary

	inplace?	stable?	worst	average	best	remarks
selection	x		$N^2 / 2$	$N^2 / 2$	$N^2 / 2$	N exchanges
insertion	x	x	$N^2 / 2$	$N^2 / 4$	N	use for small N or partially ordered
shell	x		?	?	N	tight code, subquadratic
quick	x		$N^2 / 2$	$2N \ln N$	$N \lg N$	$N \log N$ probabilistic guarantee fastest in practice
3-way quick	x		$N^2 / 2$	$2N \ln N$	N	improves quicksort in presence of duplicate keys
merge		x	$N \lg N$	$N \lg N$	$N \lg N$	$N \log N$ guarantee, stable
heap	x		$2N \lg N$	$2N \lg N$	$N \lg N$	$N \log N$ guarantee, in-place
radix		x	kN	kN	kN	for numbers with k digits, or for string
???	x	x	$N \lg N$	$N \lg N$	$N \lg N$	holy sorting grail

Which sort algorithm?

lifo	find	data	data	data	data
fifo	fifo	fifo	fifo	exch	exch
data	data	find	find	fifo	fifo
type	exch	hash	hash	find	find
hash	hash	heap	heap	hash	hash
heap	heap	lifo	lifo	heap	heap
sort	less	link	link	leaf	leaf
link	left	list	list	left	left
list	leaf	push	push	less	less
push	lifo	root	root	lifo	lifo
find	push	sort	sort	link	link
root	root	type	type	list	list
leaf	list	leaf	leaf	sort	next
tree	tree	left	tree	tree	node
null	null	node	null	null	null
path	path	null	path	path	path
node	node	path	node	node	push
left	link	tree	left	type	root
less	sort	exch	less	root	sink
exch	type	less	exch	push	sort
sink	sink	next	sink	sink	swap
swim	swim	sink	swim	swim	swim
next	next	swap	next	next	tree
swap	swap	swim	swap	swap	type
original	?	?	?	?	sorted

Which sort algorithm?

lifo	find	data	data	data	data
fifo	fifo	fifo	fifo	exch	exch
data	data	find	find	fifo	fifo
type	exch	hash	hash	find	find
hash	hash	heap	heap	hash	hash
heap	heap	fifo	fifo	heap	heap
sort	less	link	link	leaf	leaf
link	left	list	list	left	left
list	leaf	push	push	less	less
push	lifo	root	root	lifo	lifo
find	push	sort	sort	link	link
root	root	type	type	list	list
leaf	list	leaf	leaf	sort	next
tree	tree	left	tree	tree	node
null	null	node	null	null	null
path	path	null	path	path	path
node	node	path	node	node	push
left	link	tree	left	type	root
less	sort	exch	less	root	sink
exch	type	less	exch	push	sort
sink	sink	next	sink	sink	swap
swim	swim	sink	swim	swim	swim
next	next	swap	next	next	tree
swap	swap	swim	swap	swap	type
original	quicksort	mergesort	insertion	selection	sorted

Heapsort Demo

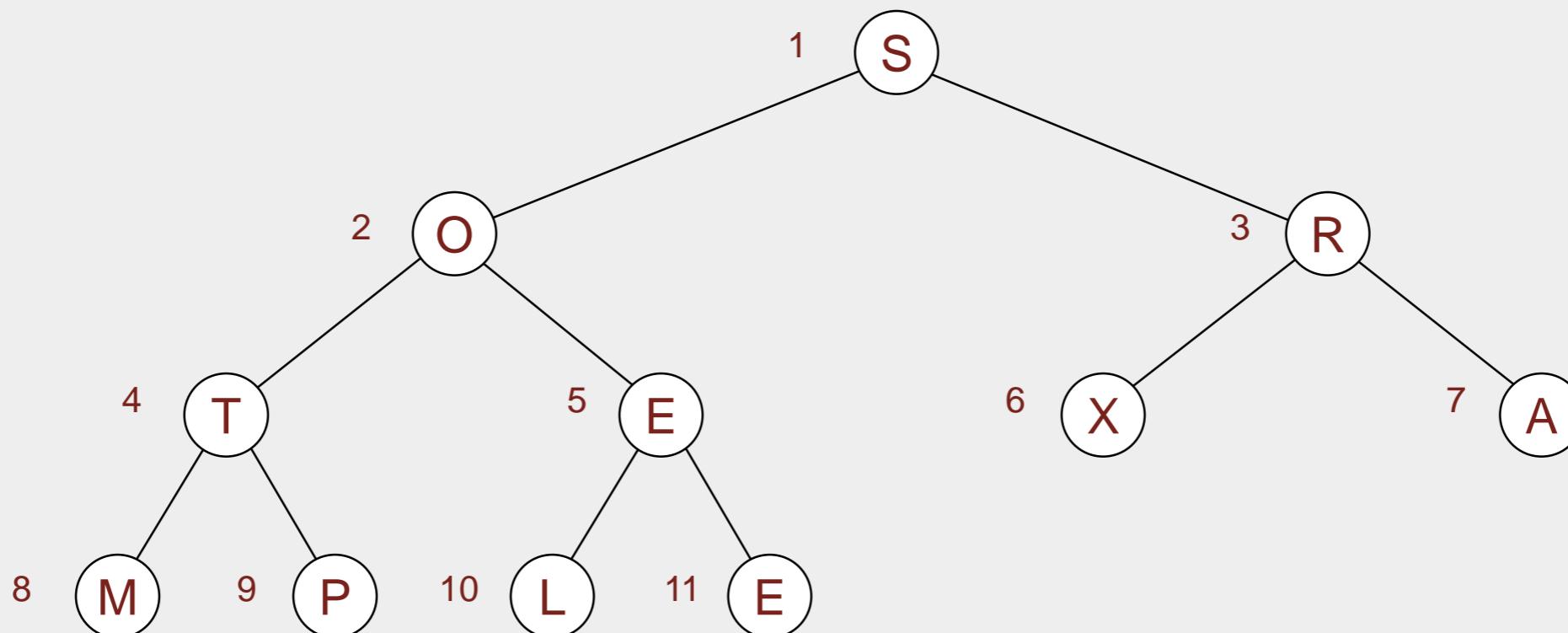


click to begin demo

Starting point. Array in arbitrary order.



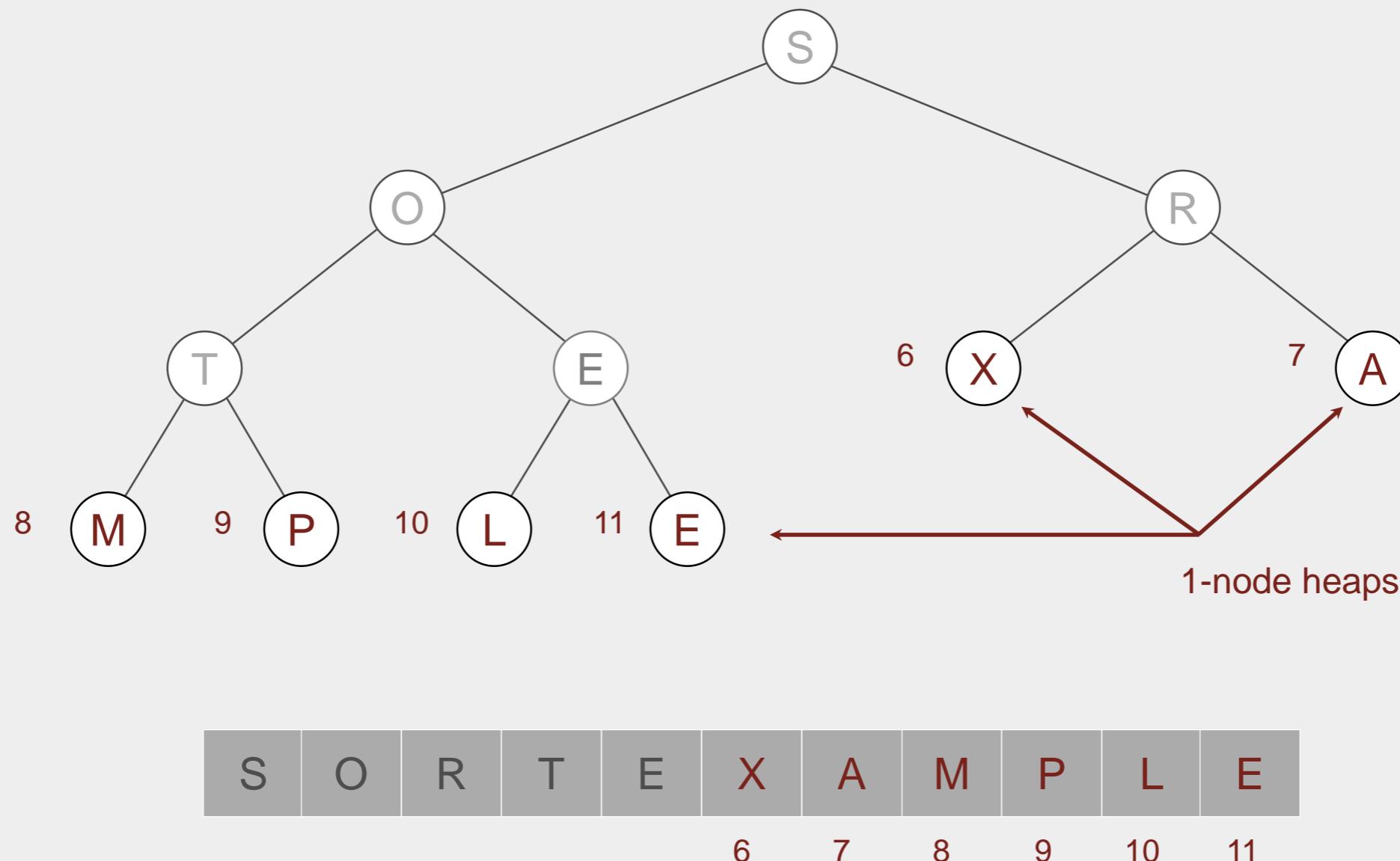
we assume array entries are indexed 1 to N



S	O	R	T	E	X	A	M	P	L	E
1	2	3	4	5	6	7	8	9	10	11

Heapsort

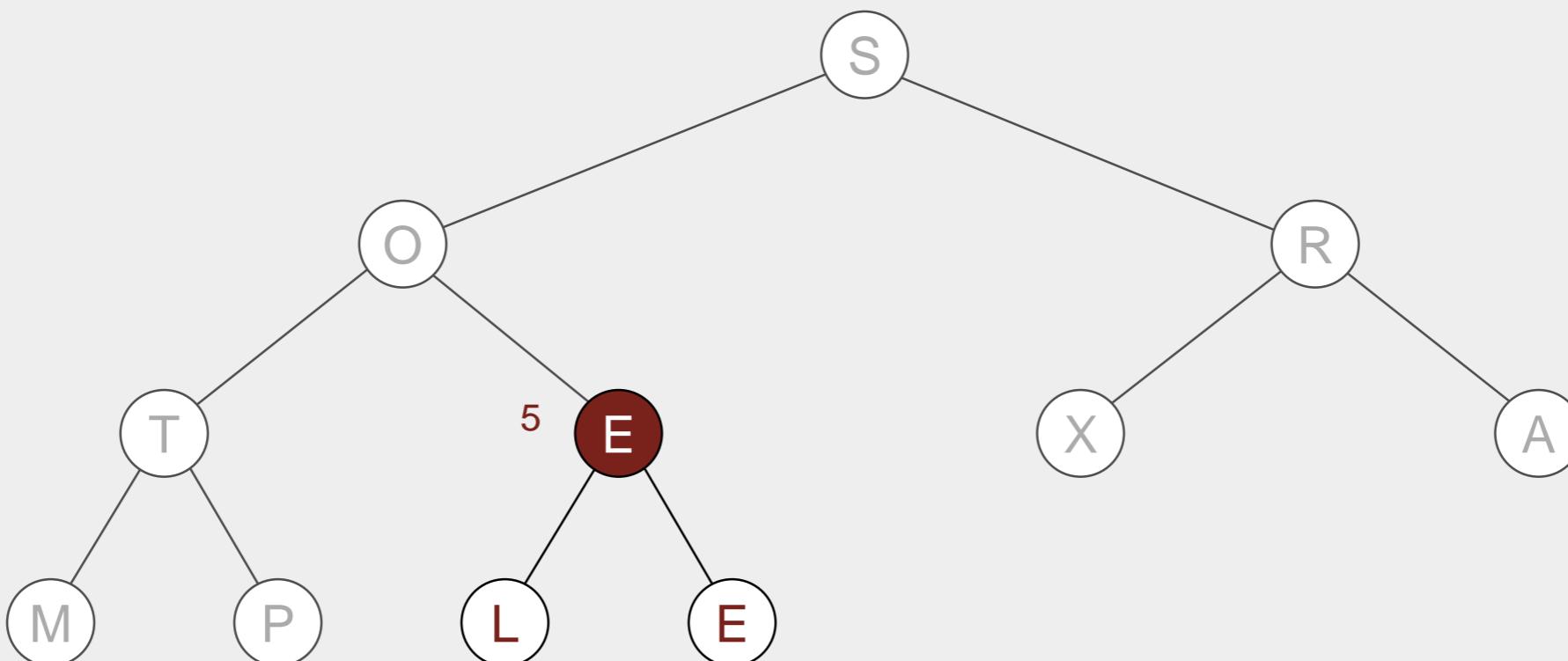
Heap construction. Build max heap using bottom-up method.



Heapsort

Heap construction. Build max heap using bottom-up method.

sink 5

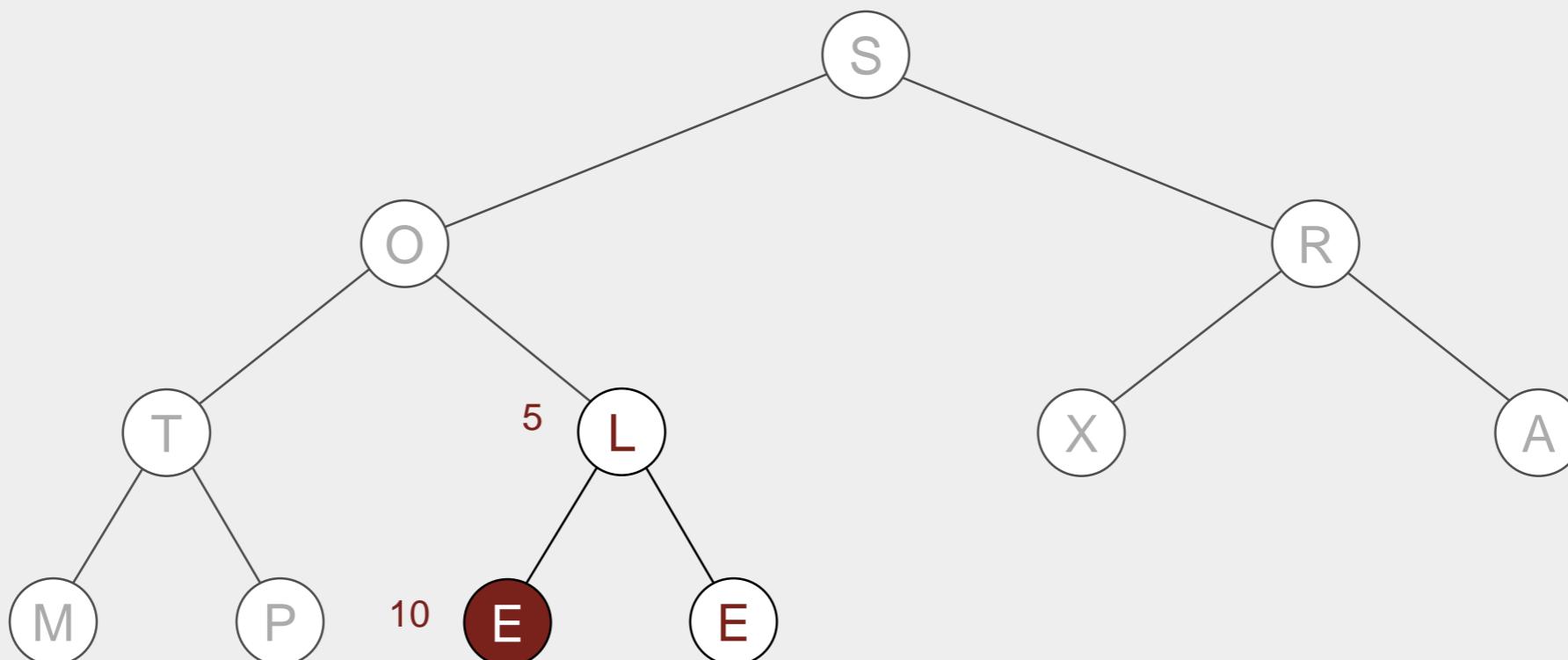


S	O	R	T	E	X	A	M	P	L	E
---	---	---	---	---	---	---	---	---	---	---

Heapsort

Heap construction. Build max heap using bottom-up method.

sink 5



S	O	R	T	L	X	A	M	P	E	E
---	---	---	---	---	---	---	---	---	---	---

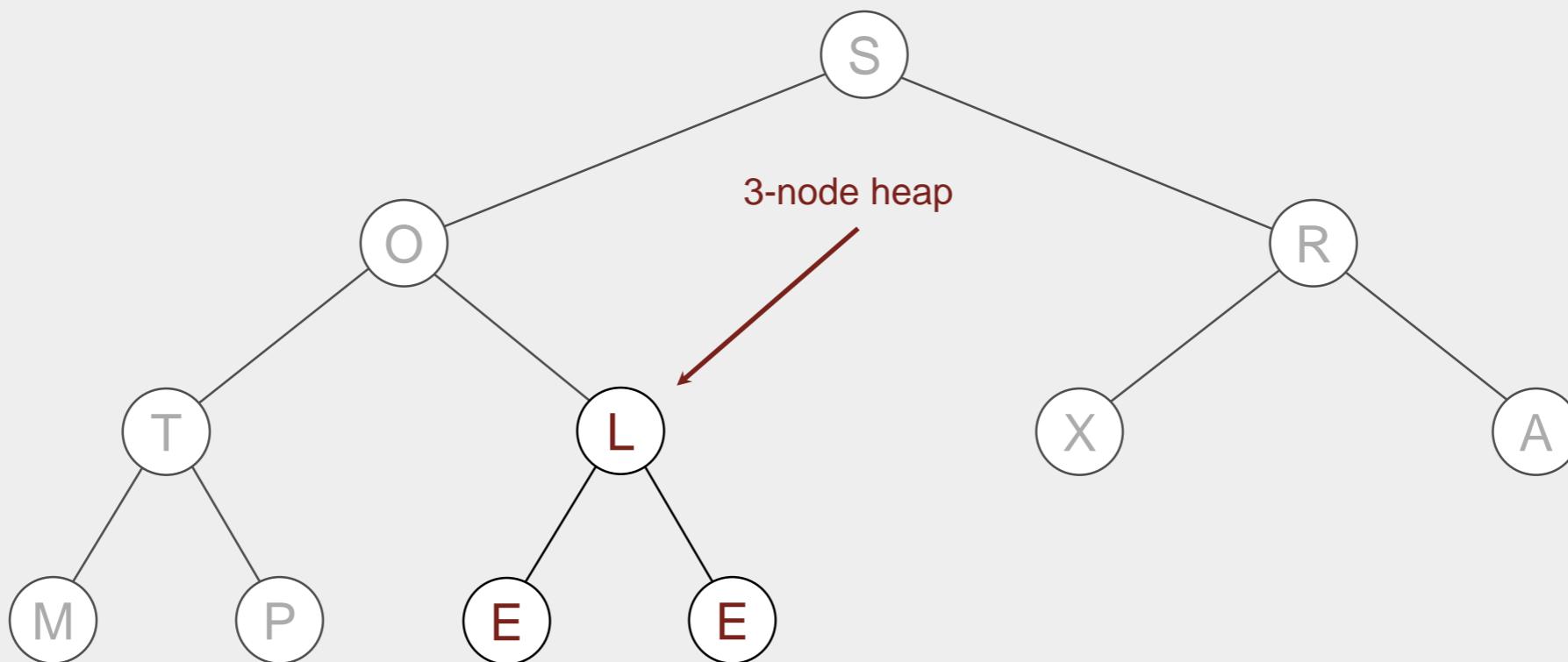
5

10

Heapsort

Heap construction. Build max heap using bottom-up method.

sink 5

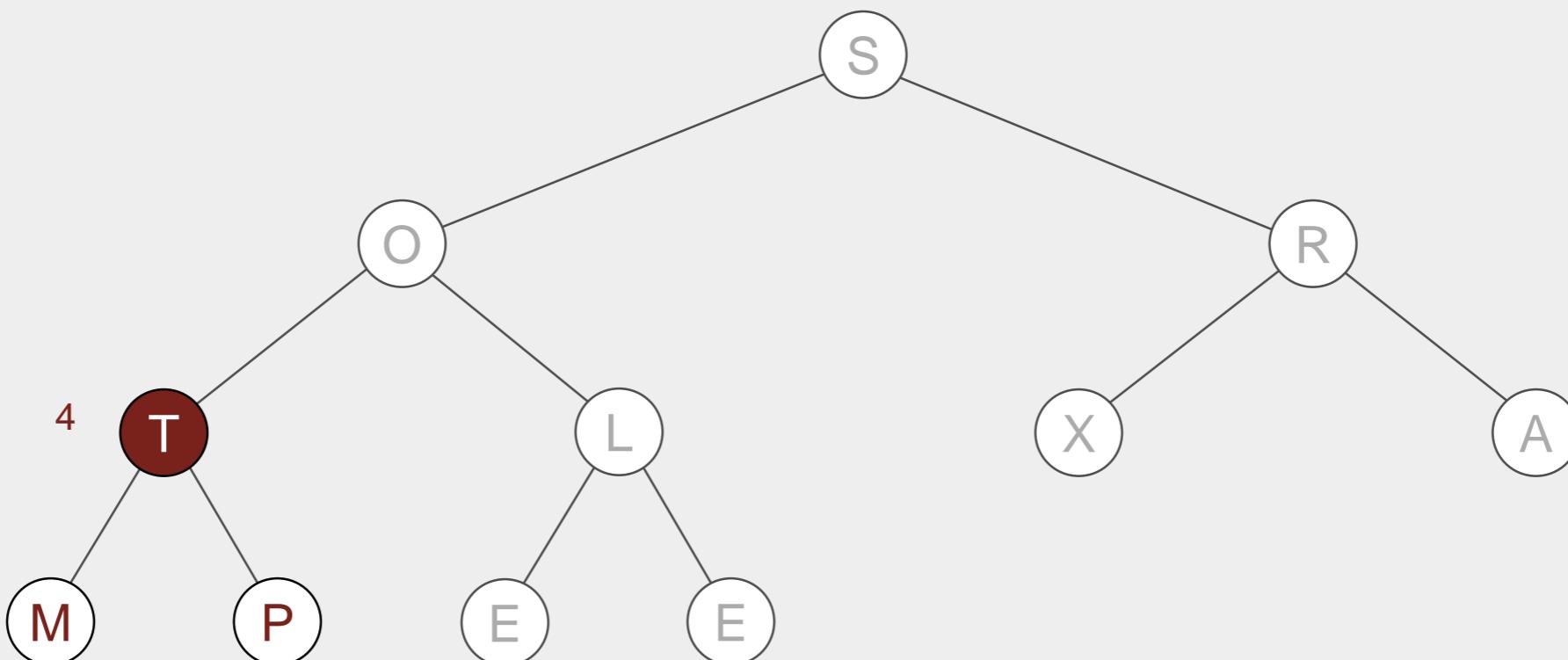


S	O	R	T	L	X	A	M	P	E	E
---	---	---	---	---	---	---	---	---	---	---

Heapsort

Heap construction. Build max heap using bottom-up method.

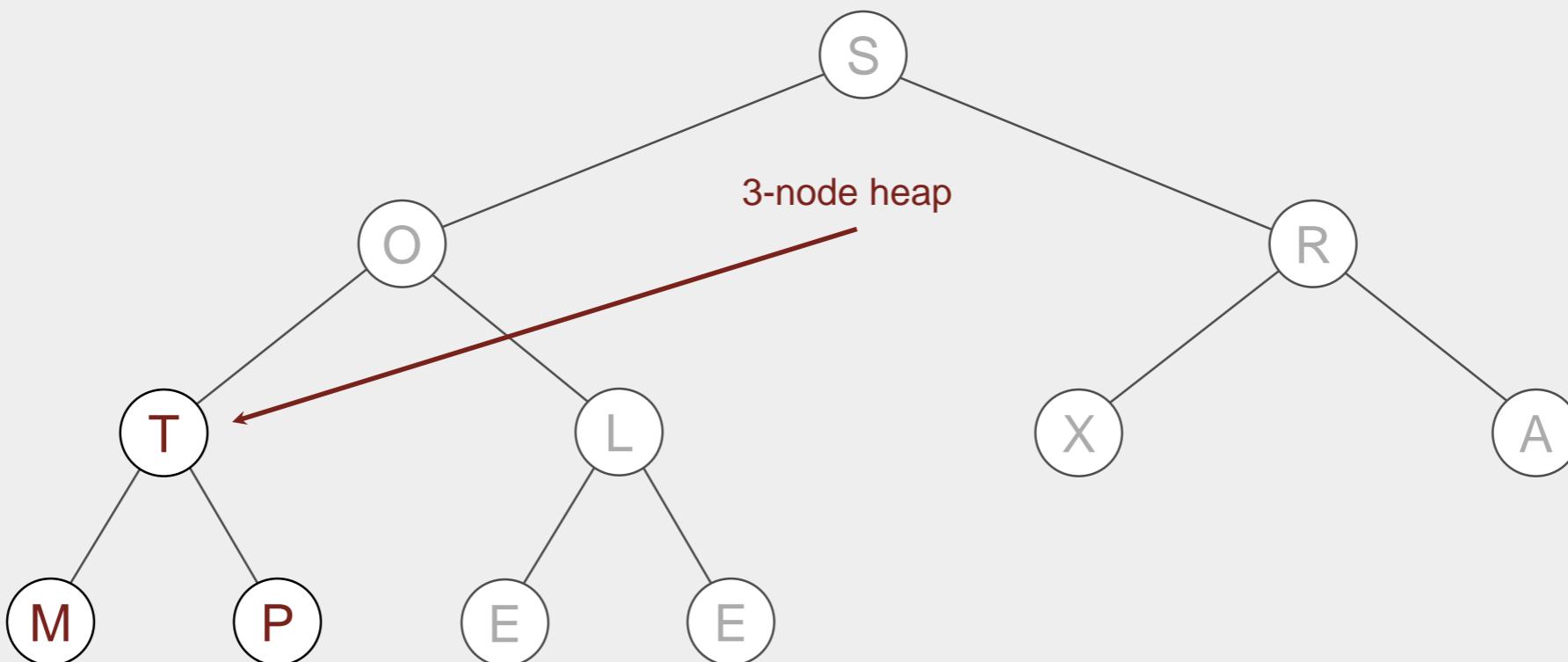
sink 4



S	O	R	T	L	X	A	M	P	E	E
---	---	---	---	---	---	---	---	---	---	---

Heap construction. Build max heap using bottom-up method.

sink 4

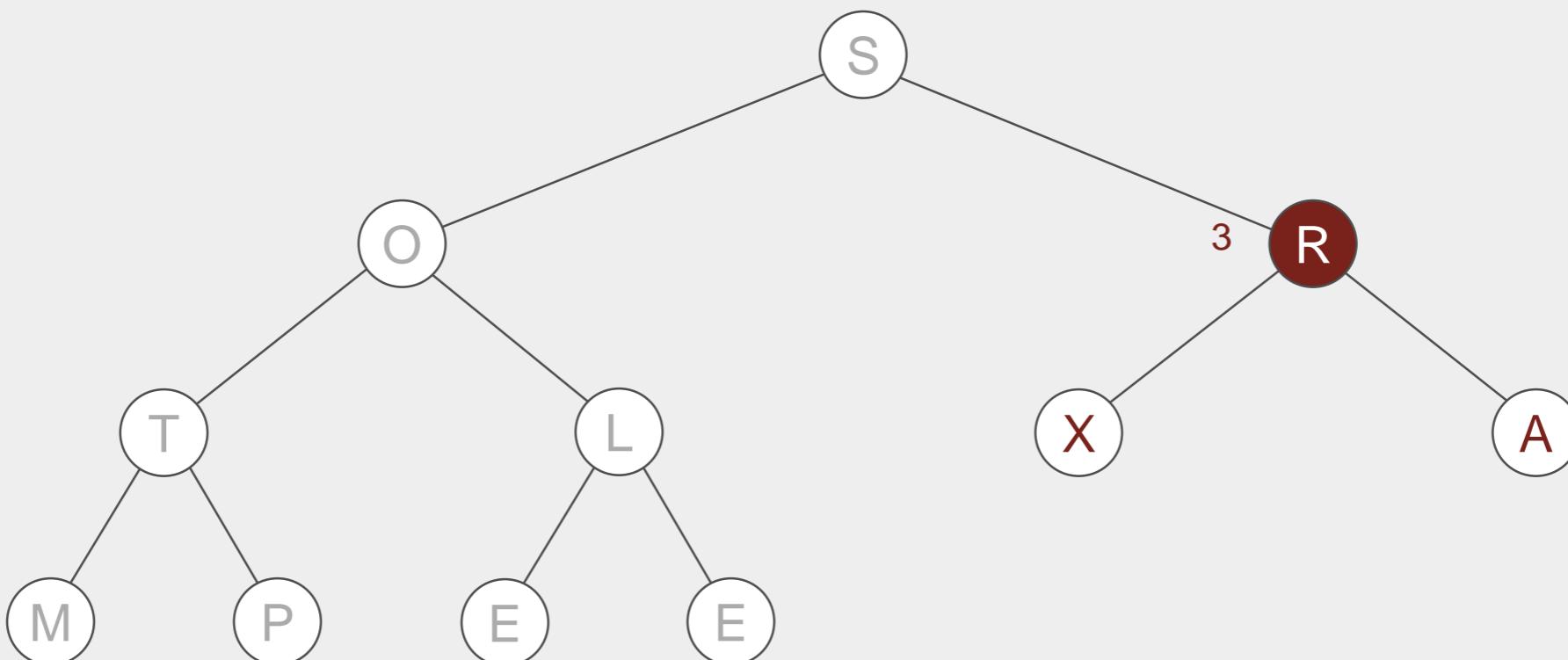


S	O	R	T	L	X	A	M	P	E	E
---	---	---	---	---	---	---	---	---	---	---

Heapsort

Heap construction. Build max heap using bottom-up method.

sink 3

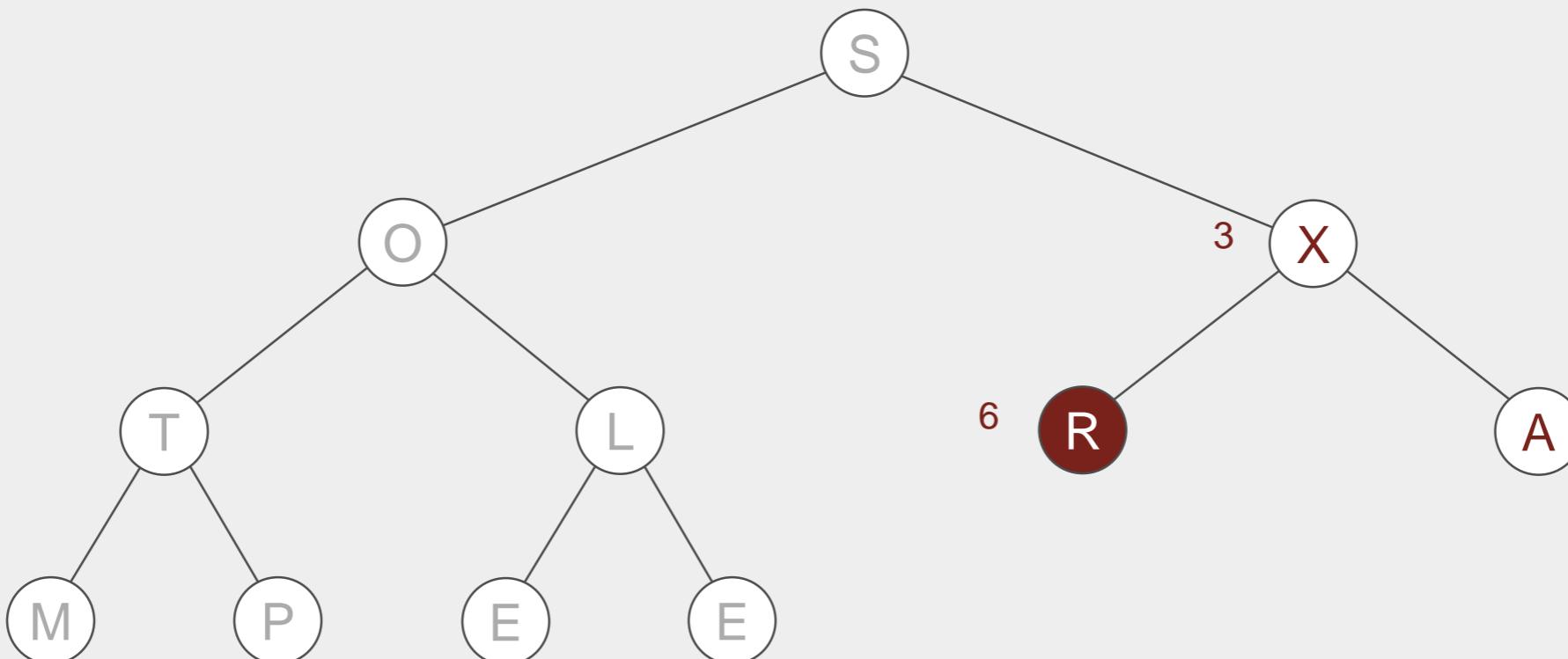


S	O	R	T	L	X	A	M	P	E	E
---	---	---	---	---	---	---	---	---	---	---

Heapsort

Heap construction. Build max heap using bottom-up method.

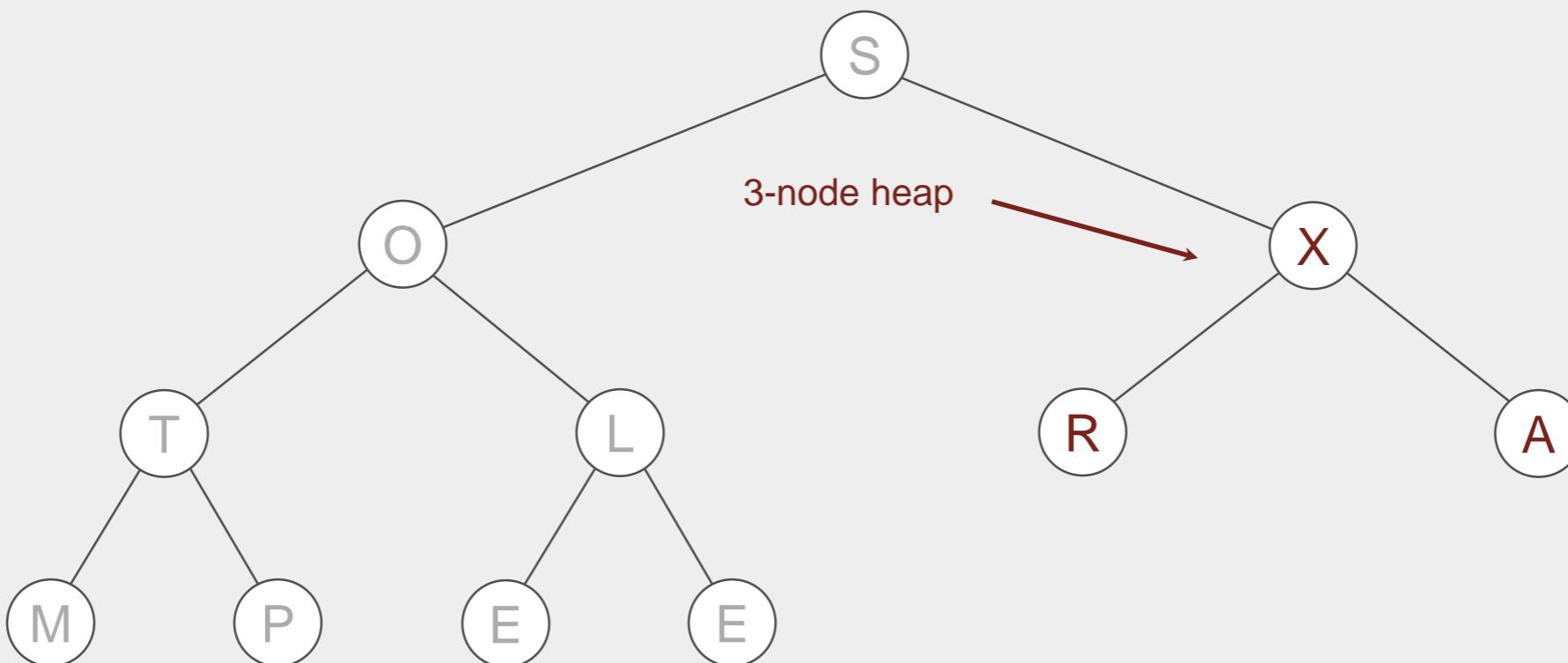
sink 3



S	O	X	T	L	R	A	M	P	E	E
		3			6					

Heap construction. Build max heap using bottom-up method.

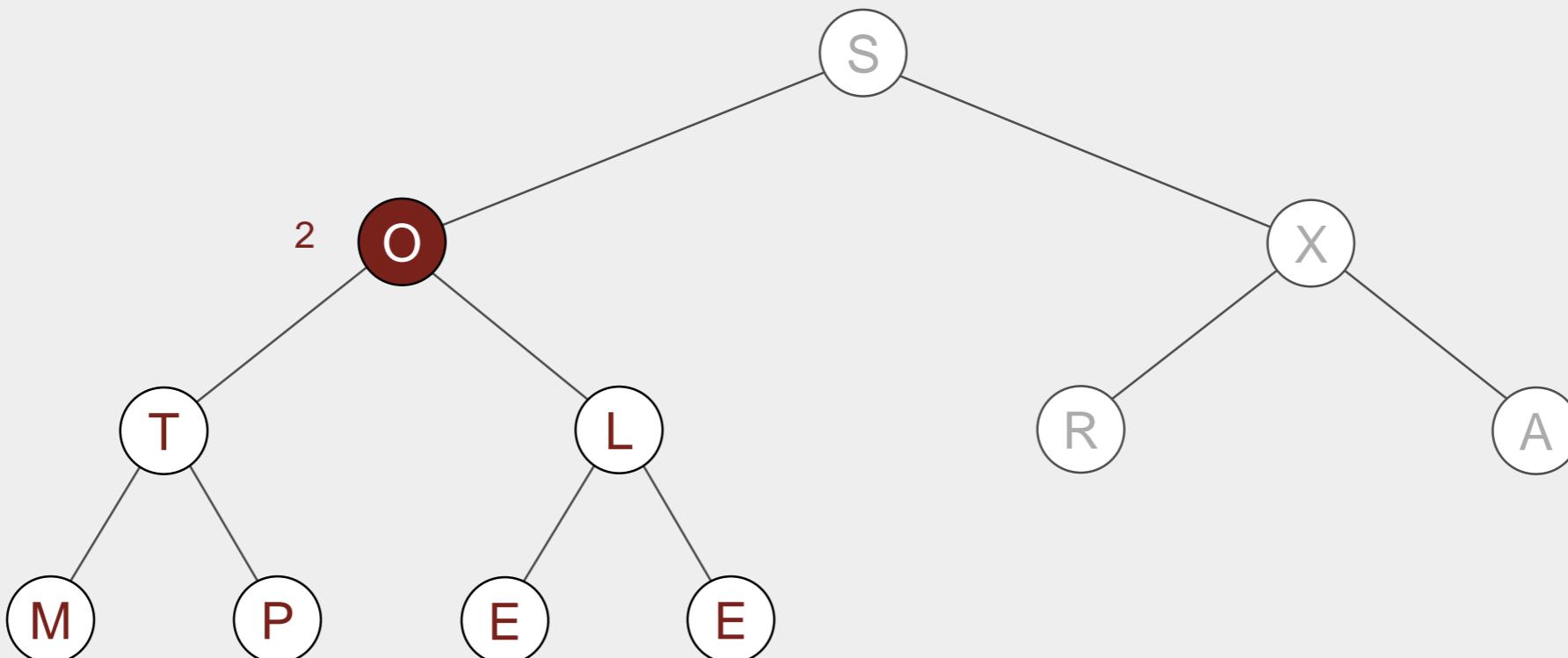
sink 3



S	O	X	T	L	A	A	M	P	E	E
---	---	---	---	---	---	---	---	---	---	---

Heap construction. Build max heap using bottom-up method.

sink 2

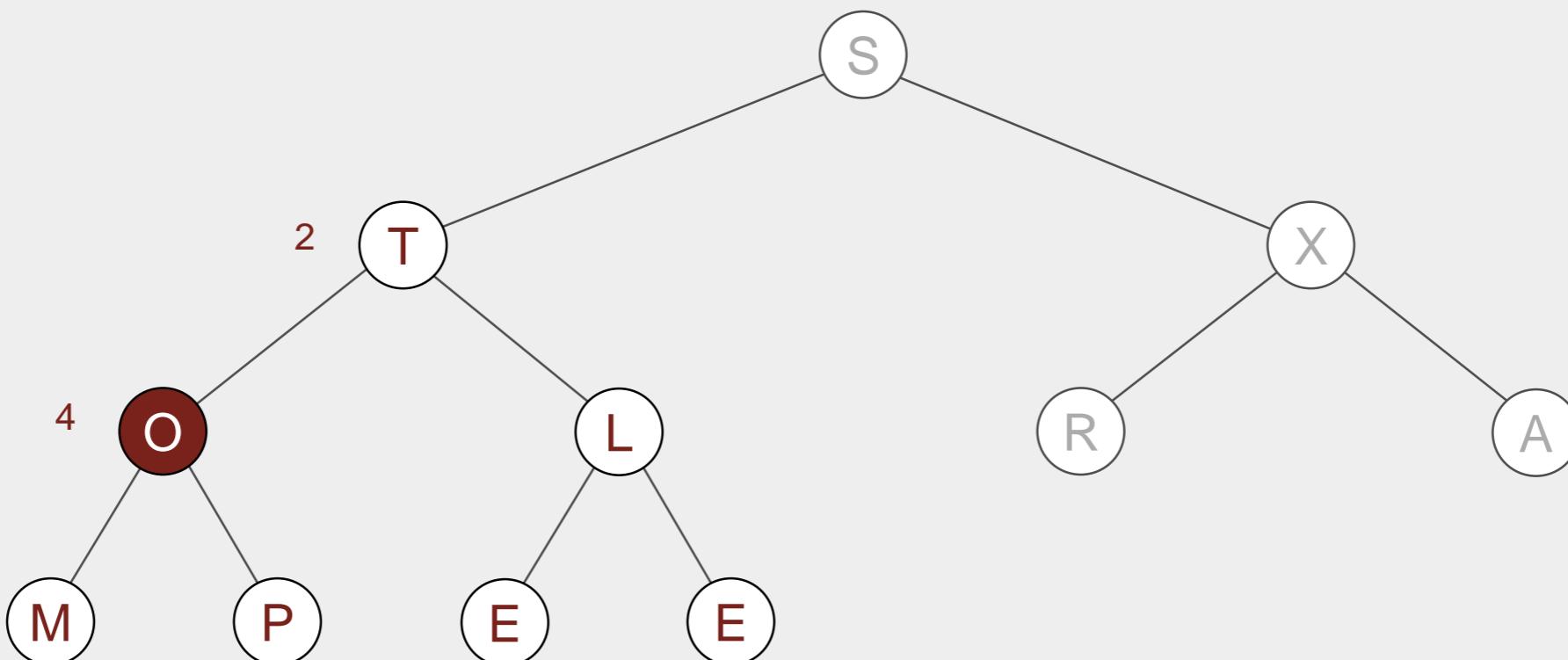


S	O	X	T	L	R	A	M	P	E	E
---	---	---	---	---	---	---	---	---	---	---

Heapsort

Heap construction. Build max heap using bottom-up method.

sink 2

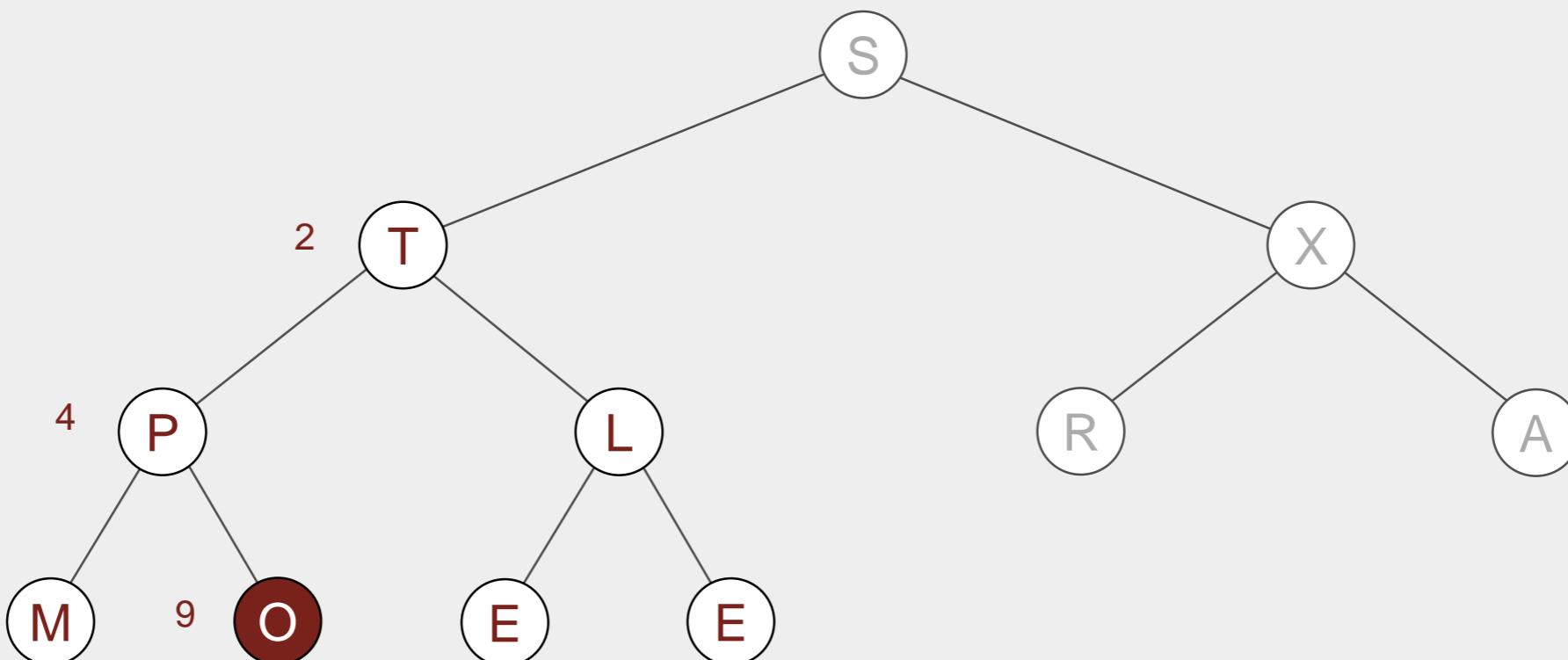


S	T	X	O	L	R	A	M	P	E	E
2	4									

Heapsort

Heap construction. Build max heap using bottom-up method.

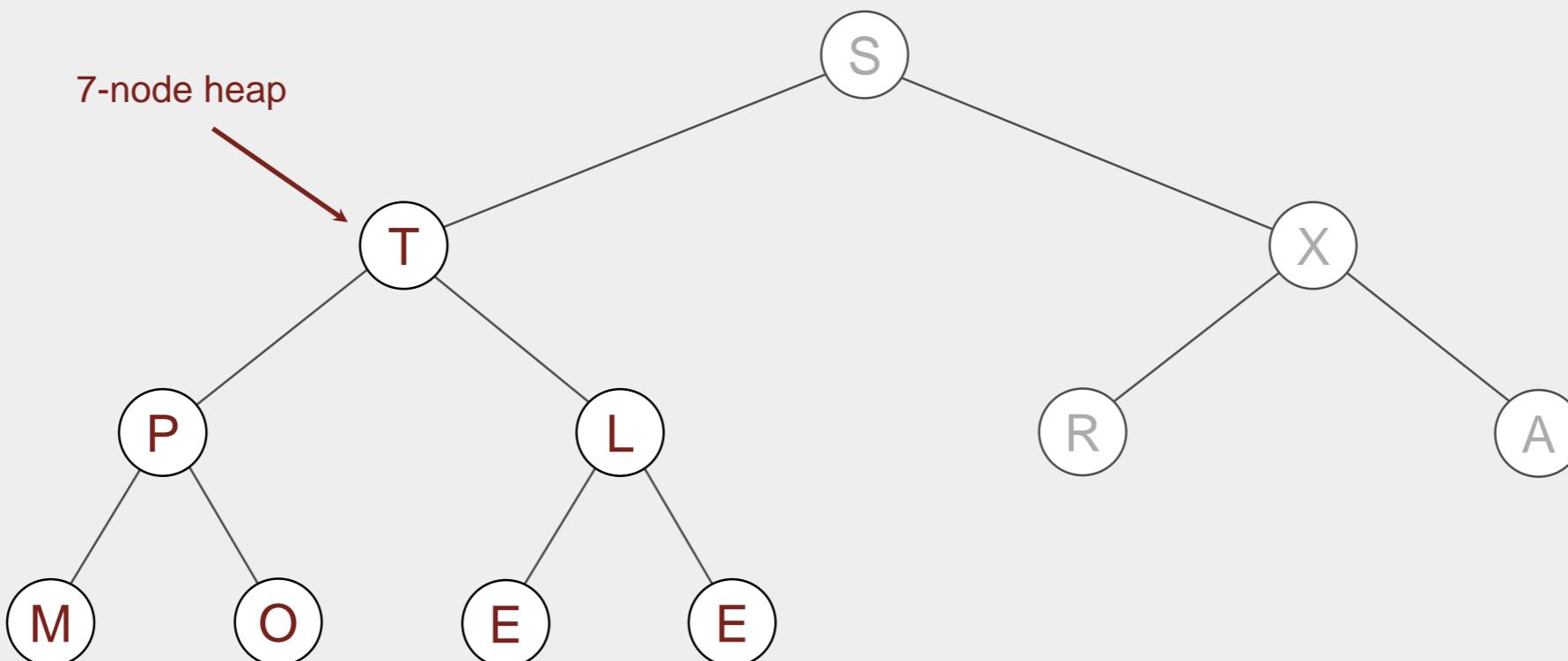
sink 2



S	T	X	P	L	R	A	M	O	E	E
2	4			4			9			

Heap construction. Build max heap using bottom-up method.

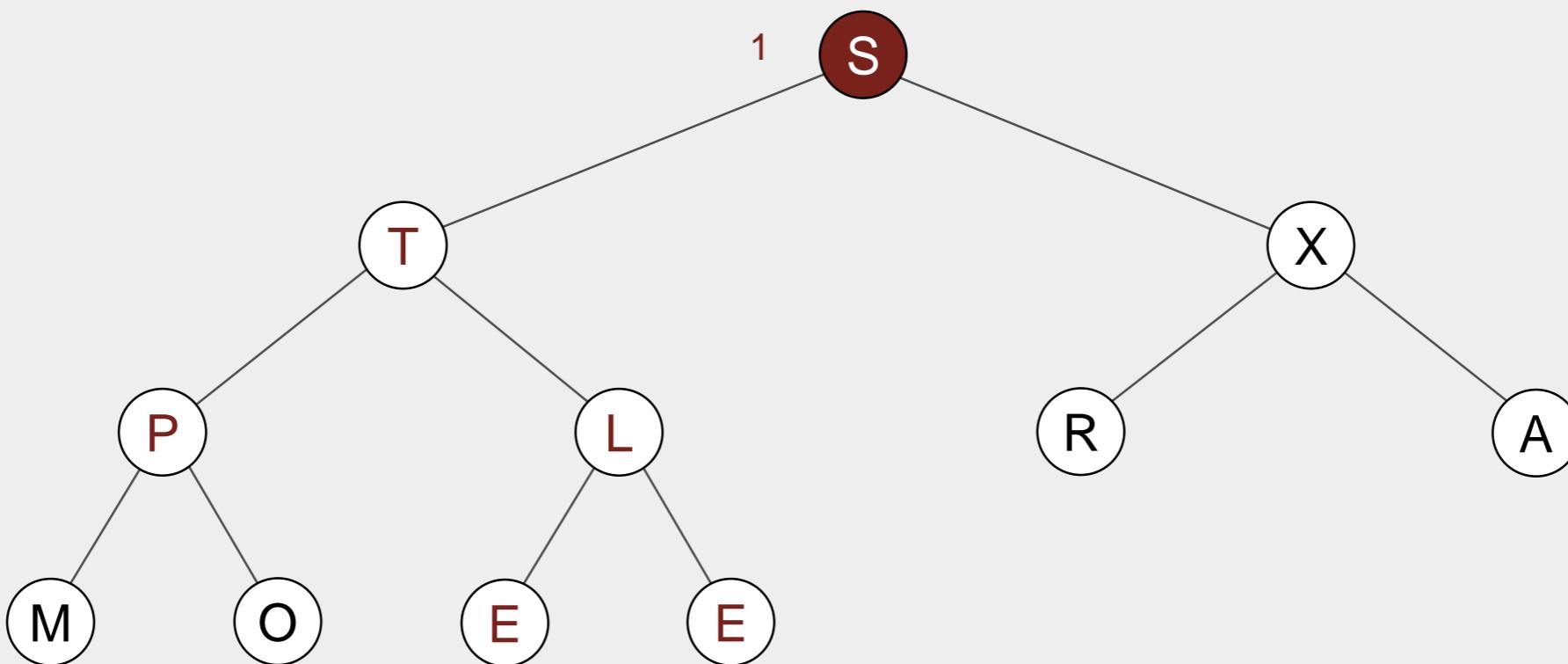
sink 2



S	T	X	P	L	R	A	M	O	E	E
---	---	---	---	---	---	---	---	---	---	---

Heap construction. Build max heap using bottom-up method.

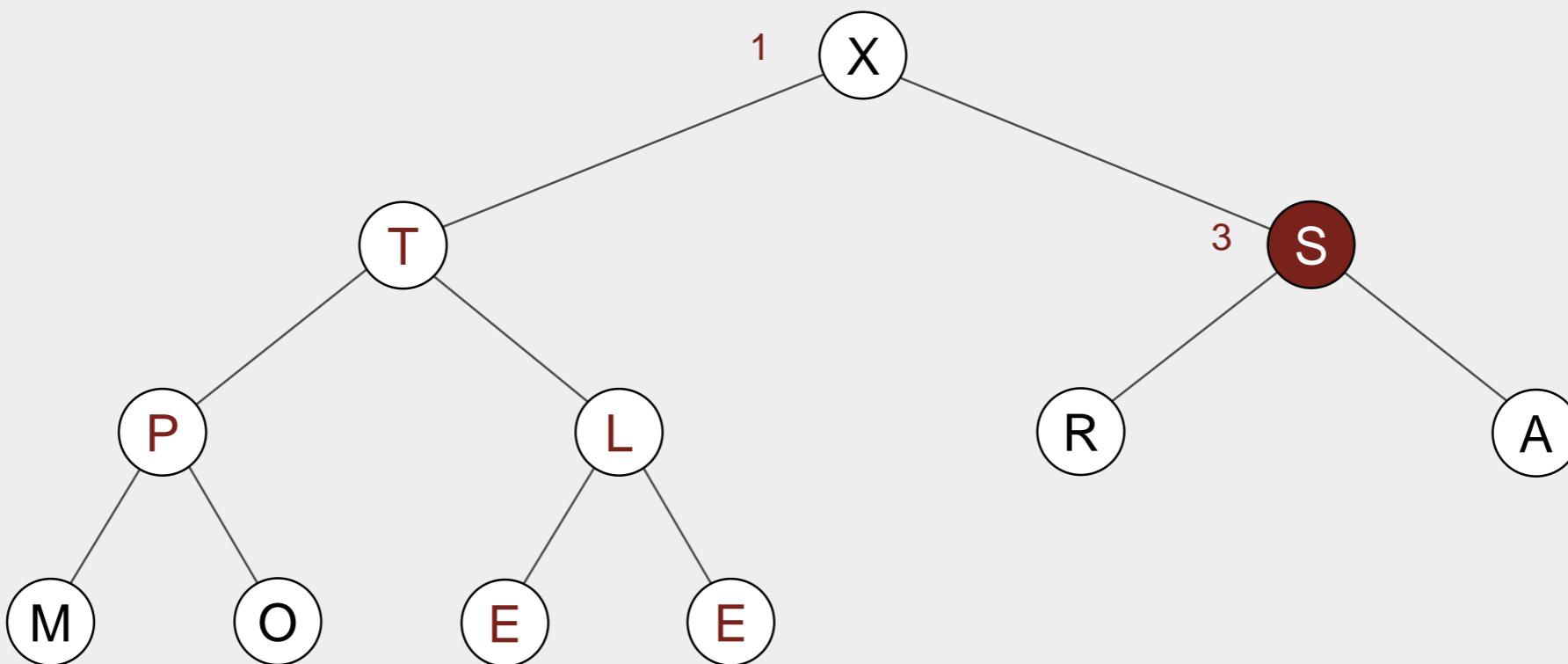
sink 1



S	T	X	P	L	R	A	M	O	E	E
---	---	---	---	---	---	---	---	---	---	---

Heap construction. Build max heap using bottom-up method.

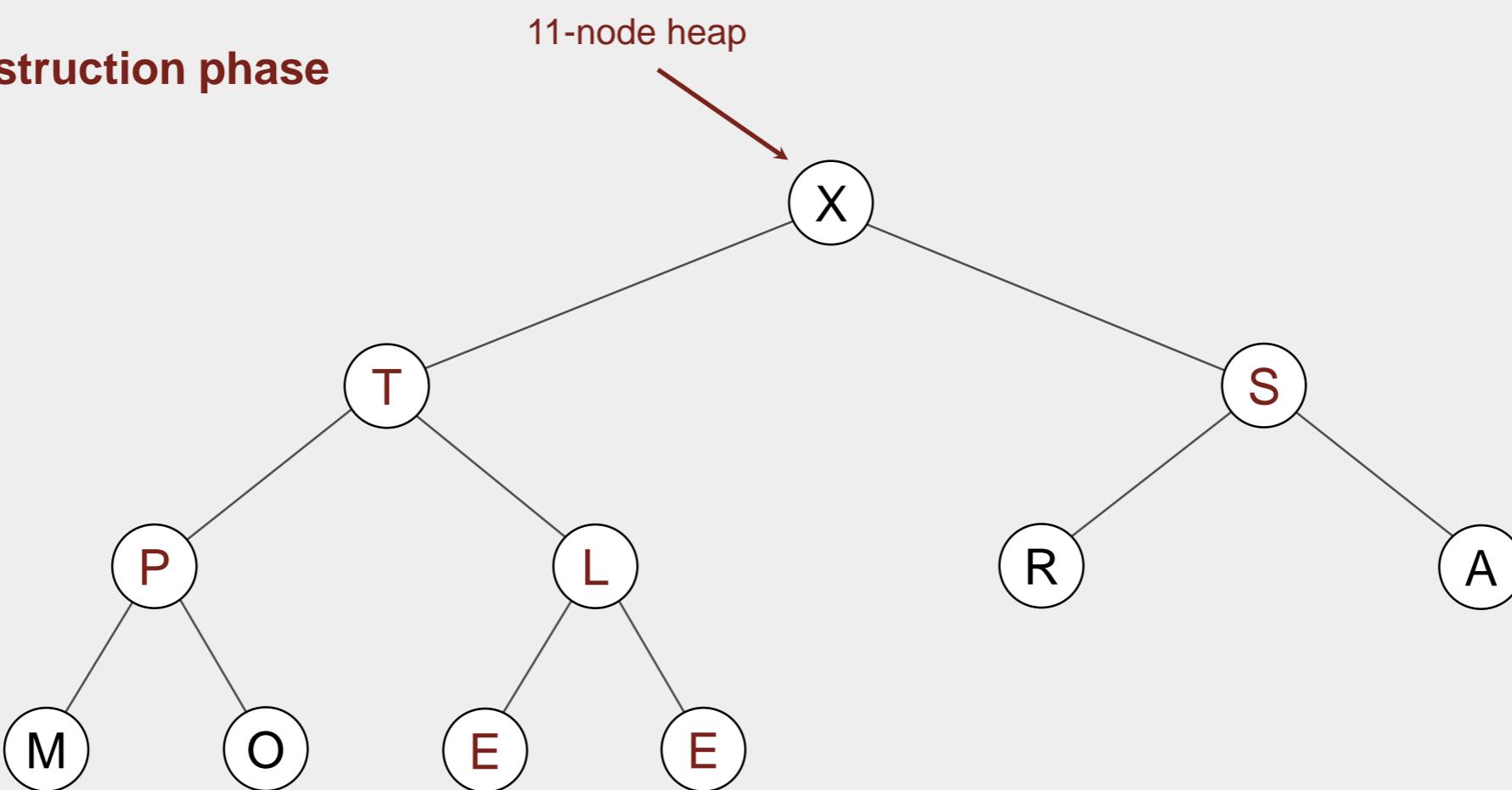
sink 1



X	T	S	P	L	R	A	M	O	E	E
1		3								

Heap construction. Build max heap using bottom-up method.

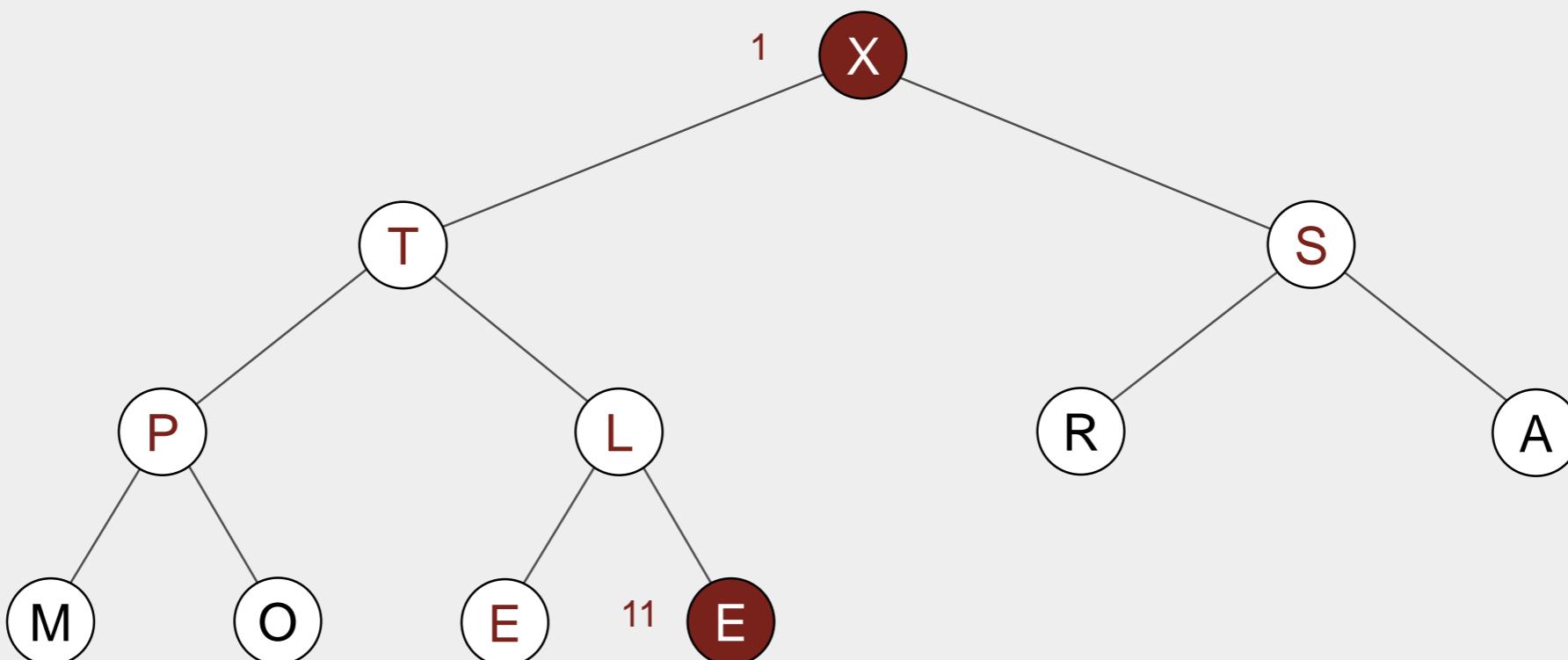
end of construction phase



X	T	S	P	L	R	A	M	O	E	E
---	---	---	---	---	---	---	---	---	---	---

Sortdown. Repeatedly delete the largest remaining item.

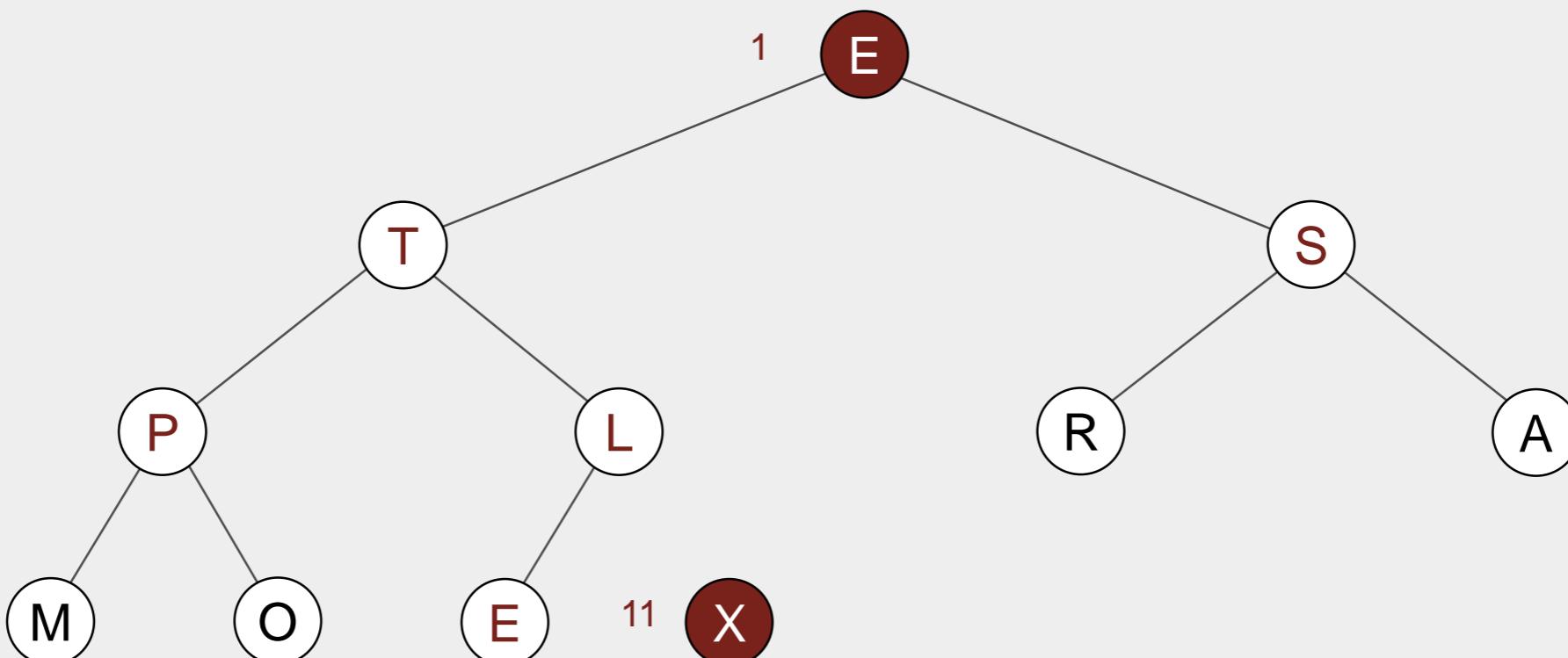
exchange 1 and 11



X	T	S	P	L	R	A	M	O	E	E
1										11

Sortdown. Repeatedly delete the largest remaining item.

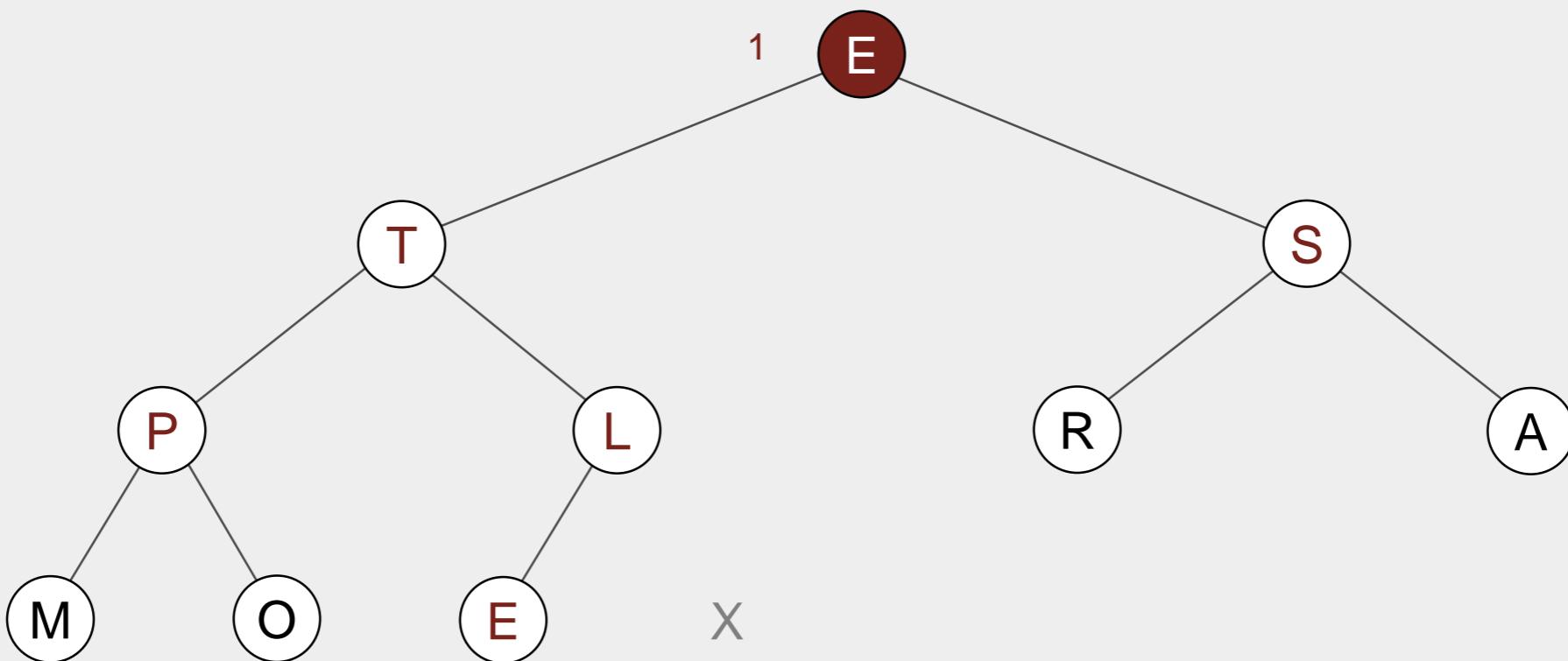
exchange 1 and 11



E	T	S	P	L	R	A	M	O	E	X
1									11	

Sortdown. Repeatedly delete the largest remaining item.

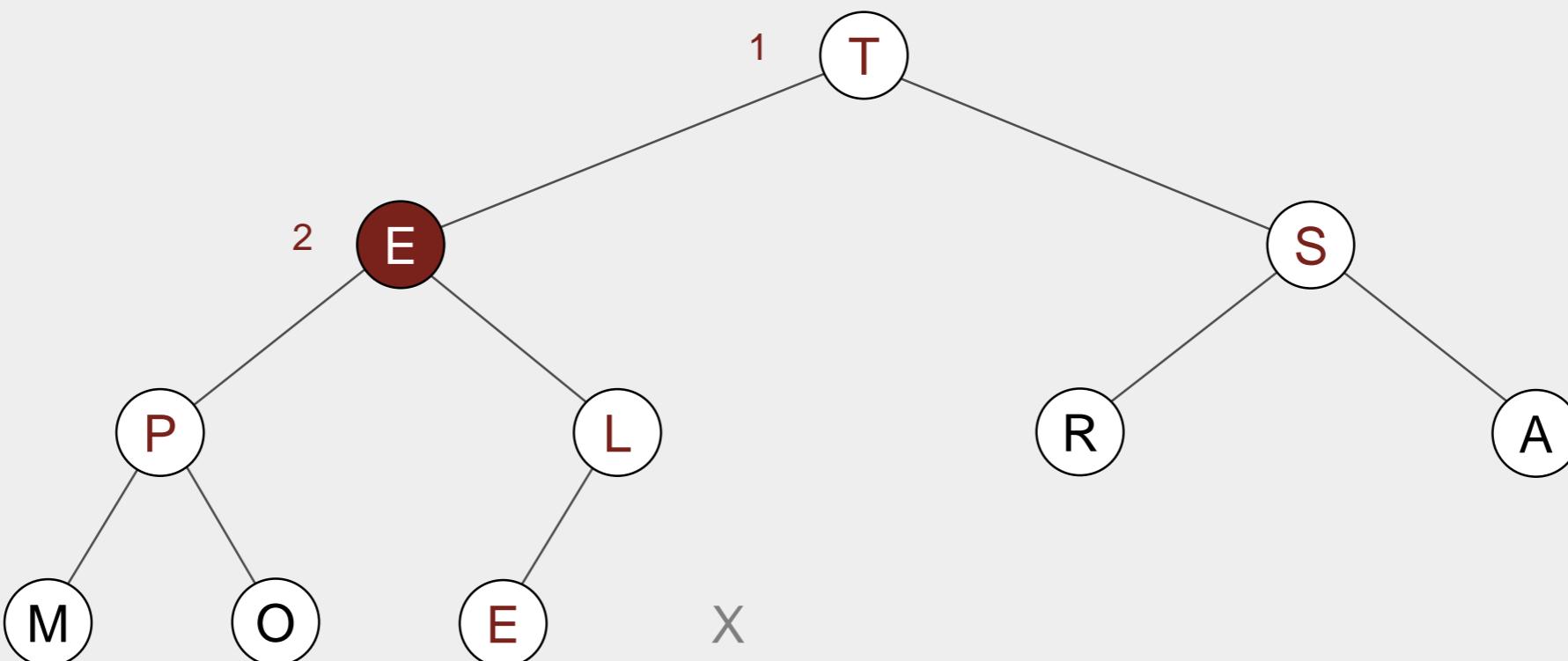
sink 1



E	T	S	P	L	R	A	M	O	E	X
---	---	---	---	---	---	---	---	---	---	---

Sortdown. Repeatedly delete the largest remaining item.

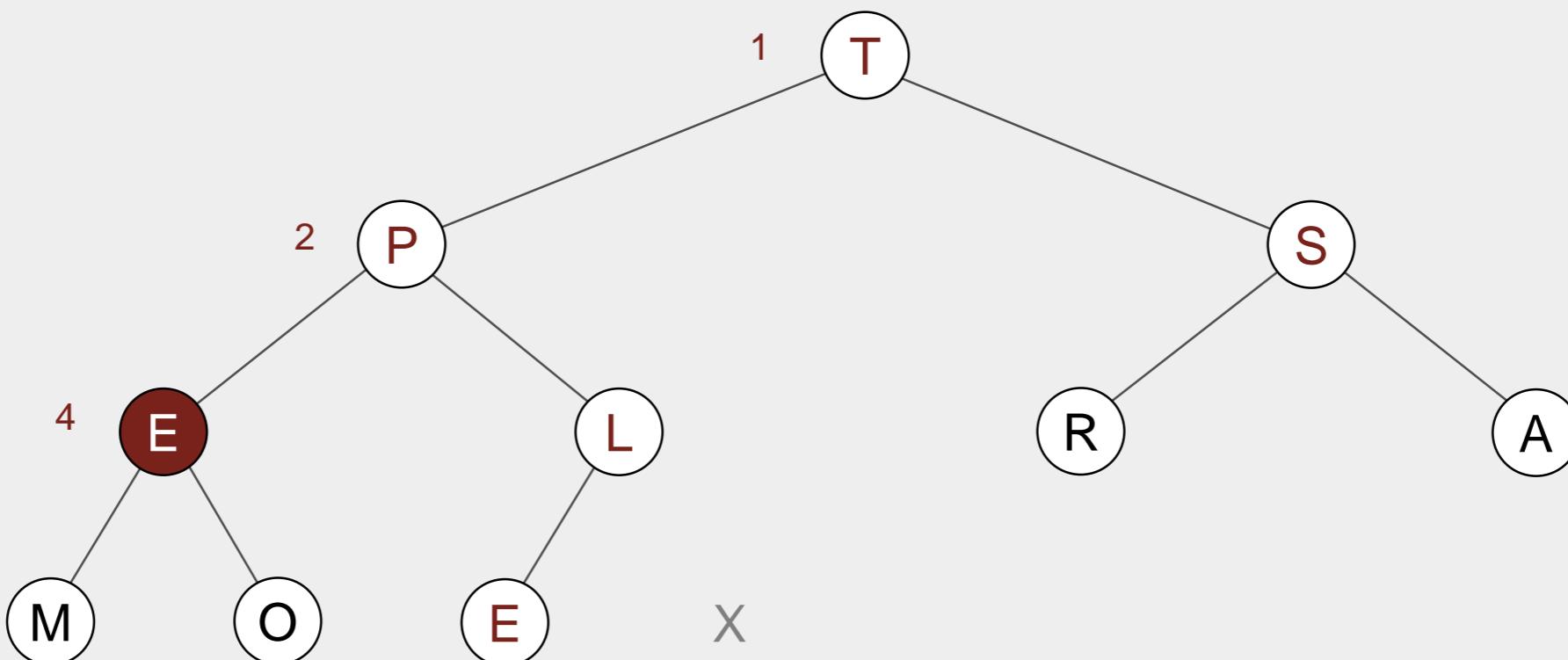
sink 1



T	E	S	P	L	R	A	M	O	E	X
1	2									

Sortdown. Repeatedly delete the largest remaining item.

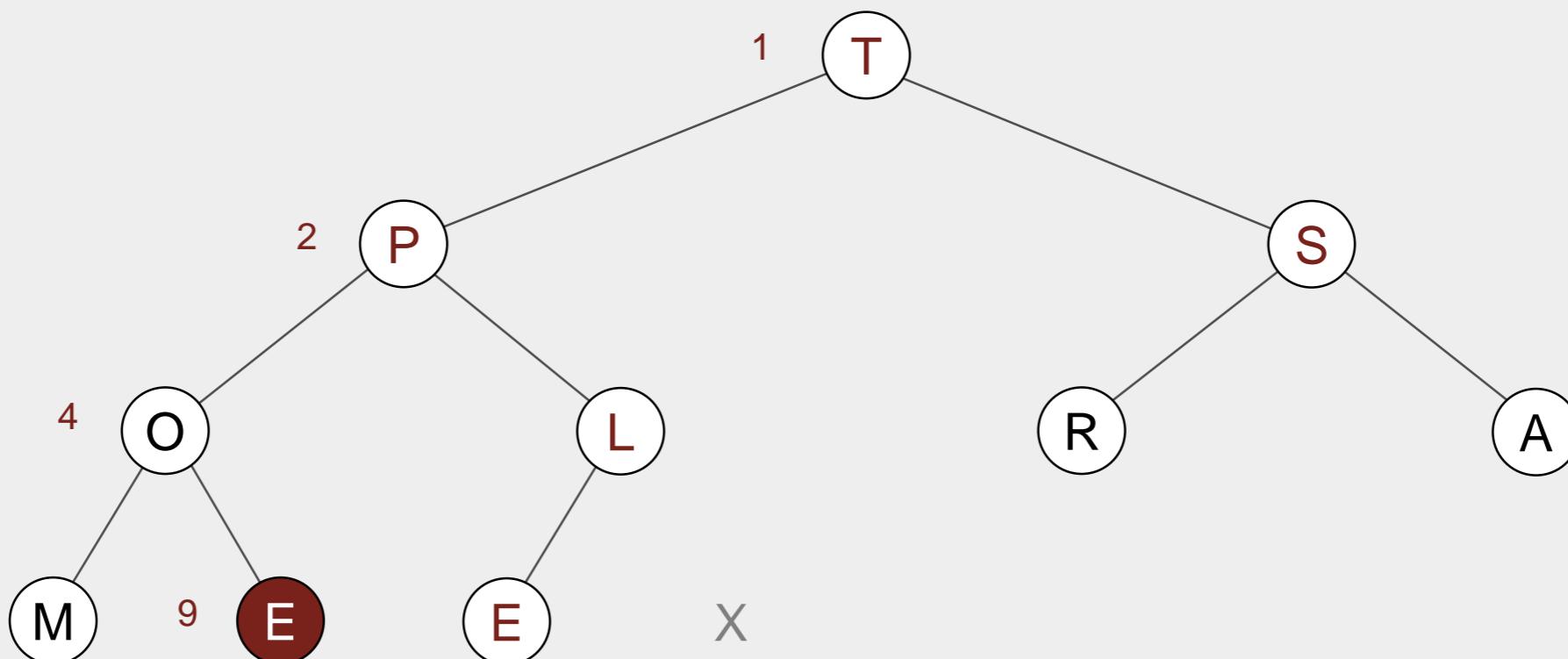
sink 1



T	P	S	E	L	R	A	M	O	E	X
1	2	4								

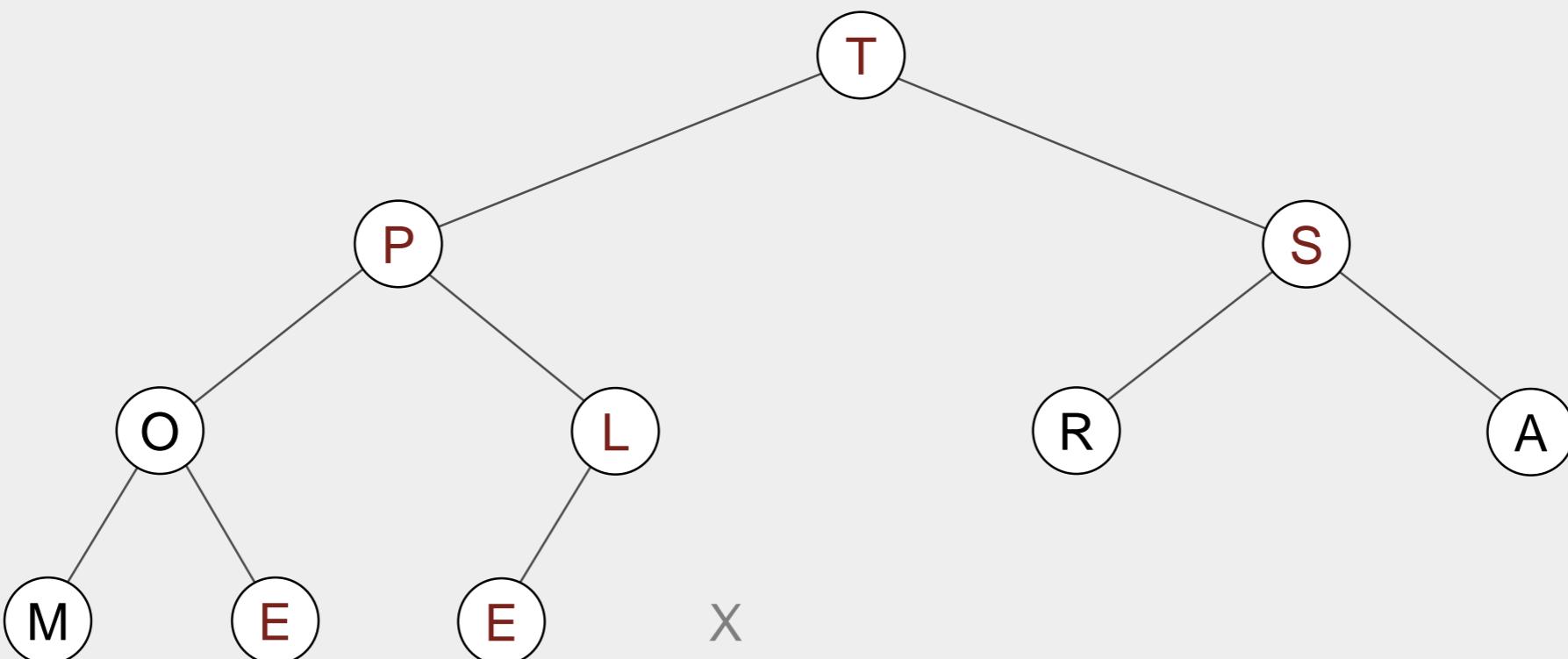
Sortdown. Repeatedly delete the largest remaining item.

sink 1



T	P	S	O	L	R	A	M	E	E	X
1	2	4						9		

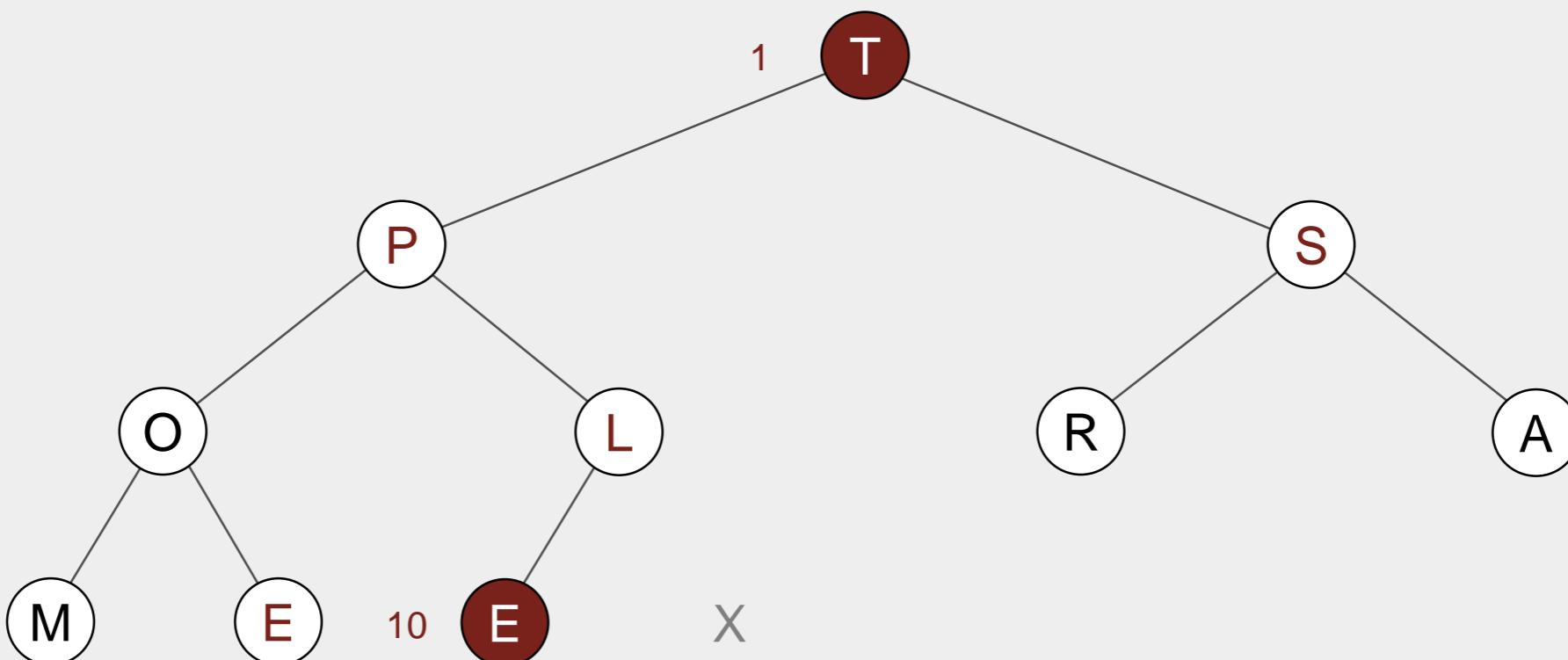
Sortdown. Repeatedly delete the largest remaining item.



T	P	S	O	L	R	A	M	E	E	X
---	---	---	---	---	---	---	---	---	---	---

Sortdown. Repeatedly delete the largest remaining item.

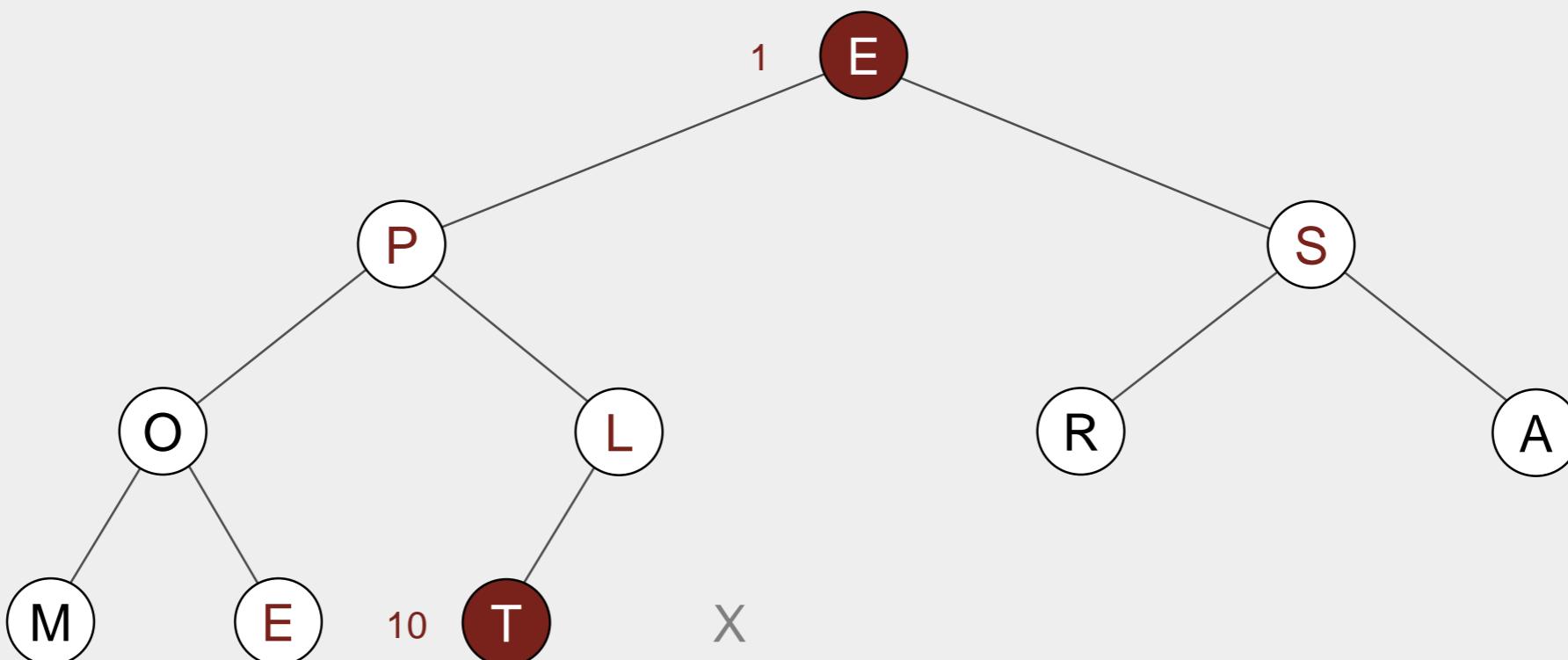
exchange 1 and 10



T	P	S	O	L	R	A	M	E	E	X
1									10	

Sortdown. Repeatedly delete the largest remaining item.

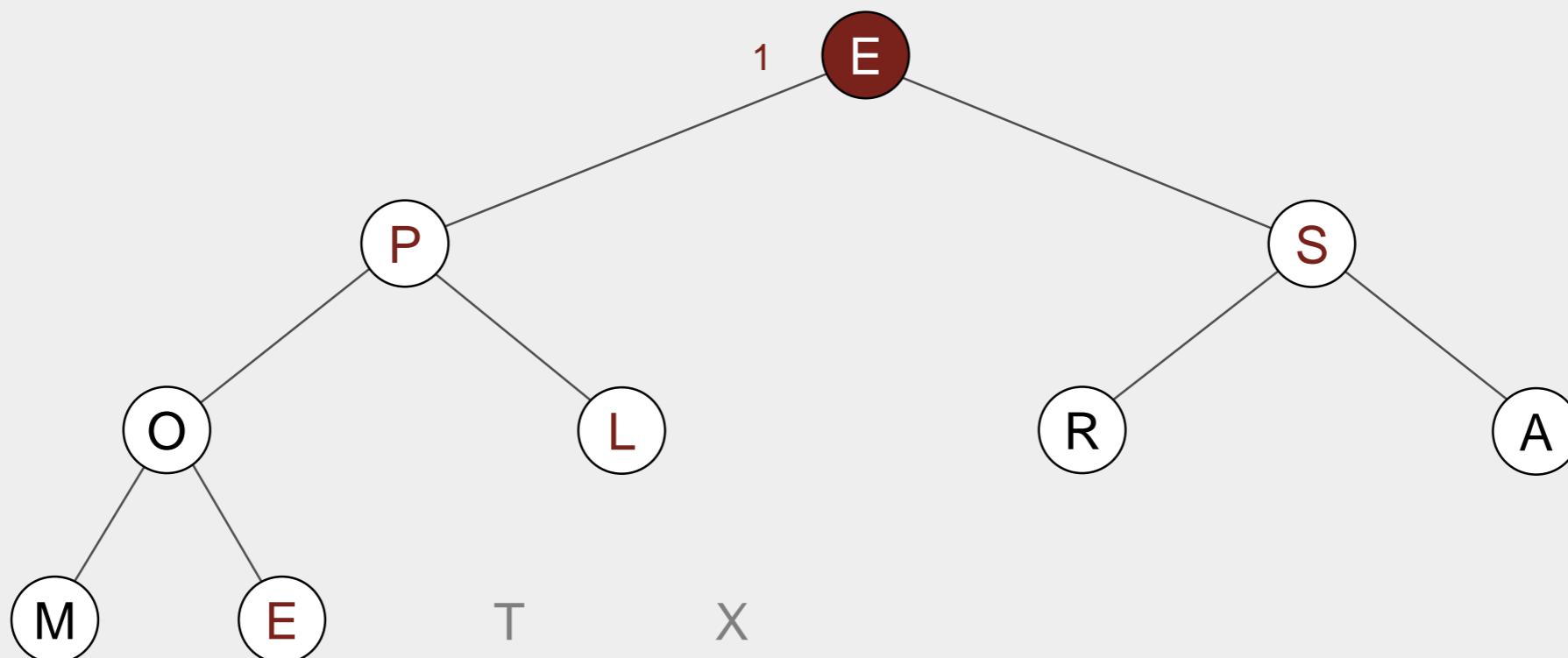
exchange 1 and 10



E	P	S	O	L	R	A	M	E	T	X
1								10		

Sortdown. Repeatedly delete the largest remaining item.

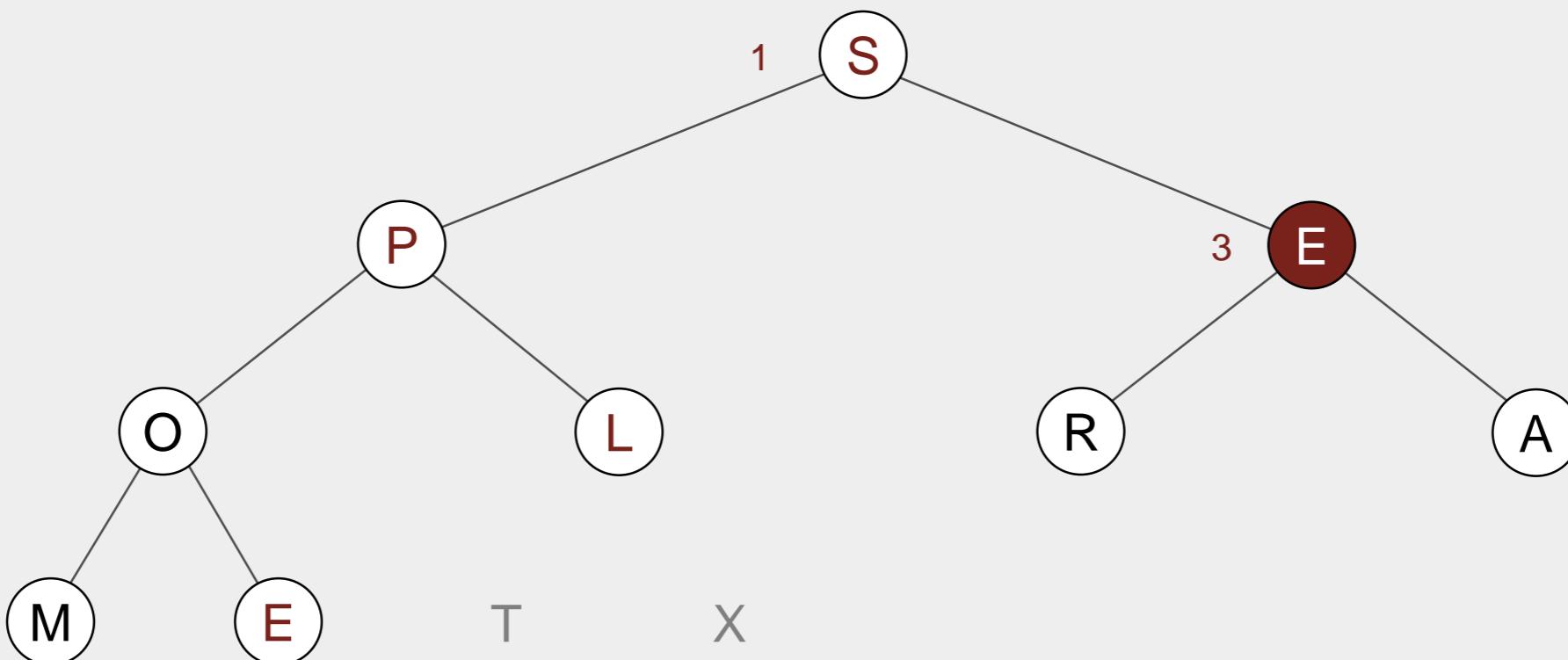
sink 1



E	P	S	O	L	R	A	M	E	T	X
---	---	---	---	---	---	---	---	---	---	---

Sortdown. Repeatedly delete the largest remaining item.

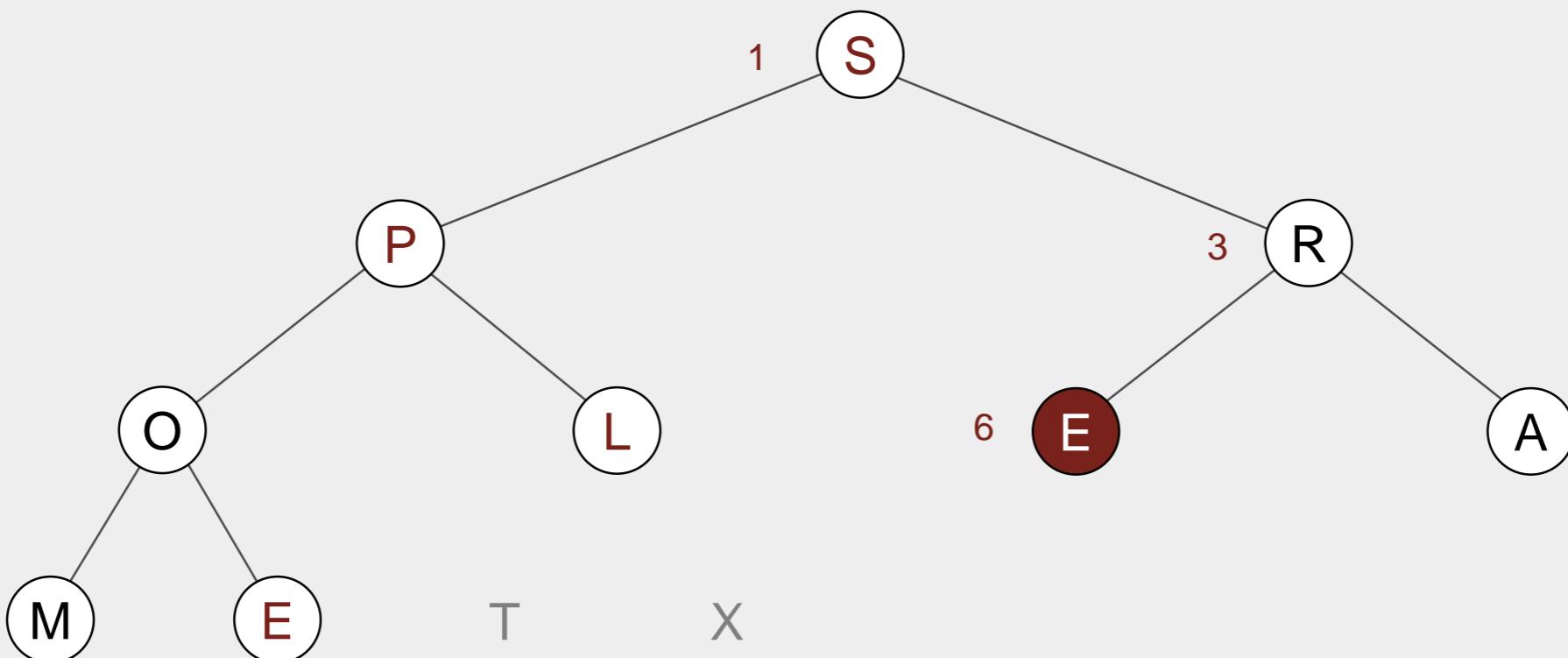
sink 1



S	P	E	O	L	R	A	M	E	T	X
1		3								

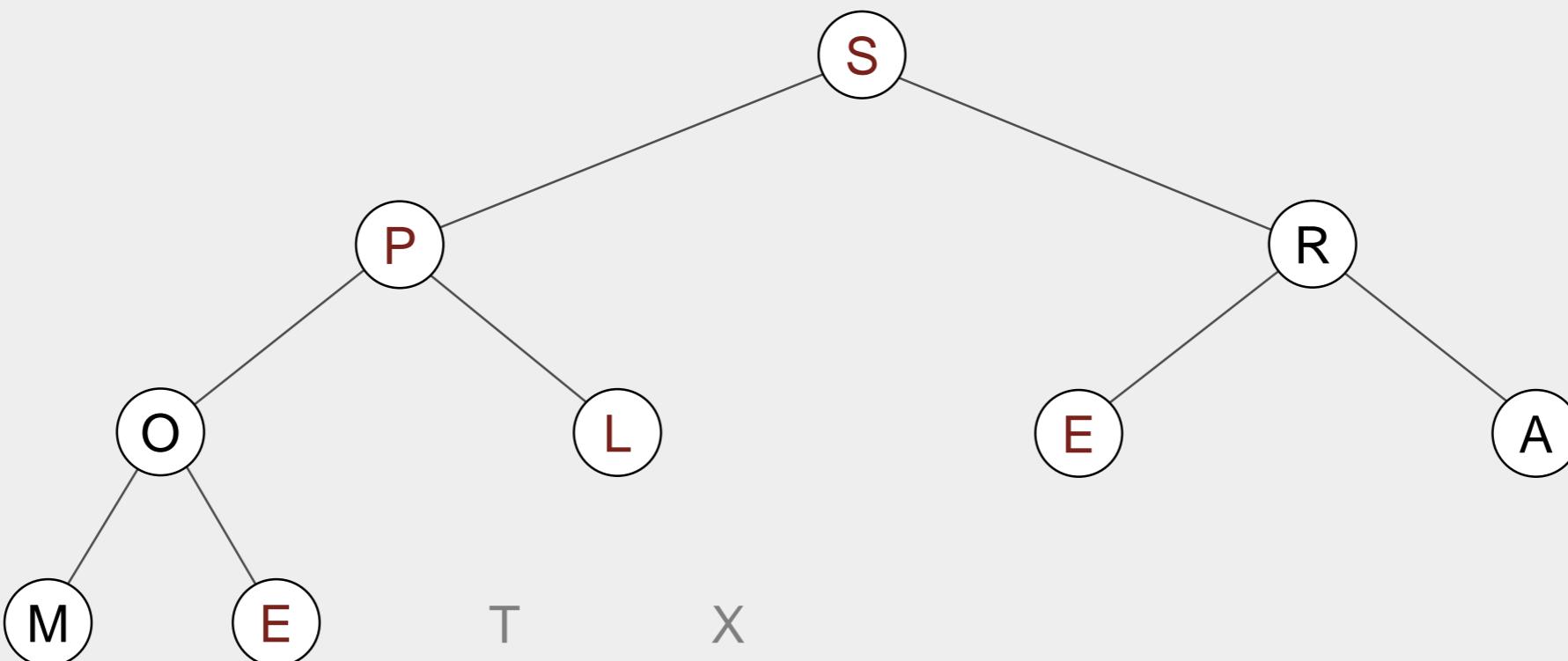
Sortdown. Repeatedly delete the largest remaining item.

sink 1



S	P	R	O	L	E	A	M	E	T	X
1	3				6					

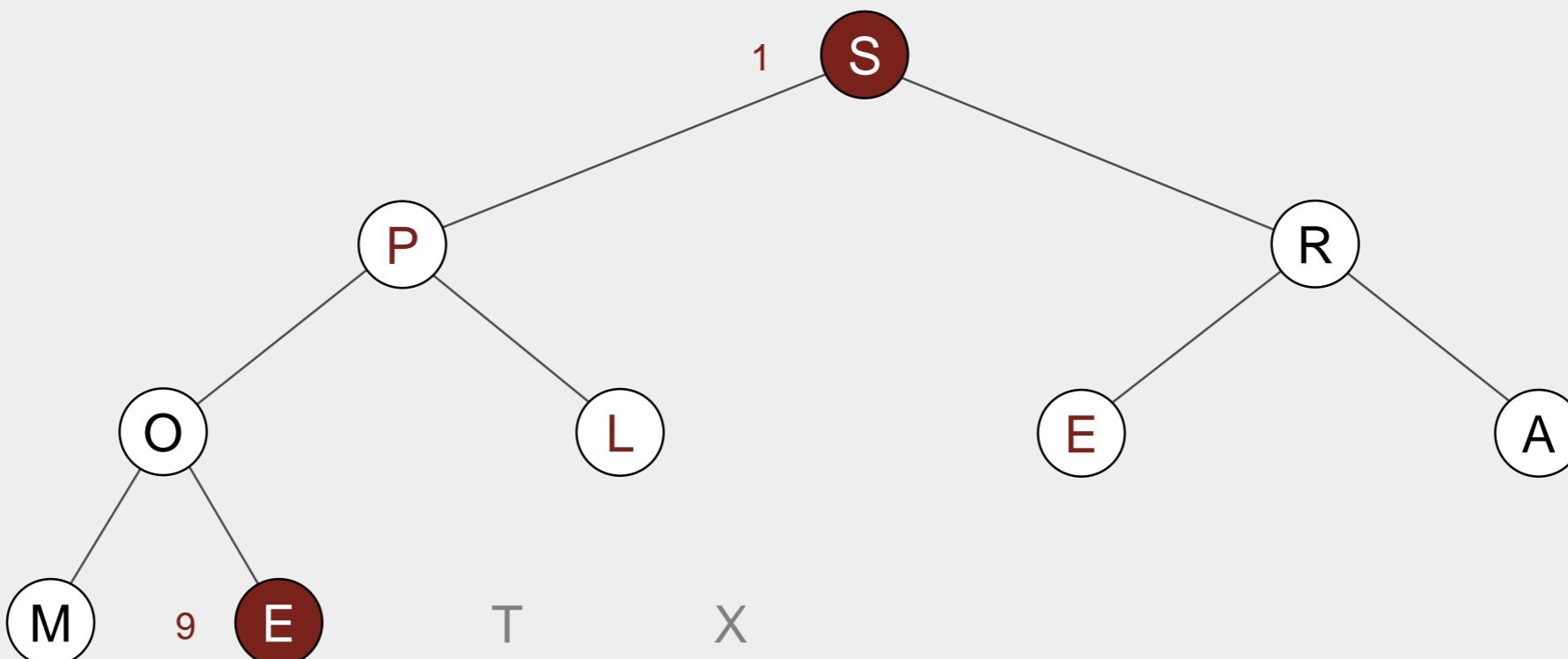
Sortdown. Repeatedly delete the largest remaining item.



S	P	R	O	L	E	A	M	E	T	X
---	---	---	---	---	---	---	---	---	---	---

Sortdown. Repeatedly delete the largest remaining item.

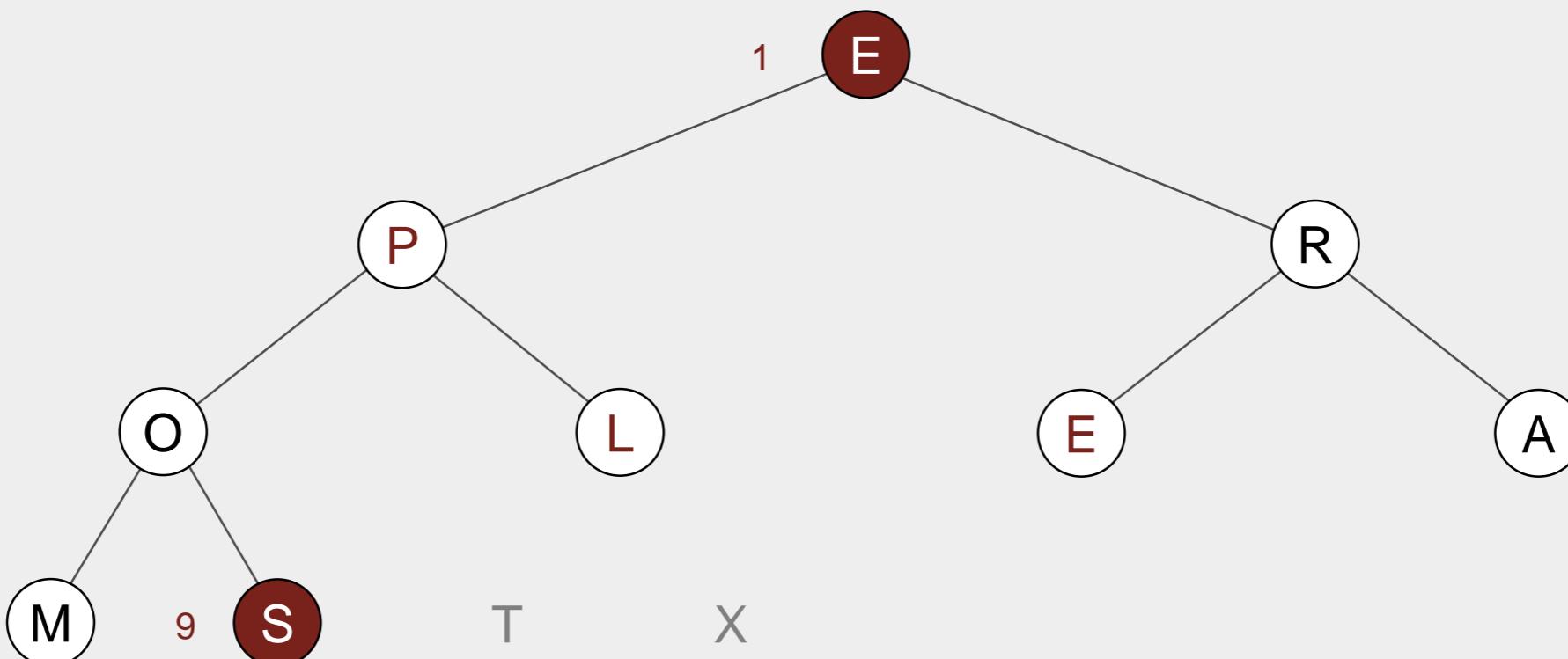
exchange 1 and 9



S	P	R	O	L	E	A	M	E	T	X
1									9	

Sortdown. Repeatedly delete the largest remaining item.

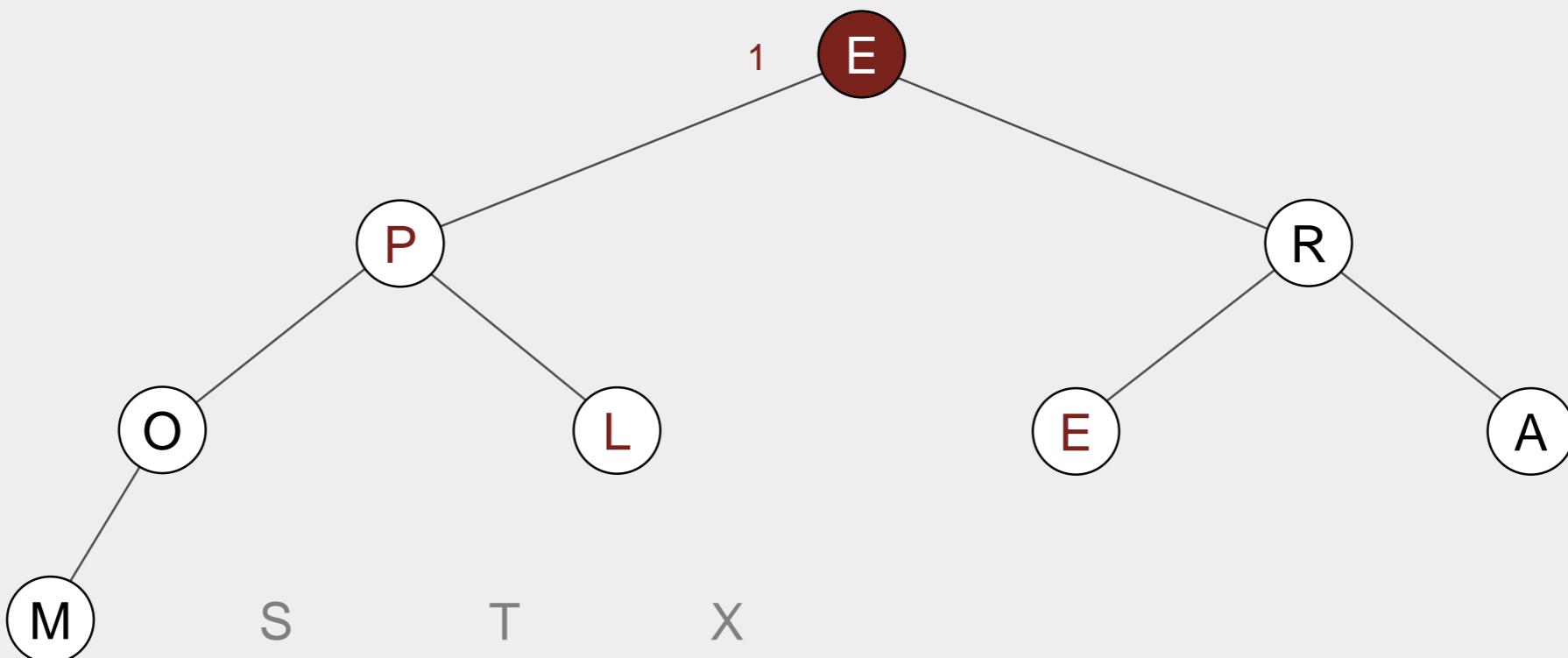
exchange 1 and 9



E	P	R	O	L	E	A	M	S	T	X
1								9		

Sortdown. Repeatedly delete the largest remaining item.

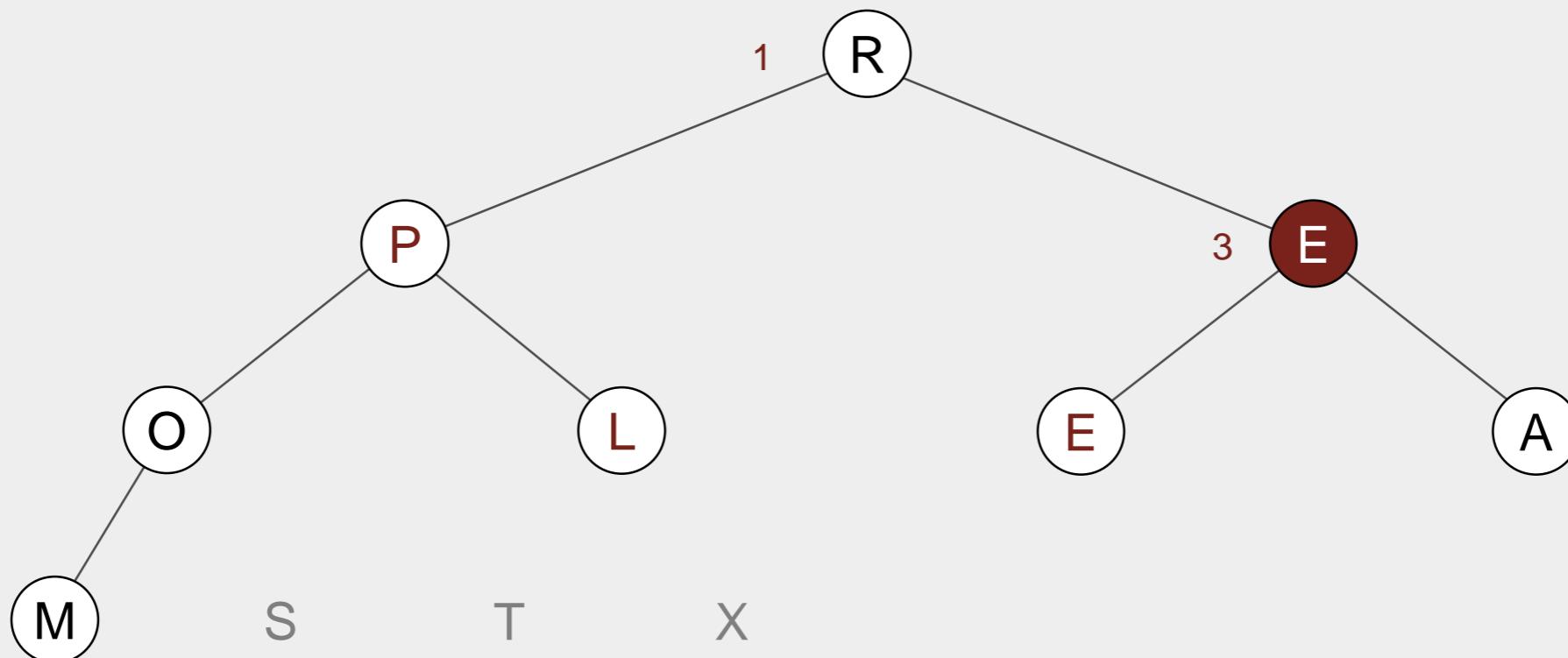
sink 1



E	P	R	O	L	E	A	M	S	T	X
---	---	---	---	---	---	---	---	---	---	---

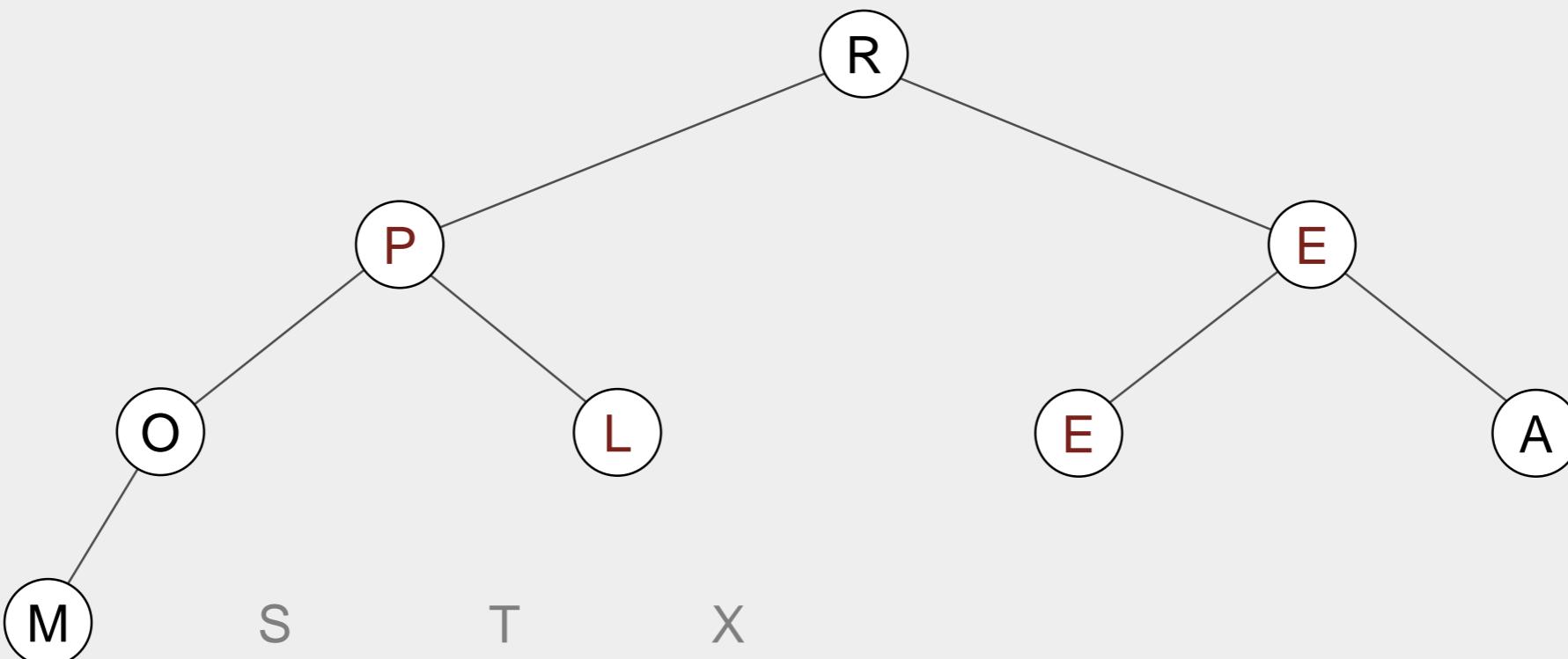
Sortdown. Repeatedly delete the largest remaining item.

sink 1



R	P	E	O	L	E	A	M	S	T	X
1		3								

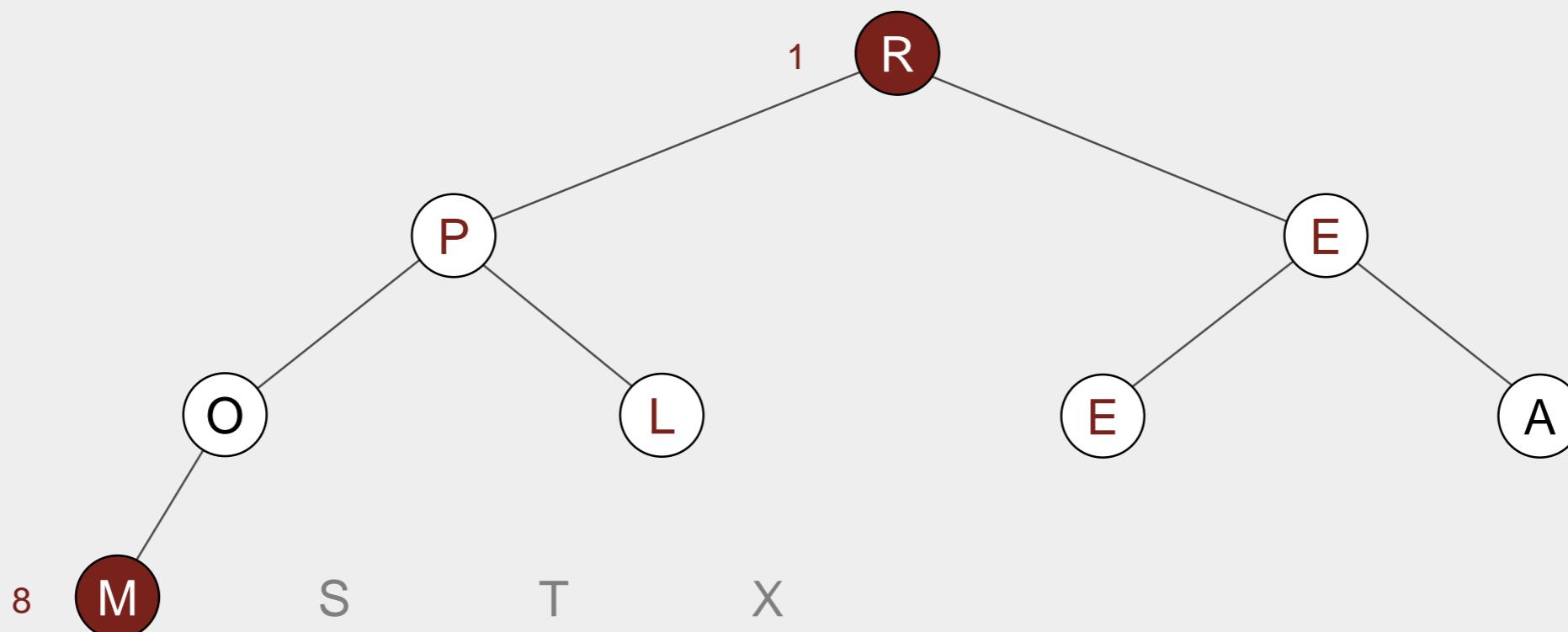
Sortdown. Repeatedly delete the largest remaining item.



R	P	E	O	L	E	A	M	S	T	X
---	---	---	---	---	---	---	---	---	---	---

Sortdown. Repeatedly delete the largest remaining item.

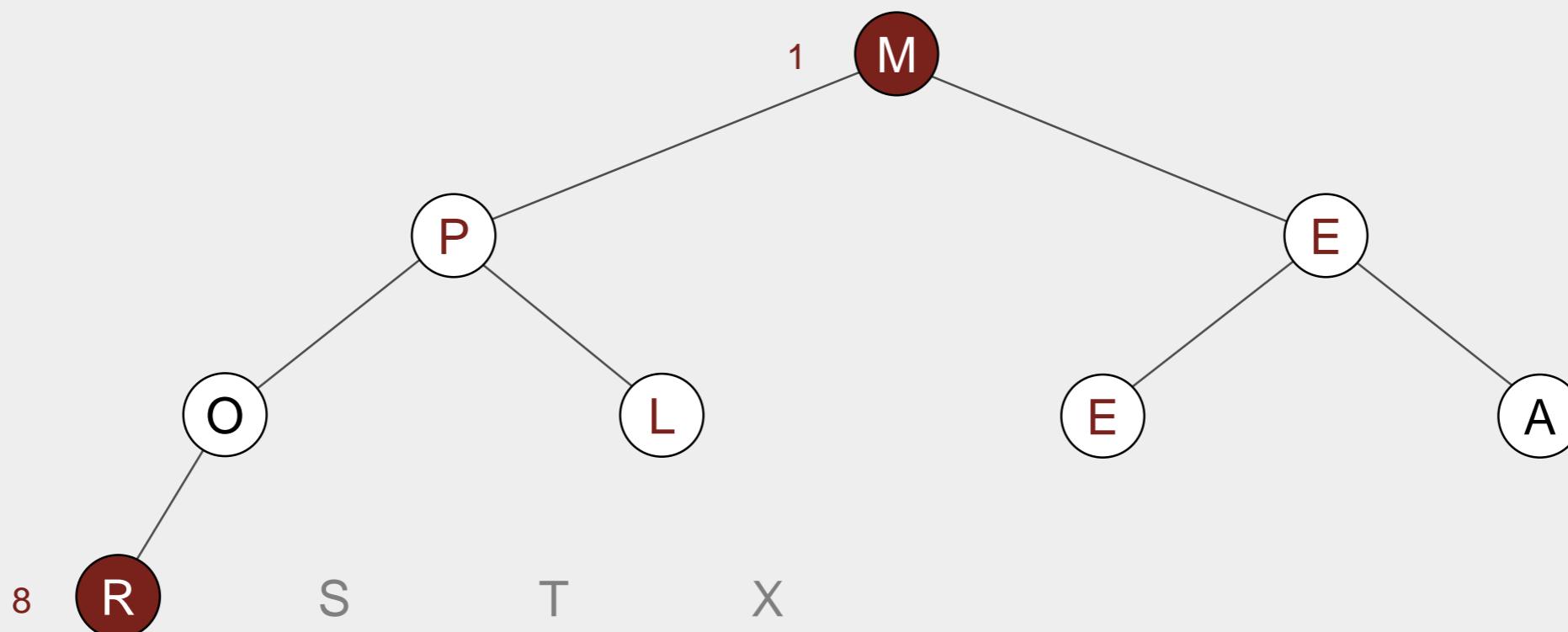
exchange 1 and 8



R	P	E	O	L	E	A	M	S	T	X
1							8			

Sortdown. Repeatedly delete the largest remaining item.

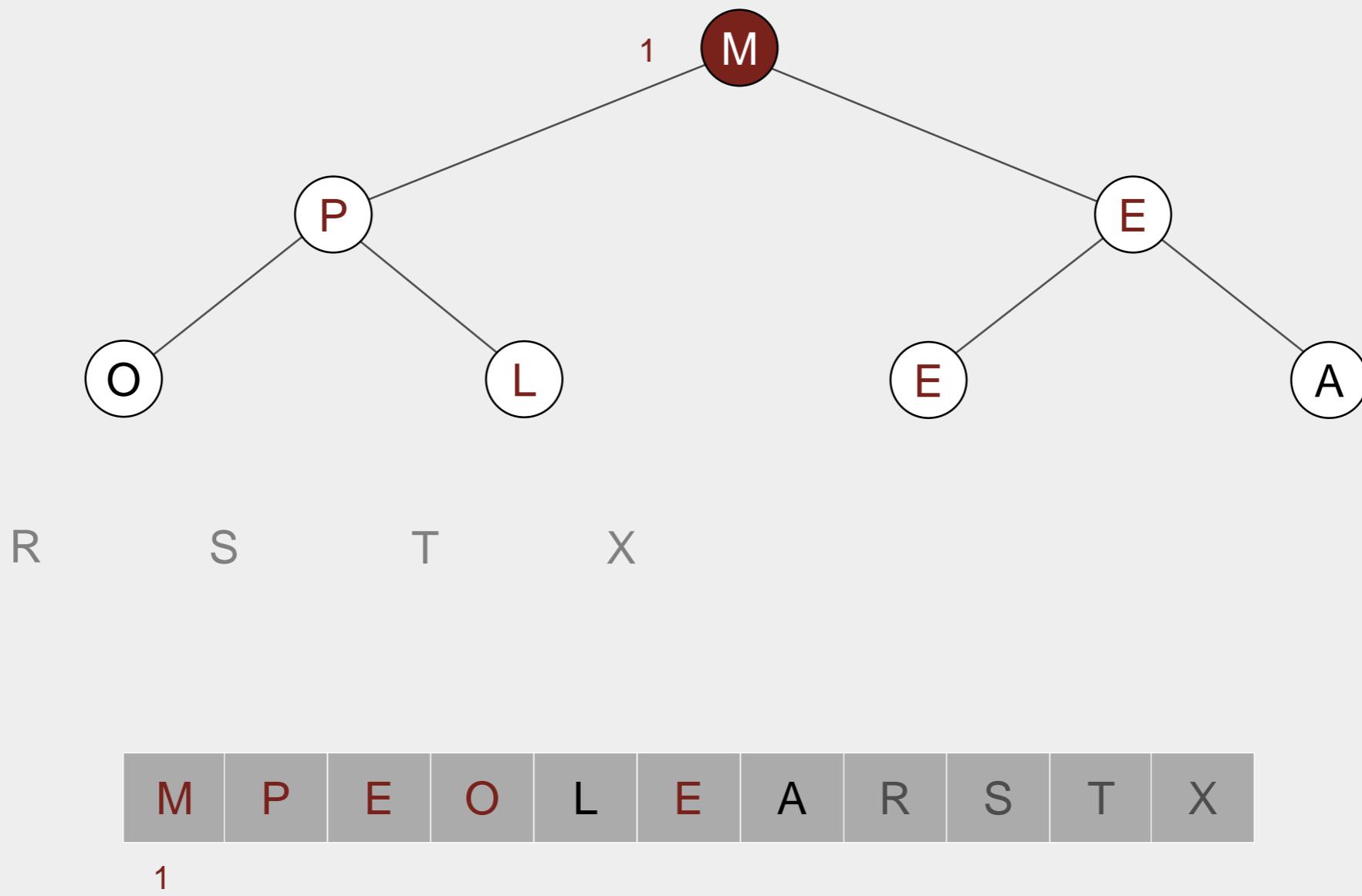
exchange 1 and 8



M	P	E	O	L	E	A	R	S	T	X
1							8			

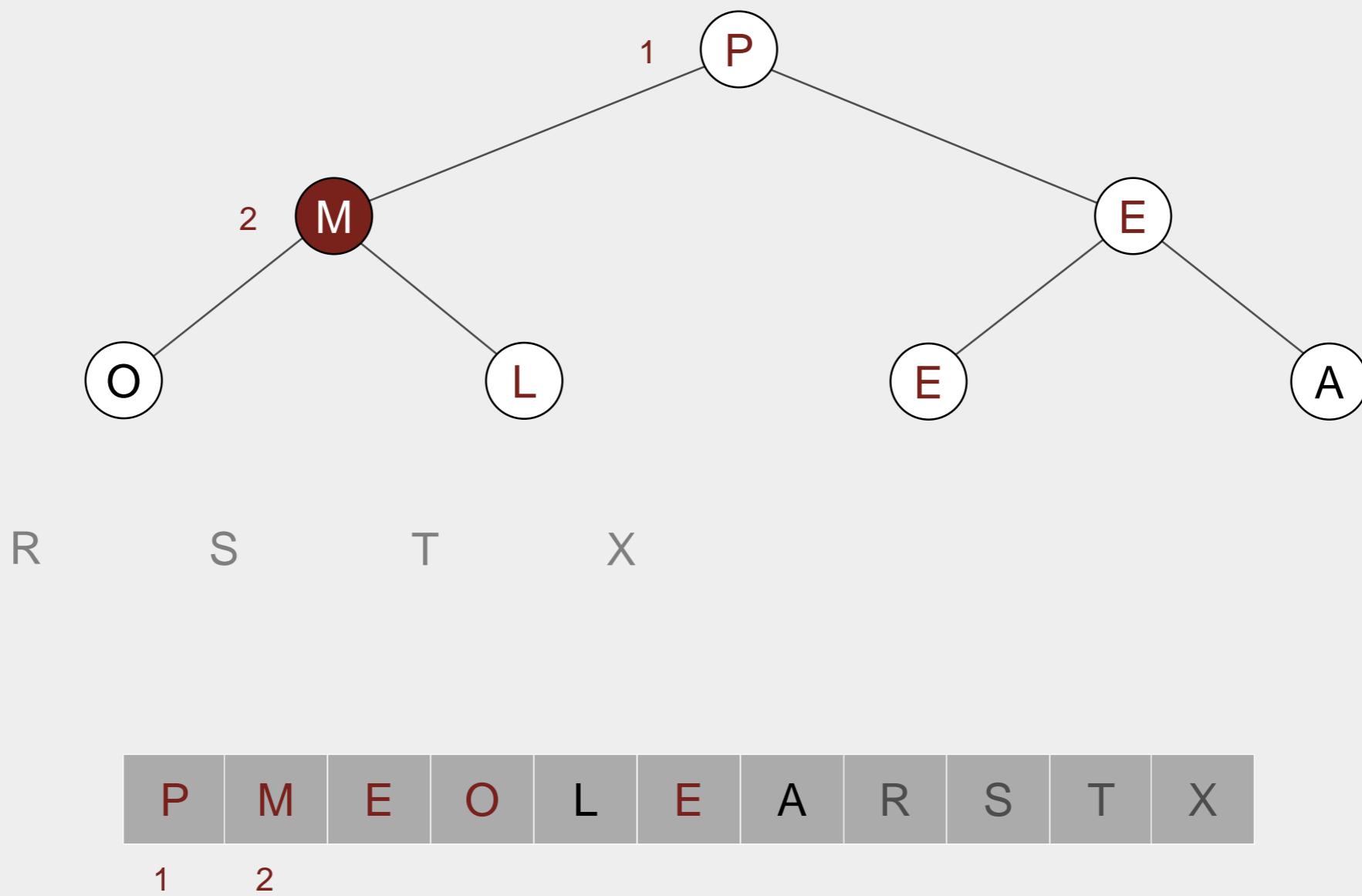
Sortdown. Repeatedly delete the largest remaining item.

sink 1



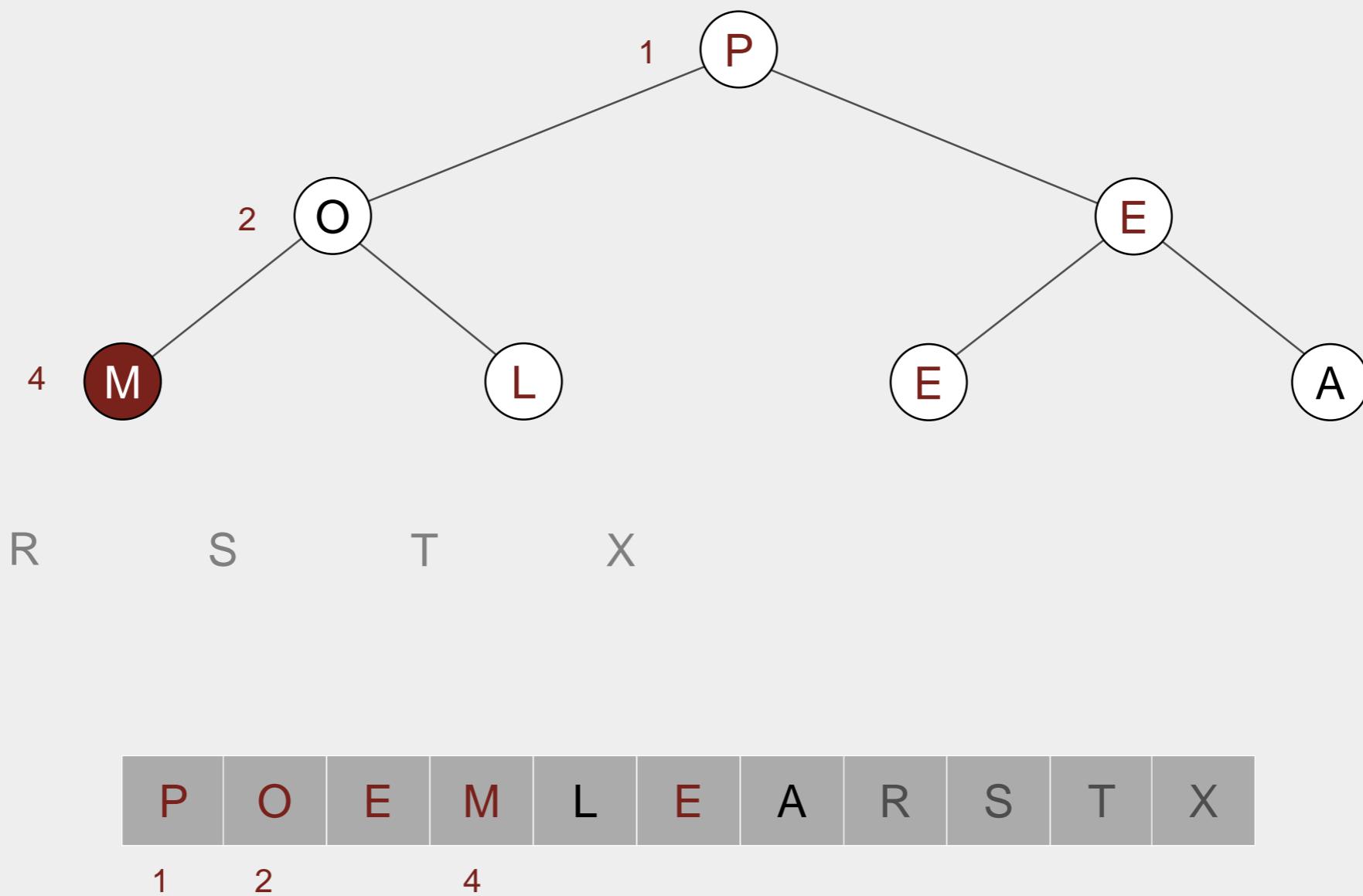
Sortdown. Repeatedly delete the largest remaining item.

sink 1

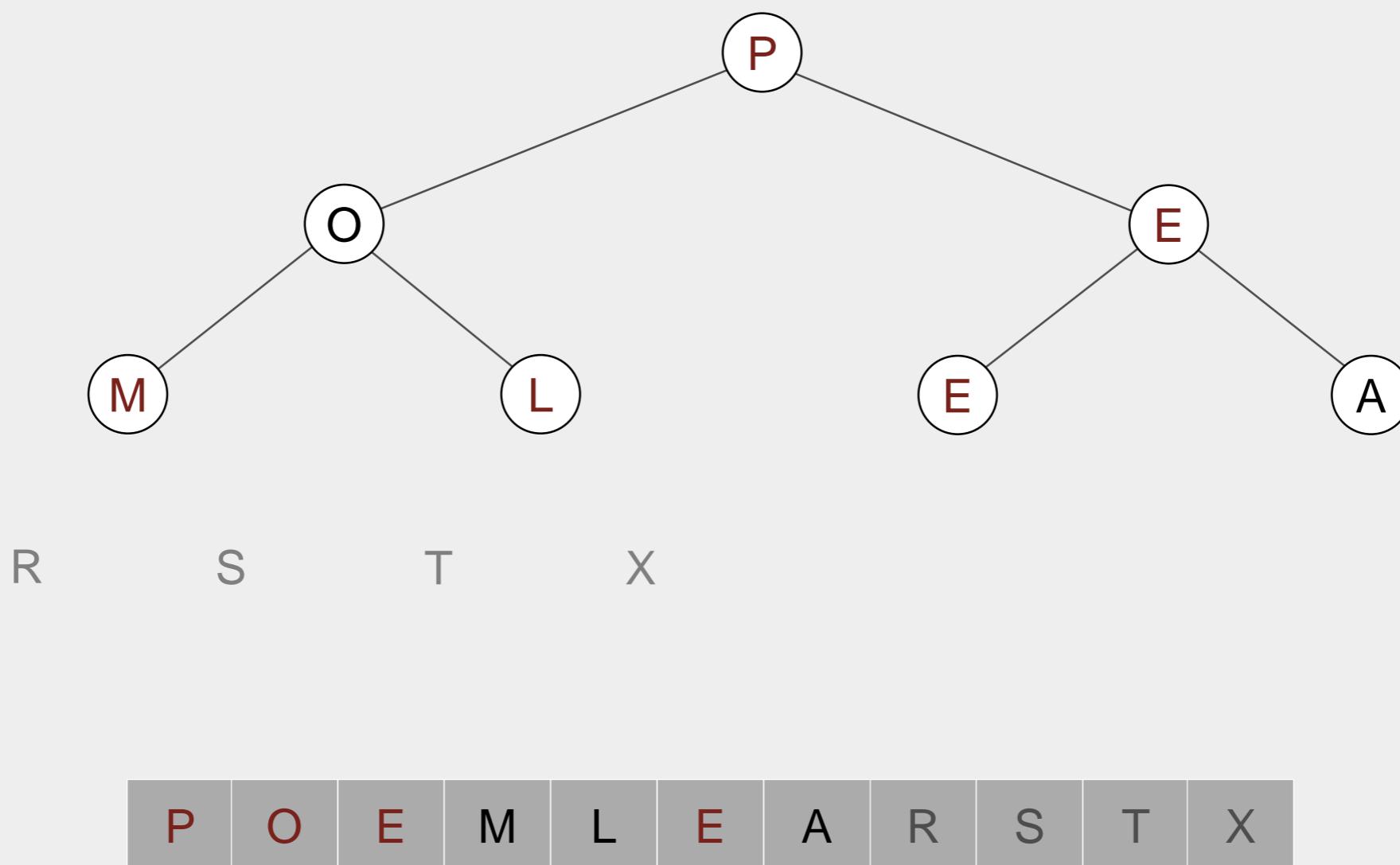


Sortdown. Repeatedly delete the largest remaining item.

sink 1

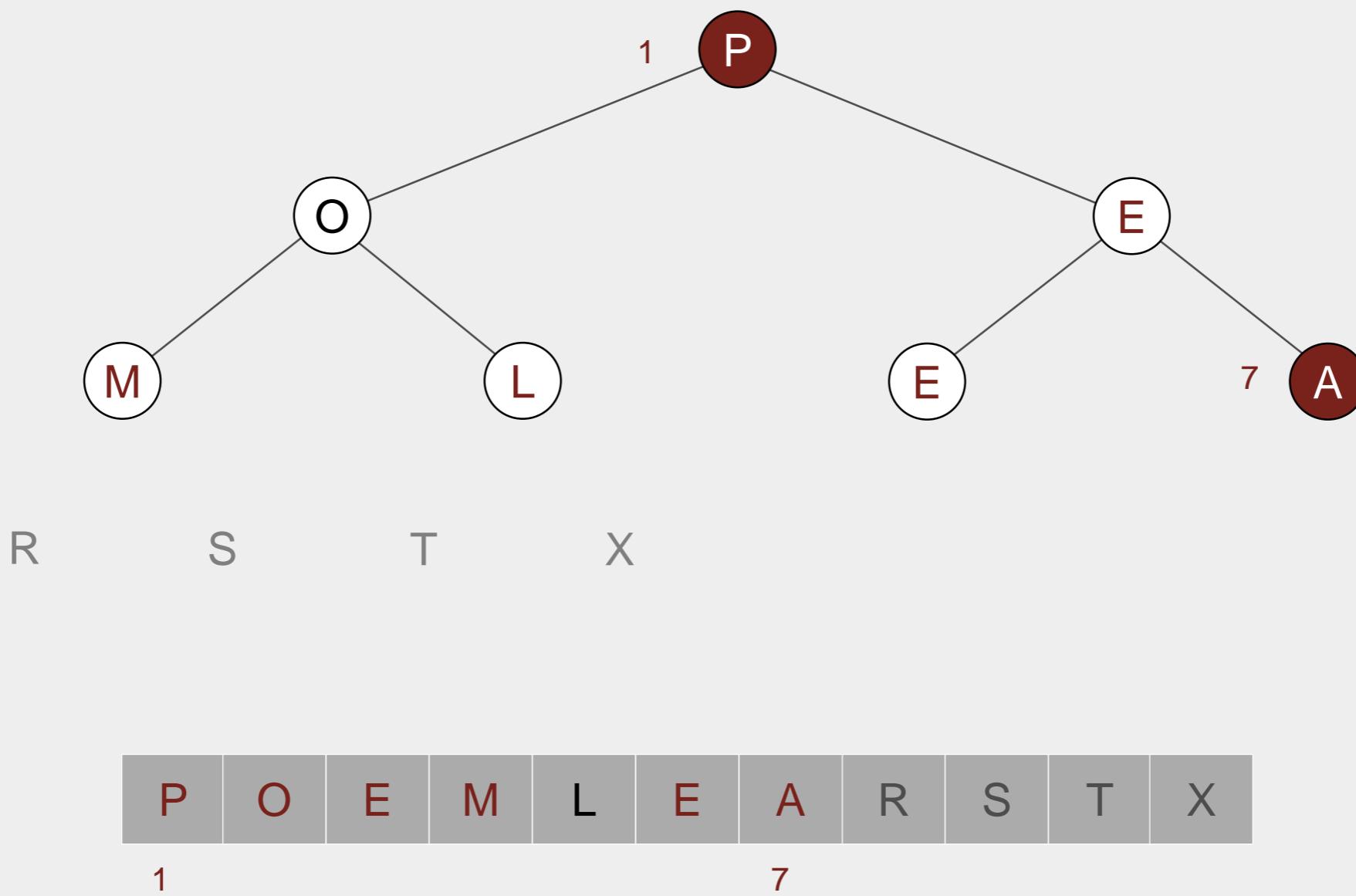


Sortdown. Repeatedly delete the largest remaining item.



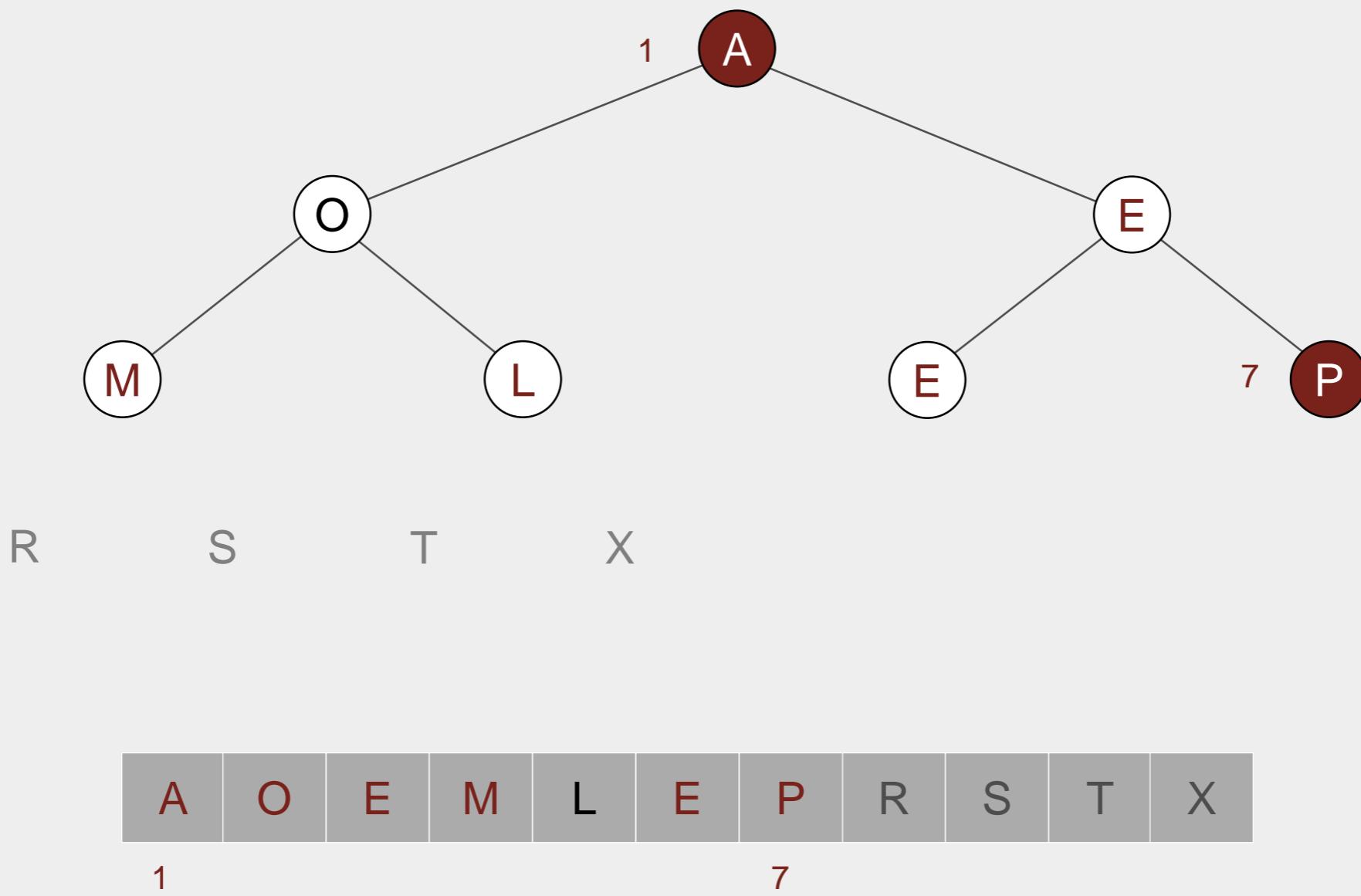
Sortdown. Repeatedly delete the largest remaining item.

exchange 1 and 7



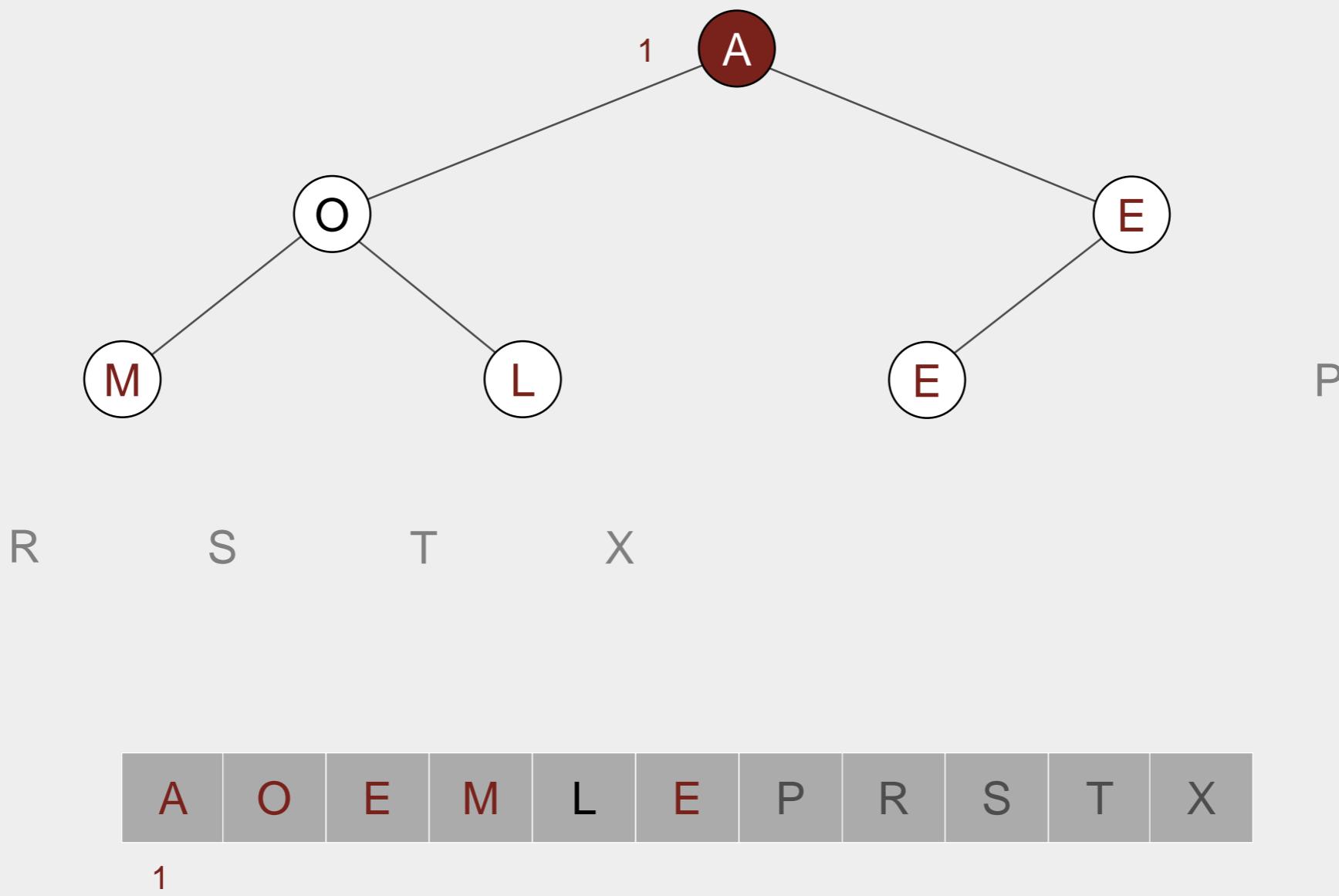
Sortdown. Repeatedly delete the largest remaining item.

exchange 1 and 7



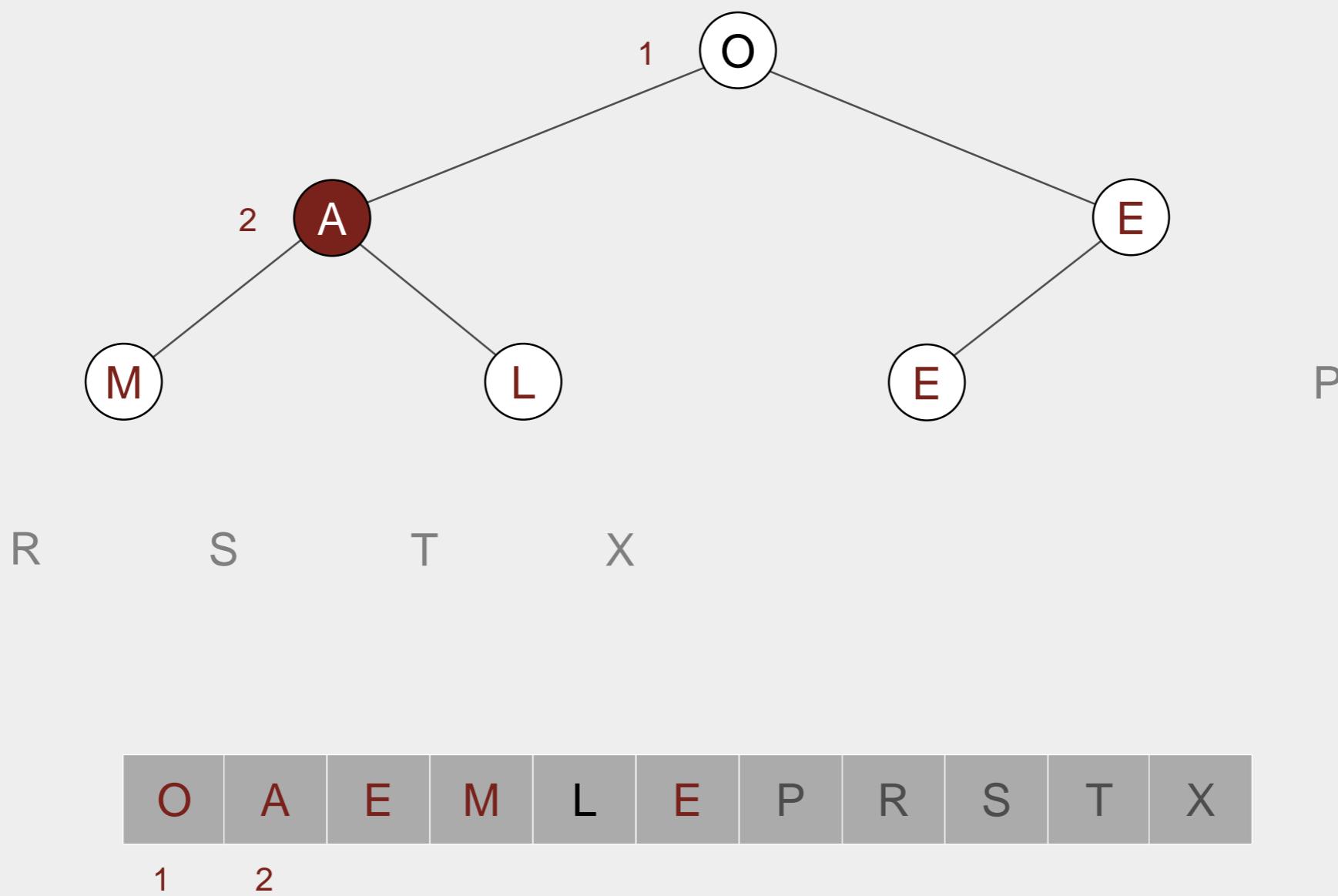
Sortdown. Repeatedly delete the largest remaining item.

sink 1



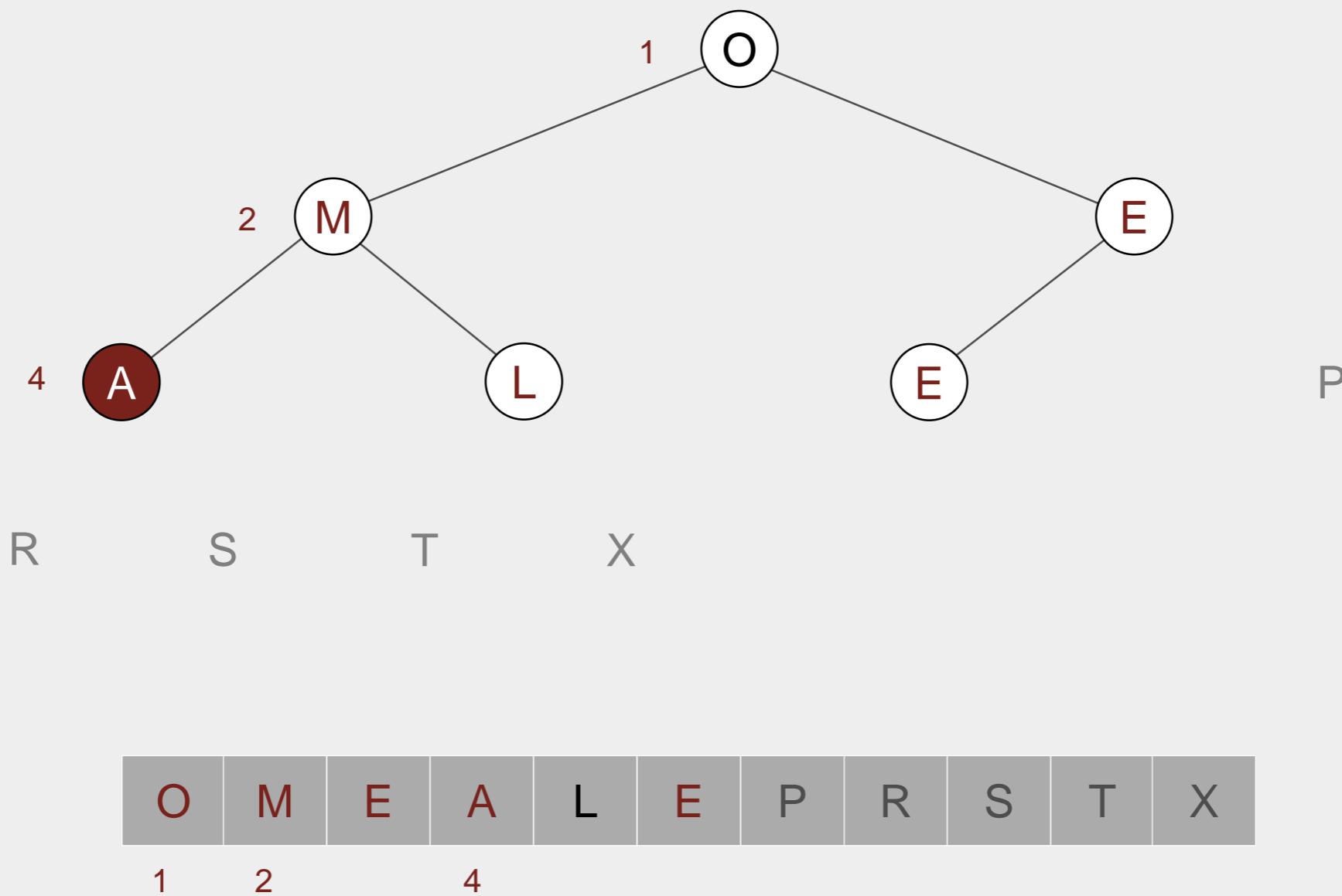
Sortdown. Repeatedly delete the largest remaining item.

sink 1



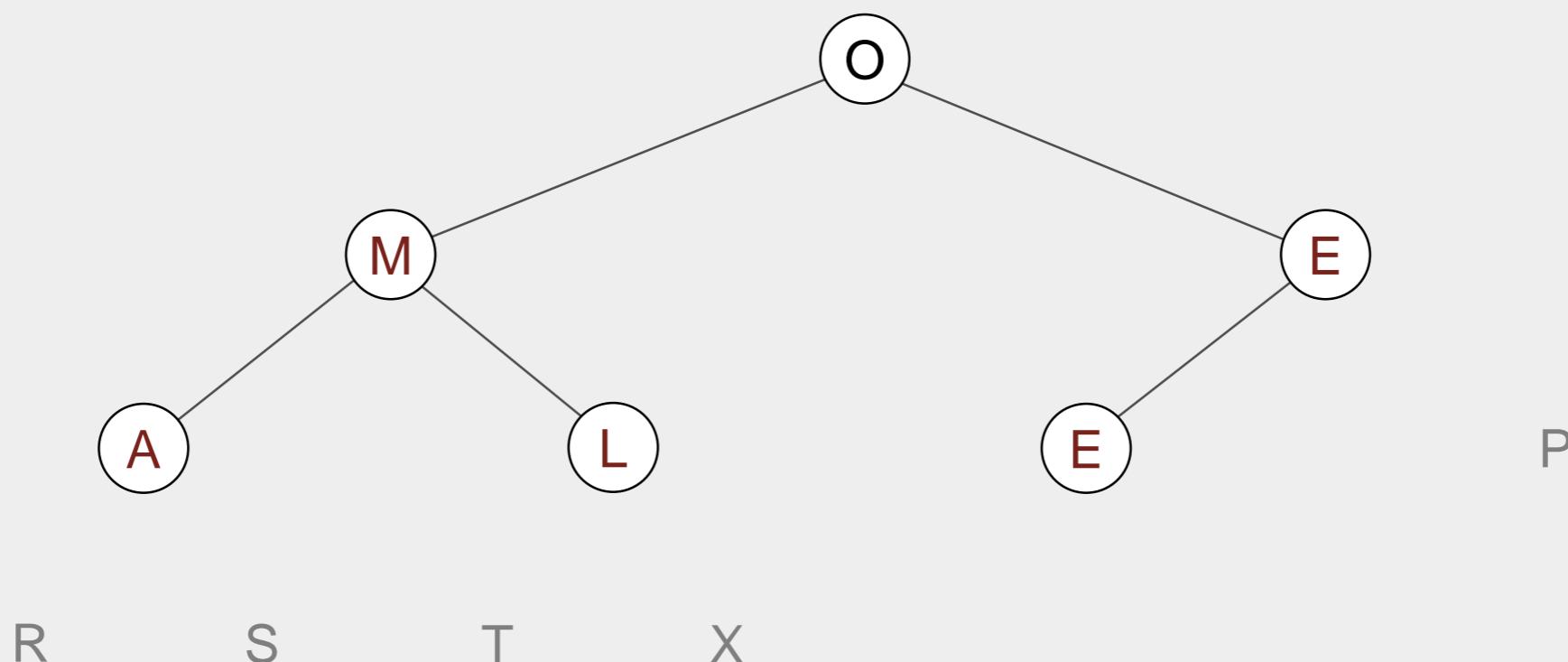
Sortdown. Repeatedly delete the largest remaining item.

sink 1



Sortdown. Repeatedly delete the largest remaining item.

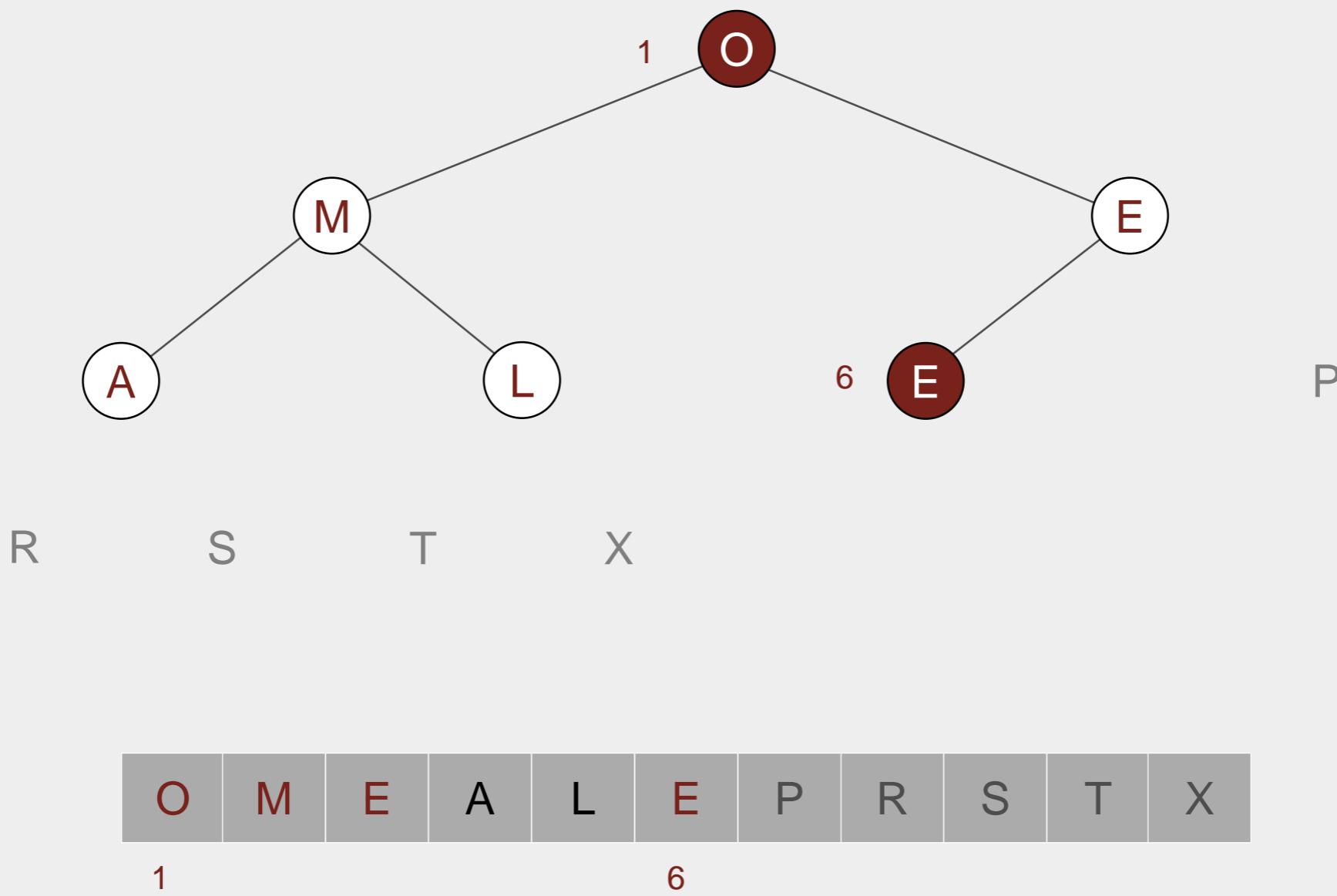
sink 1



O	M	E	A	L	E	P	R	S	T	X
---	---	---	---	---	---	---	---	---	---	---

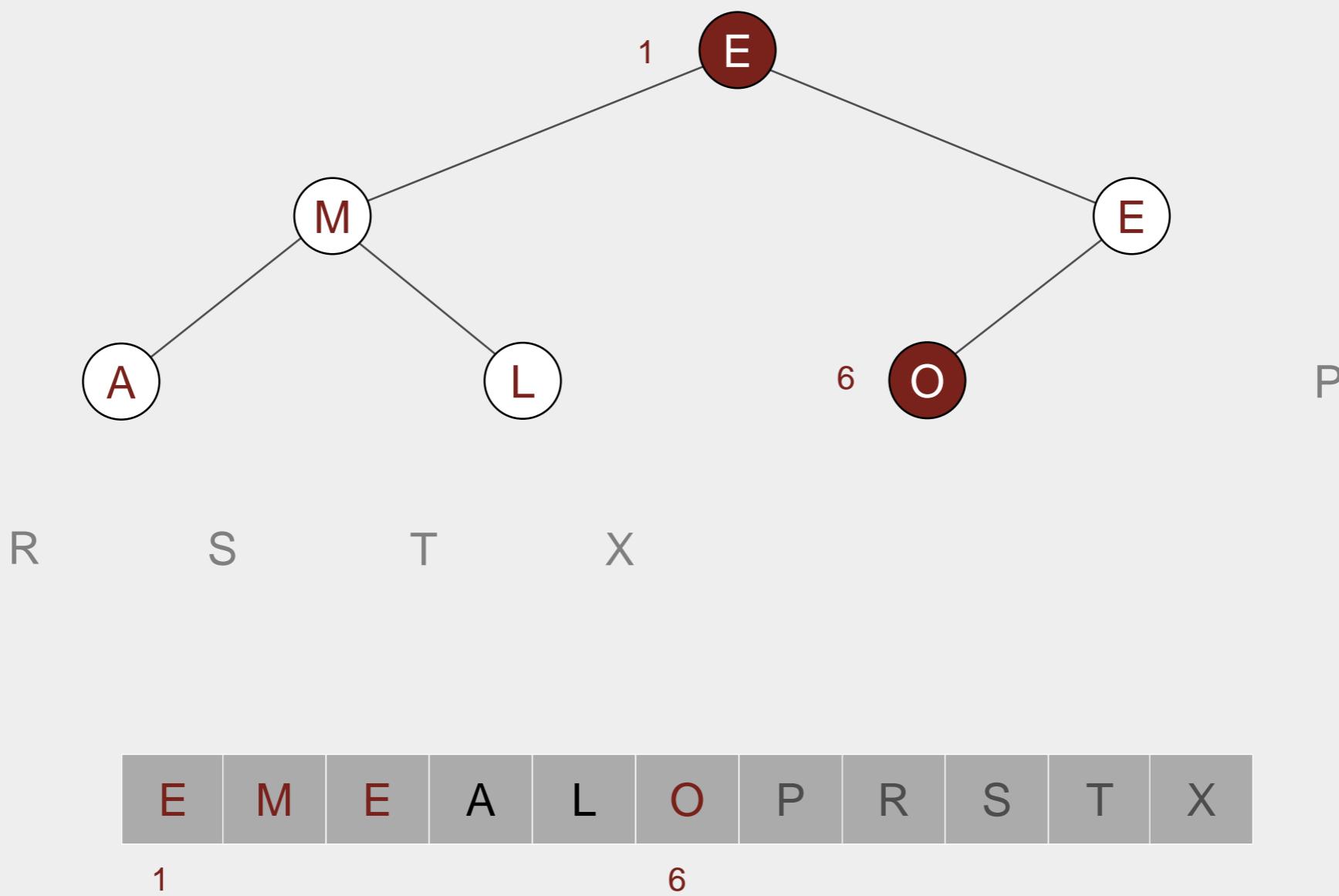
Sortdown. Repeatedly delete the largest remaining item.

exchange 1 and 6



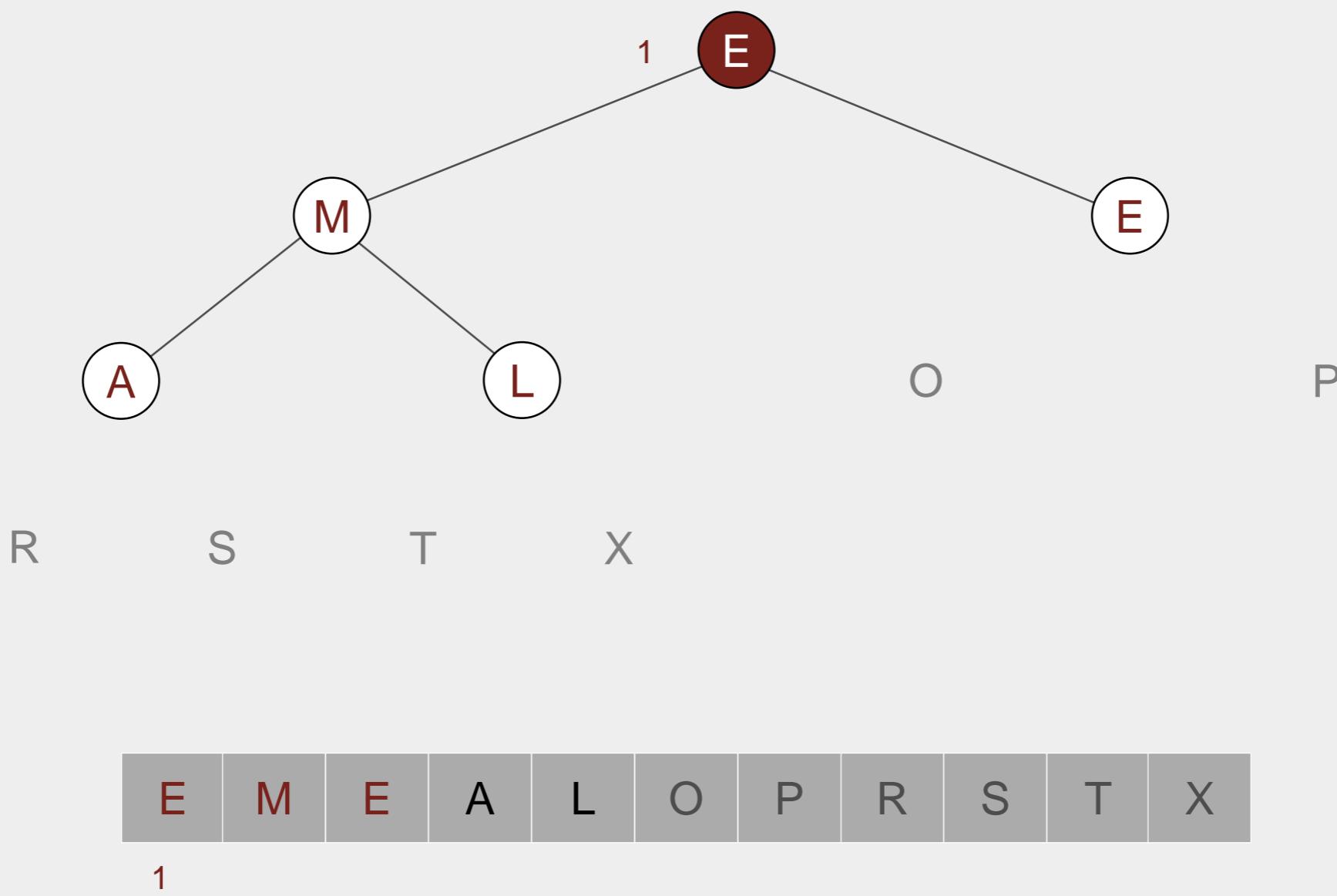
Sortdown. Repeatedly delete the largest remaining item.

exchange 1 and 6



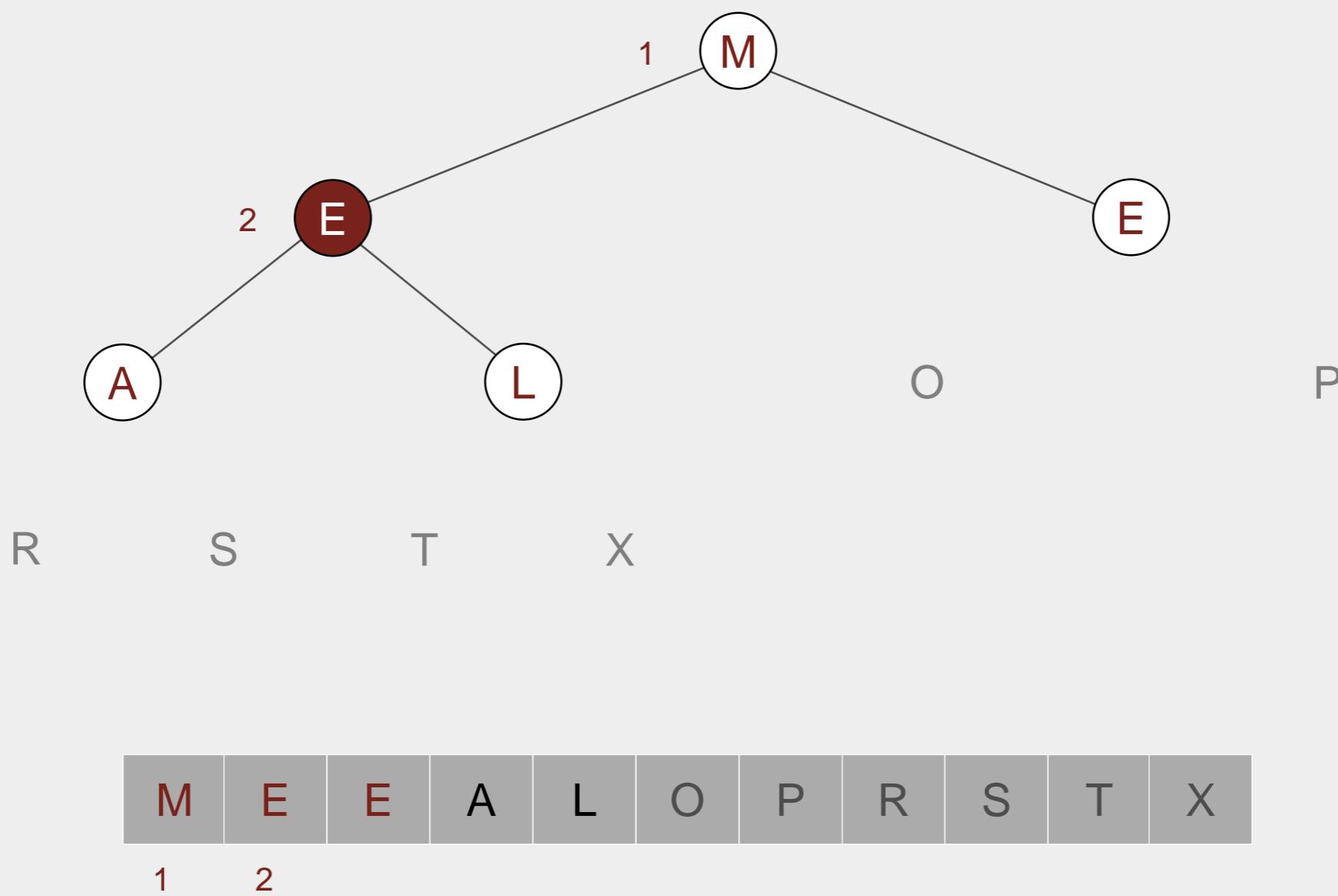
Sortdown. Repeatedly delete the largest remaining item.

sink 1



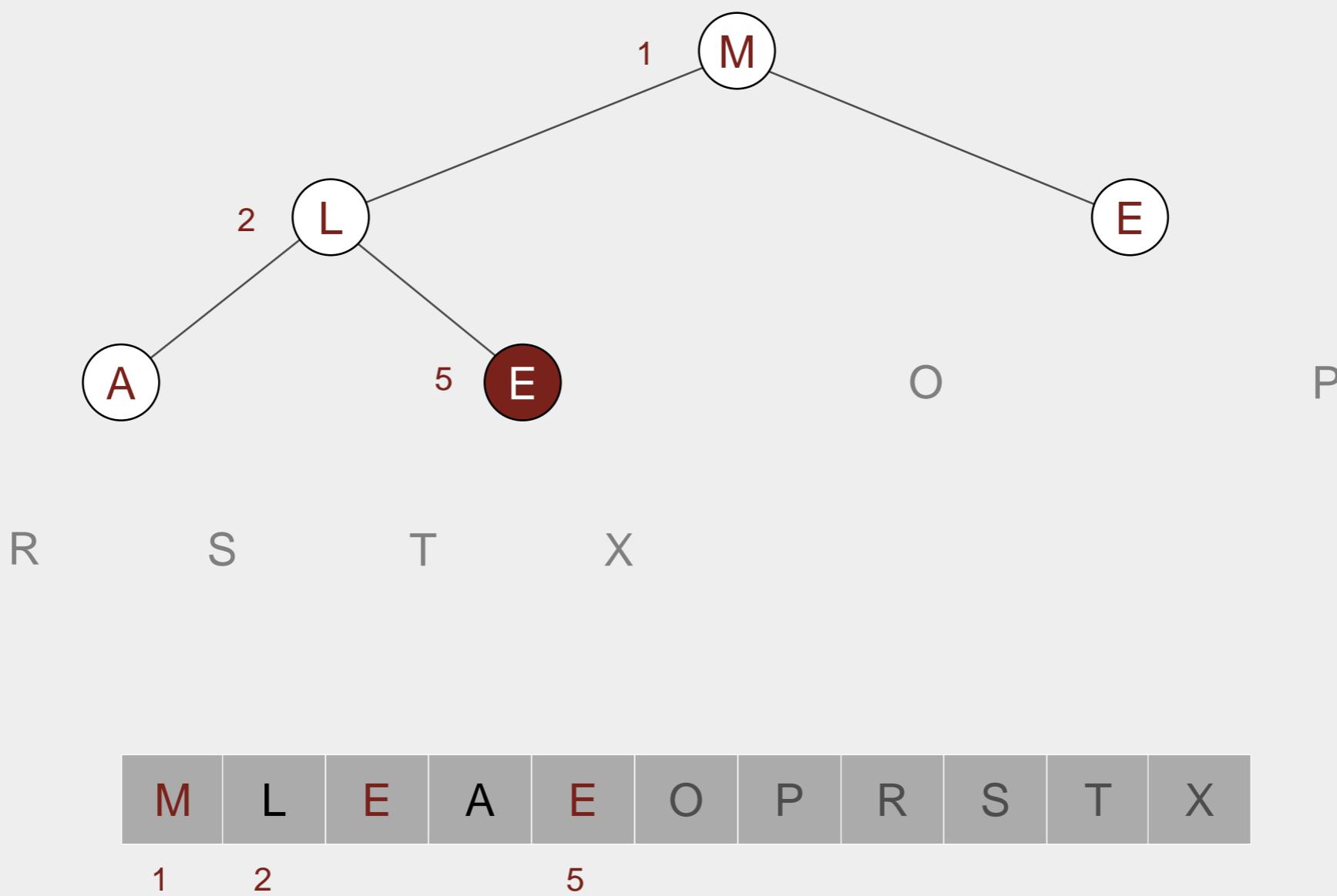
Sortdown. Repeatedly delete the largest remaining item.

sink 1



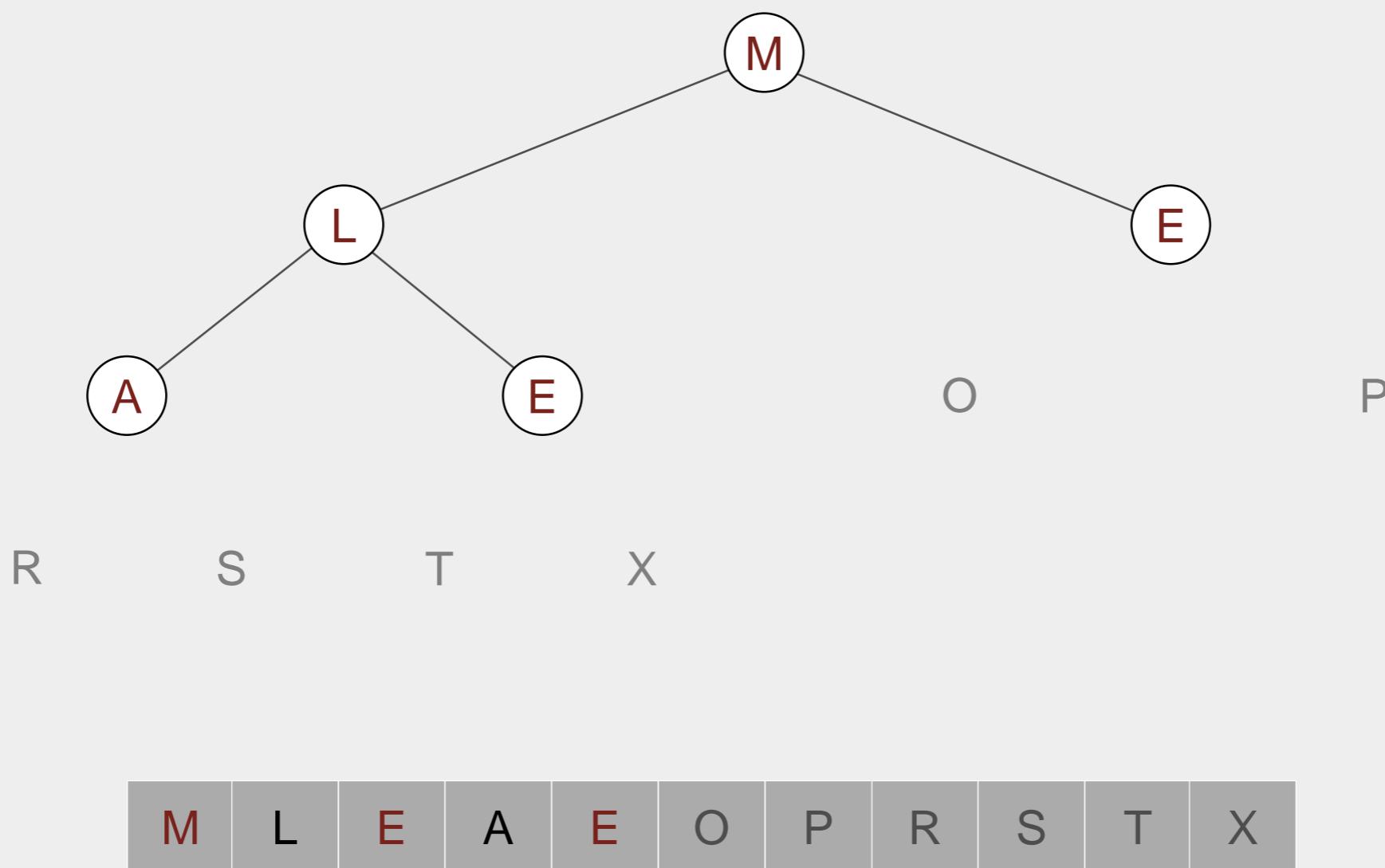
Sortdown. Repeatedly delete the largest remaining item.

sink 1



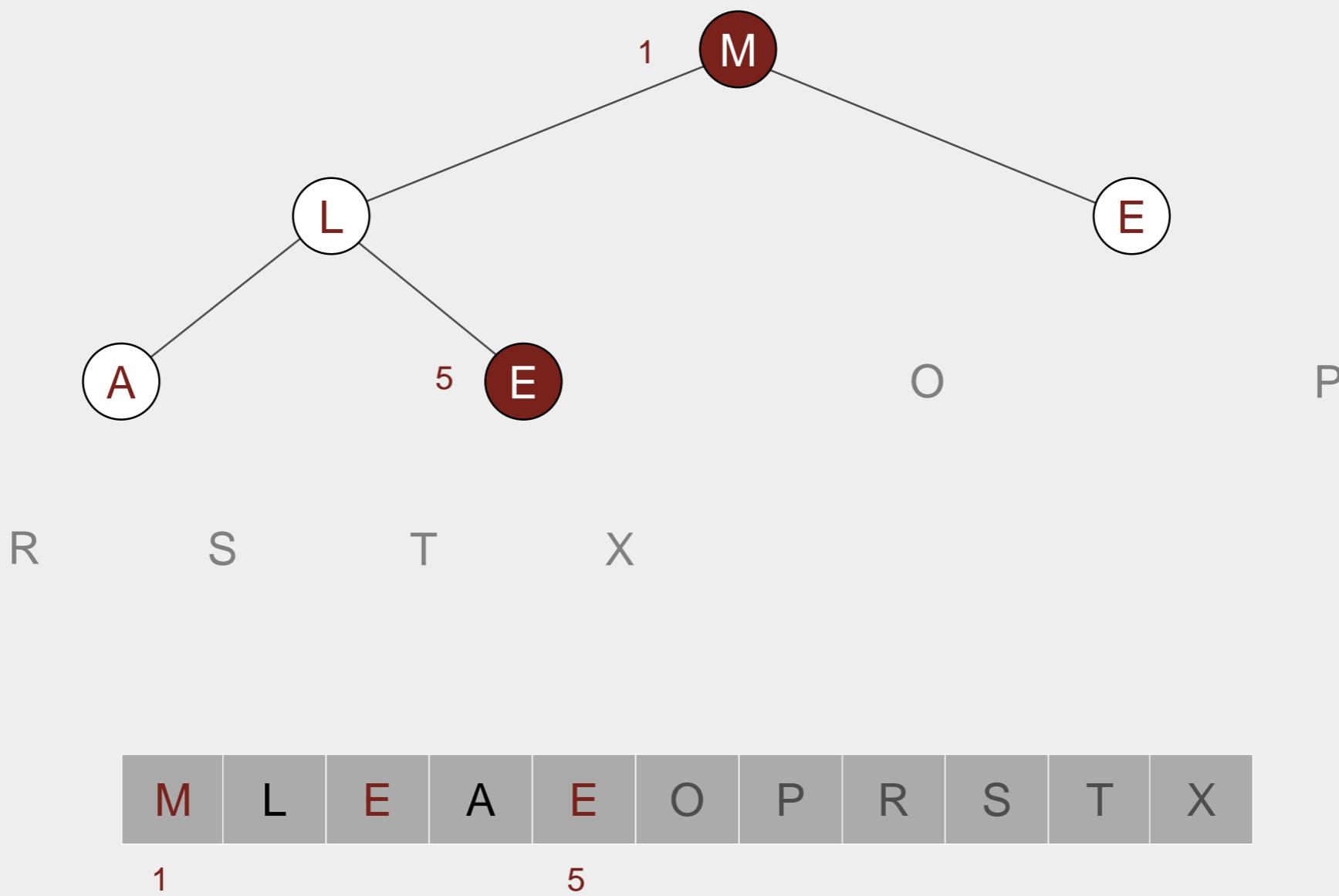
Heapsort

Sortdown. Repeatedly delete the largest remaining item.



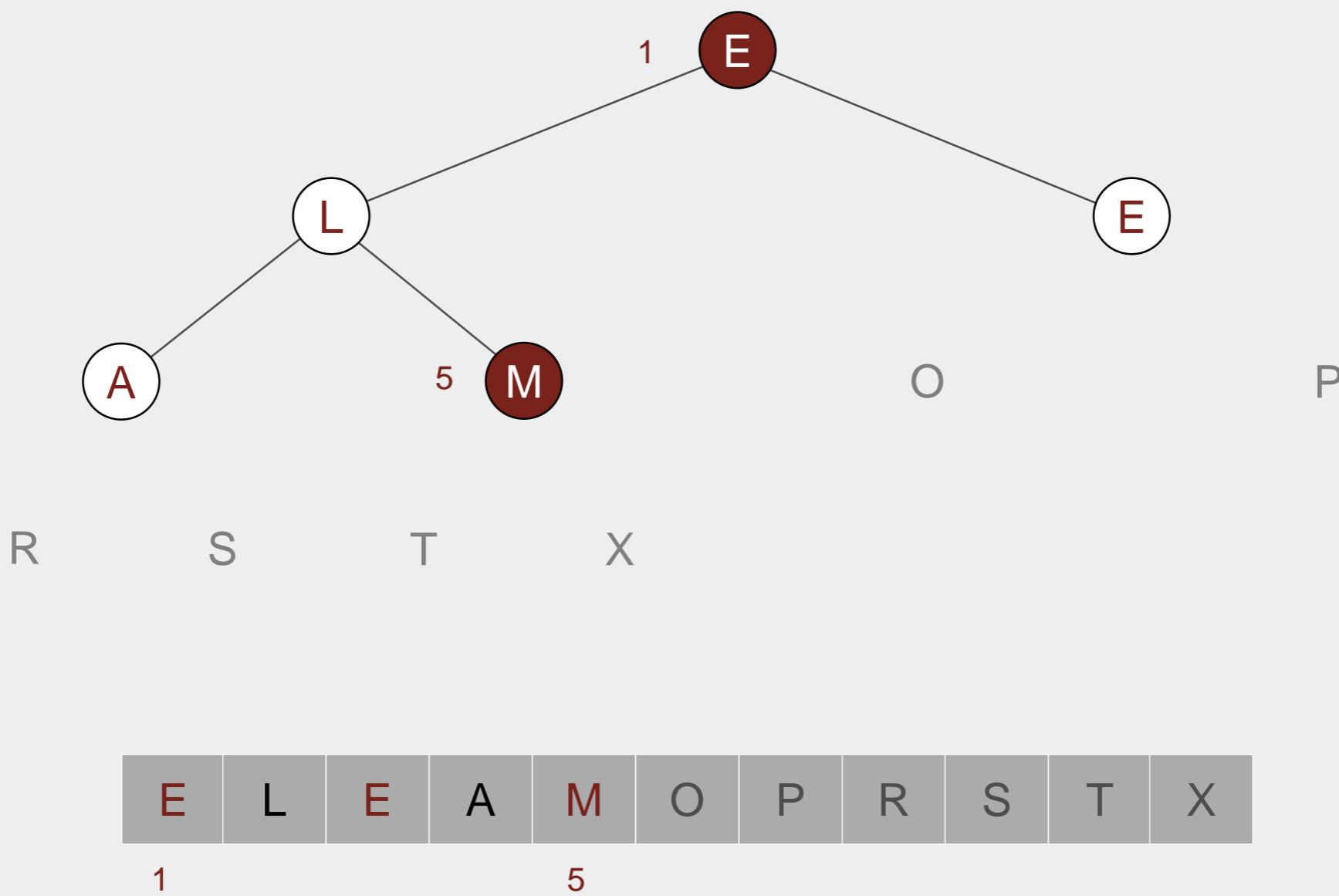
Sortdown. Repeatedly delete the largest remaining item.

exchange 1 and 5



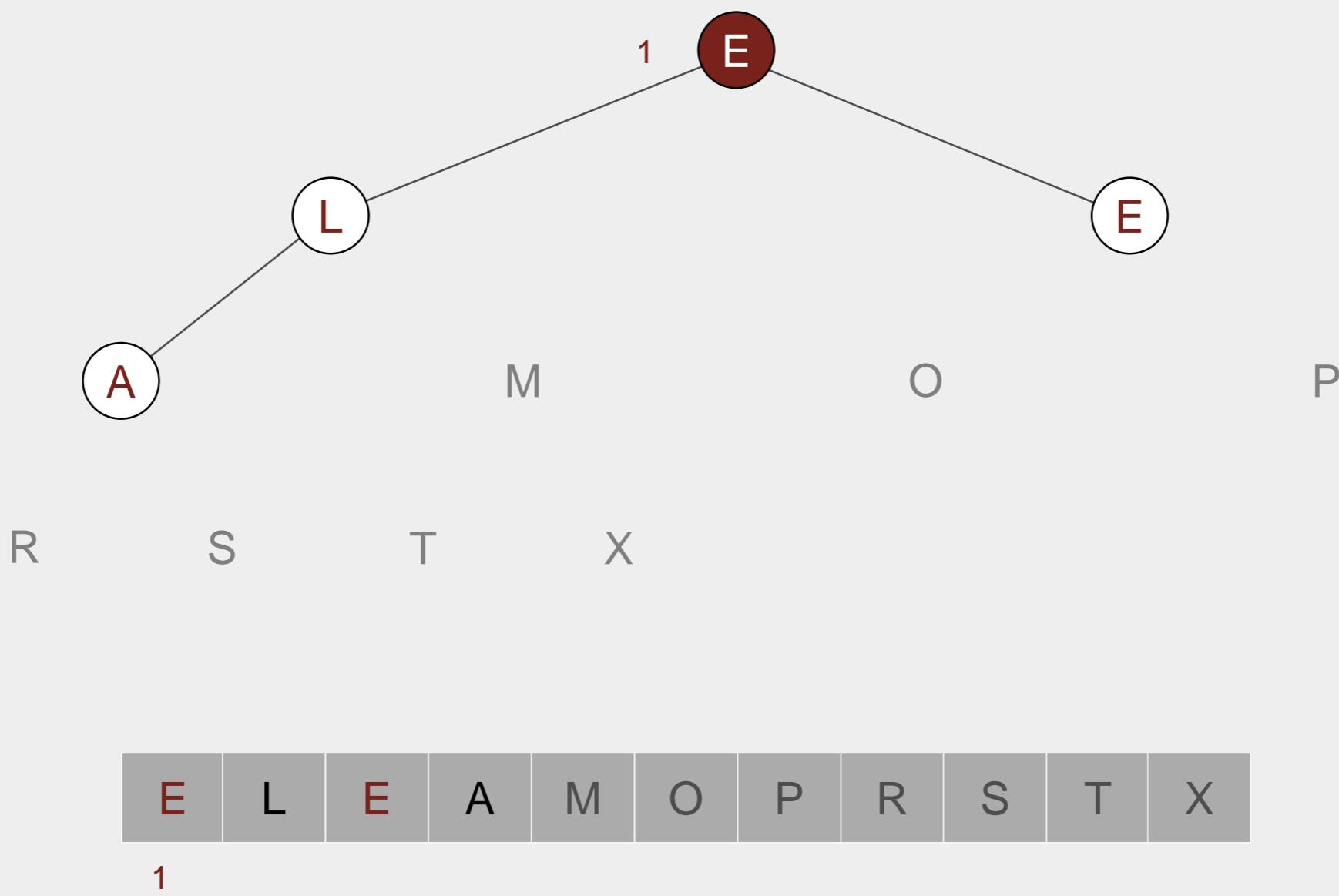
Sortdown. Repeatedly delete the largest remaining item.

exchange 1 and 5



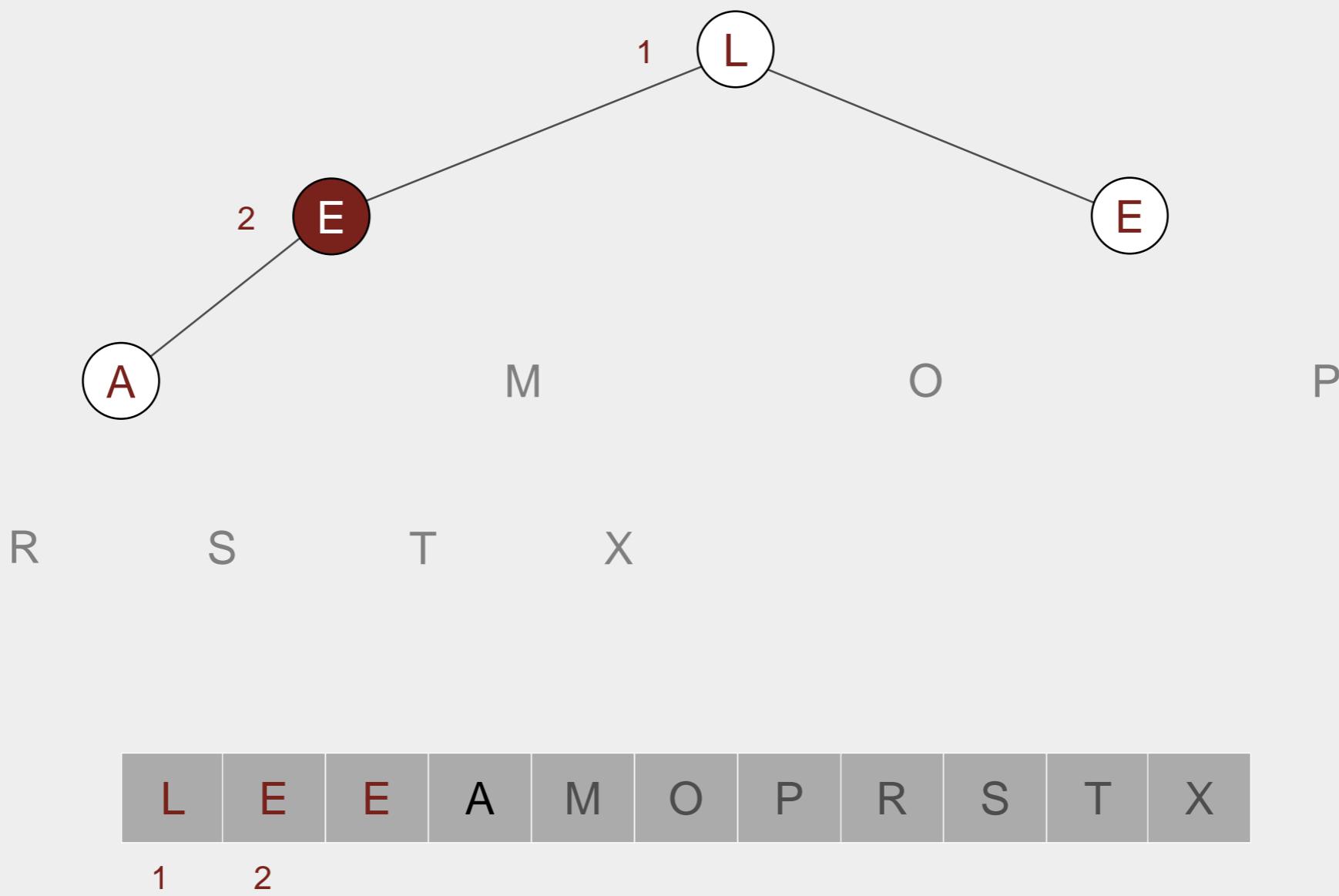
Sortdown. Repeatedly delete the largest remaining item.

sink 1

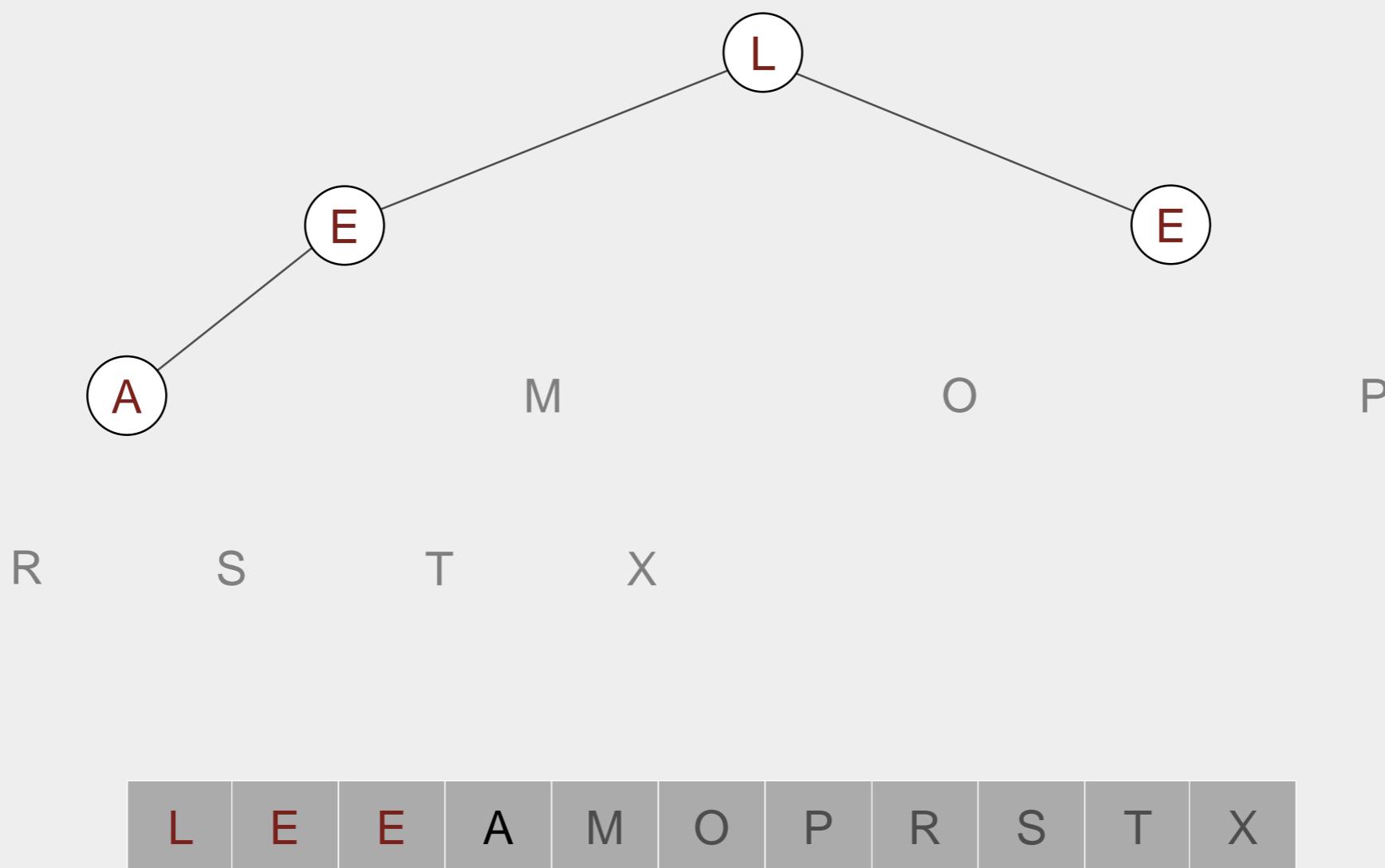


Sortdown. Repeatedly delete the largest remaining item.

sink 1

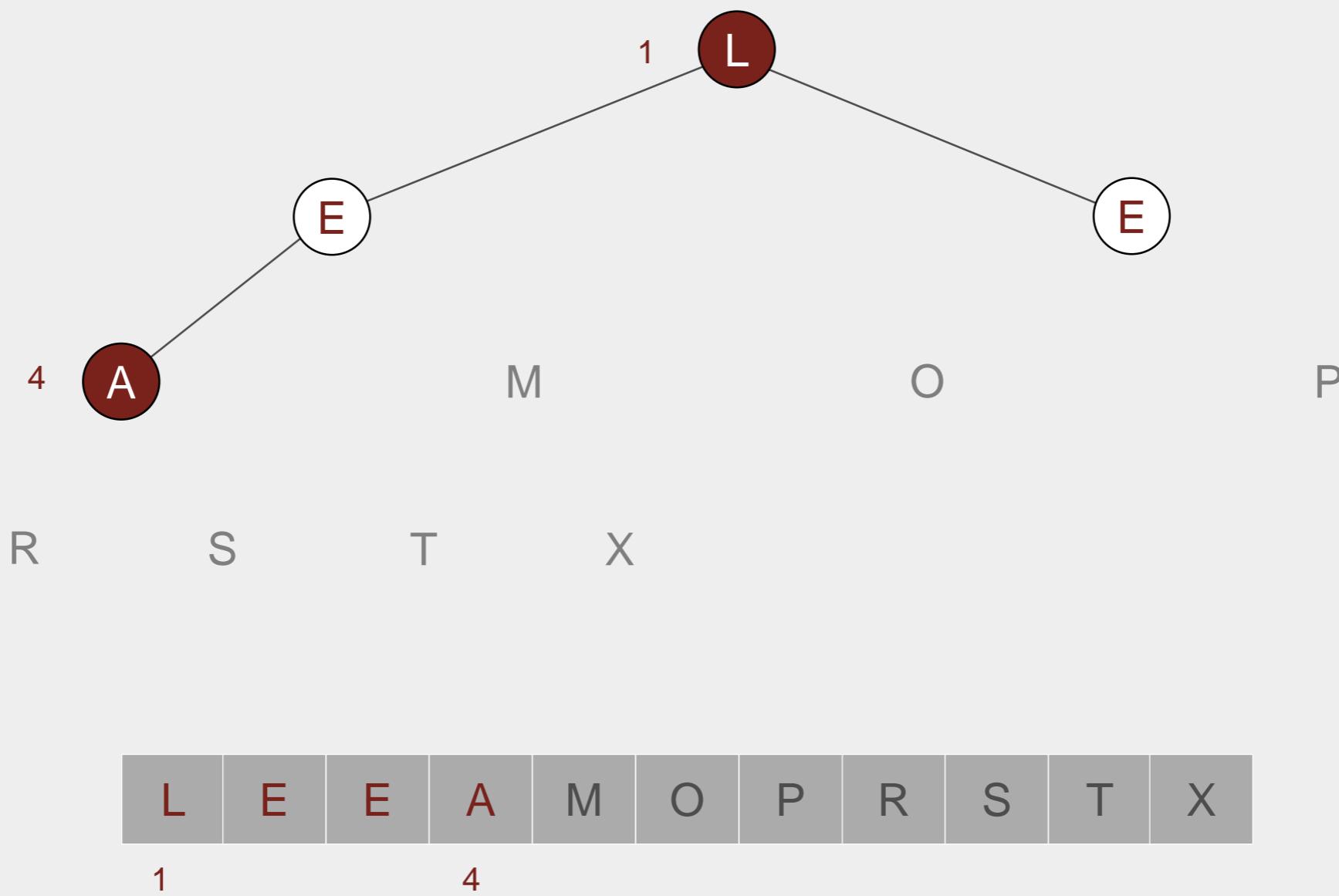


Sortdown. Repeatedly delete the largest remaining item.



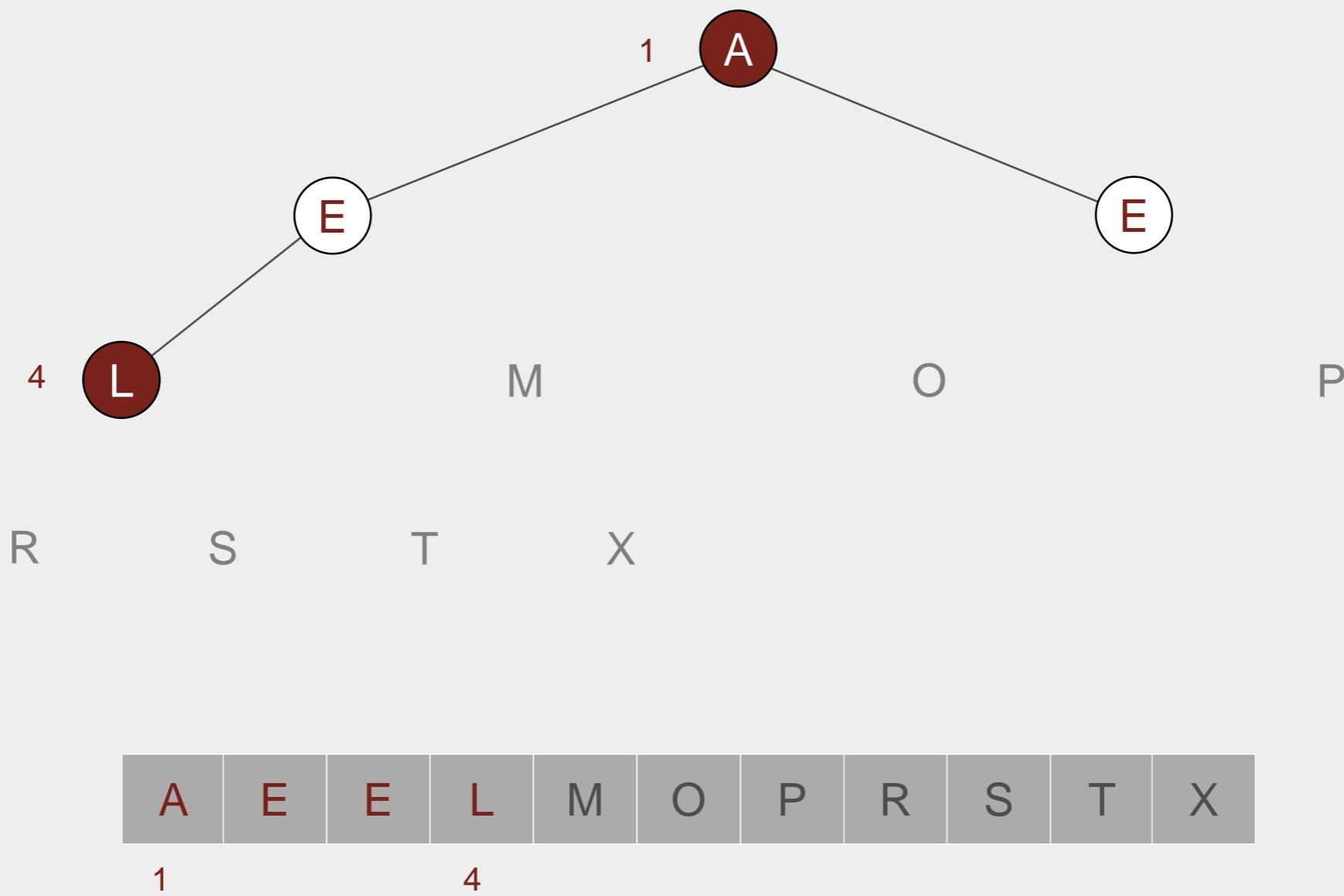
Sortdown. Repeatedly delete the largest remaining item.

exchange 1 and 4



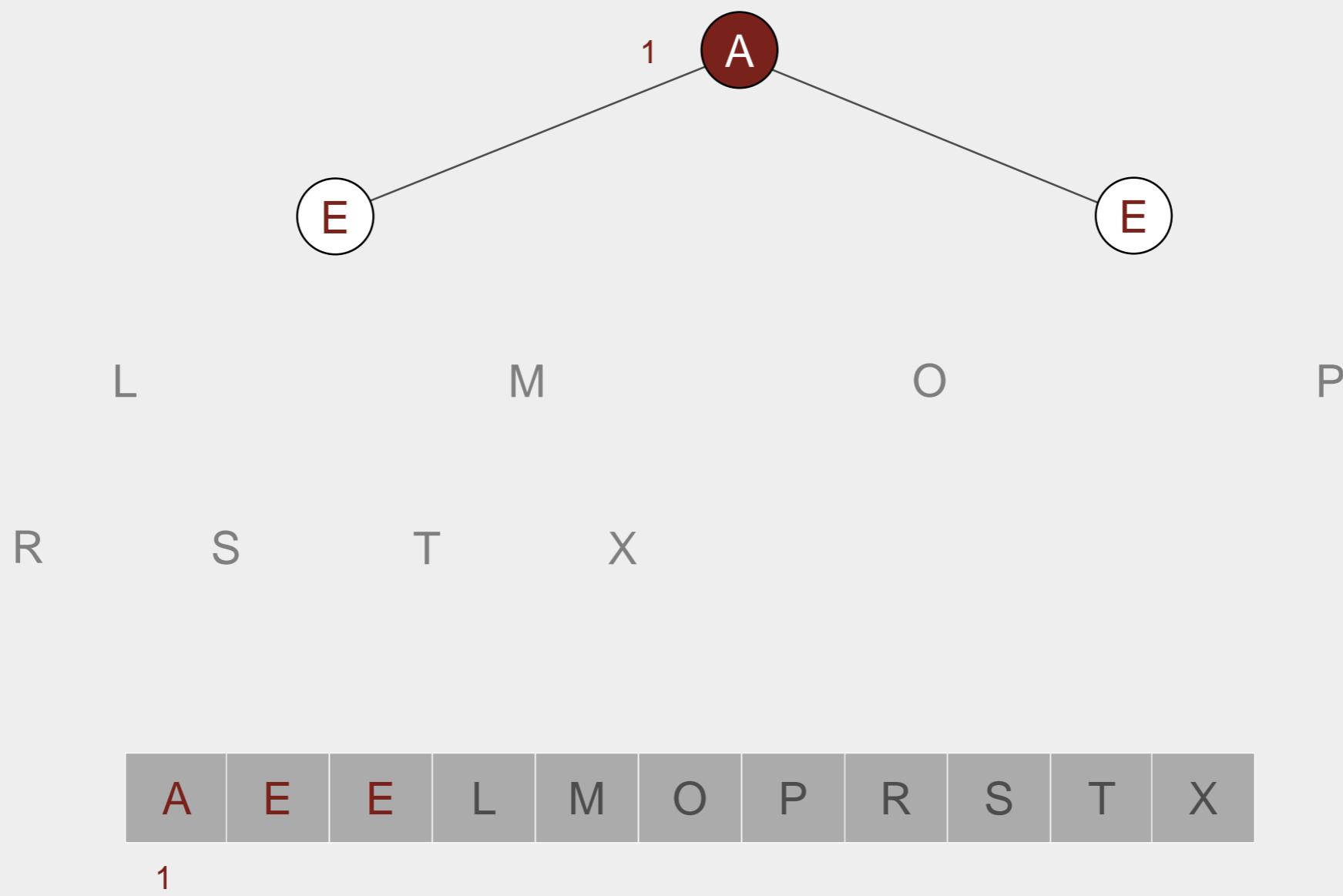
Sortdown. Repeatedly delete the largest remaining item.

exchange 1 and 4



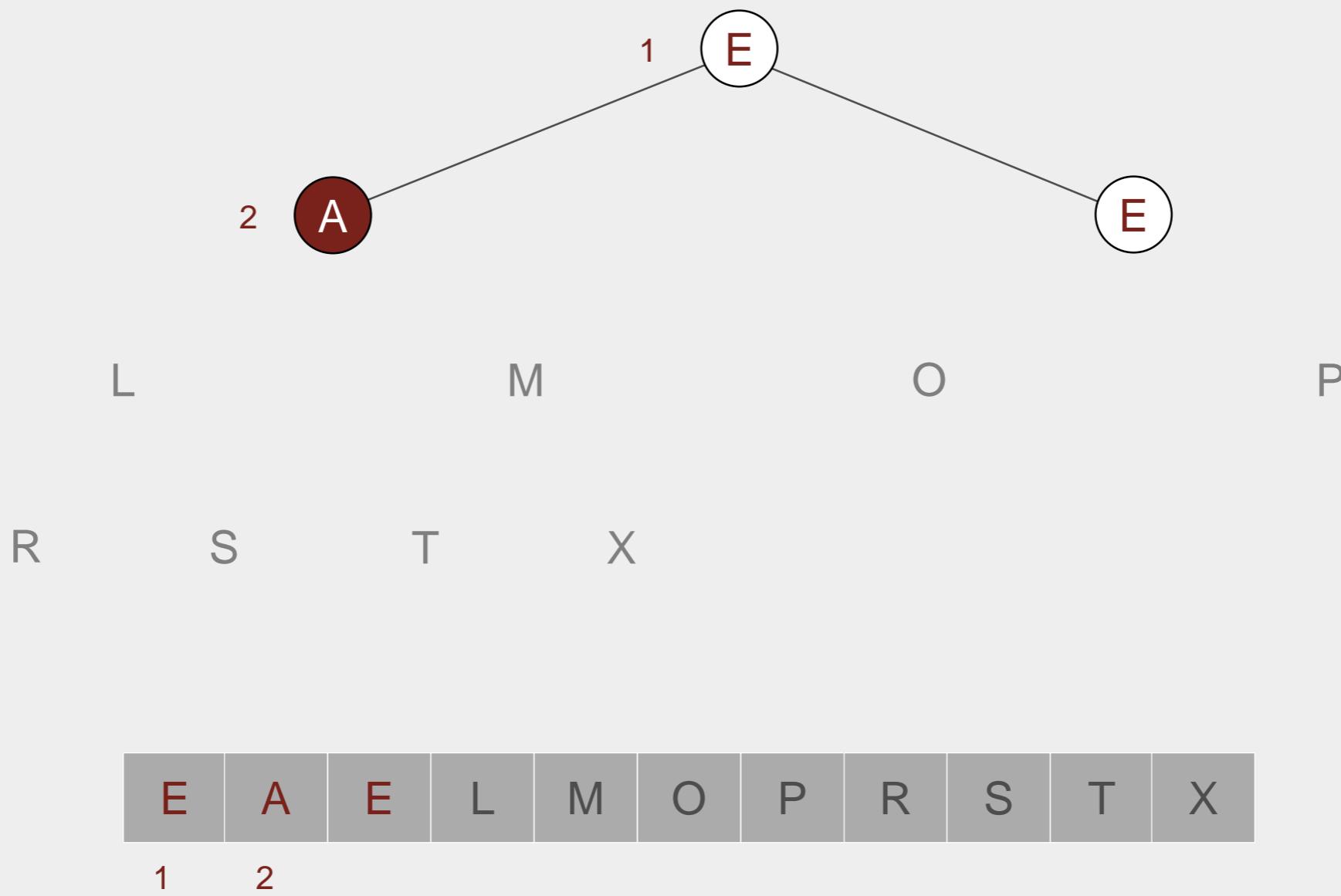
Sortdown. Repeatedly delete the largest remaining item.

sink 1



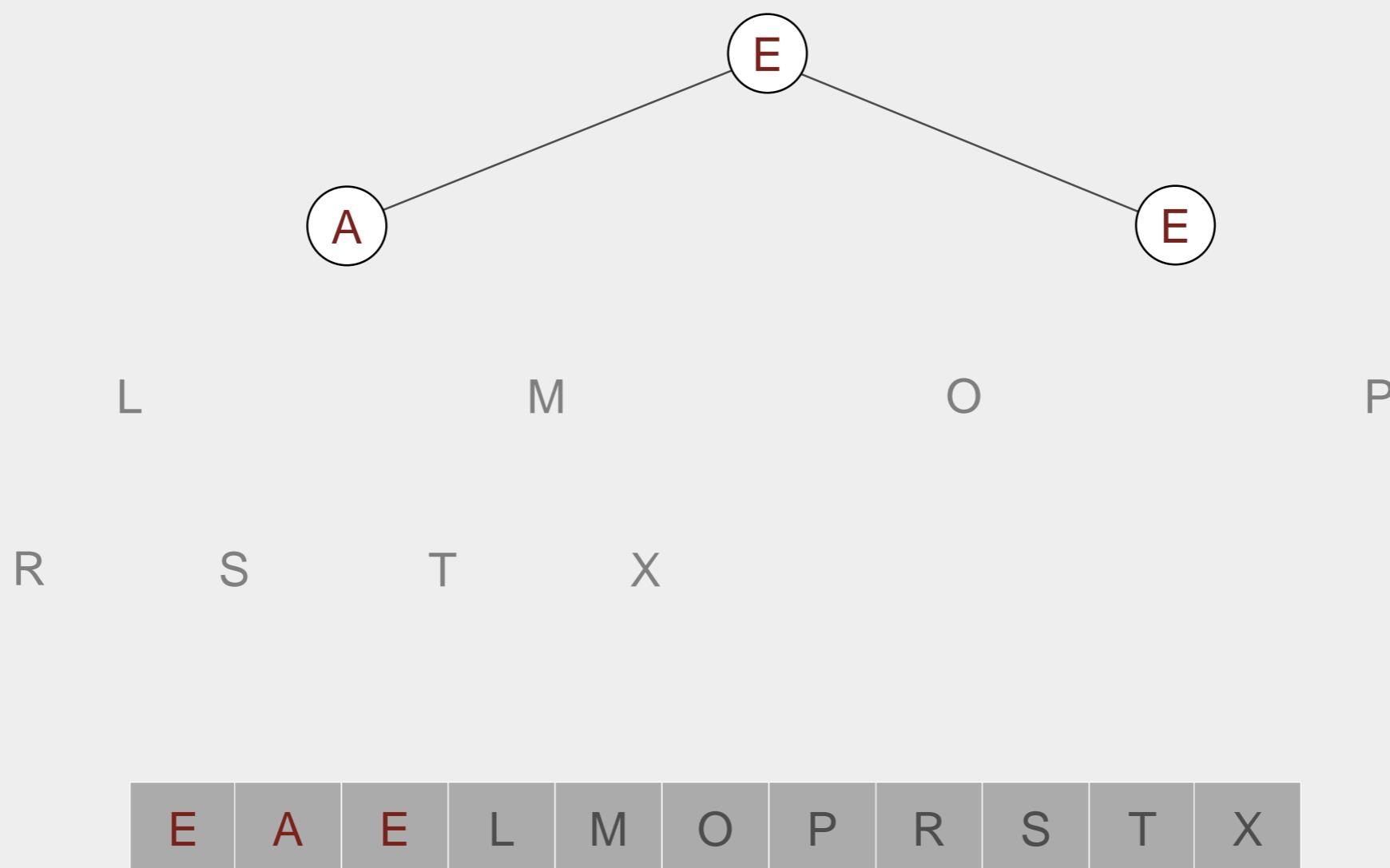
Sortdown. Repeatedly delete the largest remaining item.

sink 1



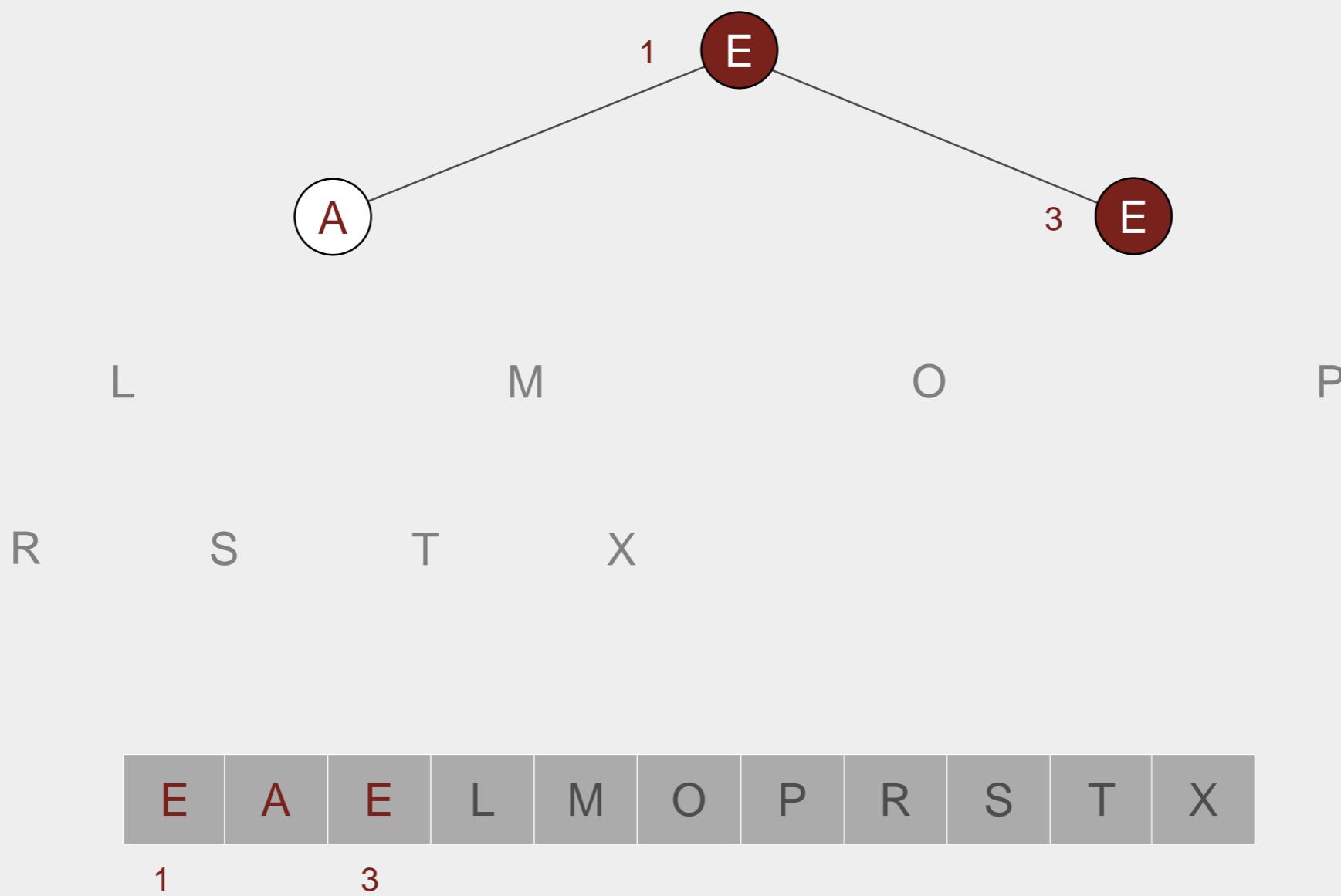
Heapsort

Sortdown. Repeatedly delete the largest remaining item.



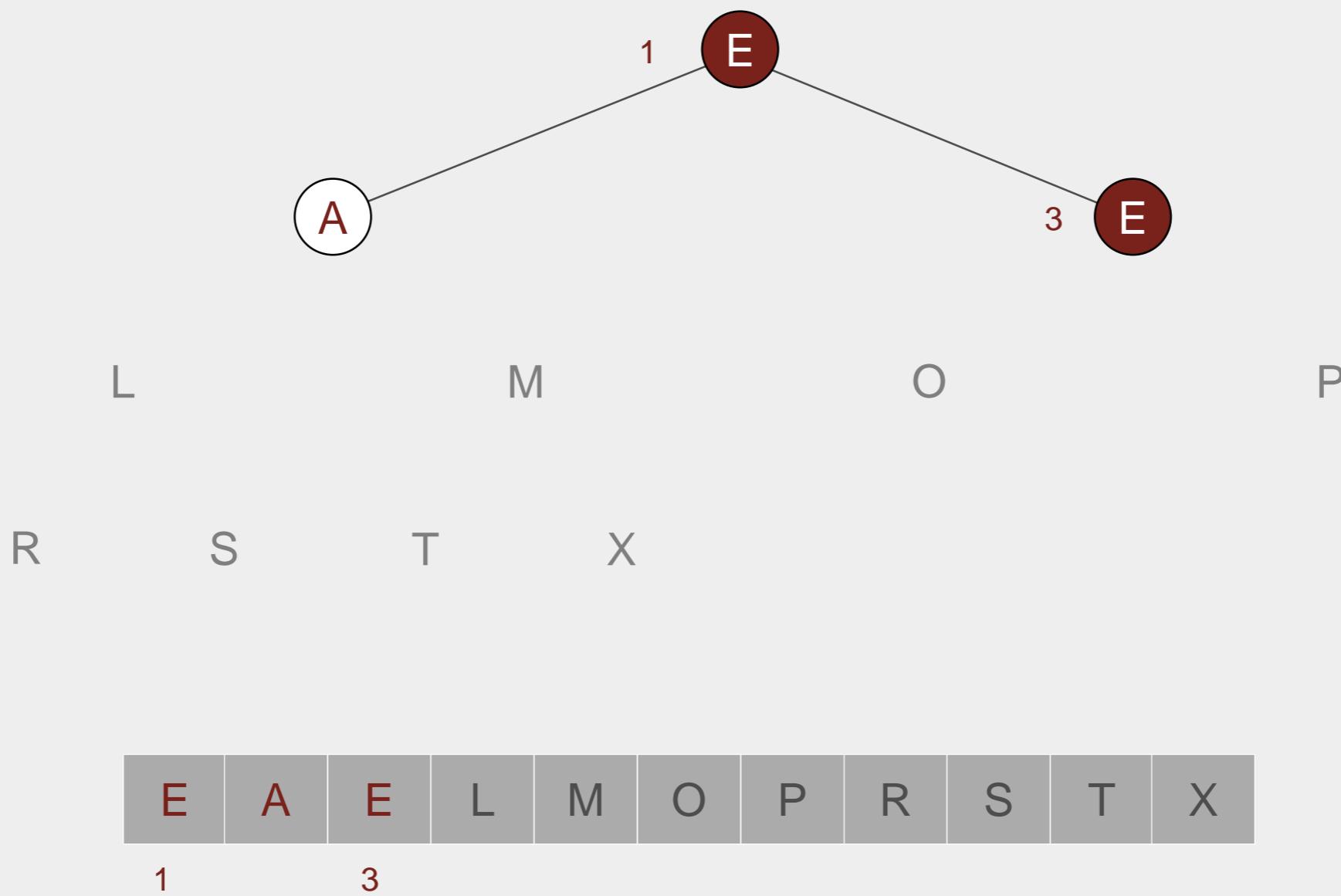
Sortdown. Repeatedly delete the largest remaining item.

exchange 1 and 3



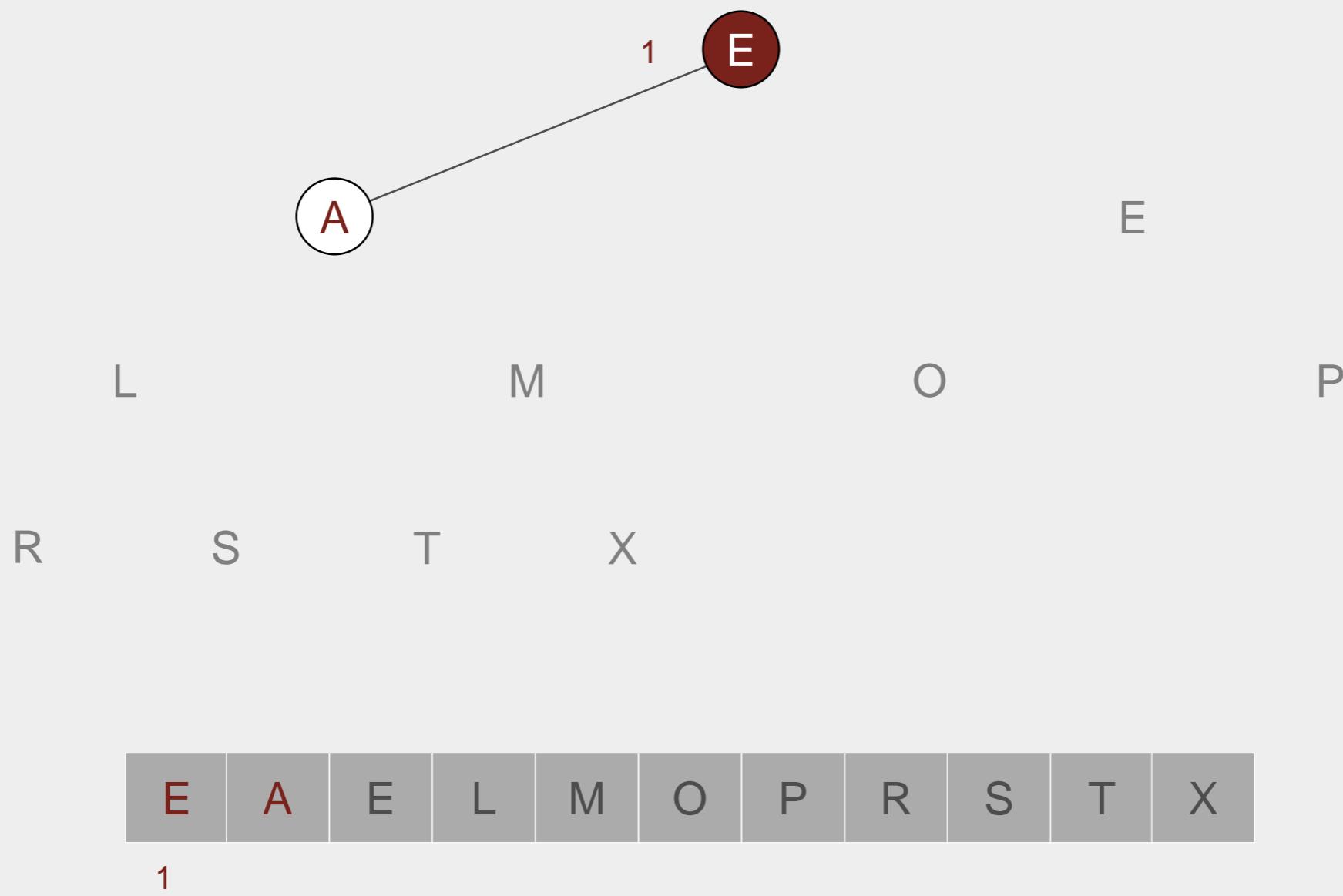
Sortdown. Repeatedly delete the largest remaining item.

exchange 1 and 3

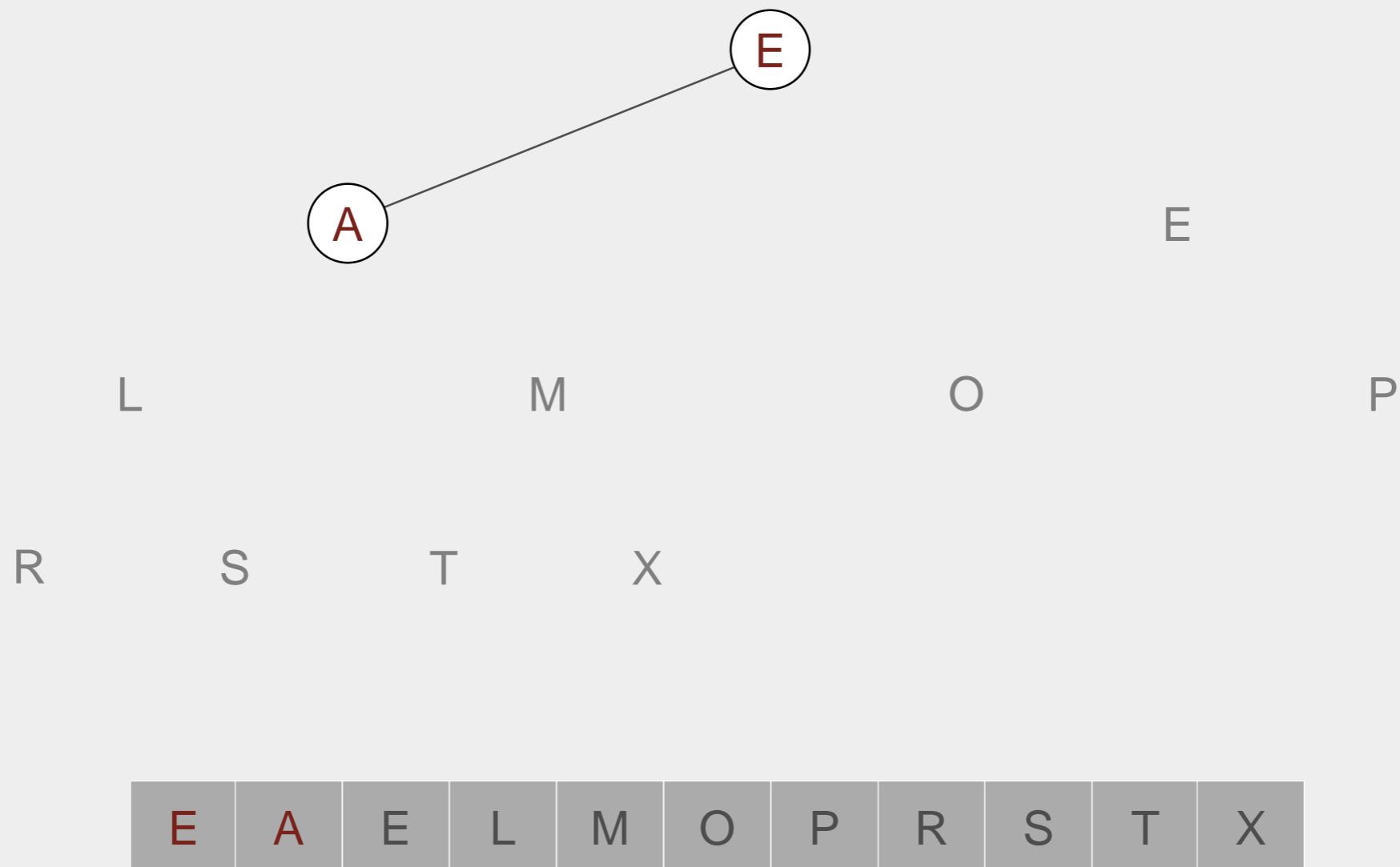


Sortdown. Repeatedly delete the largest remaining item.

sink 1

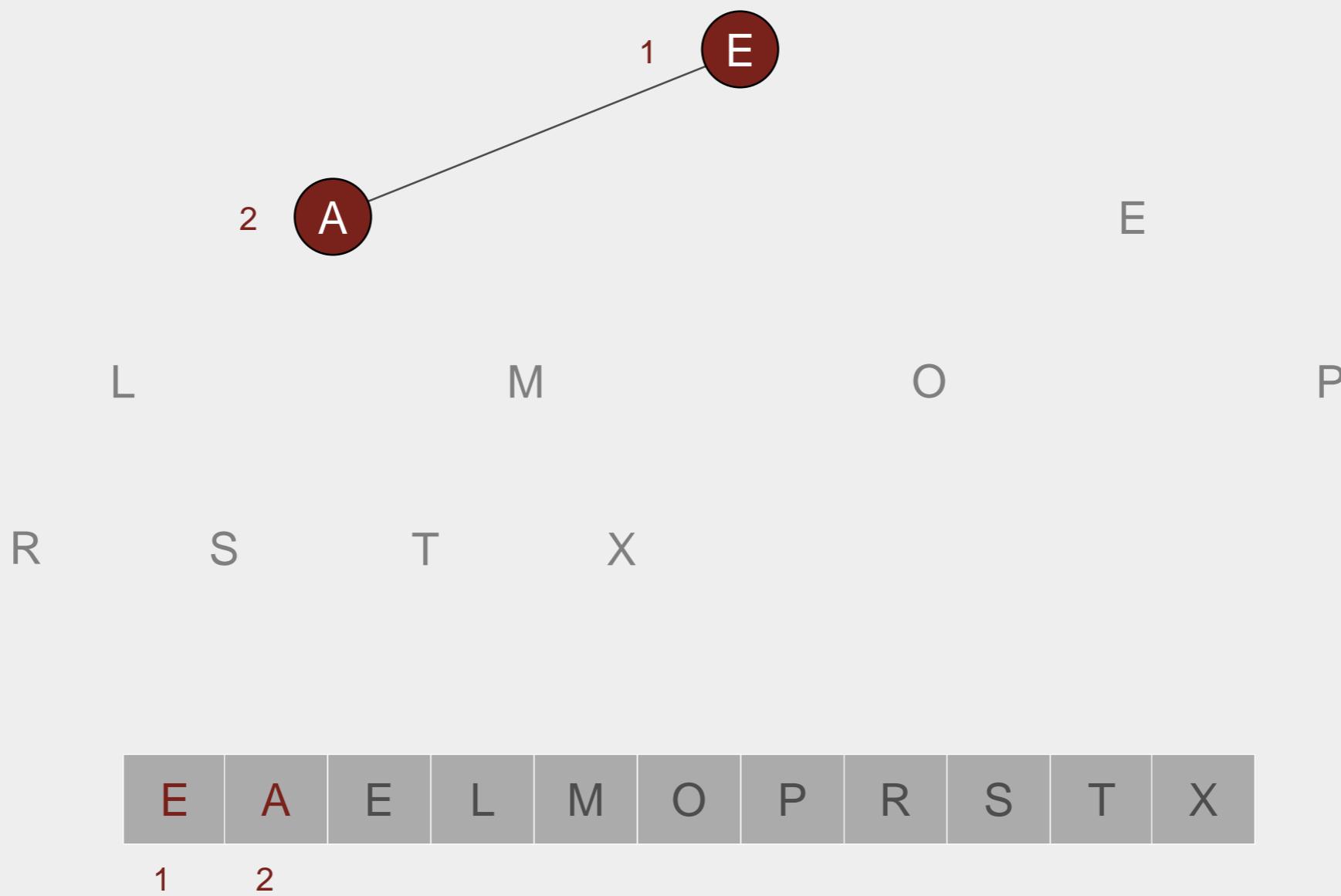


Sortdown. Repeatedly delete the largest remaining item.



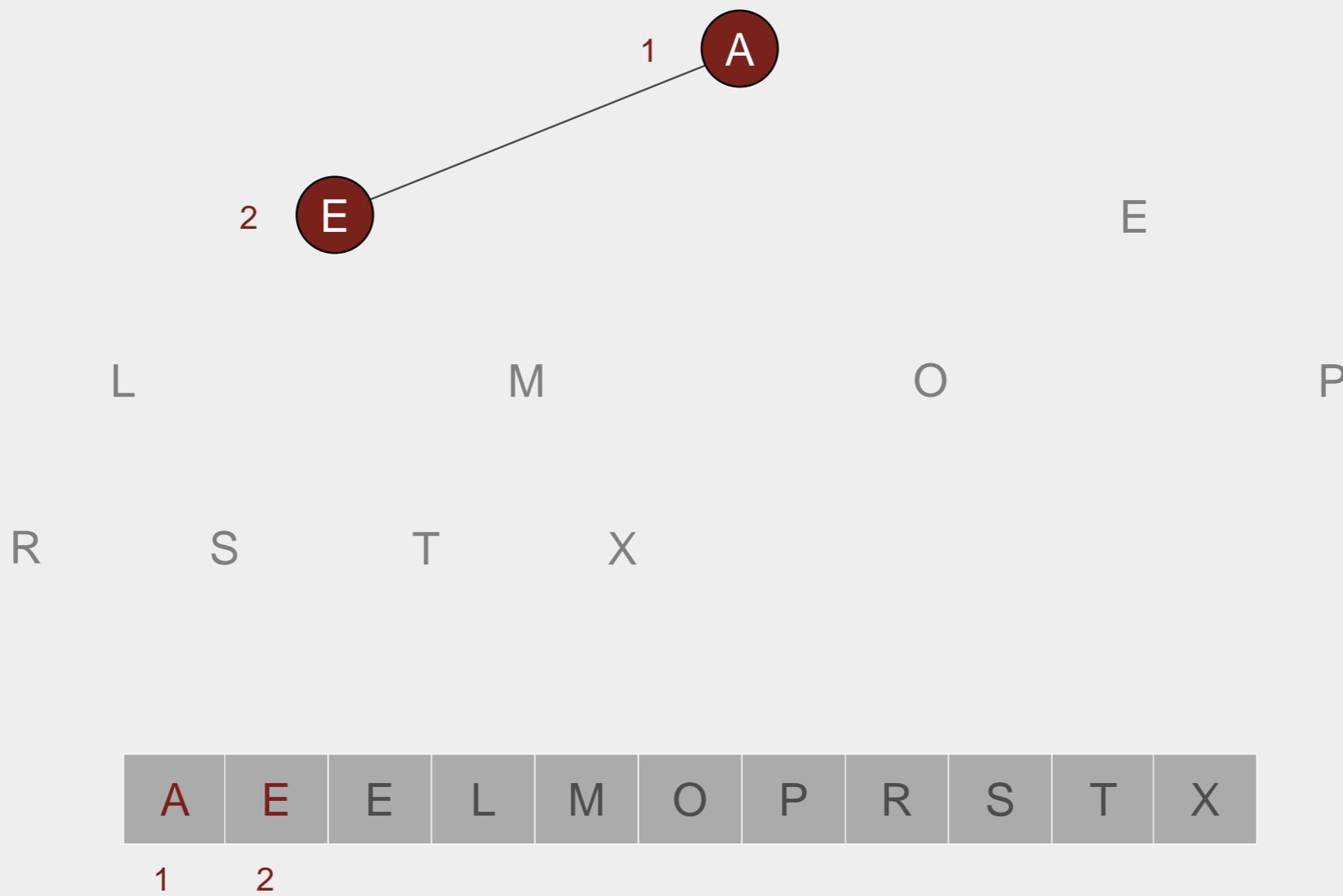
Sortdown. Repeatedly delete the largest remaining item.

exchange 1 and 2



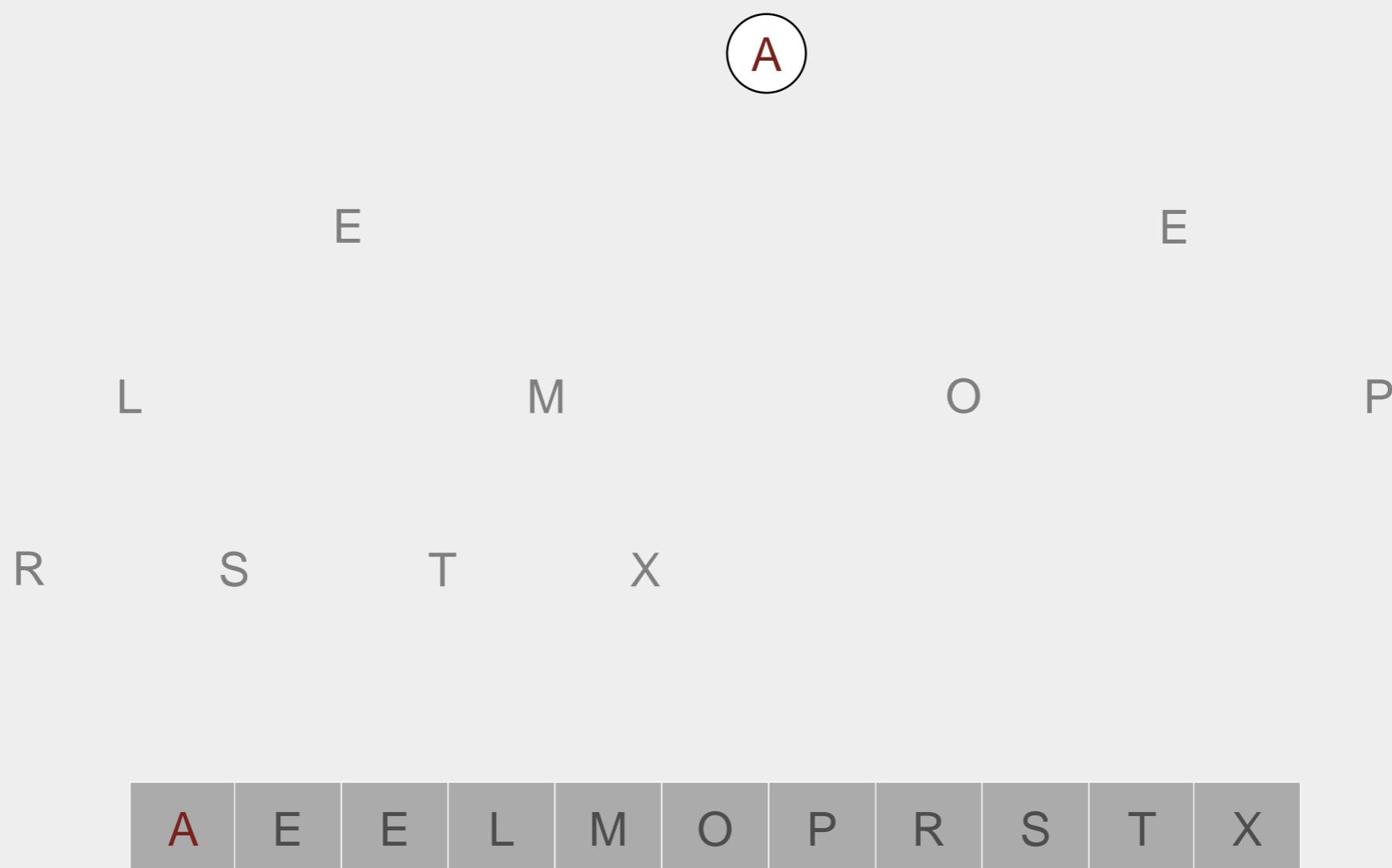
Sortdown. Repeatedly delete the largest remaining item.

exchange 1 and 2



Heapsort

Sortdown. Repeatedly delete the largest remaining item.



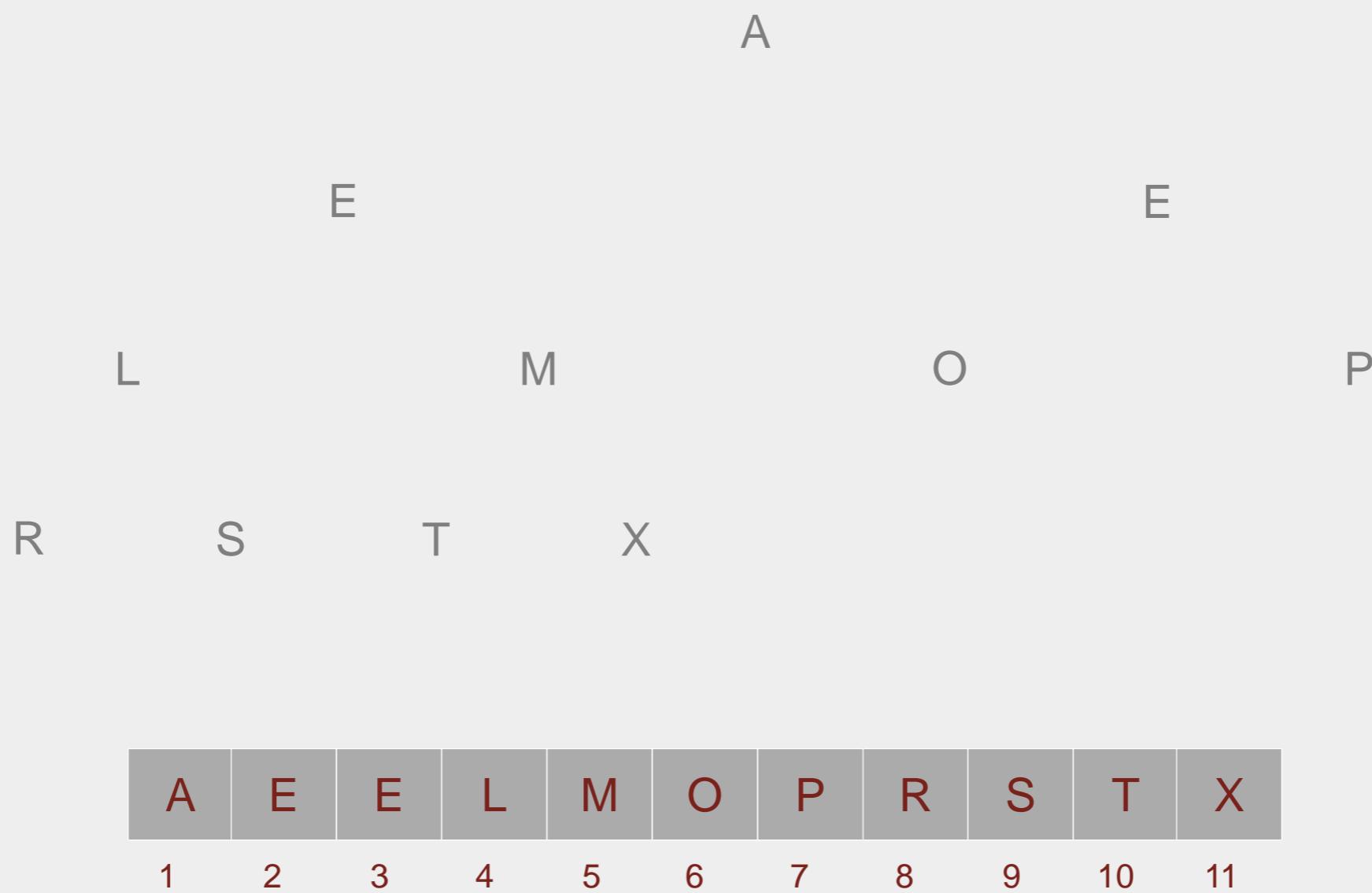
Sortdown. Repeatedly delete the largest remaining item.

end of sortdown phase



Heapsort

Ending point. Array in sorted order.



Integer Exponential

Problem: IntegerExponetial

Input: a real number x and a nonnegative integer n .

Output: x^n

Algorithm 5.4 ExpRec

Input: A real number x and a nonnegative integer n .

Output: x^n .

1. $\text{power}(x, n)$

Procedure $\text{power}(x, m)$

1. if $m = 0$ then $y \leftarrow 1$

2. else

3. $y \leftarrow \text{power}(x, \lfloor m/2 \rfloor)$

4. $y \leftarrow y^2$

5. if m is odd then $y \leftarrow xy$

6. end if

7. return y

Lecture 6 Divide-and-Conquer

- Master theorem
- Design divide-and-conquer algorithms

Overview

- Master theorem
- Design divide-and-conquer algorithms
 - Min-Max
 - Multiplication
 - Matrix multiplication
 - Selection

Divide-and-conquer

- **Divide Step:** the input is partitioned into $p \geq 1$ parts, each of which is strictly less than n .
- **Conquer Step:** p recursive call(s) of the smaller parts.
- **Combine Step:** combine the solved parts to obtain the desired output.

Divide-and-conquer

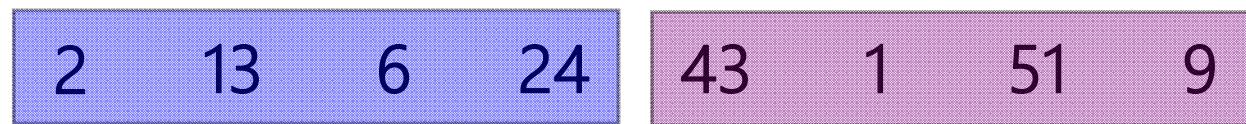
- **Divide Step:** the input is partitioned into $p \geq 1$ parts, each of which is strictly less than n .
- **Conquer Step:** p recursive call(s) of the smaller parts.
- **Combine Step:** combine the solved parts to obtain the desired output.



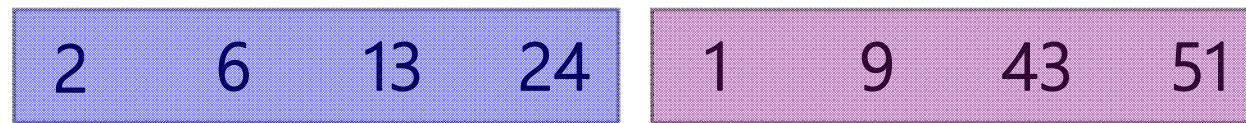
Which step is more difficult?

Mergesort

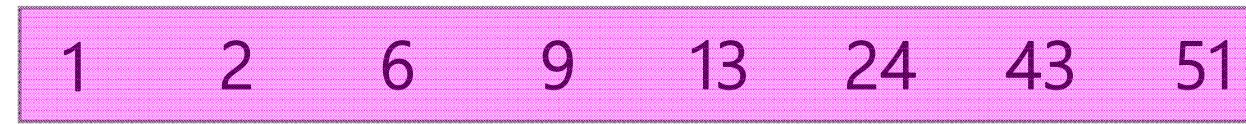
Divide



Conquer



Combine



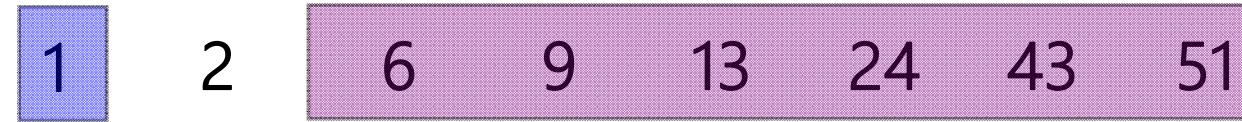
$$D(N) = 2 D(N/2) + N$$

Quicksort

Divide



Conquer



Combine



$$D(N) = 2 D(N/2) + N$$

Divide-and-conquer

- **Divide Step:** the input is partitioned into $p \geq 1$ parts, each of which is strictly less than n .
- **Conquer Step:** p recursive call(s) of the smaller parts.
- **Combine Step:** combine the solved parts to obtain the desired output.

$$f(n) = af(n/b) + cn^d$$

Master theorem

Simplified Master Theorem

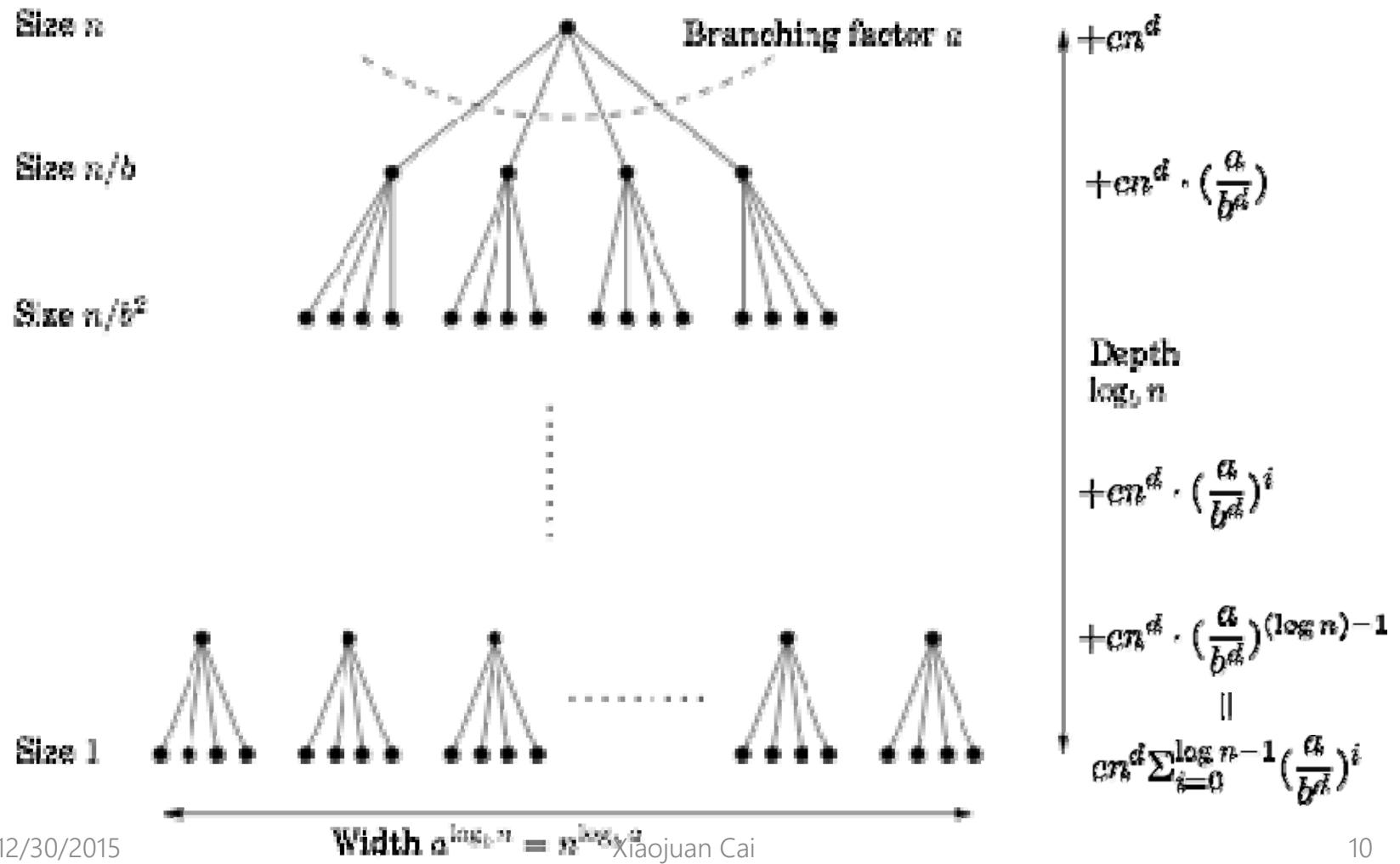
Let $a \geq 1, b > 1$ and $c, d, w \geq 0$ be constants, and let $T(n)$ be defined on the nonnegative integers by the recurrence

$$f(n) = \begin{cases} w & \text{if } n = 1 \\ af(n/b) + cn^d & \text{if } n > 1 \end{cases}$$

Then

$$f(n) = \begin{cases} \Theta(n^d) & \text{if } a < b^d \\ \Theta(n^d \log n) & \text{if } a = b^d \\ \Theta(n^{\log_b a}) & \text{if } a > b^d \end{cases}$$

Master theorem



Quiz

Give the growth order of following functions:

$$f(n) = f(n/2) + 1$$

$$f(n) = 2f(n/2) + n^2$$

$$f(n) = 9f(n/3) + n$$

$$f(n) = 5f(n/2) + n^3$$

- A. $\Theta(\log n)$
- B. $\Theta(n \log n)$
- C. $\Theta(n^2)$
- D. $\Theta(n^3)$

Where are we?

- Master theorem
- Design divide-and-conquer algorithms
 - Min-Max
 - Multiplication
 - Matrix multiplication
 - Selection

MinMax: Which one is better?

Problem: MinMax

Input: a sequence of integers $A[1\dots n]$

Output: the maximum element and the minimum element in $A[1\dots n]$

```
1.  $x \leftarrow A[1]; y \leftarrow A[1]$ 
2. for  $i \leftarrow 2$  to  $n$ 
3.   if  $A[i] < x$  then  $x \leftarrow A[i]$ 
4.   if  $A[i] > y$  then  $y \leftarrow A[i]$ 
5. end for
6. return  $(x, y)$ 
```

Procedure *minmax*(*low, high*)

```
1. if  $high - low = 1$  then
2.   if  $A[low] < A[high]$  then return  $(A[low], A[high])$ 
3.   else return  $(A[high], A[low])$ 
4. end if
5. else
6.    $mid' \leftarrow \lfloor (low + high)/2 \rfloor$ 
7.    $(x_1, y_1) \leftarrow minmax(low, mid')$ 
8.    $(x_2, y_2) \leftarrow minmax(mid' + 1, high)$ 
9.    $x \leftarrow \min\{x_1, x_2\}$ 
10.   $y \leftarrow \max\{y_1, y_2\}$ 
11. return  $(x, y)$ 
12. end if
```

Xiaojuan Cai

MinMax: Which one is better?

Problem: MinMax

Input: a sequence of integers $A[1\dots n]$, $n = 2^k$

Output: the maximum element and the minimum element in $A[1\dots n]$

```
1.  $x \leftarrow A[1]; y \leftarrow A[1]$ 
2. for  $i \leftarrow 2$  to  $n$ 
3.   if  $A[i] < x$  then  $x \leftarrow A[i]$ 
4.   if  $A[i] > y$  then  $y \leftarrow A[i]$ 
5. end for
6. return  $(x, y)$ 
```

#comp = $2n - 2$

Procedure minmax($low, high$)

```
1. if  $high - low = 1$  then
2.   if  $A[low] < A[high]$  then return  $(A[low], A[high])$ 
3.   else return  $(A[high], A[low])$ 
4. end if
5. else
6.    $mid \leftarrow \lfloor (low + high)/2 \rfloor$ 
7.    $(x_1, y_1) \leftarrow \text{minmax}(low, mid)$ 
8.    $(x_2, y_2) \leftarrow \text{minmax}(mid + 1, high)$ 
9.    $x \leftarrow \min\{x_1, x_2\}$ 
10.   $y \leftarrow \max\{y_1, y_2\}$ 
11. end if
12. return  $(x, y)$ 
```

#comp = $3n/2 - 2$

Cost more spaces

Multiplication

Problem: Multiplication

Input: two n-bits binary integers x and y

Output: $x \square y$

$$\begin{array}{r} & 1 & 1 & 0 & 1 \\ \times & 1 & 0 & 1 & 1 \\ \hline & 1 & 1 & 0 & 1 \\ & 1 & 1 & 0 & 1 \\ & 0 & 0 & 0 & 0 \\ + & 1 & 1 & 0 & 1 \\ \hline & 1 & 0 & 0 & 0 & 1 & 1 & 1 \end{array}$$

$$O(n) + O(n) + \dots + O(n) = (n-1) O(n) = O(n^2)$$

Multiplication

Problem: Multiplication

Input: two n-bits binary integers x and y

Output: $x \square y$

$$x = 2^{\frac{n}{2}}w + u$$

$$y = 2^{\frac{n}{2}}z + v$$

$$\begin{aligned} xy &= (2^{\frac{n}{2}}w + u)(2^{\frac{n}{2}}z + v) \\ &= 2^n wz + 2^{\frac{n}{2}}(uz + wv) + uv \end{aligned}$$

$$T(n) = 4T\left(\frac{n}{2}\right) + \Theta(n) = \Theta(n^2)$$

Multiplication

Problem: Multiplication

Input: two n-bits binary integers x and y

Output: $x \square y$

$$\begin{aligned} xy &= (2^{\frac{n}{2}}w + u)(2^{\frac{n}{2}}z + v) \\ &= 2^n wz + 2^{\frac{n}{2}}(uz + wv) + uv \end{aligned}$$

$$(w + u)(z + v) = wz + uz + wv + uv$$

$$uz + wv = (w + u)(z + v) - uv - wz$$

$$T(n) = 3T\left(\frac{n}{2}\right) + \Theta(n) = \Theta(n^{\log 3})$$

Matrix multiplication

$$\begin{pmatrix} a_{11} \dots a_{1n} \\ \vdots \\ a_{n1} \dots a_{nn} \end{pmatrix} \begin{pmatrix} b_{11} \dots b_{1n} \\ \vdots \\ b_{n1} \dots b_{nn} \end{pmatrix} = \begin{pmatrix} c_{11} \dots c_{1n} \\ \vdots \\ c_{n1} \dots c_{nn} \end{pmatrix}$$

where

$$c_{ij} = \sum_{k=1}^n a_{ik} b_{kj}$$

$$T(n) = \Theta(n^3)$$

Matrix multiplication

$$\begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix} = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix}$$

$$\begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} = \begin{pmatrix} A_{11}B_{11} + A_{12}B_{21} & A_{11}B_{12} + A_{12}B_{22} \\ A_{21}B_{11} + A_{22}B_{21} & A_{21}B_{12} + A_{22}B_{22} \end{pmatrix}$$

$$T(n) = \begin{cases} m & \text{if } n = 1 \\ 8T(n/2) + 4(n/2)^2 a & \text{if } n \geq 2 \end{cases}$$

$$T(n) = \Theta(n^3)$$

Strassen's algorithm

$$D_1 = (A_{11} + A_{22})(B_{11} + B_{22})$$

$$D_2 = (A_{21} + A_{22})B_{11}$$

$$D_3 = A_{11}(B_{12} - B_{22})$$

$$D_4 = A_{22}(B_{21} - B_{11})$$

$$D_5 = (A_{11} + A_{22})B_{22}$$

$$D_6 = (A_{21} - A_{11})(B_{11} + B_{12})$$

$$D_7 = (A_{12} - A_{22})(B_{21} + B_{22})$$

$$\begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} = \begin{pmatrix} D_1 + D_4 - D_5 + D_7 & D_3 + D_5 \\ D_2 + D_4 & D_1 + D_3 - D_2 + D_6 \end{pmatrix}$$

$$T(n) = \begin{cases} m & \text{if } n = 1 \\ 7T(n/2) + 18(n/2)^2a & \text{if } n \geq 2 \end{cases}$$

$$T(n) = \Theta(n^{\log 7}) = \Theta(n^{2.81})$$

Xiaojuan Cai

Medians and order statistic

Problem: Medians

Input: An array $A[1..n]$ of n elements

Output: The $\lceil n/2 \rceil$ -th smallest element in $A[1..n]$.

Problem: OrderStatistic (Selection)

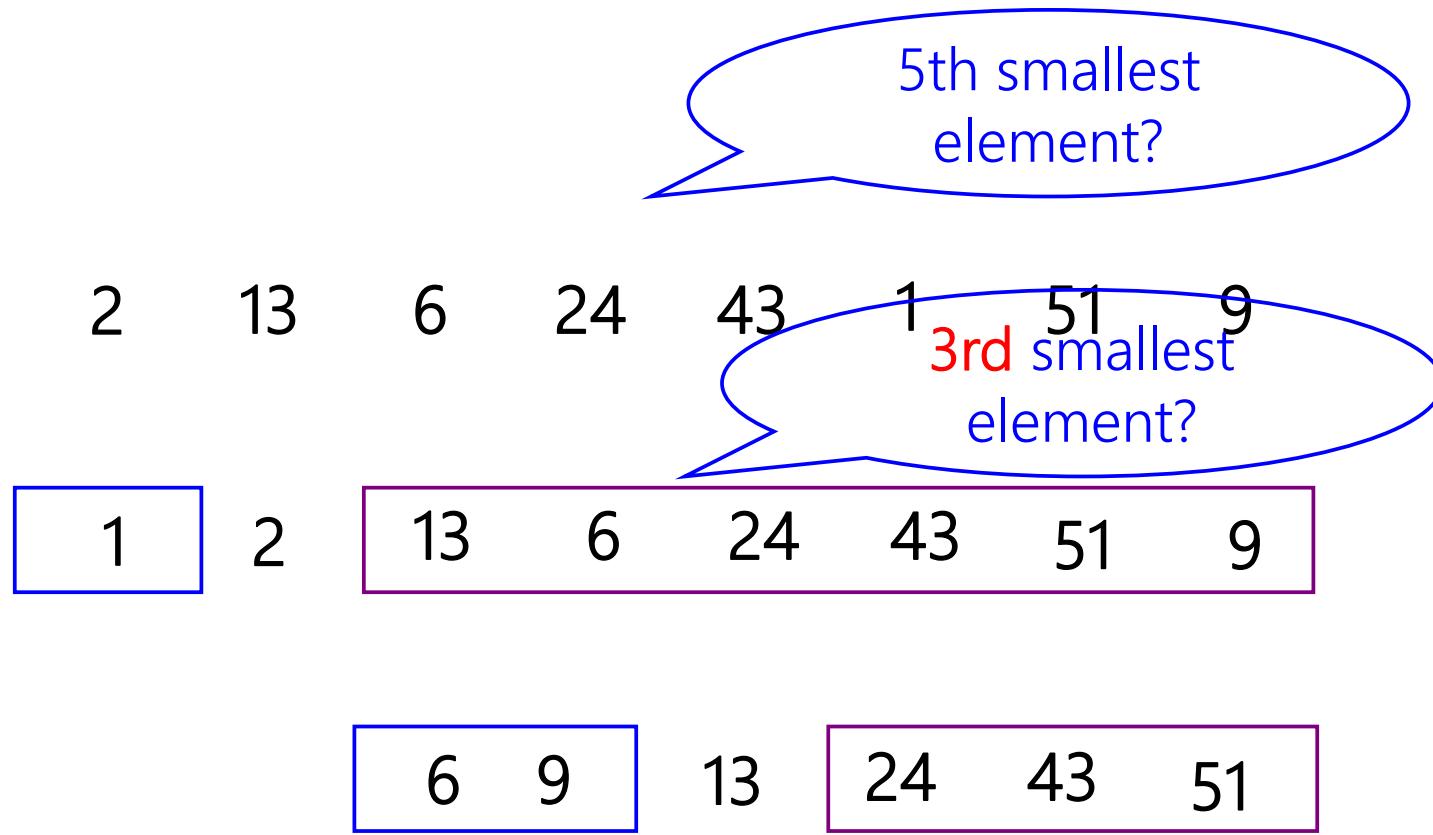
Input: An array $A[1..n]$ of n numbers, and a positive integer k

Output: The k -th smallest element in $A[1..n]$.



$O(n\log n)$

Selection



Quick-select

Partition array so that:

- Entry $a[j]$ is in place.
- No larger entry to the left of j .
- No smaller entry to the right of j .

Repeat in one subarray, depending on j ; finished when j equals k .



Worst case: $O(N^2)$

Lower bound: $O(N)$

Selection in linear time



[by M. Blum, R. Floyd, V. Pratt and R. Rivest]

Selection

Step 1. to get an integer good partitions.

3 2 4 99 92 4 2 84 84 21 328 74 83 45 75 183 38 17 37 38 92
29 451 5 3 95 27 7 7 91 9 727 41 1 42 25 45 79 26

3	4	328	183	92	95	9	25
2	2	74	38	29	27	727	45
4	84	83	17	451	7	41	79
99	84	45	37	5	7	1	26
92	21	75	38	3	91	42	

Selection

Step1. Sort every group of 5 elements.

3 2 4 99 92 4 2 84 84 21 328 74 83 45 75 183 38 17 37 38 92
29 451 5 3 95 27 7 7 91 9 727 41 1 42 25 45 79 26

2	2	45	17	3	7	1	25
3	4	74	37	5	7	9	26
4	21	75	38	29	27	41	45
92	84	83	38	92	91	42	79
99	84	328	183	451	95	727	

Selection

Step2. find the median of the medians.

3 2 4 99 92 4 2 84 84 21 328 74 83 45 75 183 38 17 37 38 92
29 451 5 3 95 27 7 7 91 9 727 41 1 42 25 45 79 26

2	2	45	17	3	7	1	25
3	4	74	37	5	7	9	
4	21	75	38	29	27	41	45
92	84	83	38	92	91	42	79
99	84	328	183	451	95	727	

A blue speech bubble with the word "Recursively" is drawn over the last row of the table.

Selection

Step3. Use the median as pivot.

3 2 4 99 92 4 2 84 84 21 328 74 83 45 75 183 38 17 37 38 92
29 451 5 3 95 27 7 7 91 9 727 41 1 42 25 45 79 26

2	2	45	17	3	7	1	25
3	4	74	37	5	7	9	
4	21	75	38	29	27	41	45
92	84	83	38	92	91	42	79
99	84	328	183	451	95	727	

A blue speech bubble with the word "Recursively" is pointing to the number 29 in the third row, fifth column of the table.

Selection

Why the median of the medians can guarantee a good partition?

3 2 4 99 92 4 2 84 84 21 328 74 83 45 75 183 38 17 37 38 92
29 451 5 3 95 27 7 7 91 9 727 41 1 42 25 45 79 26

2	2	45	17	3	7	1	25
3	4	74	37	5	7	9	26
4	21	75	38	29	27	41	45
92	84	83	38	92	91	42	79
99	84	328	183	451	95	727	

Selection

Why the median of the medians can guarantee a good partition?

3 2 4 99 92 4 2 84 84 21 328 74 83 45 75 183 38 17 37 38 92
29 451 5 3 95 27 7 7 91 9 727 41 1 42 25 45 79 26

2	2	7	3	17	45	1	25
3	4	7	5	37	74	9	26
4	21	27	29	38	75	41	45
92	84	91	92	38	83	42	79
99	84	95	451	183	328	727	

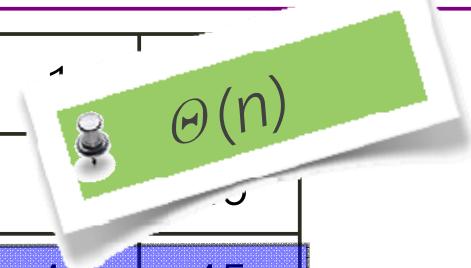
Selection

$$|\overline{A_1}| \geq 3\lceil \lfloor n/5 \rfloor / 2 \rceil \geq \frac{3}{2} \lfloor n/5 \rfloor$$

$$|A_1| \leq n - \frac{3}{2} \lfloor n/5 \rfloor \leq n - \frac{3}{2} \left(\frac{n-4}{5} \right) = 0.7n + 1.2$$

$$T(n) \leq T(\lfloor \frac{n}{5} \rfloor) + T(\lfloor \frac{3n}{4} \rfloor) + an, \text{ where } n \geq 44$$

2	2	7	3	17	45		
3	4	7	5	37	74		
4	21	27	29	38	75	41	45
92	84	91	92	38	83	42	79
99	84	95	451	183	328	727	



Selection: algorithm

```
Procedure select( $A, low, high, k$ )
```

1. $p \leftarrow high - low + 1$
2. if $p < 44$ then sort A and return ($A[k]$)
3. Let $q = \lfloor p/5 \rfloor$. Divide A into q groups of 5 elements each. If 5 does not divide p , then discard the remaining elements.
4. Sort each of the q groups individually and extract its median.
Let the set of the medians be M .
5. $mm \leftarrow select(M, 1, q, \lceil q/2 \rceil)$
6. Partition $A[low..high]$ into three arrays:
 $A_1 = \{a \mid a < mm\}$; $A_2 = \{a \mid a = mm\}$
 $A_3 = \{a \mid a > mm\}$
7. case
 - $|A_1| \geq k$: return $select(A_1, 1, |A_1|, k)$
 - $|A_1| < k$ and $|A_1| + |A_2| \geq k$: return mm
 - $|A_1| + |A_2| < k$: return $select(A_3, 1, |A_3|, k - |A_1| - |A_2|)$
8. end case

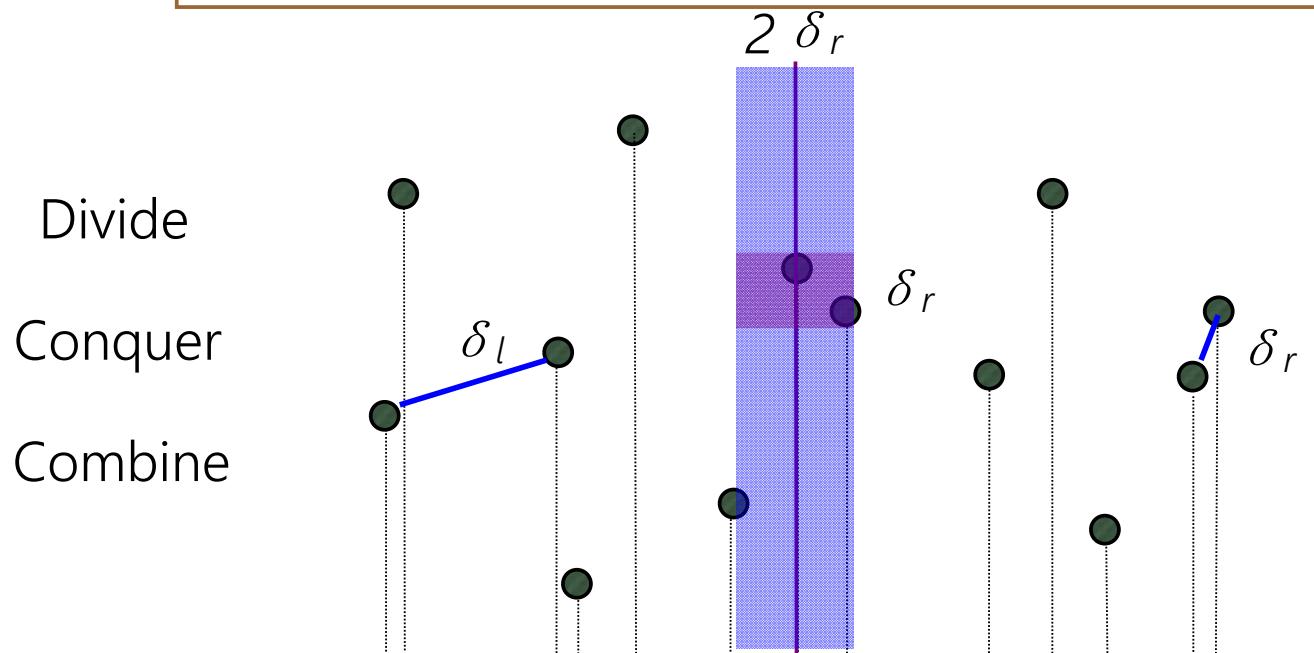


Closest pair problem

Problem: ClosestPair

Input: A set S of n points in the plane

Output: The pair $p1 = (x_1, y_1)$ and $p2 = (x_2, y_2)$ such that the Euclidean distance between $p1$ and $p2$ is minimal.



Closest pair problem

Algorithm 6.7 ClosestPair

Input: A set of n points in the plane.

Output: The minimum separation realized by two points in S .

1. Sort the points in S in nondecreasing order of their x -coordinates.
2. $Y \leftarrow$ the points in S sorted in nondecreasing order of their y -coordinates.
3. $\delta \leftarrow cp(1, n)$

Procedure $cp(low, high)$

```
1.  $p \leftarrow high - low + 1$ 
2. if  $high - low + 1 \leq 3$  then compute  $\delta$ 
3. else
4.    $mid \leftarrow \lfloor (low + high)/2 \rfloor$ 
5.    $x_0 \leftarrow x(S[mid])$ 
6.    $\delta_l \leftarrow cp(low, mid)$ 
7.    $\delta_r \leftarrow cp(mid + 1, high)$ 
8.    $\delta \leftarrow \min\{\delta_l, \delta_r\}$ 

9.    $k \leftarrow 0$ 
10.  for  $i \leftarrow 1$  to  $n$ 
11.    if  $|x(Y[i]) - x_0| \leq \delta$  then
12.       $k \leftarrow k + 1$ ;  $T[k] \leftarrow Y[i]$  end if
13.  end for
14.   $\delta' \leftarrow \delta$ 
15.  for  $i \leftarrow 1$  to  $k - 1$ 
16.    for  $j \leftarrow i + 1$  to  $\min\{i + 7, k\}$ 
17.      if  $d(T[i], T[j]) < \delta'$  then  $\delta' \leftarrow d(T[i], T[j])$ 
18.    end for
19.  end for
20. end if
21. return  $\delta'$ 
```

Closest pair problem

$$T(n) = \begin{cases} 1 & \text{if } n = 2 \\ 3 & \text{if } n = 3 \\ 2T(n/2) + \Theta(n) & \text{if } n > 3 \end{cases}$$

$$T(n) = \Theta(n \log n)$$

Conclusion

- Master theorem
- Design divide-and-conquer algorithms
 - Min-Max
 - Multiplication
 - Matrix multiplication
 - Selection

Groups in Selection algorithm

If each group consists of other than 5 elements, which of the following will not keep $\Theta(n)$ complexity?

- A. 3
- B. 7
- C. 9
- D. all of above

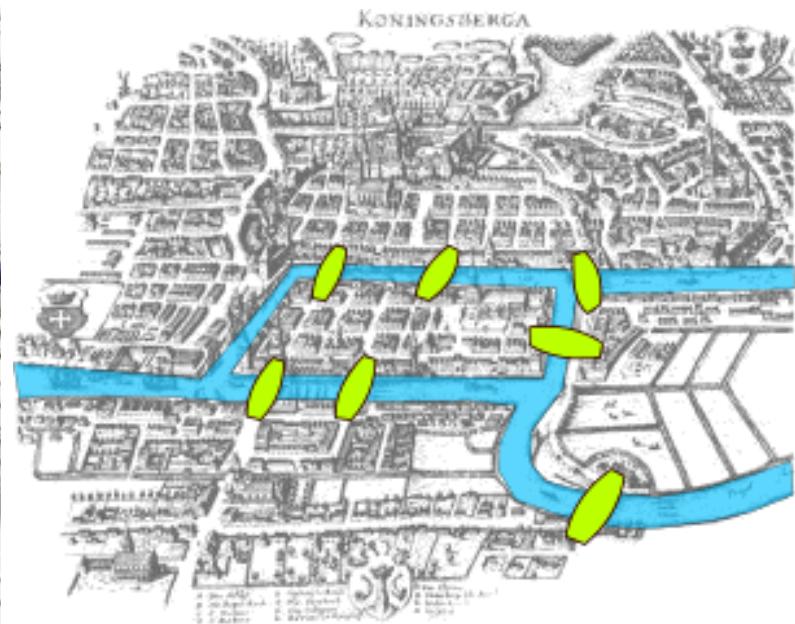
Lecture 7 Graph Traversal

- Graph
- Undirected graph
- Directed graph

Overview

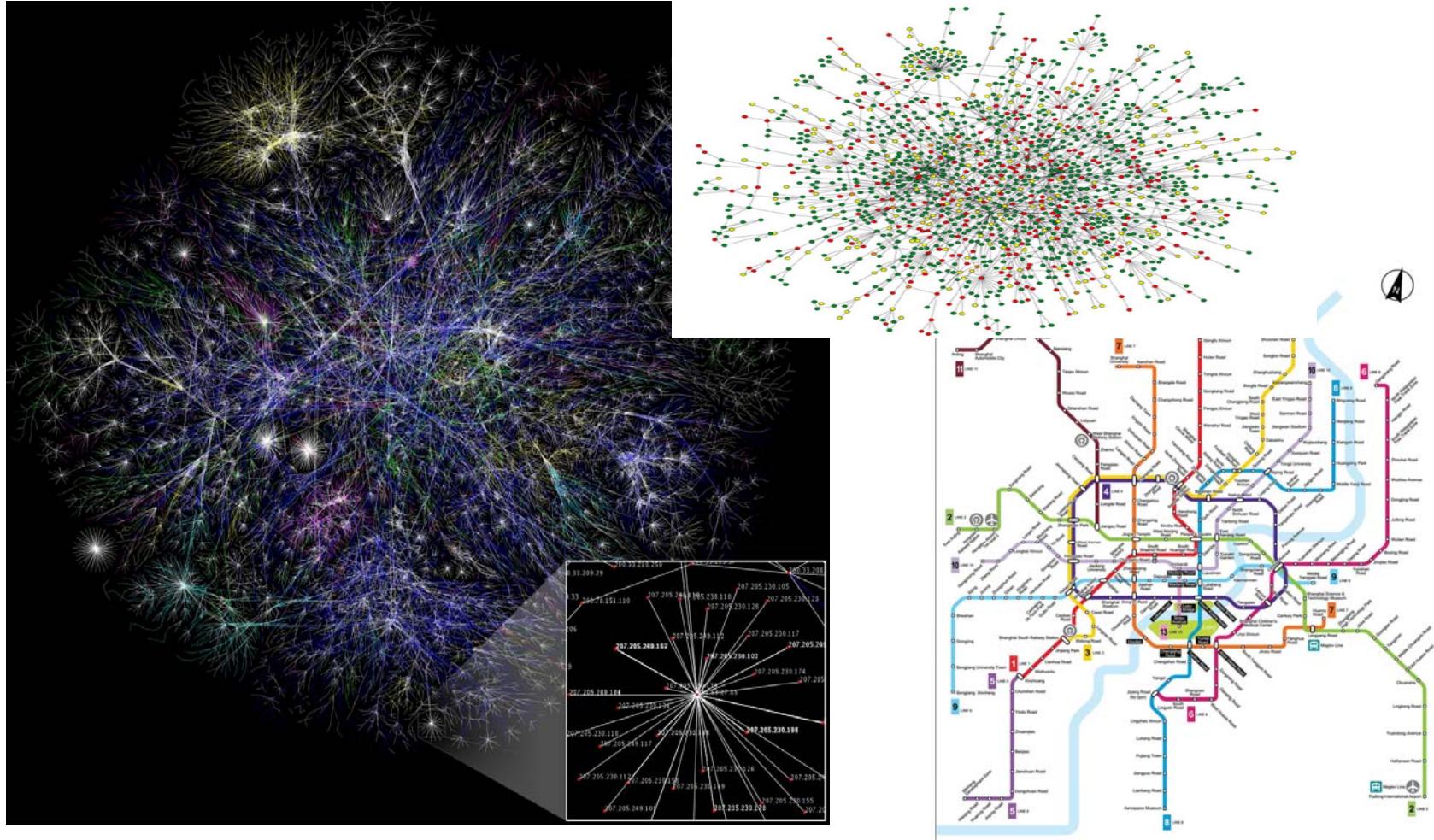
- Graph
 - Undirected graph
 - DFS, BFS, Application
 - Directed graph
 - DFS, BFS, Application

Graph theory



The Königsberg Bridge problem
(Source from Wikipedia)

Graph everywhere

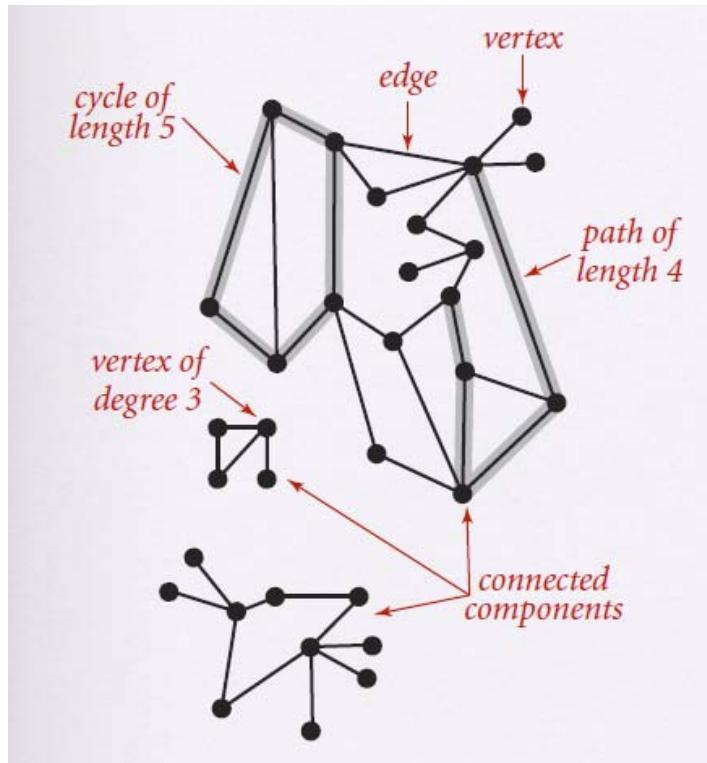


12/30/2015

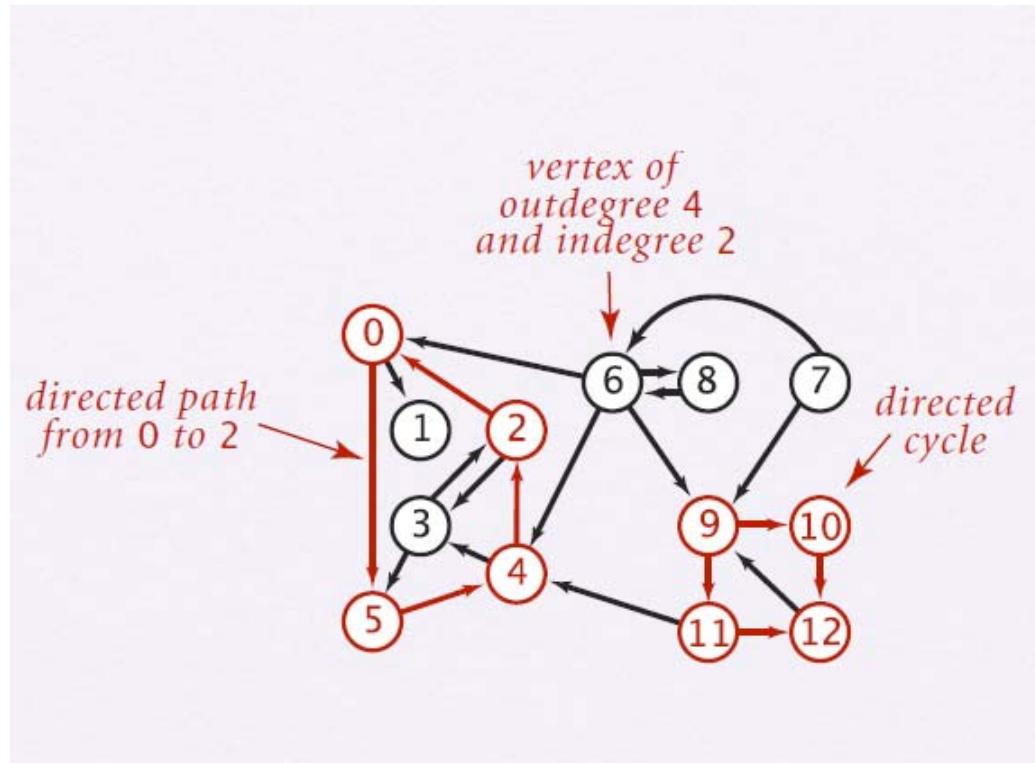
Xiaojuan Cai

4

Graph terminology



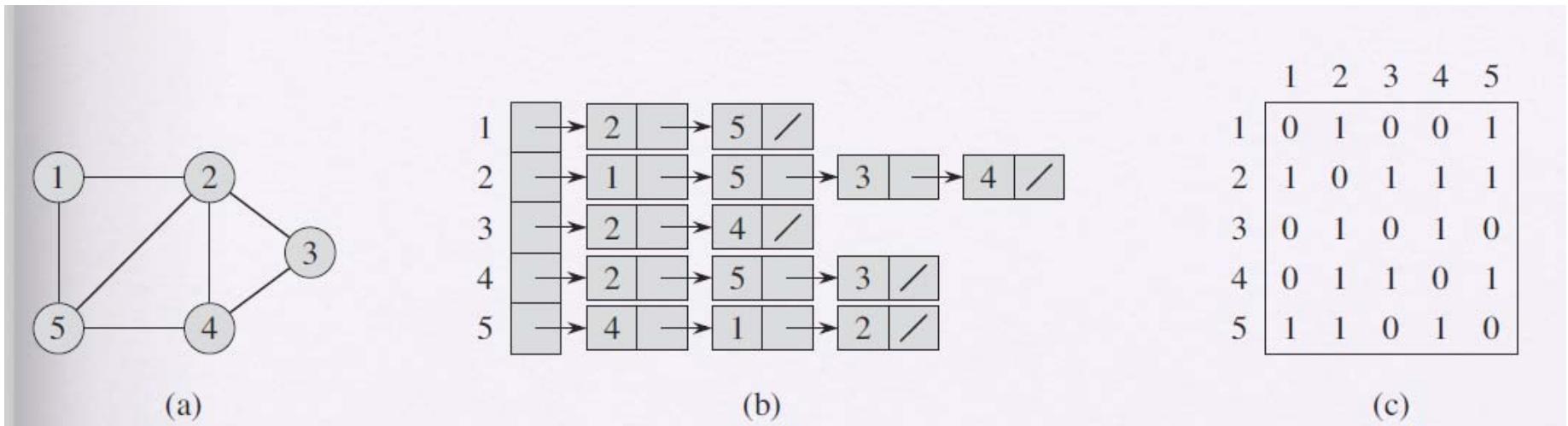
Undirected graph



Directed graph

Adjacency Matrix v.s. Adjacency List

$G = \langle V, E \rangle$



Directed: $n + m$

Undirected: $n + 2m$

Directed: n^2

Undirected: n^2

For every edge connected with v ...
Is u and v connected with an edge?

Important graph problems

Path. Is there a directed path from s to t ?

Shortest path. What is the shortest directed path from s to t ?

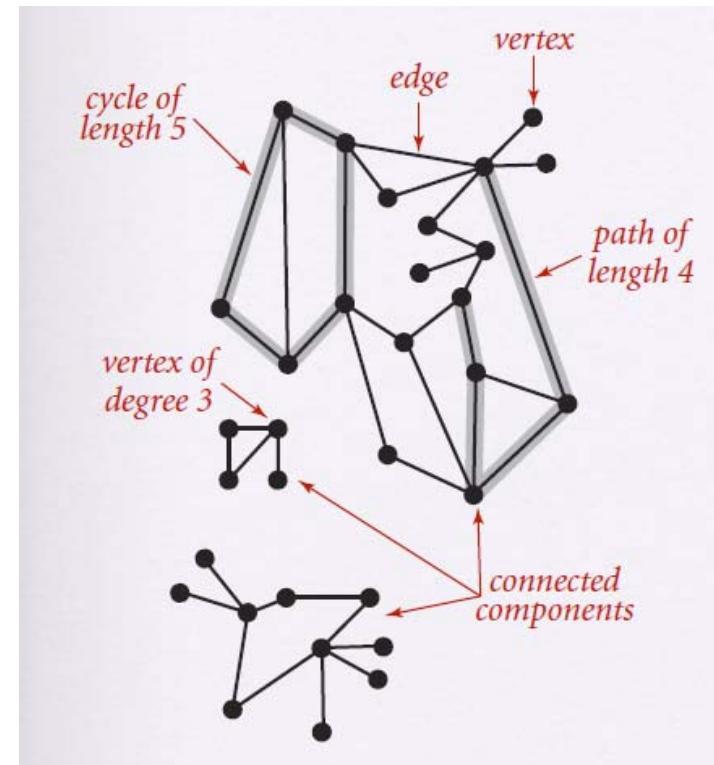
Topological sort. Can you draw the digraph so that all edges point upwards?

Strong connectivity. Is there a directed path between all pairs of vertices?

Transitive closure. For each vertices v and w , is there a path from v to w ?

Where are we?

- Graph
- Undirected graph
 - DFS, BFS, Application
- Directed graph
 - DFS, BFS, Application



DFS

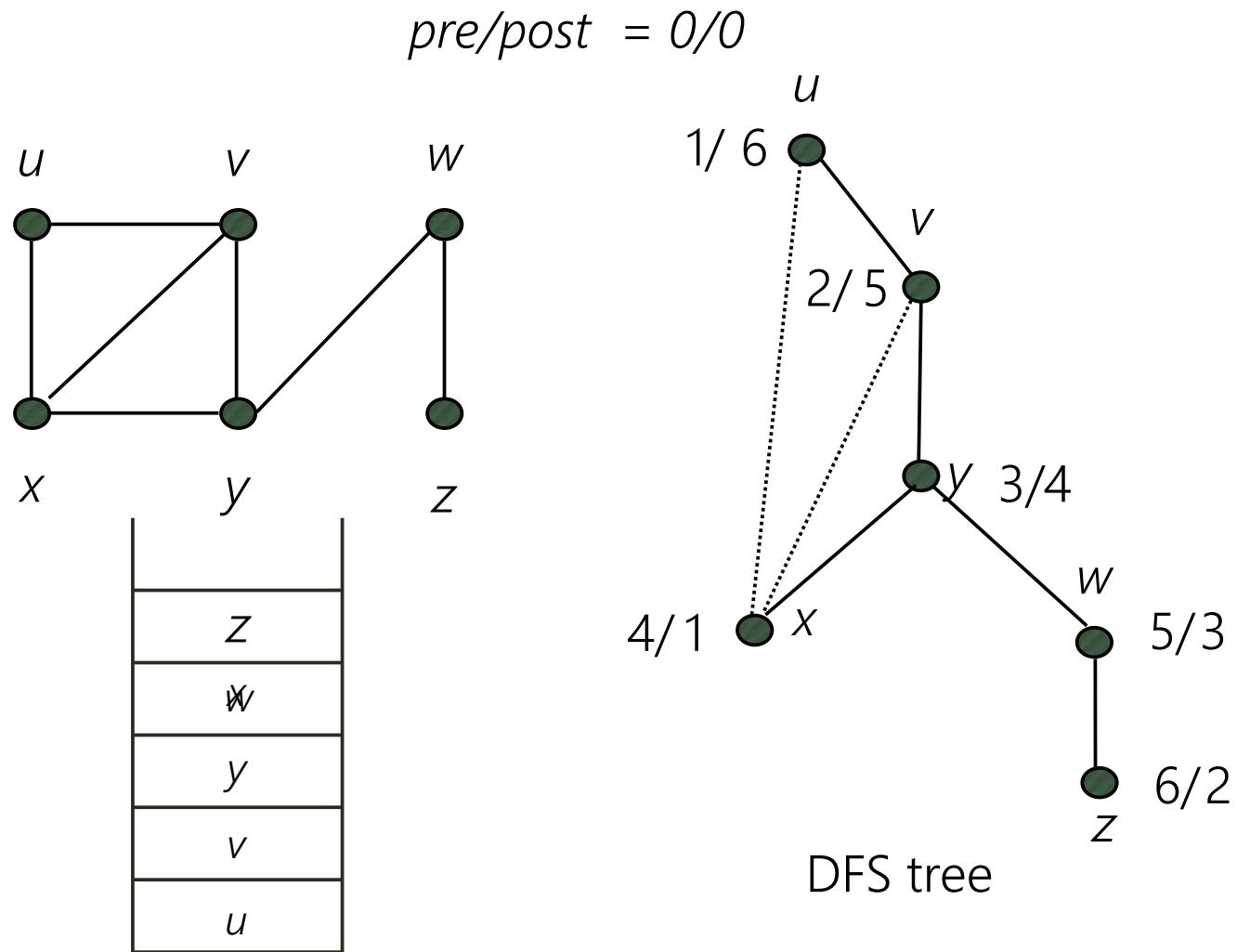


Depth-first-search.

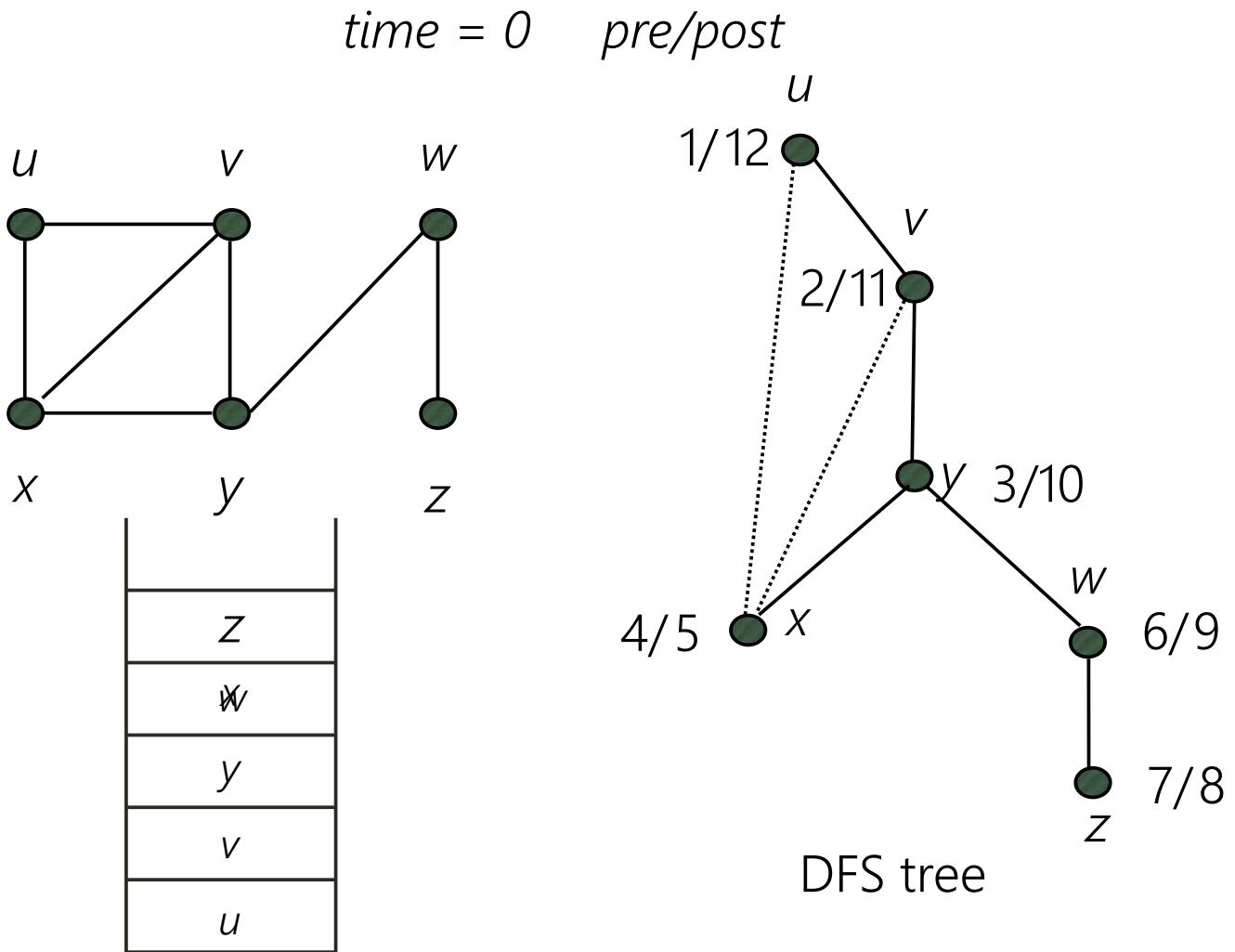
- Unroll a ball of string behind you.
- Mark each visited intersection and each visited passage.
- Retrace steps when no unvisited options.

Maze exploration

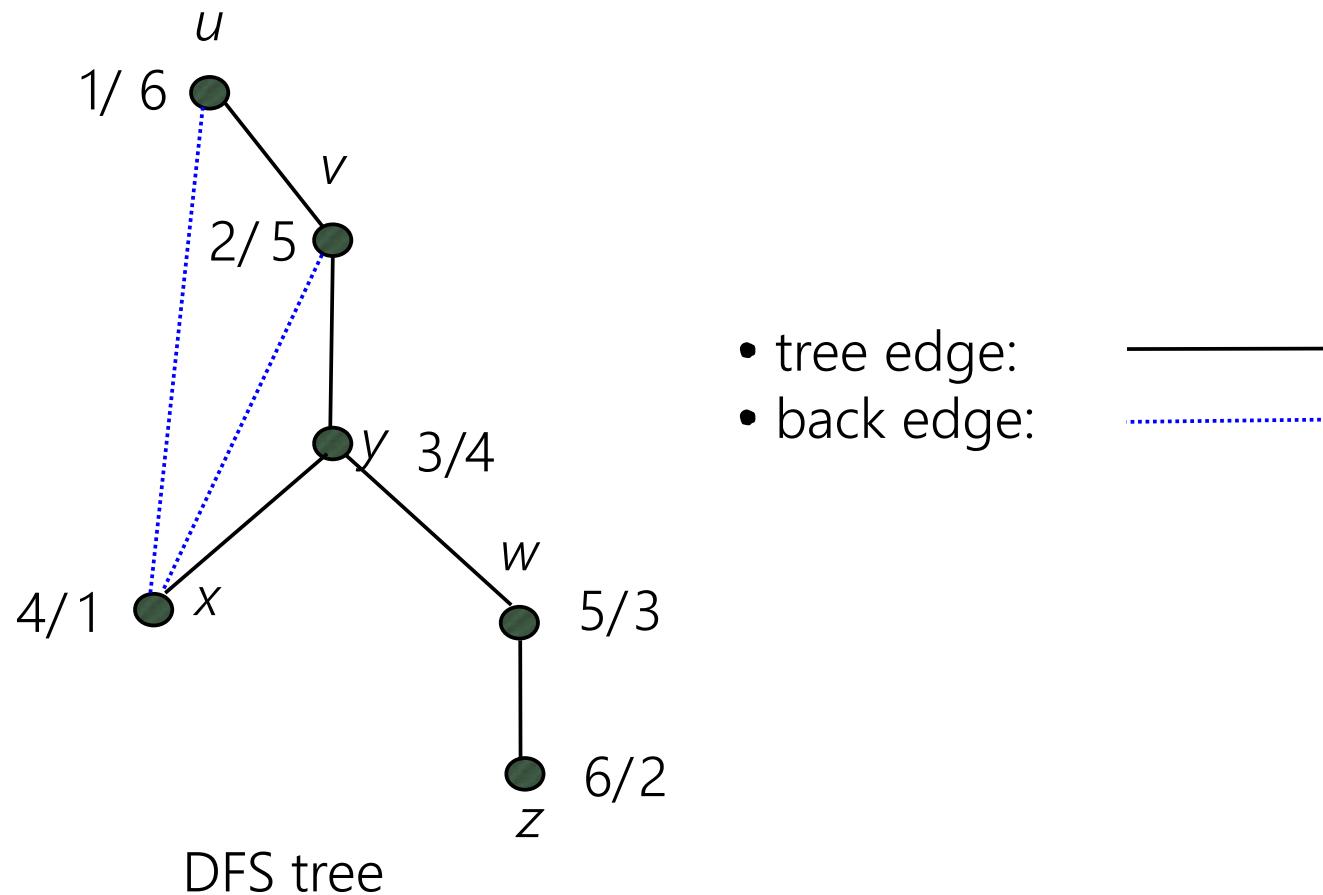
Depth-first search



Depth-first search



DFS tree: undirected



Algorithm 9.1 DFS

Input: A directed or undirected graph $G = (V, E)$.

Output: Preordering and postordering of the vertices in the corresponding depth first tree.

1. $\text{predfn} \leftarrow 0; \text{postdfn} \leftarrow 0$
2. **for** each vertex $v \in V$
3. mark v unvisited
4. **end for**
5. **for** each vertex $v \in V$
6. **if** v is marked unvisited **then** $\text{dfs}(v)$
7. **end for**

Procedure $\text{dfs}(v)$

1. mark v visited
2. $\text{predfn} \leftarrow \text{predfn} + 1$
3. **for** each edge $(v, w) \in E$
4. **if** w is marked unvisited **then** $\text{dfs}(w)$
5. **end for**
6. $\text{postdfn} \leftarrow \text{postdfn} + 1$

Algorithm 9.1 DFS

Input: A directed or undirected graph $G = (V, E)$.

Output: Preordering and postordering of the vertices in the corresponding depth first tree.

1. ~~predfn~~ $\leftarrow 0$; ~~postdfn~~ $\leftarrow 0$ time $\leftarrow 0$
2. for each vertex $v \in V$
3. mark v unvisited v.pre $\leftarrow \text{infinity}$
4. end for v.post $\leftarrow \text{infinity}$
5. for each vertex $v \in V$
6. if v is marked unvisited then $\text{dfs}(v)$
7. end for

Procedure $\text{dfs}(v)$

1. mark v visited
2. ~~predfn~~ \leftarrow ~~predfn~~ + 1 time \leftarrow time + 1
v.pre \leftarrow time
3. for each edge $(v, w) \in E$
4. if w is marked unvisited then $\text{dfs}(w)$
5. end for
6. ~~postdfn~~ \leftarrow ~~postdfn~~ + 1 time \leftarrow time + 1
v.post \leftarrow time

Quiz: Complexity

		Time	Space
Undirected	Adj. Matrix	?	?
	Adj. List	?	

Complexity

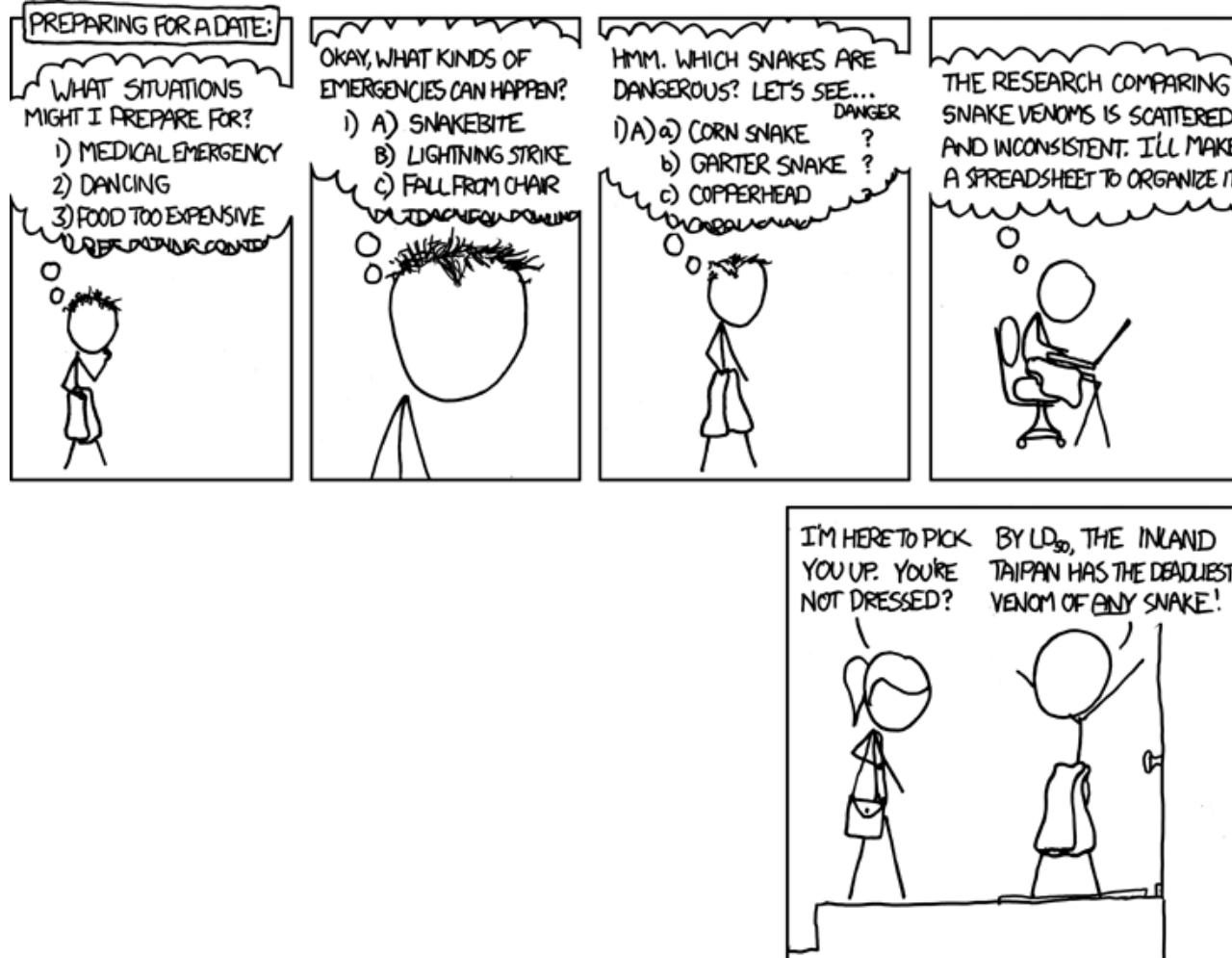
		Time	Space
Undirected	Adj. Matrix	$O(V ^2)$	$O(V)$
	Adj. List	$O(V + 2 E)$	

DFS correctness

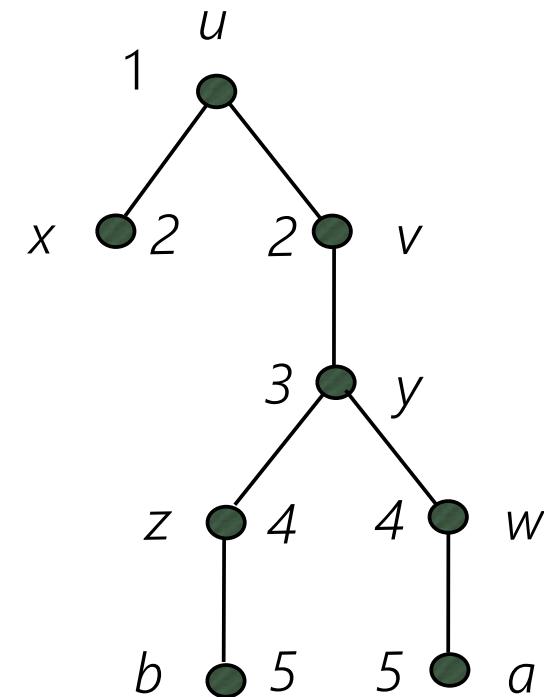
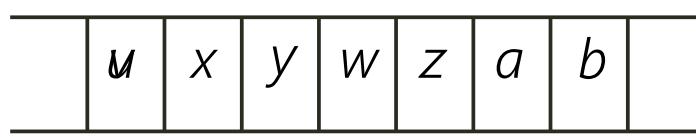
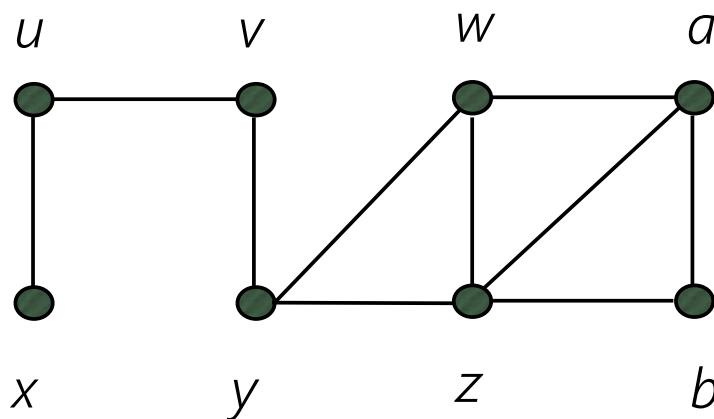
Proposition

DFS marks all vertices connected to s in time proportional to the sum of their degrees.

DFS application?



Breadth-first search



BFS tree

Algorithm 9.4 BFS

Input: A directed or undirected graph $G = (V, E)$.

Output: Numbering of the vertices in breadth-first search order.

1. $bfn \leftarrow 0$
2. for each vertex $v \in V$
3. mark v unvisited
4. end for
5. for each vertex $v \in V$
6. if v is marked unvisited then $bfs(v)$
7. end for

Procedure $bfs(v)$

1. $Q \leftarrow \{v\}$
2. mark v visited
3. while $Q \neq \emptyset$
4. $v \leftarrow Pop(Q)$
5. $bfn \leftarrow bfn + 1$
6. for each edge $(v, w) \in E$
7. if w is marked unvisited then
8. $Push(w, Q)$
9. mark w visited
10. end if
11. end for
12. end while

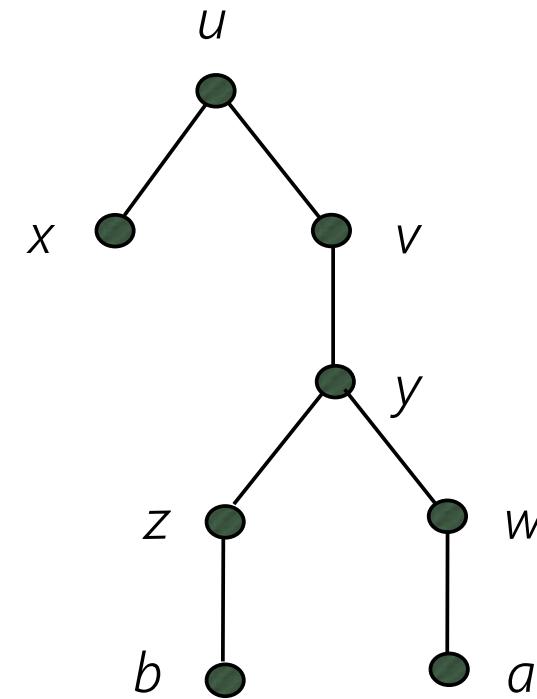
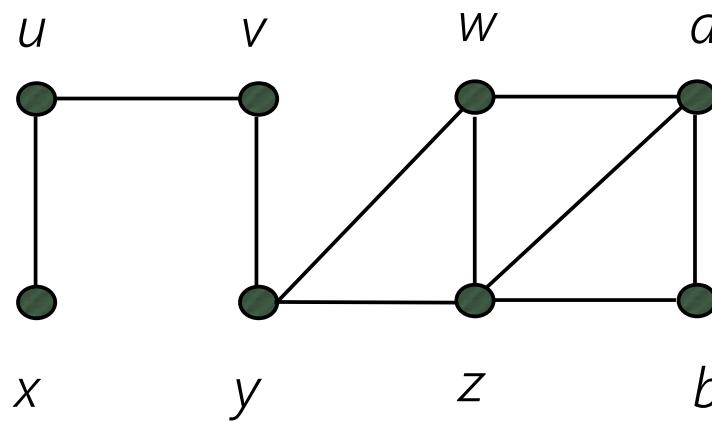
Quiz: Complexity

		Time	Space
Undirected	Adj. Matrix	?	?
	Adj. List	?	

Complexity

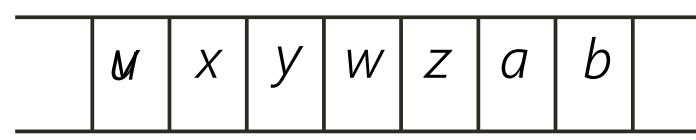
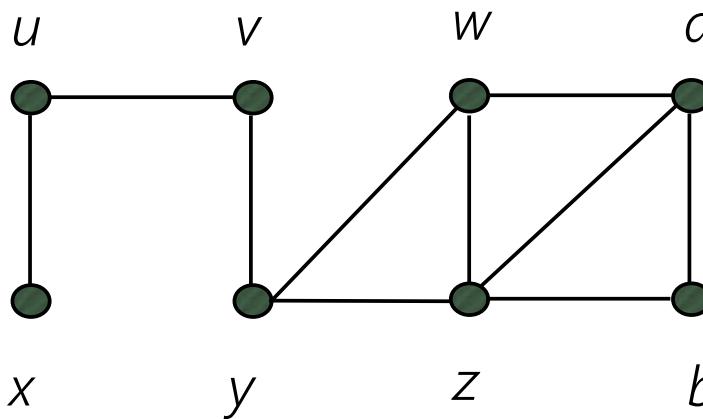
		Time	Space
Undirected	Adj. Matrix	$O(V ^2)$	$O(V)$
	Adj. List	$O(V + 2 E)$	

BFS Application: The shortest path

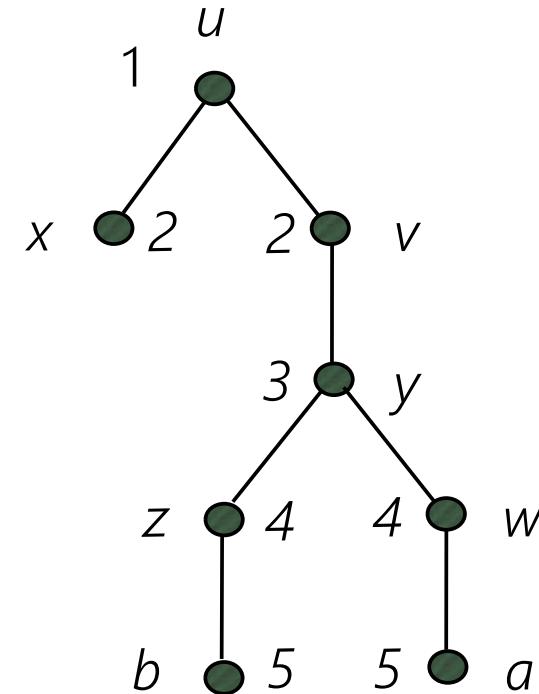


Discussion: How to record the shortest path?

Breadth-first search



parent	u	v	x	y	w	z	a	b
		u	u	v	y	y	w	z



BFS tree

Algorithm 9.4 BFS

Input: A directed or undirected graph $G = (V, E)$.

Output: Numbering of the vertices in breadth-first search order.

1. $bfn \leftarrow 0$
2. for each vertex $v \in V$
3. mark v unvisited
4. end for
5. for each vertex $v \in V$
6. if v is marked unvisited then $bfs(v)$
7. end for

Procedure $bfs(v)$

1. $Q \leftarrow \{v\}$
2. mark v visited
3. while $Q \neq \emptyset$
4. $v \leftarrow Pop(Q)$
5. $bfn \leftarrow bfn + 1$
6. for each edge $(v, w) \in E$
7. if w is marked unvisited then
8. $Push(w, Q)$
9. mark w visited
10. end if
11. end for
12. end while

Algorithm 9.4 BFS

Input: A directed or undirected graph $G = (V, E)$.

Output: Numbering of the vertices in breadth-first search order.

1. $bfn \leftarrow 0$
2. for each vertex $v \in V$
3. mark v unvisited
4. end for
5. for each vertex $v \in V$
6. if v is marked unvisited then $bfs(v)$
7. end for

Procedure $bfs(v)$

1. $Q \leftarrow \{v\}$
2. mark v visited
3. while $Q \neq \emptyset$
4. $v \leftarrow Pop(Q)$
5. $bfn \leftarrow bfn + 1$
6. for each edge $(v, w) \in E$
7. if w is marked unvisited then
8. Push(w, Q) 
9. mark w visited
10. end if
11. end for
12. end while

Connectivity

Definition

An undirected graph is **connected**, if there is a path between any pair of vertices.

A **connected component** is a subgraph that is internally connected but has no edges to the remaining vertices.

#trees == #connected components

Graph acyclicity

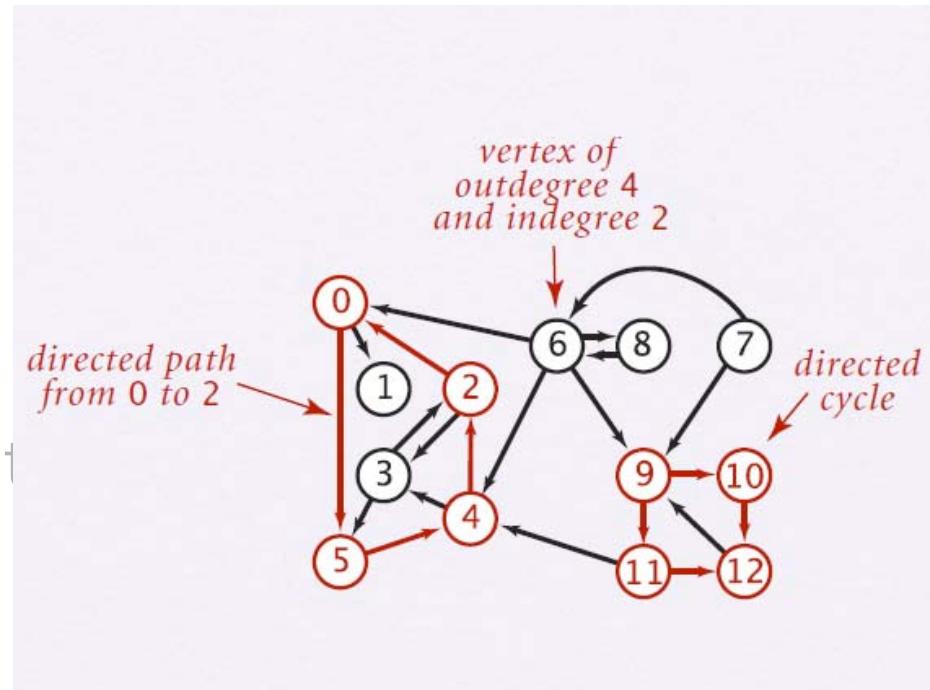
NO back edges!



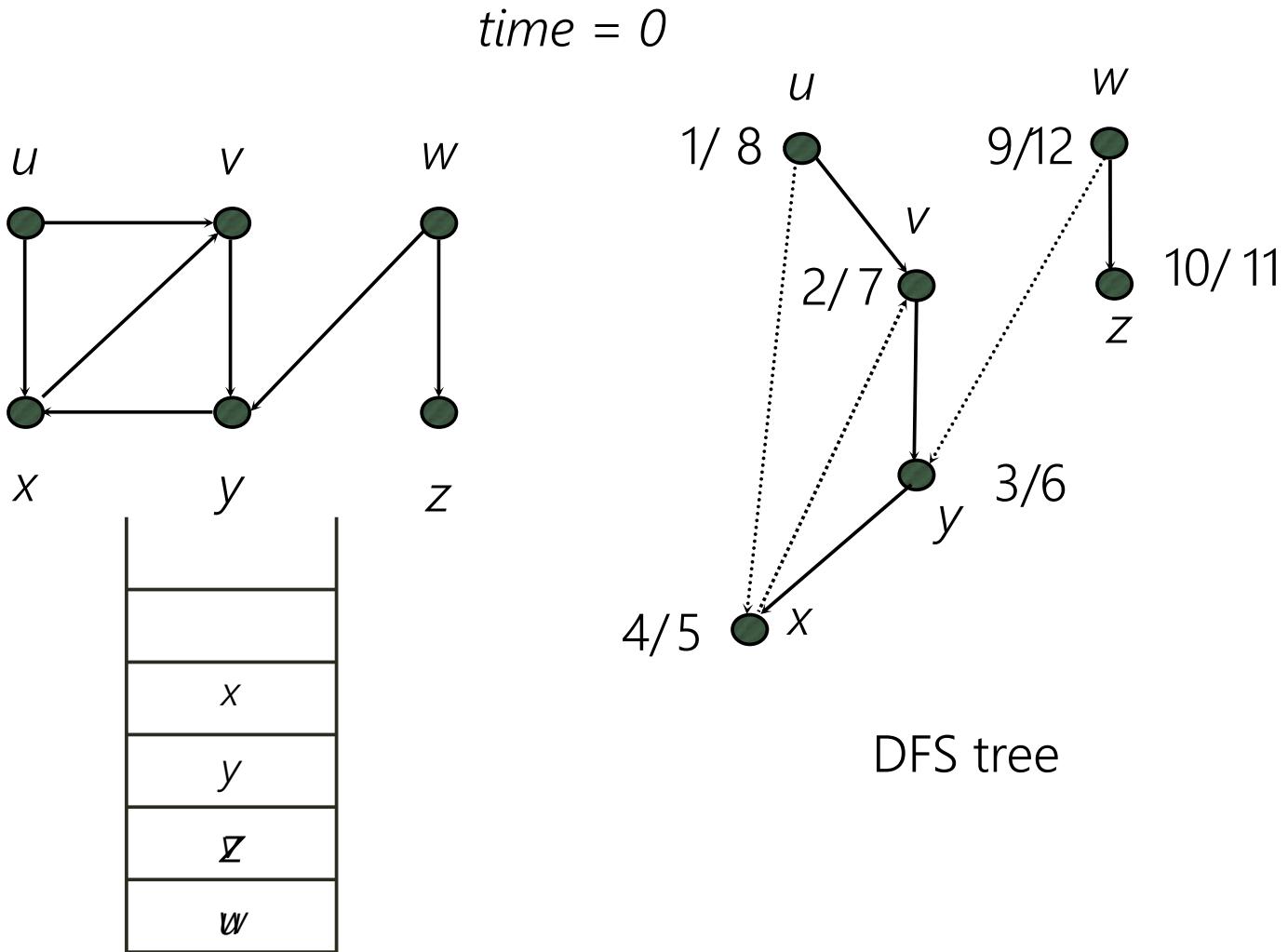
How to figure out back edges?

Where are we?

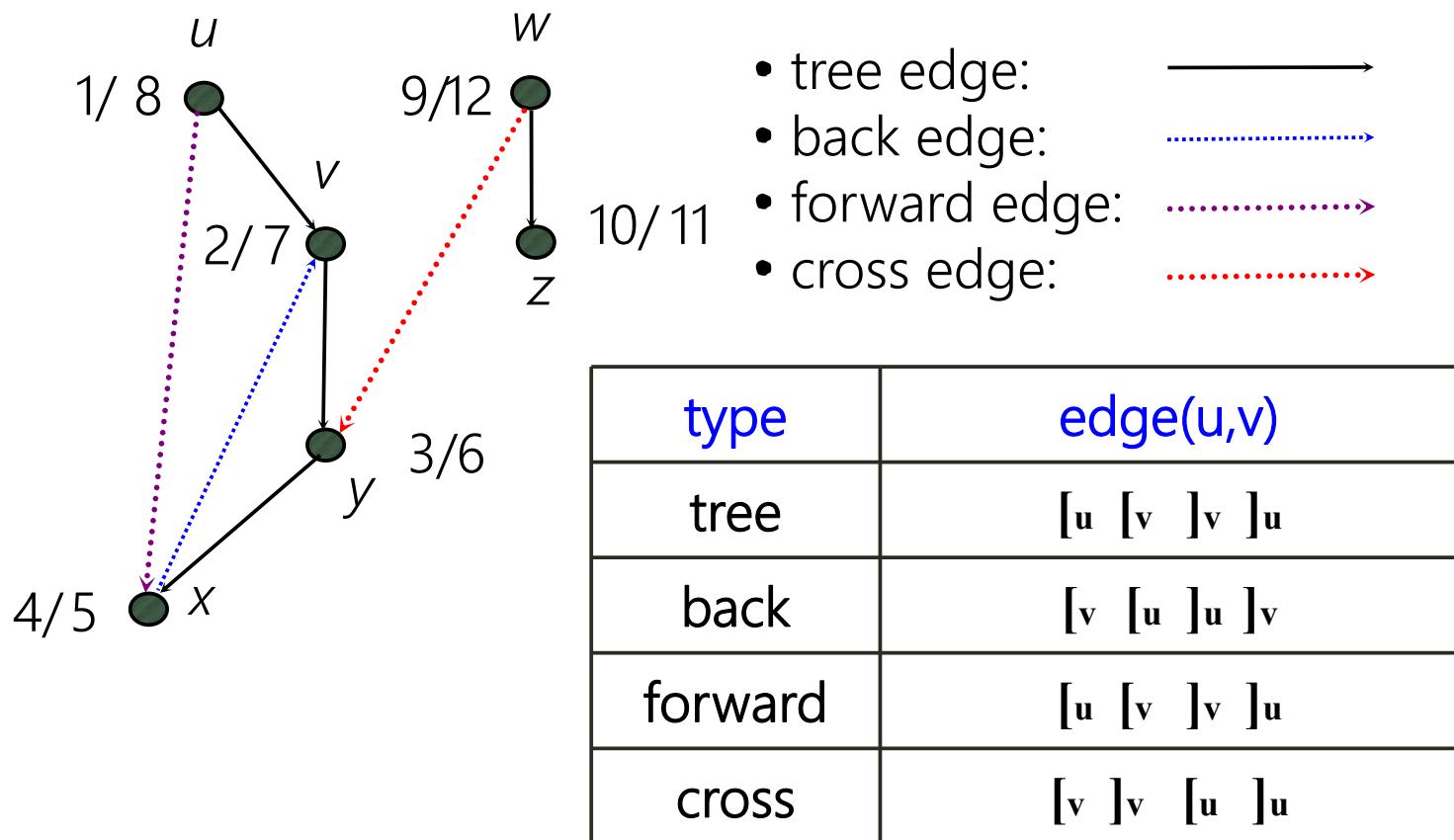
- Graph
- Undirected graph
 - DFS, BFS, Application
- Directed graph
 - DFS, BFS, Application



DFS tree: directed



DFS tree

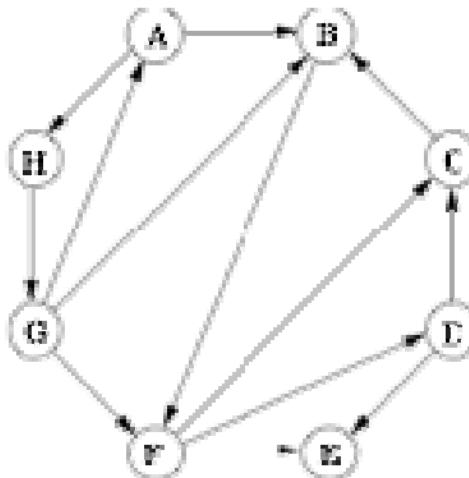


Quiz

Run DFS on the following graph. Which type are the following edges:

CB, DC, FC

(Whenever you have a choice of vertices to explore, always pick the one that is alphabetically first.)



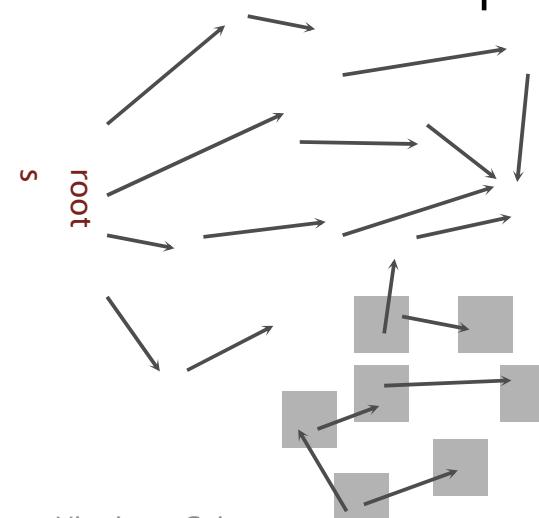
- A. tree edge
- B. back edge
- C. forward edge
- D. cross edge

Application: Garbage collector

Mark-sweep algorithm. [McCarthy, 1960]

- Mark: mark all reachable objects.
- Sweep: if object is unmarked, it is garbage (so add to free list).

Memory cost. Uses 1 extra mark bit per object
(plus DFS stack).



Graph acyclicity

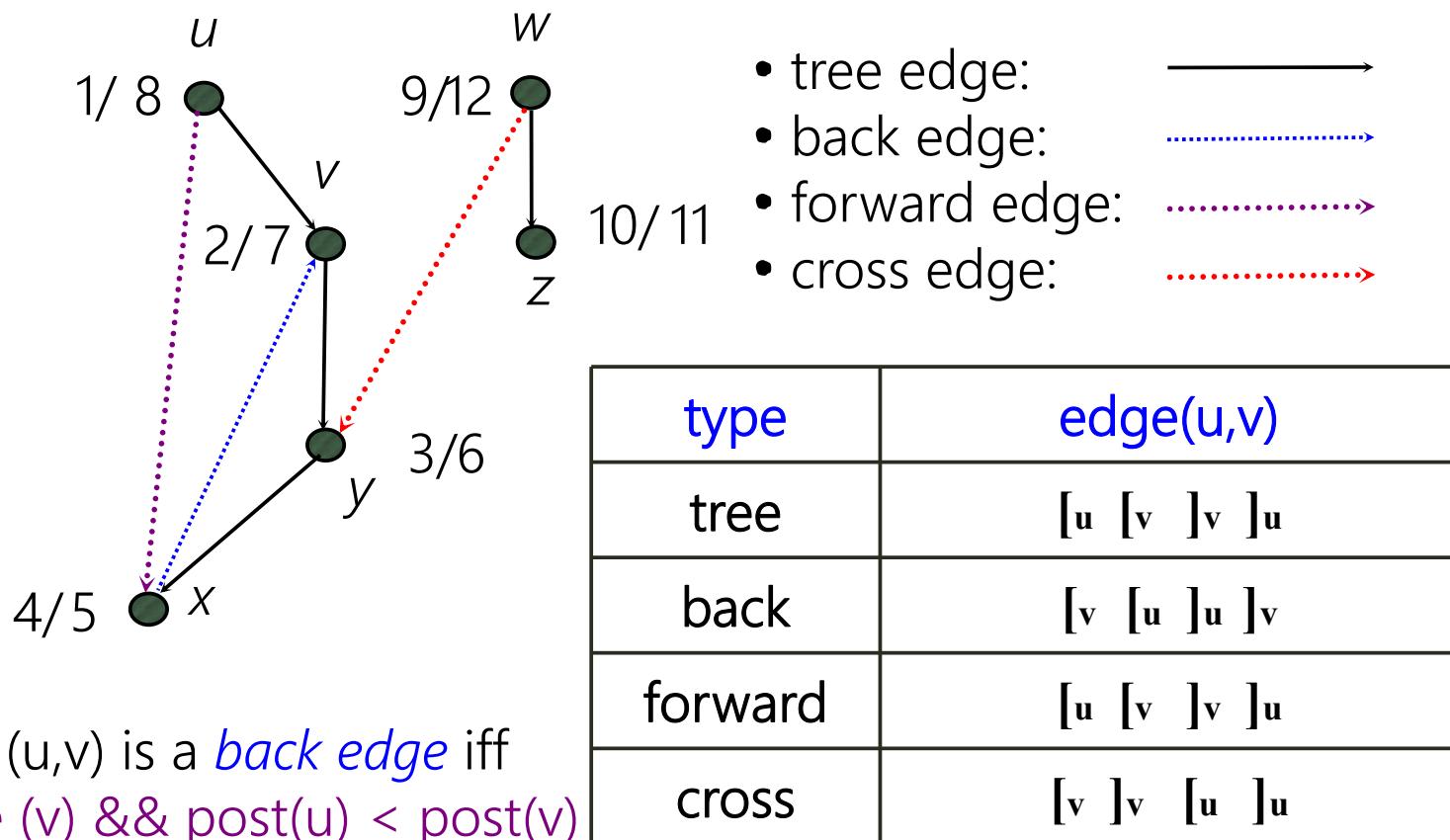
NO back edges!

A directed acyclic graph is usually called a
DAG.



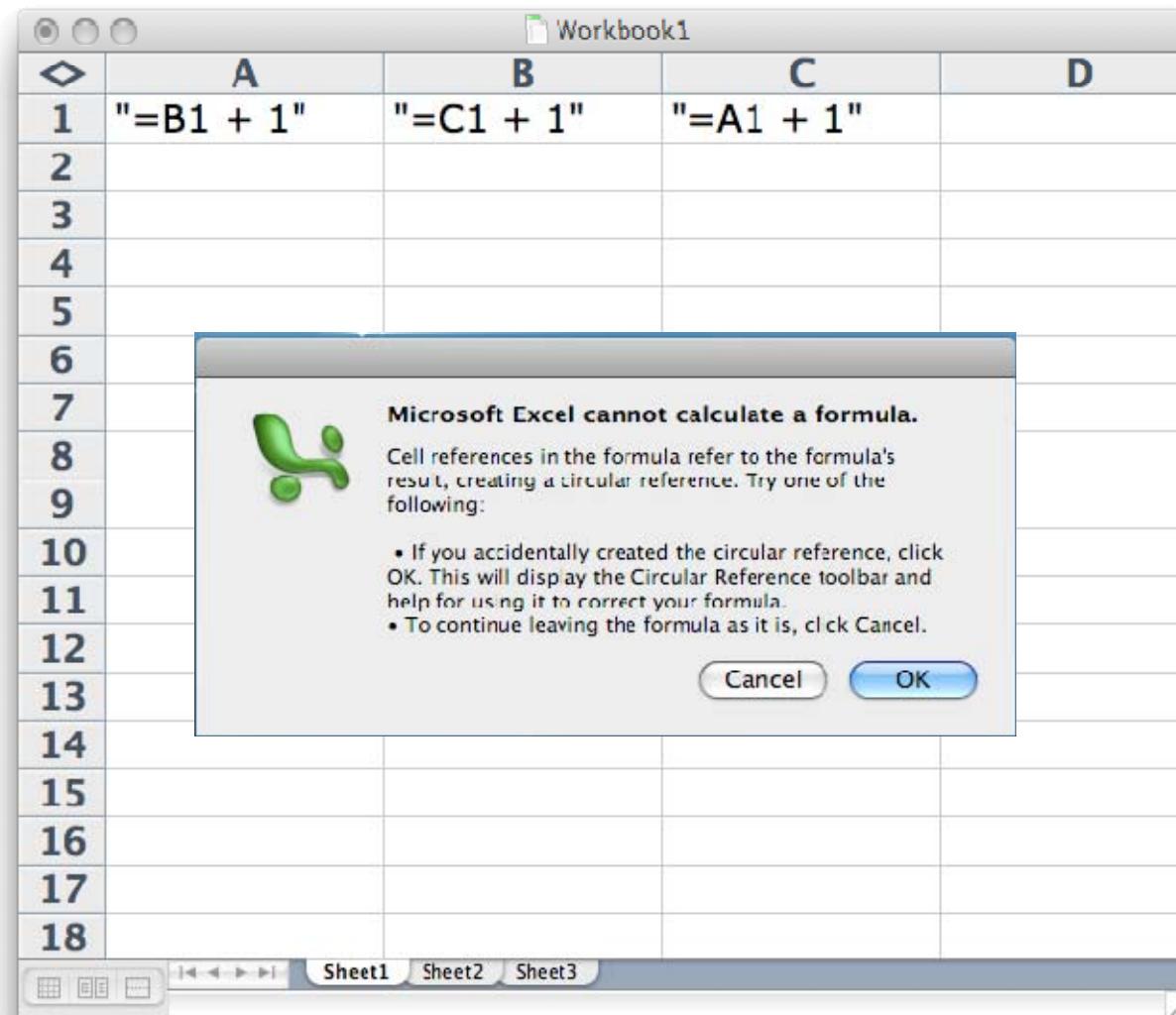
How to figure out back edges?

Graph acyclicity



An edge (u,v) is a *back edge* iff
 $\text{pre}(u) > \text{pre}(v) \text{ && } \text{post}(u) < \text{post}(v)$

Applications

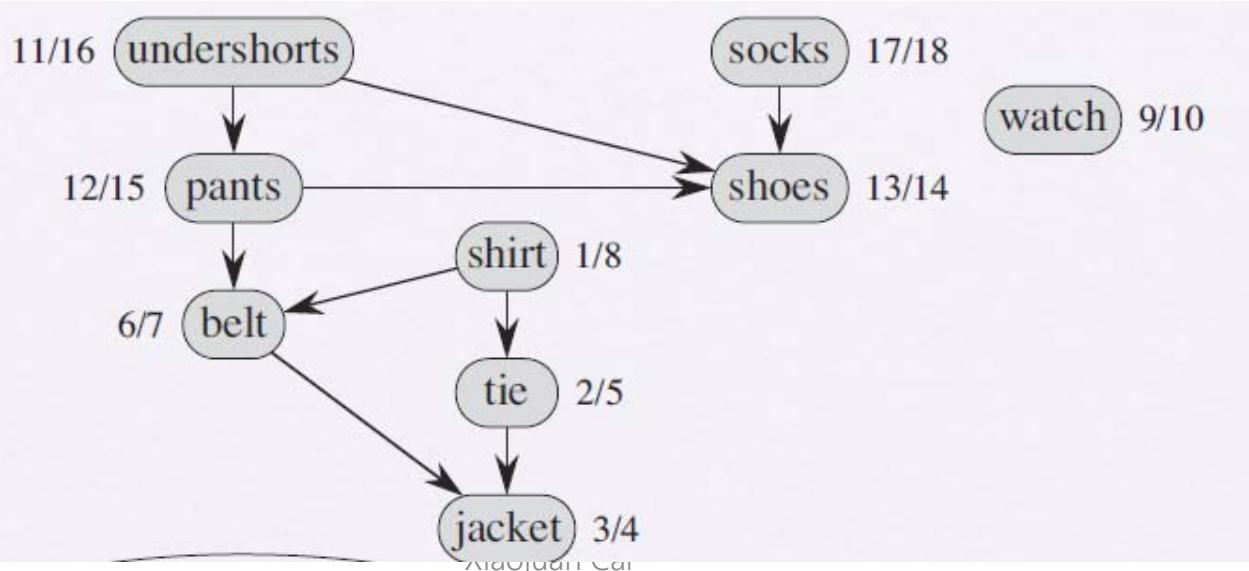


DAG: Topological sort

Problem: TopoSort

Input: A DAG (directed acyclic graph) $G = (V, E)$

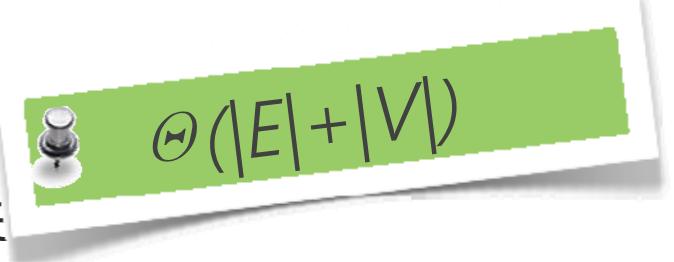
Output: A linear ordering of its vertices in such a way that if $(v, w) \in E$, then v appears before w in the ordering.



DAG: Topological sort

TOPOLOGICAL-SORT(G)

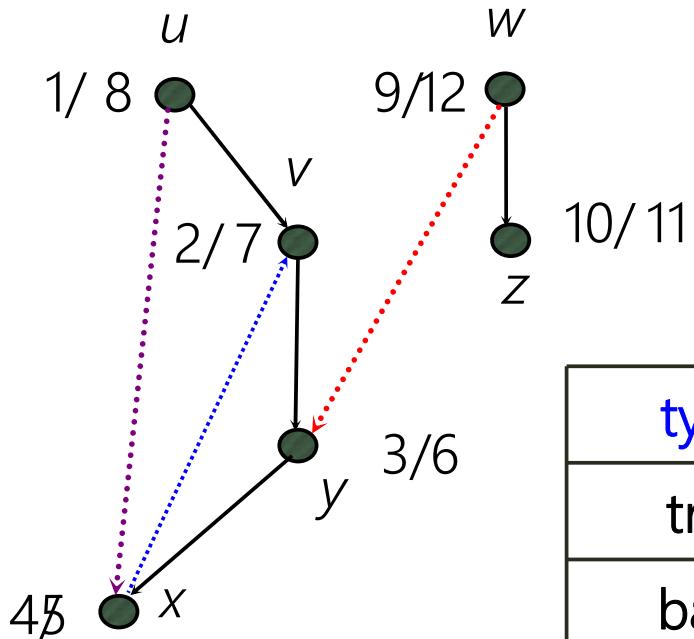
1. Call DFS(G) to compute finishing times of each vertex v
2. As each vertex is finished, insert it onto the front of a linked list
3. Return the linked list of vertices



Correctness

Proposition

For every edge (u,v) , $\text{post}[v] < \text{post}[u]$.



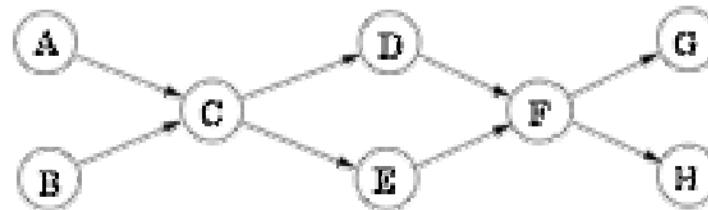
- tree edge:
- back edge:
- forward edge:
- cross edge:

type	edge(u,v)
tree	$[u \ [v \]_v \]_u$
back	$[v \ [u \]_u \]_v$
forward	$[u \ [v \]_v \]_u$
cross	$[v \]_v \ [u \]_u$

Quiz

Give the topological-sort algorithm on the following graph, give the topological order.

(Whenever you have a choice of vertices to explore, always pick the one that is alphabetically first.)



- A. BACEDFGH
- B. ABCDEFGH
- C. ABCEDFHG
- D. None of above

Connectivity

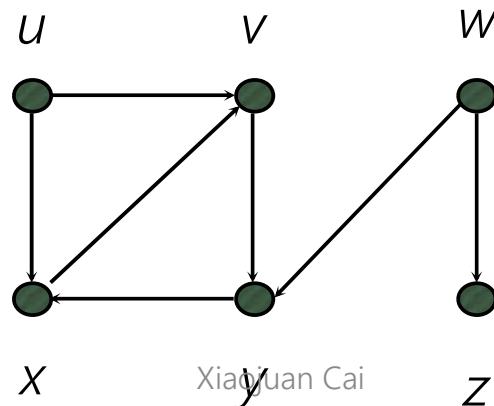
Definition

A directed graph is **connected**, if there is a path from any vertices to another.

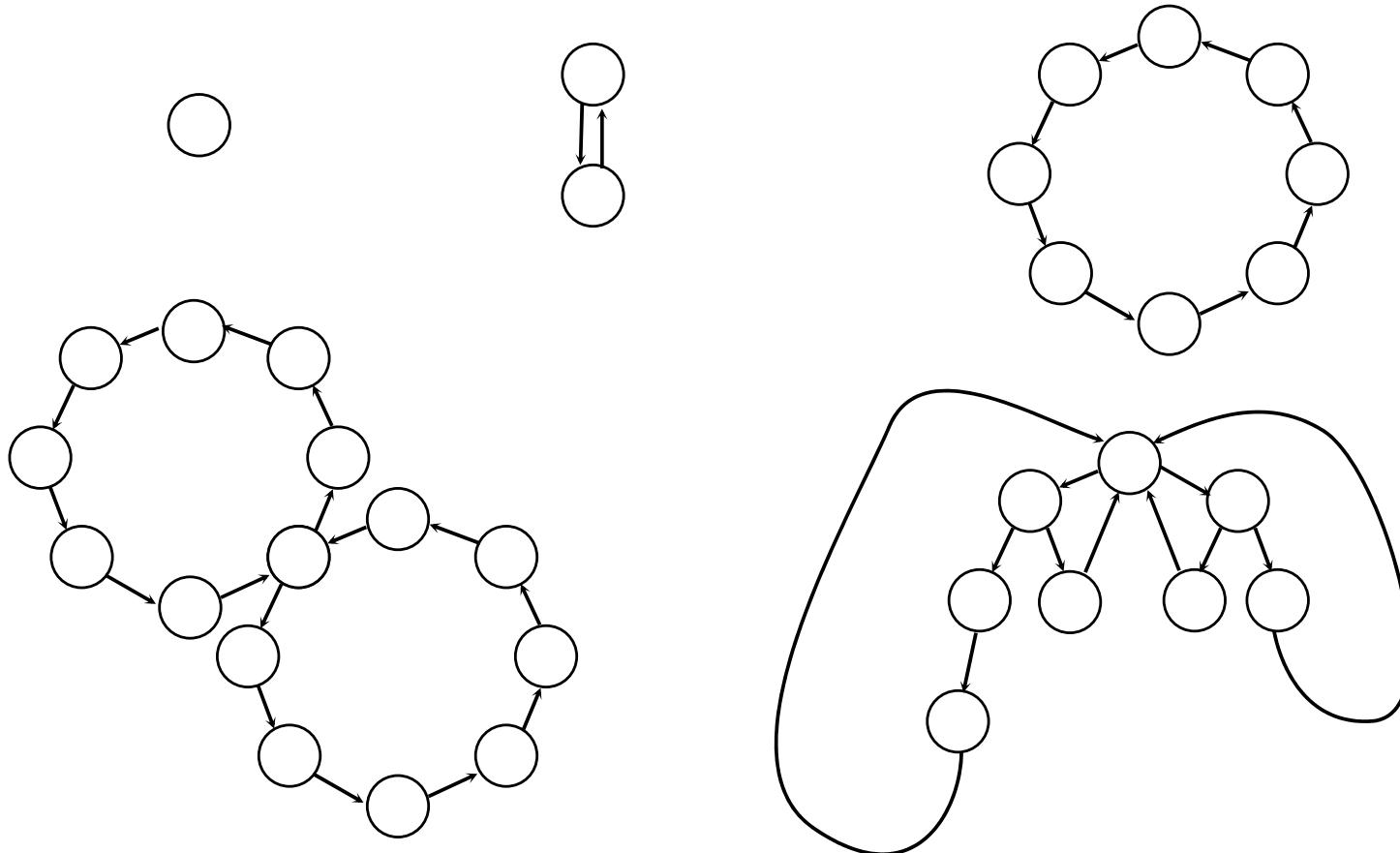
A **strong connected component** is a maximal connected subgraph.

#trees == #connected components

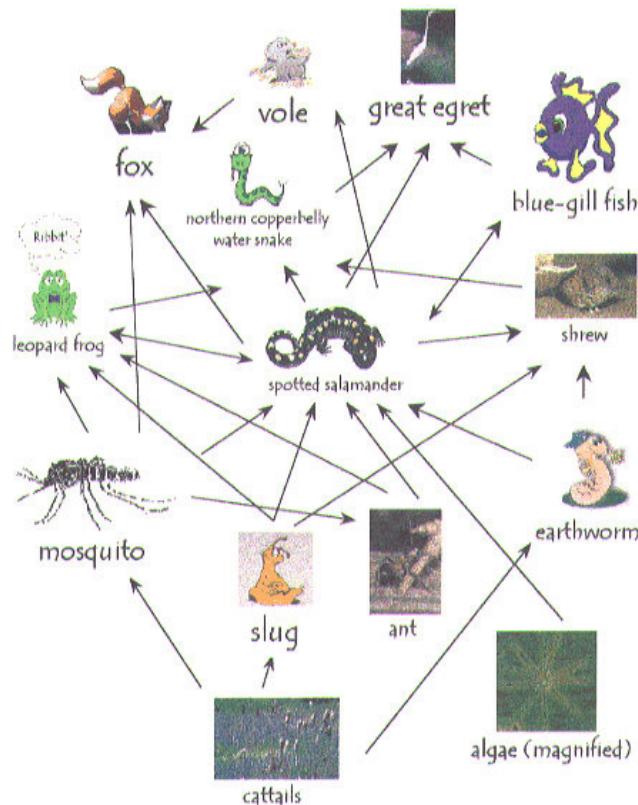
?



Strong connected components

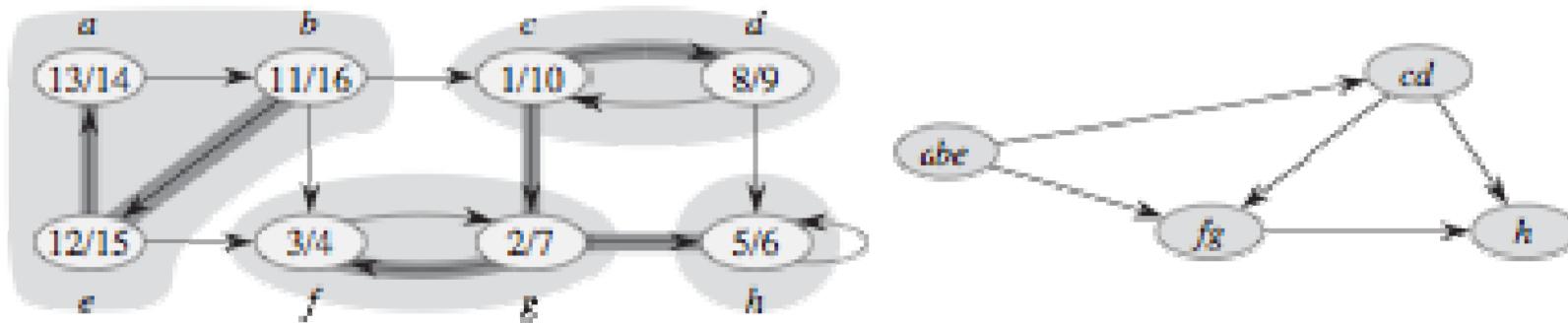


Ecological food webs



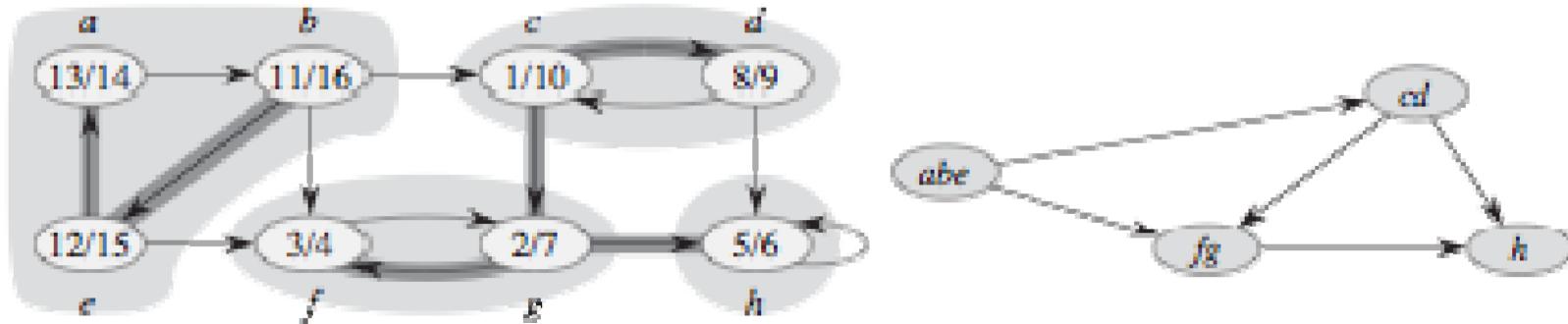
<http://www.twinkl.com/resource/T2-1000-Wetlands-Salamander-SalGraphics/salfoodweb.gif>

Strong connected components



Lemma Every directed graph is a DAG
of its strongly connected component.

Some properties



Property 1

If **dfs** is started at node u , then it will terminate precisely when all nodes reachable from u have been visited.

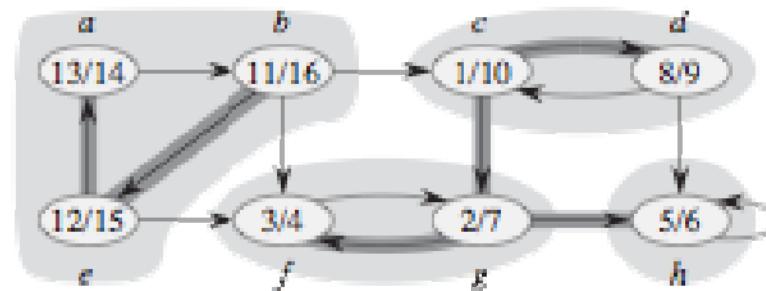
Property 2

The node that receives the highest post number in **DFS** must lie in a source strongly connected component.

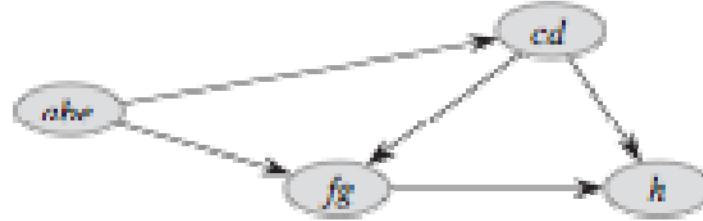
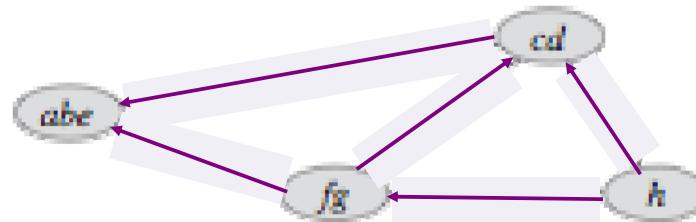
Property 3

If C and C' are scc, and there is an edge from a node in C to a node in C' , then the highest post number in C is bigger than the highest post number in C' .

Kosaraju-Sharir algorithm



Reversed graph



Kosaraju-Sharir algorithm

1. Run DFS on G^R .
2. Run the undirected connected components algorithm on G , and during the DFS, process the vertices in *decreasing order* of their post numbers from step 1.



$$\Theta(|E| + |V|)$$

Tarjan

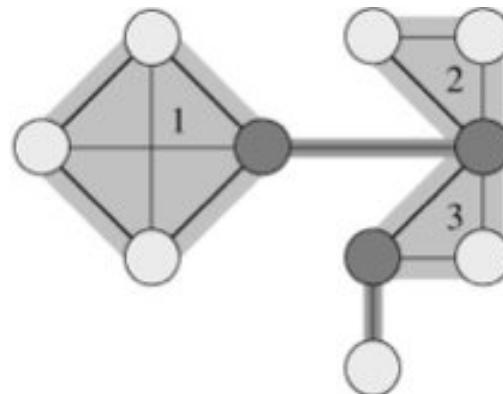
- try all vertex u, if u hasn't been visited, $\text{DFS}(u)$
- $\text{DFS}(u)$, add u to stack, initiate $\text{num}[u]$ and $\text{low}[u]$
- try all neighbor v of u
 - if v is free, $\text{DFS}(v)$
 - update $\text{low}[u]$
 - if $\text{low}[u] == \text{num}[u]$
 - pop from stack until we get u;

Articulation points

Problem: ArticulationPoints

Input: An undirected graph $G = (V, E)$

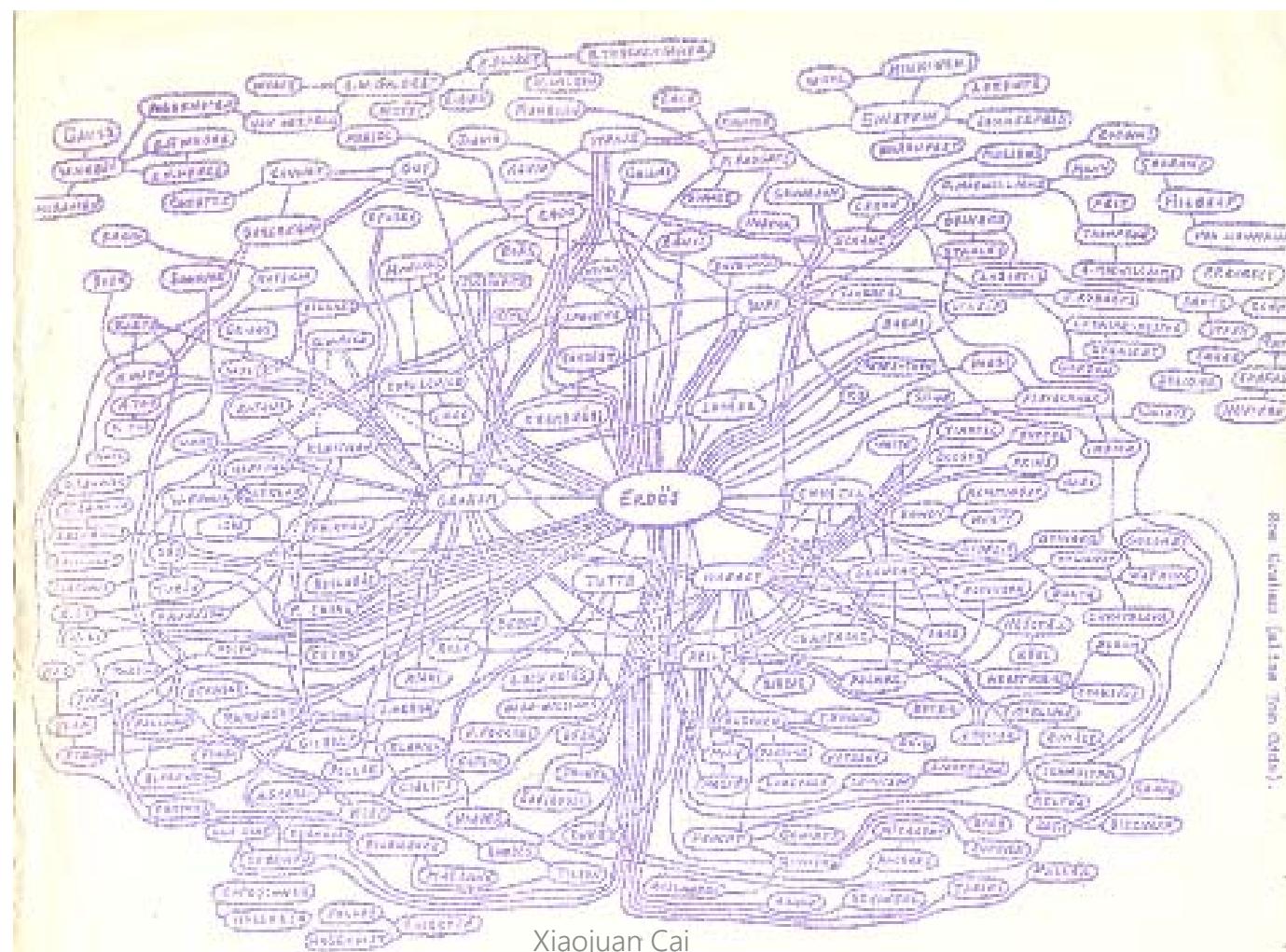
Output: Articulation points. (A point v is *articulate* iff any path between other two points must pass through v).



Conclusion

- Graph
 - Undirected graph
 - DFS, BFS, Application
 - Directed graph
 - DFS, BFS, Application

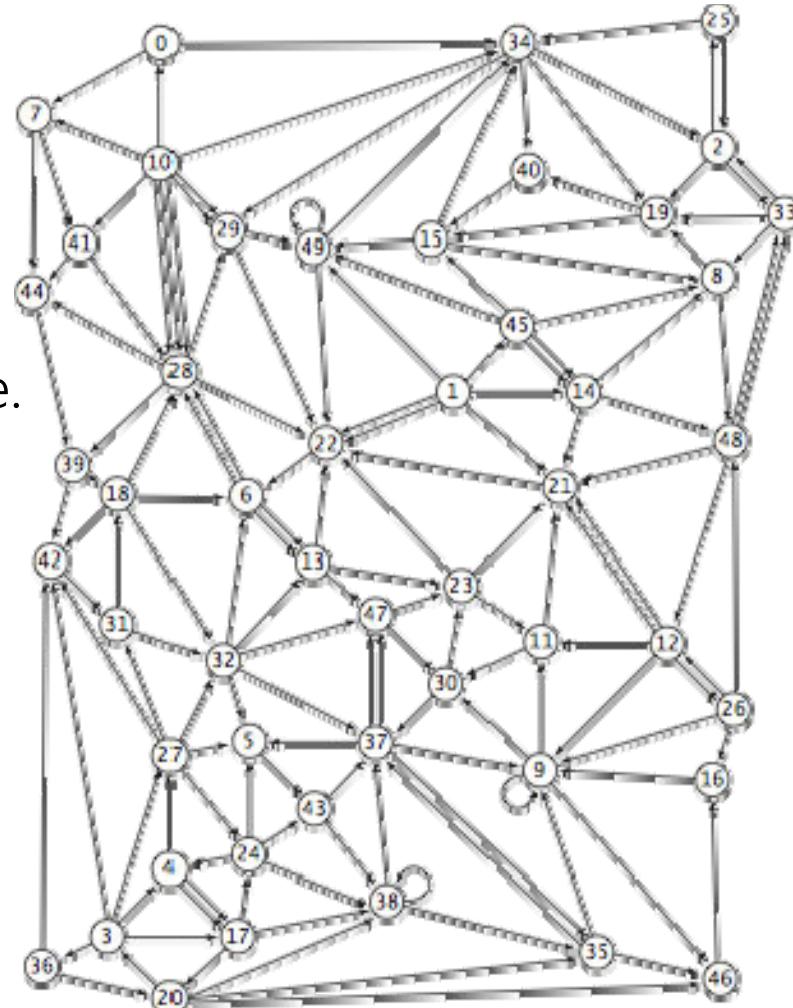
BFS, DFS applications



BFS, DFS applications

BFS.

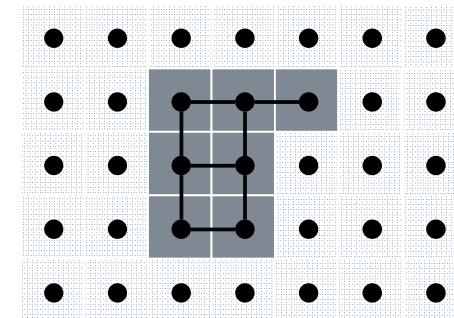
- Choose root web page as source s .
- Maintain a **Queue** of websites to explore.
- Maintain a **SET** of discovered websites.
- Dequeue the next website and enqueue websites to which it links
(provided you haven't done so before).



BFS, DFS applications



- Vertex: pixel.
- Edge: between two adjacent gray pixels.
- Blob: all pixels connected to given pixel.



BFS, DFS applications

Every data structure is a digraph.

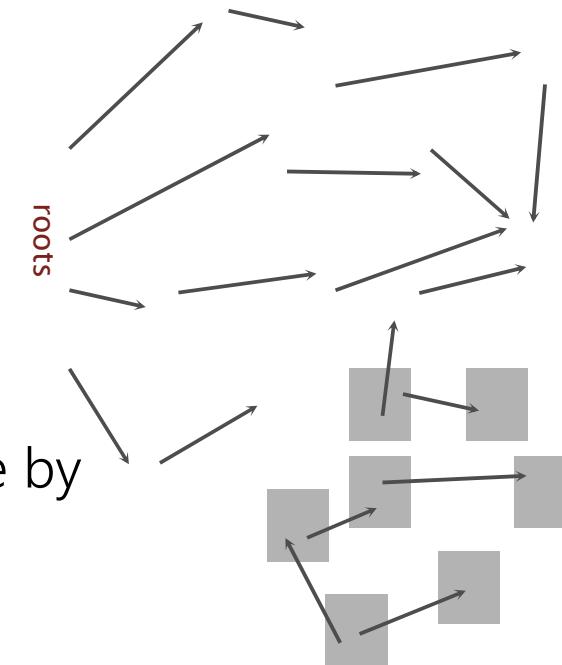
- Vertex = object.
- Edge = reference.

Roots. Objects known to be directly accessible by program (e.g., stack).

Reachable objects. Objects indirectly accessible by program

(starting at a root)

 *DFS needs $O(|V|)$ space.*
 *How to do Mark-sweep with $O(1)$ space?*



BFS, DFS applications

Every program is a digraph.

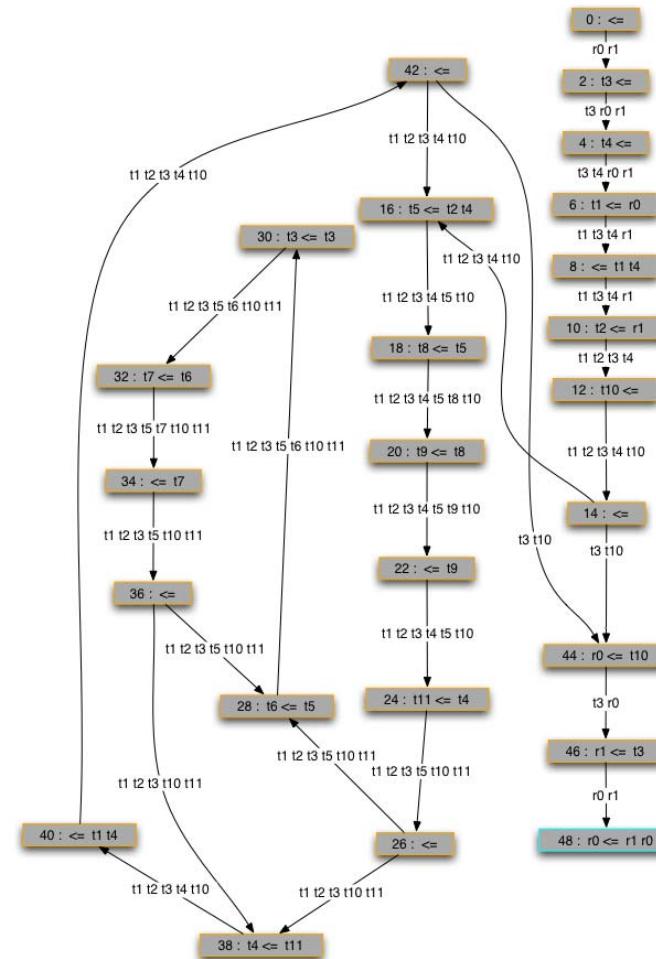
- Vertex = basic block of instructions
- Edge = jump.

Dead-code elimination.

Find (and remove) unreachable code.

Infinite-loop detection.

Determine whether exit is unreachable.



Quiz

Which of the following statements are true for *undirected graph*?

- A. If u is connected to v and v is connected to w , then u is connected to w .
- B. Removing any edge from a connected graph G breaks the graph into two connected components.
- C. An acyclic graph G is connected if and only if it has $V-1$ edges.
- D. If you add two edges to a graph G , its number of components can decrease by at most 2.

Quiz

Which of the following statements are true for *directed graph*?

- A. A digraph on V vertices with fewer than V edges contains more than one strong component.
- B. If we modify the Kosaraju-Sharir algorithm to replace the second DFS with BFS, then it will still find the strong components.
- C. If u is strongly connected to v and v is strongly connected to w , then u is strongly connected to w .
- D. If you add two edges to a digraph, its number of strong components can decrease by at most 2.

Exercise

- Download [hw7.pdf](#) from our course homepage
- Due on Oct. 28

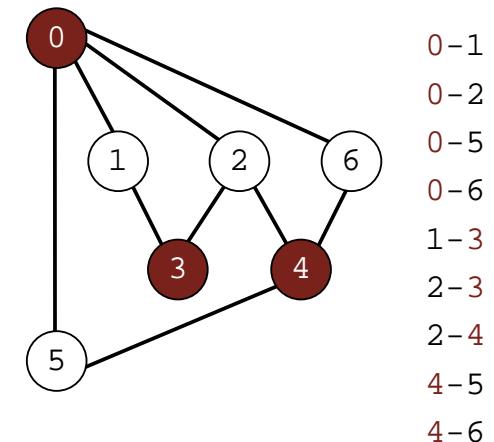
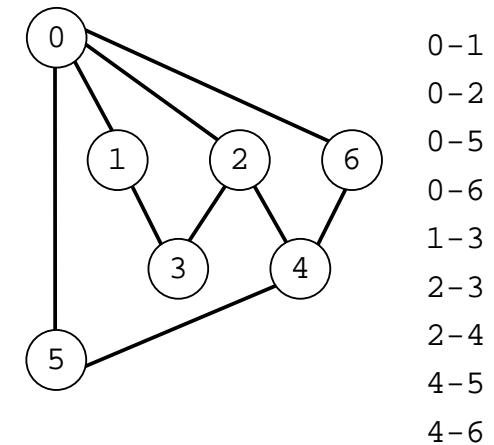
Challenges

Challenge 1

Problem. Is a graph bipartite?

How difficult?

- Any programmer could do it.
- Need to be a typical diligent SE222 student.
- Hire an expert.
- Intractable.
- No one knows.
- Impossible.

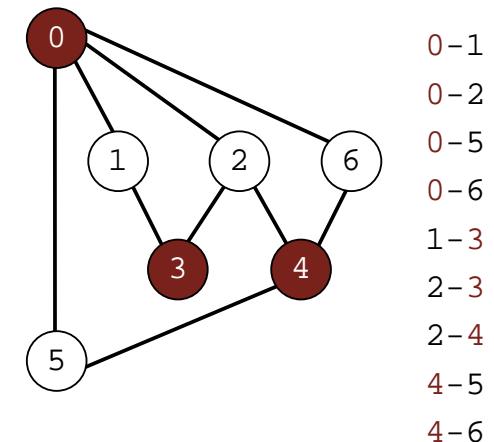
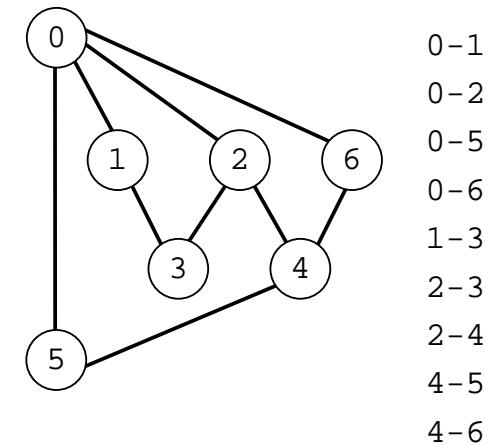


Challenge 1

Problem. Is a graph bipartite?

How difficult?

- Any programmer could do it.
- Need to be a typical diligent SE222 student. ✓
- Hire an expert.
- Intractable.
- No one knows.
- Impossible.

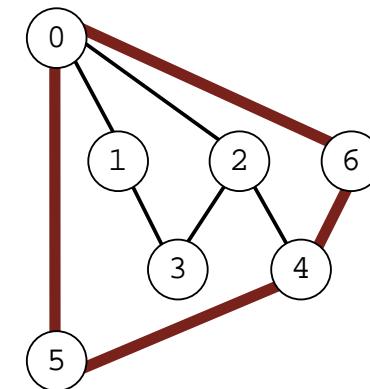
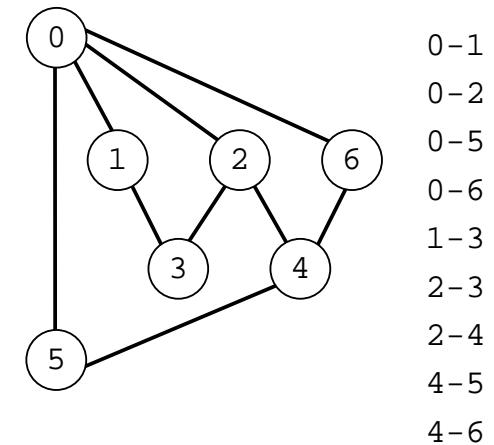


Challenge 2

Problem. Find a cycle.

How difficult?

- Any programmer could do it.
- Need to be a typical diligent SE222 student. ✓
- Hire an expert.
- Intractable.
- No one knows.
- Impossible.

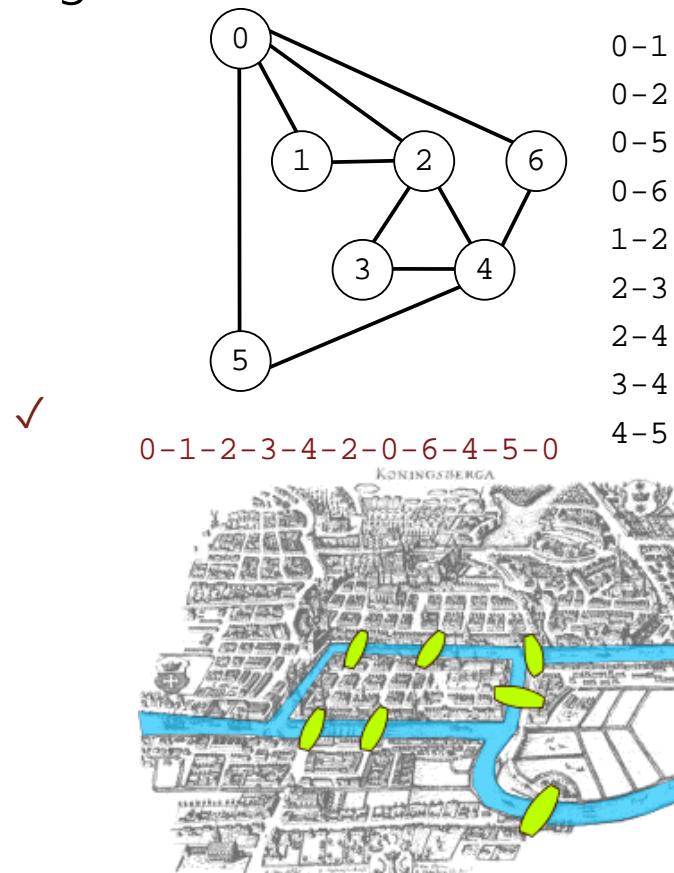


Challenge 3

Problem. Find a cycle that uses every edge exactly once. (Eulerian tour)

How difficult?

- Any programmer could do it.
- Need to be a typical diligent SE222 student.
- Hire an expert.
- Intractable.
- No one knows.
- Impossible.

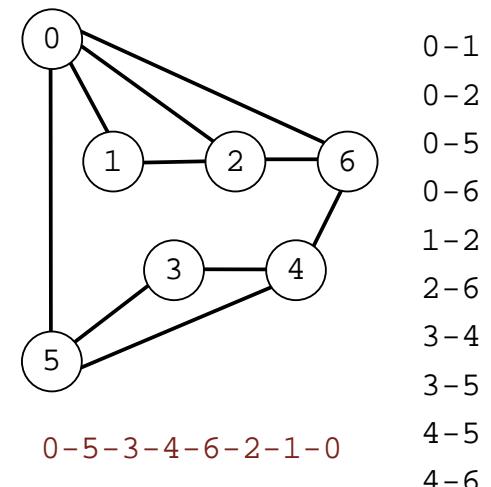


Challenge 4

Problem. Find a cycle that uses every vertex exactly once. (Hamiltonian tour)

How difficult?

- Any programmer could do it.
- Need to be a typical diligent SE222 student.
- Hire an expert.
- Intractable. ✓
- No one knows.
- Impossible.

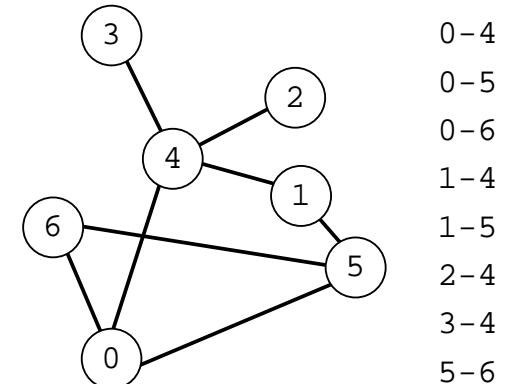
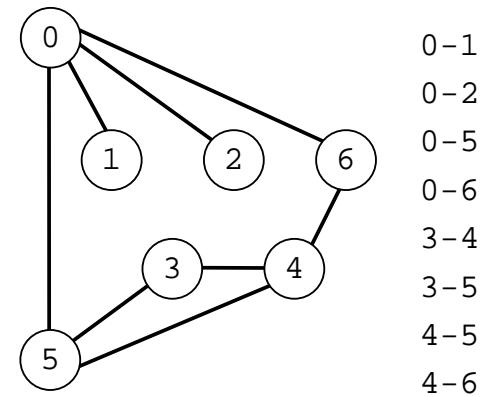


Challenge 5

Problem. Are two graphs identical except for vertex names? (Graph isomorphism)

How difficult?

- Any programmer could do it.
- Need to be a typical diligent SE222 student.
- Hire an expert.
- Intractable.
- No one knows. ✓
- Impossible.



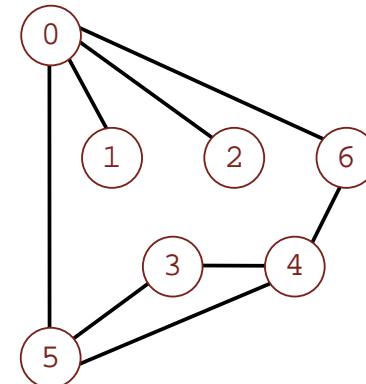
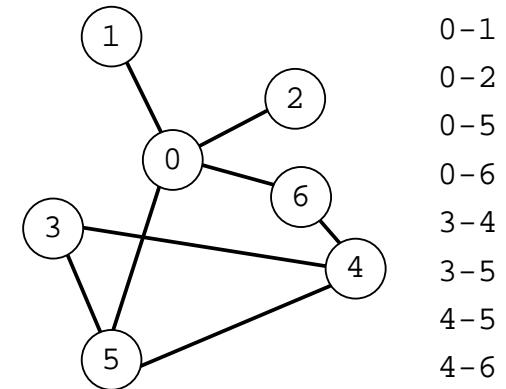
0□4, 1□3, 2□2, 3□6, 4□5, 5□0, 6□1

Challenge 6

Problem. Lay out a graph in the plane without crossing edges?

How difficult?

- Any programmer could do it.
- Need to be a typical diligent SE222 student.
- Hire an expert. ✓
- Intractable.
- No one knows.
- Impossible.



4.1 Breadth-First Search

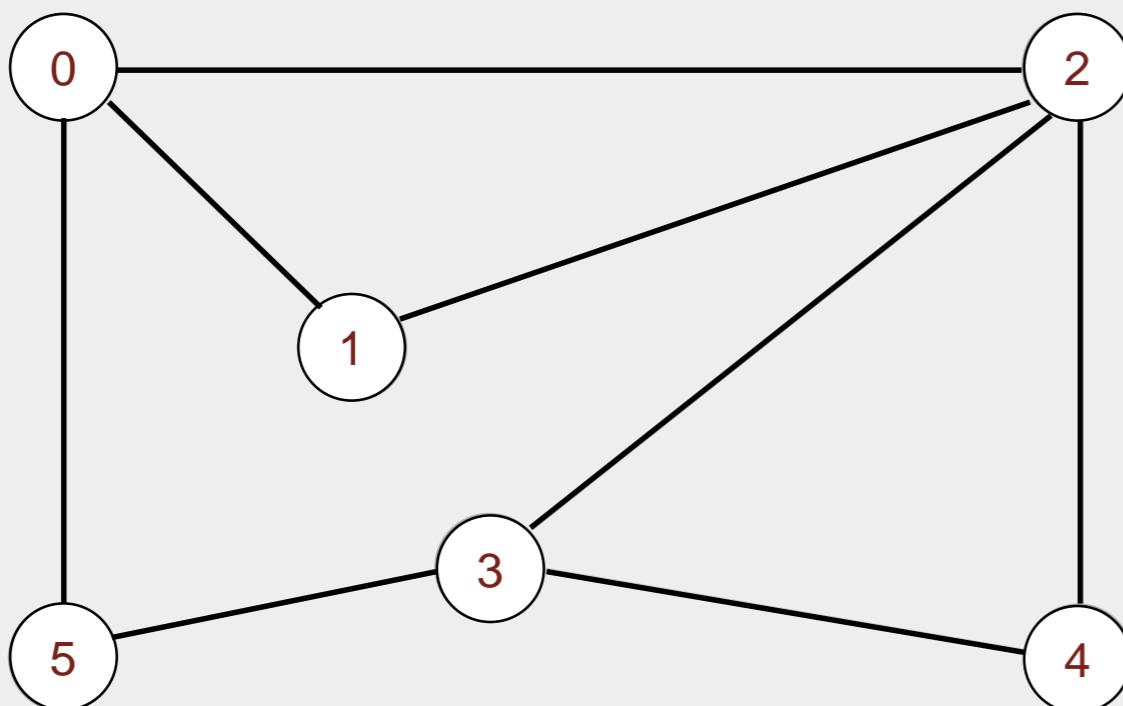


click to begin demo

Breadth-first search

Repeat until queue is empty:

- Remove vertex v from queue.
- Add to queue all unmarked vertices adjacent to v and mark them.



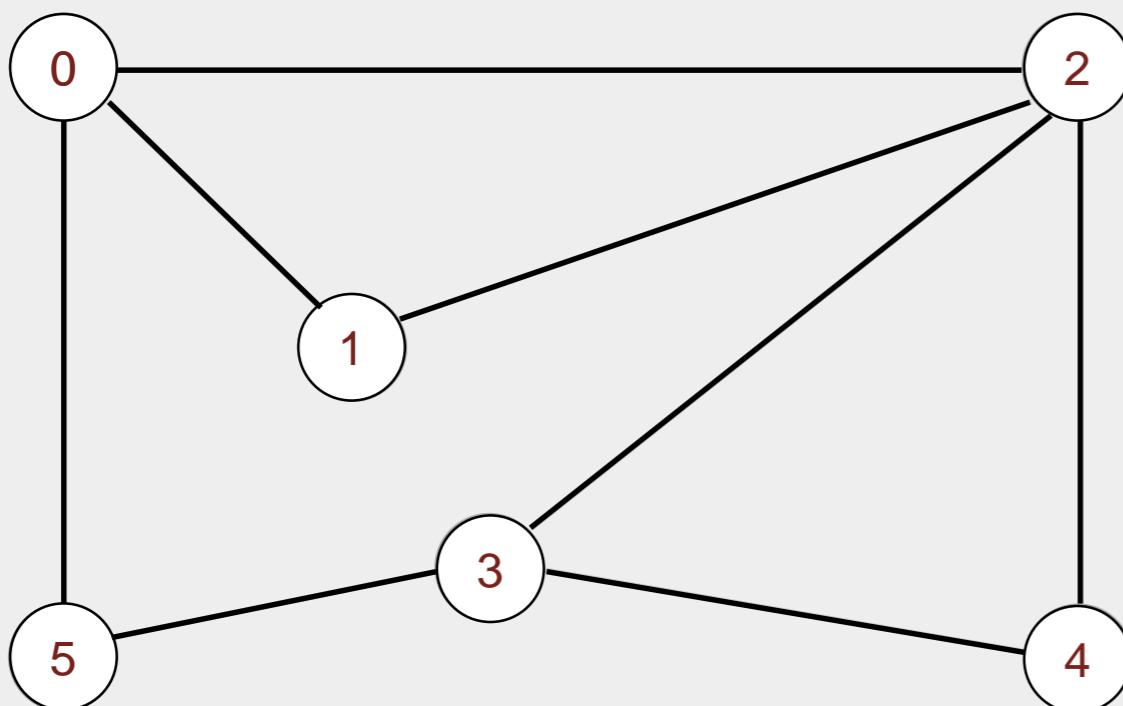
tinyG6.txt	
V	6
E	6
0	5
1	4
2	3
3	2
4	1
5	0

graph G

Breadth-first search

Repeat until queue is empty:

- Remove vertex v from queue.
- Add to queue all unmarked vertices adjacent to v and mark them.



queue



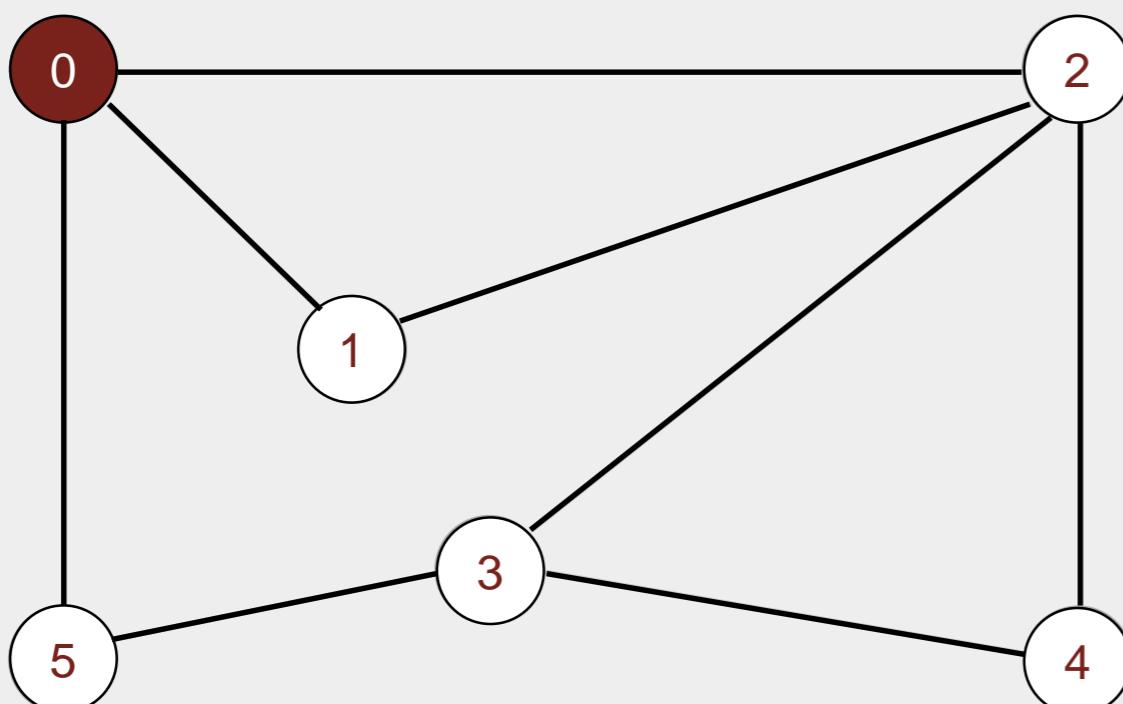
v	edgeTo[v]
0	-
1	-
2	-
3	-
4	-
5	-

add 0 to queue

Breadth-first search

Repeat until queue is empty:

- Remove vertex v from queue.
- Add to queue all unmarked vertices adjacent to v and mark them.



queue

v
0
1
2
3
4
5

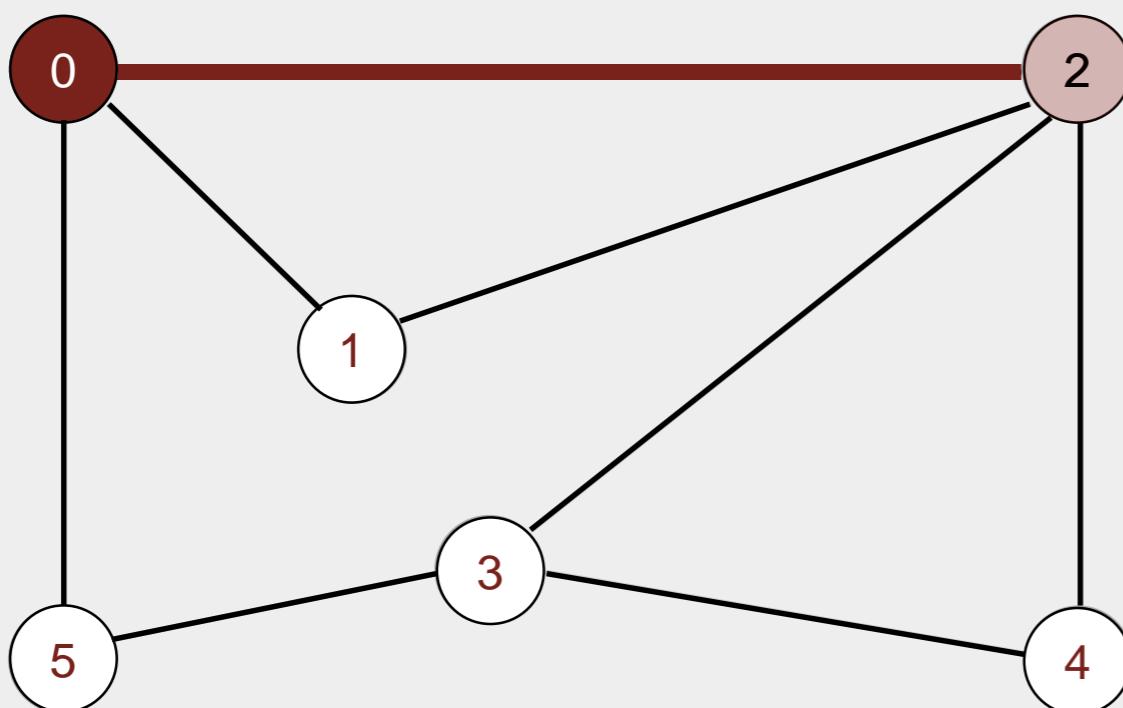
v	edgeTo[v]
0	-
1	-
2	-
3	-
4	-
5	-

dequeue 0

Breadth-first search

Repeat until queue is empty:

- Remove vertex v from queue.
- Add to queue all unmarked vertices adjacent to v and mark them.



queue

v	edgeTo[v]
0	-
1	0
2	-
3	-
4	-
5	-

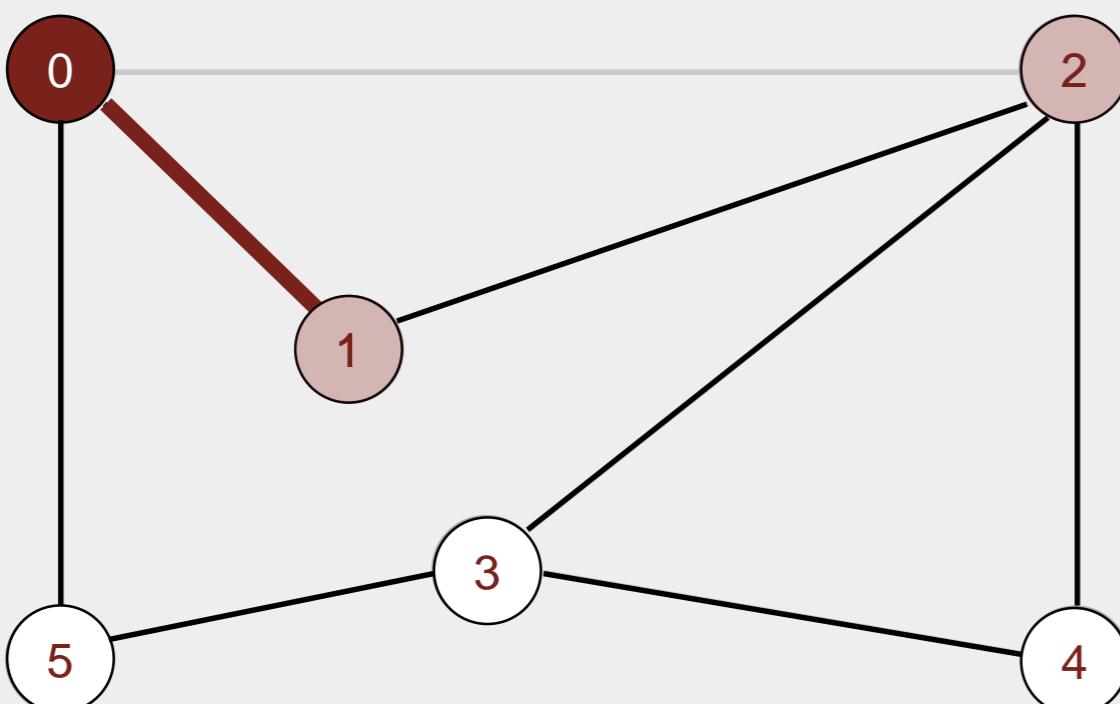
v	edgeTo[v]
0	-
1	0
2	-
3	-
4	-
5	-

dequeue 0

Breadth-first search

Repeat until queue is empty:

- Remove vertex v from queue.
- Add to queue all unmarked vertices adjacent to v and mark them.



queue

v	edgeTo[v]
0	-
1	-
2	0
3	-
4	-
5	-

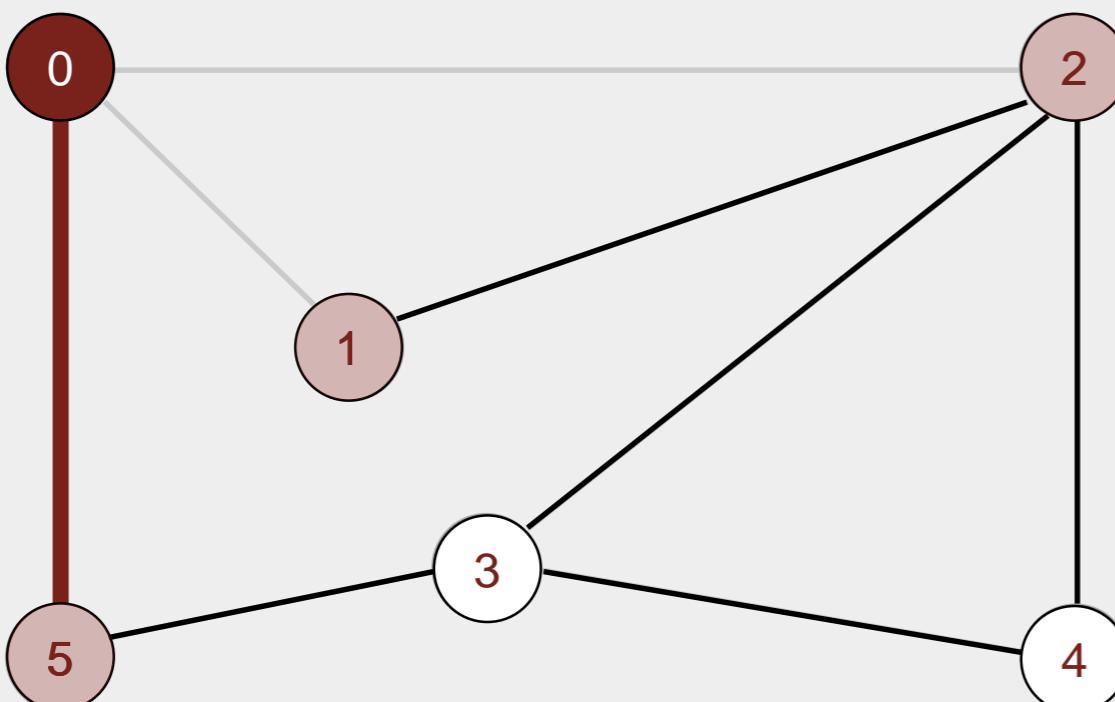
v	edgeTo[v]
0	-
1	-
2	0
3	-
4	-
5	-

dequeue 0

Breadth-first search

Repeat until queue is empty:

- Remove vertex v from queue.
- Add to queue all unmarked vertices adjacent to v and mark them.



queue

v
1
2

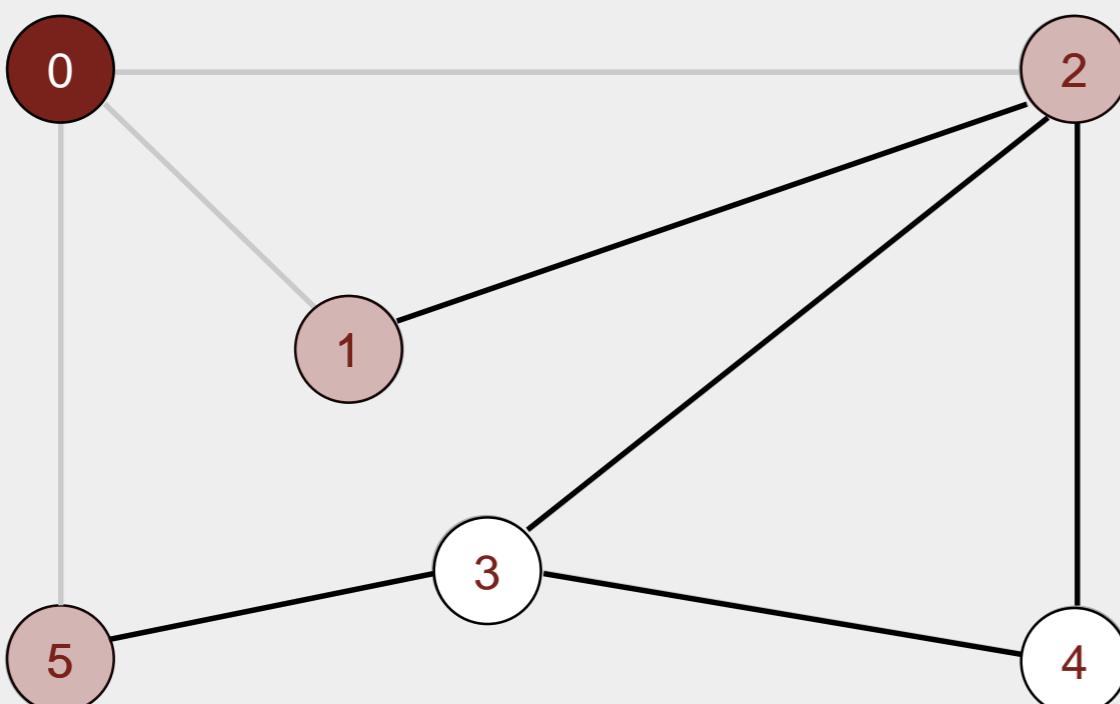
v	edgeTo[v]
0	-
1	0
2	0
3	-
4	0
5	-

dequeue 0

Breadth-first search

Repeat until queue is empty:

- Remove vertex v from queue.
- Add to queue all unmarked vertices adjacent to v and mark them.



queue

v
5
1
4
5
2

v edgeTo[v]

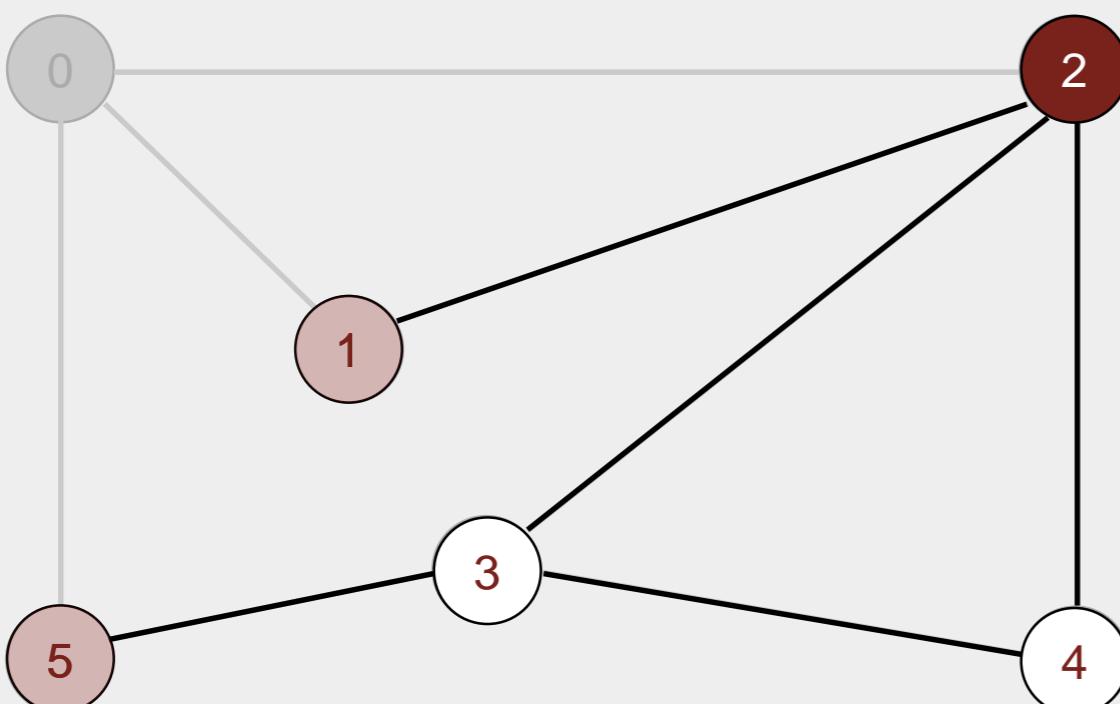
v	edgeTo[v]
0	-
1	0
2	0
3	-
4	-
5	0

0 done

Breadth-first search

Repeat until queue is empty:

- Remove vertex v from queue.
- Add to queue all unmarked vertices adjacent to v and mark them.



queue

v
5
1
2
3
4
5

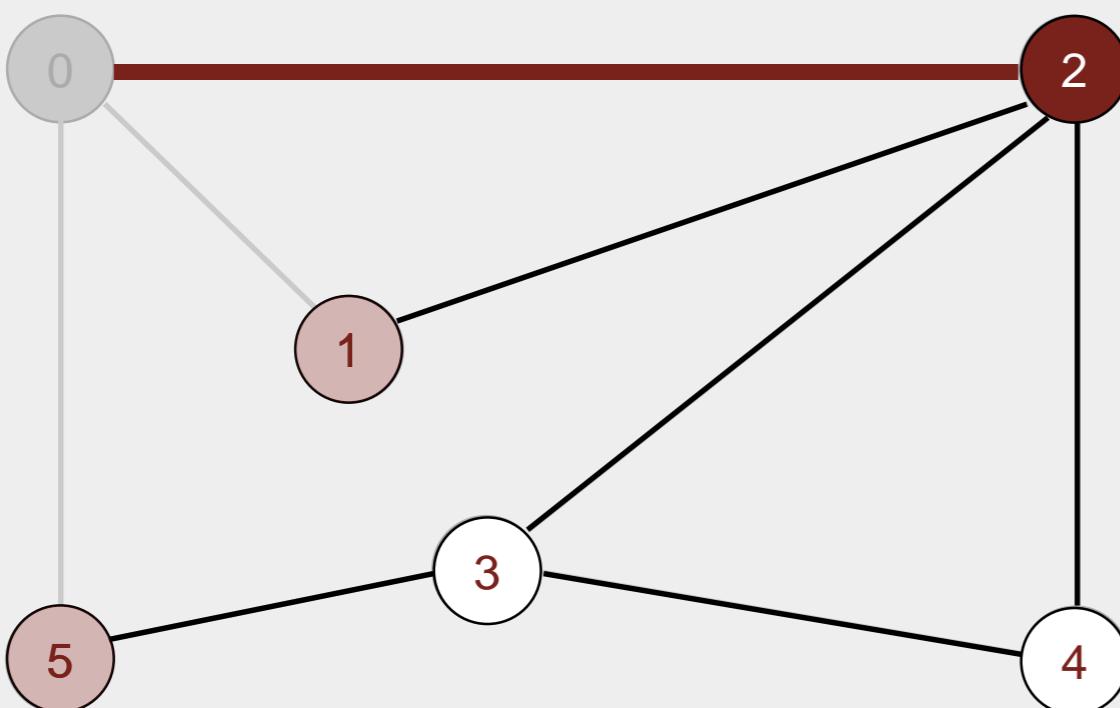
v	edgeTo[v]
0	-
1	0
2	0
3	-
4	-
5	0

dequeue 2

Breadth-first search

Repeat until queue is empty:

- Remove vertex v from queue.
- Add to queue all unmarked vertices adjacent to v and mark them.



queue

v
5
1

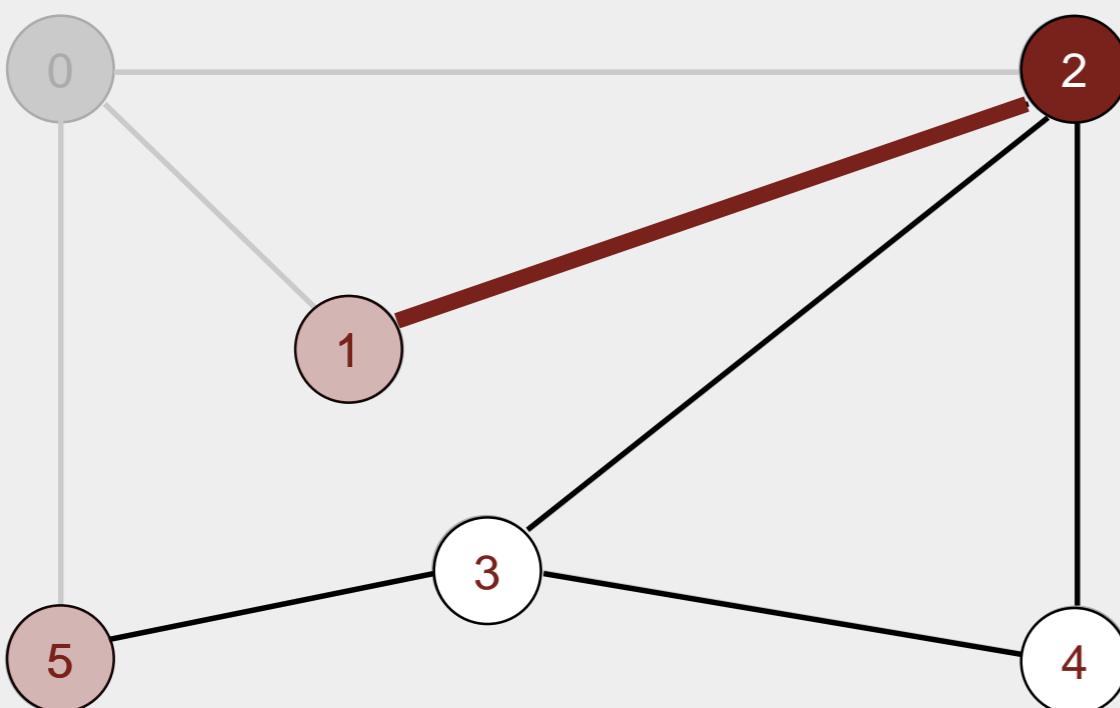
v	edgeTo[v]
0	-
1	0
2	0
3	-
4	-
5	0

dequeue 2

Breadth-first search

Repeat until queue is empty:

- Remove vertex v from queue.
- Add to queue all unmarked vertices adjacent to v and mark them.



queue

v
5
1

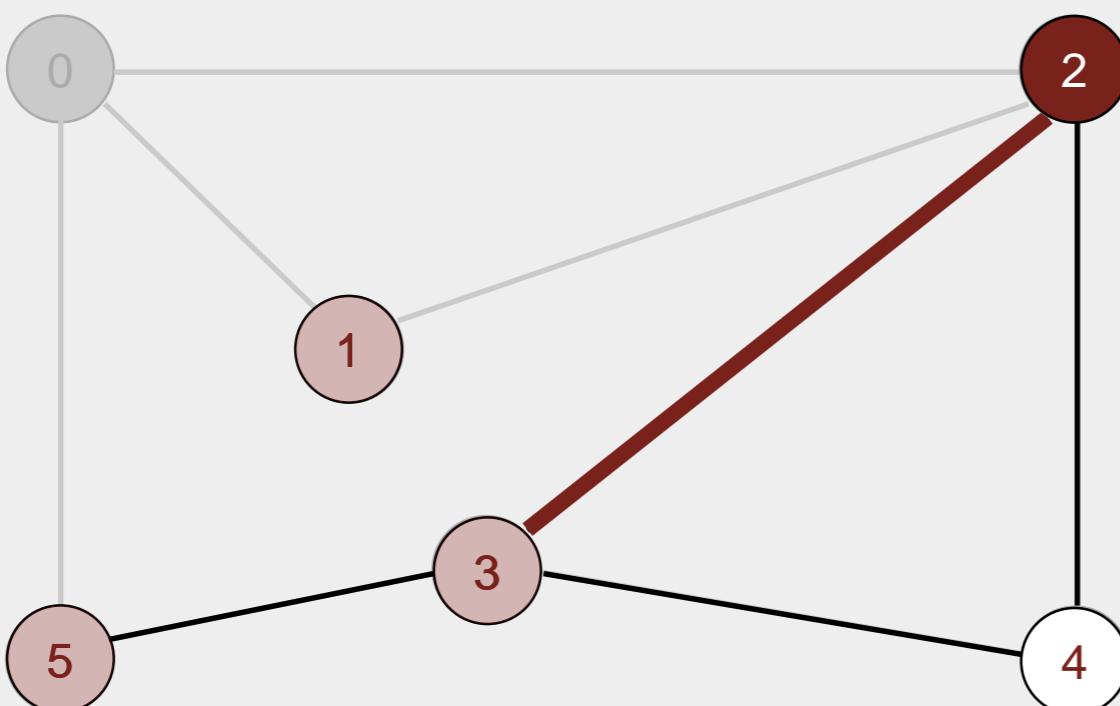
v	edgeTo[v]
0	-
1	0
2	0
3	-
4	-
5	0

dequeue 2

Breadth-first search

Repeat until queue is empty:

- Remove vertex v from queue.
- Add to queue all unmarked vertices adjacent to v and mark them.



queue

v
5
1

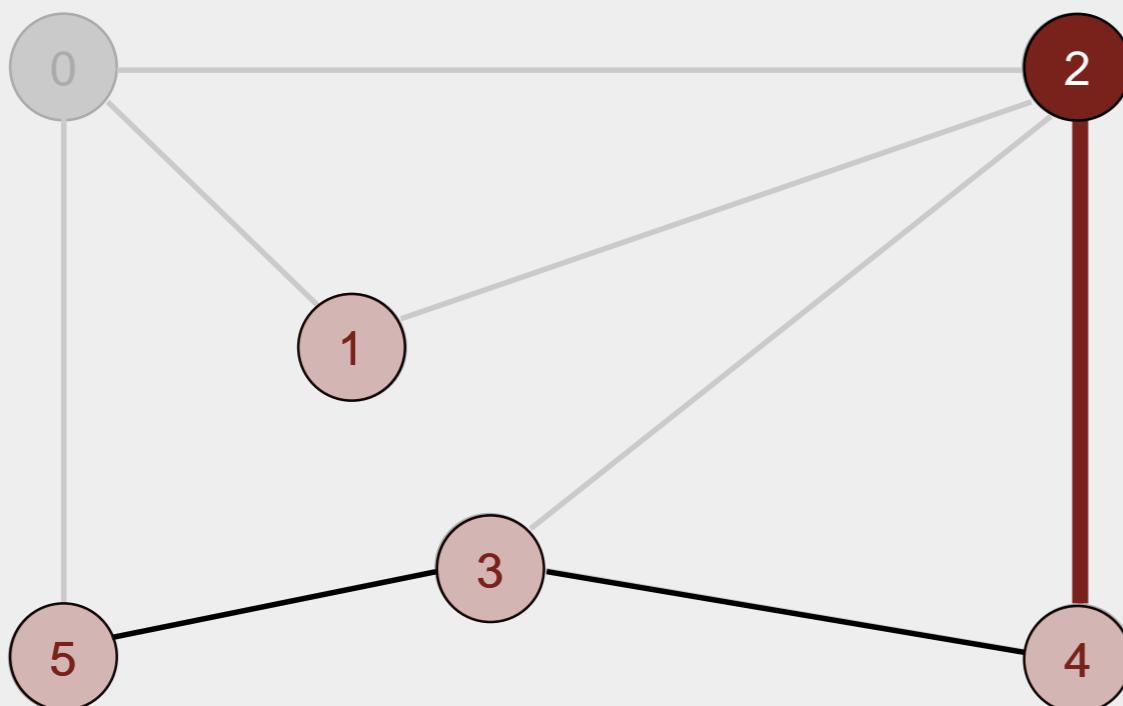
v	edgeTo[v]
0	-
1	0
2	2
3	-
4	-
5	0

dequeue 2

Breadth-first search

Repeat until queue is empty:

- Remove vertex v from queue.
- Add to queue all unmarked vertices adjacent to v and mark them.



queue

v
0
1
2
3
4
5

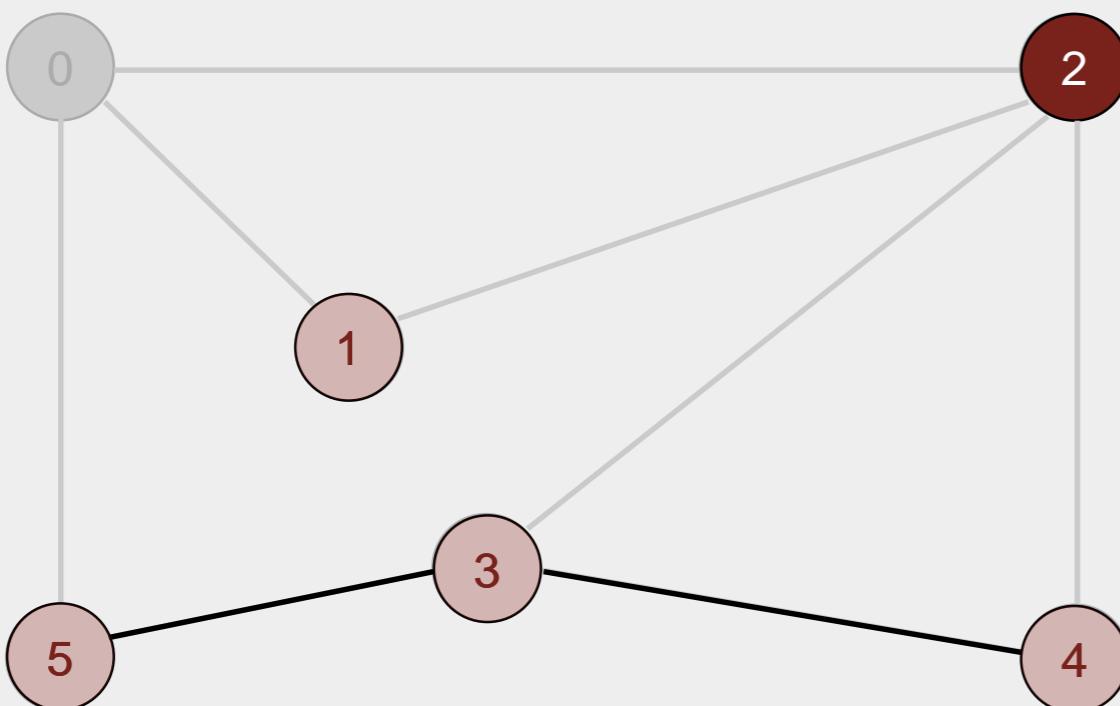
v	edgeTo[v]
0	-
1	0
2	0
3	2
4	-
5	0

dequeue 2

Breadth-first search

Repeat until queue is empty:

- Remove vertex v from queue.
- Add to queue all unmarked vertices adjacent to v and mark them.



queue

v
4
3
5
1

v edgeTo[v]

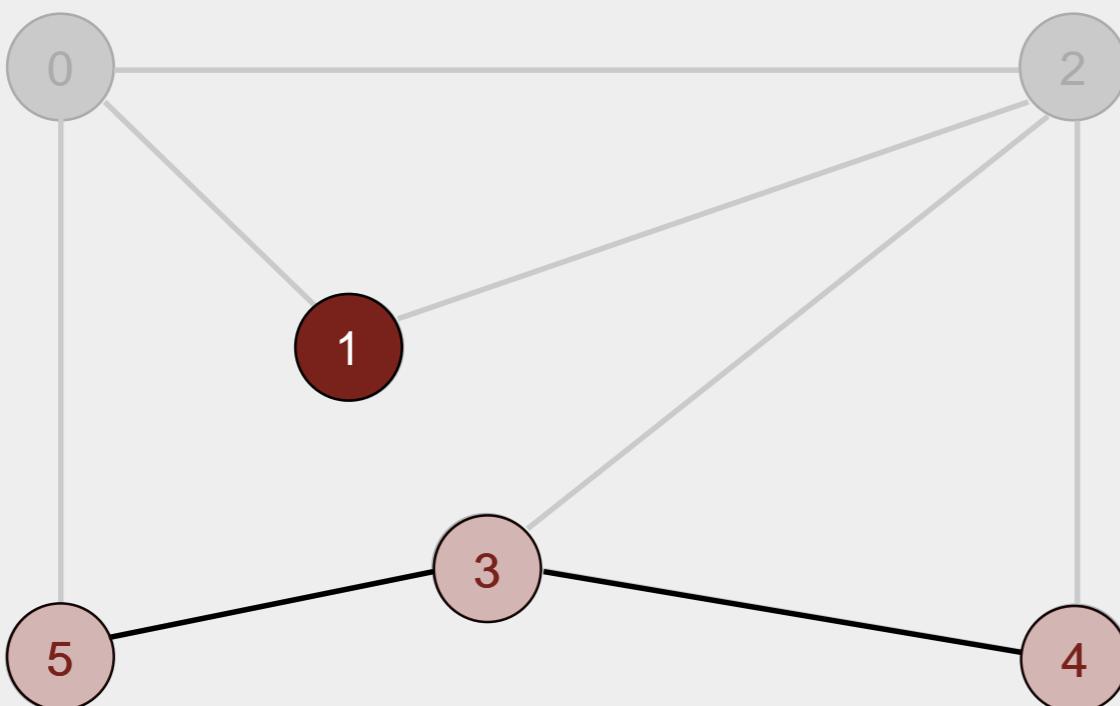
v	edgeTo[v]
0	-
1	0
2	0
3	2
4	2
5	0

2 done

Breadth-first search

Repeat until queue is empty:

- Remove vertex v from queue.
- Add to queue all unmarked vertices adjacent to v and mark them.



queue

v
4
3
5
1

v edgeTo[v]

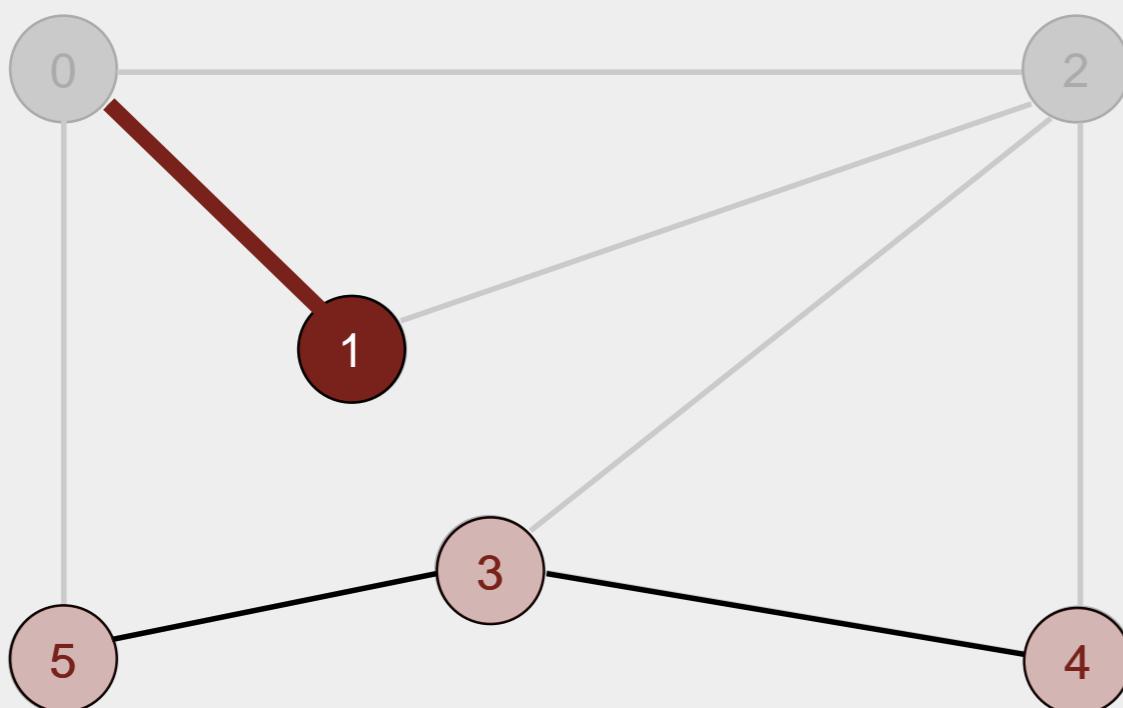
v	edgeTo[v]
0	-
1	0
2	0
3	2
4	2
5	0

dequeue 1

Breadth-first search

Repeat until queue is empty:

- Remove vertex v from queue.
- Add to queue all unmarked vertices adjacent to v and mark them.



queue

v
4
3
4
5
5

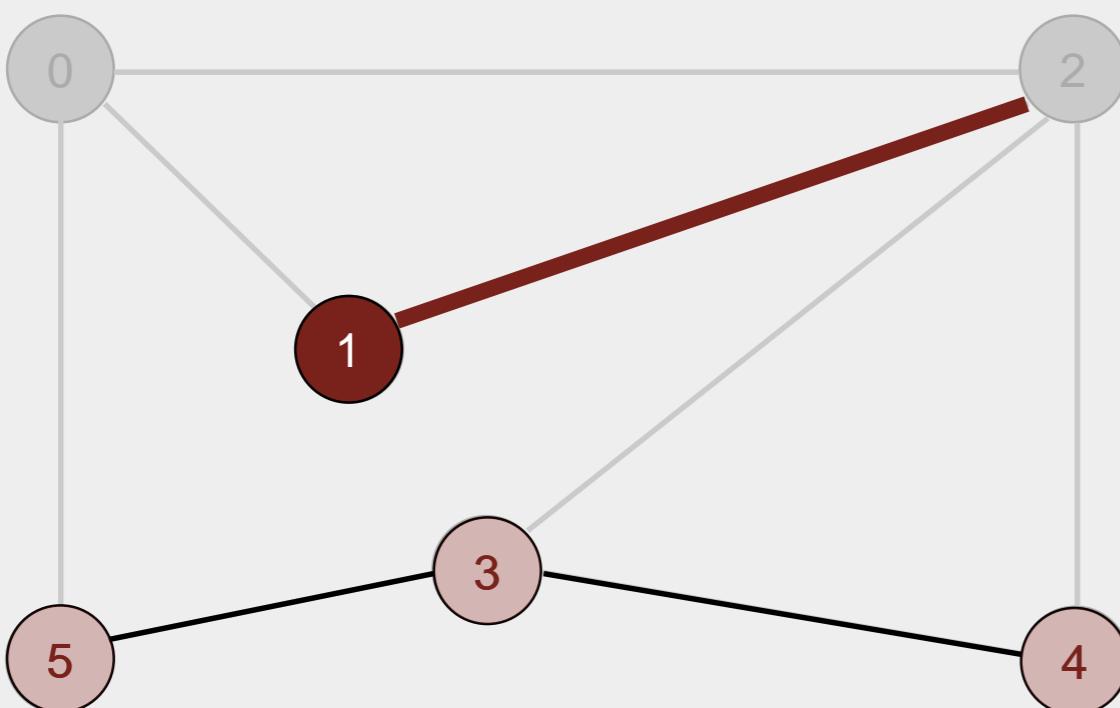
v	edgeTo[v]
0	-
1	0
2	0
3	2
4	2
5	0

dequeue 1

Breadth-first search

Repeat until queue is empty:

- Remove vertex v from queue.
- Add to queue all unmarked vertices adjacent to v and mark them.



queue

v
4
3
4
5
5

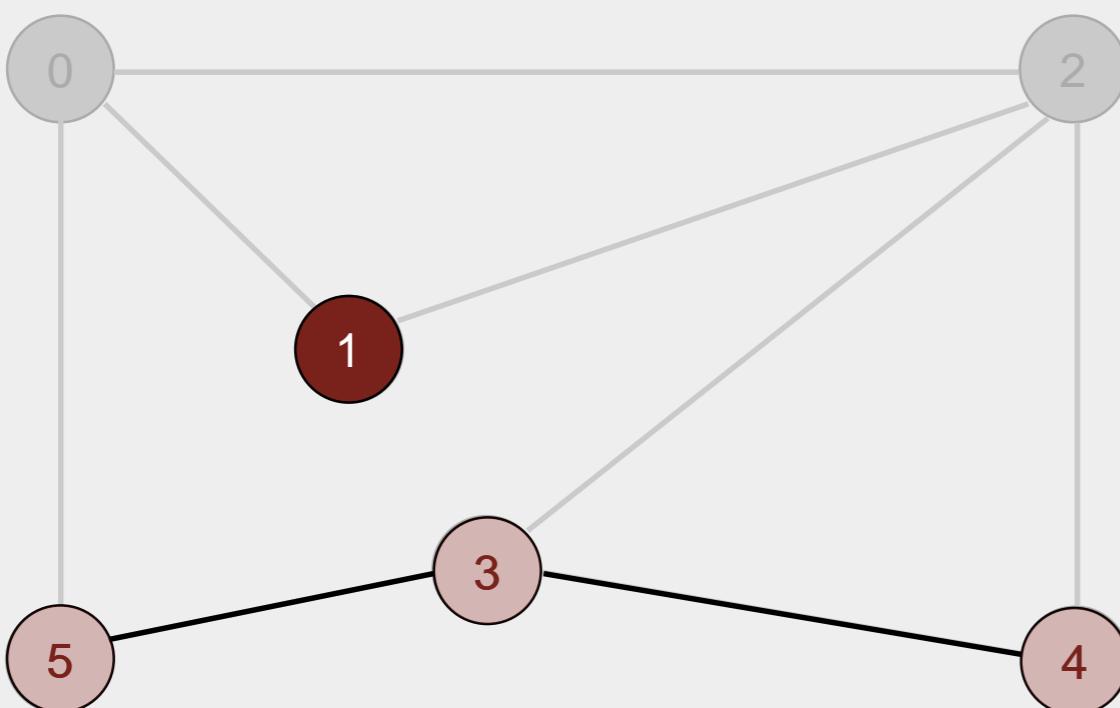
v	edgeTo[v]
0	-
1	0
2	0
3	2
4	2
5	0

dequeue 1

Breadth-first search

Repeat until queue is empty:

- Remove vertex v from queue.
- Add to queue all unmarked vertices adjacent to v and mark them.



queue

v
4
3
4
5
5

v edgeTo[v]

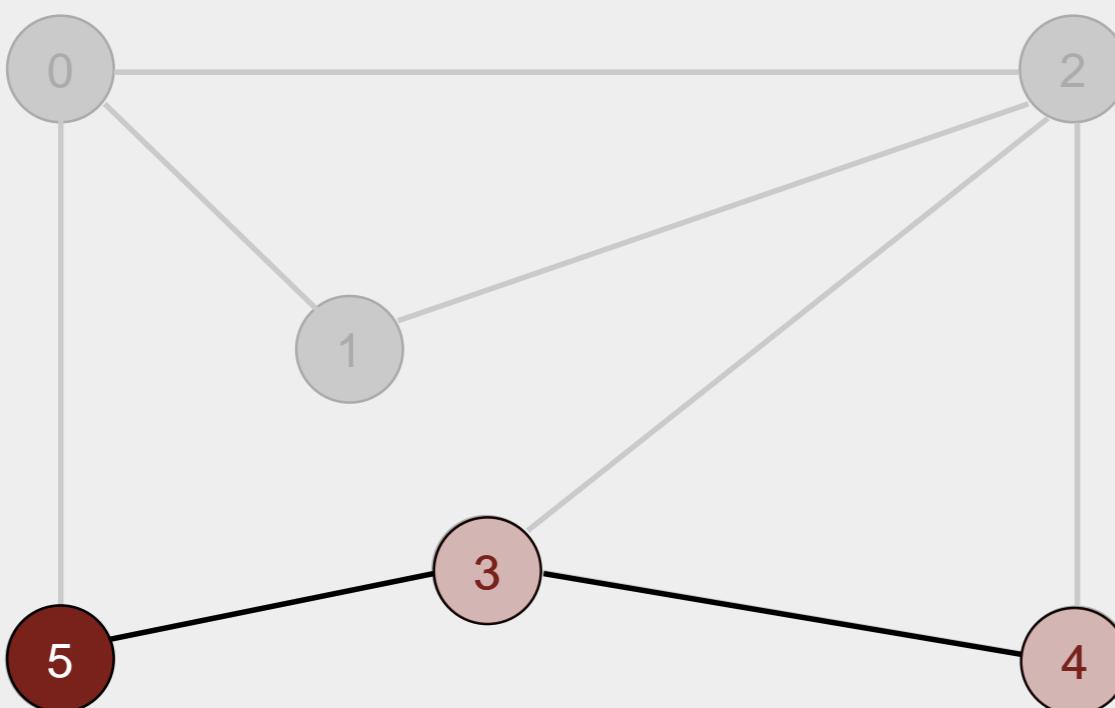
v	edgeTo[v]
0	-
1	0
2	0
3	2
4	2
5	0

1 done

Breadth-first search

Repeat until queue is empty:

- Remove vertex v from queue.
- Add to queue all unmarked vertices adjacent to v and mark them.



queue

v
4
3
4
5
5

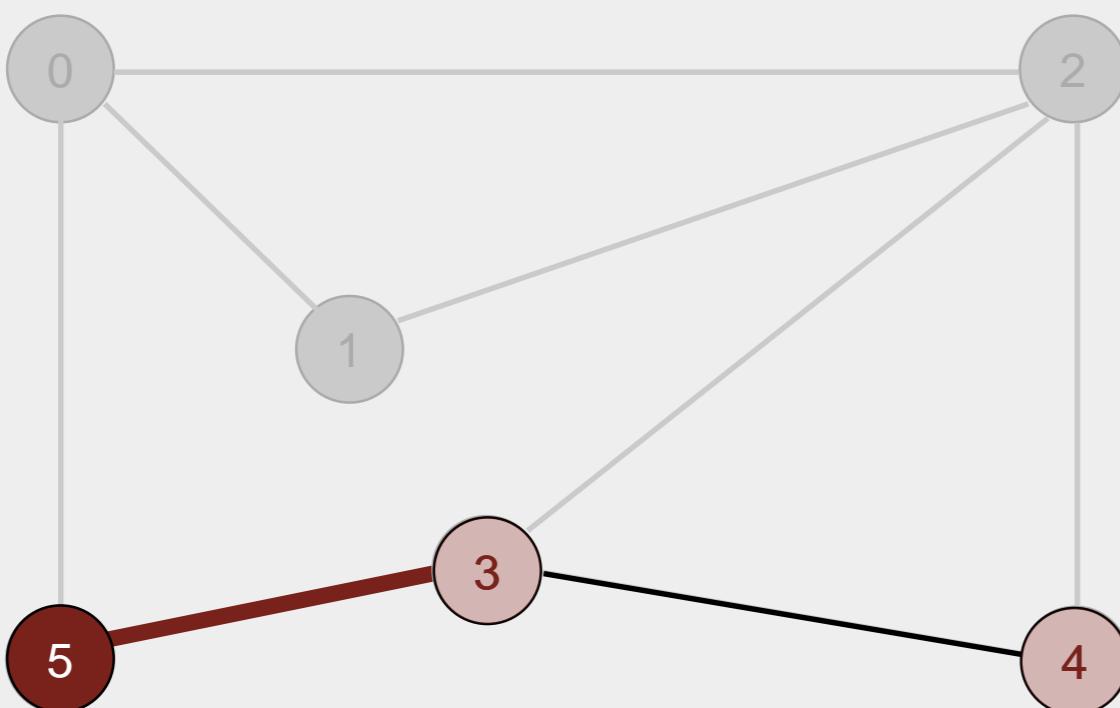
v	edgeTo[v]
0	-
1	0
2	0
3	2
4	2
5	0

dequeue 5

Breadth-first search

Repeat until queue is empty:

- Remove vertex v from queue.
- Add to queue all unmarked vertices adjacent to v and mark them.



queue

v
4
5
3

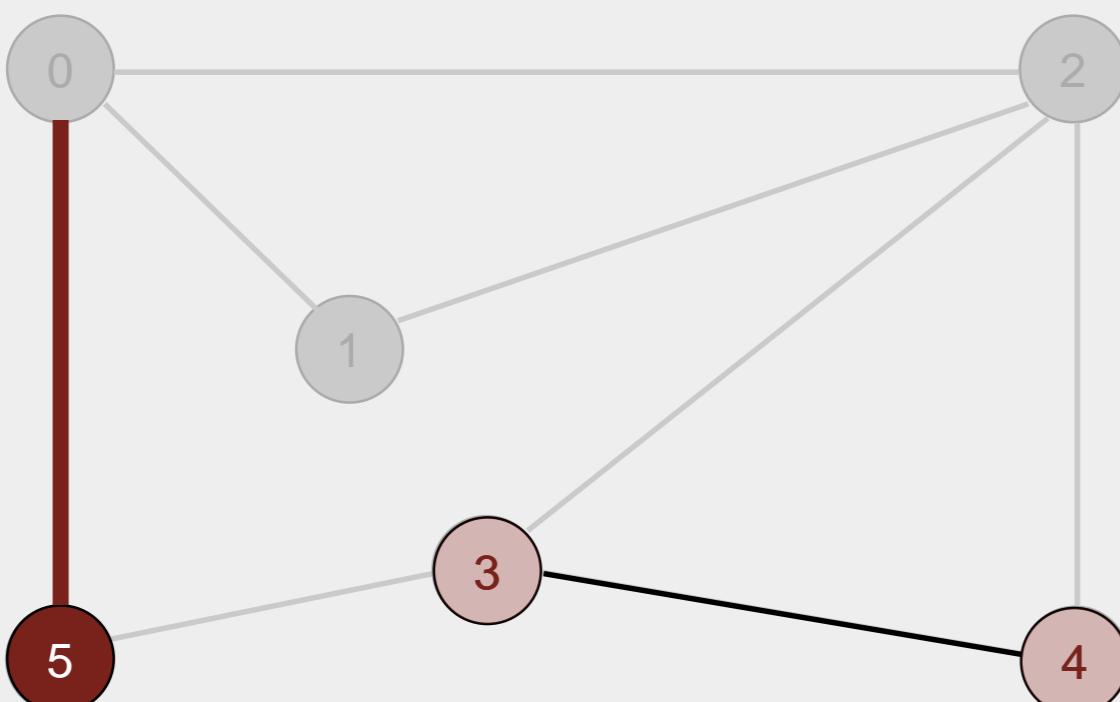
v	edgeTo[v]
0	-
1	0
2	0
3	2
4	2
5	0

dequeue 5

Breadth-first search

Repeat until queue is empty:

- Remove vertex v from queue.
- Add to queue all unmarked vertices adjacent to v and mark them.



queue

v
4
5
3

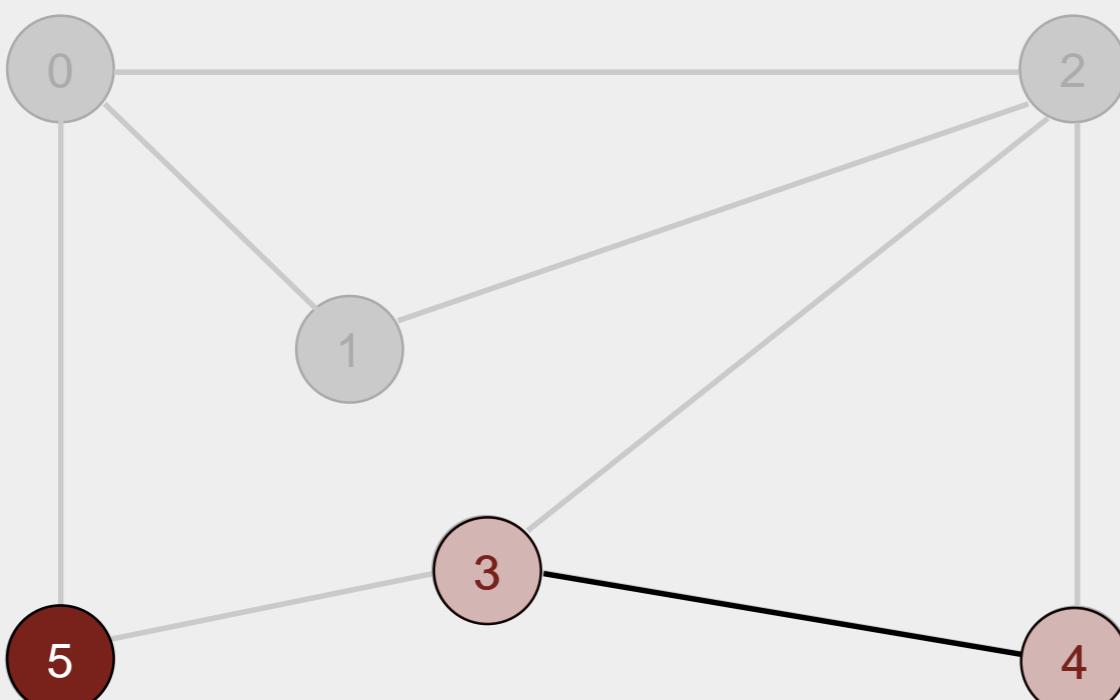
v	edgeTo[v]
0	-
1	0
2	0
3	2
4	2
5	0

dequeue 5

Breadth-first search

Repeat until queue is empty:

- Remove vertex v from queue.
- Add to queue all unmarked vertices adjacent to v and mark them.



queue

v
4
5
3

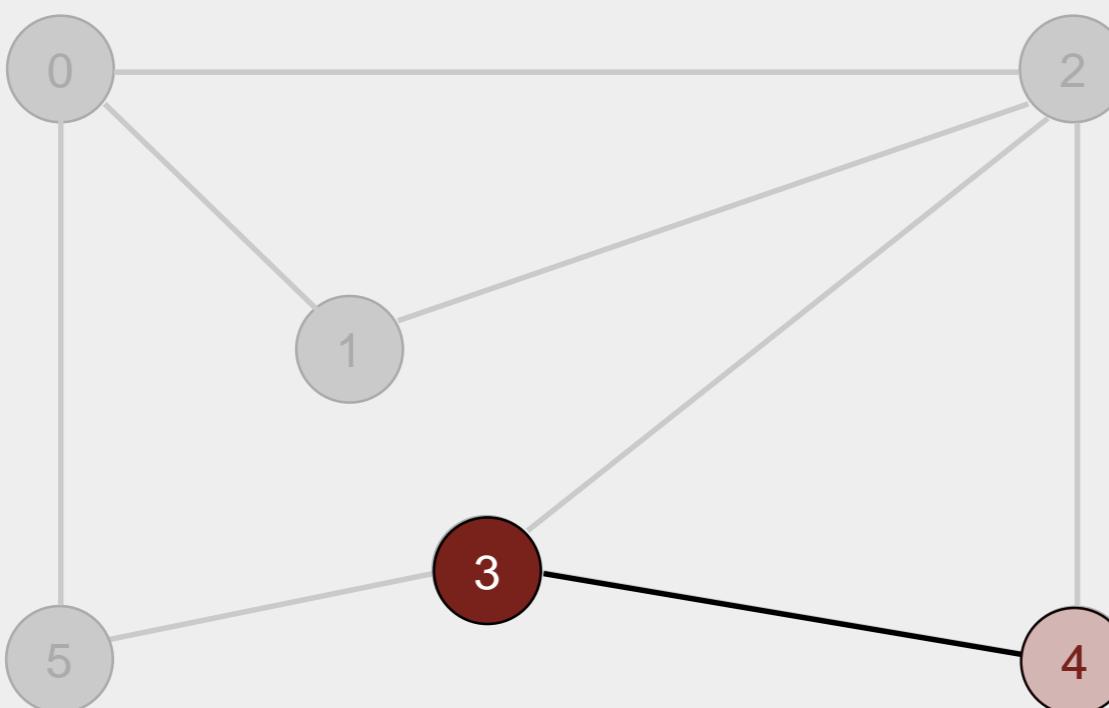
v	edgeTo[v]
0	-
1	0
2	0
3	2
4	2
5	0

5 done

Breadth-first search

Repeat until queue is empty:

- Remove vertex v from queue.
- Add to queue all unmarked vertices adjacent to v and mark them.



queue

v
4
5
3

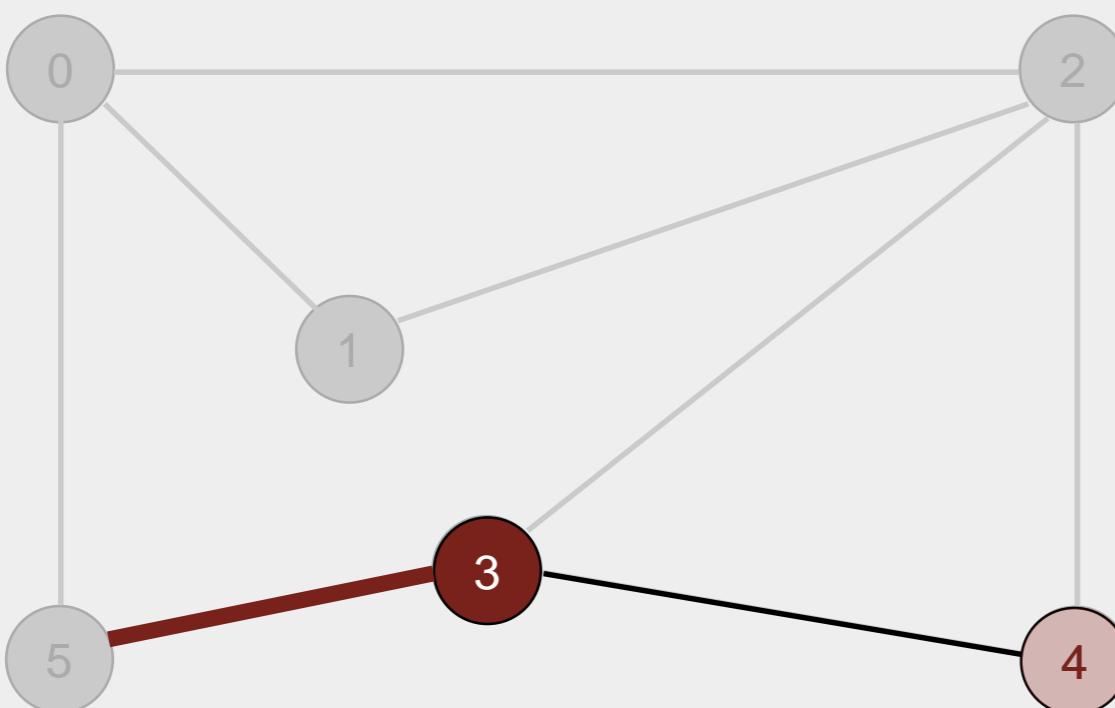
v	edgeTo[v]
0	-
1	0
2	0
3	2
4	2
5	0

dequeue 3

Breadth-first search

Repeat until queue is empty:

- Remove vertex v from queue.
- Add to queue all unmarked vertices adjacent to v and mark them.



dequeue 3

queue

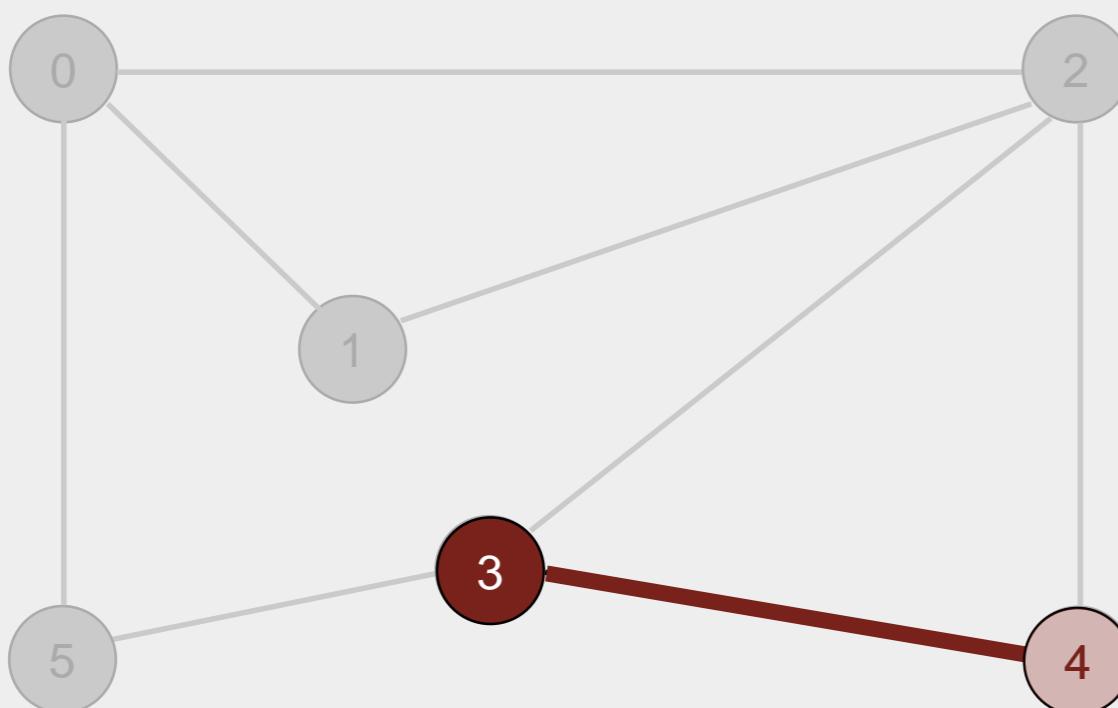
v	edgeTo[v]
0	-
1	0
2	0
3	2
4	2
5	0

v	edgeTo[v]
0	-
1	0
2	0
3	2
4	2
5	0

Breadth-first search

Repeat until queue is empty:

- Remove vertex v from queue.
- Add to queue all unmarked vertices adjacent to v and mark them.



queue

v	edgeTo[v]
0	-
1	0
2	0
3	2
4	2
5	0

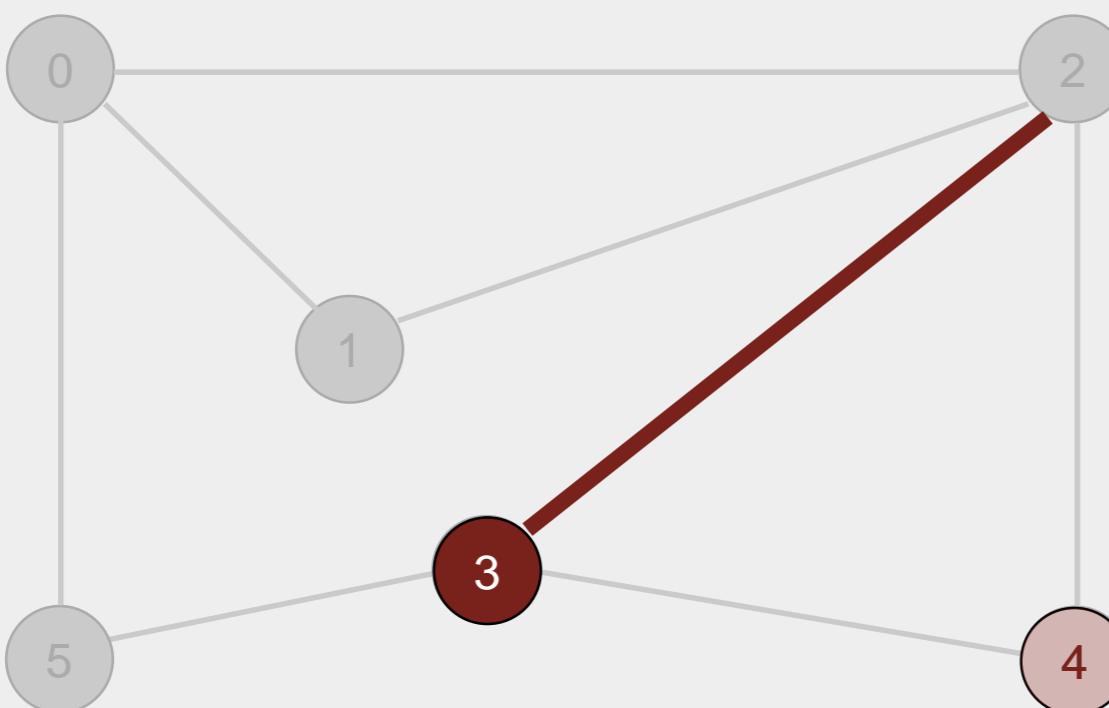
v	edgeTo[v]
0	-
1	0
2	0
3	2
4	2
5	0

dequeue 3

Breadth-first search

Repeat until queue is empty:

- Remove vertex v from queue.
- Add to queue all unmarked vertices adjacent to v and mark them.



queue

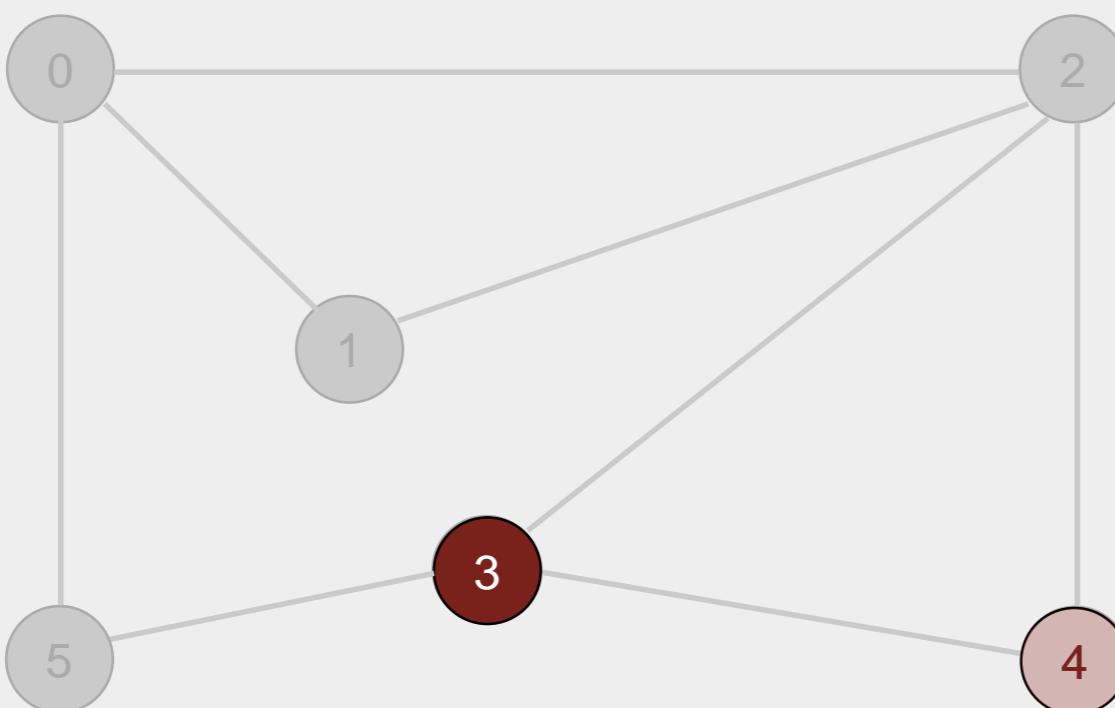
v	edgeTo[v]
0	-
1	0
2	0
3	2
4	2
5	0

dequeue 3

Breadth-first search

Repeat until queue is empty:

- Remove vertex v from queue.
- Add to queue all unmarked vertices adjacent to v and mark them.



queue

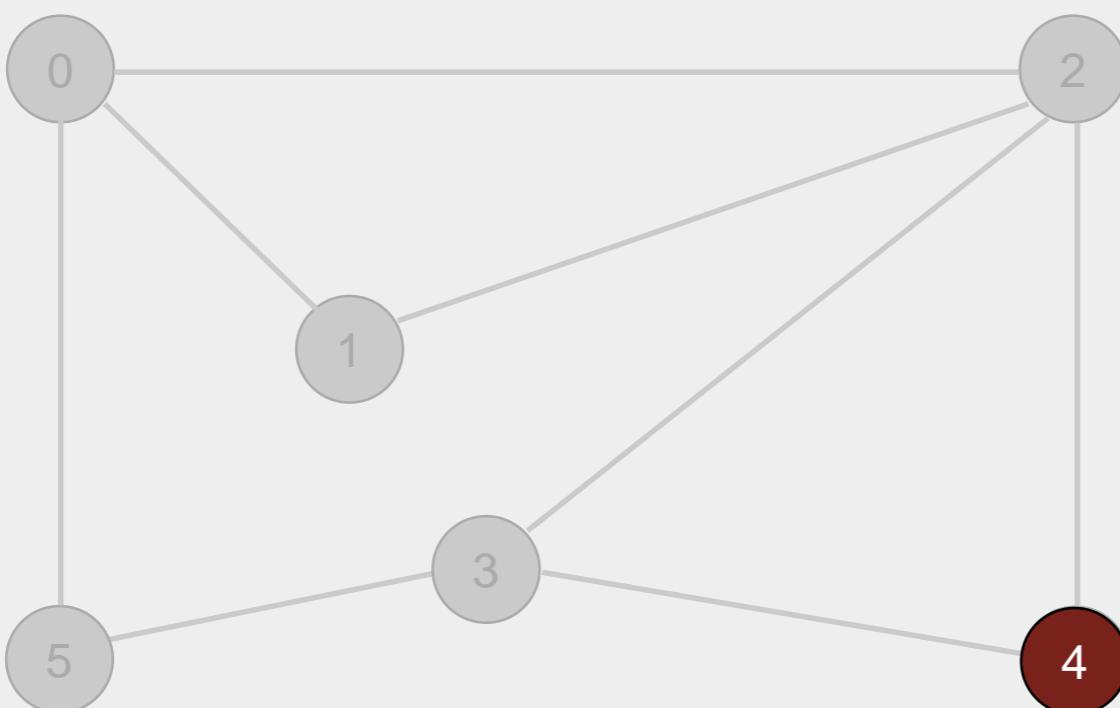
v	edgeTo[v]
0	-
1	0
2	0
3	2
4	2
5	0

3 done

Breadth-first search

Repeat until queue is empty:

- Remove vertex v from queue.
- Add to queue all unmarked vertices adjacent to v and mark them.



queue

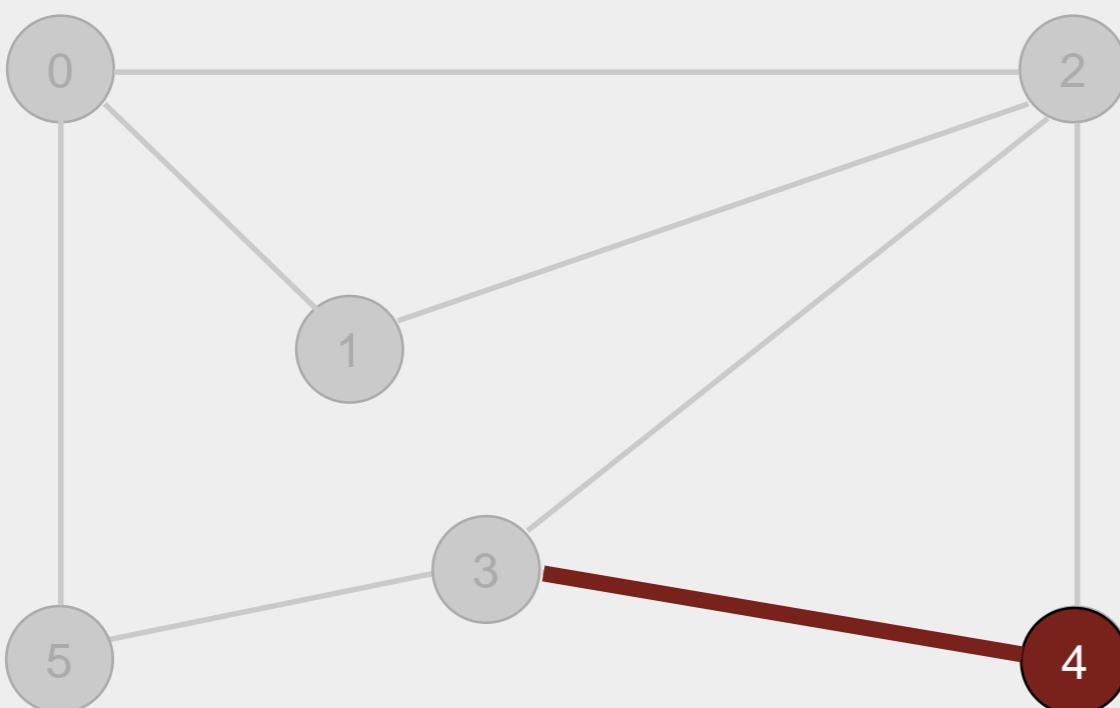
v	edgeTo[v]
0	-
1	0
2	0
3	2
4	2
5	0

dequeue 4

Breadth-first search

Repeat until queue is empty:

- Remove vertex v from queue.
- Add to queue all unmarked vertices adjacent to v and mark them.



queue

v	edgeTo[v]
0	-
1	0
2	0
3	2
4	2
5	0

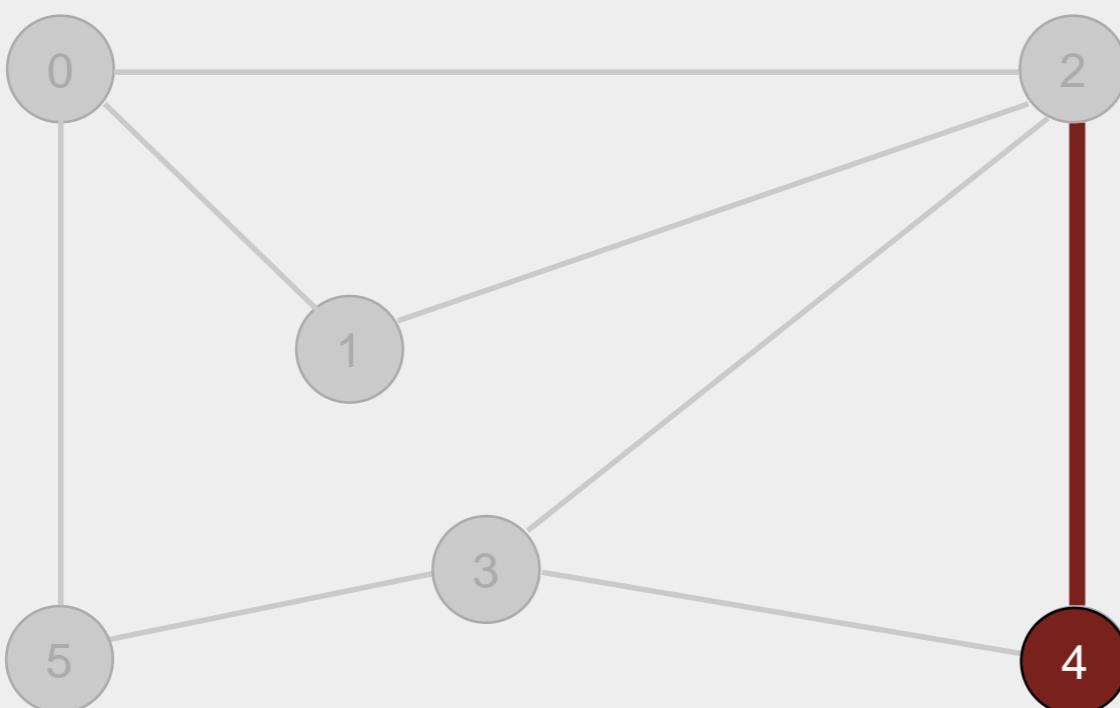
v	edgeTo[v]
0	-
1	0
2	0
3	2
4	2
5	0

dequeue 4

Breadth-first search

Repeat until queue is empty:

- Remove vertex v from queue.
- Add to queue all unmarked vertices adjacent to v and mark them.



queue

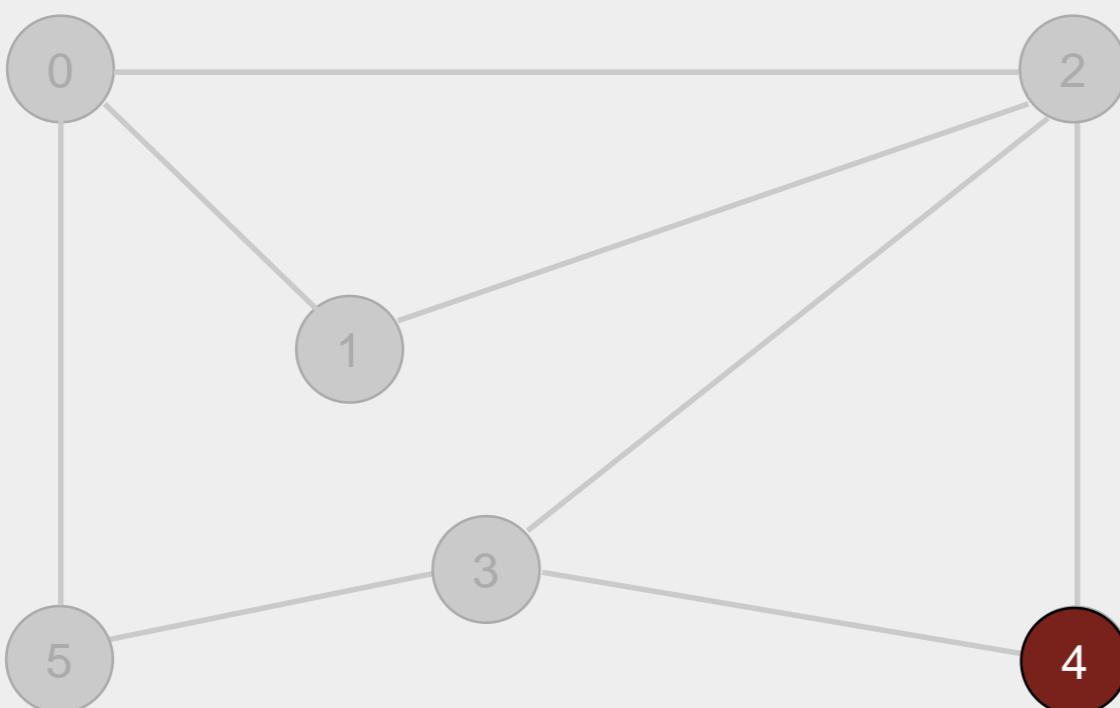
v	edgeTo[v]
0	-
1	0
2	0
3	2
4	2
5	0

dequeue 4

Breadth-first search

Repeat until queue is empty:

- Remove vertex v from queue.
- Add to queue all unmarked vertices adjacent to v and mark them.



queue

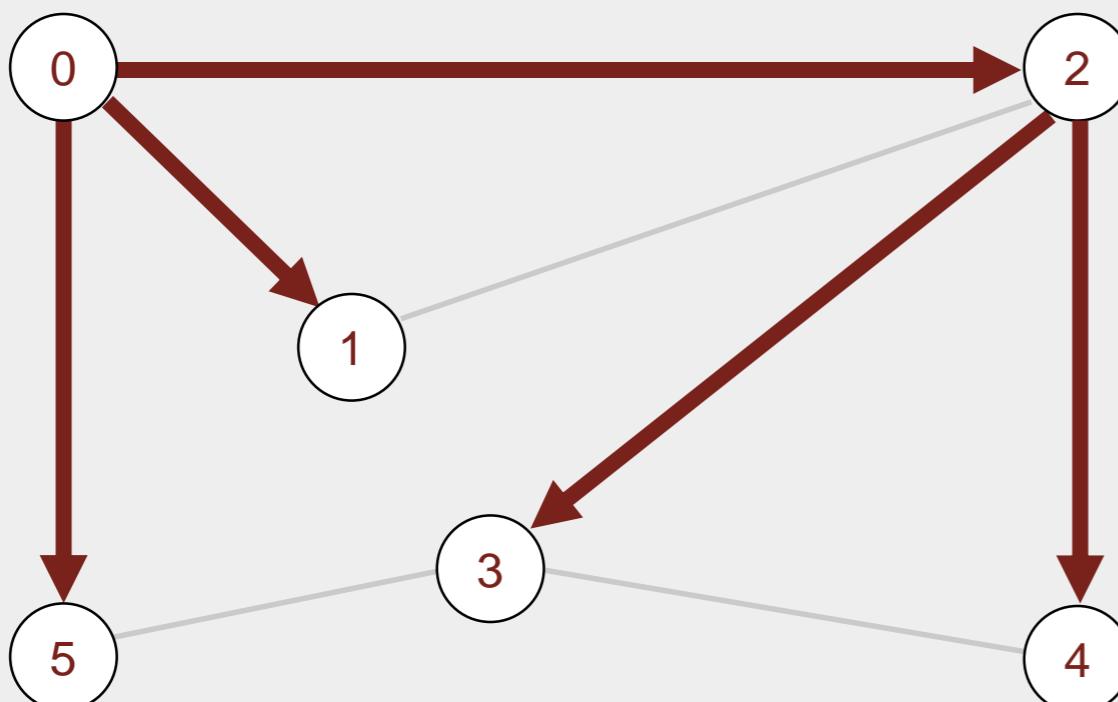
v	edgeTo[v]
0	-
1	0
2	0
3	2
4	2
5	0

4 done

Breadth-first search

Repeat until queue is empty:

- Remove vertex v from queue.
- Add to queue all unmarked vertices adjacent to v and mark them.



v	$\text{edgeTo}[v]$
0	-
1	0
2	0
3	2
4	2
5	0

done

4.2 Depth-First Search

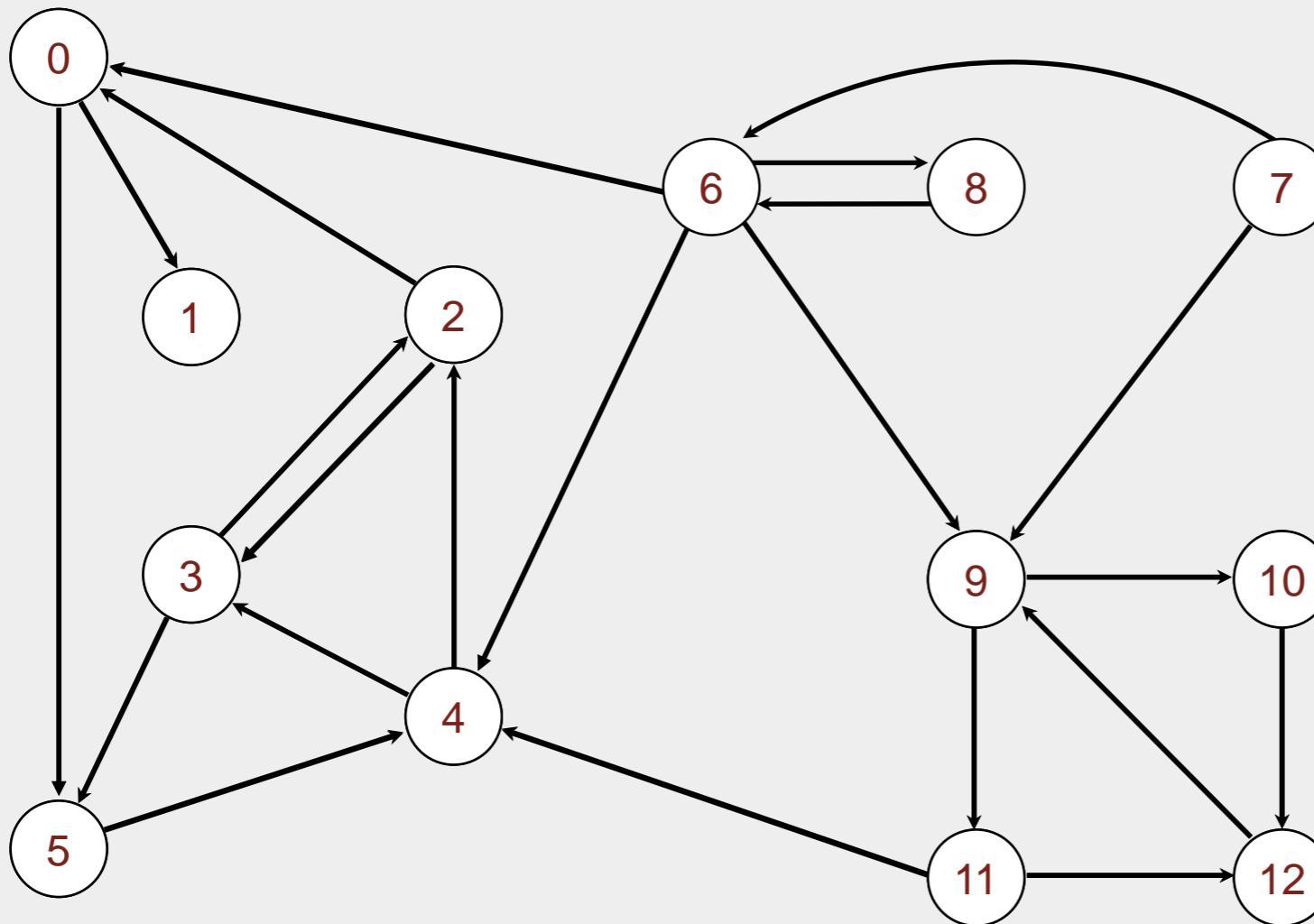


click to begin demo

Depth-first search

To visit a vertex v :

- Mark vertex v as visited.
- Recursively visit all unmarked vertices pointing from v .



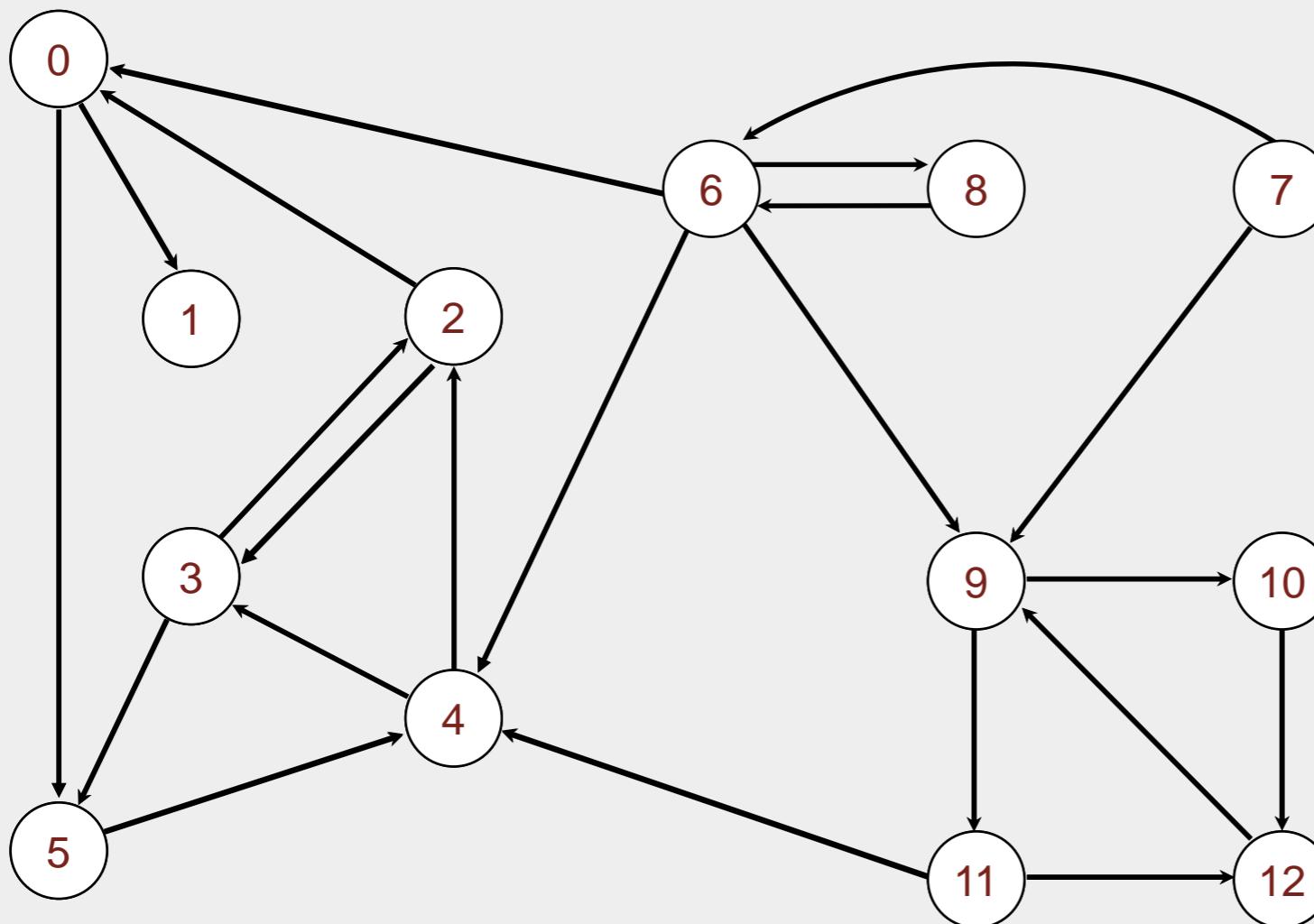
a directed graph

4□2
2□3
3□2
6□0
0□1
2□0
11□12
12□9
9□10
9□11
8□9
10□12
11□4
4□3
3□5
6□8
8□6
5□4
0□5
6□4
6□9
7□6

Depth-first search

To visit a vertex v :

- Mark vertex v as visited.
- Recursively visit all unmarked vertices pointing from v .



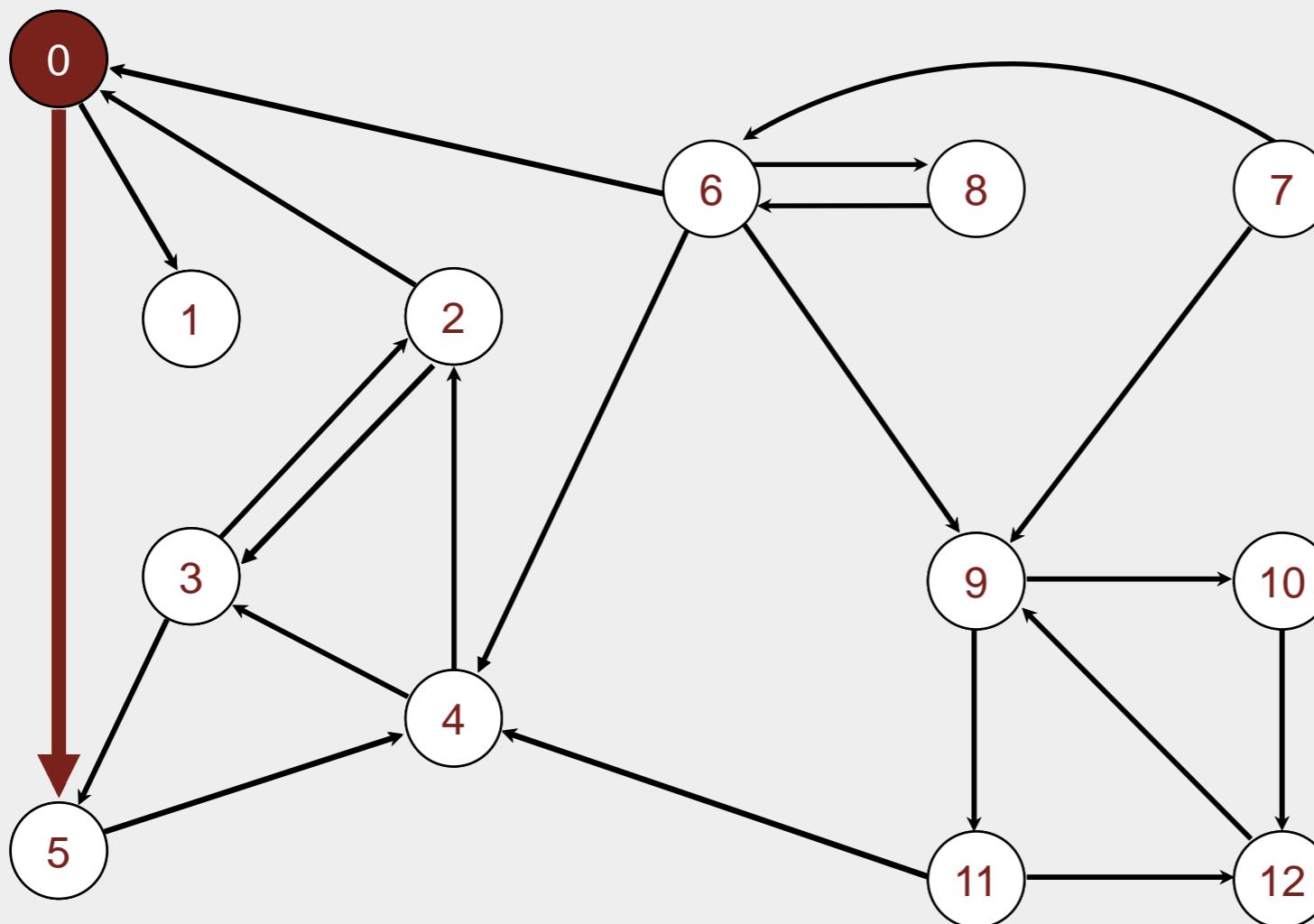
a directed graph

v	marked[]	edgeTo[]
0	F	-
1	F	-
2	F	-
3	F	-
4	F	-
5	F	-
6	F	-
7	F	-
8	F	-
9	F	-
10	F	-
11	F	-
12	F	-

Depth-first search

To visit a vertex v :

- Mark vertex v as visited.
- Recursively visit all unmarked vertices pointing from v .



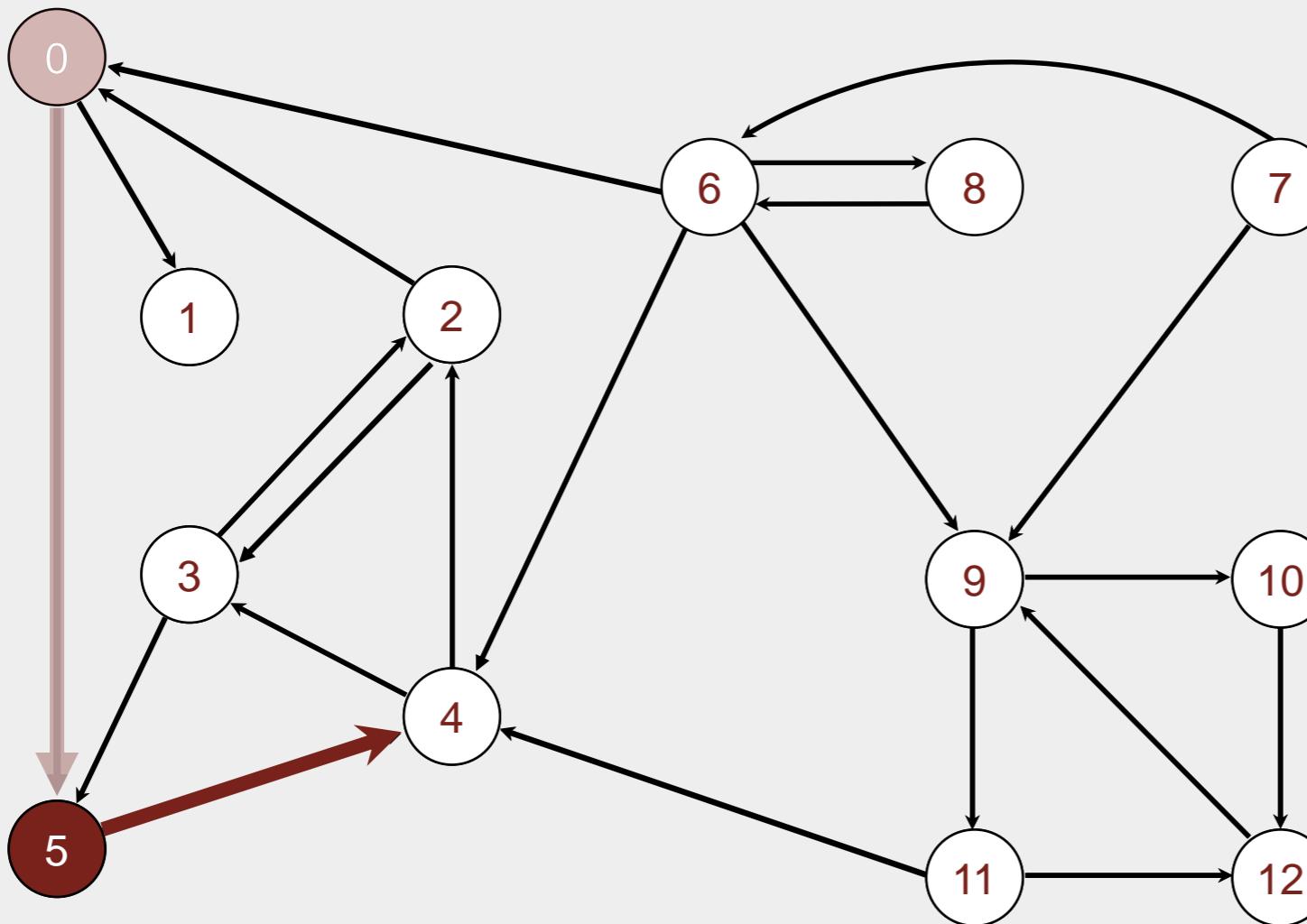
visit 0

v	marked[]	edgeTo[]
0	T	-
1	F	-
2	F	-
3	F	-
4	F	-
5	F	-
6	F	-
7	F	-
8	F	-
9	F	-
10	F	-
11	F	-
12	F	-

Depth-first search

To visit a vertex v :

- Mark vertex v as visited.
- Recursively visit all unmarked vertices pointing from v .



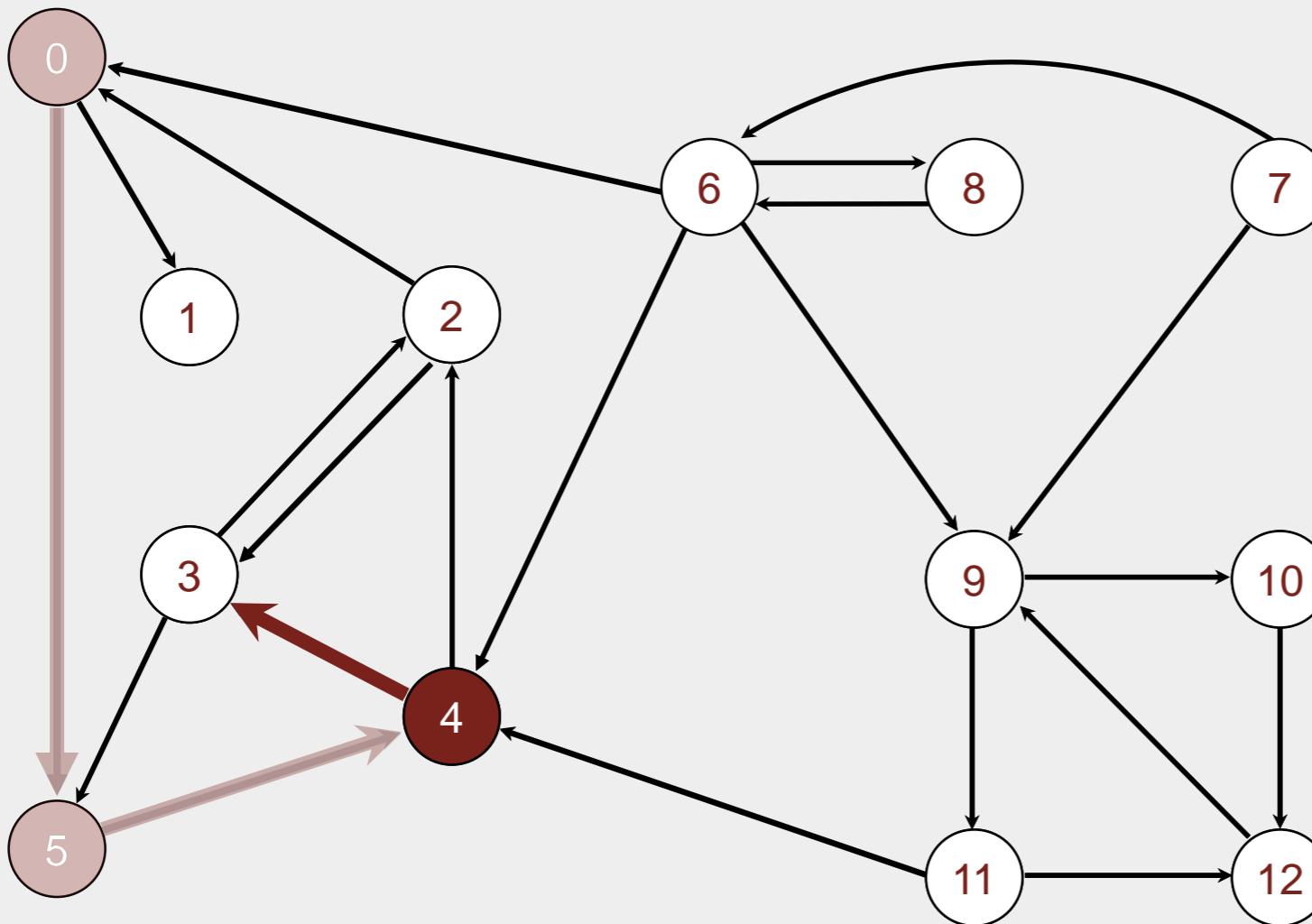
visit 5

v	marked[]	edgeTo[]
0	T	-
1	F	-
2	F	-
3	F	-
4	F	-
5	T	0
6	F	-
7	F	-
8	F	-
9	F	-
10	F	-
11	F	-
12	F	-

Depth-first search

To visit a vertex v :

- Mark vertex v as visited.
- Recursively visit all unmarked vertices pointing from v .

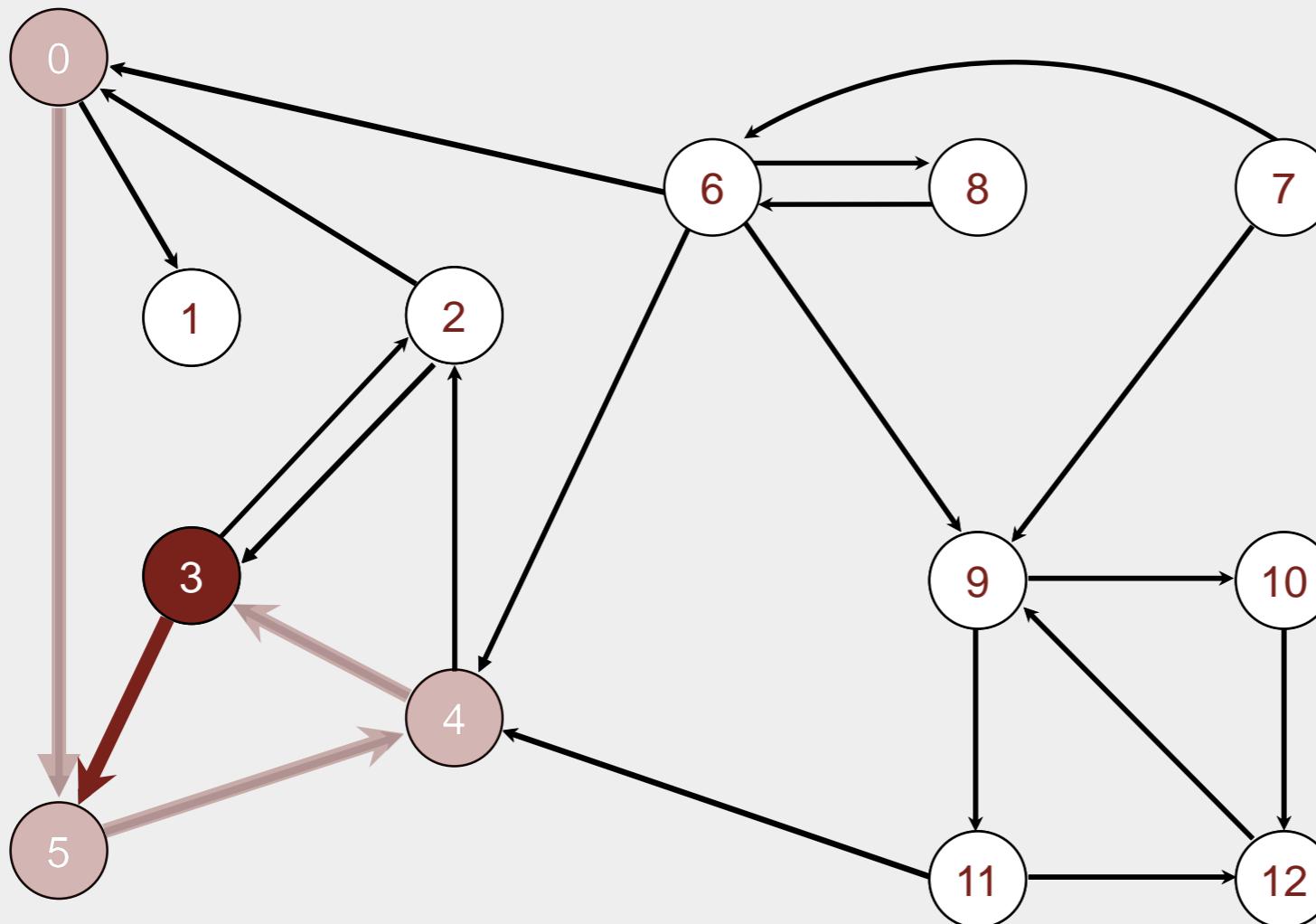


v	marked[]	edgeTo[]
0	T	-
1	F	-
2	F	-
3	F	-
4	T	5
5	T	0
6	F	-
7	F	-
8	F	-
9	F	-
10	F	-
11	F	-
12	F	-

Depth-first search

To visit a vertex v :

- Mark vertex v as visited.
- Recursively visit all unmarked vertices pointing from v .

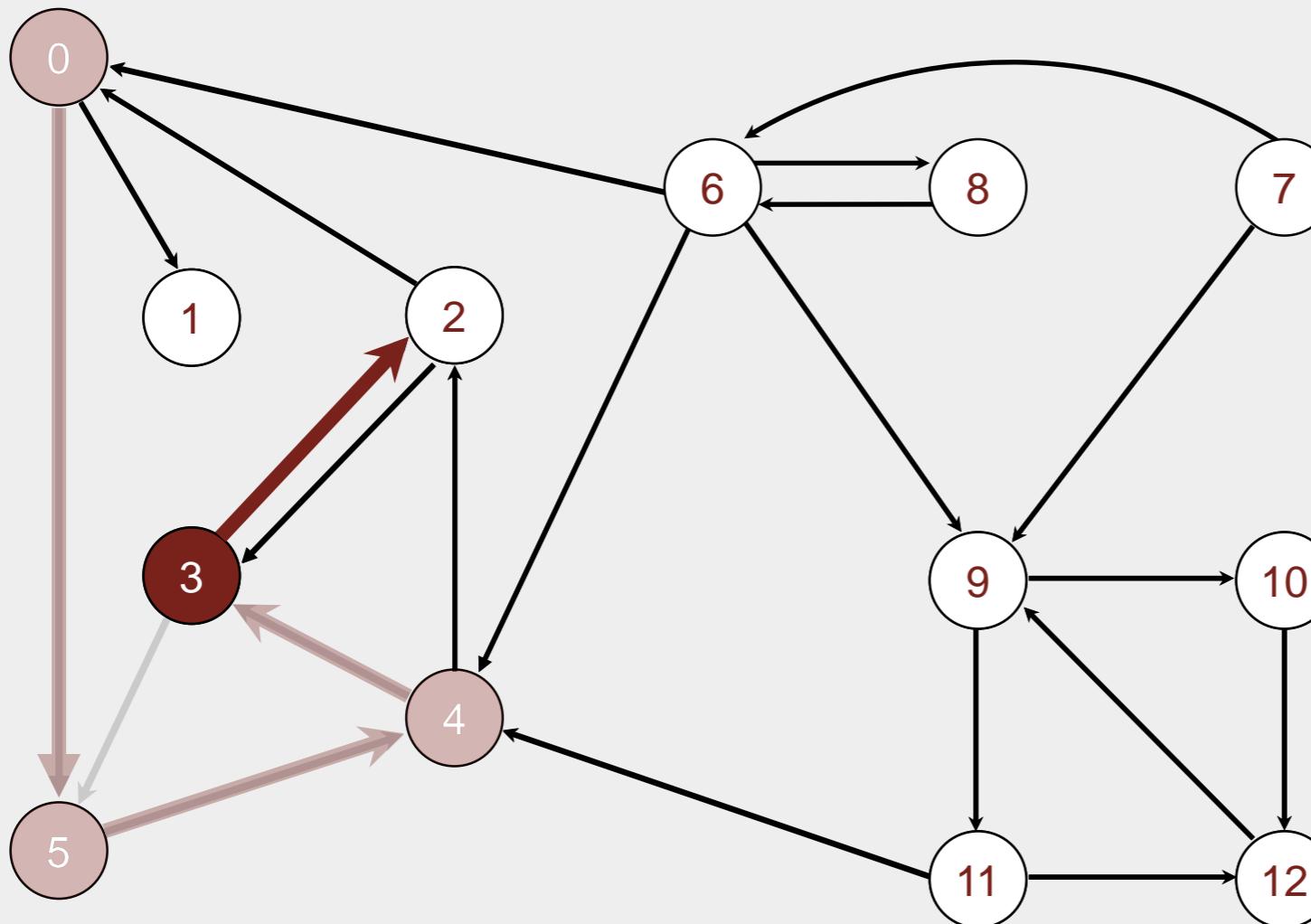


v	marked[]	edgeTo[]
0	T	-
1	F	-
2	F	-
3	T	4
4	T	5
5	T	0
6	F	-
7	F	-
8	F	-
9	F	-
10	F	-
11	F	-
12	F	-

Depth-first search

To visit a vertex v :

- Mark vertex v as visited.
- Recursively visit all unmarked vertices pointing from v .



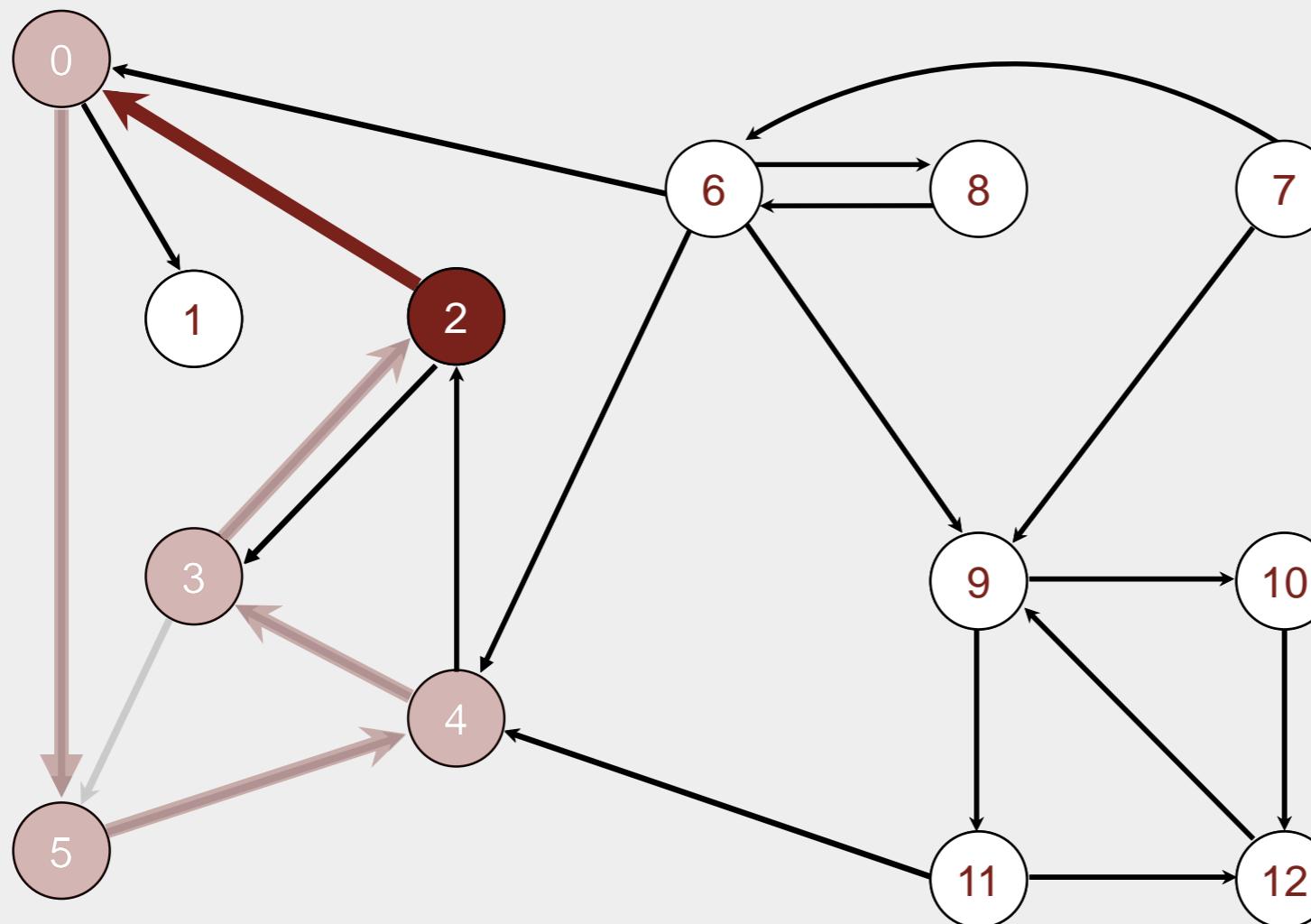
visit 3

v	marked[]	edgeTo[]
0	T	-
1	F	-
2	F	-
3	T	4
4	T	5
5	T	0
6	F	-
7	F	-
8	F	-
9	F	-
10	F	-
11	F	-
12	F	-

Depth-first search

To visit a vertex v :

- Mark vertex v as visited.
 - Recursively visit all unmarked vertices pointing from v .



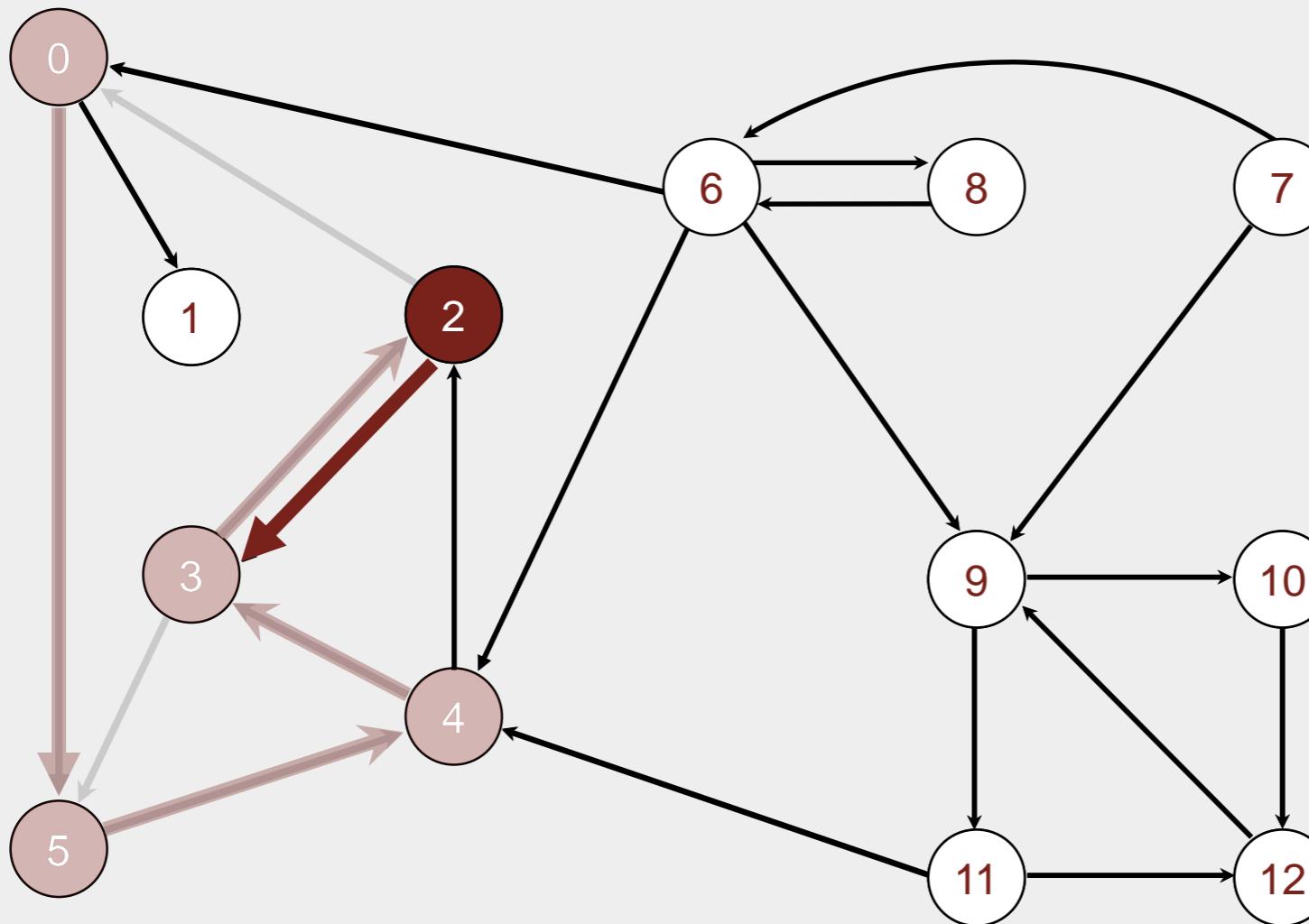
visit 2

v	marked[]	edgeTo[]
0	T	-
1	F	-
2	T	3
3	T	4
4	T	5
5	T	0
6	F	-
7	F	-
8	F	-
9	F	-
10	F	-
11	F	-
12	F	-

Depth-first search

To visit a vertex v :

- Mark vertex v as visited.
- Recursively visit all unmarked vertices pointing from v .



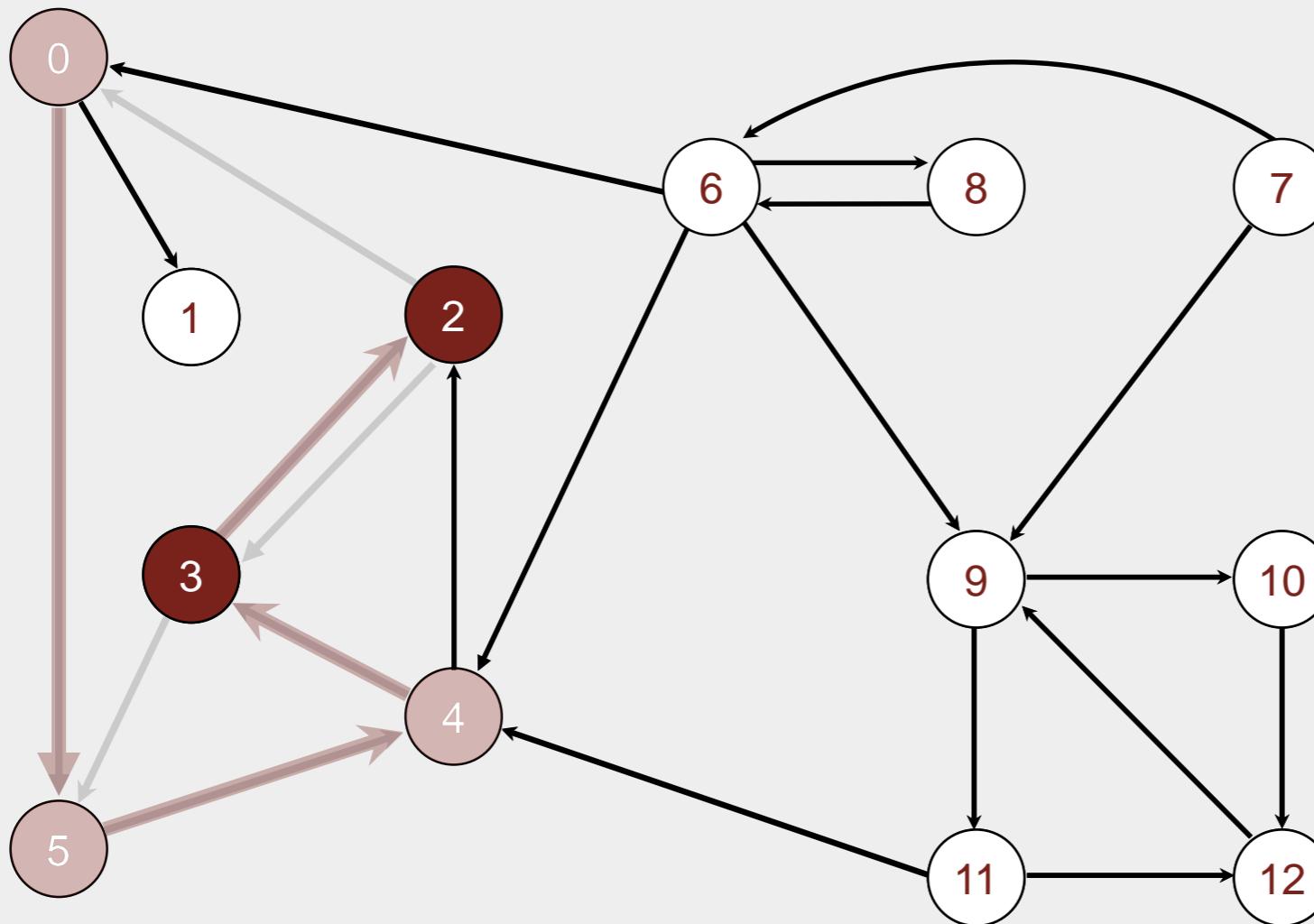
visit 2

v	marked[]	edgeTo[]
0	T	-
1	F	-
2	T	3
3	T	4
4	T	5
5	T	0
6	F	-
7	F	-
8	F	-
9	F	-
10	F	-
11	F	-
12	F	-

Depth-first search

To visit a vertex v :

- Mark vertex v as visited.
- Recursively visit all unmarked vertices pointing from v .



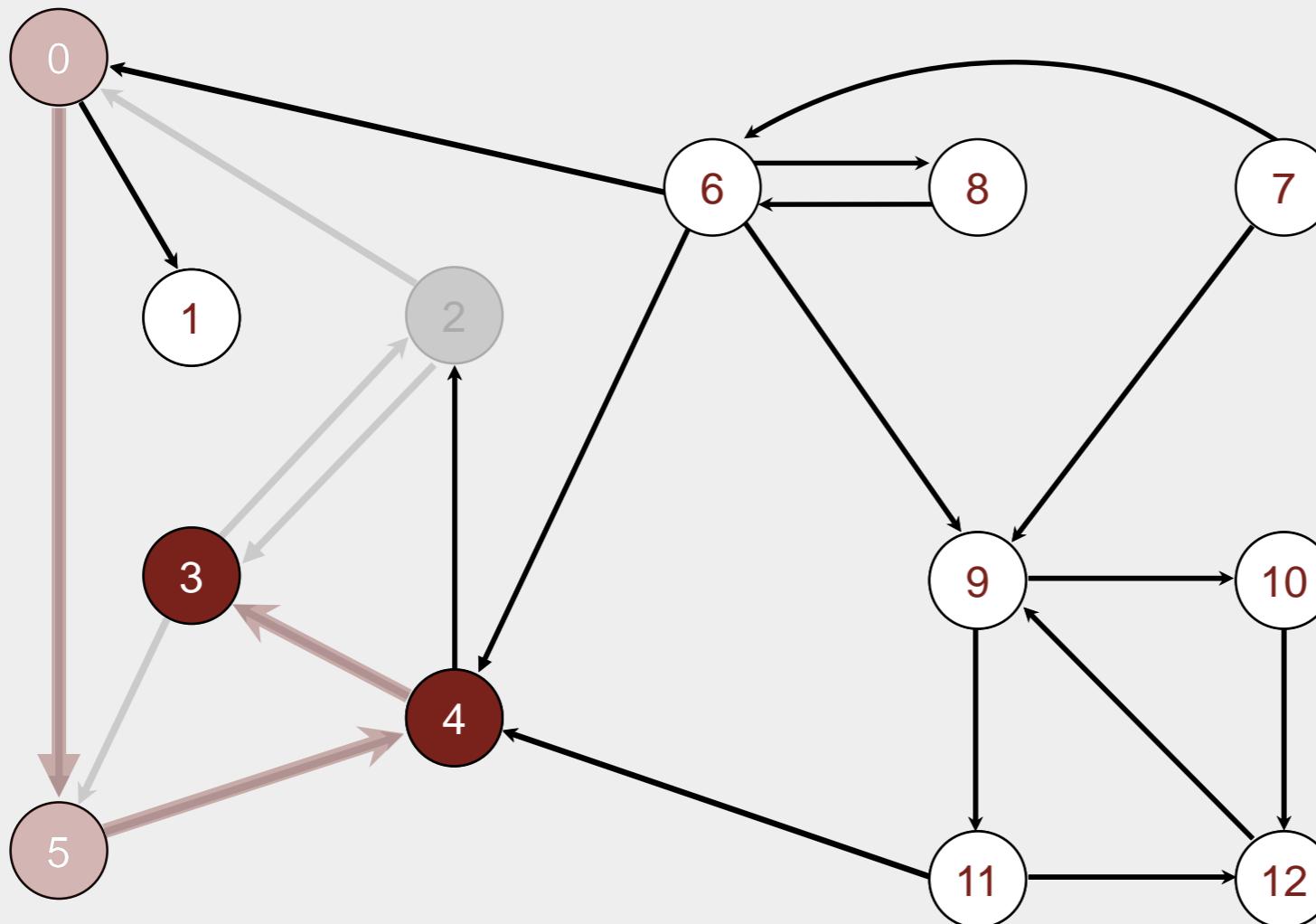
done 2

v	marked[]	edgeTo[]
0	T	-
1	F	-
2	T	3
3	T	4
4	T	5
5	T	0
6	F	-
7	F	-
8	F	-
9	F	-
10	F	-
11	F	-
12	F	-

Depth-first search

To visit a vertex v :

- Mark vertex v as visited.
- Recursively visit all unmarked vertices pointing from v .



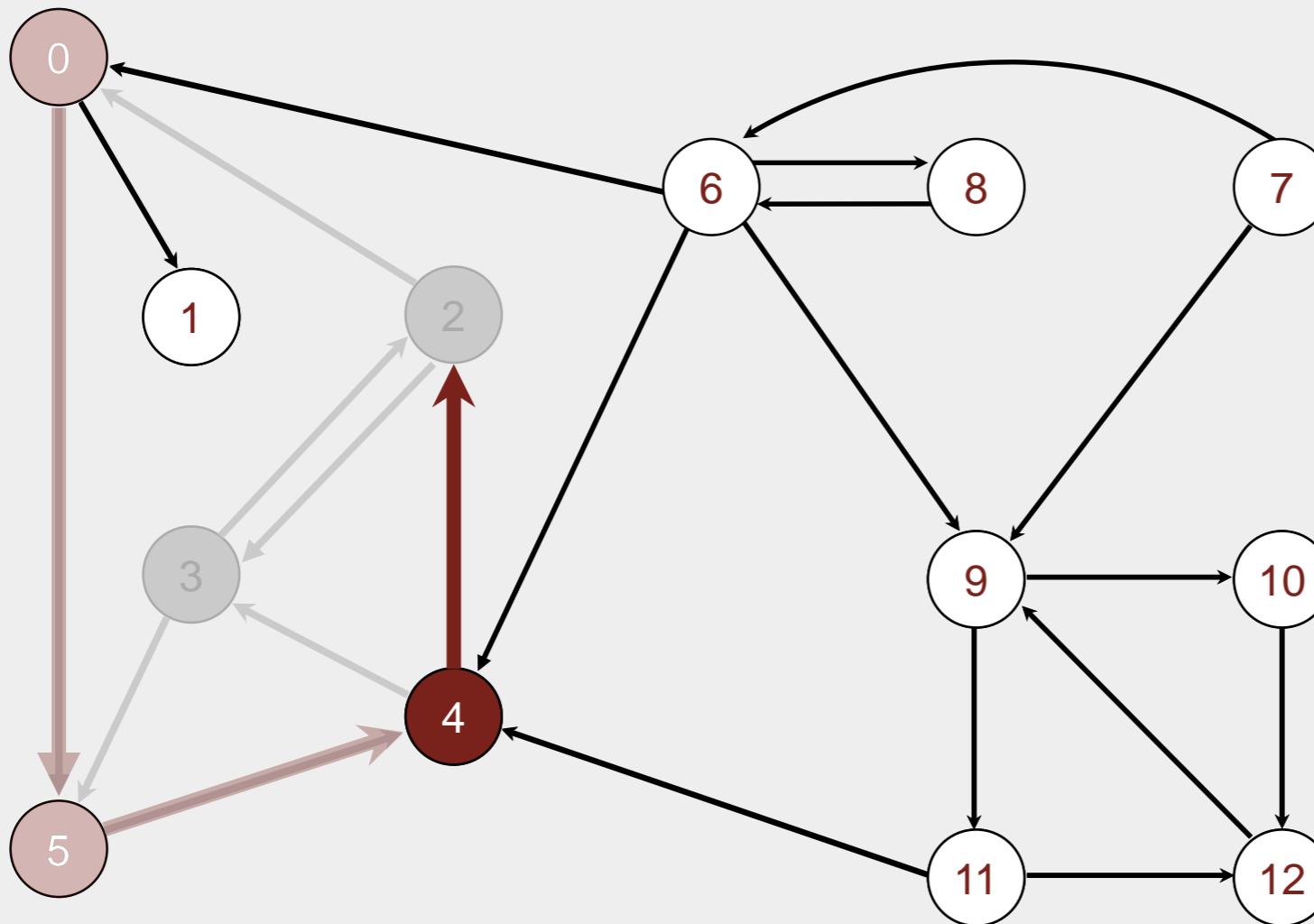
done 3

v	marked[]	edgeTo[]
0	T	-
1	F	-
2	T	3
3	T	4
4	T	5
5	T	0
6	F	-
7	F	-
8	F	-
9	F	-
10	F	-
11	F	-
12	F	-

Depth-first search

To visit a vertex v :

- Mark vertex v as visited.
- Recursively visit all unmarked vertices pointing from v .



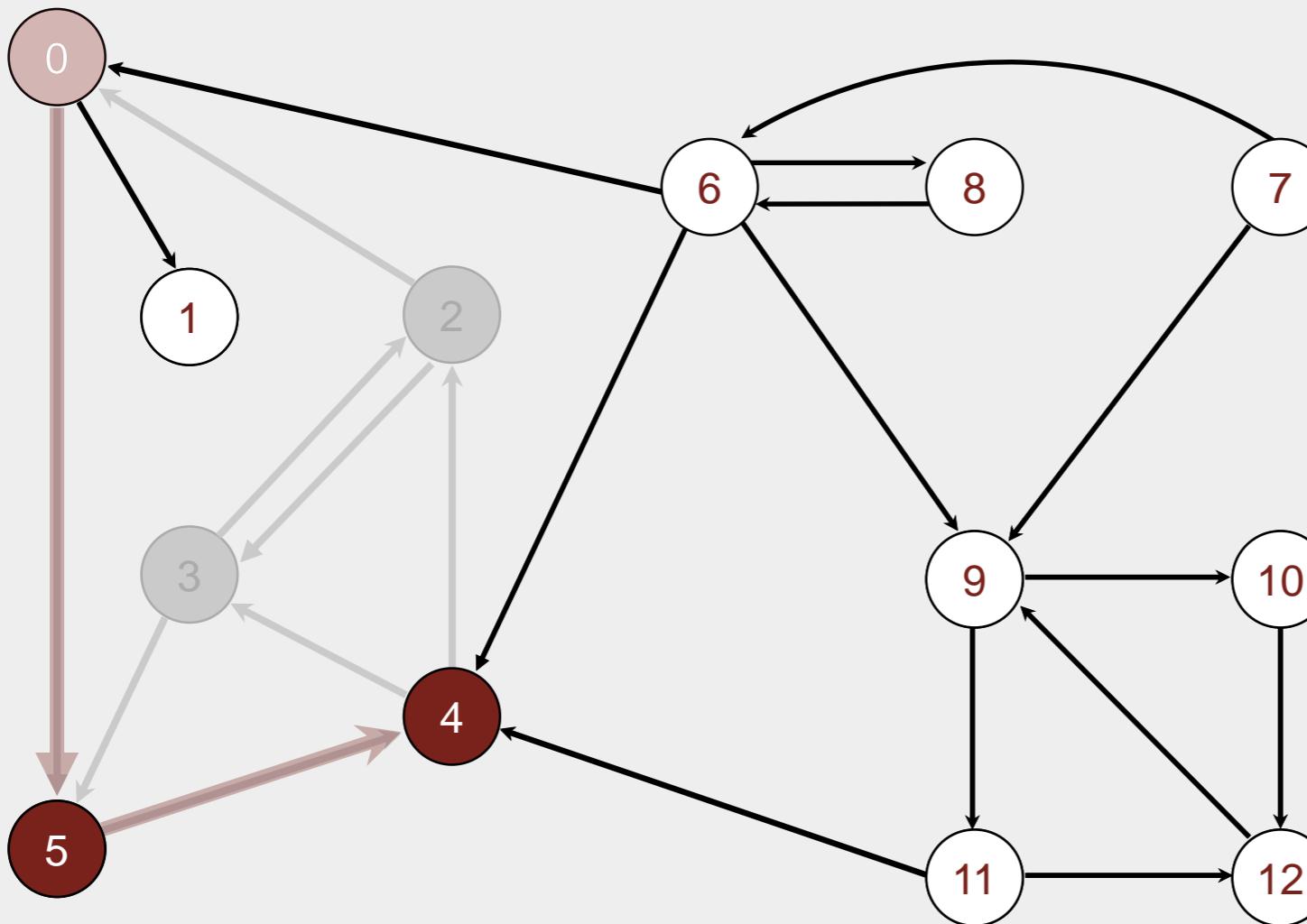
visit 4

v	marked[]	edgeTo[]
0	T	-
1	F	-
2	T	3
3	T	4
4	T	5
5	T	0
6	F	-
7	F	-
8	F	-
9	F	-
10	F	-
11	F	-
12	F	-

Depth-first search

To visit a vertex v :

- Mark vertex v as visited.
- Recursively visit all unmarked vertices pointing from v .

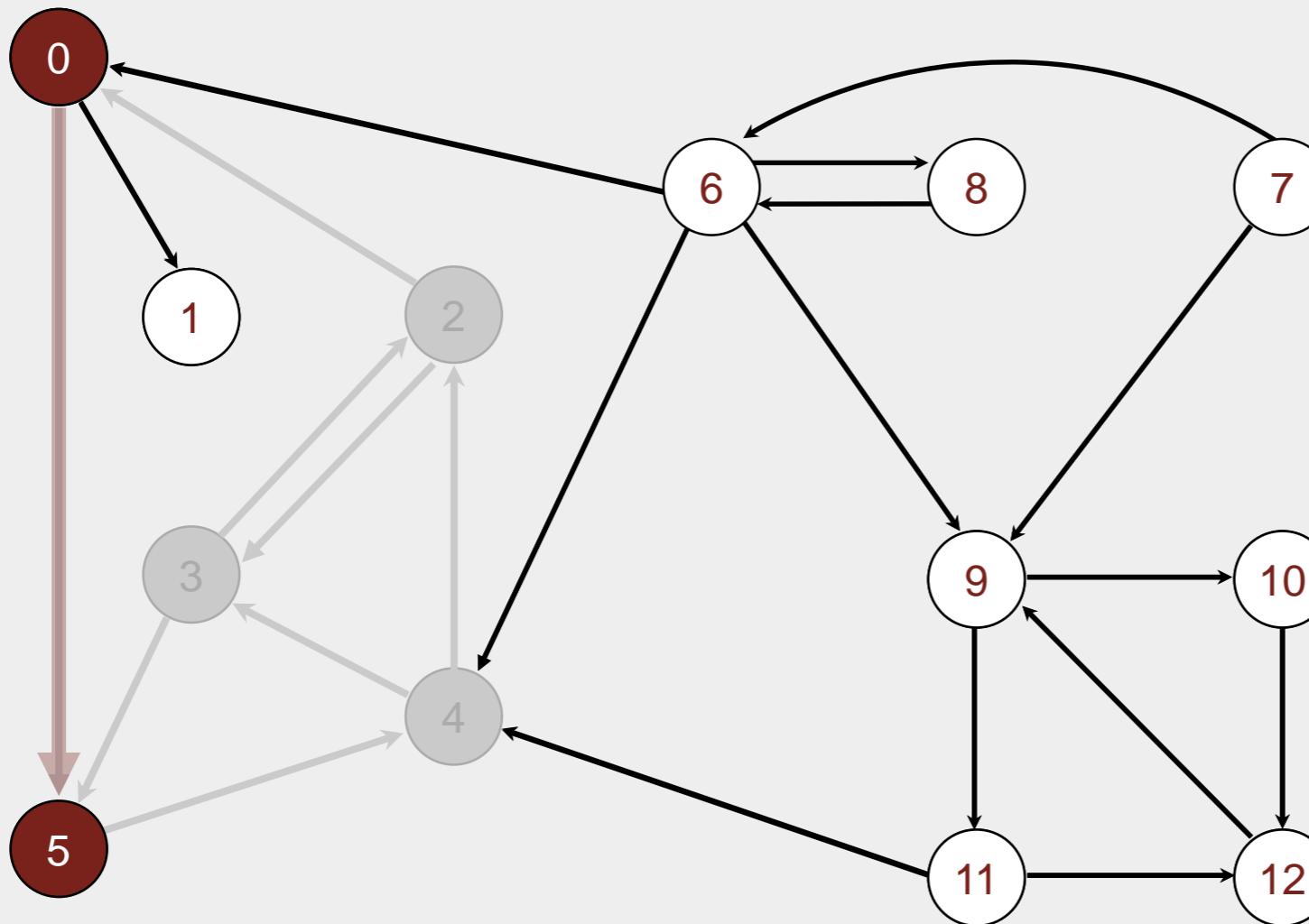


v	marked[]	edgeTo[]
0	T	-
1	F	-
2	T	3
3	T	4
4	T	5
5	T	0
6	F	-
7	F	-
8	F	-
9	F	-
10	F	-
11	F	-
12	F	-

Depth-first search

To visit a vertex v :

- Mark vertex v as visited.
- Recursively visit all unmarked vertices pointing from v .



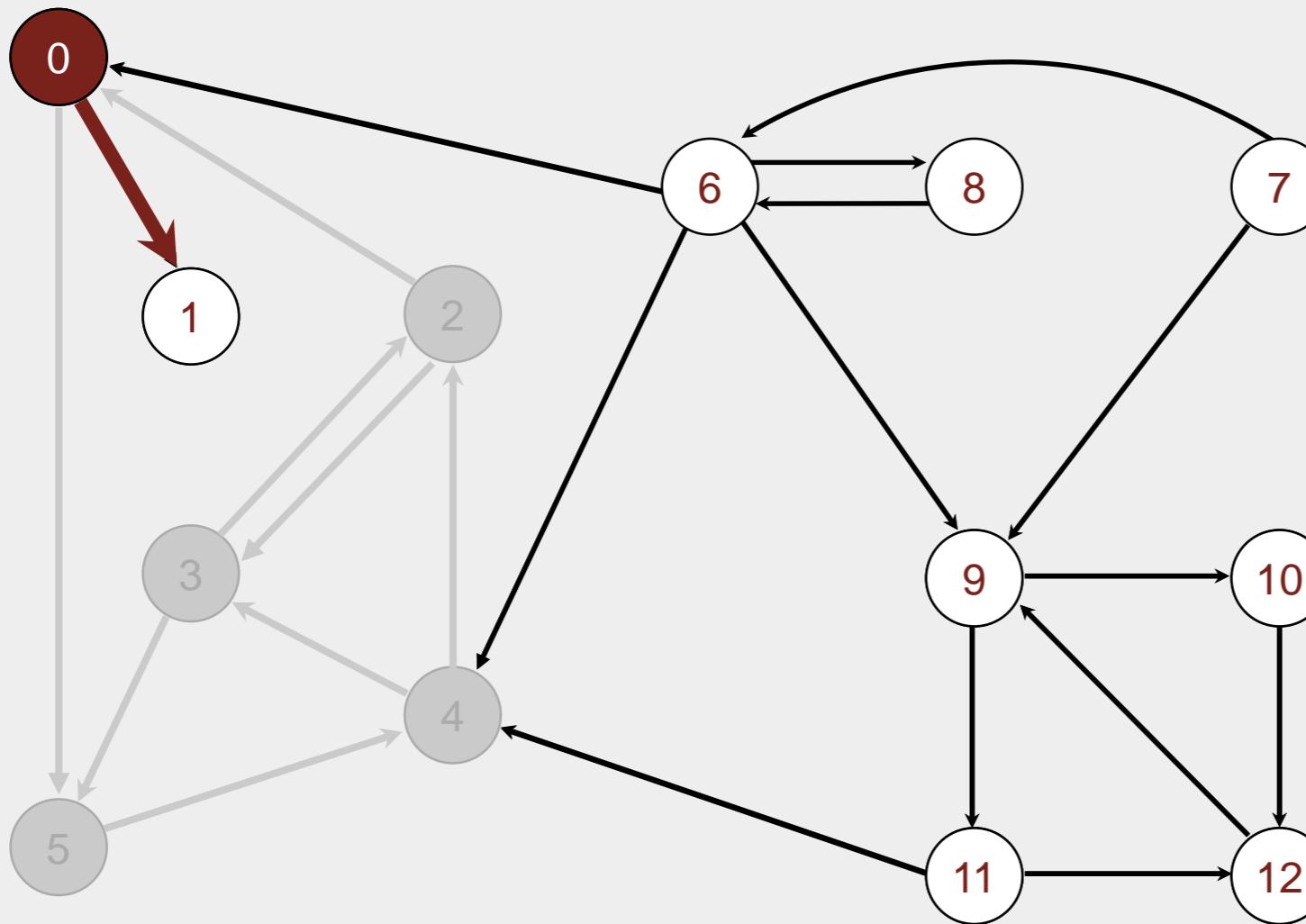
done 5

<code>v</code>	<code>marked[]</code>	<code>edgeTo[]</code>
0	T	-
1	F	-
2	T	3
3	T	4
4	T	5
5	T	0
6	F	-
7	F	-
8	F	-
9	F	-
10	F	-
11	F	-
12	F	-

Depth-first search

To visit a vertex v :

- Mark vertex v as visited.
- Recursively visit all unmarked vertices pointing from v .



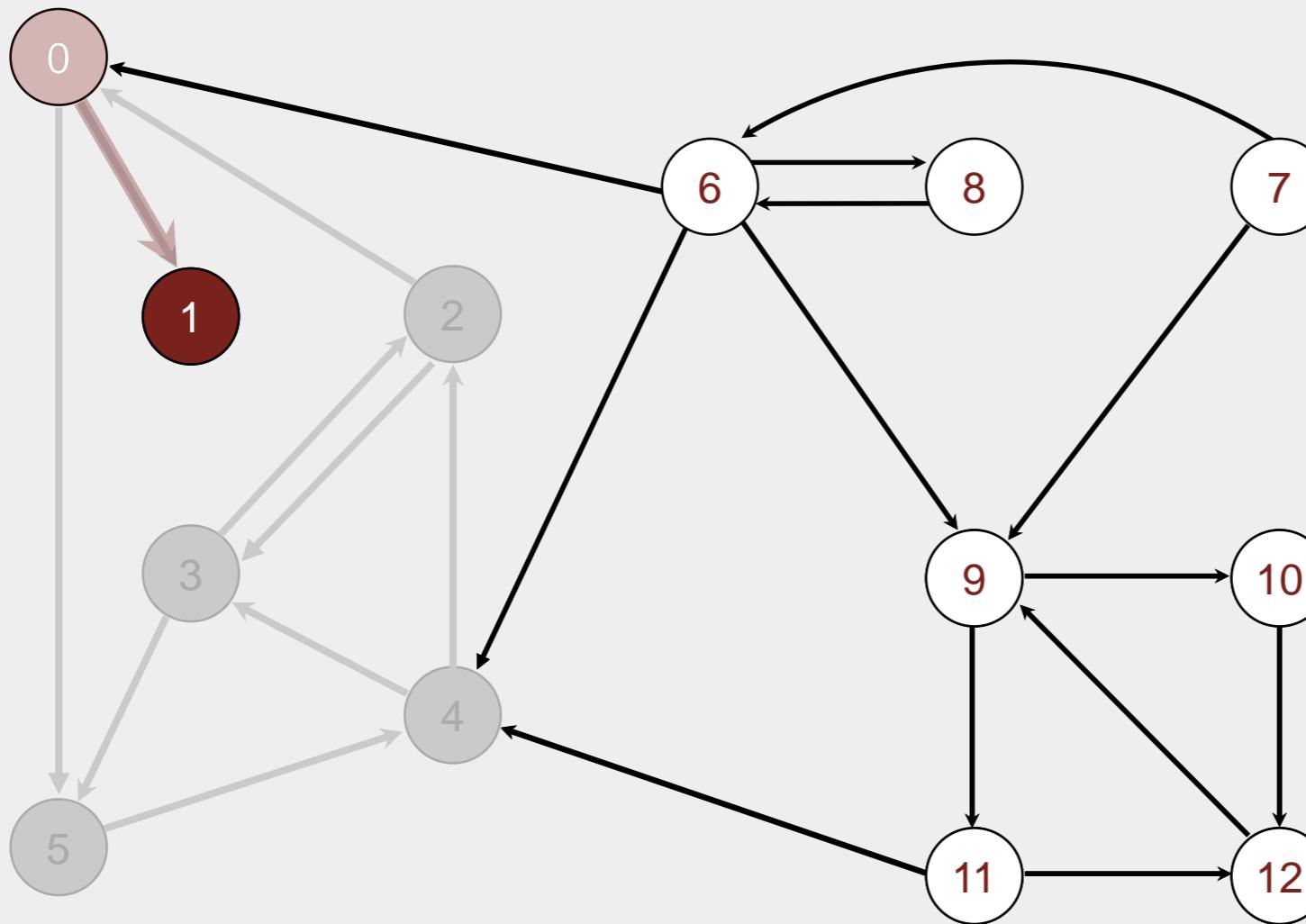
visit 0

v	marked[]	edgeTo[]
0	T	-
1	F	-
2	T	3
3	T	4
4	T	5
5	T	0
6	F	-
7	F	-
8	F	-
9	F	-
10	F	-
11	F	-
12	F	-

Depth-first search

To visit a vertex v :

- Mark vertex v as visited.
- Recursively visit all unmarked vertices pointing from v .



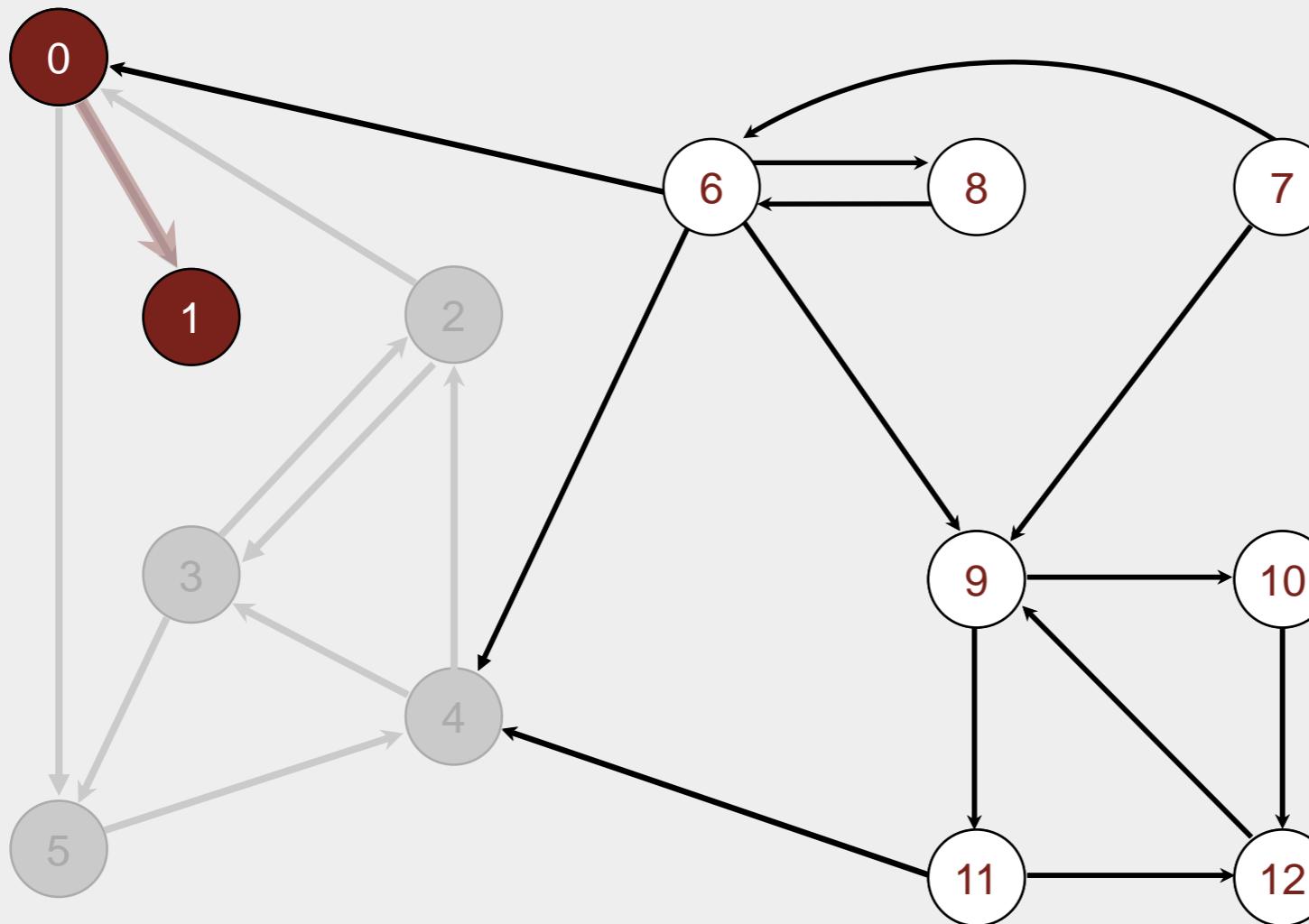
visit 1

v	marked[]	edgeTo[]
0	T	-
1	T	0
2	T	3
3	T	4
4	T	5
5	T	0
6	F	-
7	F	-
8	F	-
9	F	-
10	F	-
11	F	-
12	F	-

Depth-first search

To visit a vertex v :

- Mark vertex v as visited.
- Recursively visit all unmarked vertices pointing from v .



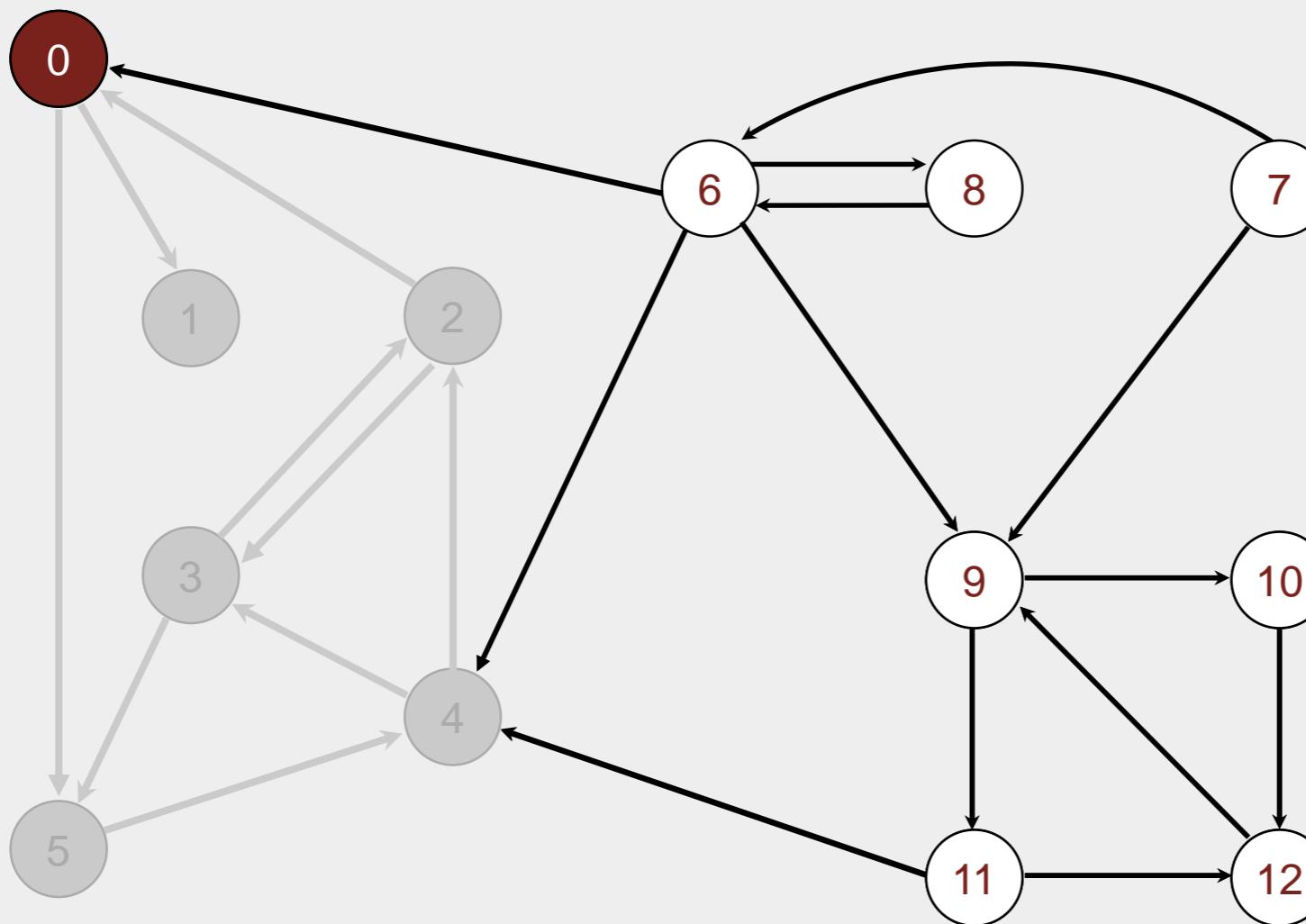
done 1

v	marked[]	edgeTo[]
0	T	-
1	T	0
2	T	3
3	T	4
4	T	5
5	T	0
6	F	-
7	F	-
8	F	-
9	F	-
10	F	-
11	F	-
12	F	-

Depth-first search

To visit a vertex v :

- Mark vertex v as visited.
- Recursively visit all unmarked vertices pointing from v .



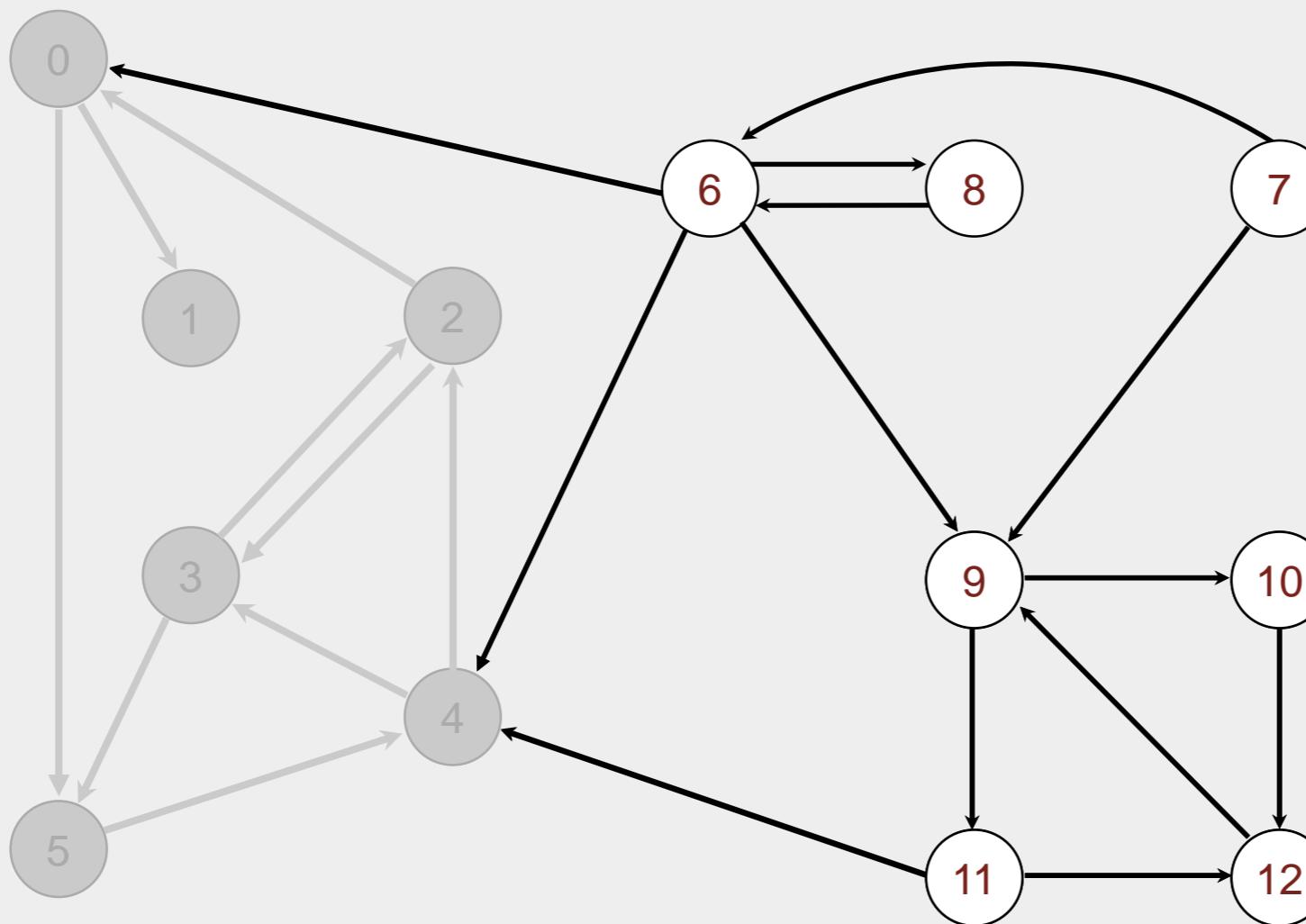
done 0

v	marked[]	edgeTo[]
0	T	-
1	T	0
2	T	3
3	T	4
4	T	5
5	T	0
6	F	-
7	F	-
8	F	-
9	F	-
10	F	-
11	F	-
12	F	-

Depth-first search

To visit a vertex v :

- Mark vertex v as visited.
- Recursively visit all unmarked vertices pointing from v .



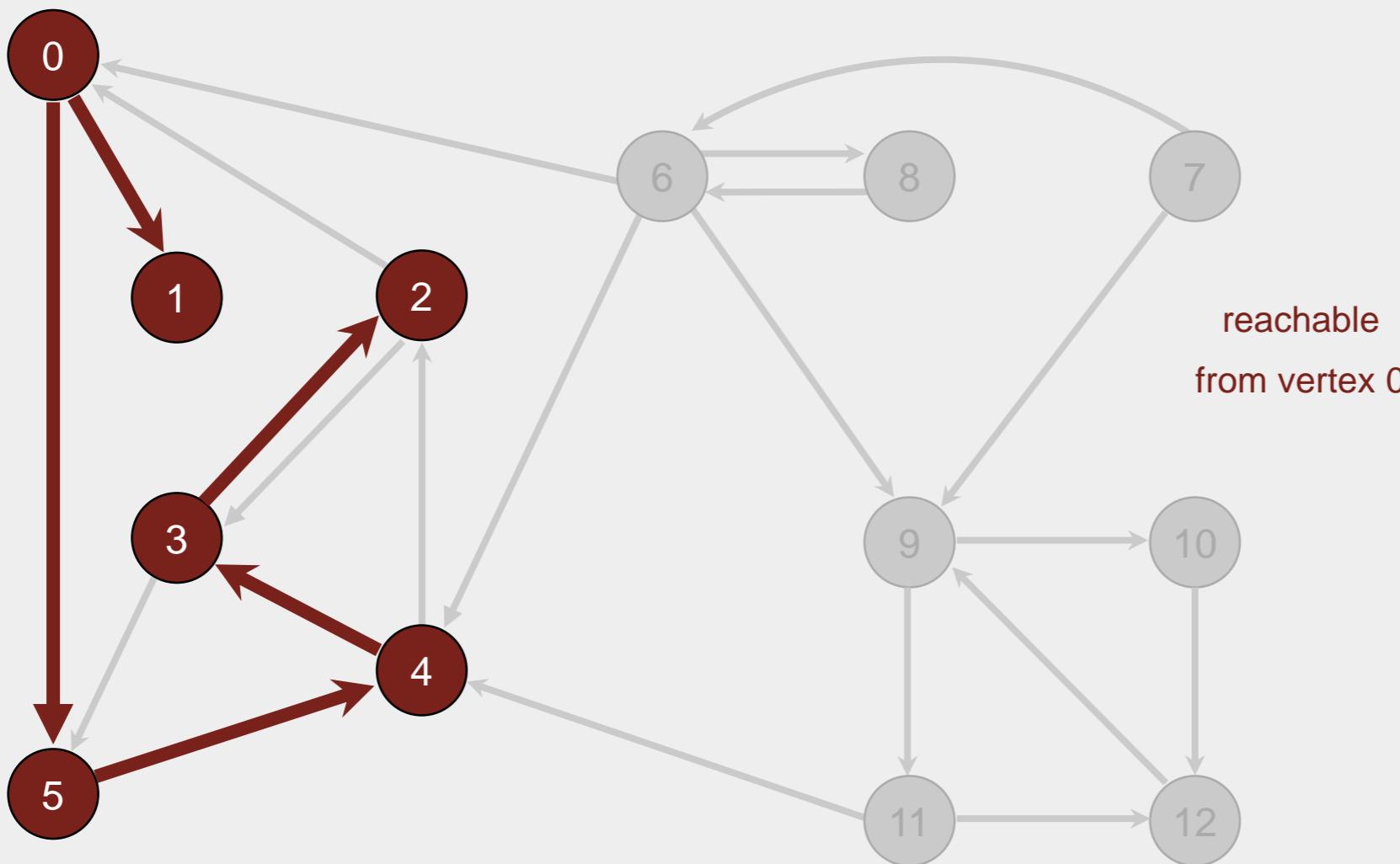
v	marked[]	edgeTo[]
0	T	-
1	T	0
2	T	3
3	T	4
4	T	5
5	T	0
6	F	-
7	F	-
8	F	-
9	F	-
10	F	-
11	F	-
12	F	-

done

Depth-first search

To visit a vertex v :

- Mark vertex v as visited.
- Recursively visit all unmarked vertices pointing from v .



reachable from 0

v	marked[]	edgeTo[]
0	T	-
1	T	0
2	T	3
3	T	4
4	T	5
5	T	0
6	F	-
7	F	-
8	F	-
9	F	-
10	F	-
11	F	-
12	F	-

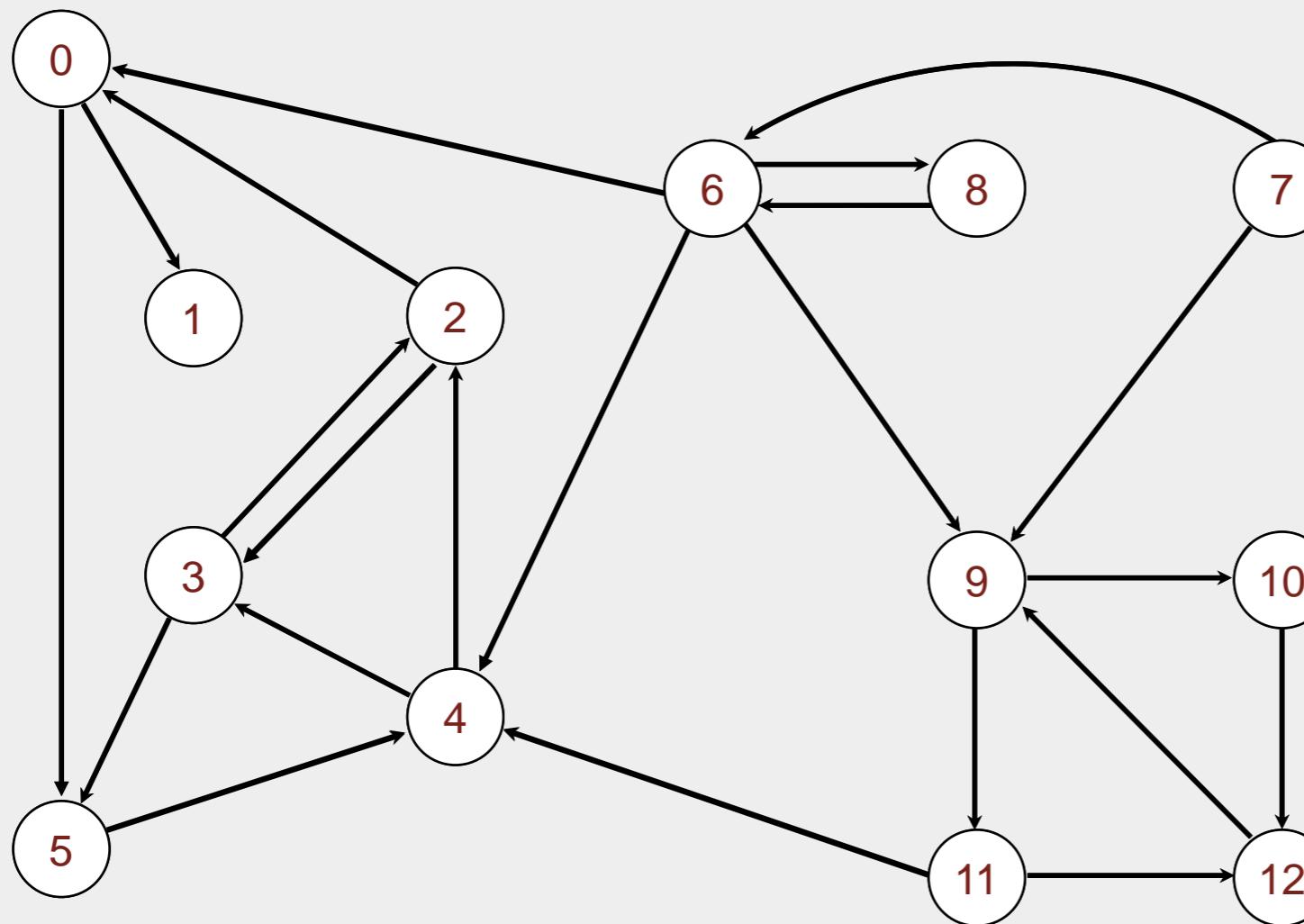
4.2 Kosaraju-Sharir Algorithm



click to begin demo

Phase 1. Compute reverse postorder in G^R .

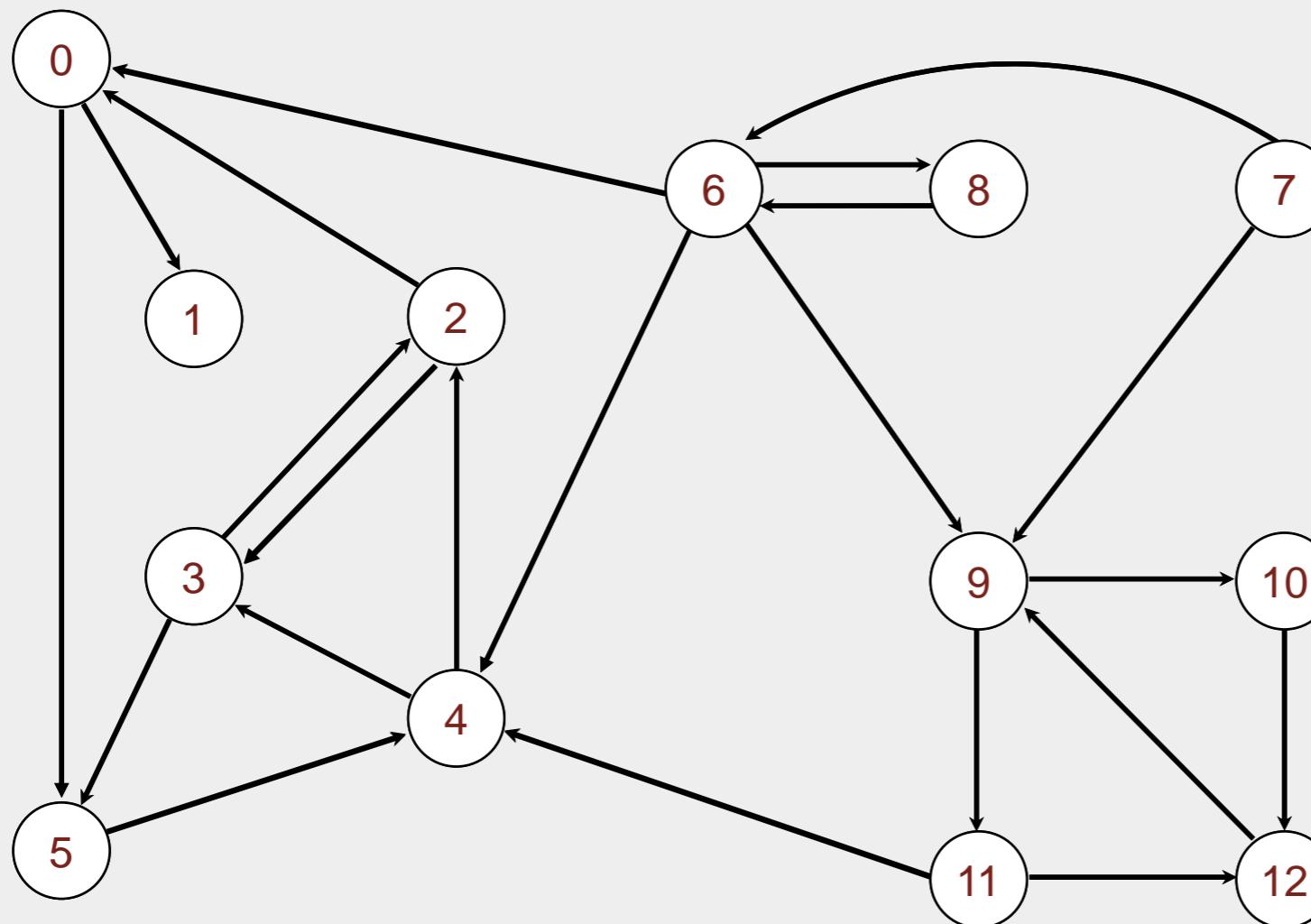
Phase 2. Run DFS in G , visiting unmarked vertices in reverse postorder of G^R .



digraph G

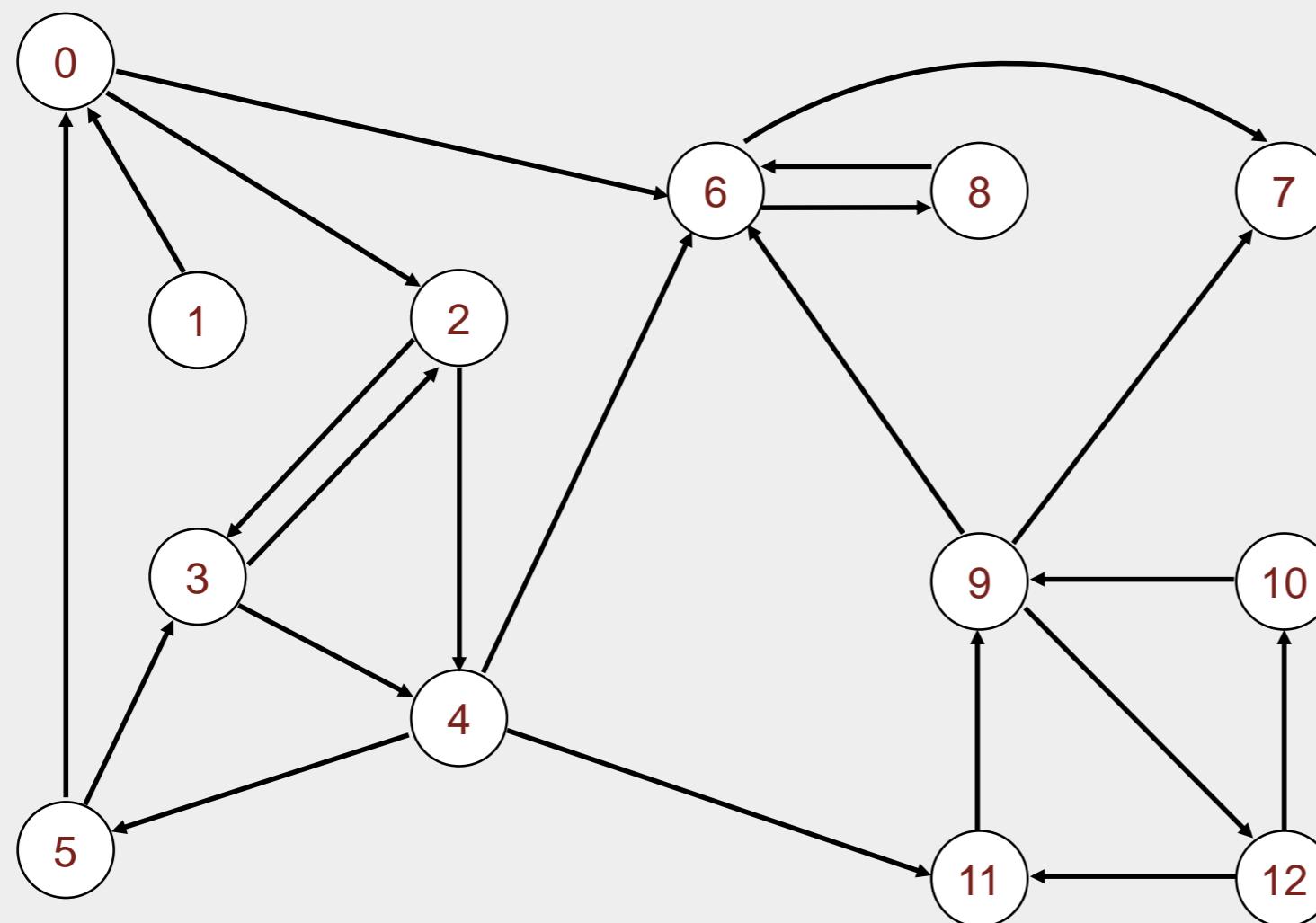
- ▶ DFS in reverse graph
- ▶ DFS in original graph

Phase 1. Compute reverse postorder in G^R .



digraph G

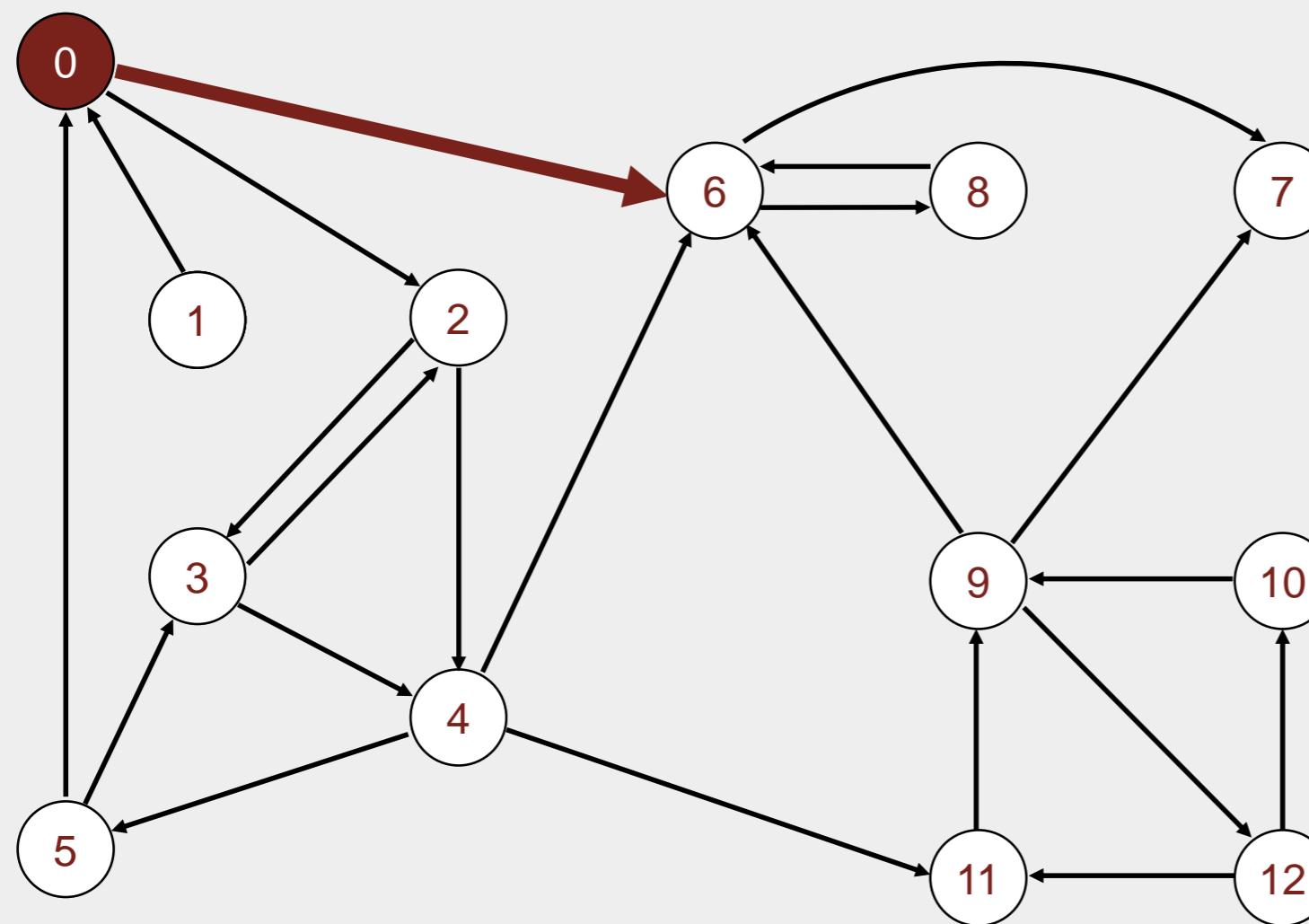
Phase 1. Compute reverse postorder in G^R .



reverse digraph G^R

v	marked[v]
0	-
1	-
2	-
3	-
4	-
5	-
6	-
7	-
8	-
9	-
10	-
11	-
12	-

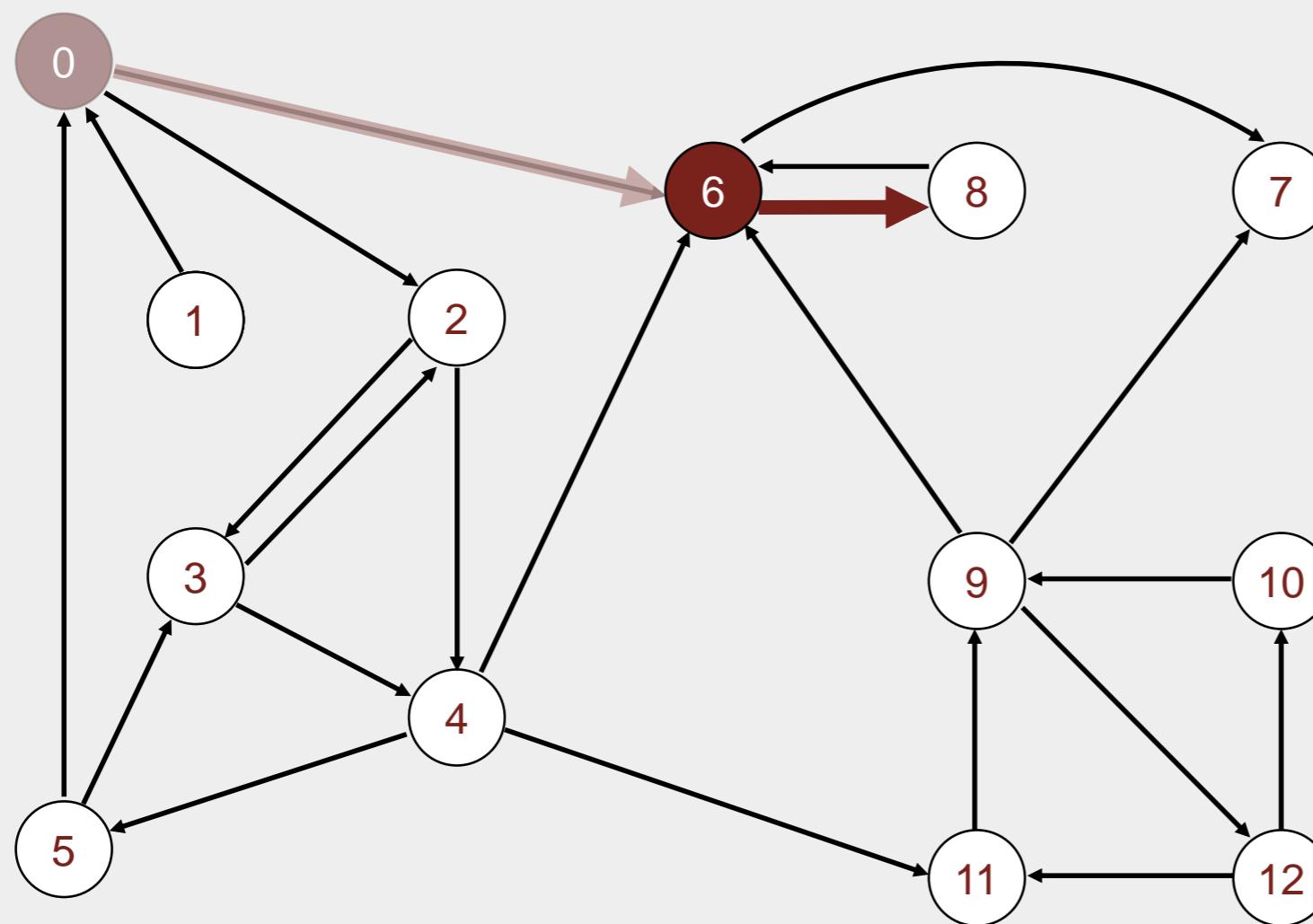
Phase 1. Compute reverse postorder in G^R .



v	marked[v]
0	T
1	F
2	F
3	F
4	F
5	F
6	F
7	F
8	F
9	F
10	F
11	F
12	F

visit 0

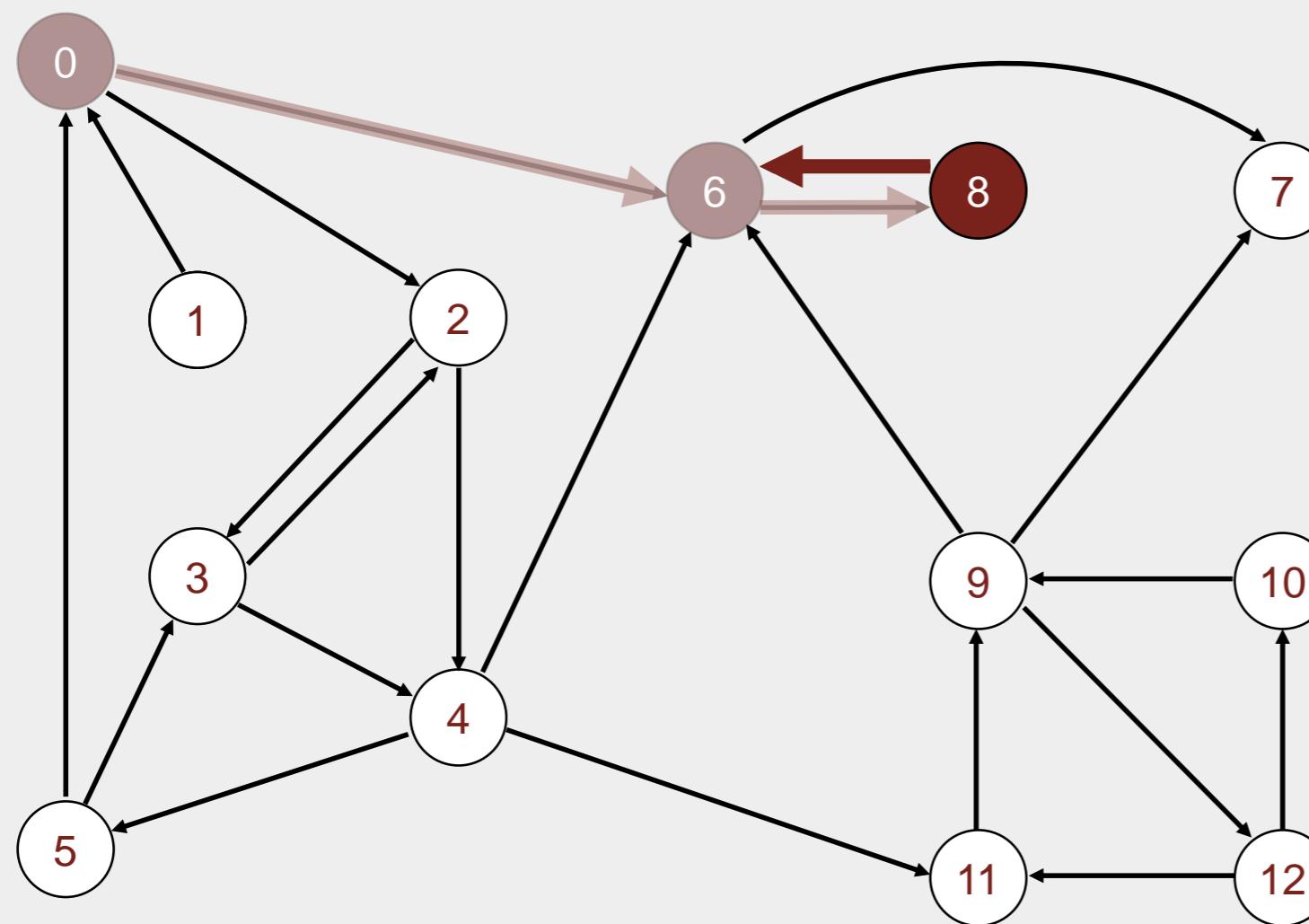
Phase 1. Compute reverse postorder in G^R .



visit 6

v	marked[v]
0	T
1	F
2	F
3	F
4	F
5	F
6	T
7	F
8	F
9	F
10	F
11	F
12	F

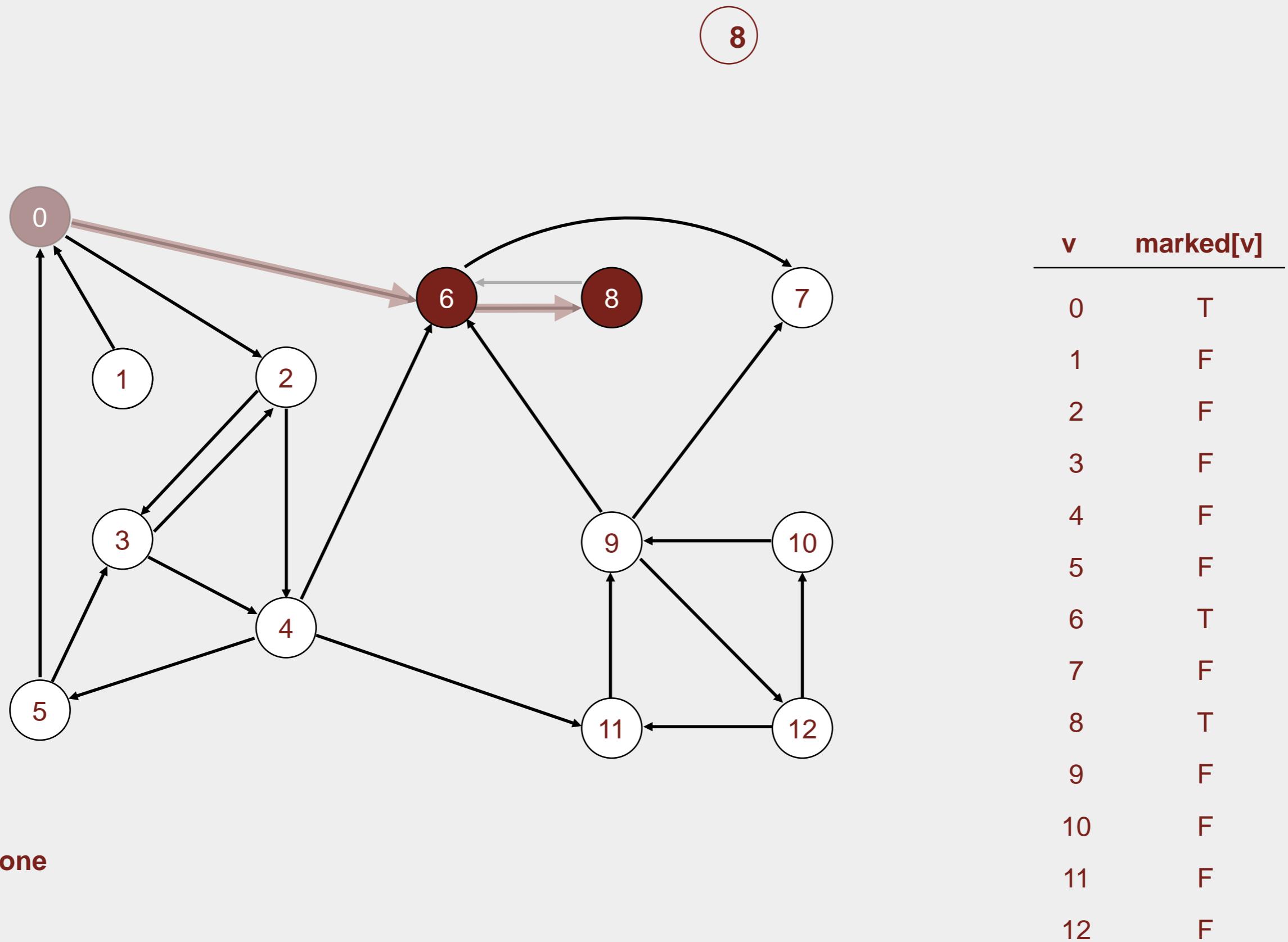
Phase 1. Compute reverse postorder in G^R .



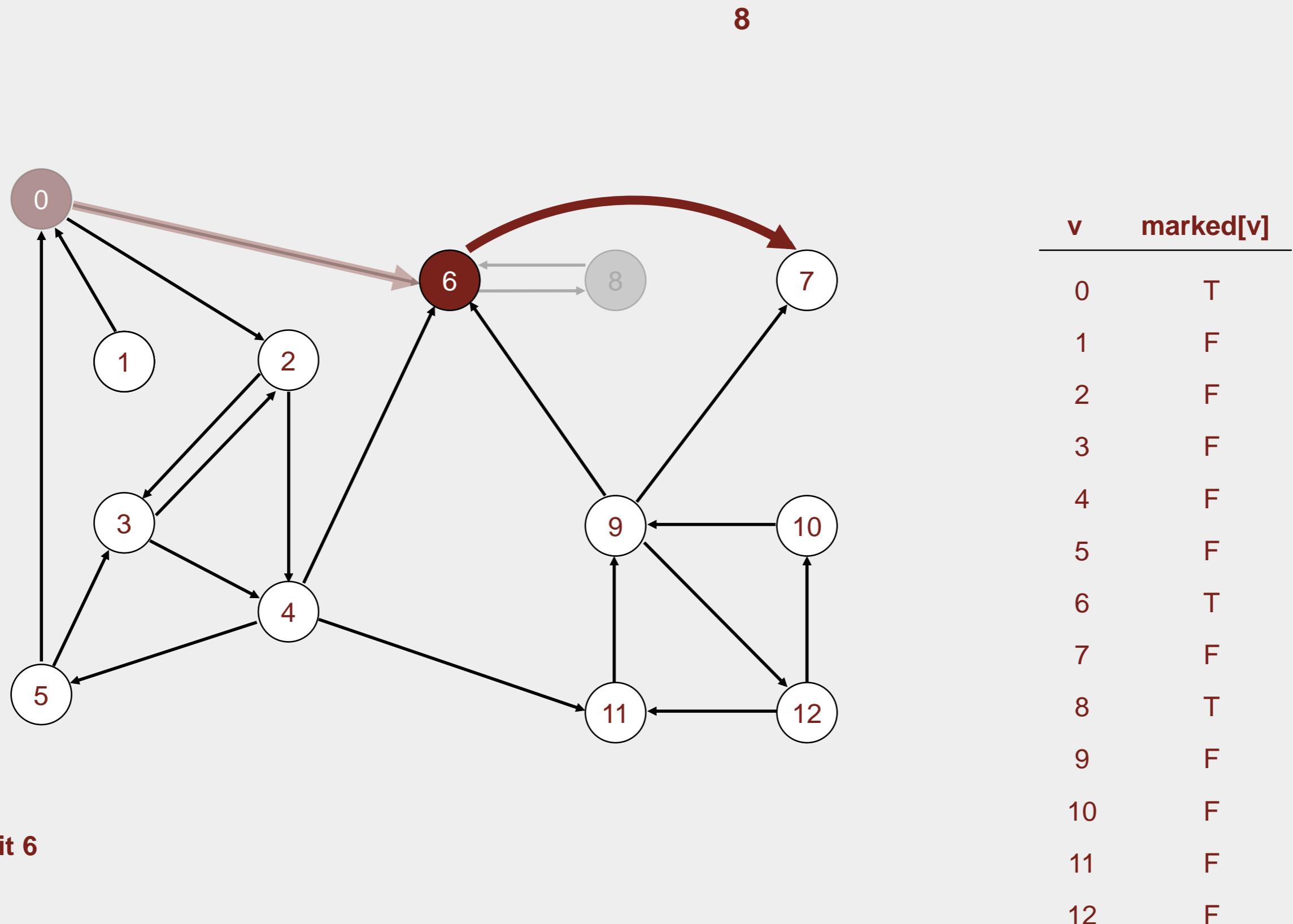
v	marked[v]
0	T
1	F
2	F
3	F
4	F
5	F
6	T
7	F
8	T
9	F
10	F
11	F
12	F

visit 8

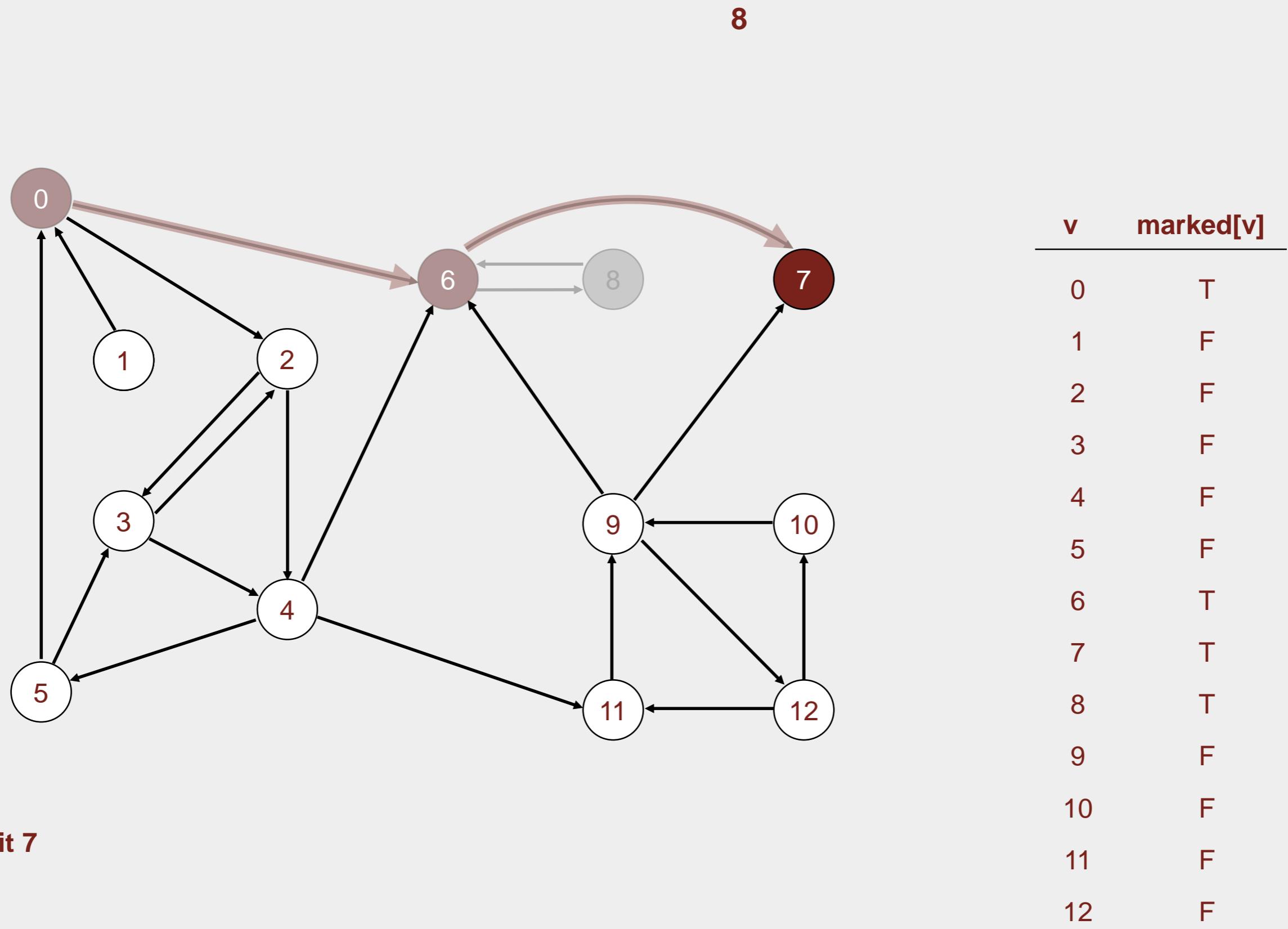
Phase 1. Compute reverse postorder in G^R .



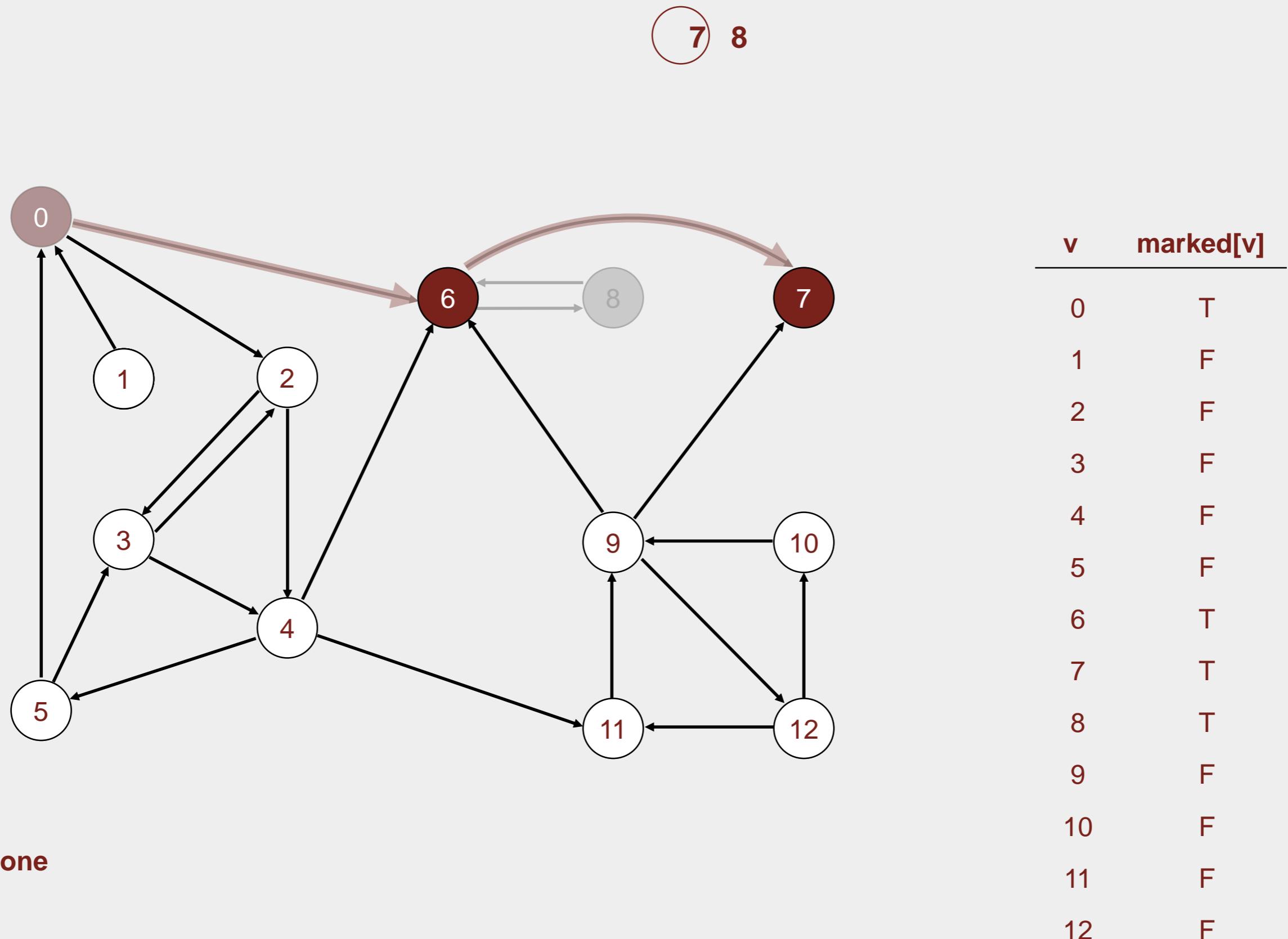
Phase 1. Compute reverse postorder in G^R .



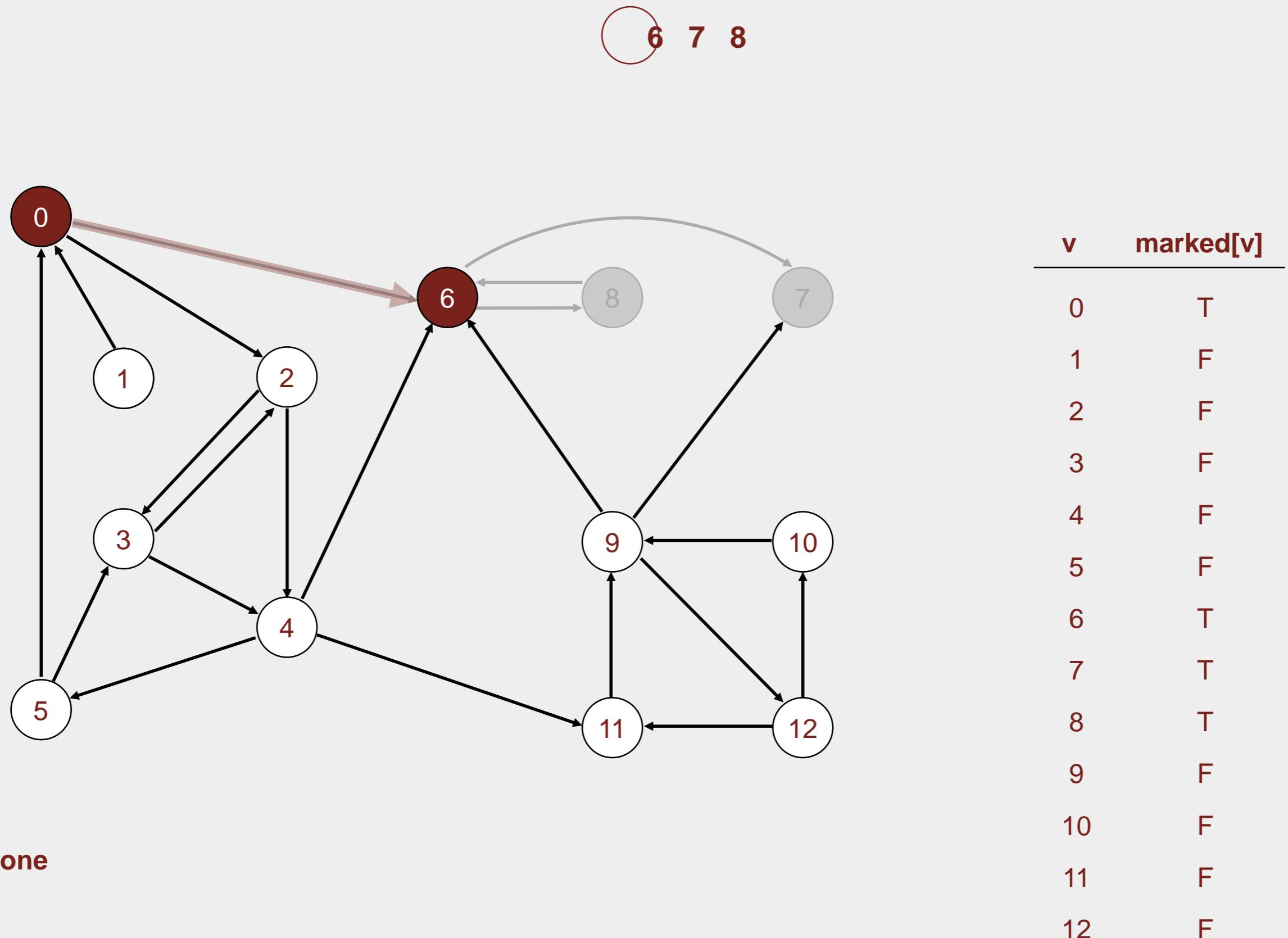
Phase 1. Compute reverse postorder in G^R .



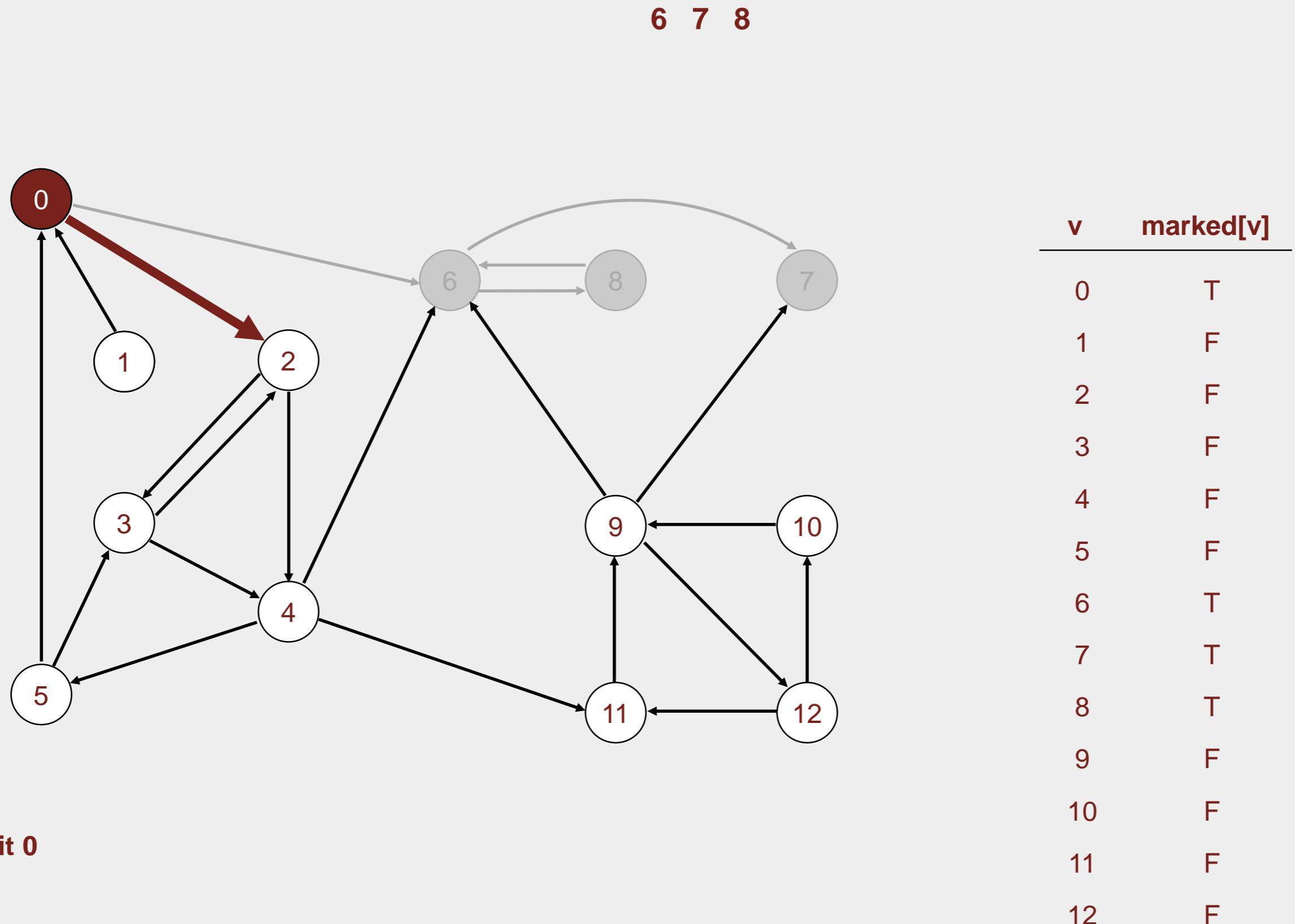
Phase 1. Compute reverse postorder in G^R .



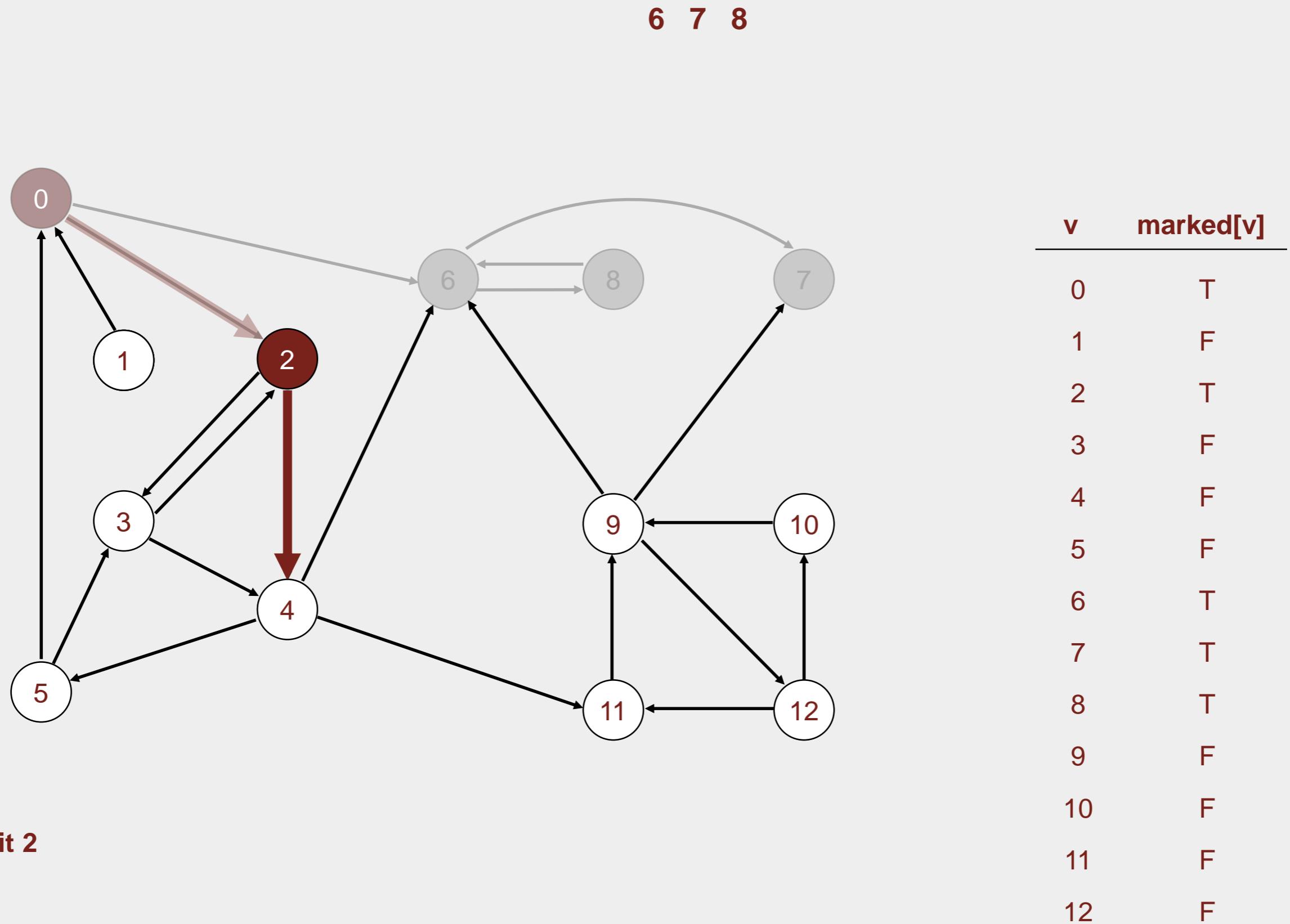
Phase 1. Compute reverse postorder in G^R .



Phase 1. Compute reverse postorder in G^R .

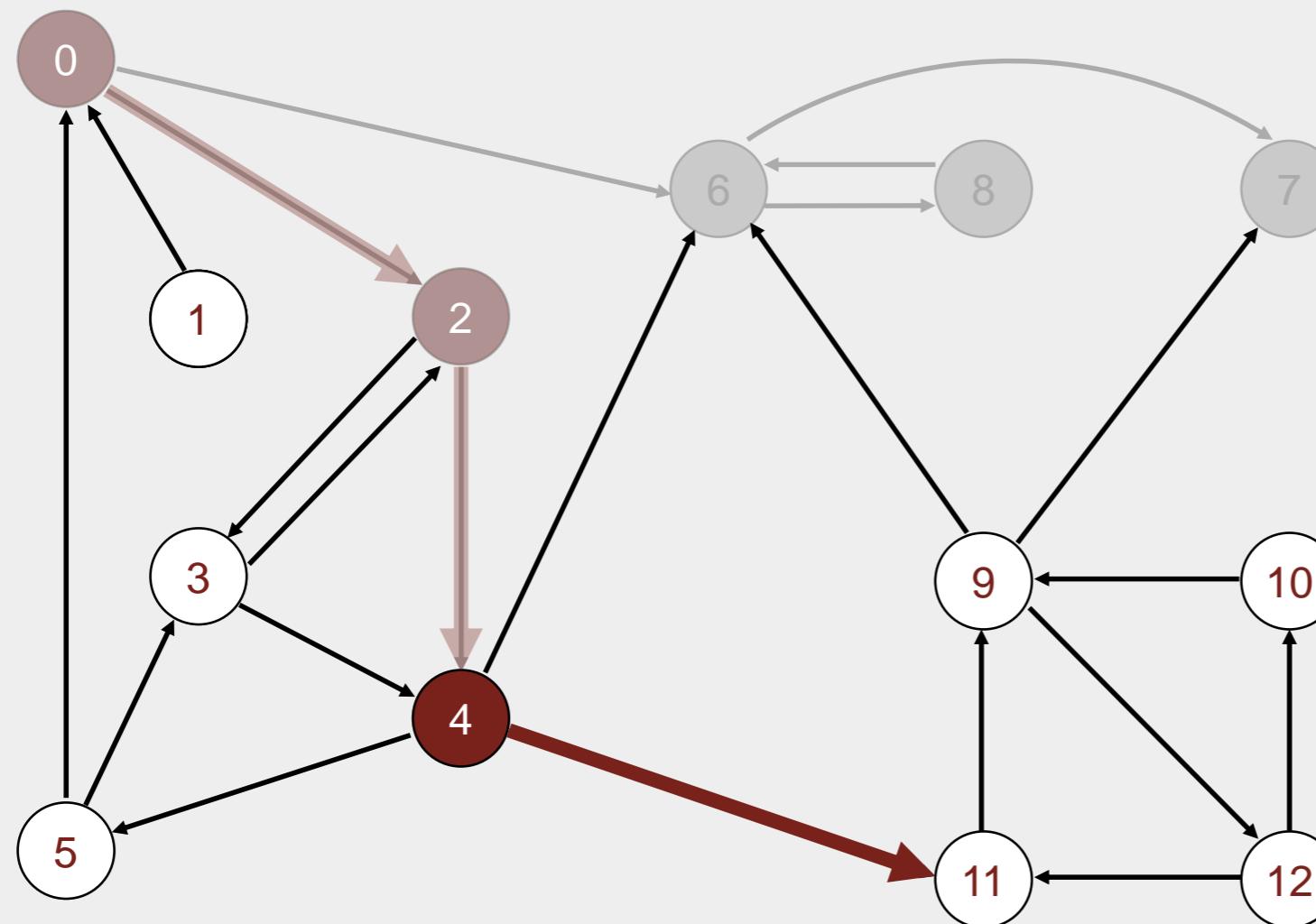


Phase 1. Compute reverse postorder in G^R .



Phase 1. Compute reverse postorder in G^R .

6 7 8

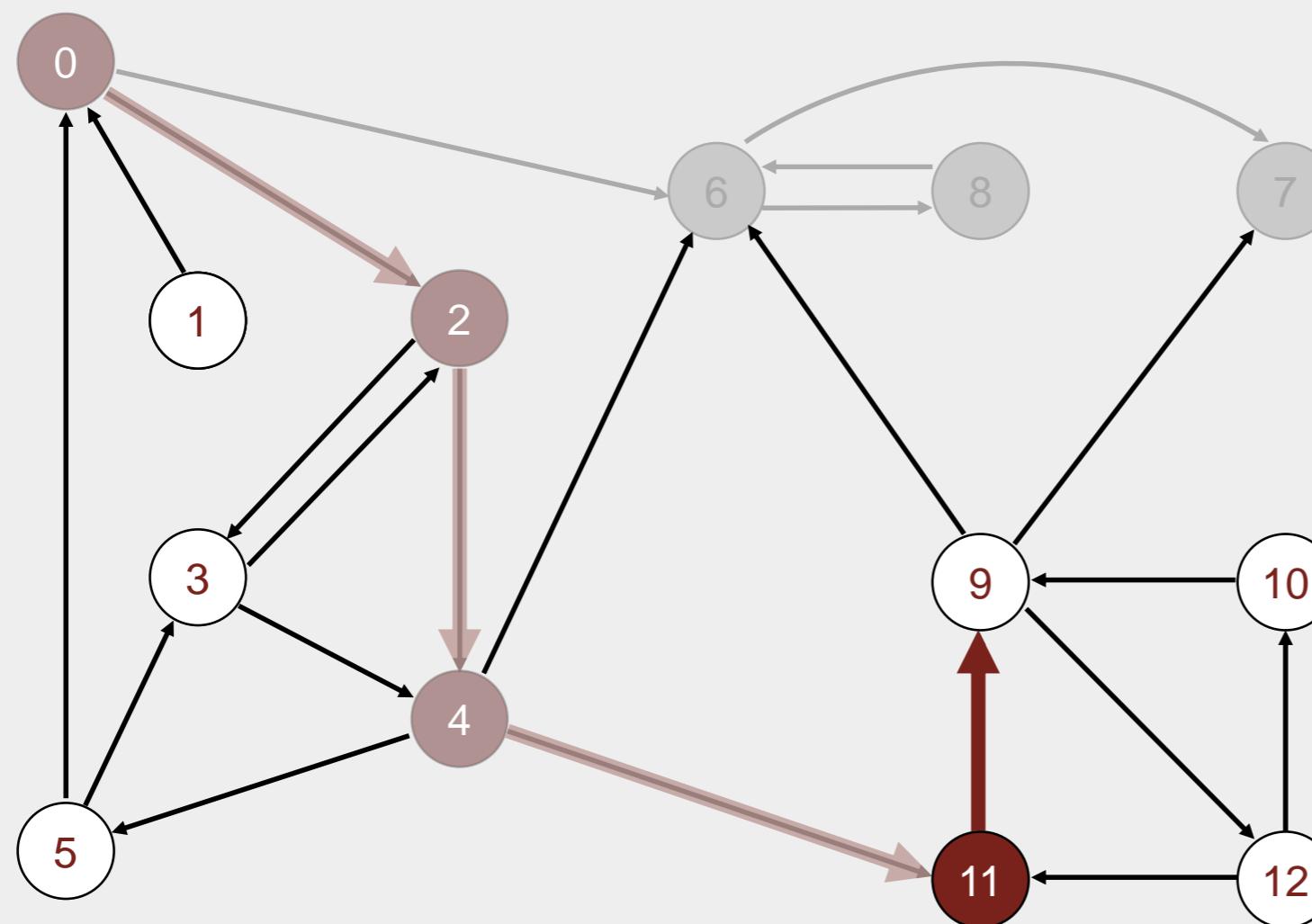


visit 4

v	marked[v]
0	T
1	F
2	T
3	F
4	T
5	F
6	T
7	T
8	T
9	F
10	F
11	F
12	F

Phase 1. Compute reverse postorder in G^R .

6 7 8

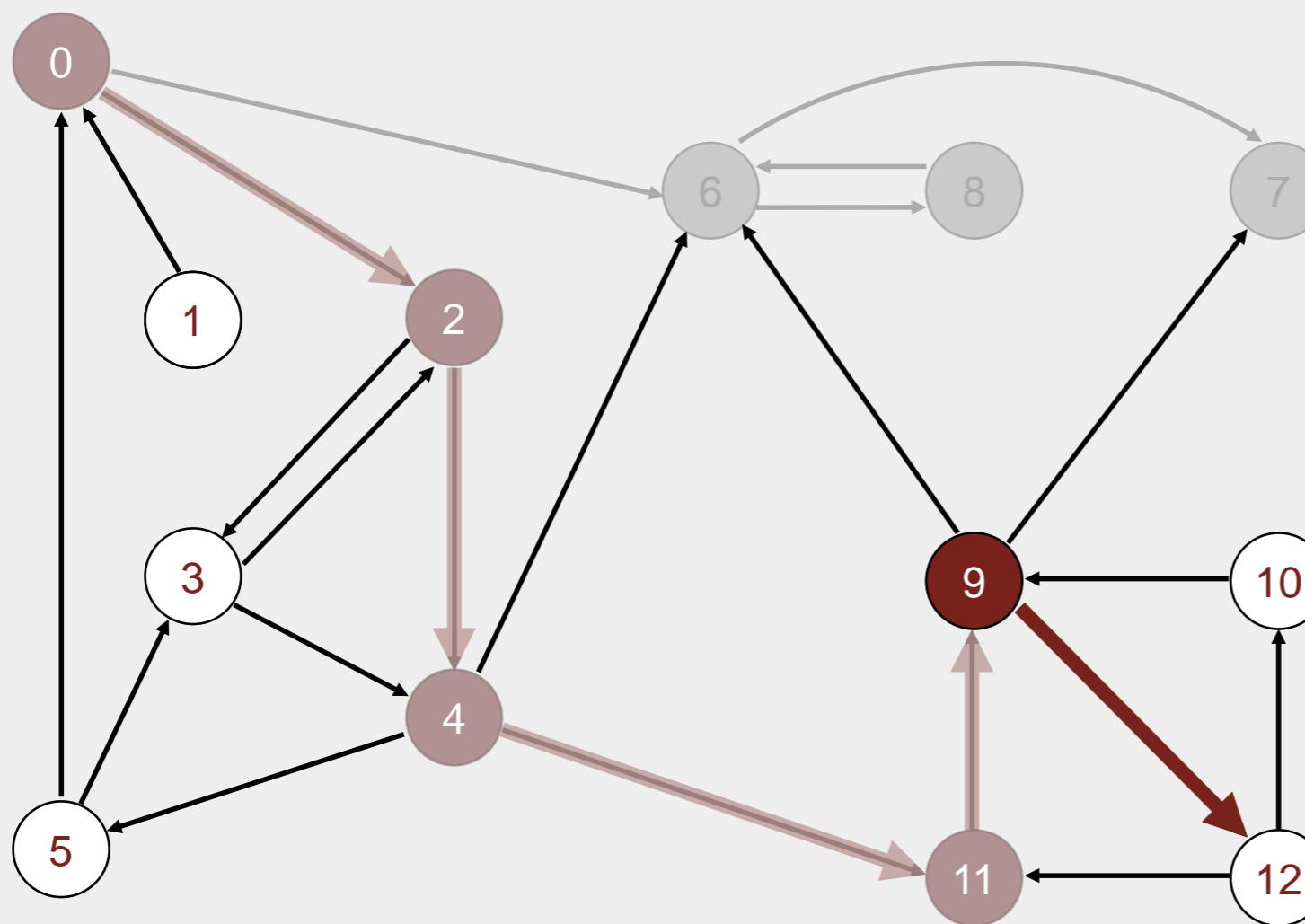


visit 11

v	marked[v]
0	T
1	F
2	T
3	F
4	T
5	F
6	T
7	T
8	T
9	F
10	F
11	T
12	F

Phase 1. Compute reverse postorder in G^R .

6 7 8

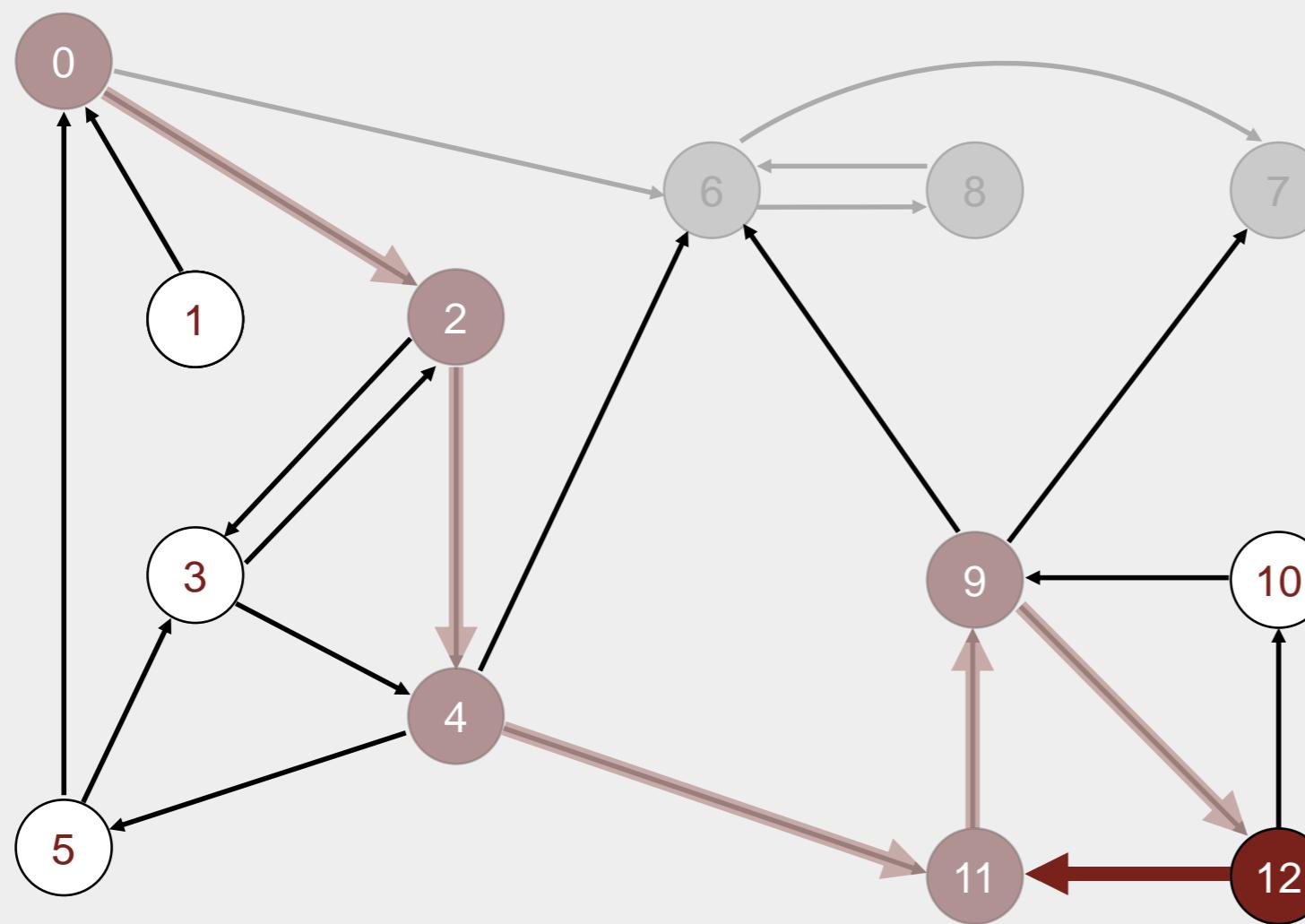


v	marked[v]
0	T
1	F
2	T
3	F
4	T
5	F
6	T
7	T
8	T
9	T
10	F
11	T
12	F

visit 9

Phase 1. Compute reverse postorder in G^R .

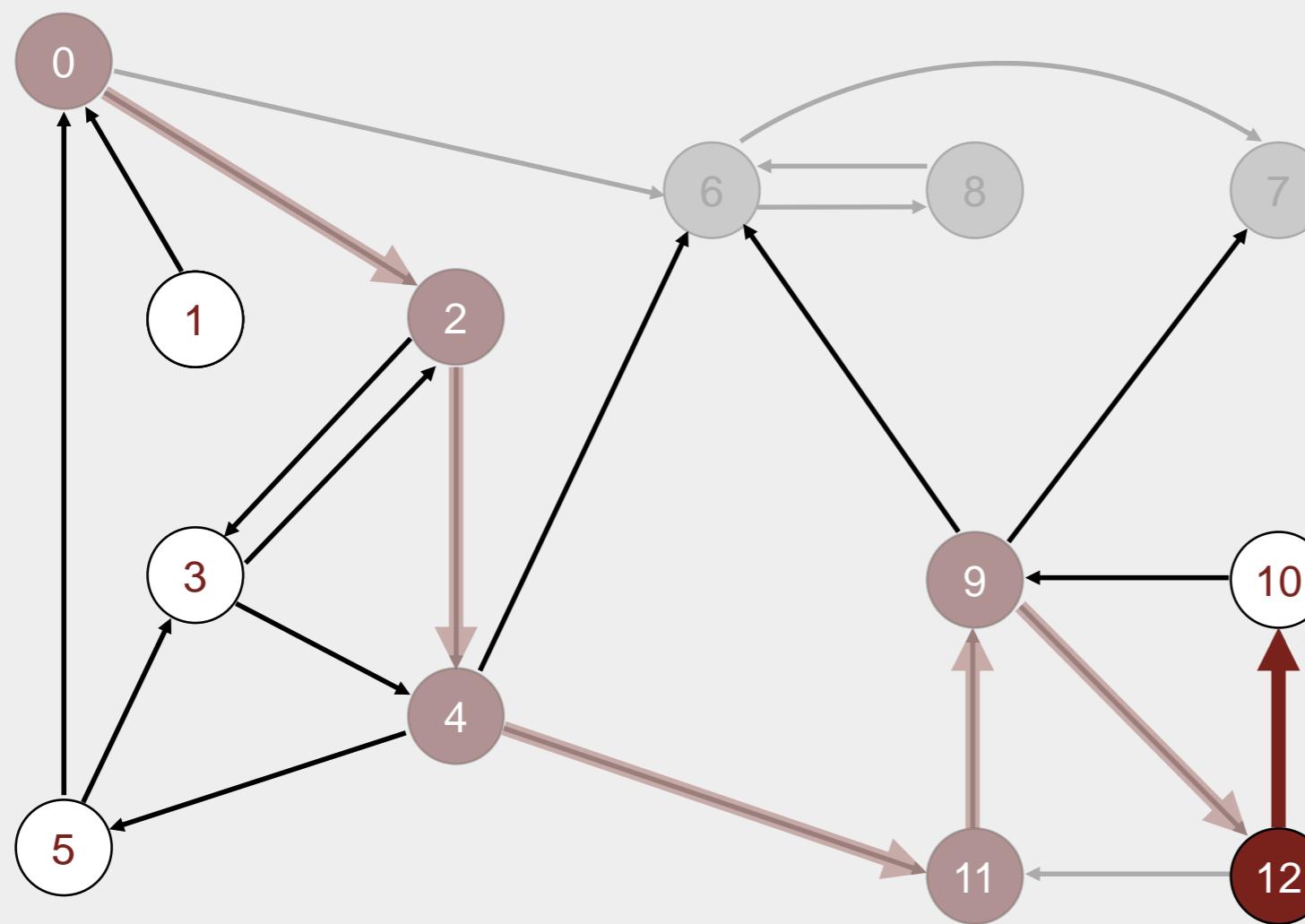
6 7 8



v	marked[v]
0	T
1	F
2	T
3	F
4	T
5	F
6	T
7	T
8	T
9	T
10	F
11	T
12	T

Phase 1. Compute reverse postorder in G^R .

6 7 8

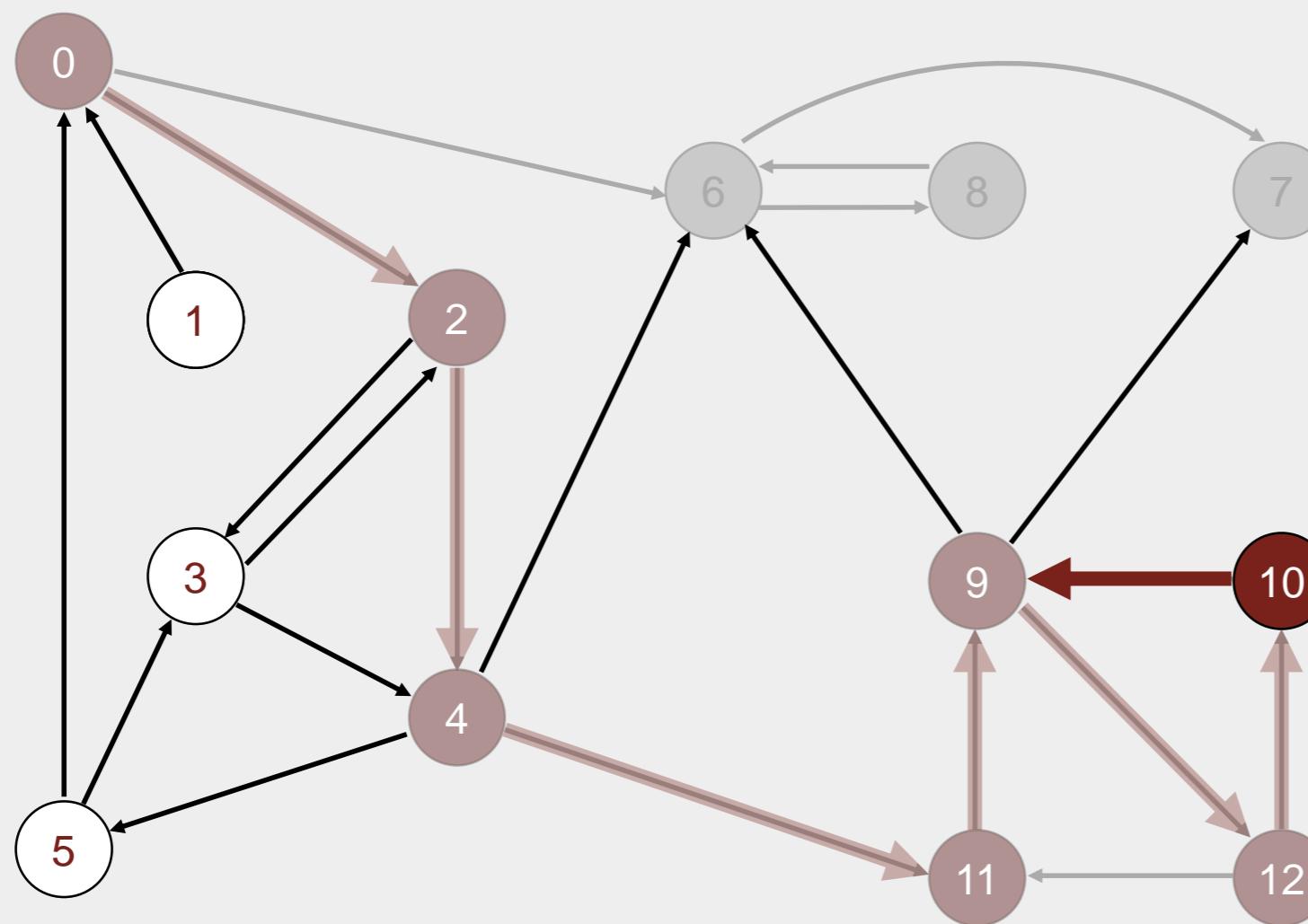


v	marked[v]
0	T
1	F
2	T
3	F
4	T
5	F
6	T
7	T
8	T
9	T
10	F
11	T
12	T

visit 12

Phase 1. Compute reverse postorder in G^R .

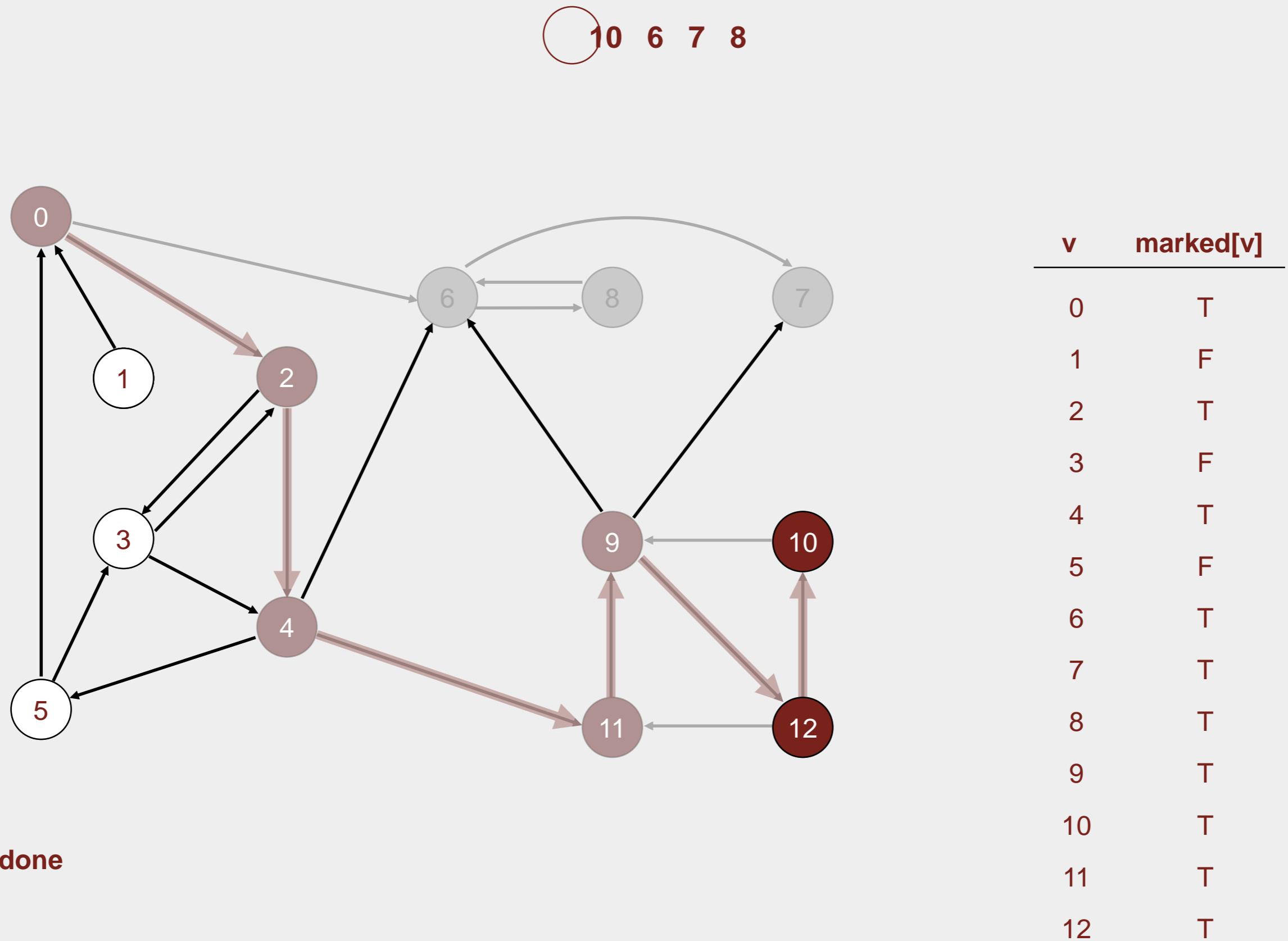
6 7 8



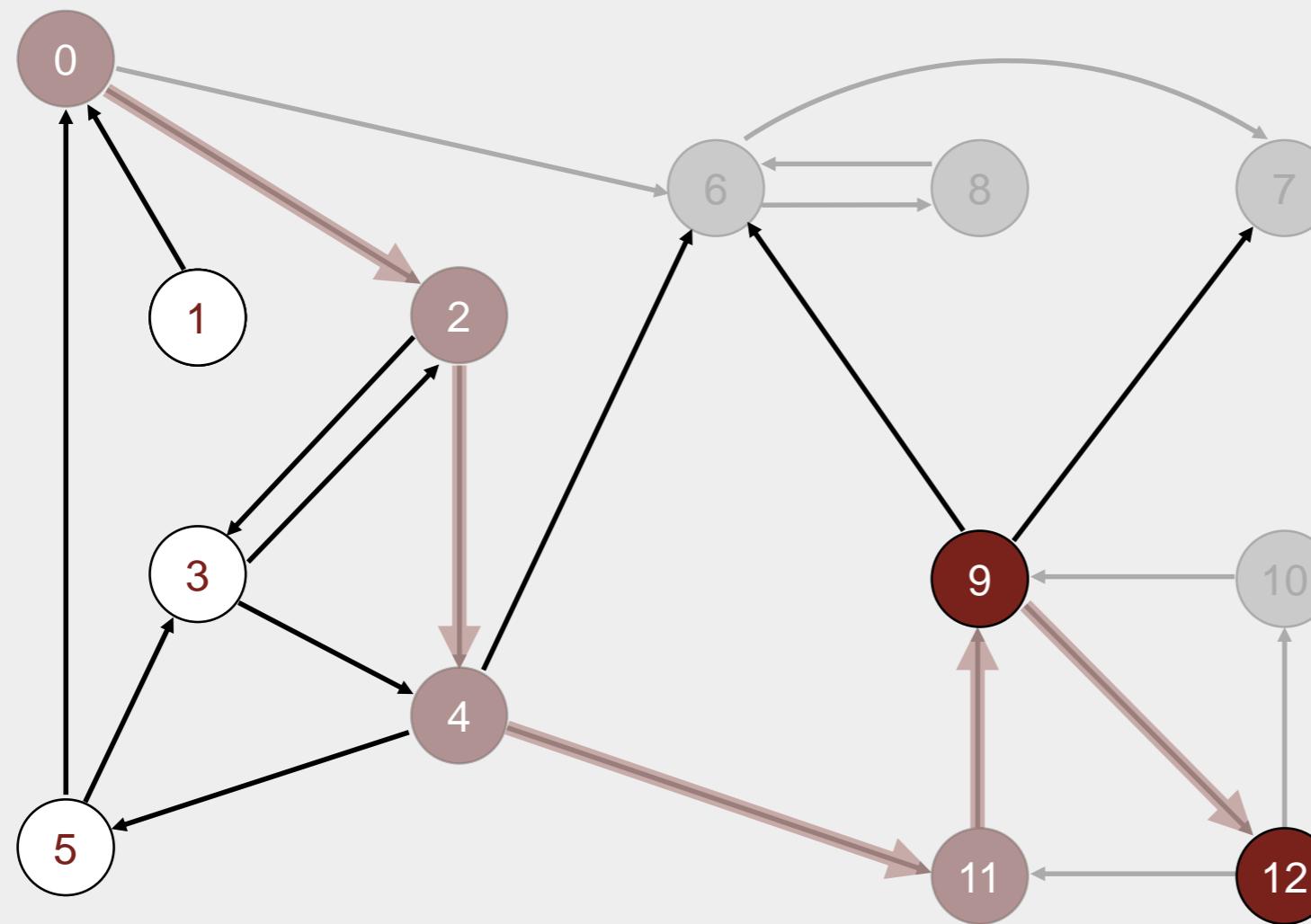
v	marked[v]
0	T
1	F
2	T
3	F
4	T
5	F
6	T
7	T
8	T
9	T
10	T
11	T
12	T

visit 10

Phase 1. Compute reverse postorder in G^R .



Phase 1. Compute reverse postorder in G^R .

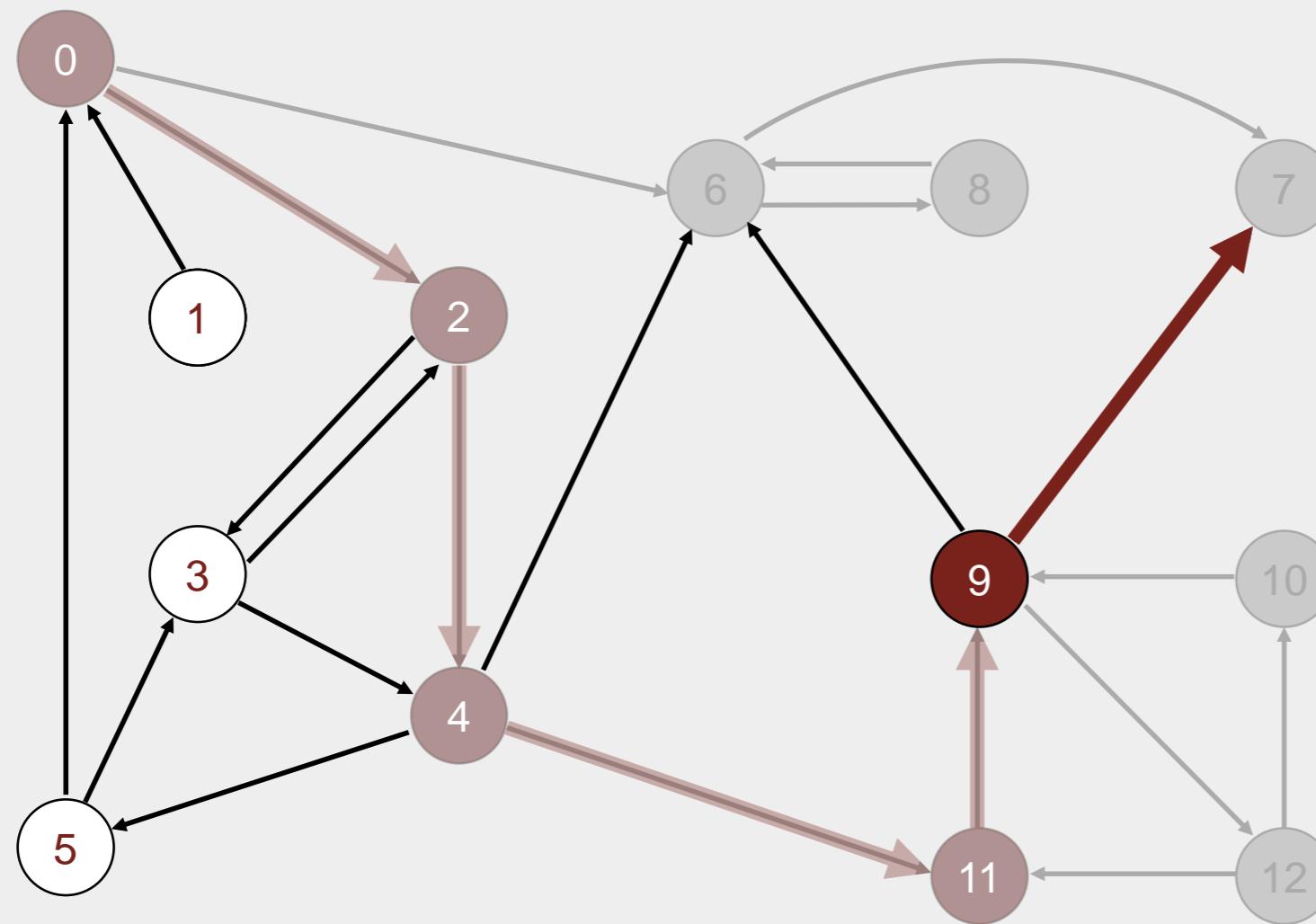


v	marked[v]
0	T
1	F
2	T
3	F
4	T
5	F
6	T
7	T
8	T
9	T
10	T
11	T
12	T

12 done

Phase 1. Compute reverse postorder in G^R .

12 10 6 7 8

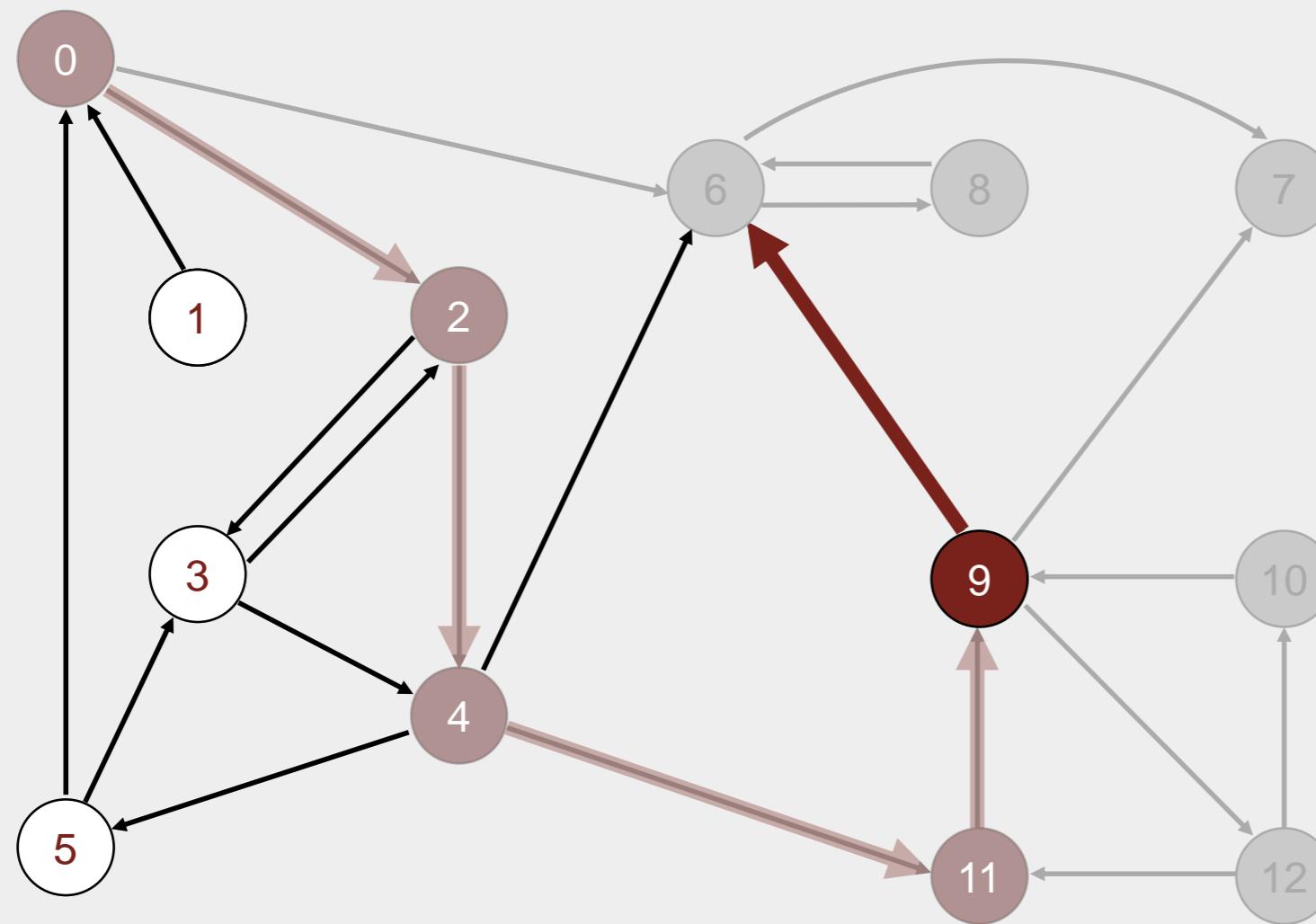


visit 9

v	marked[v]
0	T
1	F
2	T
3	F
4	T
5	F
6	T
7	T
8	T
9	T
10	T
11	T
12	T

Phase 1. Compute reverse postorder in G^R .

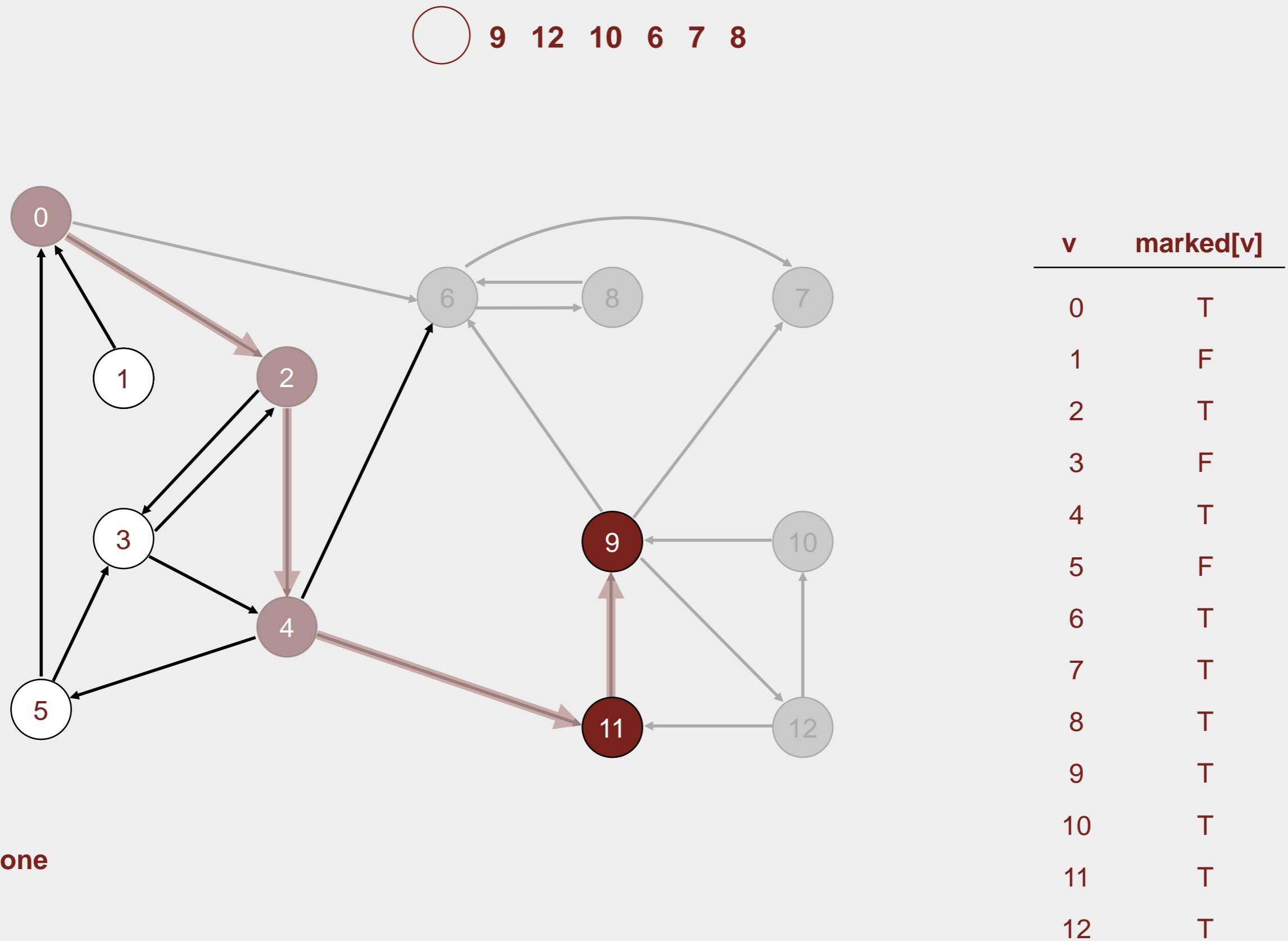
12 10 6 7 8



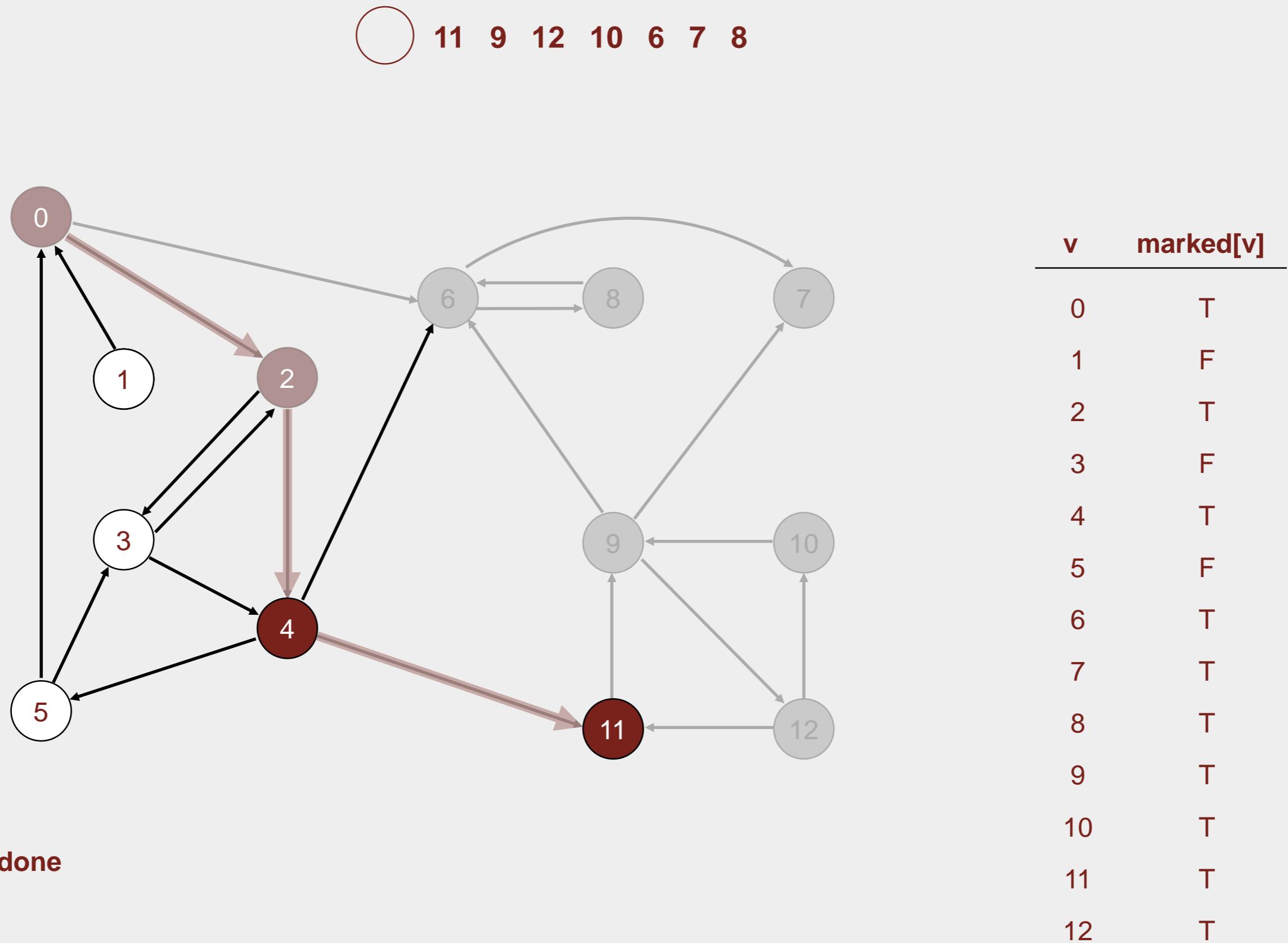
visit 9

v	marked[v]
0	T
1	F
2	T
3	F
4	T
5	F
6	T
7	T
8	T
9	T
10	T
11	T
12	T

Phase 1. Compute reverse postorder in G^R .

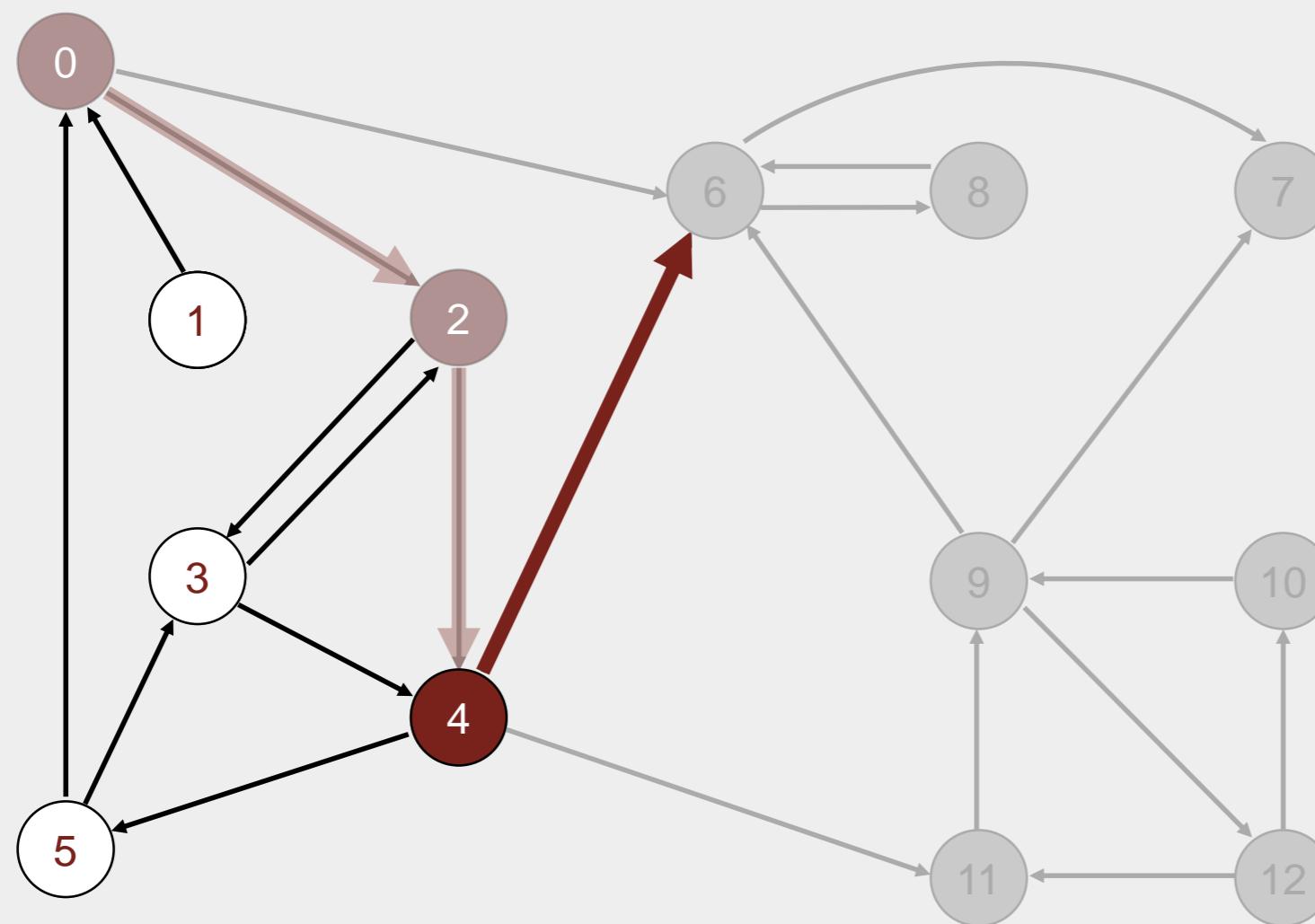


Phase 1. Compute reverse postorder in G^R .



Phase 1. Compute reverse postorder in G^R .

11 9 12 10 6 7 8

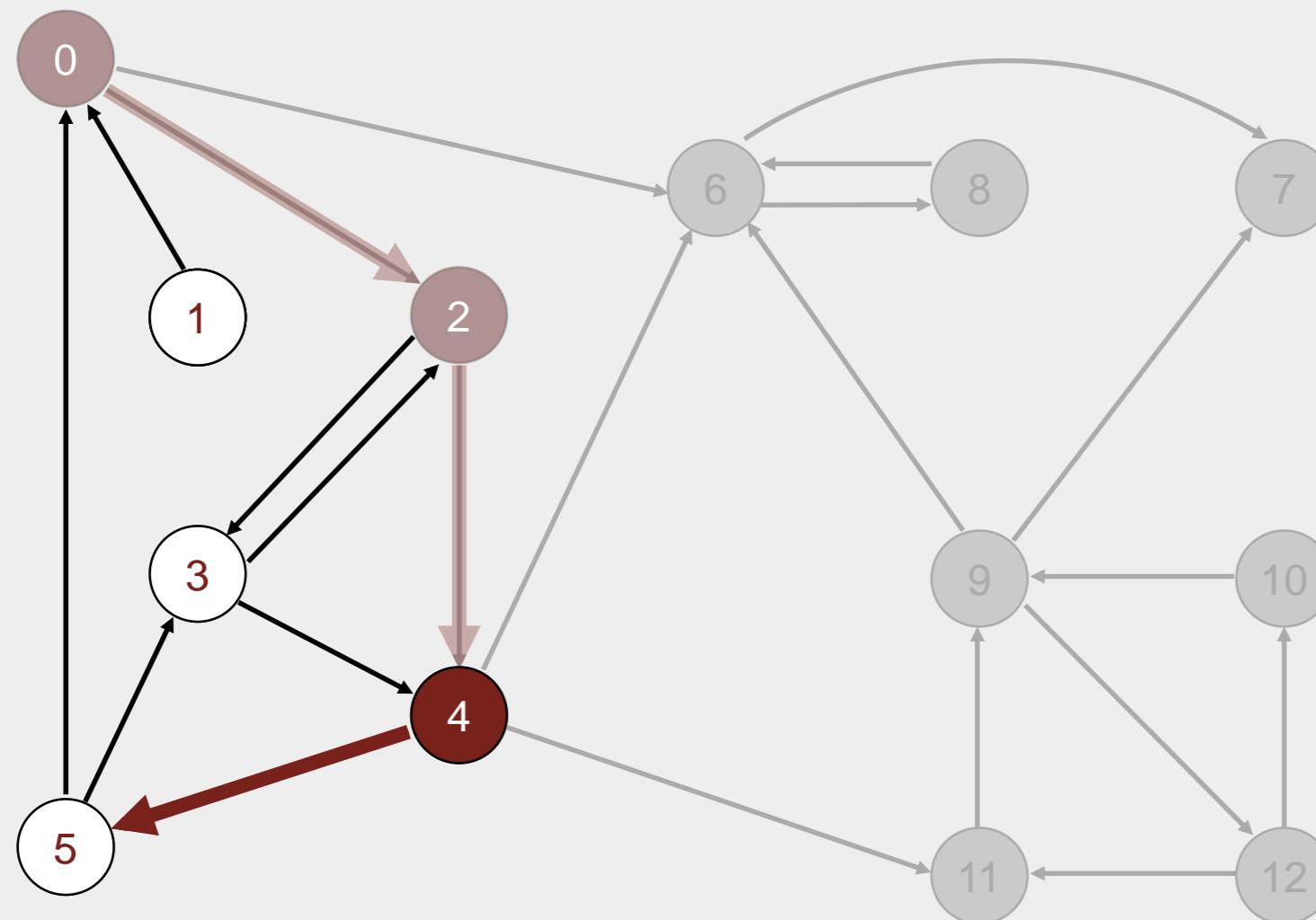


visit 4

v	marked[v]
0	T
1	F
2	T
3	F
4	T
5	F
6	T
7	T
8	T
9	T
10	T
11	T
12	T

Phase 1. Compute reverse postorder in G^R .

11 9 12 10 6 7 8

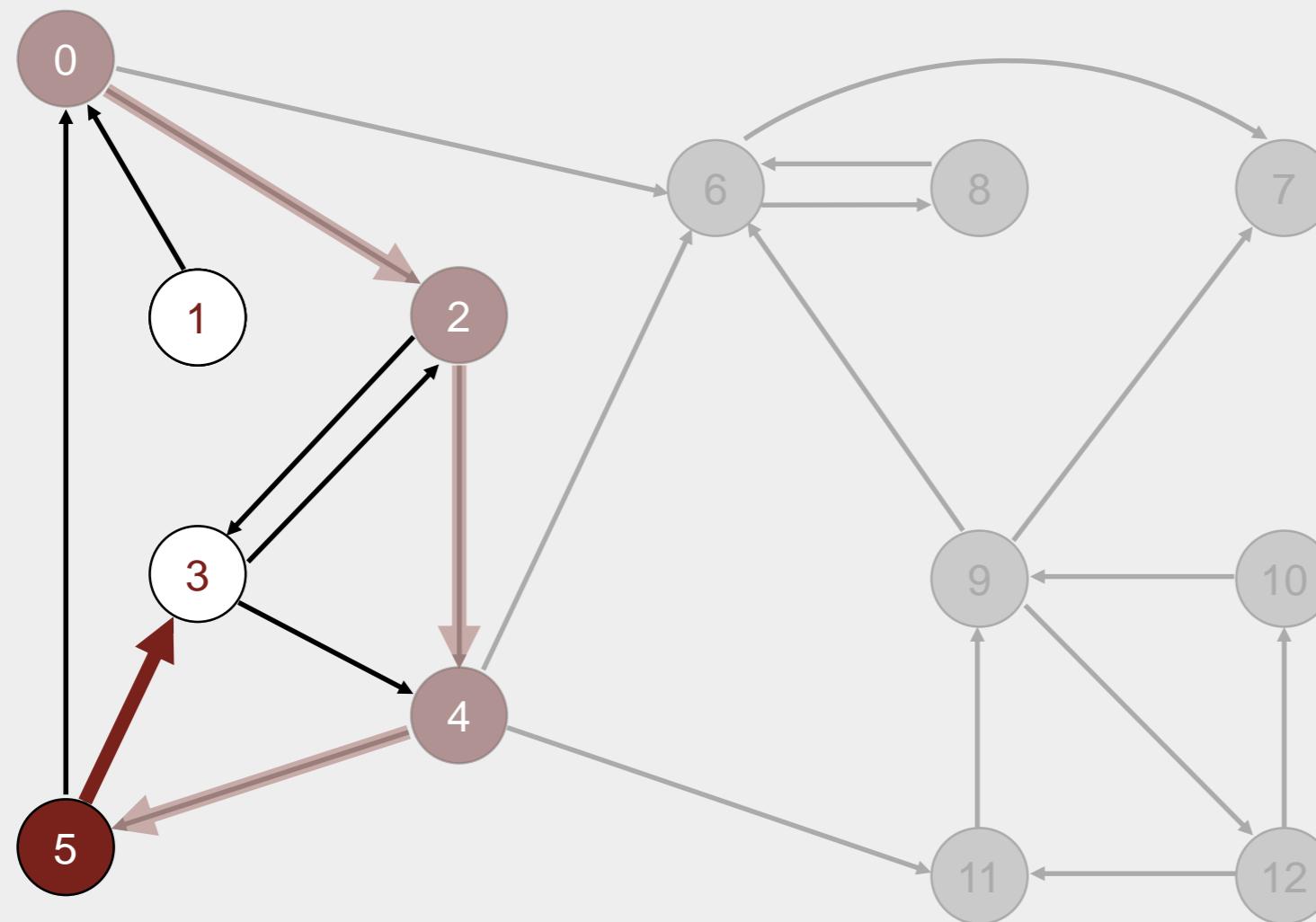


visit 4

v	marked[v]
0	T
1	F
2	T
3	F
4	T
5	F
6	T
7	T
8	T
9	T
10	T
11	T
12	T

Phase 1. Compute reverse postorder in G^R .

11 9 12 10 6 7 8

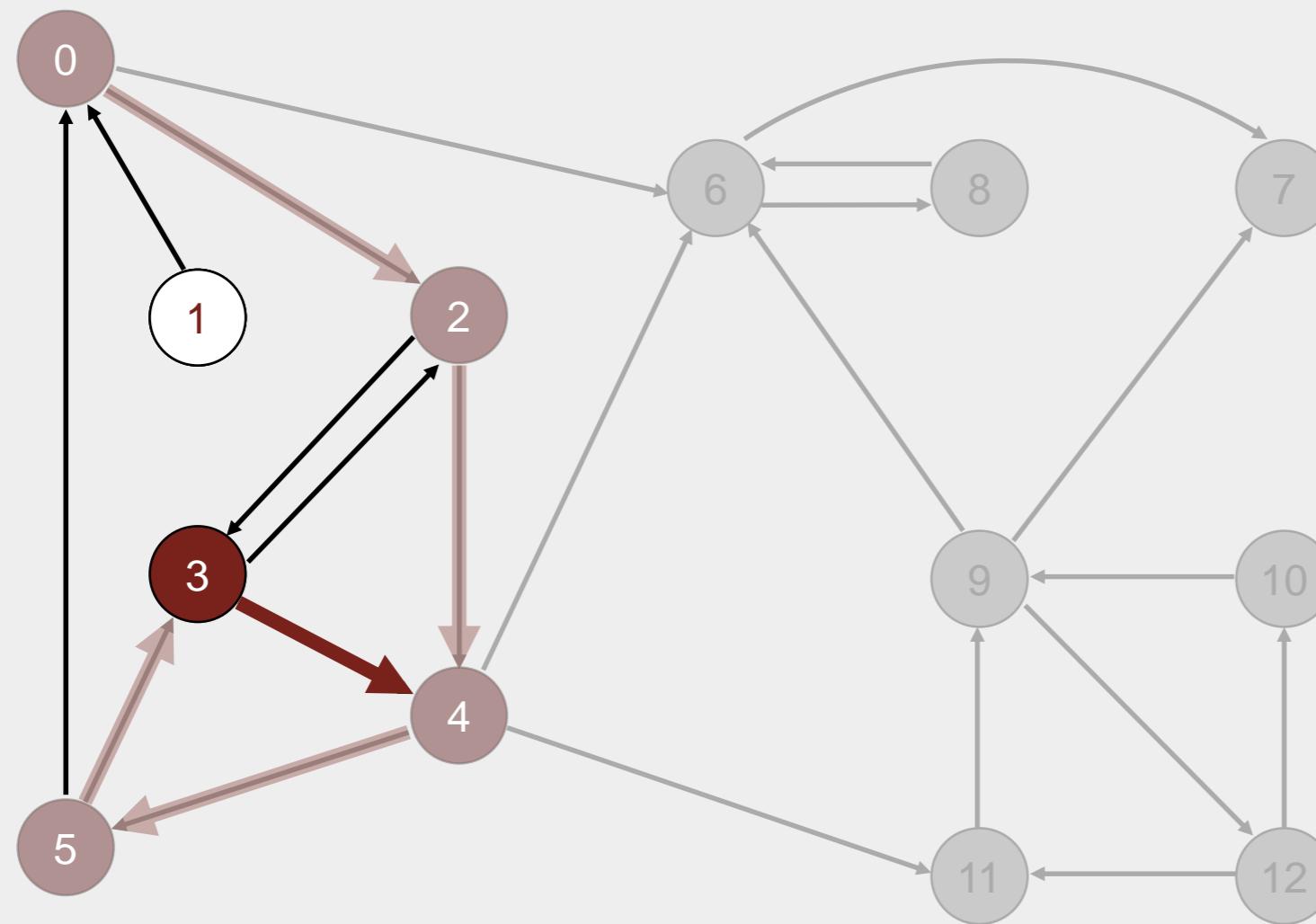


visit 5

v	marked[v]
0	T
1	F
2	T
3	F
4	T
5	T
6	T
7	T
8	T
9	T
10	T
11	T
12	T

Phase 1. Compute reverse postorder in G^R .

11 9 12 10 6 7 8

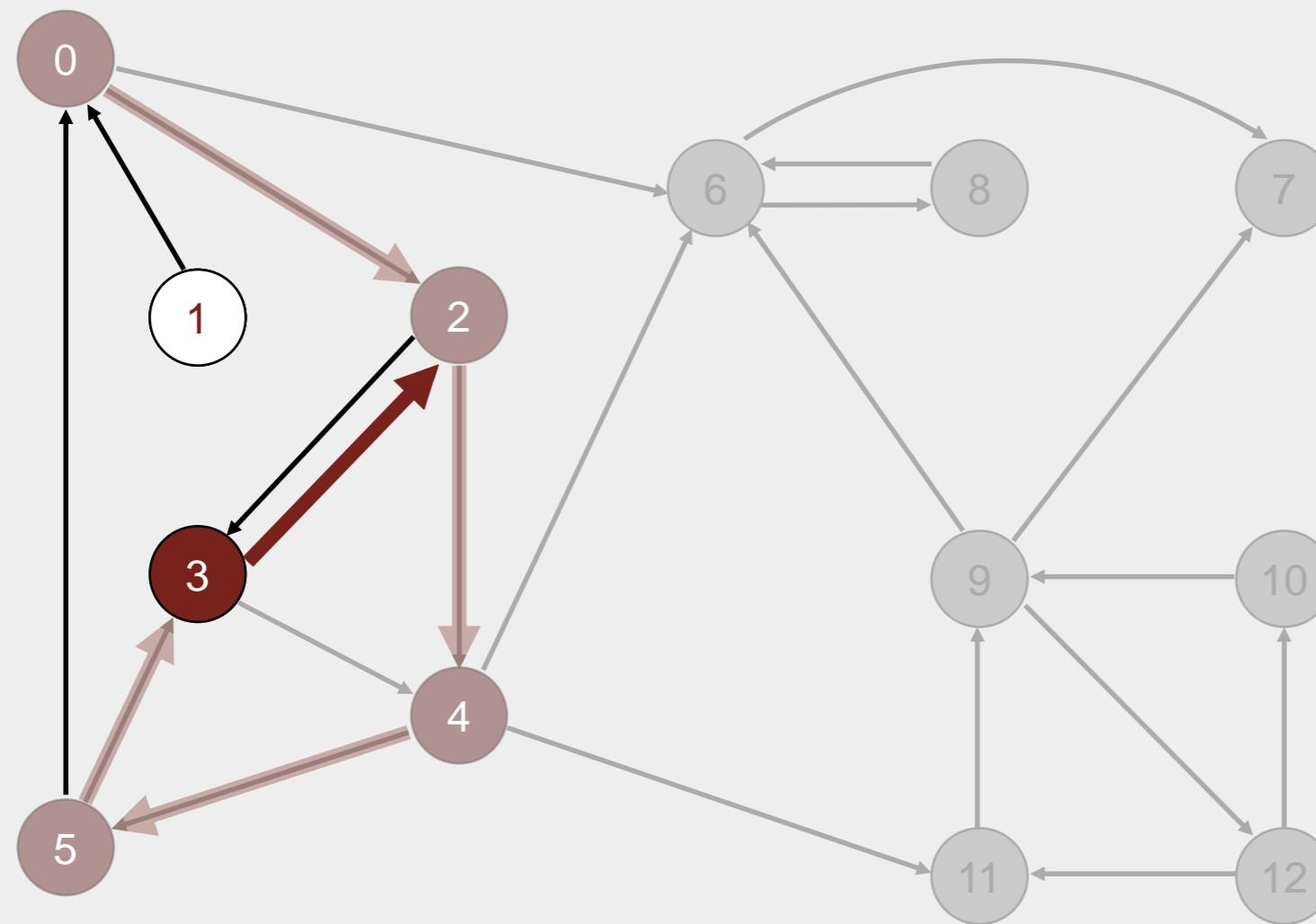


visit 3

v	marked[v]
0	T
1	F
2	T
3	T
4	T
5	T
6	T
7	T
8	T
9	T
10	T
11	T
12	T

Phase 1. Compute reverse postorder in G^R .

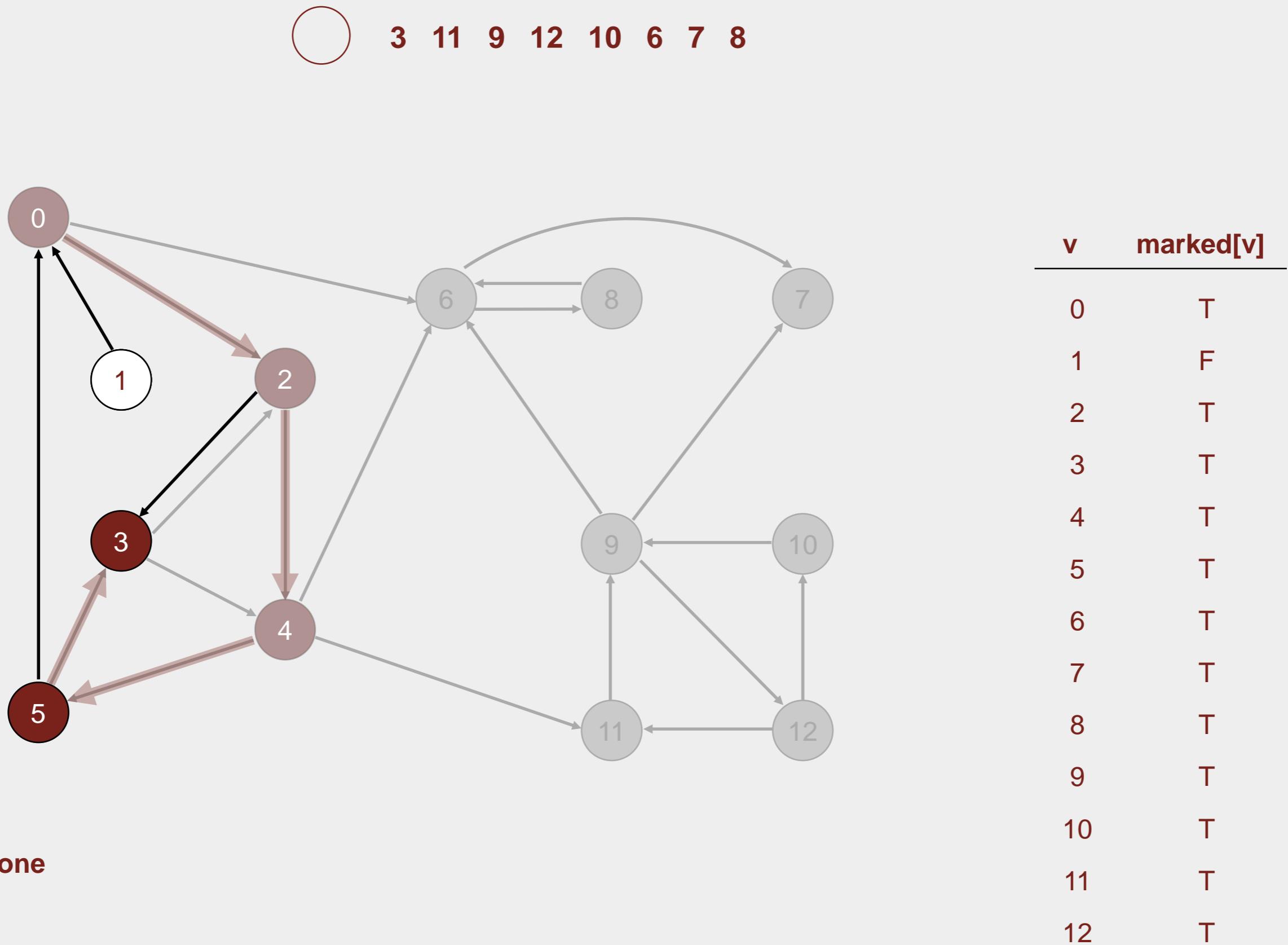
11 9 12 10 6 7 8



visit 3

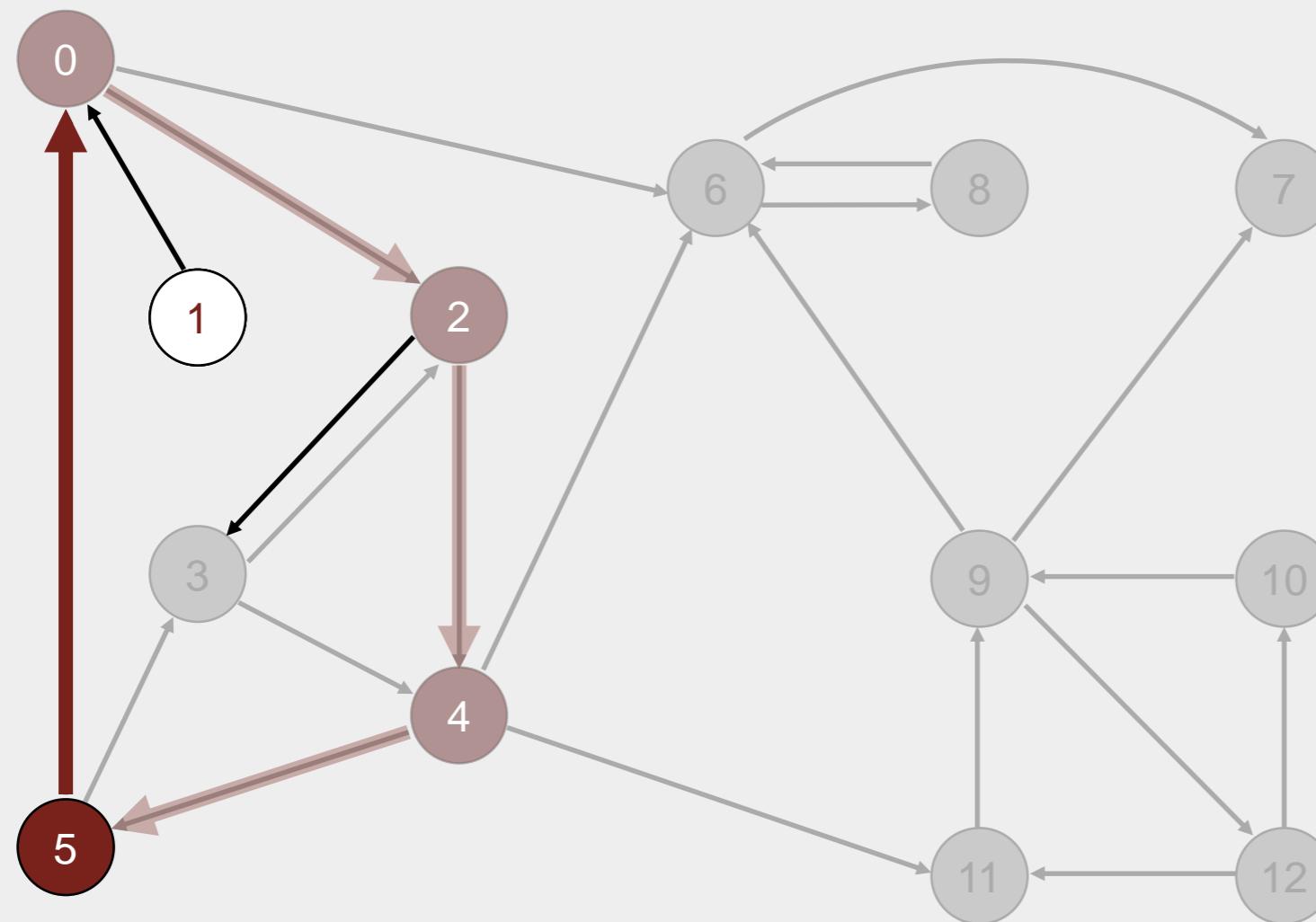
v	marked[v]
0	T
1	F
2	T
3	T
4	T
5	T
6	T
7	T
8	T
9	T
10	T
11	T
12	T

Phase 1. Compute reverse postorder in G^R .



Phase 1. Compute reverse postorder in G^R .

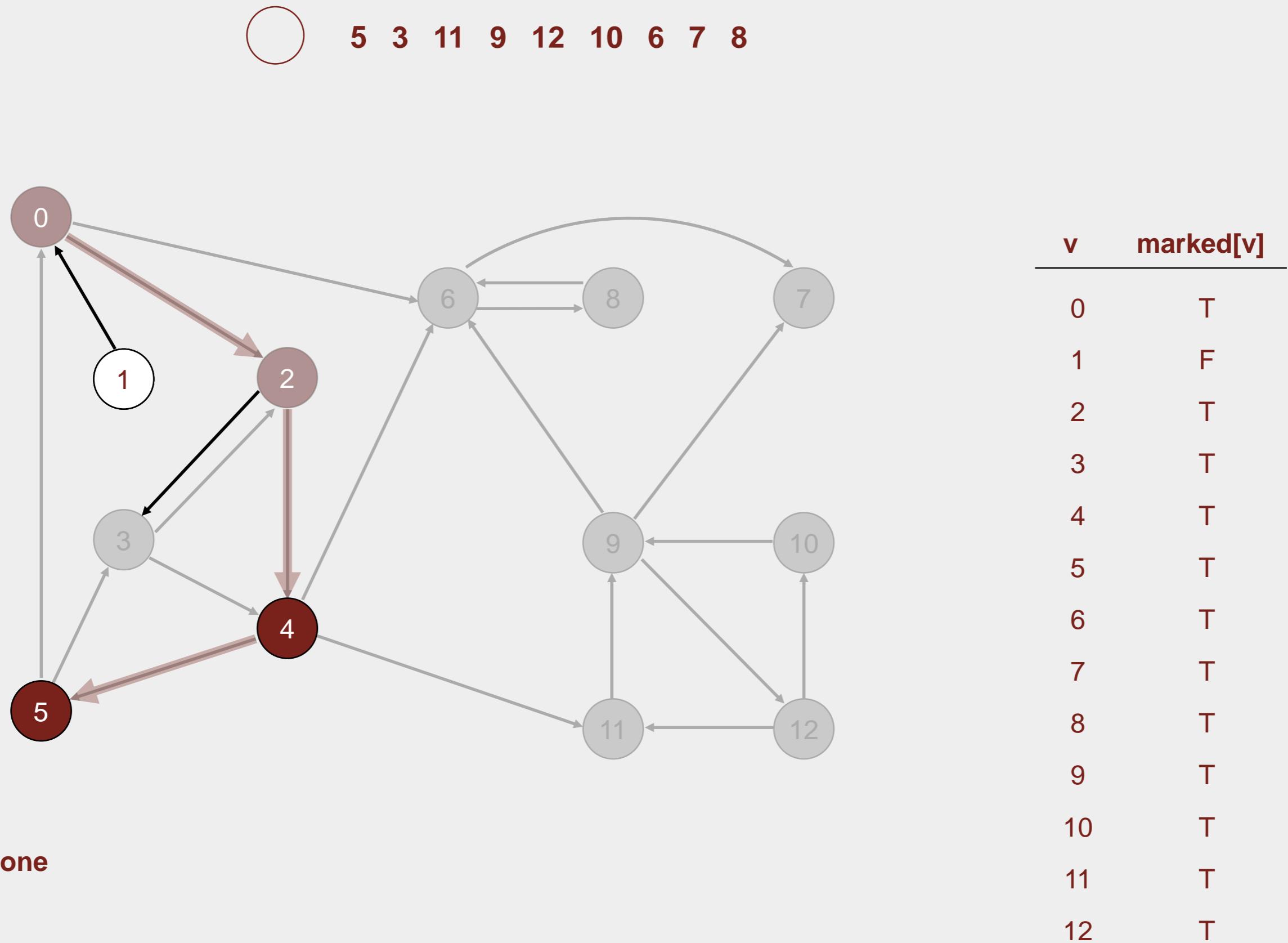
3 11 9 12 10 6 7 8



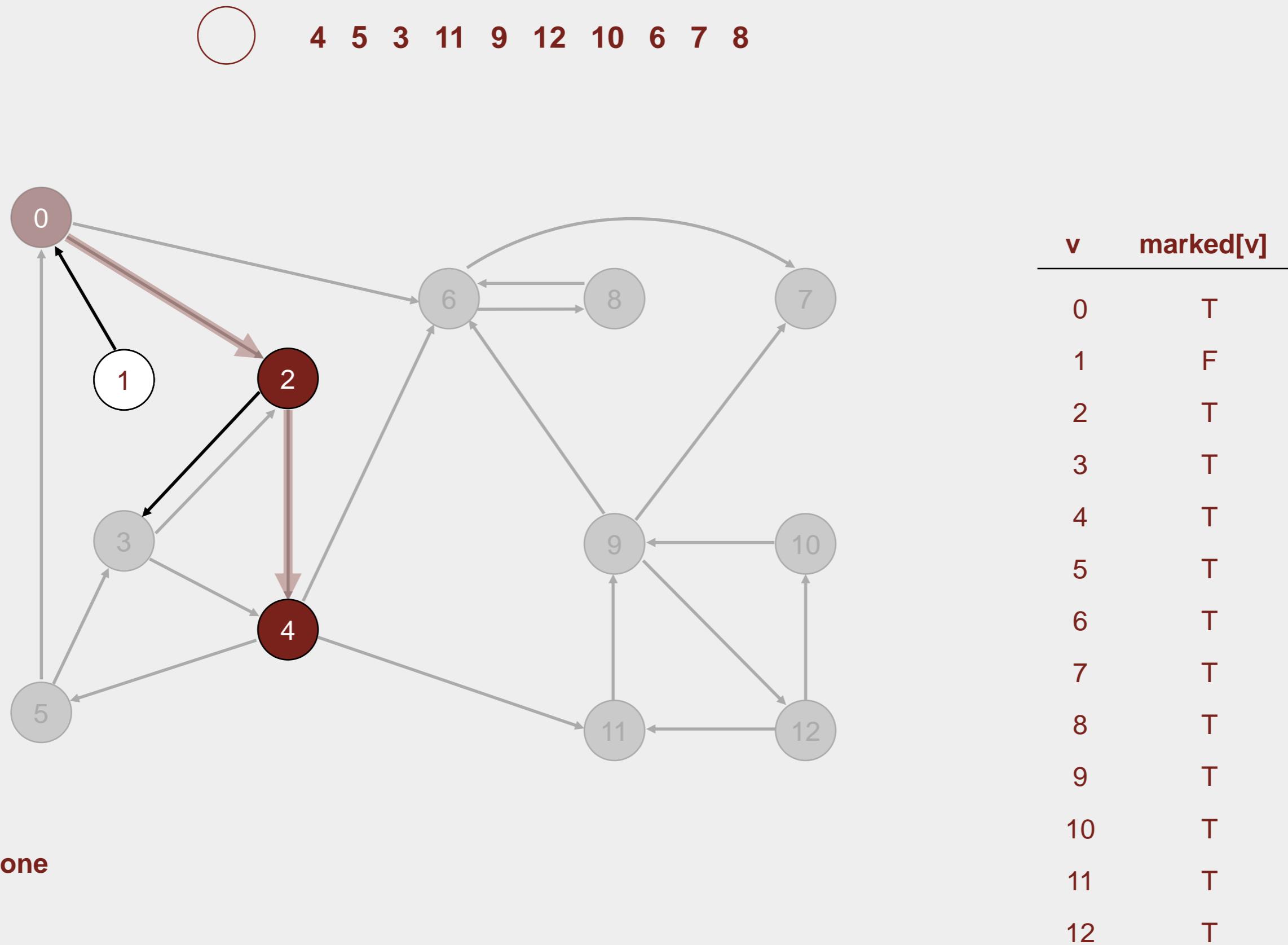
visit 5

v	marked[v]
0	T
1	F
2	T
3	T
4	T
5	T
6	T
7	T
8	T
9	T
10	T
11	T
12	T

Phase 1. Compute reverse postorder in G^R .

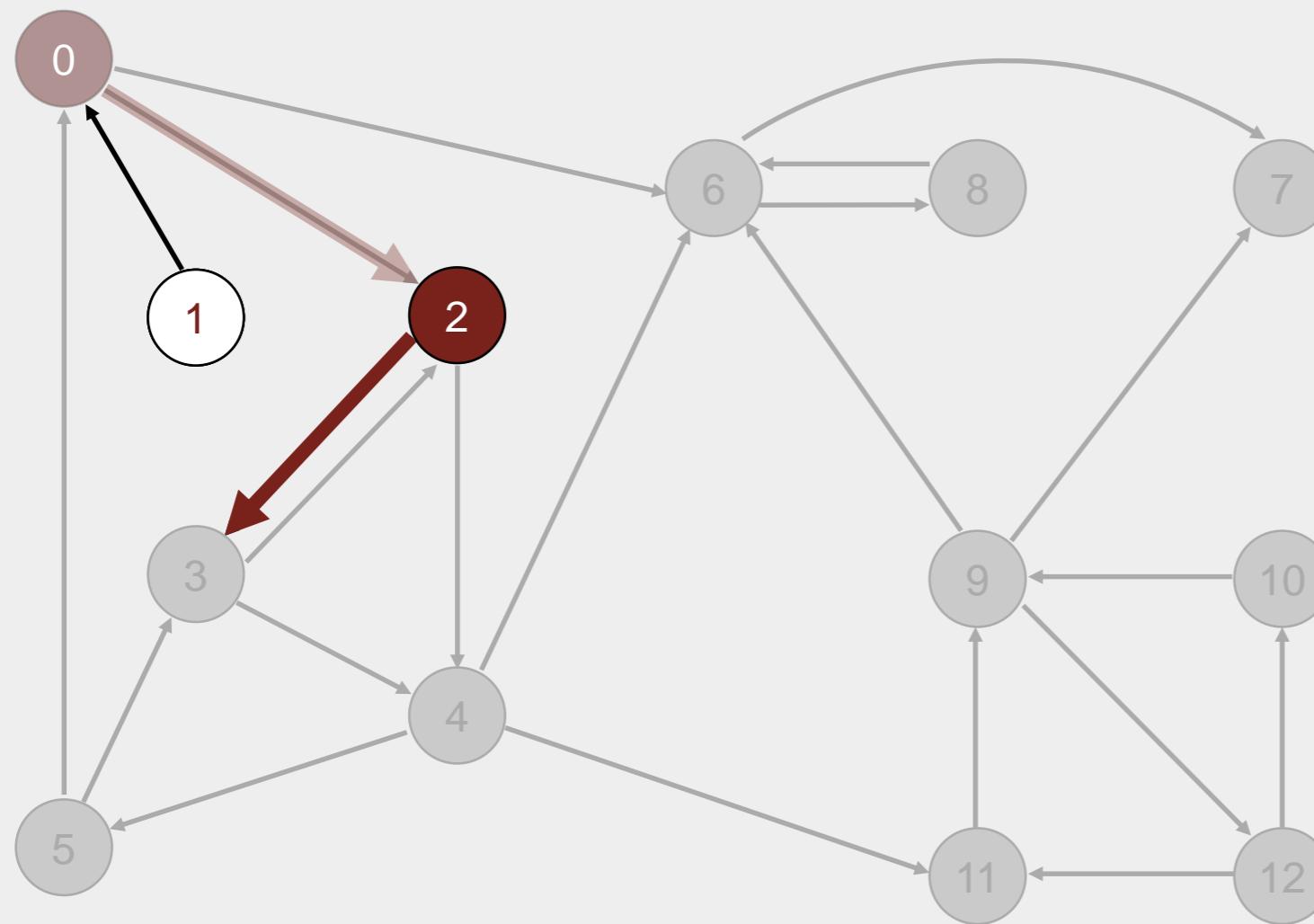


Phase 1. Compute reverse postorder in G^R .



Phase 1. Compute reverse postorder in G^R .

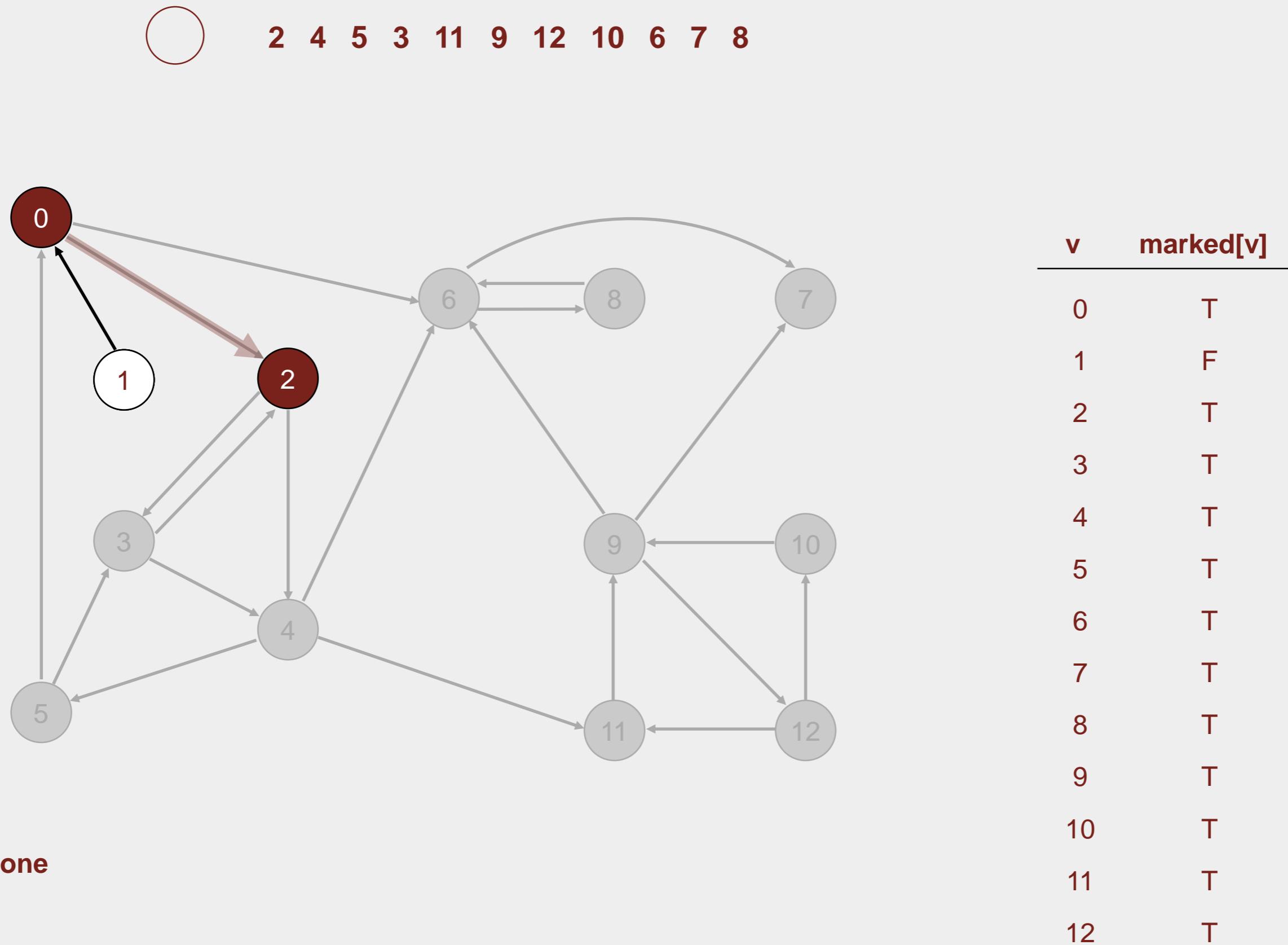
4 5 3 11 9 12 10 6 7 8



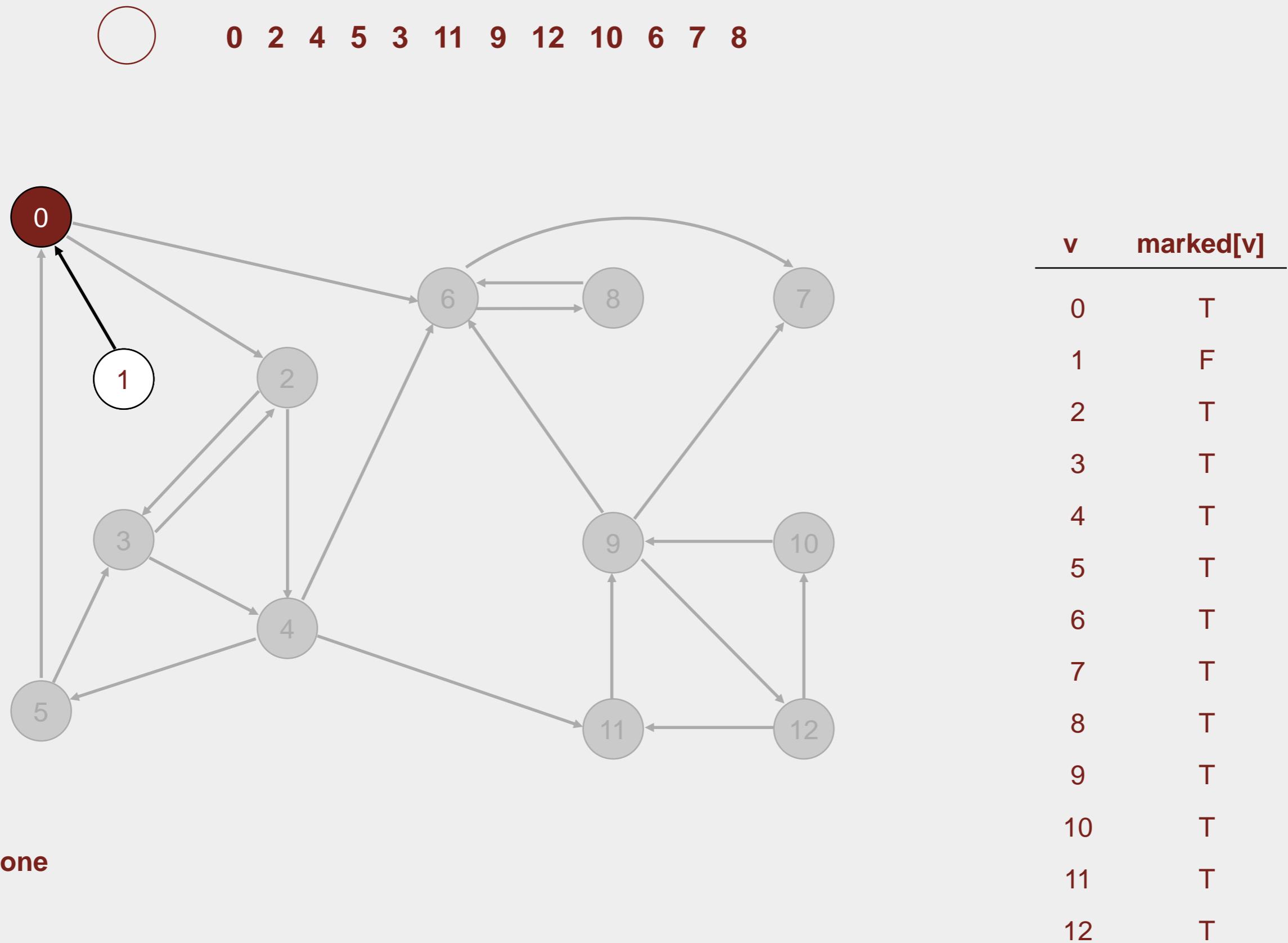
visit 2

v	marked[v]
0	T
1	F
2	T
3	T
4	T
5	T
6	T
7	T
8	T
9	T
10	T
11	T
12	T

Phase 1. Compute reverse postorder in G^R .

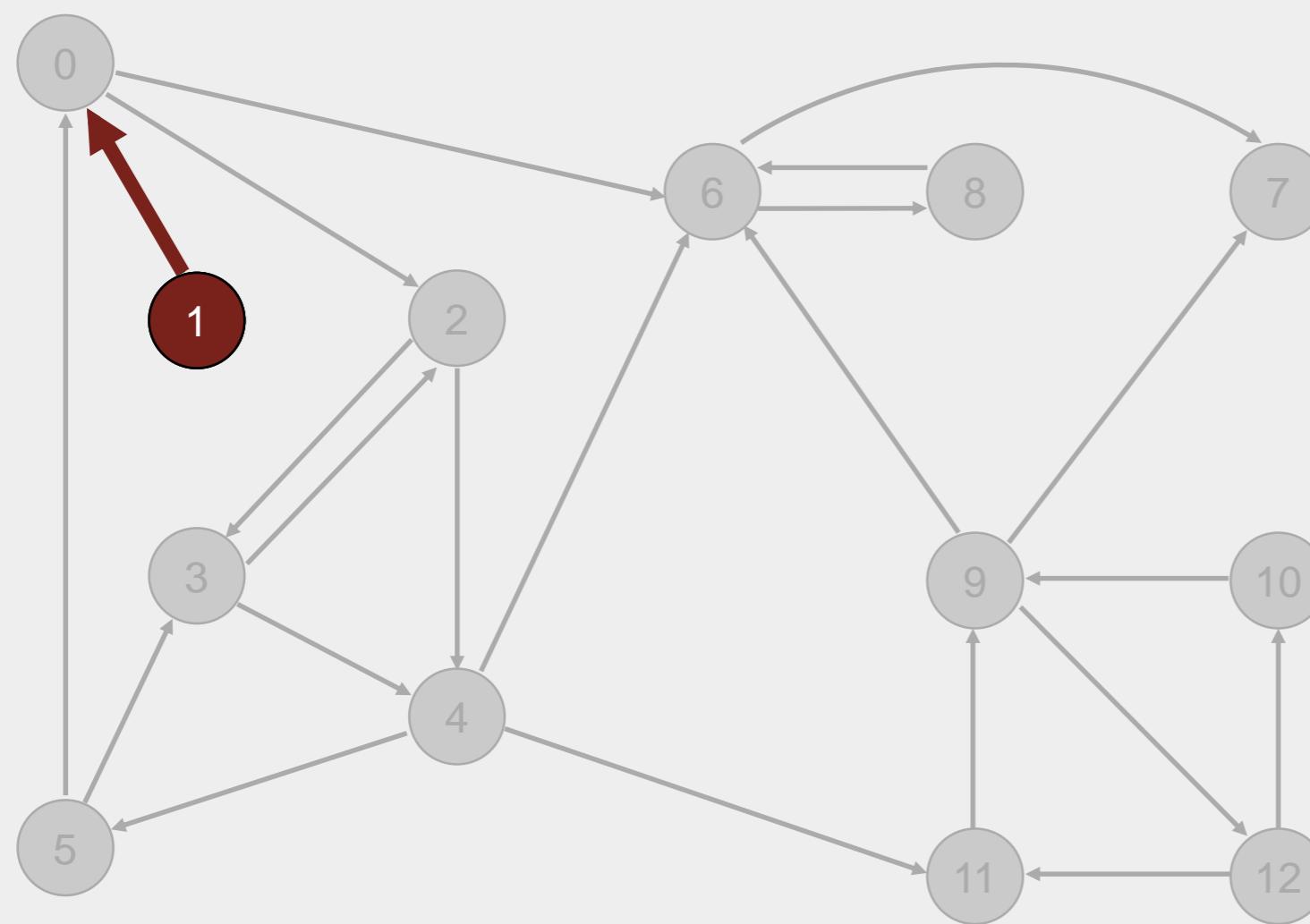


Phase 1. Compute reverse postorder in G^R .



Phase 1. Compute reverse postorder in G^R .

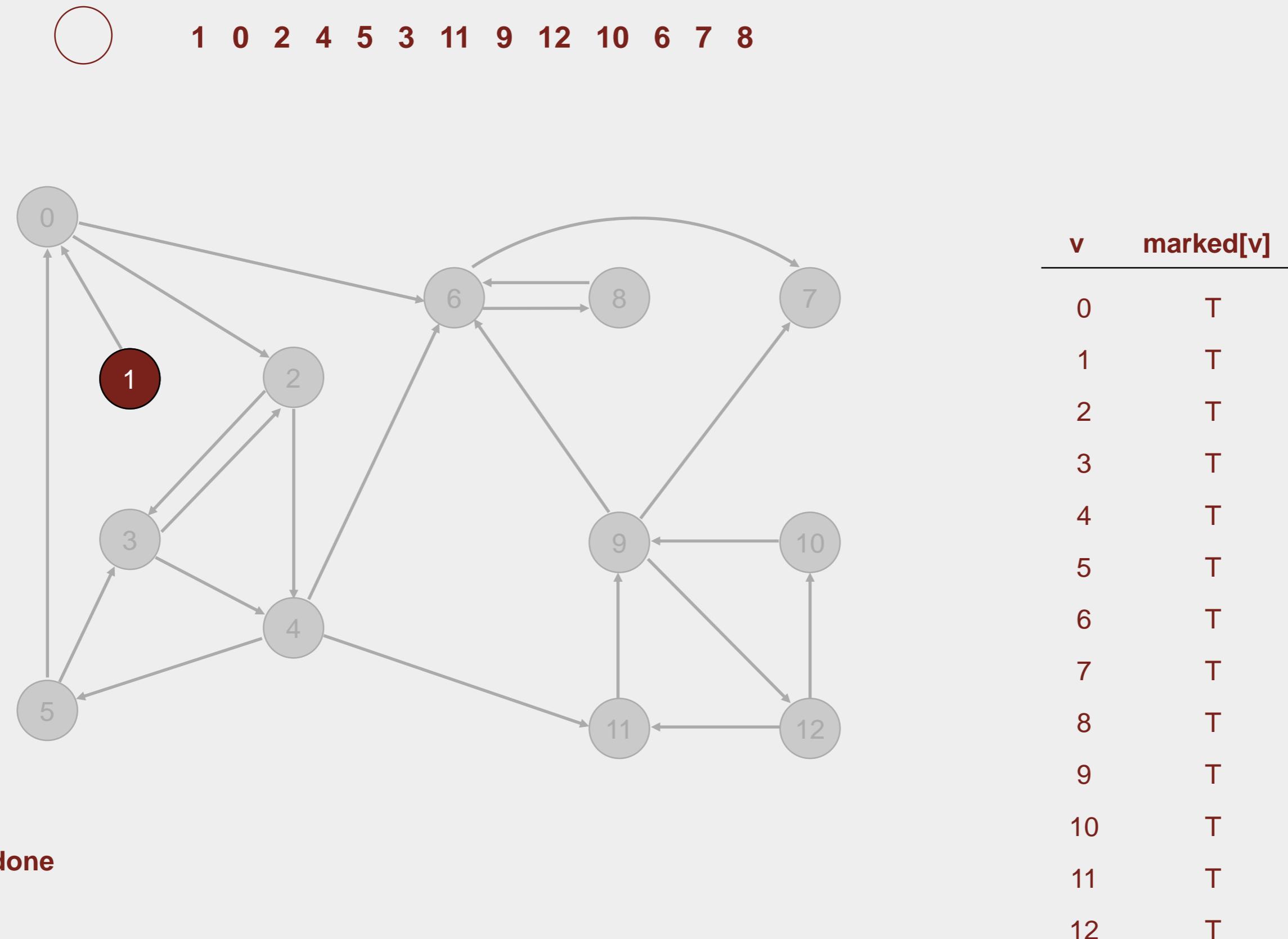
0 2 4 5 3 11 9 12 10 6 7 8



visit 1

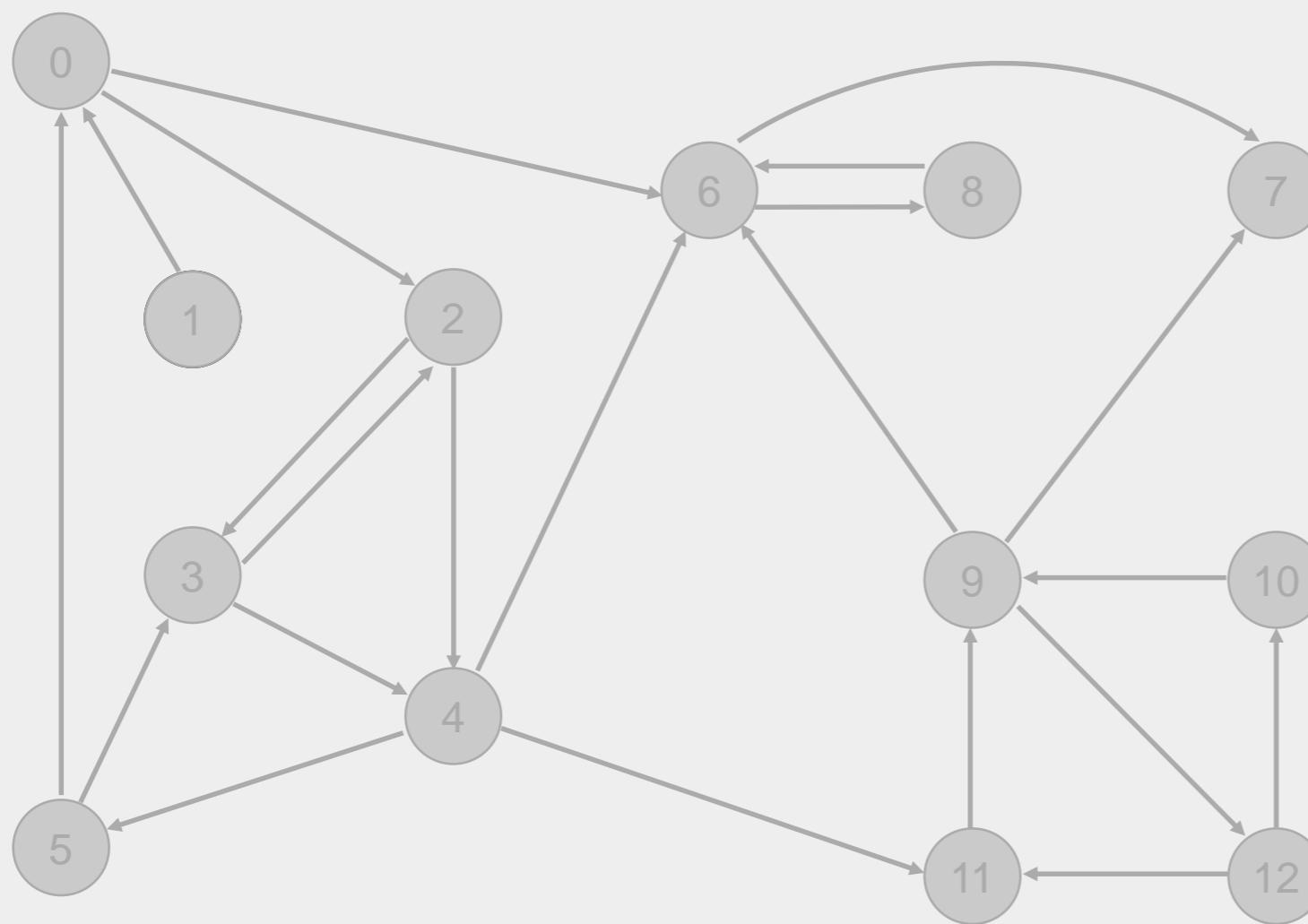
v	marked[v]
0	T
1	T
2	T
3	T
4	T
5	T
6	T
7	T
8	T
9	T
10	T
11	T
12	T

Phase 1. Compute reverse postorder in G^R .



Phase 1. Compute reverse postorder in G^R .

1 0 2 4 5 3 11 9 12 10 6 7 8

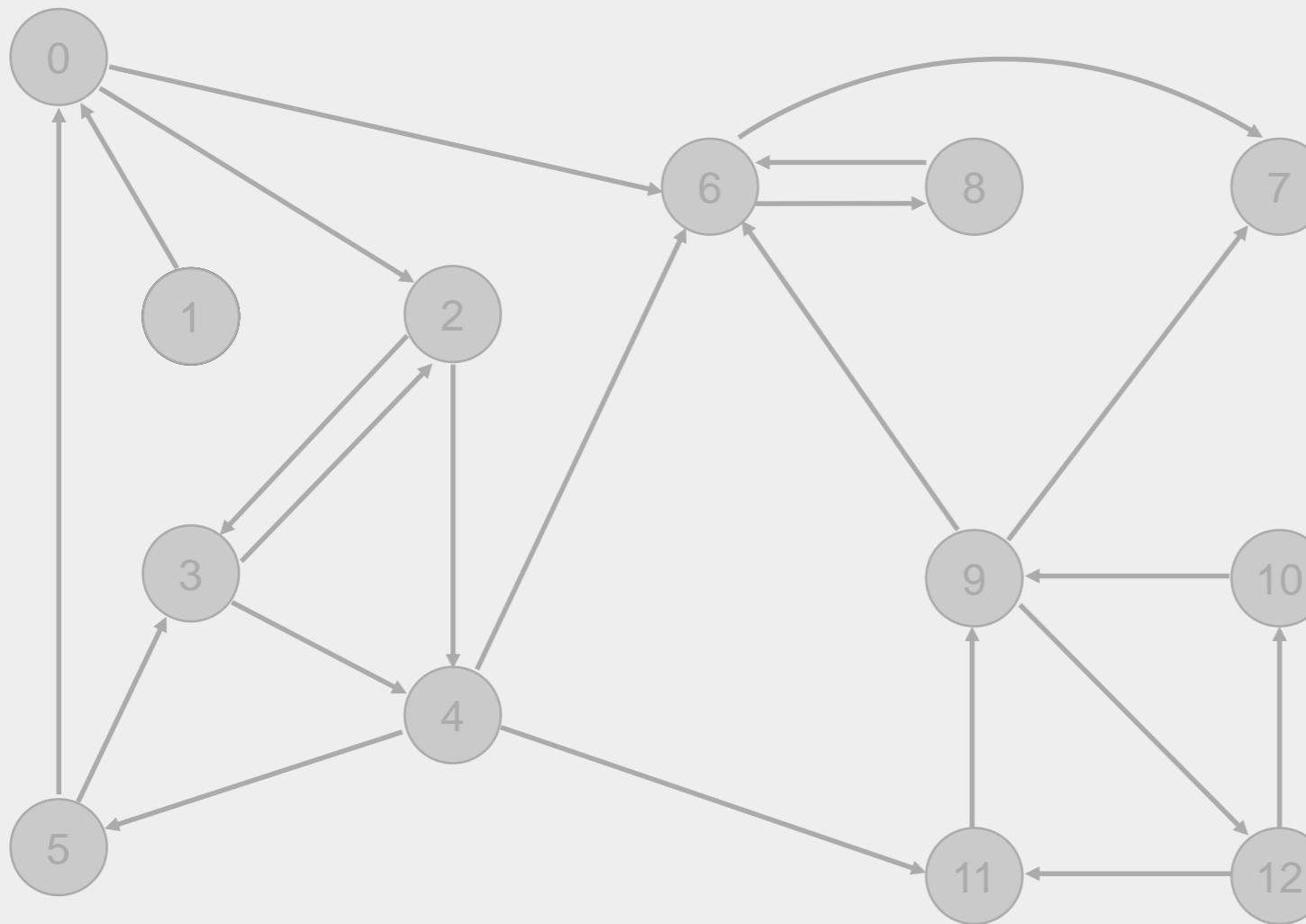


v	marked[v]
0	T
1	T
2	T
3	T
4	T
5	T
6	T
7	T
8	T
9	T
10	T
11	T
12	T

check 2 3 4 5 6 7 8 9 10 11 12

Phase 1. Compute reverse postorder in G^R .

1 0 2 4 5 3 11 9 12 10 6 7 8

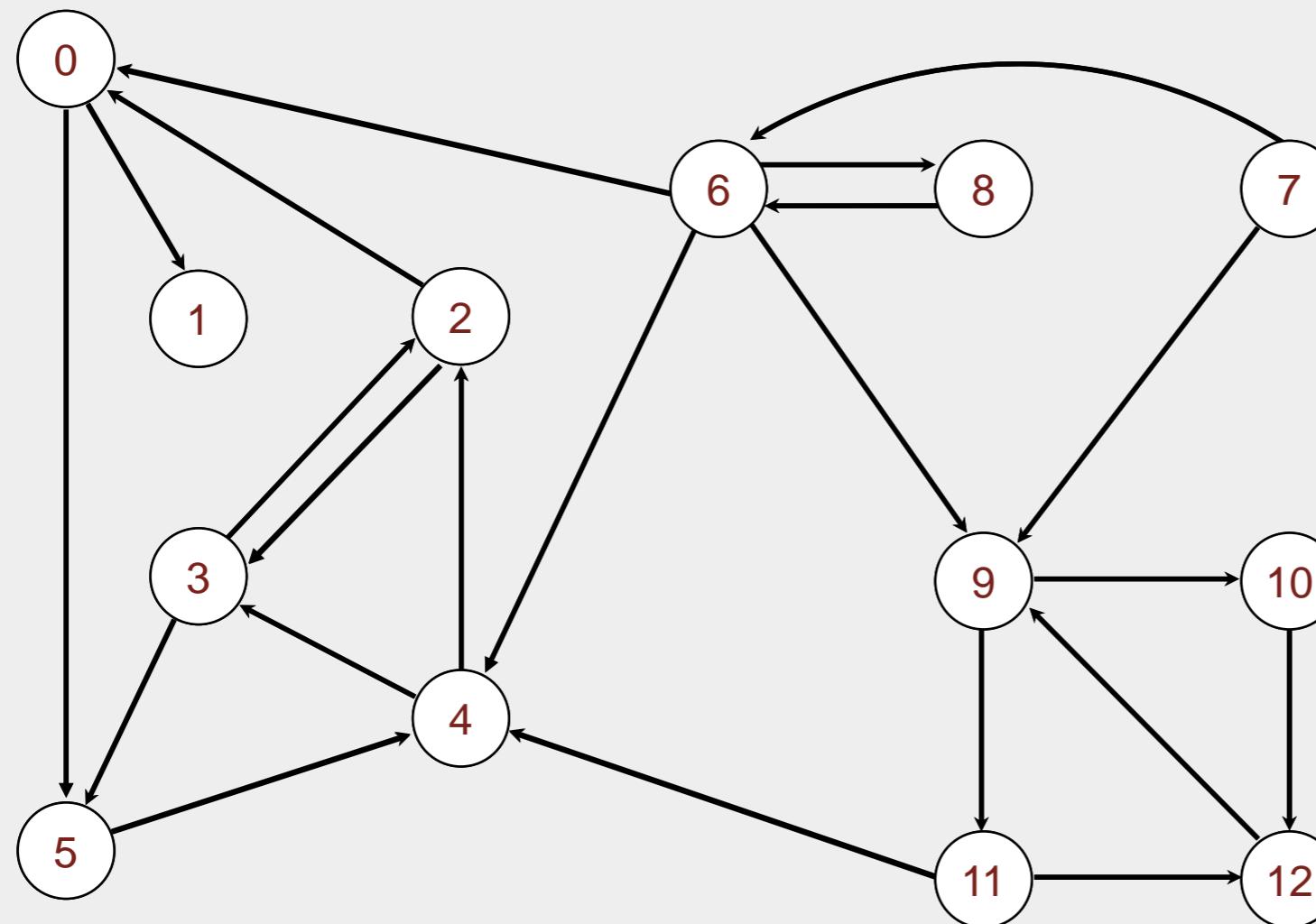


reverse digraph G^R

- ▶ DFS in reverse graph
- ▶ DFS in original graph

Phase 2. Run DFS in G , visiting unmarked vertices in reverse postorder of G^R .

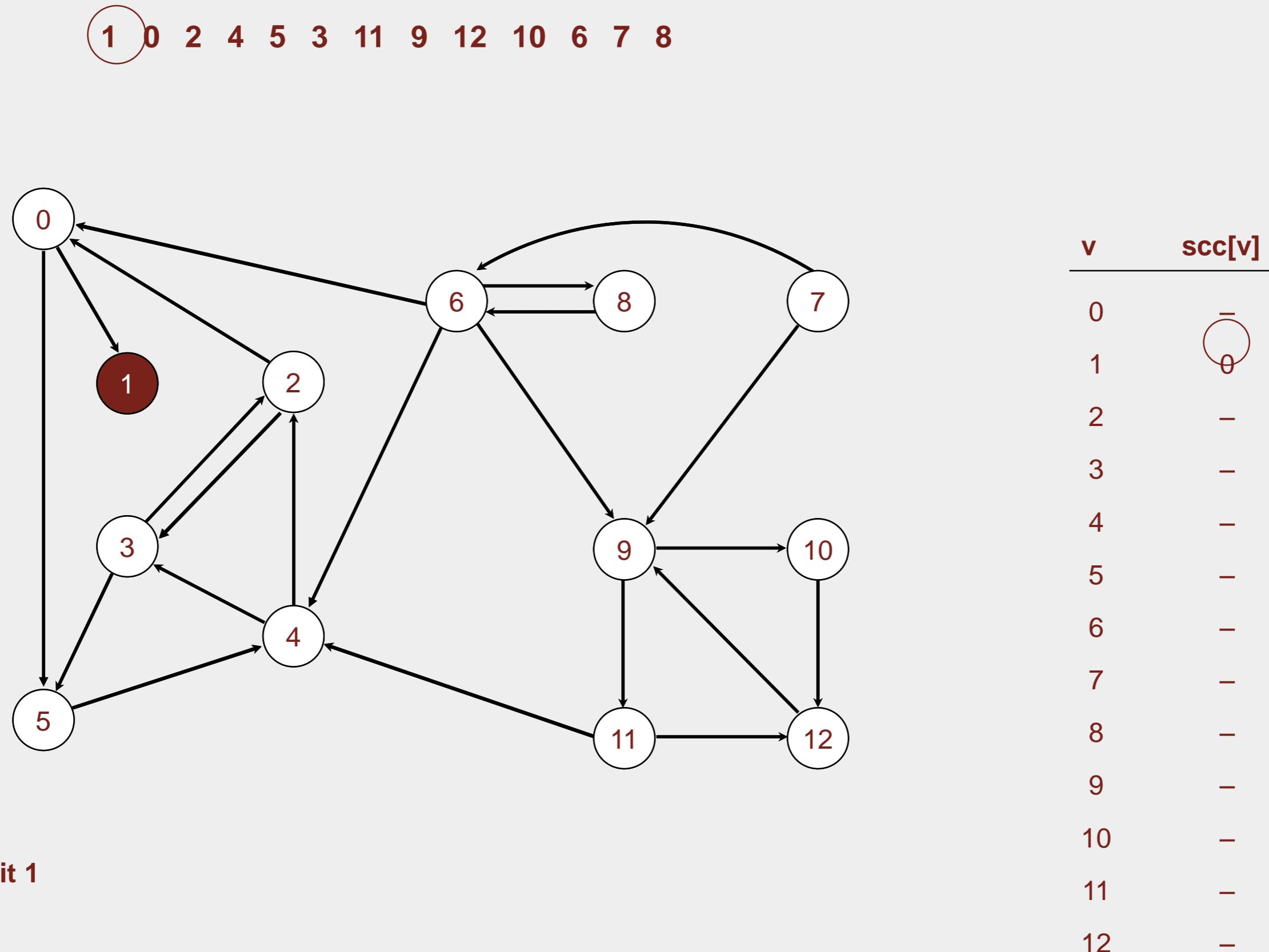
1 0 2 4 5 3 11 9 12 10 6 7 8



original digraph G

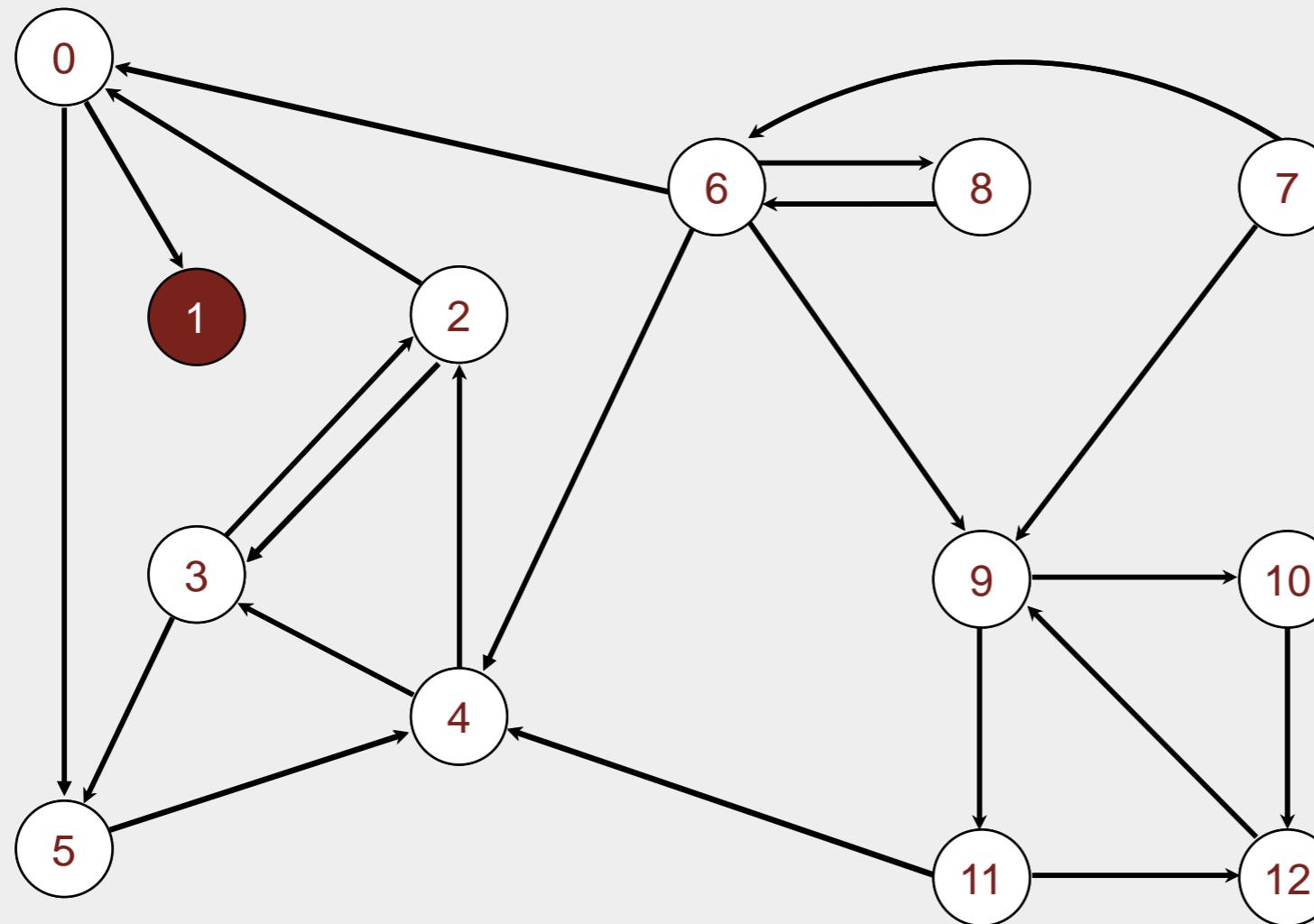
v	$scc[v]$
0	-
1	-
2	-
3	-
4	-
5	-
6	-
7	-
8	-
9	-
10	-
11	-
12	-

Phase 2. Run DFS in G , visiting unmarked vertices in reverse postorder of G^R .



Phase 2. Run DFS in G , visiting unmarked vertices in reverse postorder of G^R .

1 0 2 4 5 3 11 9 12 10 6 7 8



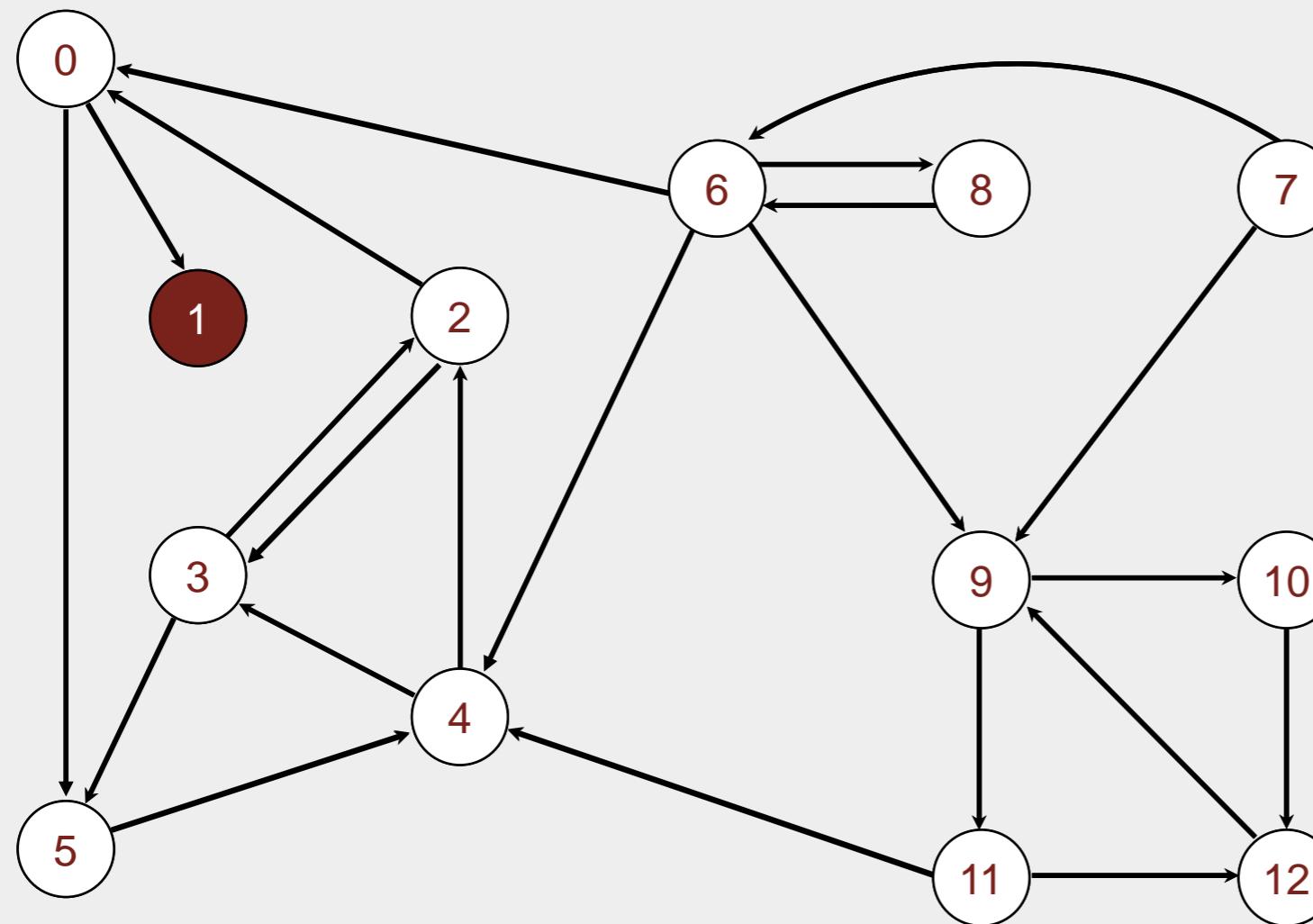
v $scc[v]$

v	$scc[v]$
0	-
1	0
2	-
3	-
4	-
5	-
6	-
7	-
8	-
9	-
10	-
11	-
12	-

1 done

Phase 2. Run DFS in G , visiting unmarked vertices in reverse postorder of G^R .

1 0 2 4 5 3 11 9 12 10 6 7 8



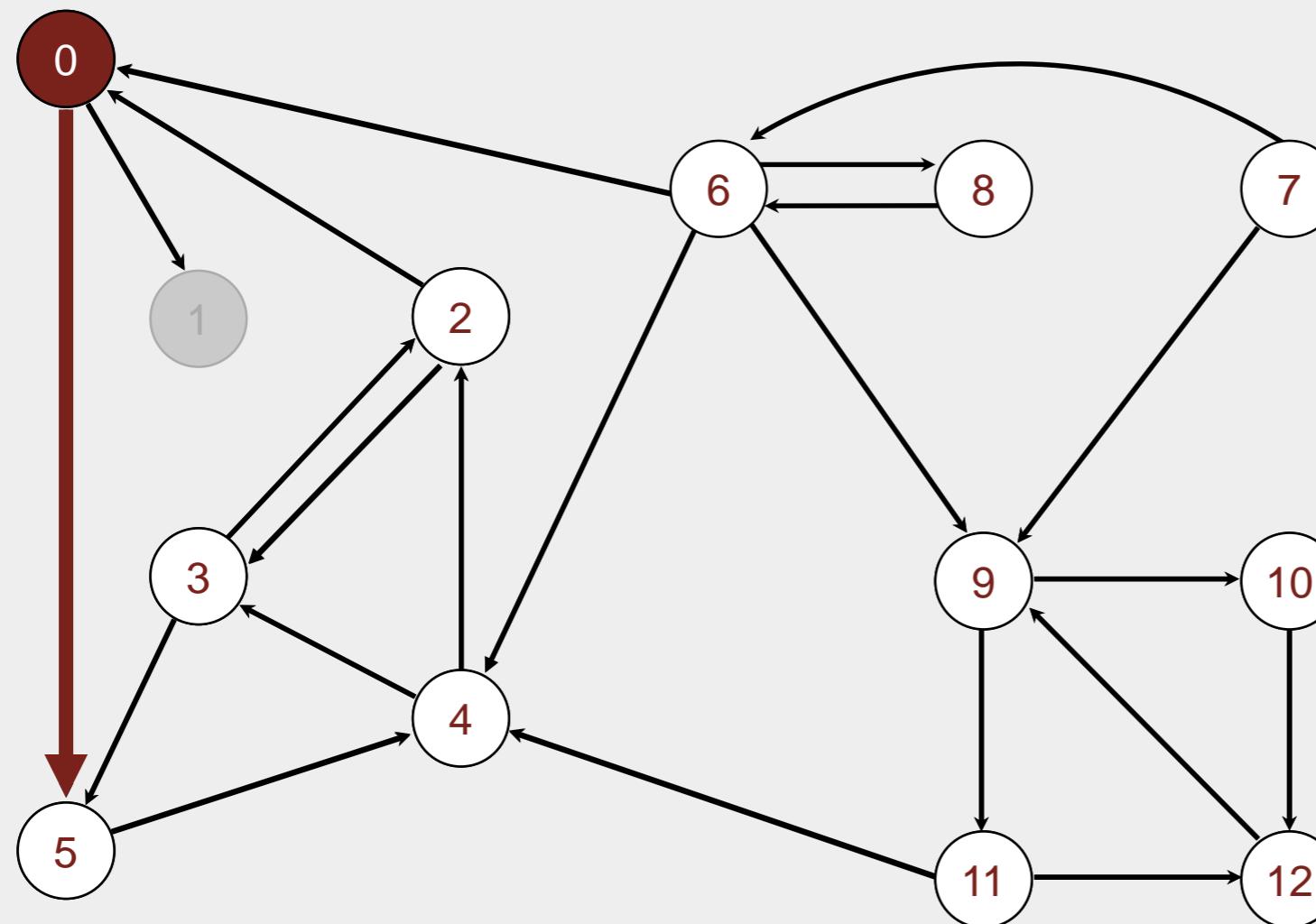
v $scc[v]$

v	$scc[v]$
0	-
1	0
2	-
3	-
4	-
5	-
6	-
7	-
8	-
9	-
10	-
11	-
12	-

strong component: 1

Phase 2. Run DFS in G , visiting unmarked vertices in reverse postorder of G^R .

1 0 2 4 5 3 11 9 12 10 6 7 8

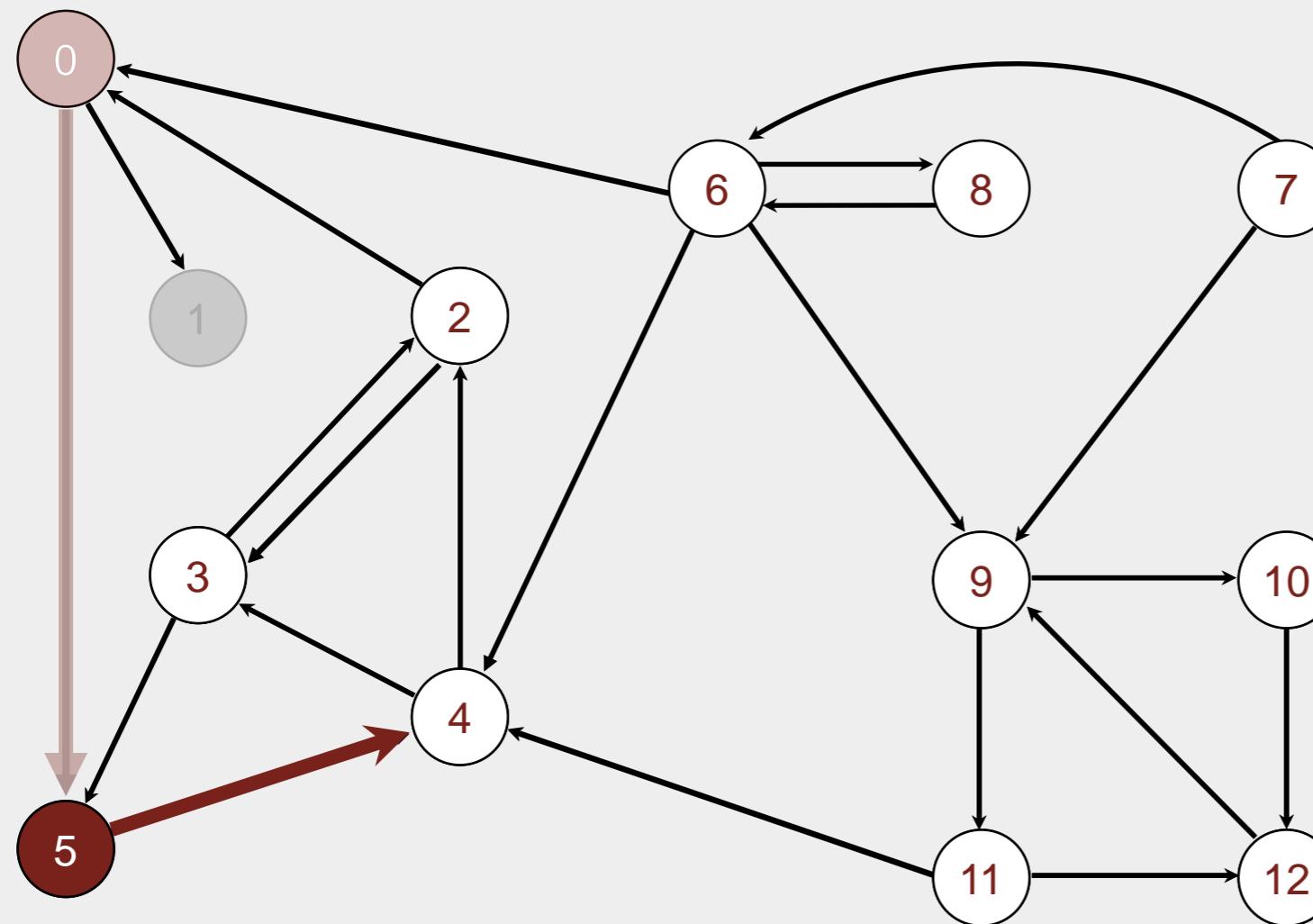


visit 0

v	scc[v]
0	1
1	0
2	-
3	-
4	-
5	-
6	-
7	-
8	-
9	-
10	-
11	-
12	-

Phase 2. Run DFS in G , visiting unmarked vertices in reverse postorder of G^R .

1 0 2 4 5 3 11 9 12 10 6 7 8



v $scc[v]$

0 1

1 0

2 -

3 -

4 -

5 1

6 -

7 -

8 -

9 -

10 -

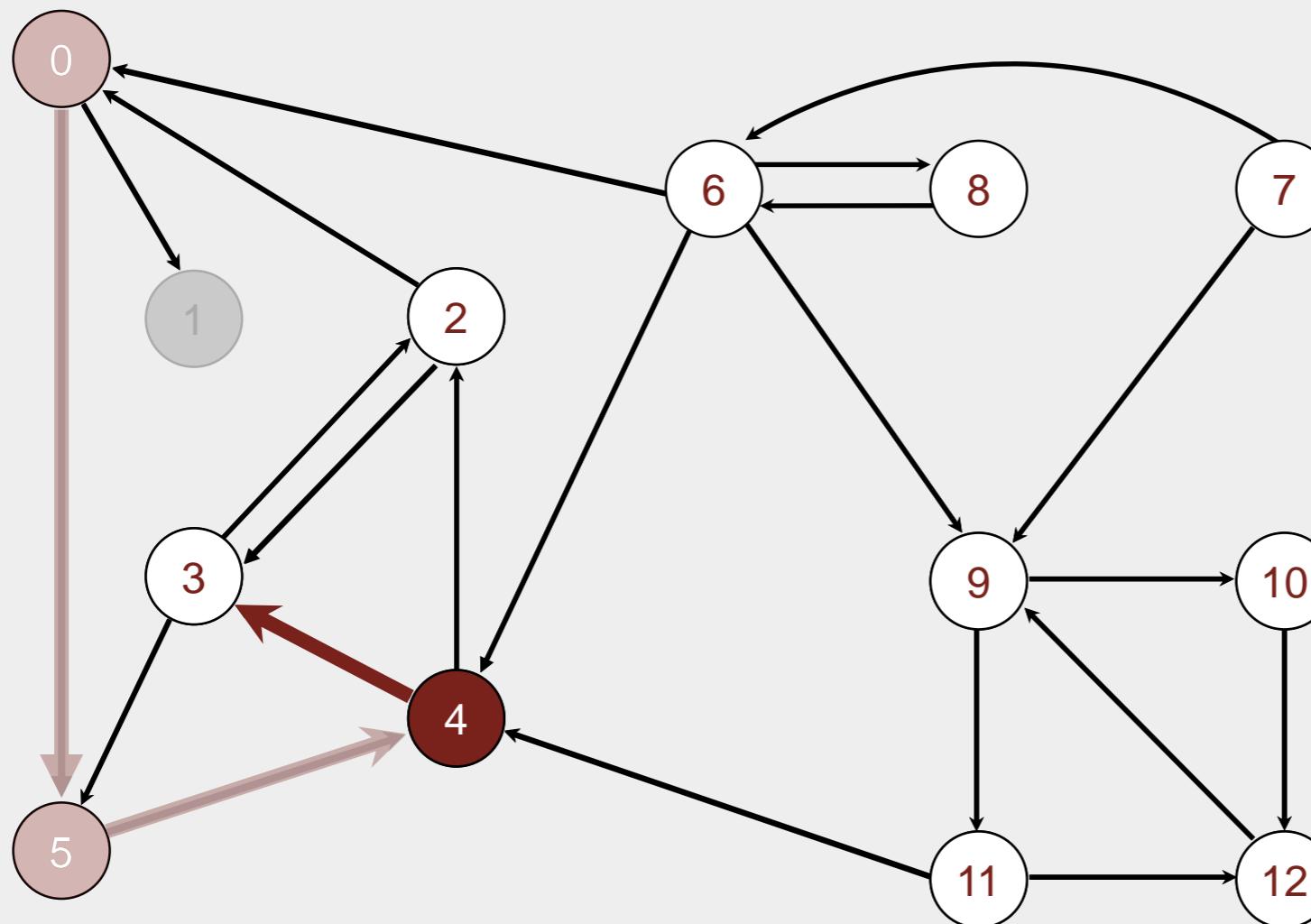
11 -

12 -

visit 5

Phase 2. Run DFS in G , visiting unmarked vertices in reverse postorder of G^R .

1 0 2 4 5 3 11 9 12 10 6 7 8

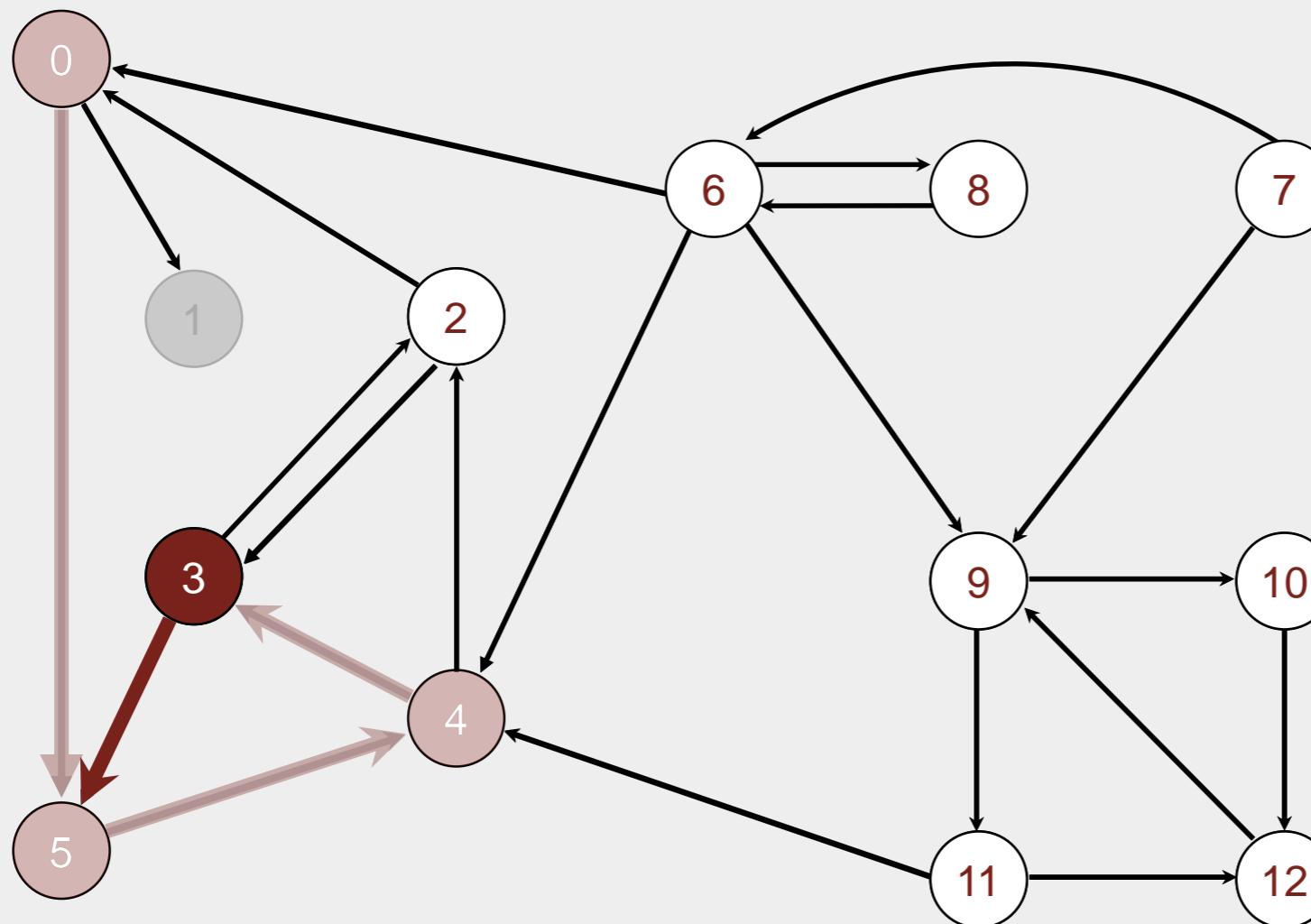


visit 4

v	scc[v]
0	1
1	0
2	-
3	-
4	1
5	1
6	-
7	-
8	-
9	-
10	-
11	-
12	-

Phase 2. Run DFS in G , visiting unmarked vertices in reverse postorder of G^R .

1 0 2 4 5 3 11 9 12 10 6 7 8

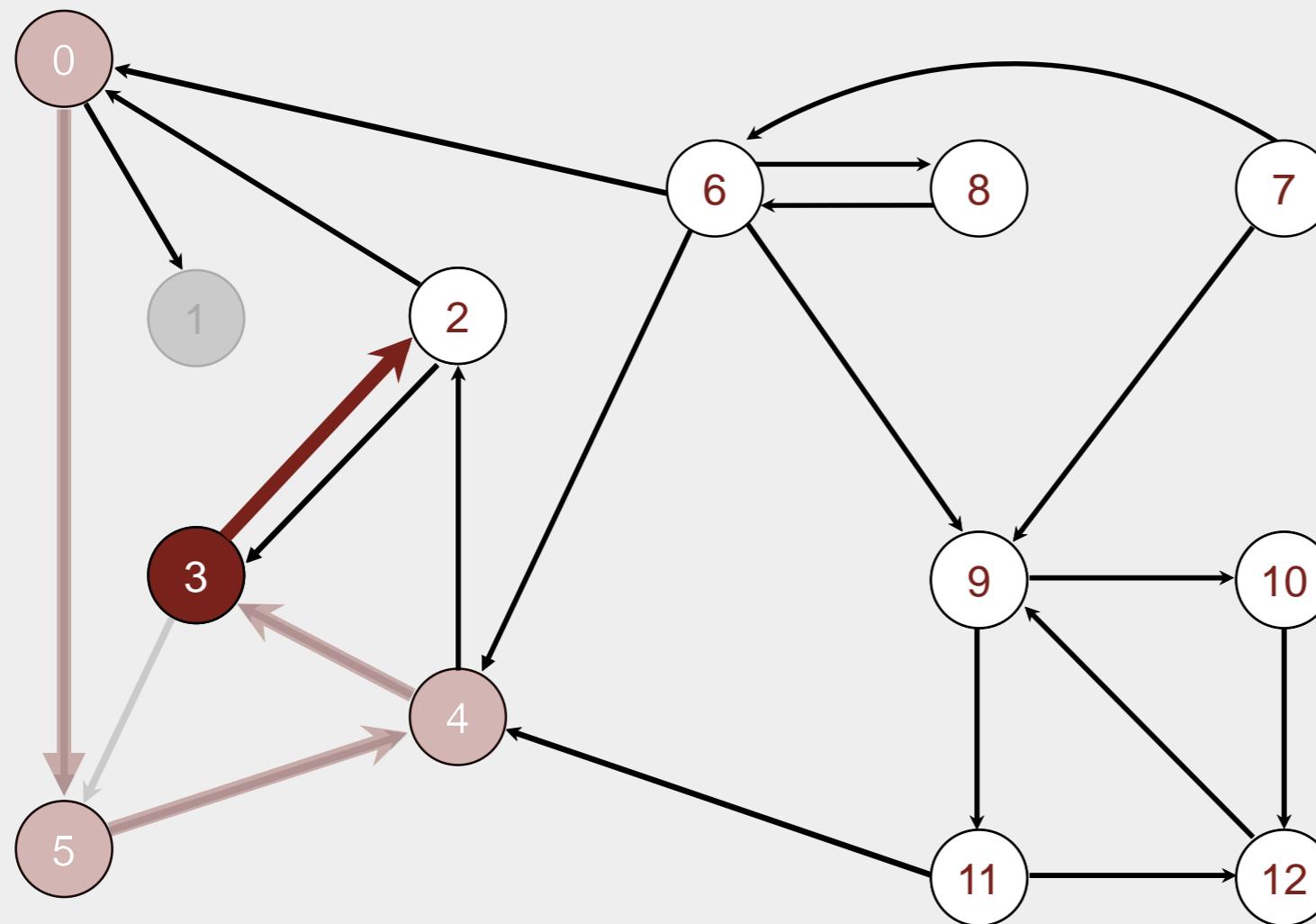


v	scc[v]
0	1
1	0
2	-
3	1
4	1
5	-
6	-
7	-
8	-
9	-
10	-
11	-
12	-

visit 3

Phase 2. Run DFS in G , visiting unmarked vertices in reverse postorder of G^R .

1 0 2 4 5 3 11 9 12 10 6 7 8

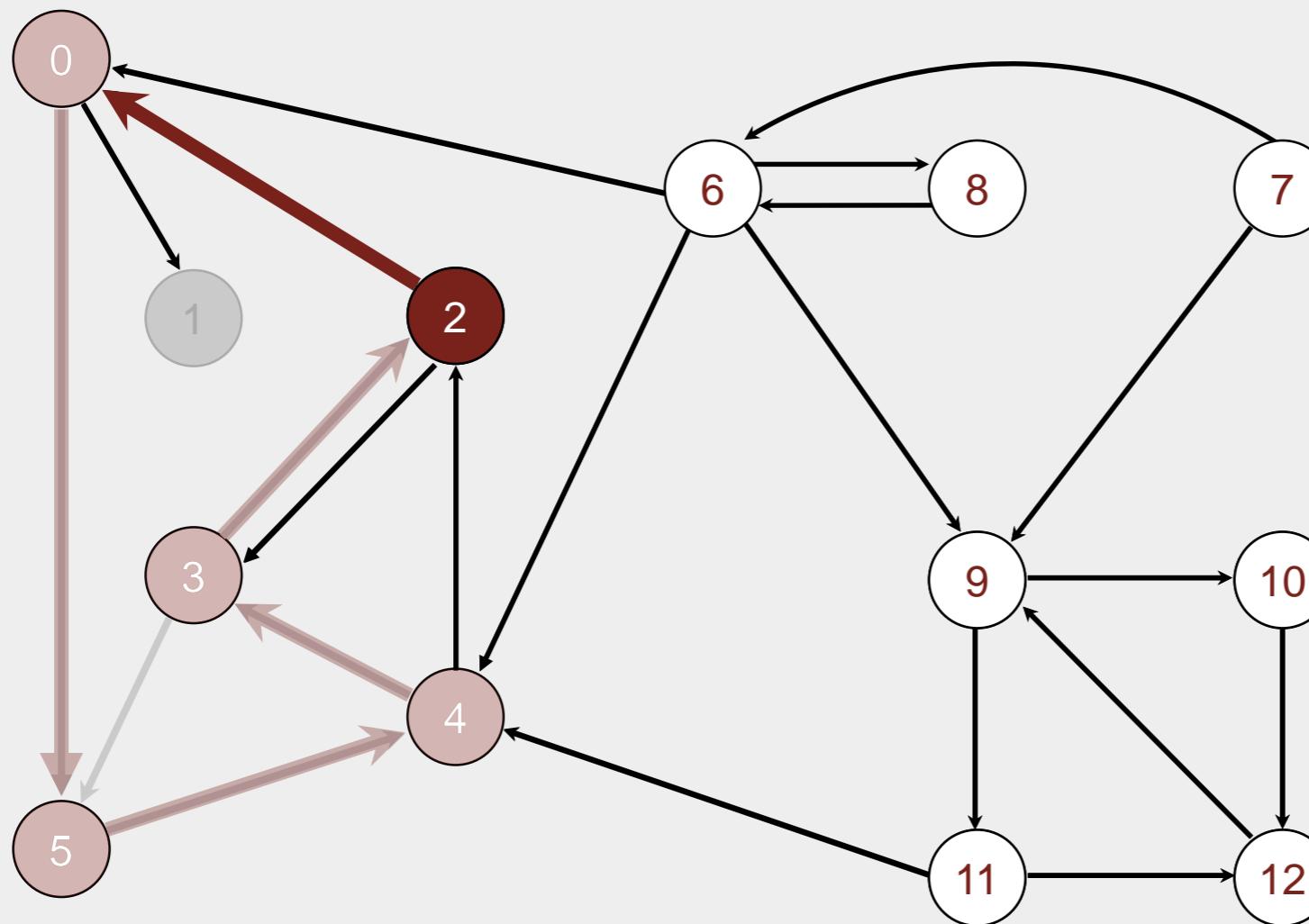


visit 3

v	scc[v]
0	1
1	0
2	-
3	1
4	1
5	1
6	-
7	-
8	-
9	-
10	-
11	-
12	-

Phase 2. Run DFS in G , visiting unmarked vertices in reverse postorder of G^R .

1 0 2 4 5 3 11 9 12 10 6 7 8

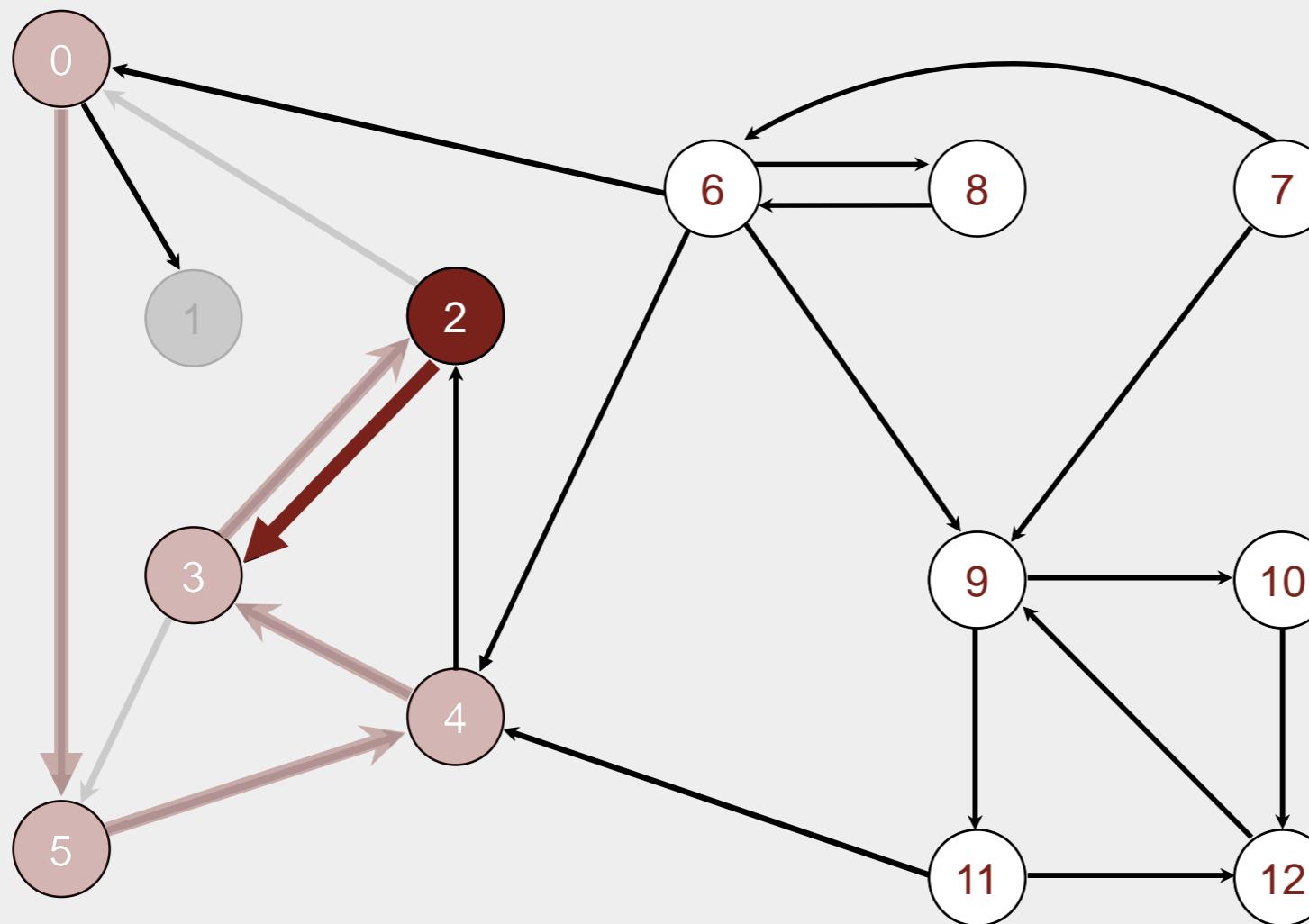


v	scc[v]
0	1
1	0
2	1
3	1
4	1
5	1
6	-
7	-
8	-
9	-
10	-
11	-
12	-

visit 2

Phase 2. Run DFS in G , visiting unmarked vertices in reverse postorder of G^R .

1 0 2 4 5 3 11 9 12 10 6 7 8



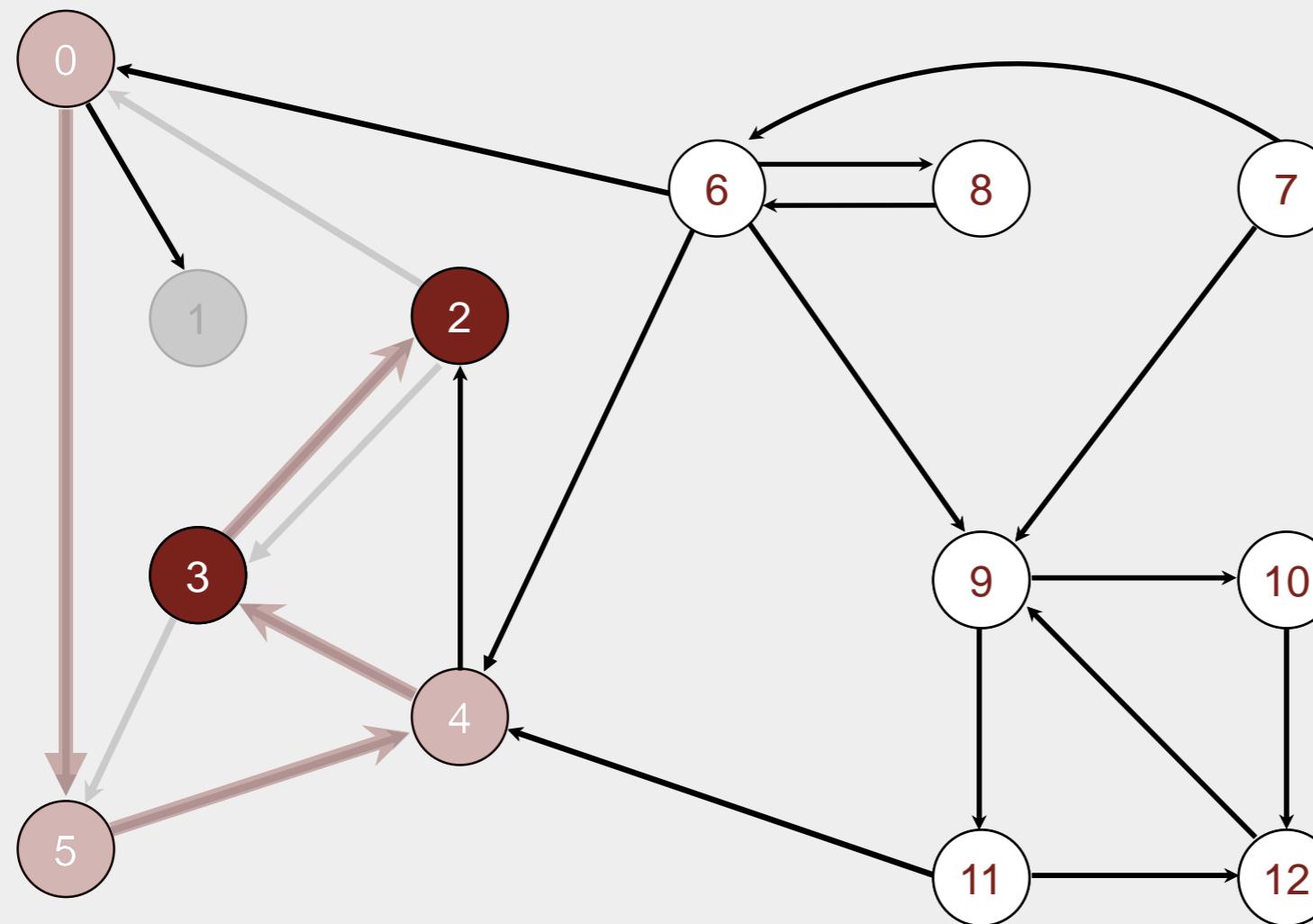
v $scc[v]$

v	$scc[v]$
0	1
1	0
2	1
3	1
4	1
5	1
6	-
7	-
8	-
9	-
10	-
11	-
12	-

visit 2

Phase 2. Run DFS in G , visiting unmarked vertices in reverse postorder of G^R .

1 0 2 4 5 3 11 9 12 10 6 7 8



v $scc[v]$

0 1

1 0

2 1

3 1

4 1

5 1

6 -

7 -

8 -

9 -

10 -

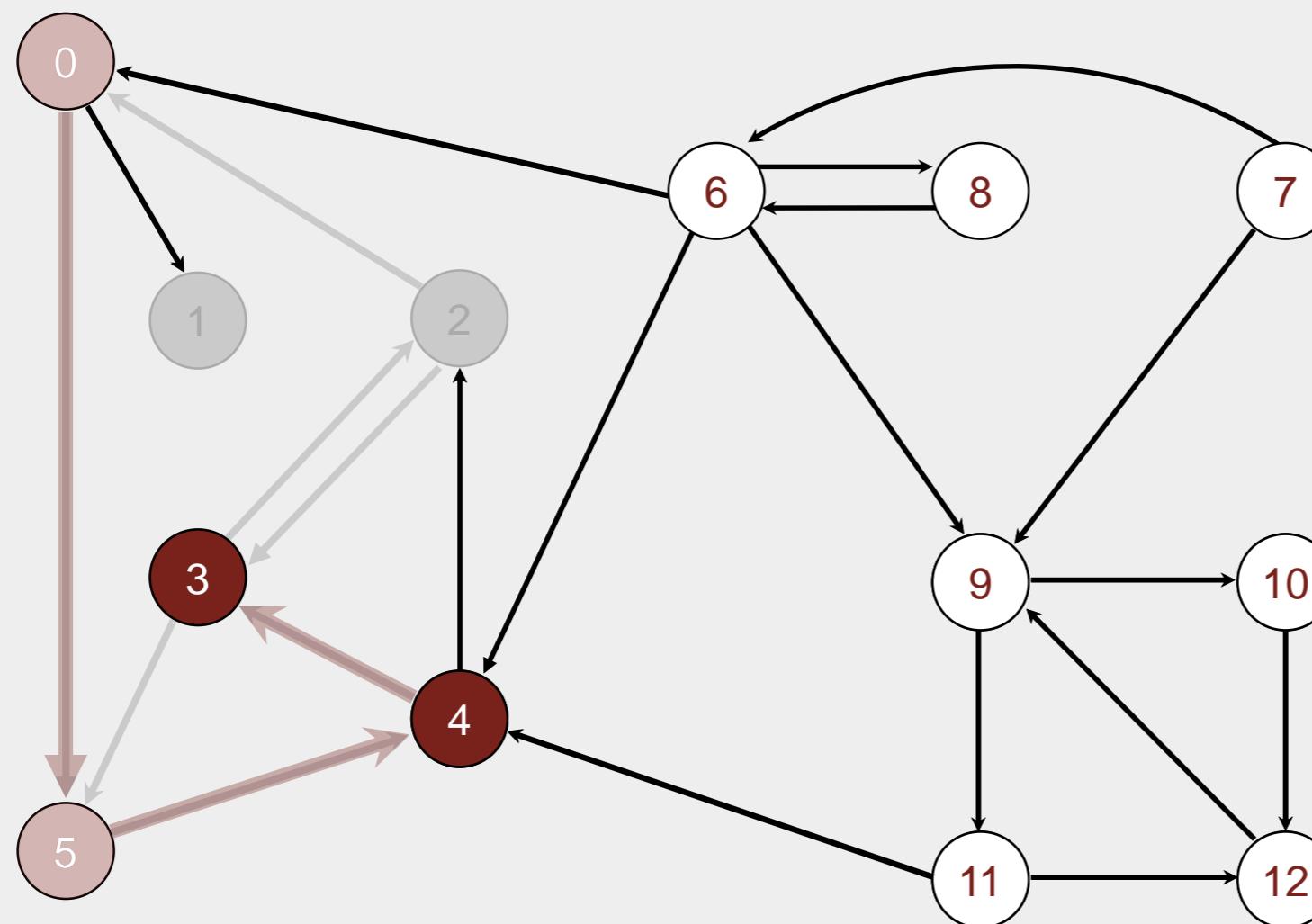
11 -

12 -

2 done

Phase 2. Run DFS in G , visiting unmarked vertices in reverse postorder of G^R .

1 0 2 4 5 3 11 9 12 10 6 7 8



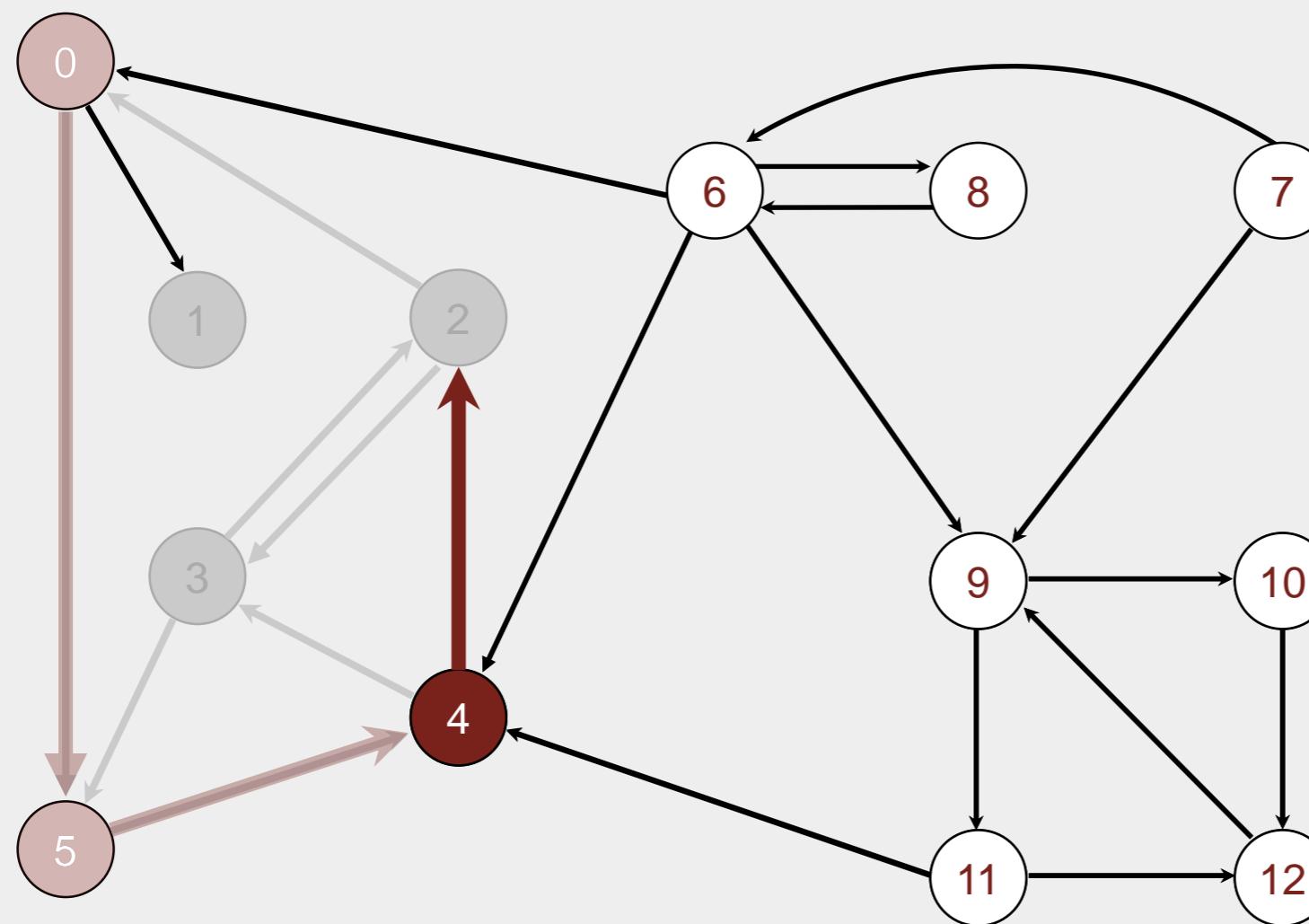
v $scc[v]$

v	$scc[v]$
0	1
1	0
2	1
3	1
4	1
5	1
6	-
7	-
8	-
9	-
10	-
11	-
12	-

3 done

Phase 2. Run DFS in G , visiting unmarked vertices in reverse postorder of G^R .

1 0 2 4 5 3 11 9 12 10 6 7 8

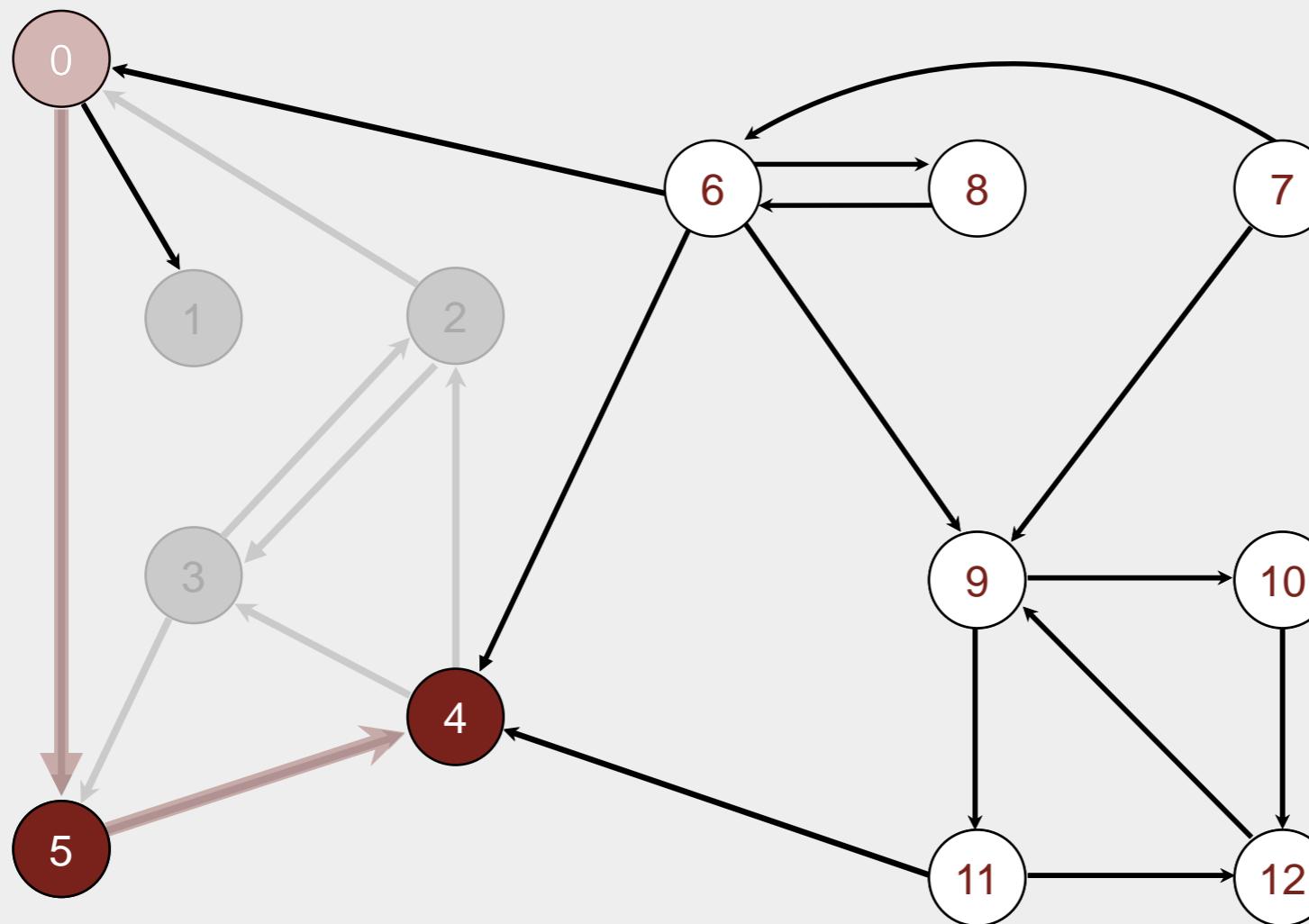


visit 4

v	scc[v]
0	1
1	0
2	1
3	1
4	1
5	1
6	-
7	-
8	-
9	-
10	-
11	-
12	-

Phase 2. Run DFS in G , visiting unmarked vertices in reverse postorder of G^R .

1 0 2 4 5 3 11 9 12 10 6 7 8



v $scc[v]$

0 1

1 0

2 1

3 1

4 1

5 1

6 -

7 -

8 -

9 -

10 -

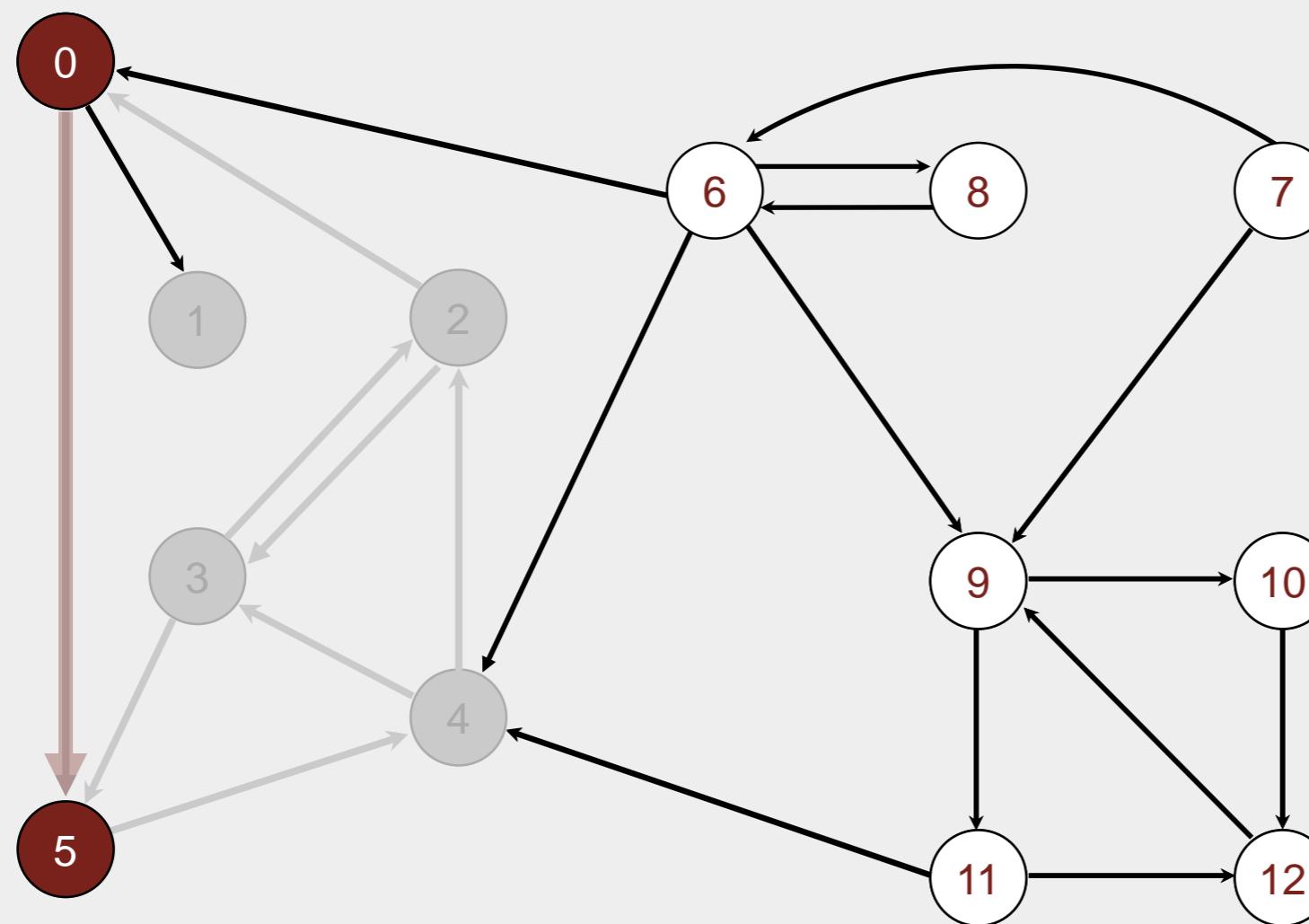
11 -

12 -

4 done

Phase 2. Run DFS in G , visiting unmarked vertices in reverse postorder of G^R .

1 0 2 4 5 3 11 9 12 10 6 7 8



v $scc[v]$

0 1

1 0

2 1

3 1

4 1

5 1

6 -

7 -

8 -

9 -

10 -

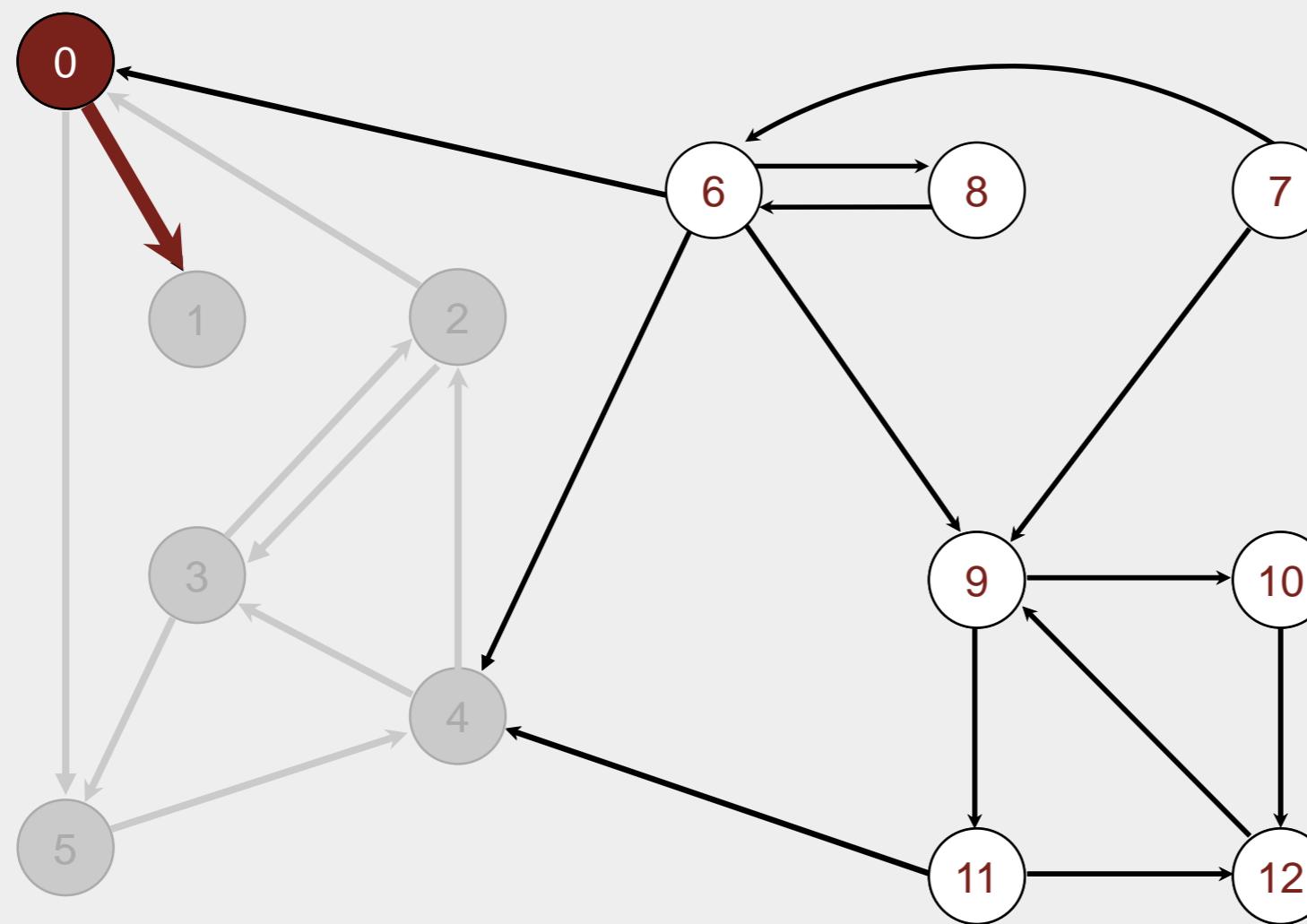
11 -

12 -

5 done

Phase 2. Run DFS in G , visiting unmarked vertices in reverse postorder of G^R .

1 0 2 4 5 3 11 9 12 10 6 7 8



v $scc[v]$

0 1

1 0

2 1

3 1

4 1

5 1

6 -

7 -

8 -

9 -

10 -

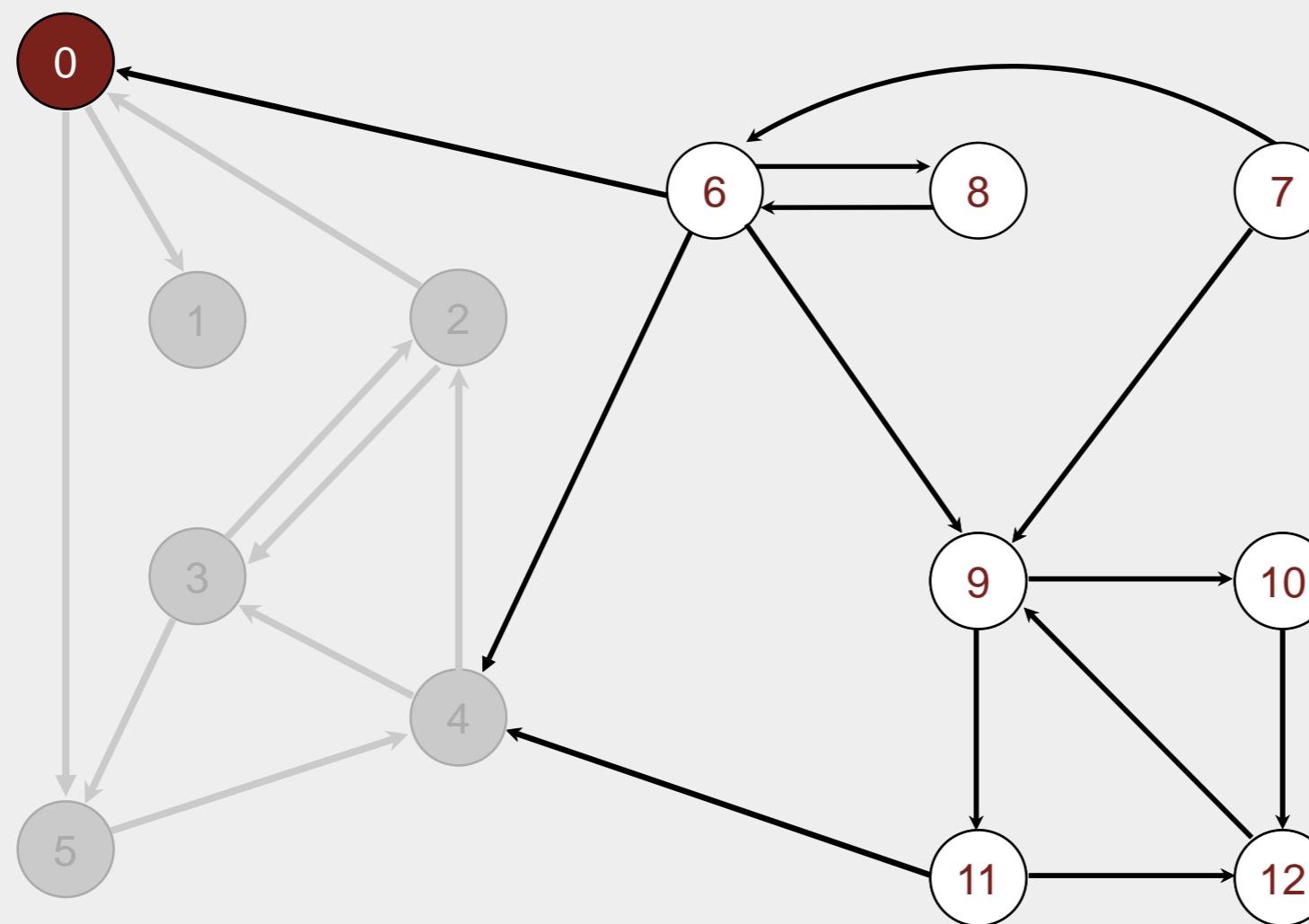
11 -

12 -

visit 0

Phase 2. Run DFS in G , visiting unmarked vertices in reverse postorder of G^R .

1 0 2 4 5 3 11 9 12 10 6 7 8



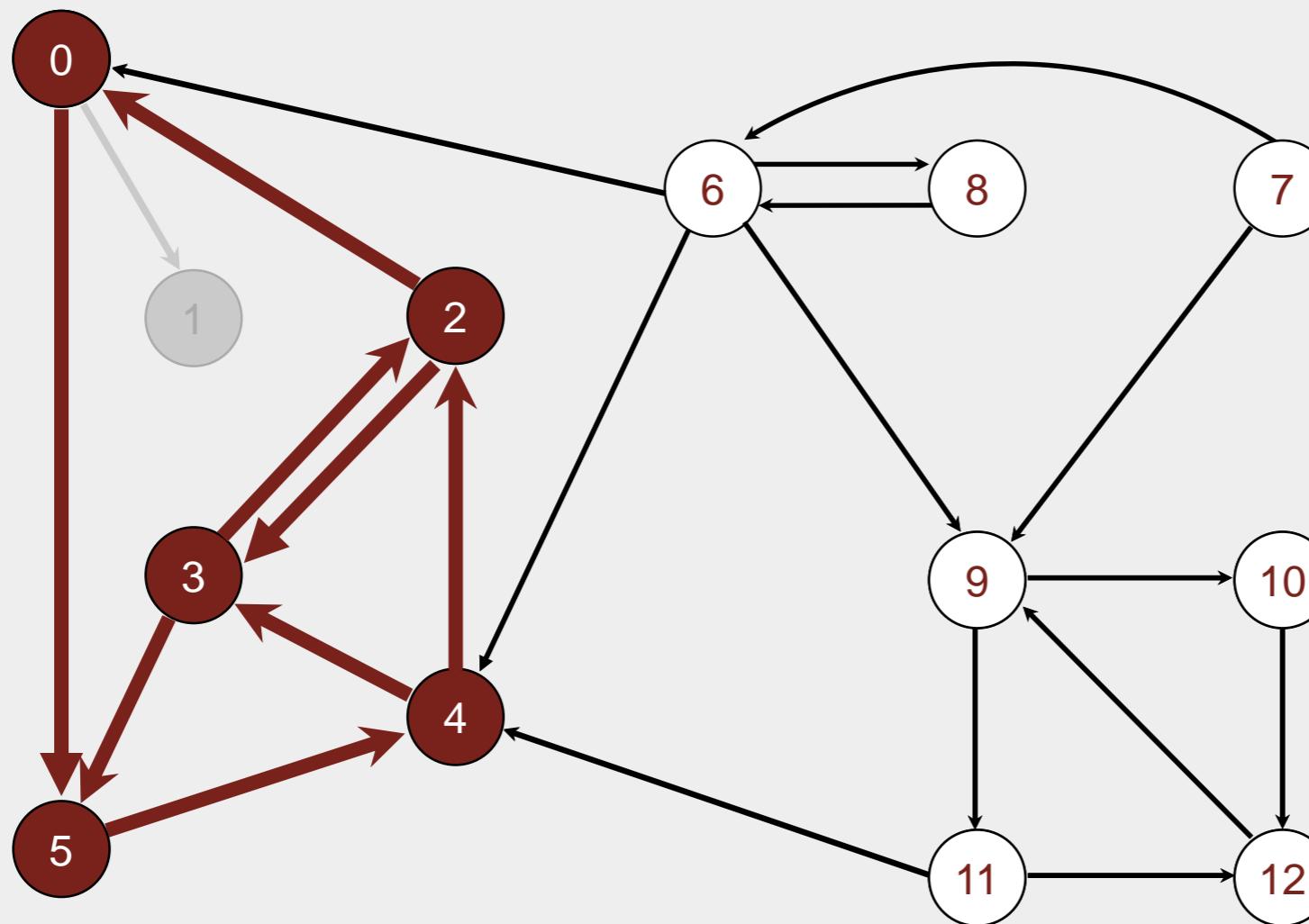
v $scc[v]$

v	$scc[v]$
0	1
1	0
2	1
3	1
4	1
5	1
6	-
7	-
8	-
9	-
10	-
11	-
12	-

0 done

Phase 2. Run DFS in G , visiting unmarked vertices in reverse postorder of G^R .

1 0 2 4 5 3 11 9 12 10 6 7 8

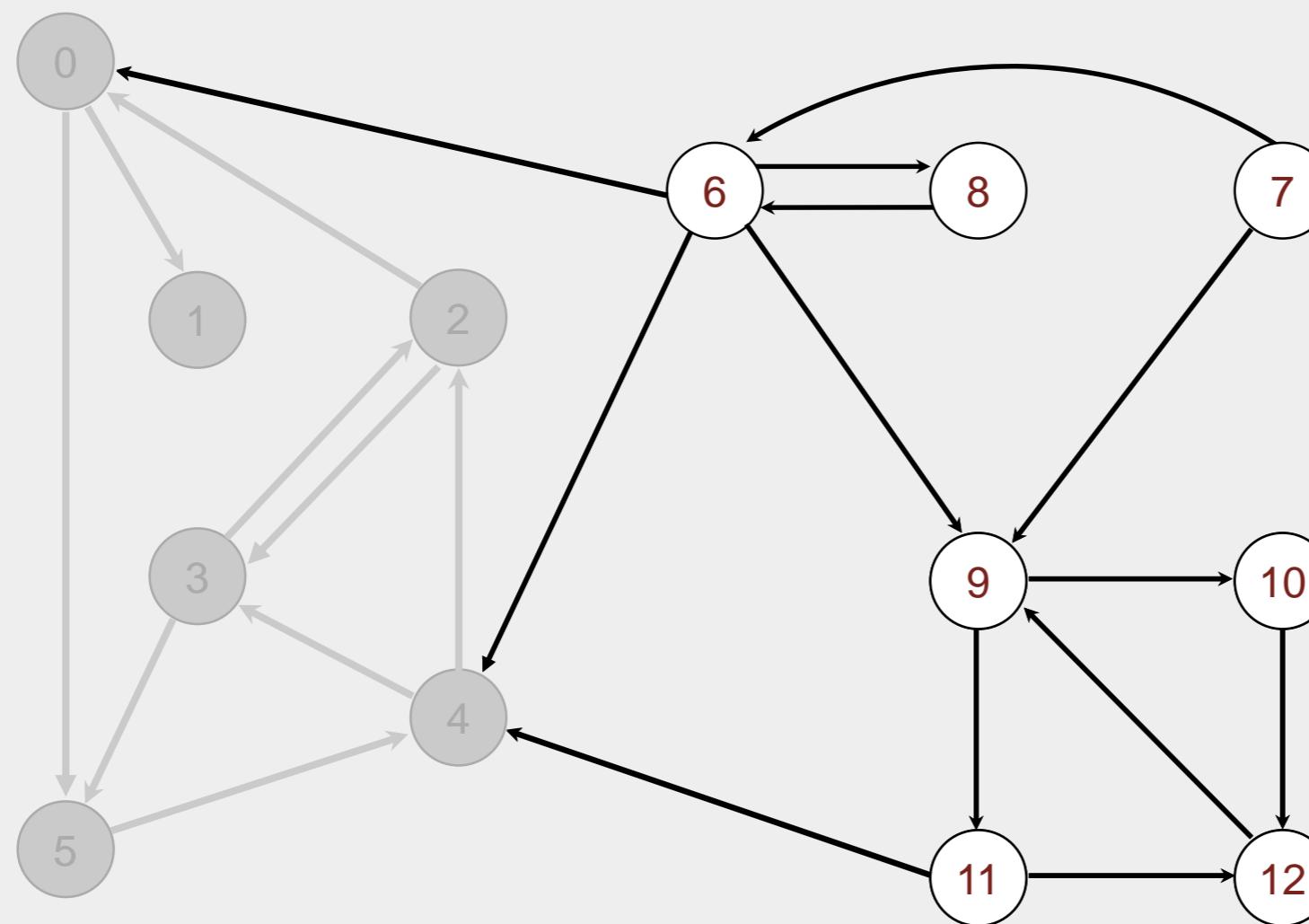


v	scc[v]
0	1
1	0
2	1
3	1
4	1
5	1
6	-
7	-
8	-
9	-
10	-
11	-
12	-

strong component: 0 2 3 4 5

Phase 2. Run DFS in G , visiting unmarked vertices in reverse postorder of G^R .

1 0 **2 4** 5 3 11 9 12 10 6 7 8



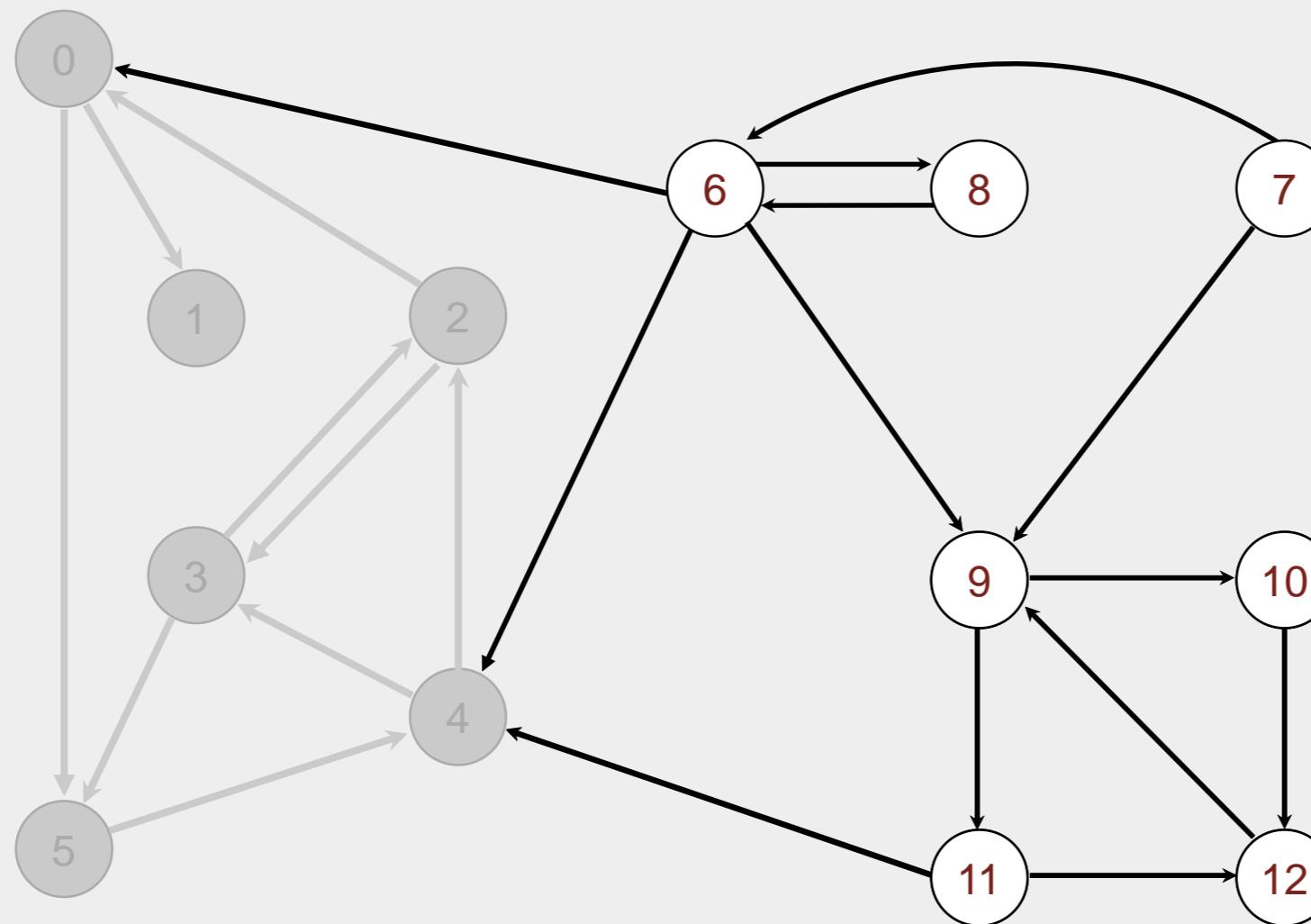
v scc[v]

v	scc[v]
0	1
1	0
2	1
3	1
4	1
5	1
6	-
7	-
8	-
9	-
10	-
11	-
12	-

check 2

Phase 2. Run DFS in G , visiting unmarked vertices in reverse postorder of G^R .

1 0 2 4 5 3 11 9 12 10 6 7 8

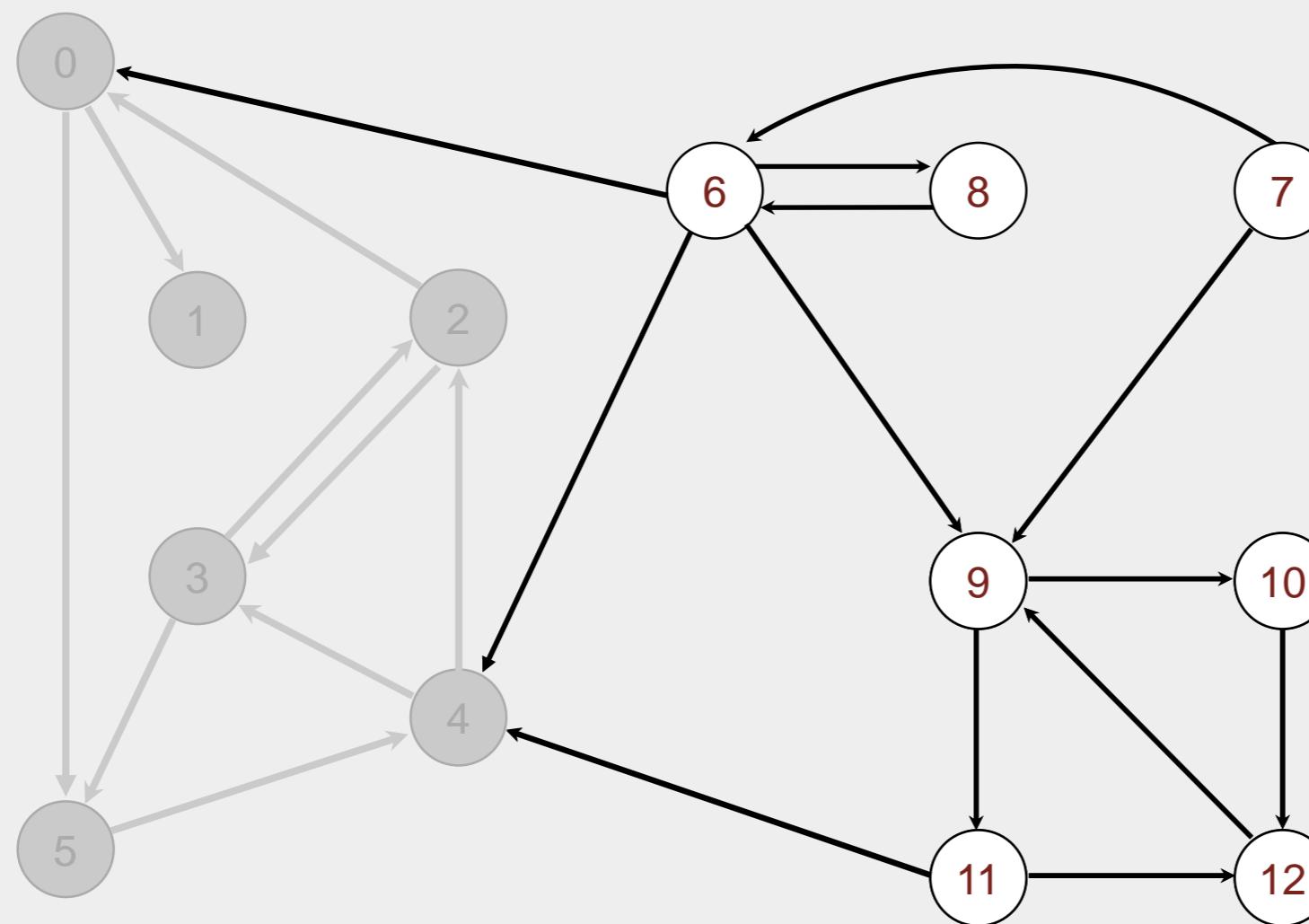


v	scc[v]
0	1
1	0
2	1
3	1
4	1
5	1
6	-
7	-
8	-
9	-
10	-
11	-
12	-

check 4

Phase 2. Run DFS in G , visiting unmarked vertices in reverse postorder of G^R .

1 0 2 4 5 3 11 9 12 10 6 7 8



v $scc[v]$

0 1

1 0

2 1

3 1

4 1

5 1

6 -

7 -

8 -

9 -

10 -

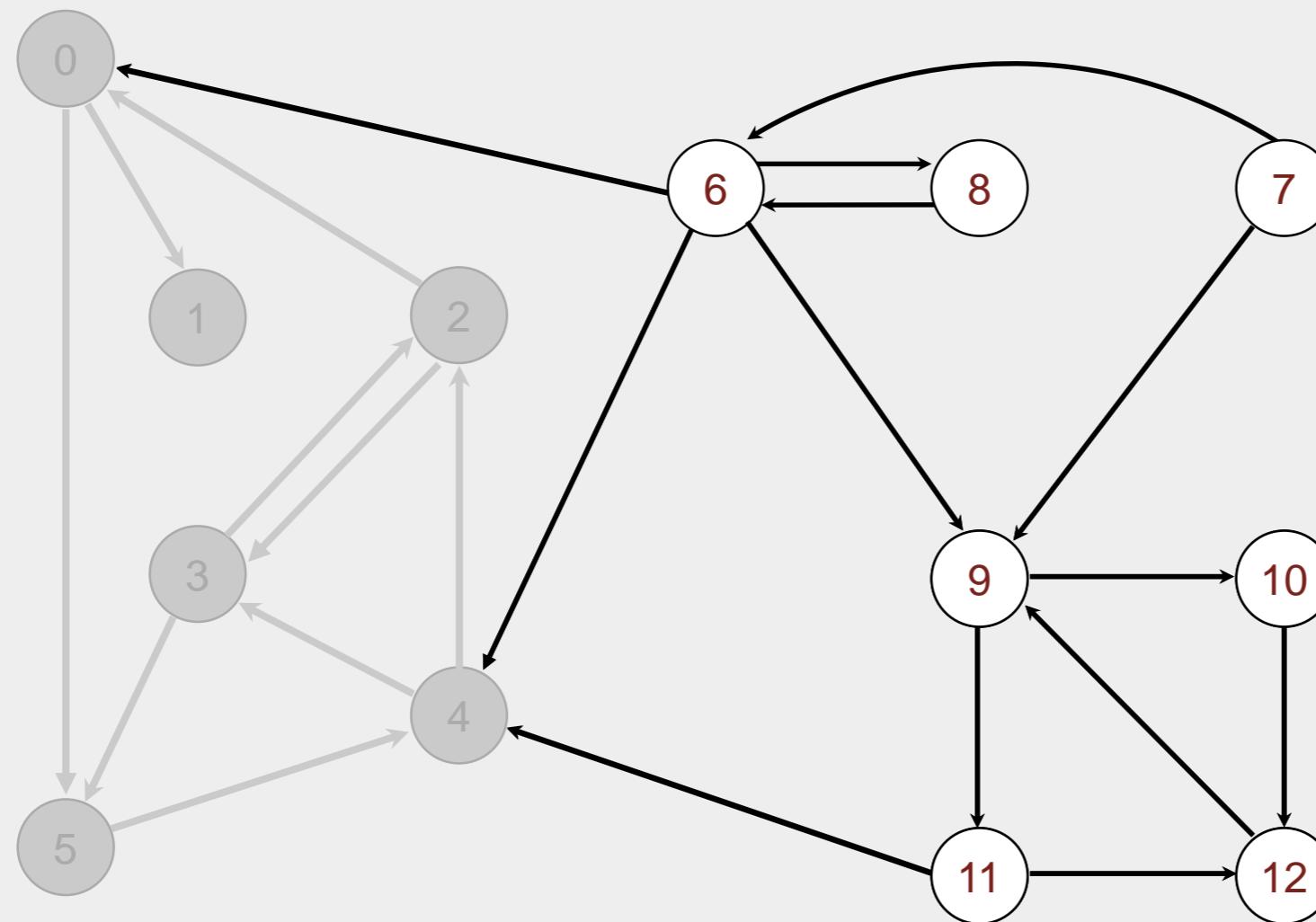
11 -

12 -

check 5

Phase 2. Run DFS in G , visiting unmarked vertices in reverse postorder of G^R .

1 0 2 4 5 3 (11) 9 12 10 6 7 8

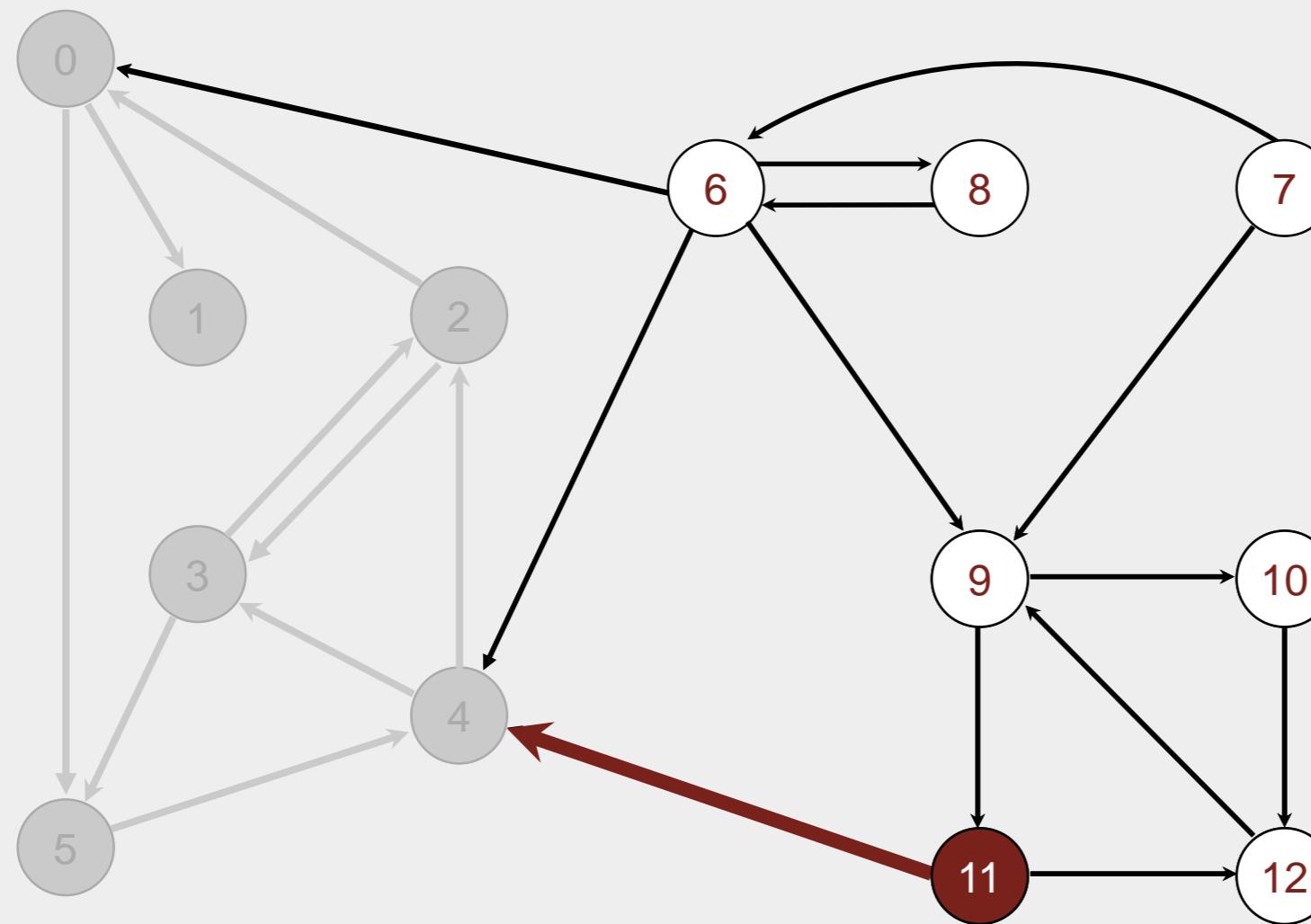


v	scc[v]
0	1
1	0
2	1
3	1
4	1
5	1
6	-
7	-
8	-
9	-
10	-
11	-
12	-

check 3

Phase 2. Run DFS in G , visiting unmarked vertices in reverse postorder of G^R .

1 0 2 4 5 3 11 9 12 10 6 7 8



v $scc[v]$

0 1

1 0

2 1

3 1

4 1

5 1

6 -

7 -

8 -

9 -

10 -

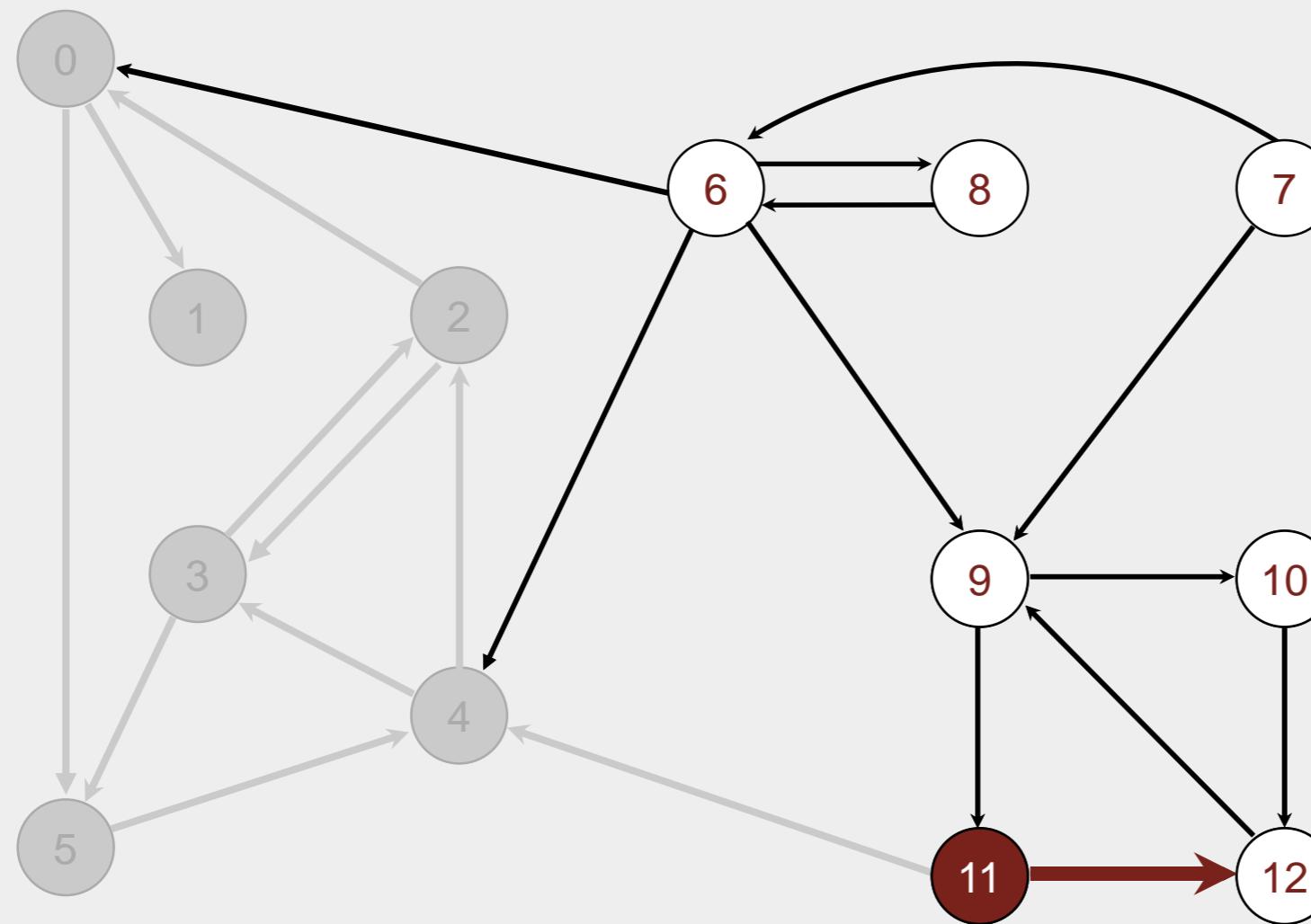
11 2

12 -

visit 11

Phase 2. Run DFS in G , visiting unmarked vertices in reverse postorder of G^R .

1 0 2 4 5 3 11 9 12 10 6 7 8



v $scc[v]$

0 1

1 0

2 1

3 1

4 1

5 1

6 -

7 -

8 -

9 -

10 -

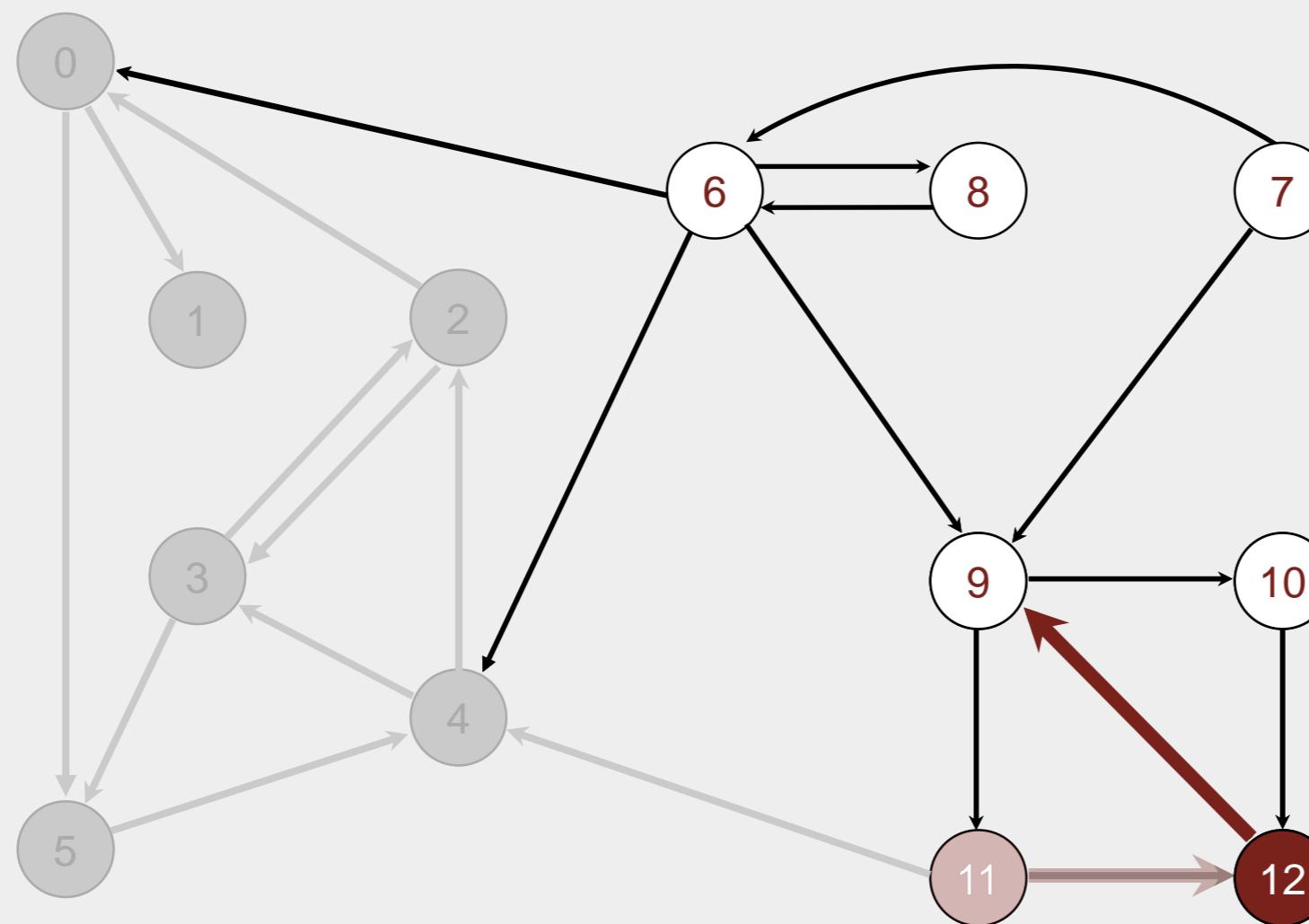
11 2

12 -

visit 11

Phase 2. Run DFS in G , visiting unmarked vertices in reverse postorder of G^R .

1 0 2 4 5 3 11 9 12 10 6 7 8



visit 12

v $scc[v]$

0	1
---	---

1	0
---	---

2	1
---	---

3	1
---	---

4	1
---	---

5	1
---	---

6	-
---	---

7	-
---	---

8	-
---	---

9	-
---	---

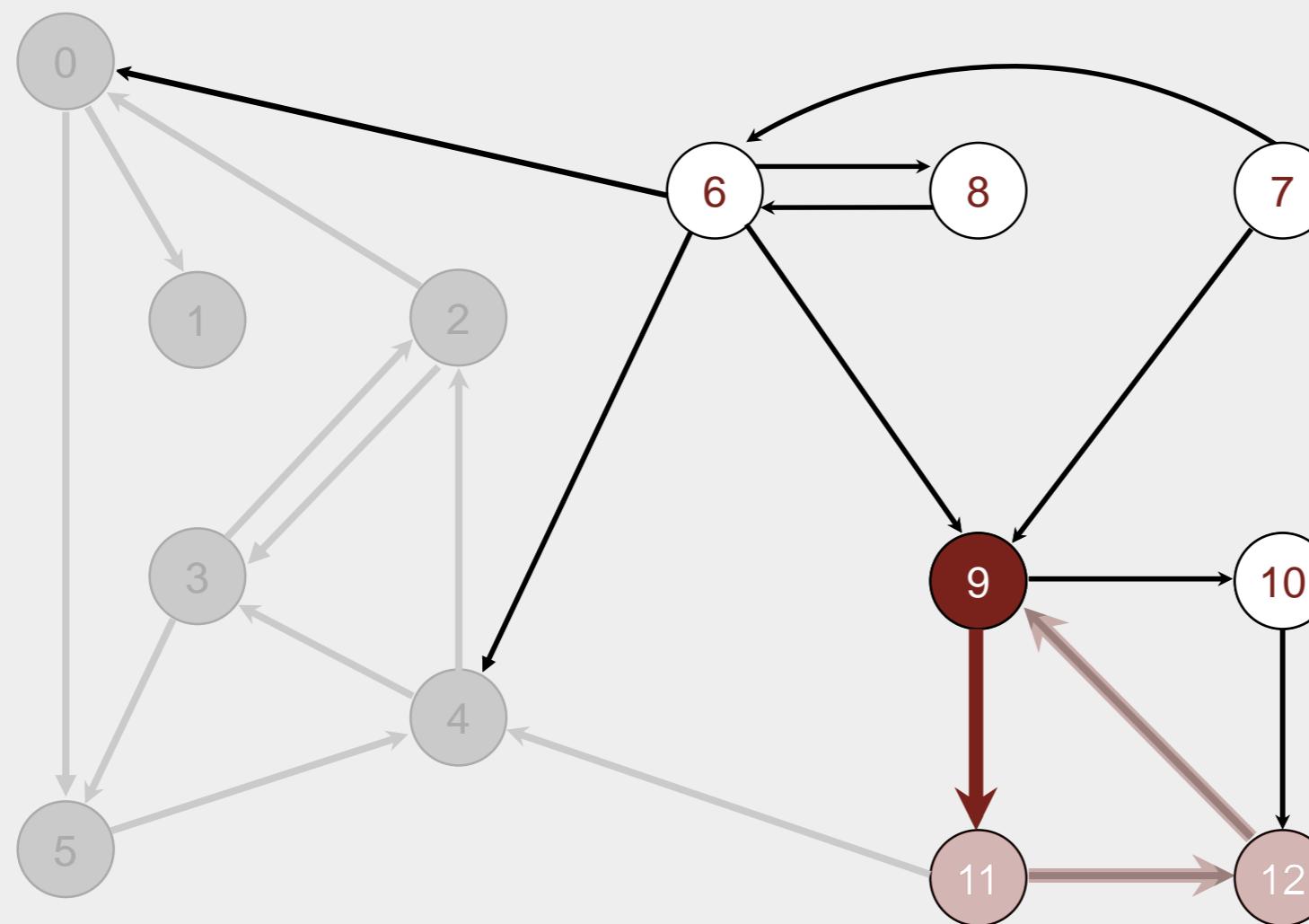
10	-
----	---

11	2
----	---

12	2
----	---

Phase 2. Run DFS in G , visiting unmarked vertices in reverse postorder of G^R .

1 0 2 4 5 3 11 9 12 10 6 7 8



v $scc[v]$

0 1

1 0

2 1

3 1

4 1

5 1

6 -

7 -

8 -

9 2

10 -

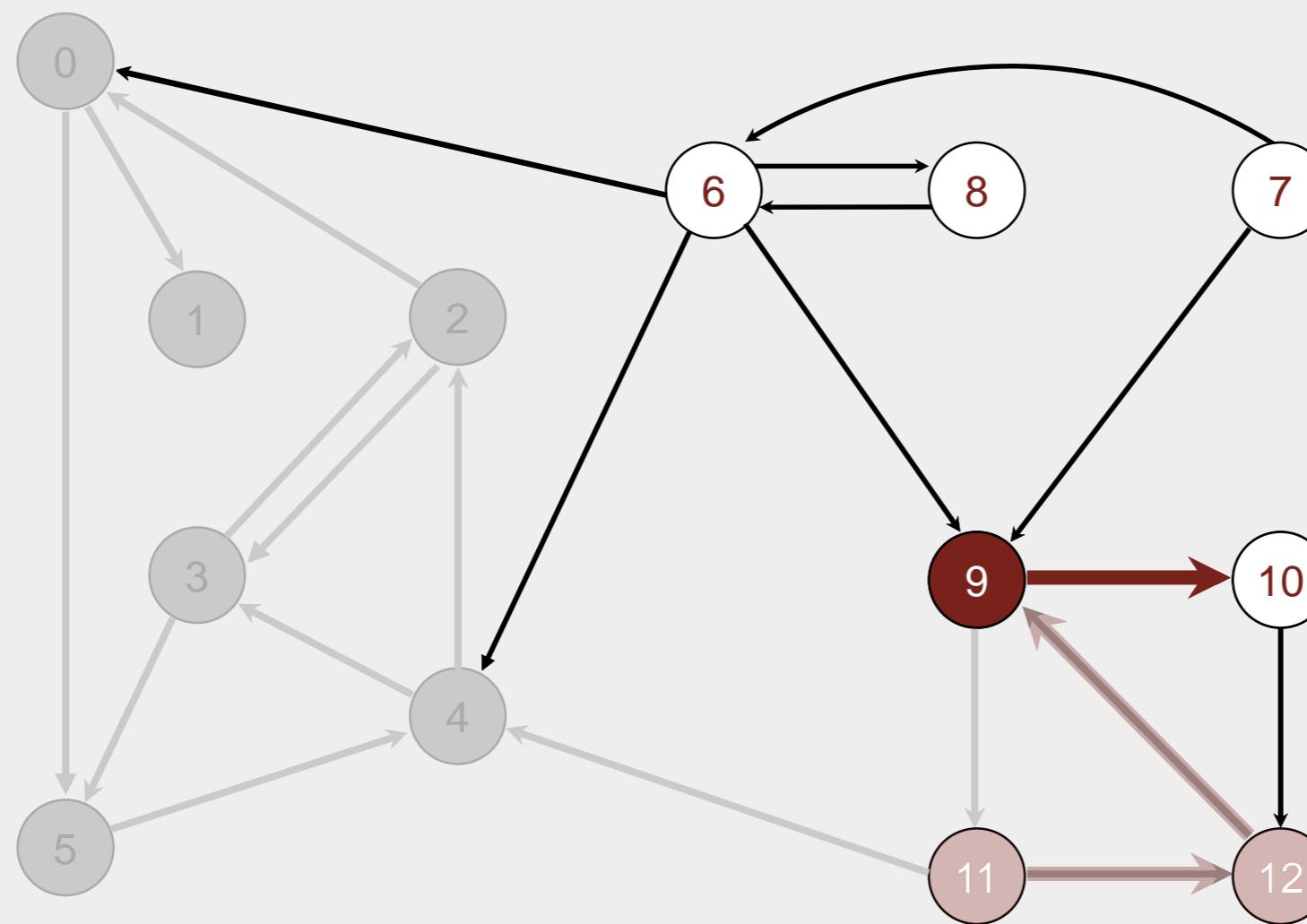
11 2

12 2

visit 9

Phase 2. Run DFS in G , visiting unmarked vertices in reverse postorder of G^R .

1 0 2 4 5 3 11 9 12 10 6 7 8



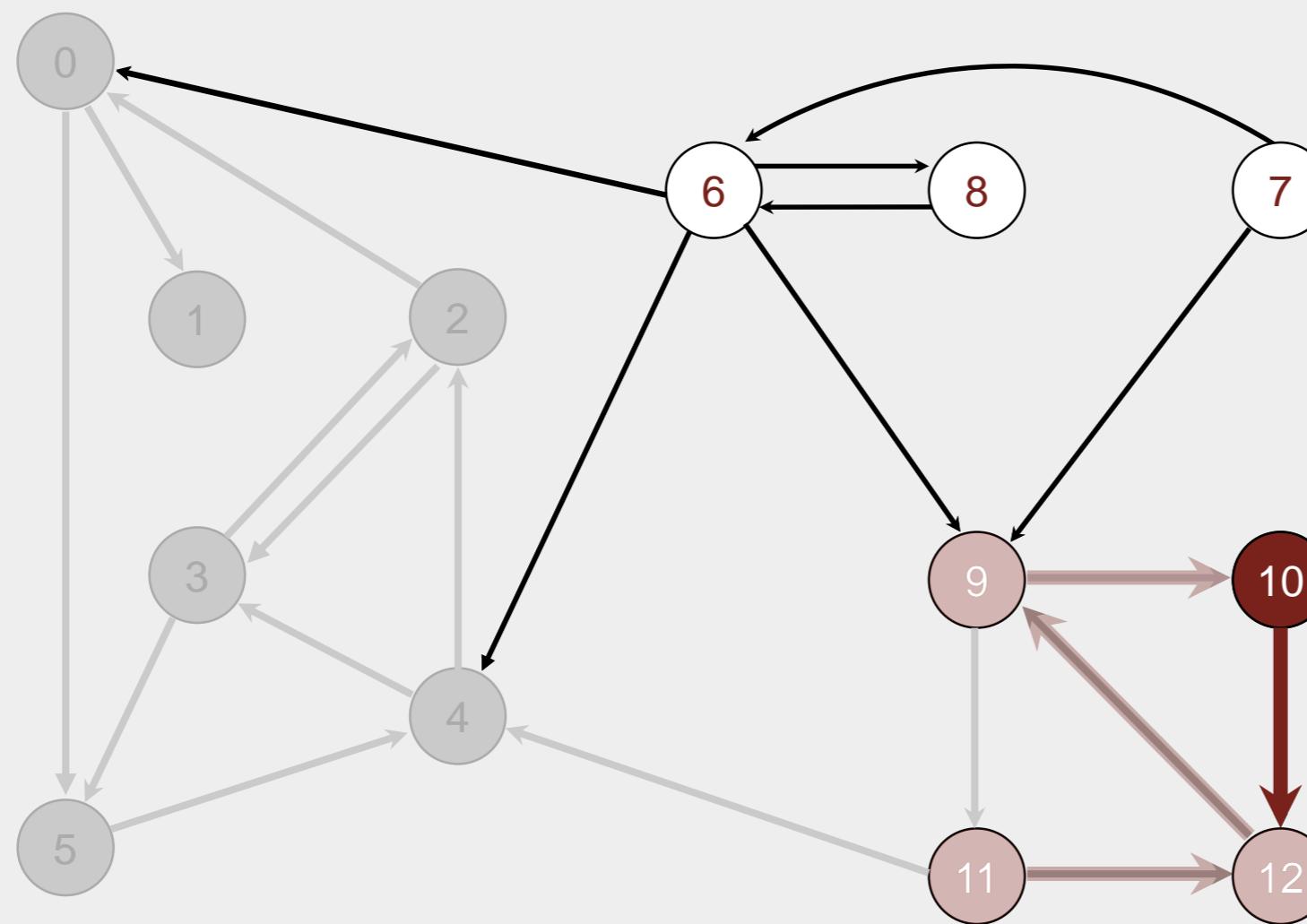
v $scc[v]$

v	$scc[v]$
0	1
1	0
2	1
3	1
4	1
5	1
6	-
7	-
8	-
9	2
10	-
11	2
12	2

visit 9

Phase 2. Run DFS in G , visiting unmarked vertices in reverse postorder of G^R .

1 0 2 4 5 3 11 9 12 10 6 7 8



v $scc[v]$

0 1

1 0

2 1

3 1

4 1

5 1

6 -

7 -

8 -

9 2

10 2

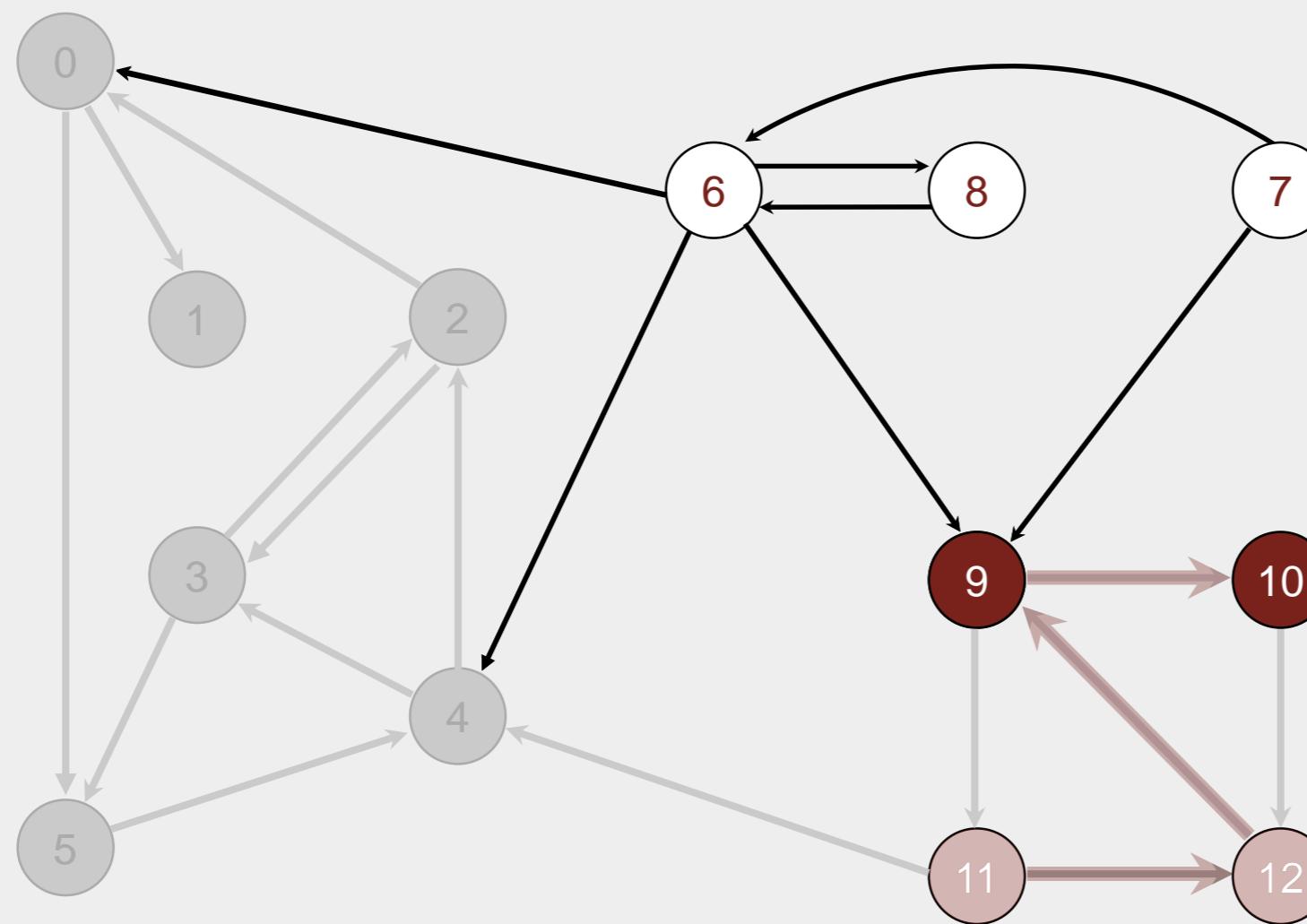
11 2

12 2

visit 10

Phase 2. Run DFS in G , visiting unmarked vertices in reverse postorder of G^R .

1 0 2 4 5 3 11 9 12 10 6 7 8



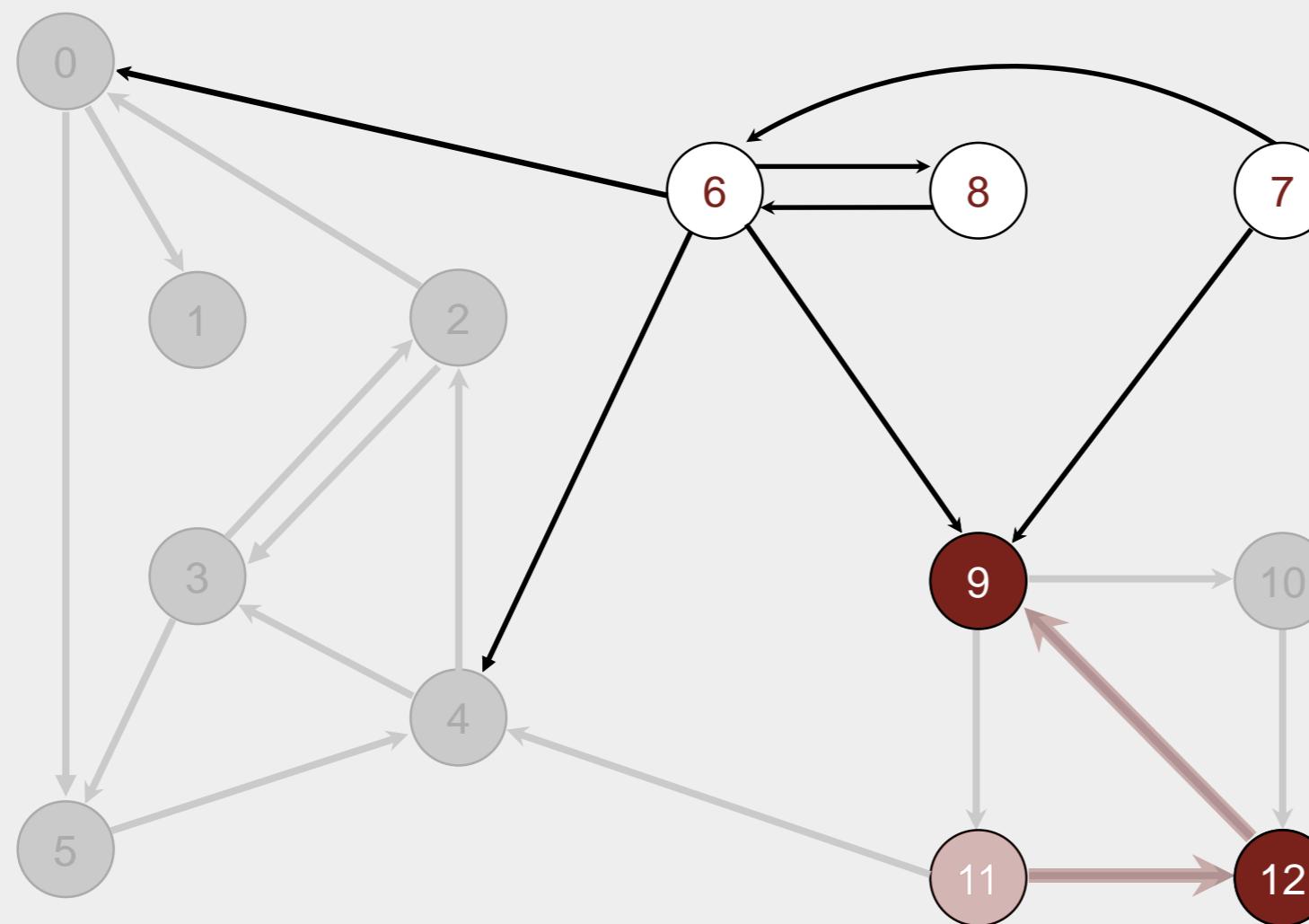
v $scc[v]$

v	$scc[v]$
0	1
1	0
2	1
3	1
4	1
5	1
6	-
7	-
8	-
9	2
10	2
11	2
12	2

10 done

Phase 2. Run DFS in G , visiting unmarked vertices in reverse postorder of G^R .

1 0 2 4 5 3 11 9 12 10 6 7 8



v $scc[v]$

0 1

1 0

2 1

3 1

4 1

5 1

6 -

7 -

8 -

9 2

10 2

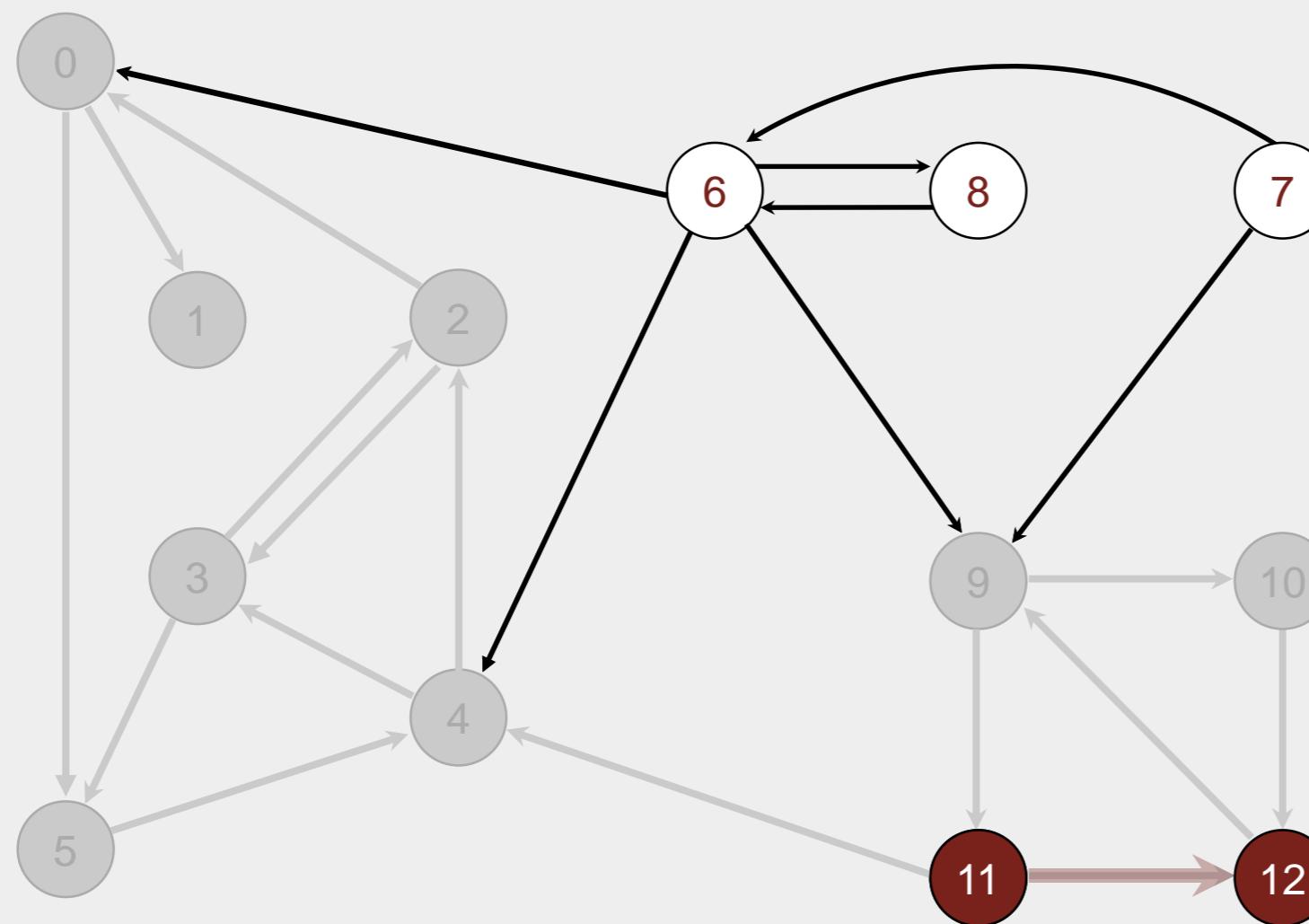
11 2

12 2

9 done

Phase 2. Run DFS in G , visiting unmarked vertices in reverse postorder of G^R .

1 0 2 4 5 3 11 9 12 10 6 7 8



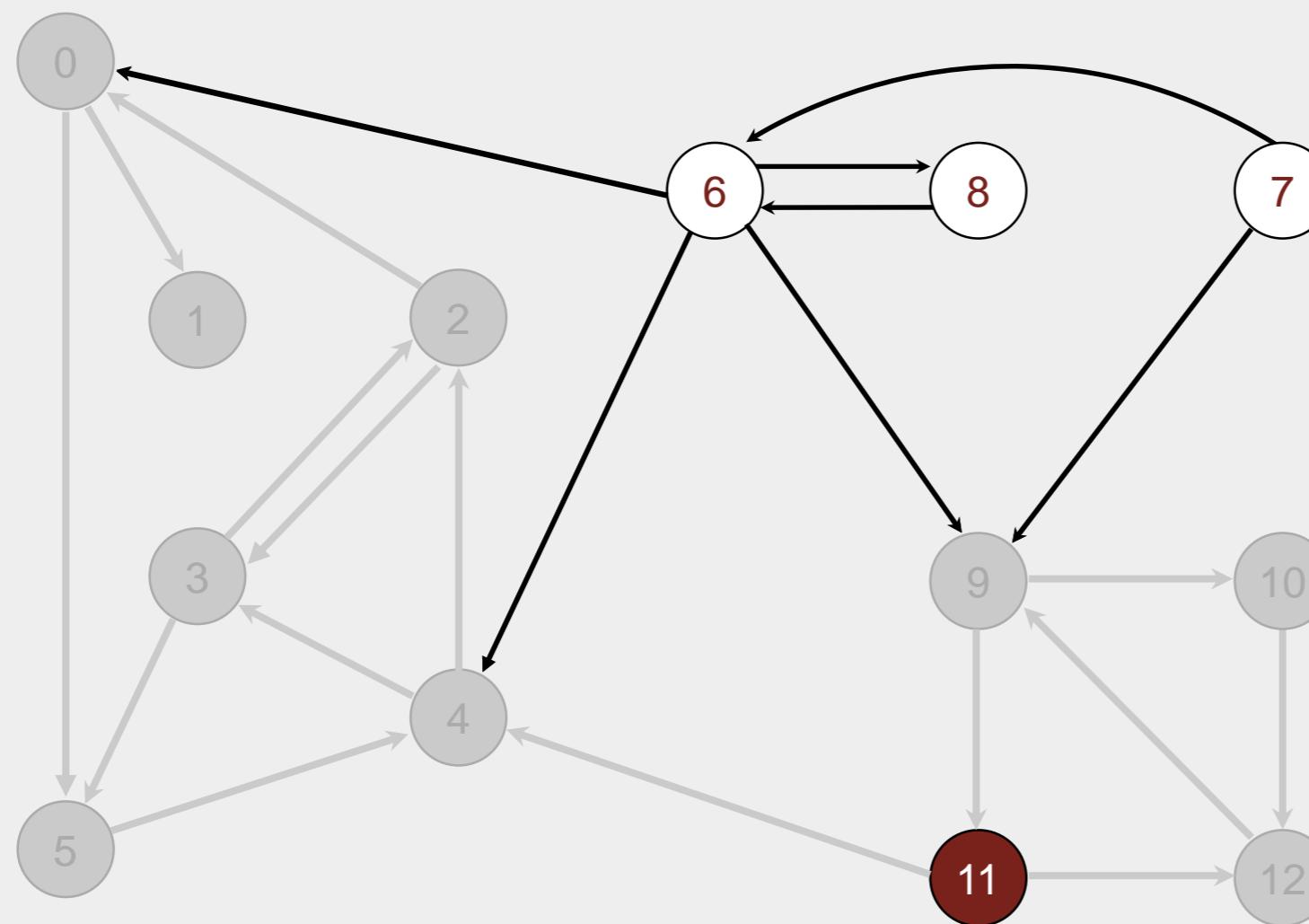
v $scc[v]$

v	$scc[v]$
0	1
1	0
2	1
3	1
4	1
5	1
6	-
7	-
8	-
9	2
10	2
11	2
12	2

12 done

Phase 2. Run DFS in G , visiting unmarked vertices in reverse postorder of G^R .

1 0 2 4 5 3 11 9 12 10 6 7 8



v $scc[v]$

0 1

1 0

2 1

3 1

4 1

5 1

6 -

7 -

8 -

9 2

10 2

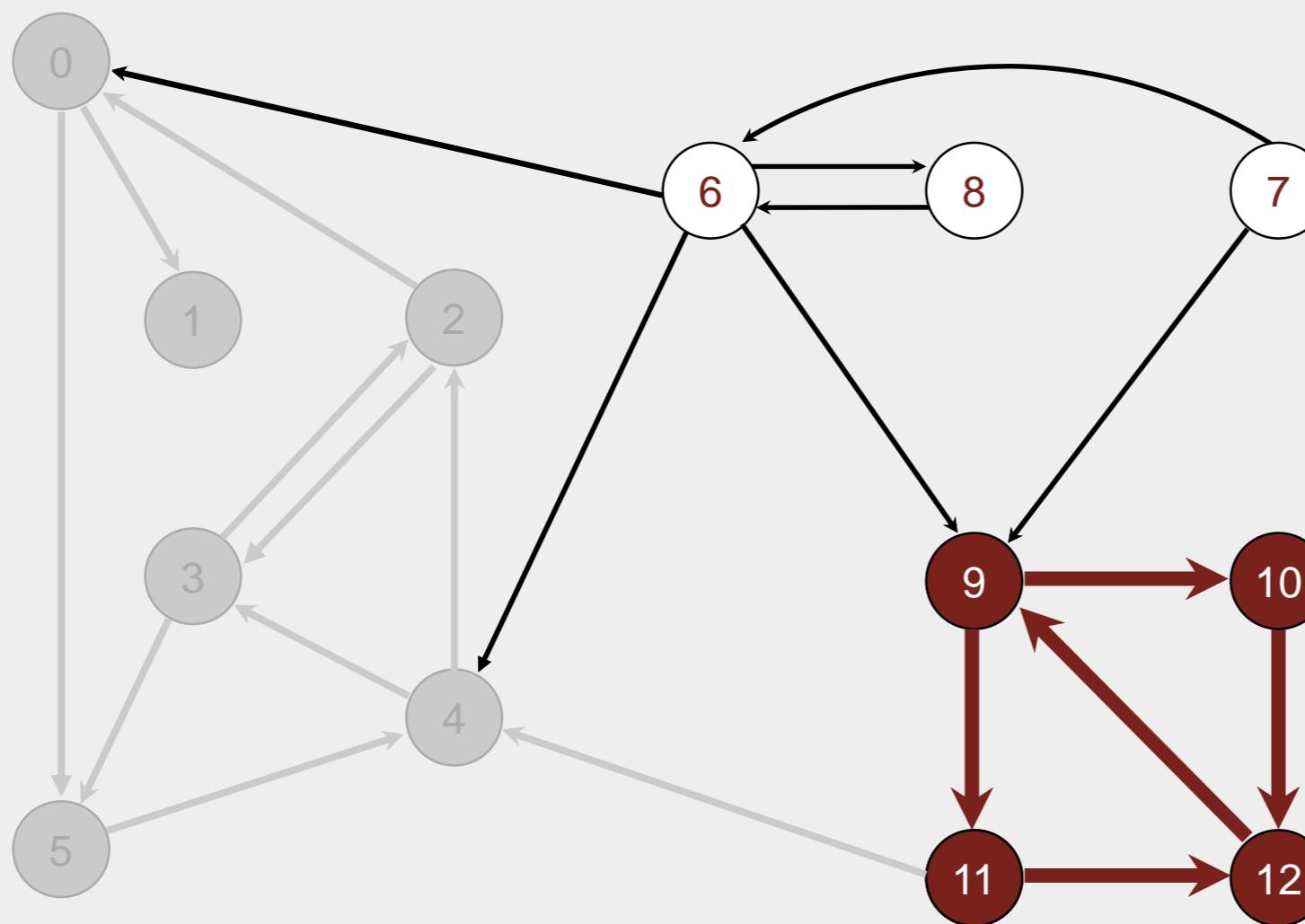
11 2

12 2

11 done

Phase 2. Run DFS in G , visiting unmarked vertices in reverse postorder of G^R .

1 0 2 4 5 3 11 9 12 10 6 7 8

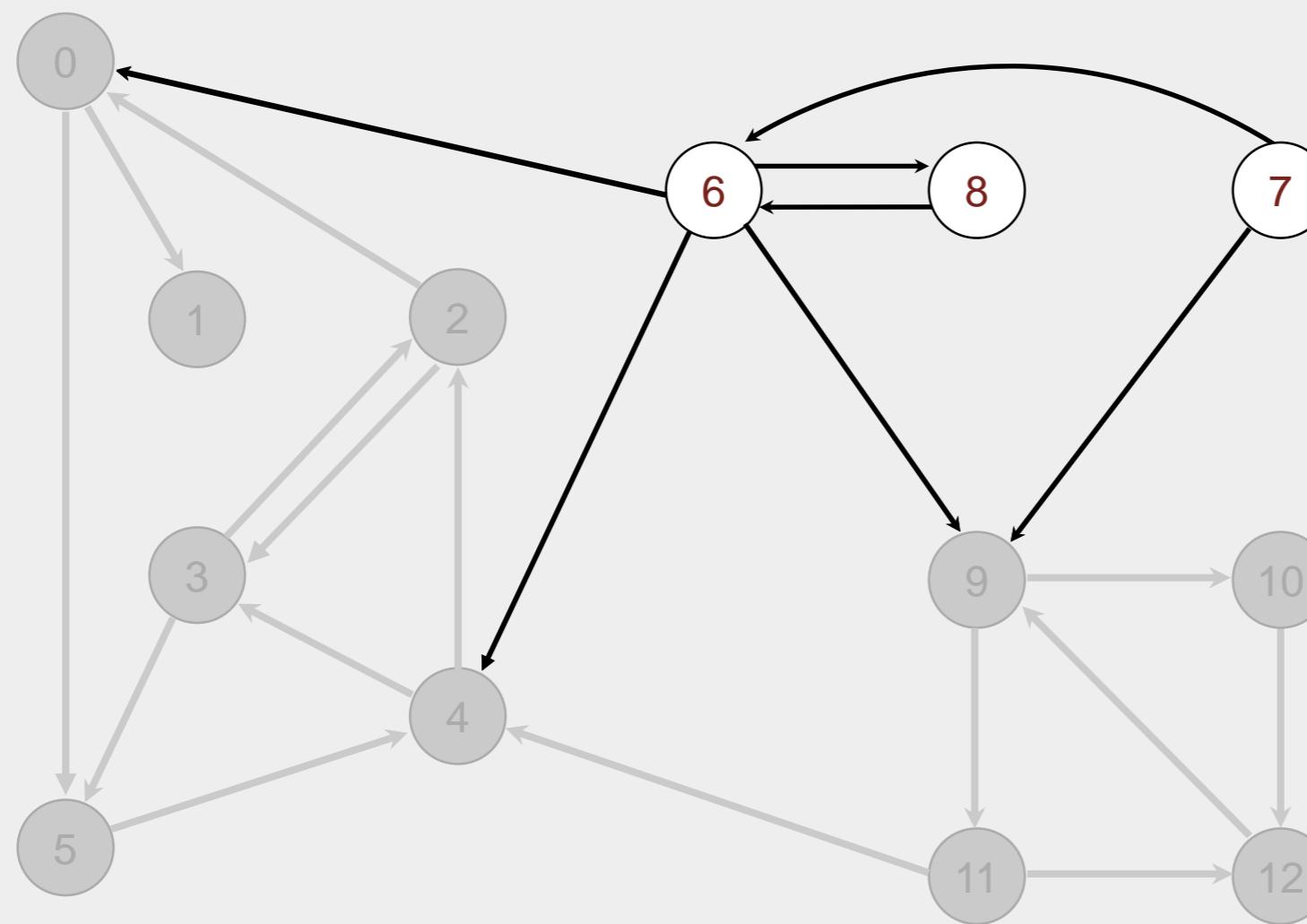


v	scc[v]
0	1
1	0
2	1
3	1
4	1
5	1
6	-
7	-
8	-
9	2
10	2
11	2
12	2

strong component: 9 10 11 12

Phase 2. Run DFS in G , visiting unmarked vertices in reverse postorder of G^R .

1 0 2 4 5 3 11 9 12 10 6 7 8



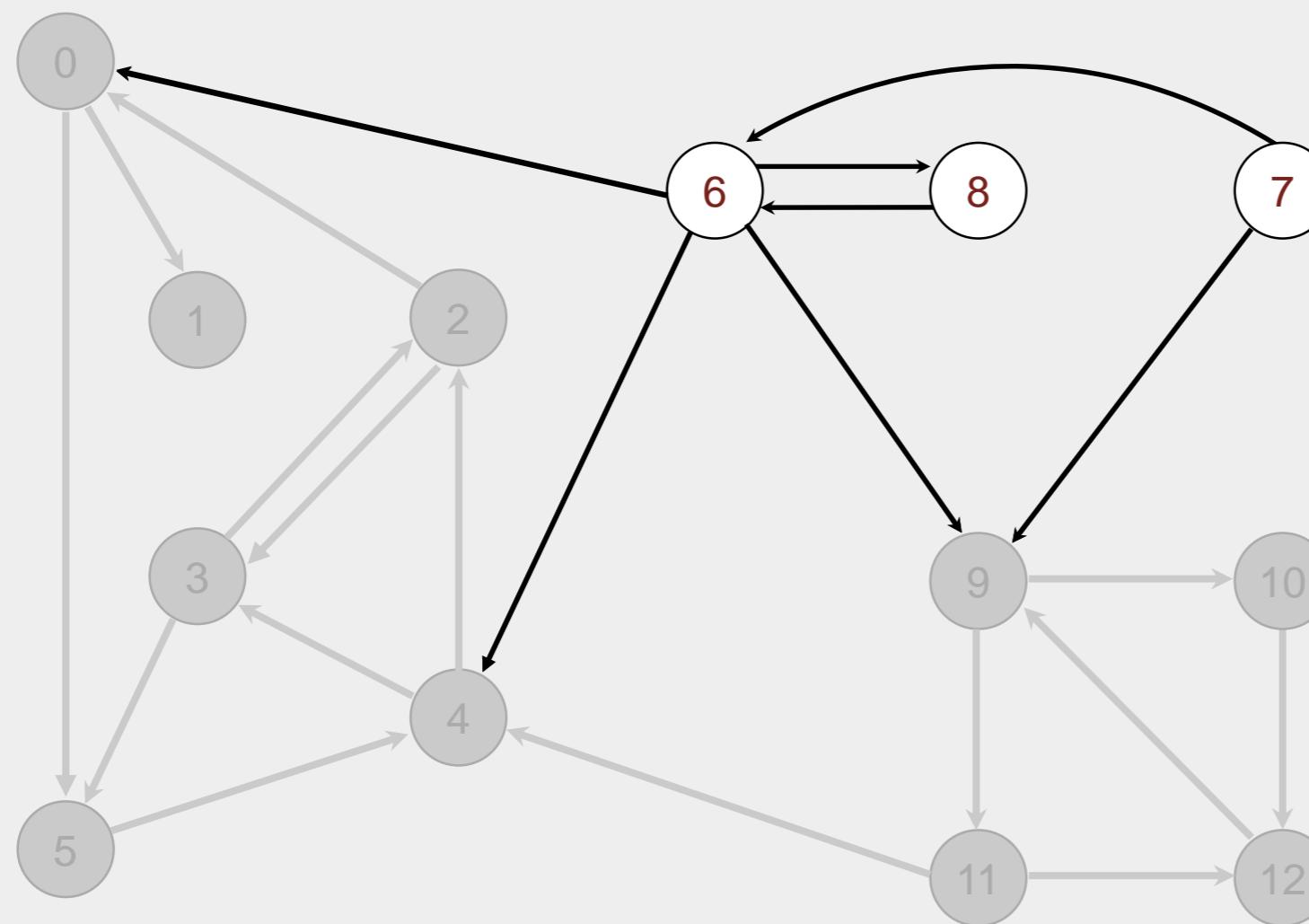
v $scc[v]$

v	$scc[v]$
0	1
1	0
2	1
3	1
4	1
5	1
6	-
7	-
8	-
9	2
10	2
11	2
12	2

check 9

Phase 2. Run DFS in G , visiting unmarked vertices in reverse postorder of G^R .

1 0 2 4 5 3 11 9 12 10 6 7 8



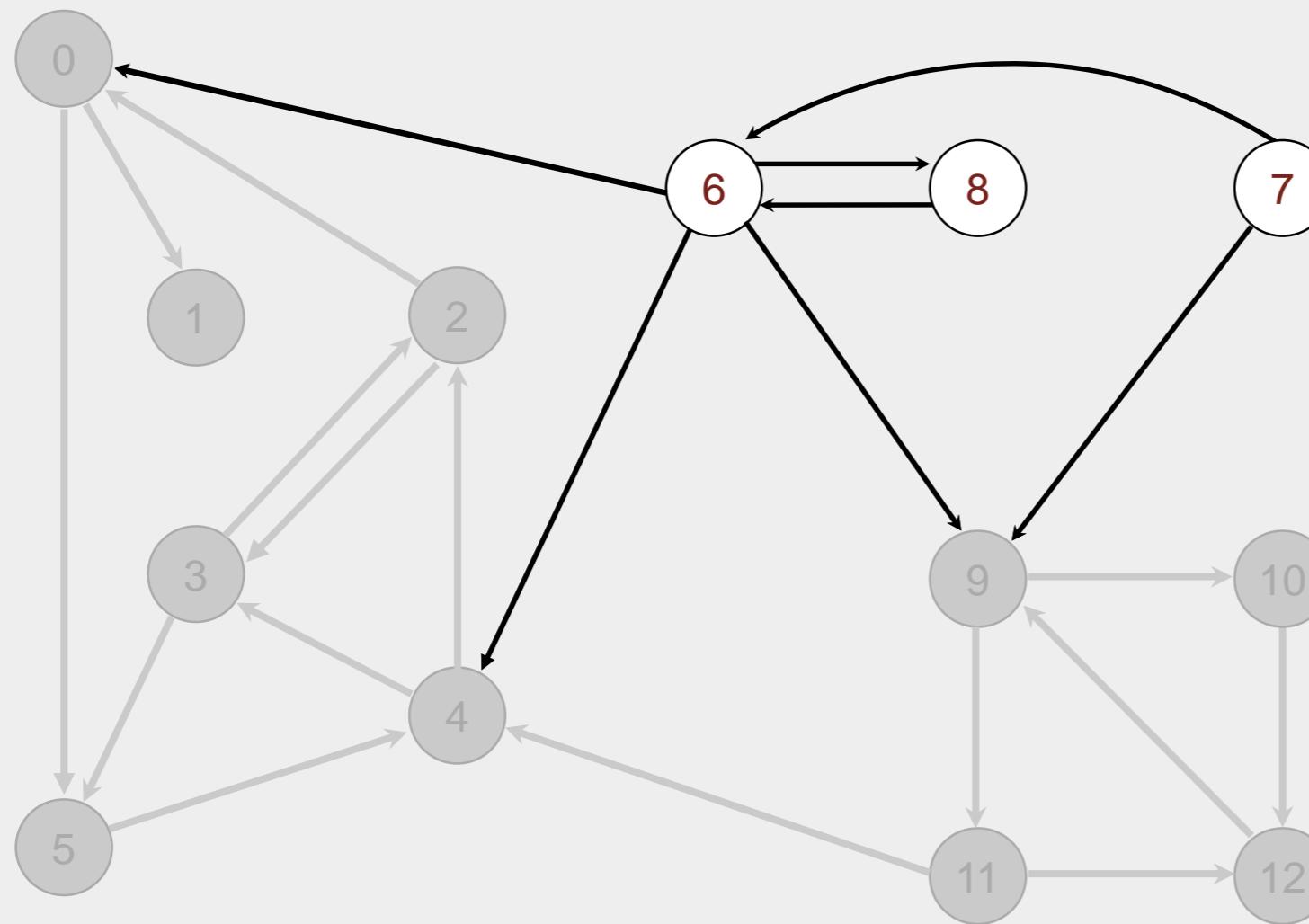
v $scc[v]$

v	$scc[v]$
0	1
1	0
2	1
3	1
4	1
5	1
6	-
7	-
8	-
9	2
10	2
11	2
12	2

check 12

Phase 2. Run DFS in G , visiting unmarked vertices in reverse postorder of G^R .

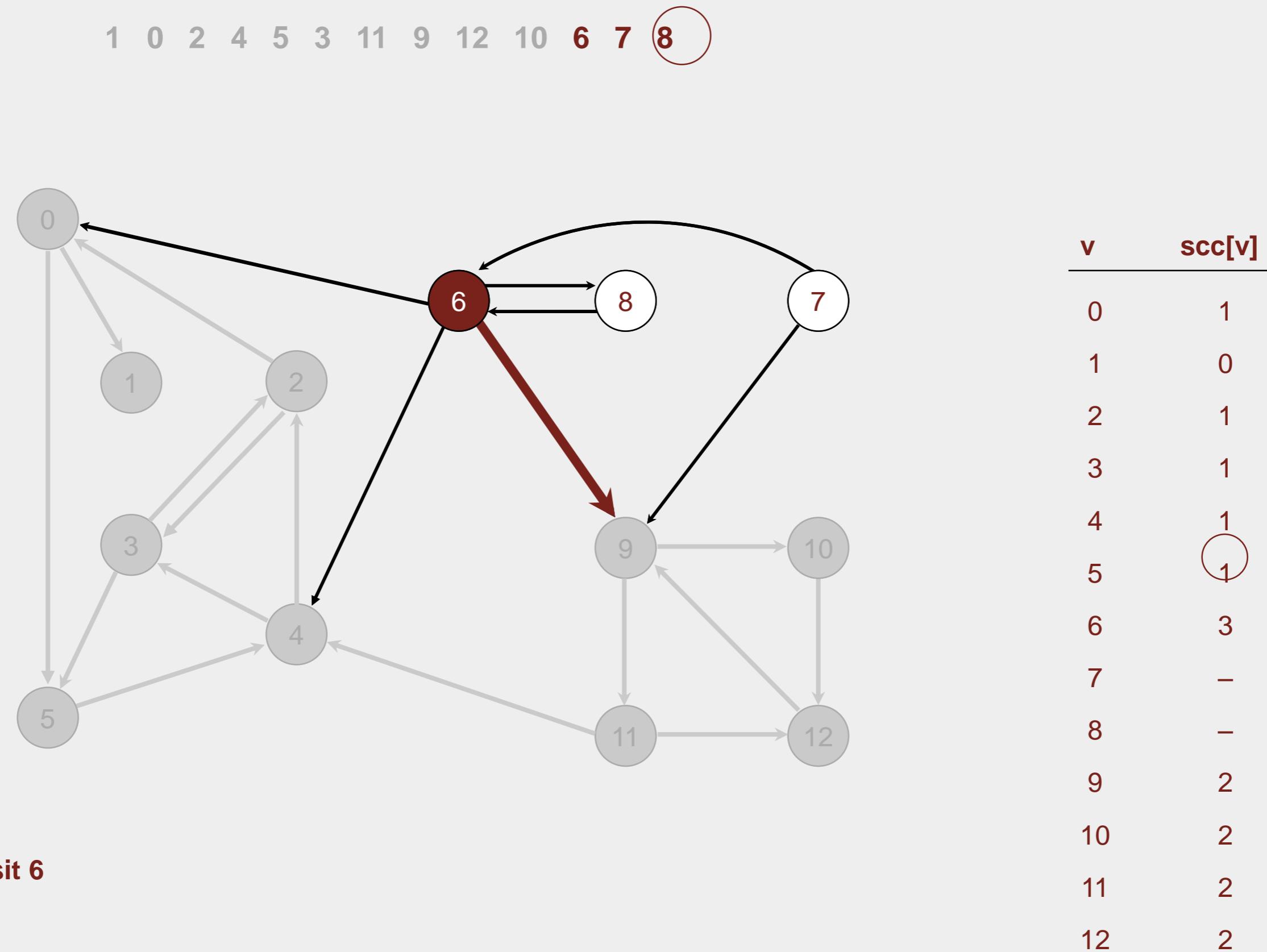
1 0 2 4 5 3 11 9 12 10 6 7 8



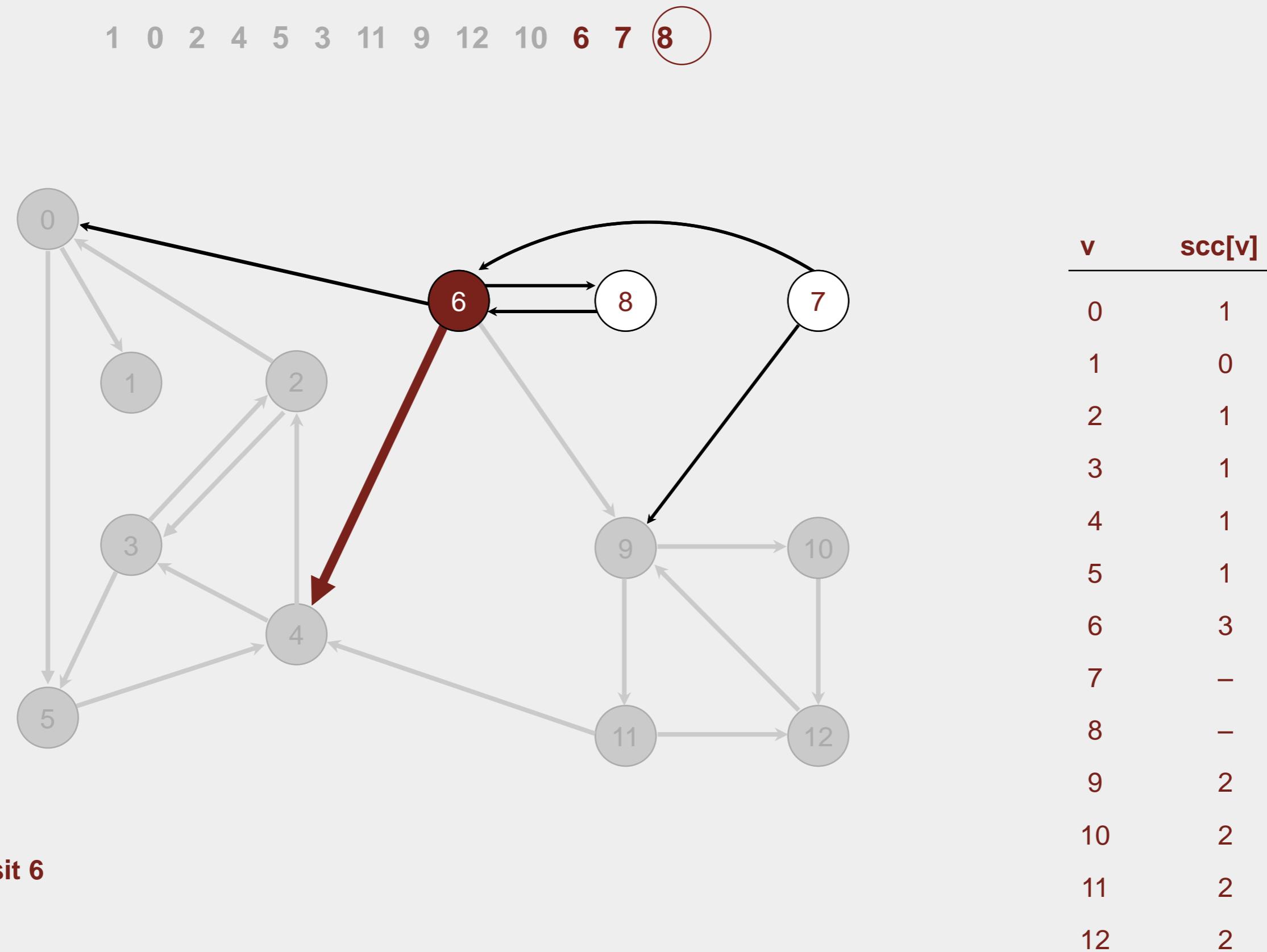
v	scc[v]
0	1
1	0
2	1
3	1
4	1
5	1
6	-
7	-
8	-
9	2
10	2
11	2
12	2

check 10

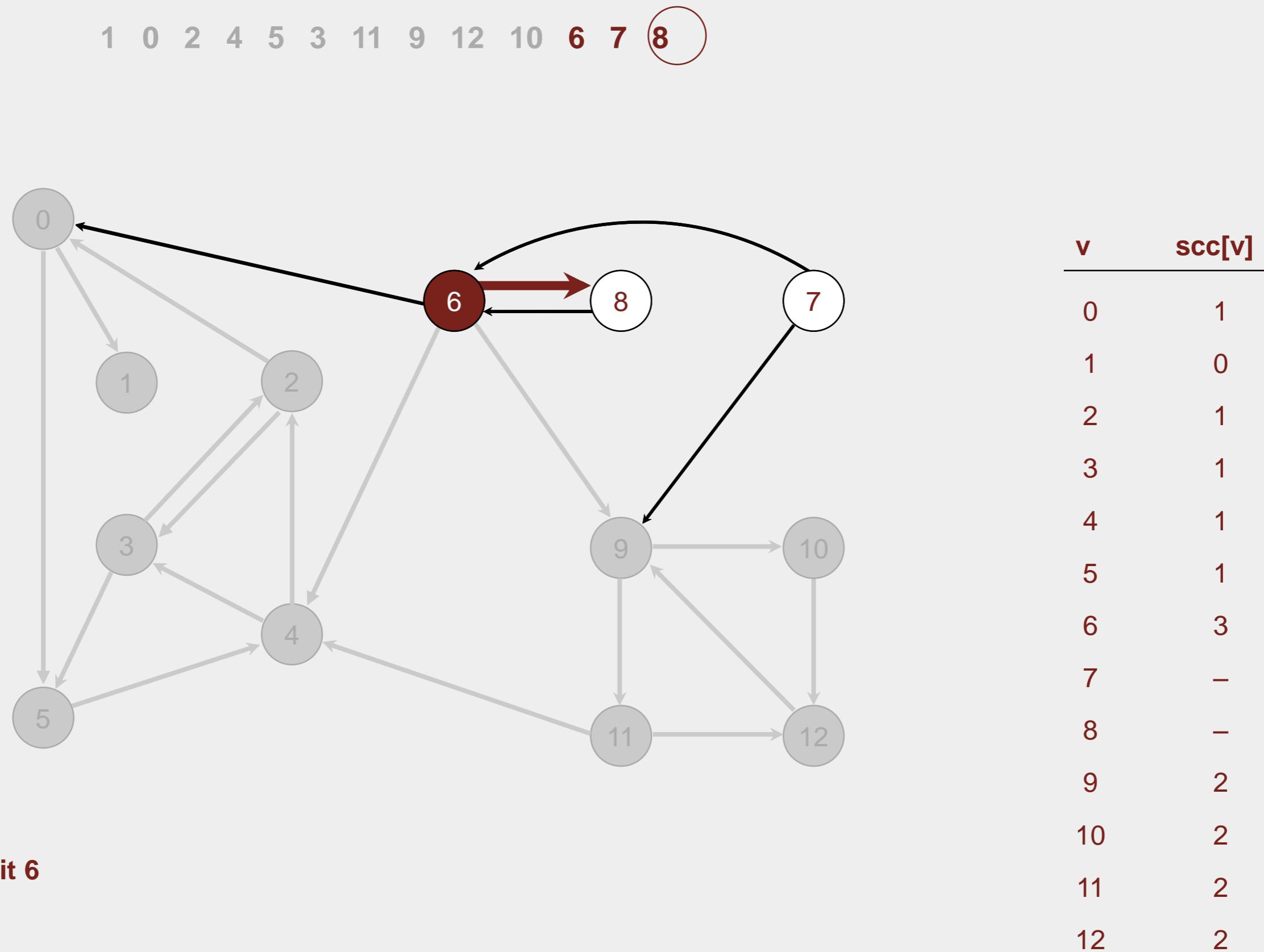
Phase 2. Run DFS in G , visiting unmarked vertices in reverse postorder of G^R .



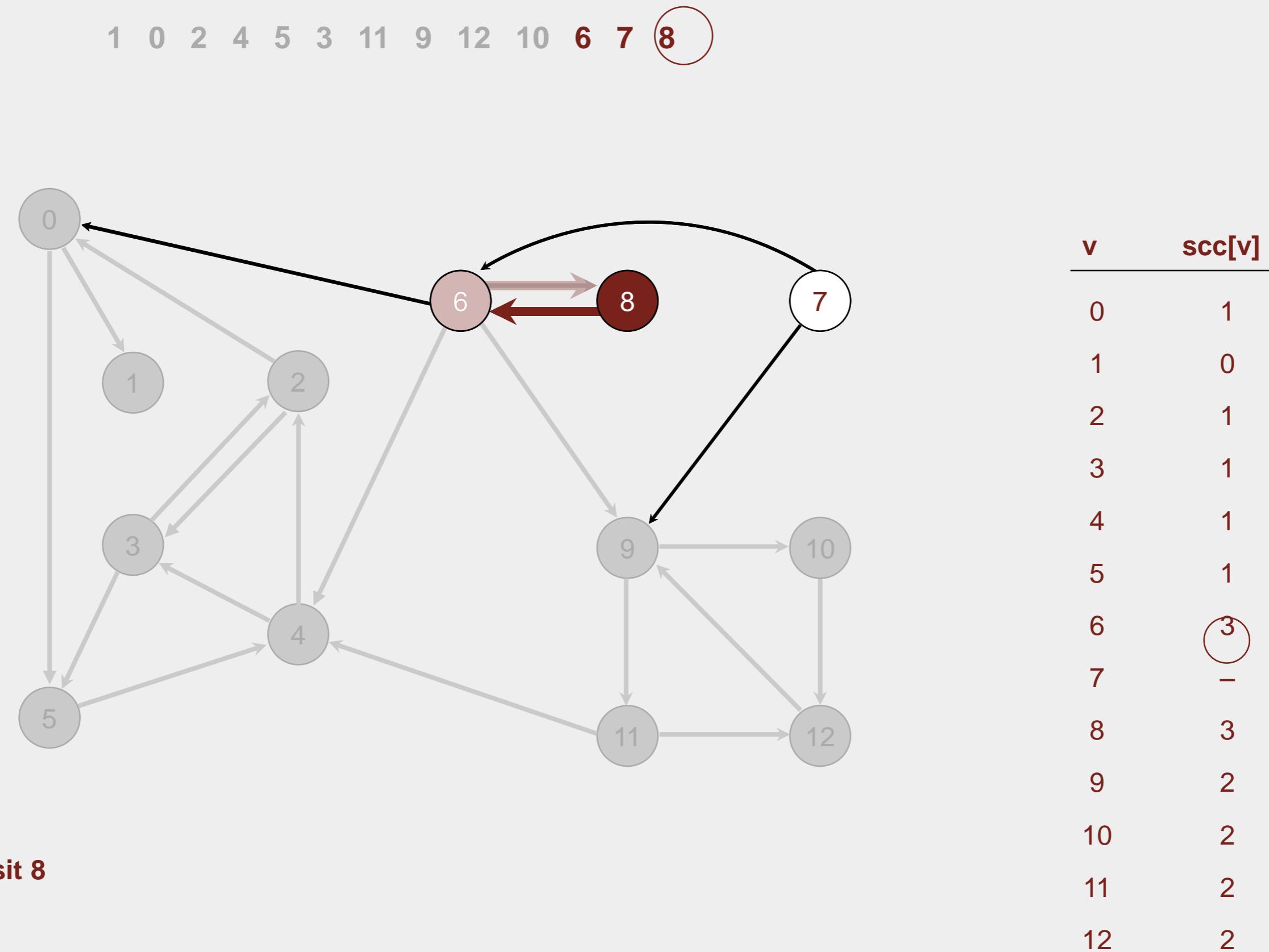
Phase 2. Run DFS in G , visiting unmarked vertices in reverse postorder of G^R .



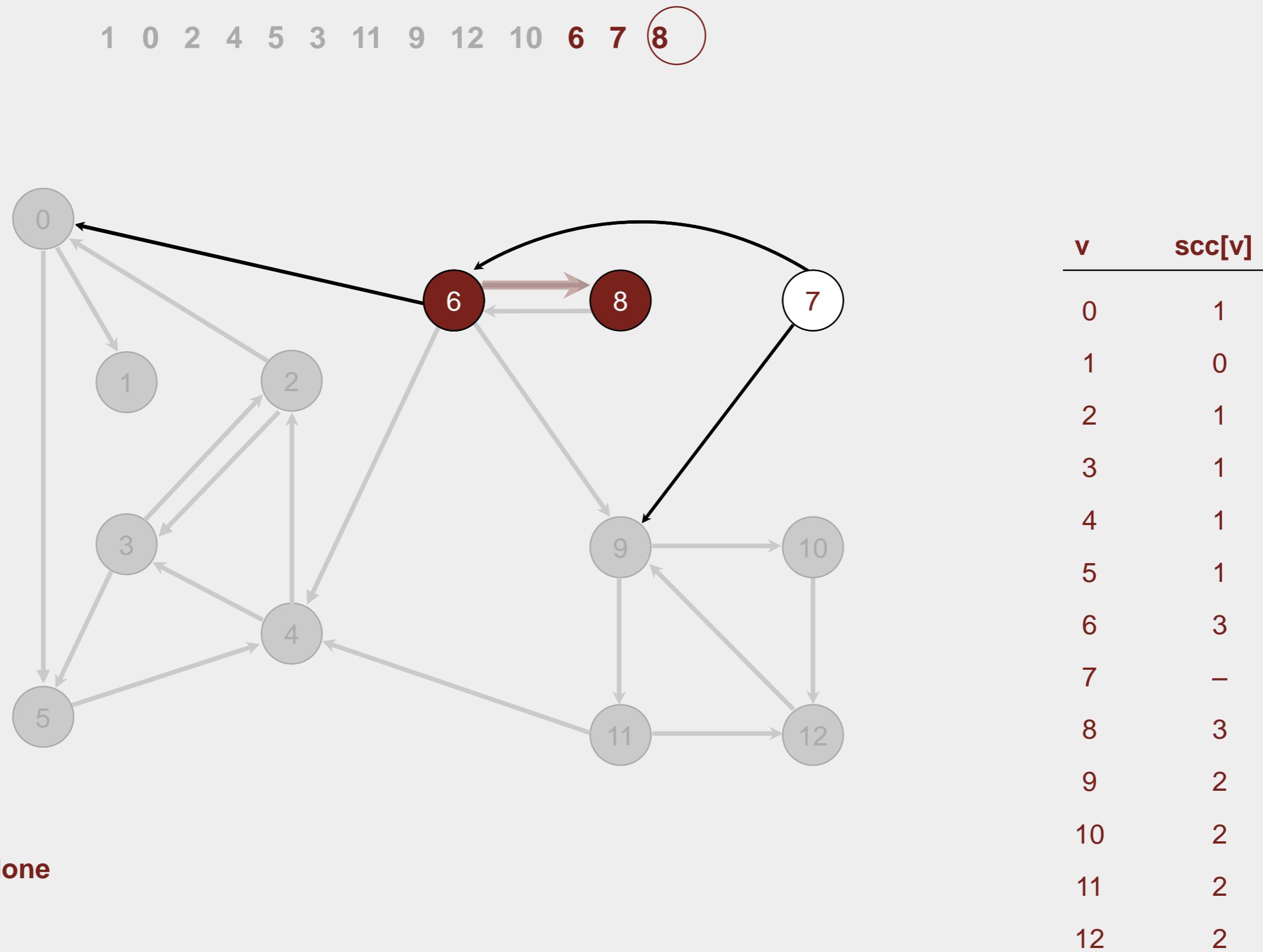
Phase 2. Run DFS in G , visiting unmarked vertices in reverse postorder of G^R .



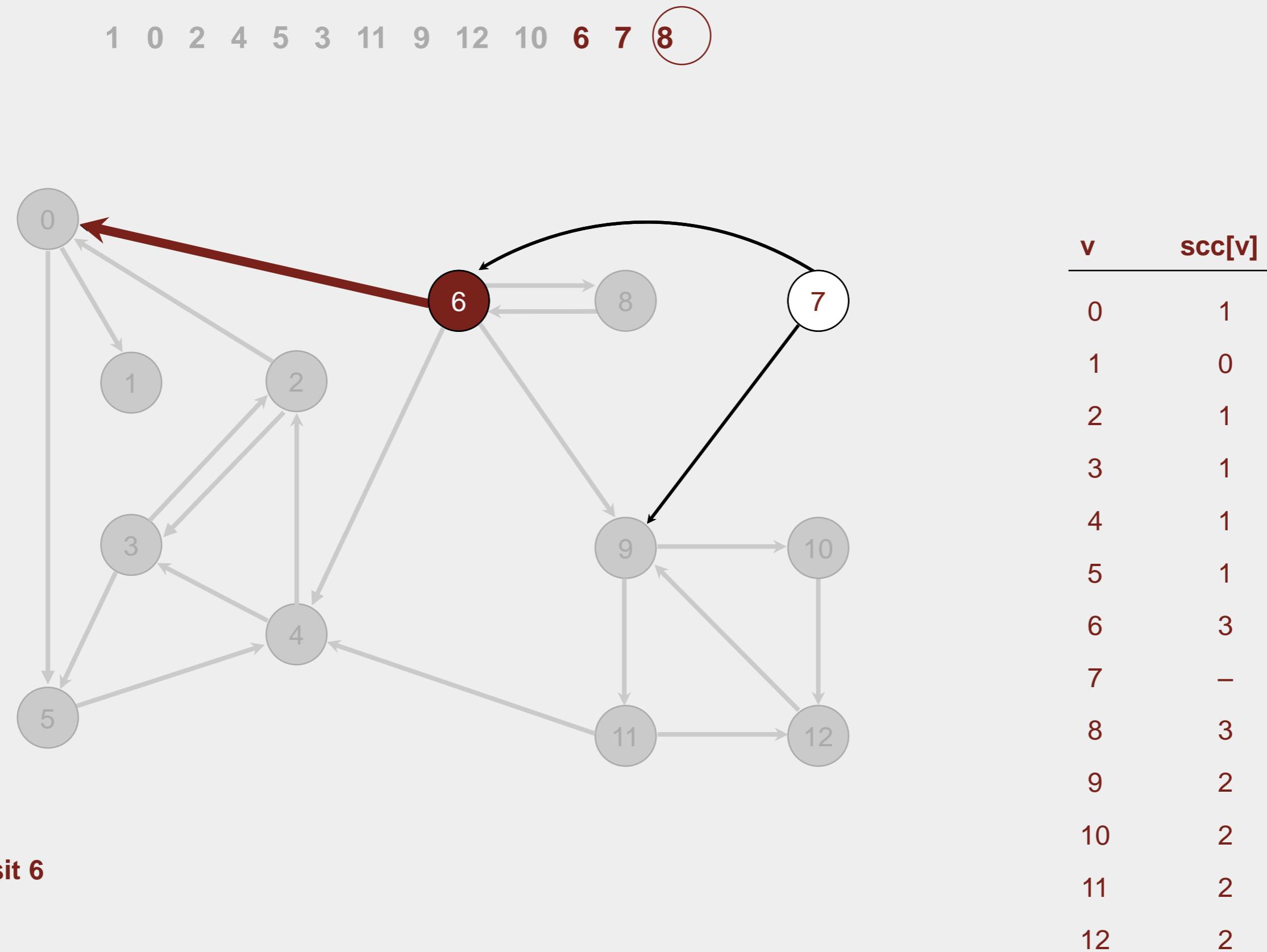
Phase 2. Run DFS in G , visiting unmarked vertices in reverse postorder of G^R .



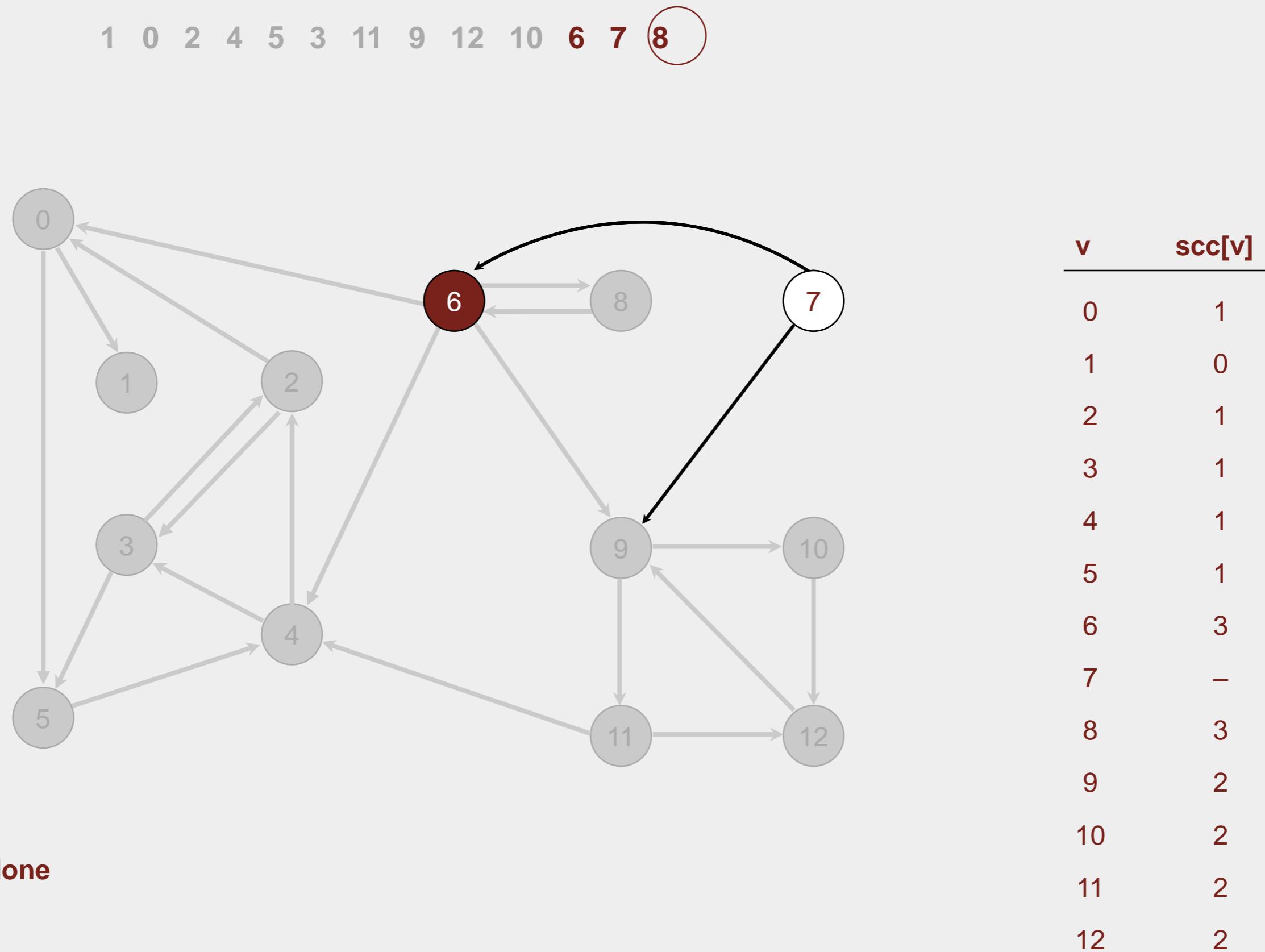
Phase 2. Run DFS in G , visiting unmarked vertices in reverse postorder of G^R .



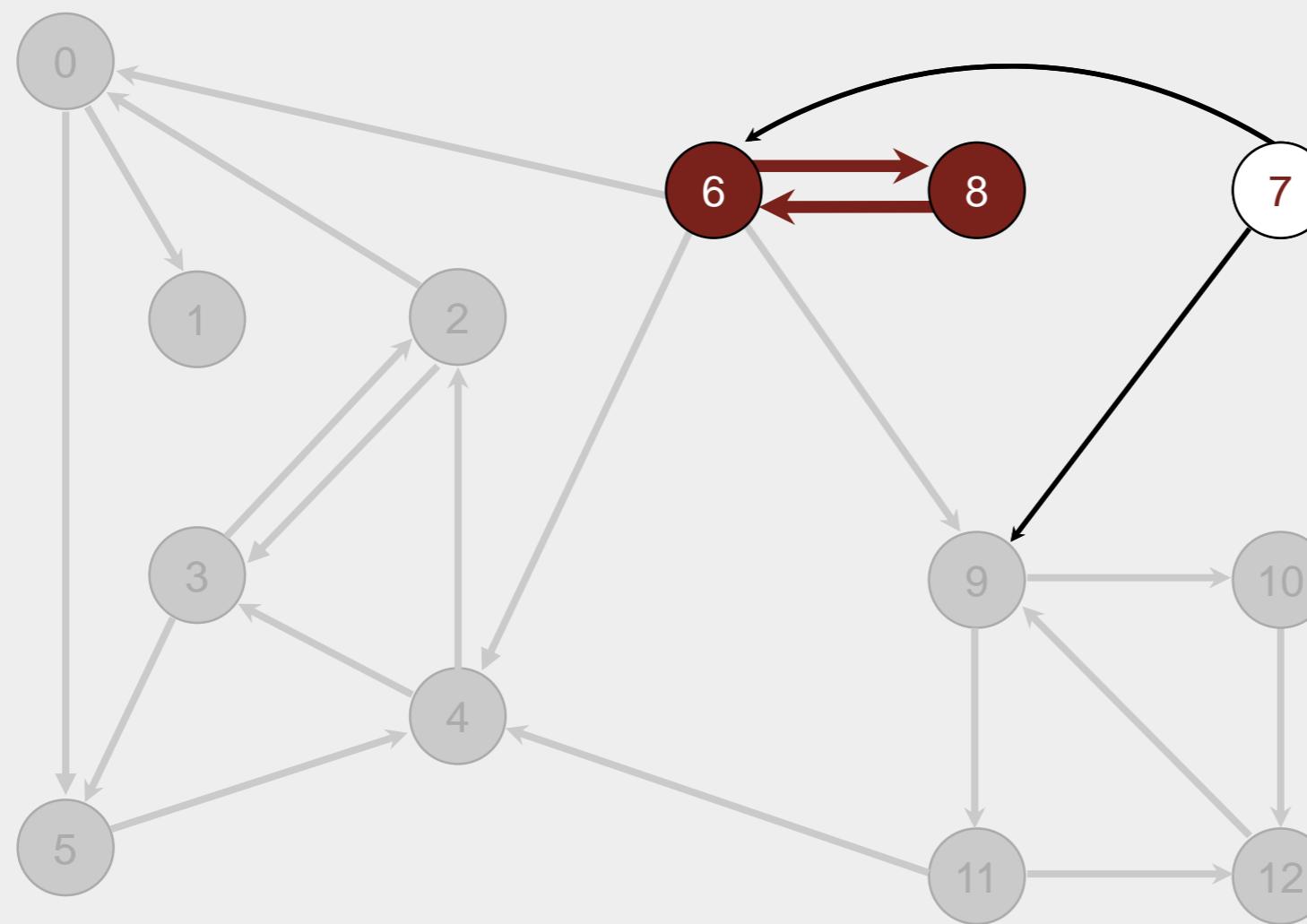
Phase 2. Run DFS in G , visiting unmarked vertices in reverse postorder of G^R .



Phase 2. Run DFS in G , visiting unmarked vertices in reverse postorder of G^R .



Phase 2. Run DFS in G , visiting unmarked vertices in reverse postorder of G^R .

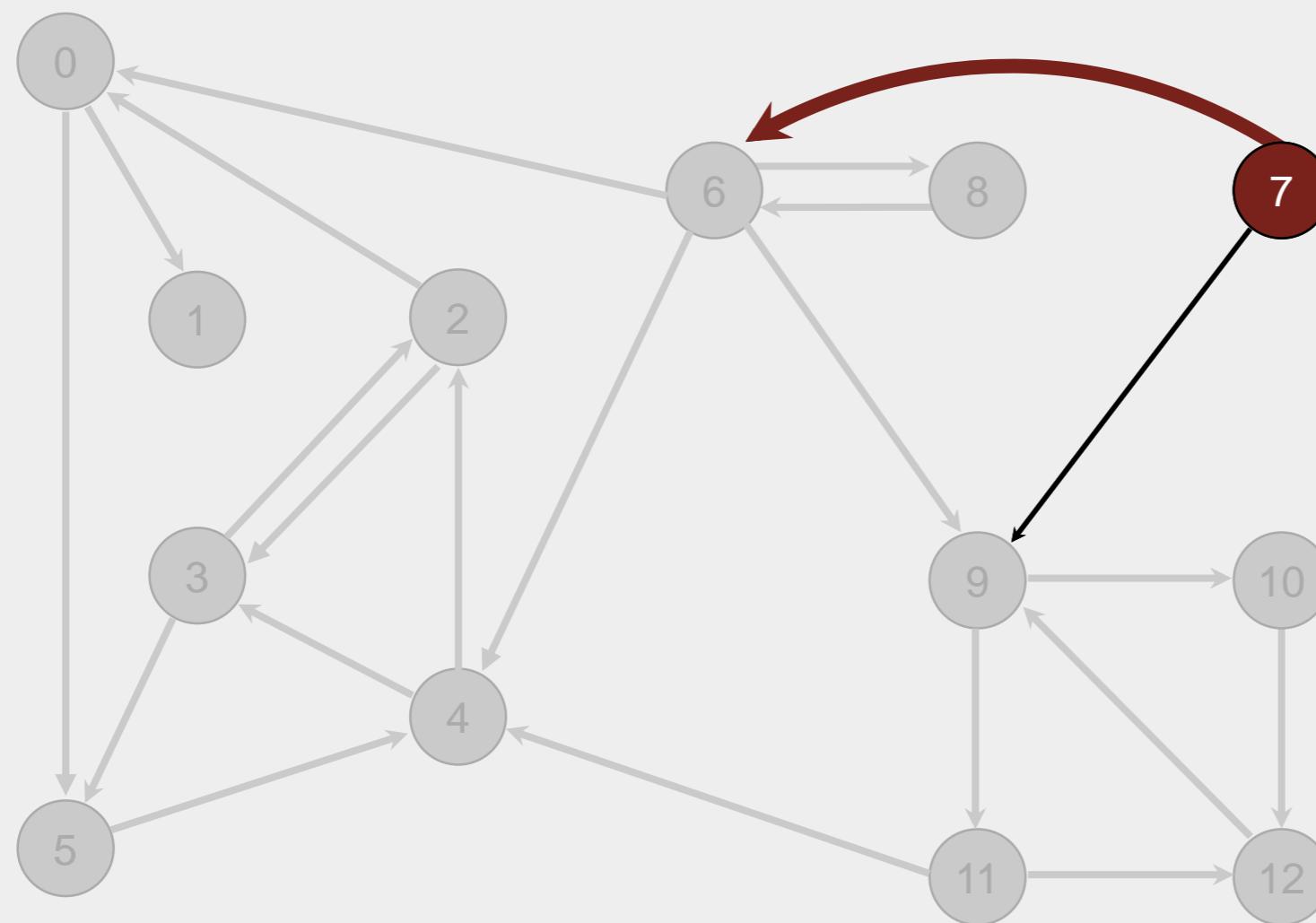


v	scc[v]
0	1
1	0
2	1
3	1
4	1
5	1
6	3
7	-
8	3
9	2
10	2
11	2
12	2

strong component: 6 8

Phase 2. Run DFS in G , visiting unmarked vertices in reverse postorder of G^R .

1 0 2 4 5 3 11 9 12 10 6 7 8



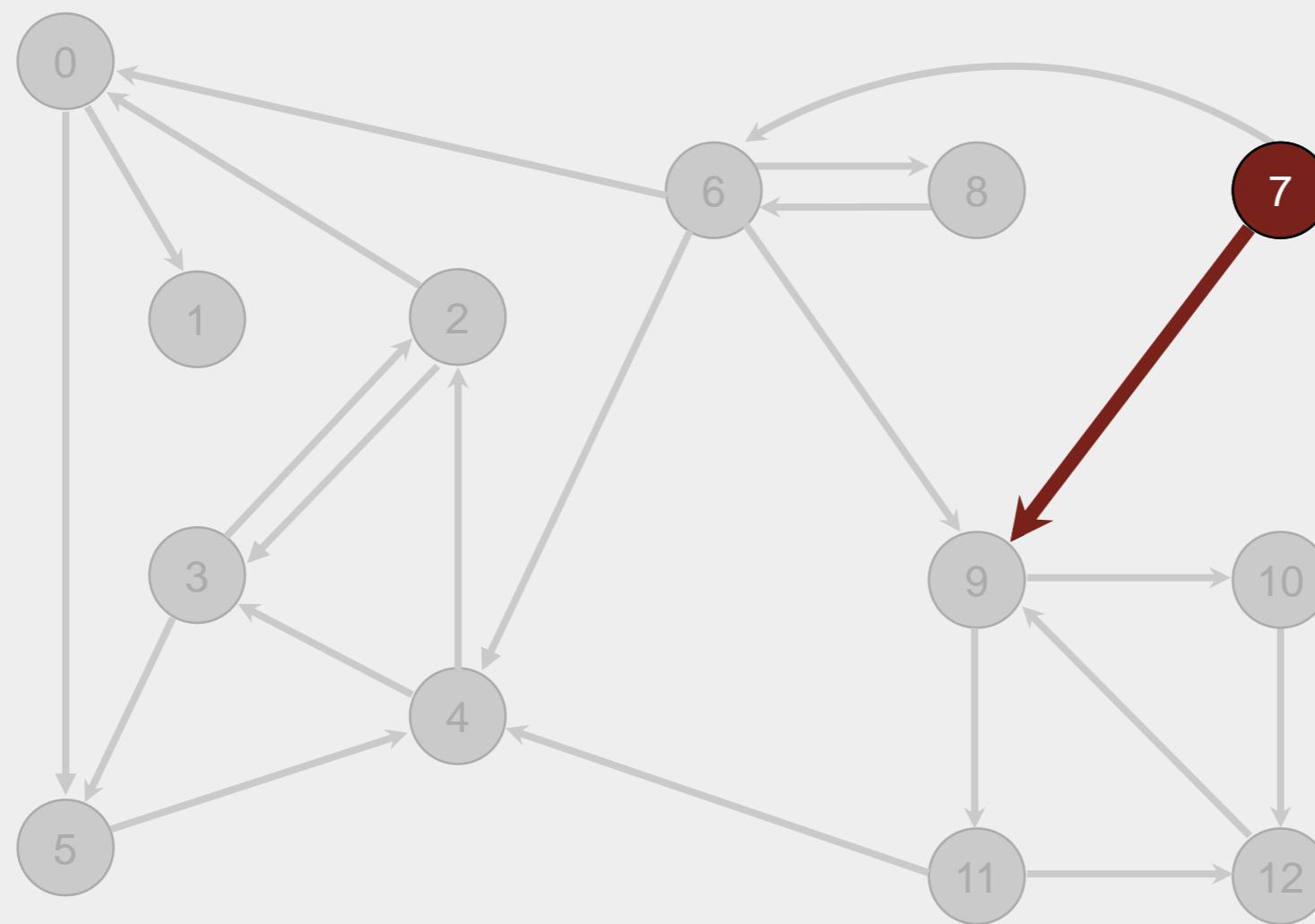
visit 7

v	$scc[v]$
0	1
1	0
2	1
3	1
4	1
5	1
6	3
7	4
8	3
9	2
10	2
11	2
12	2

Phase 2. Run DFS in G , visiting unmarked vertices in reverse postorder of G^R .

1 0 2 4 5 3 11 9 12 10 6 7 8

○



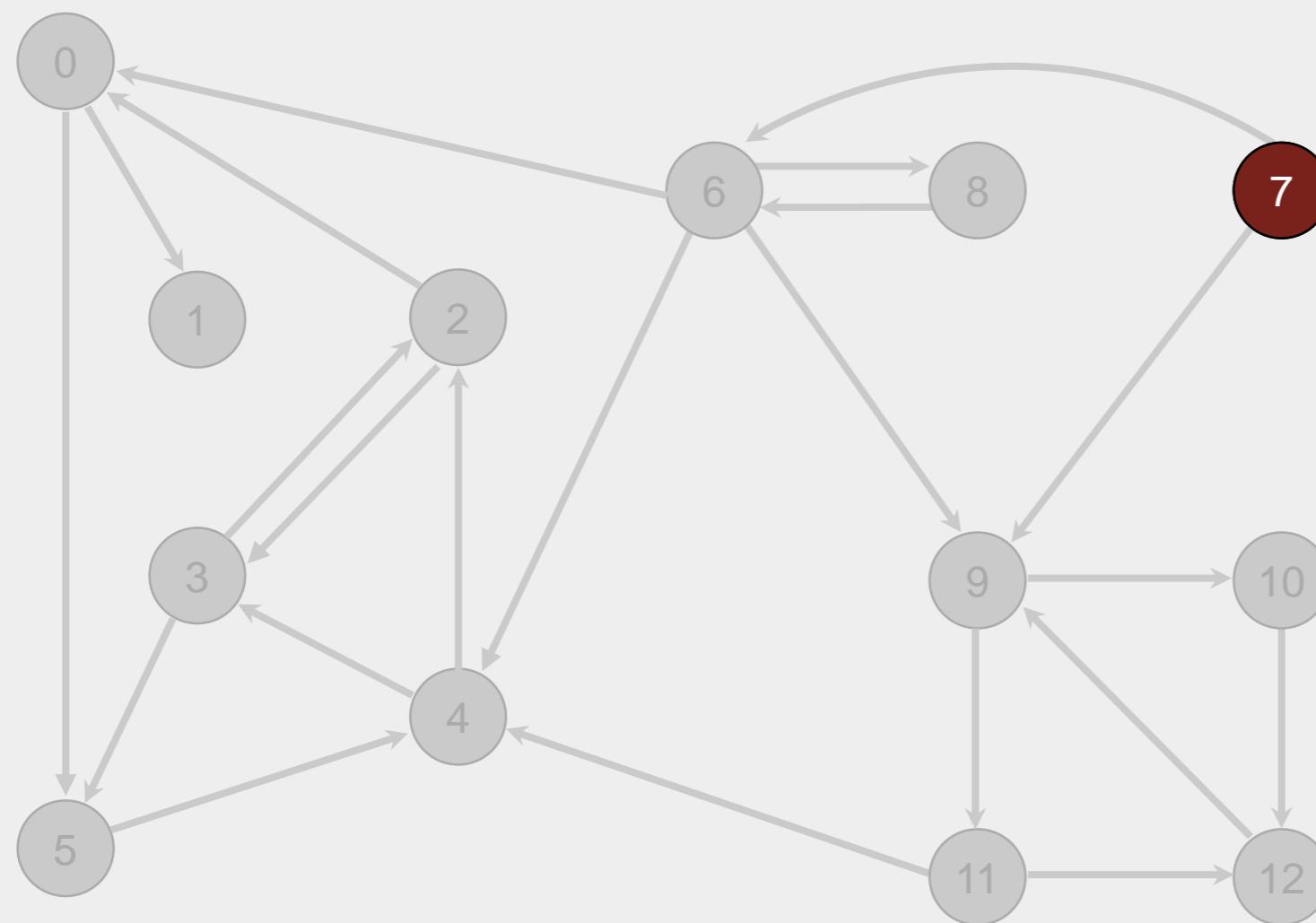
v scc[v]

v	scc[v]
0	1
1	0
2	1
3	1
4	1
5	1
6	3
7	4
8	3
9	2
10	2
11	2
12	2

visit 7

Phase 2. Run DFS in G , visiting unmarked vertices in reverse postorder of G^R .

1 0 2 4 5 3 11 9 12 10 6 7 8



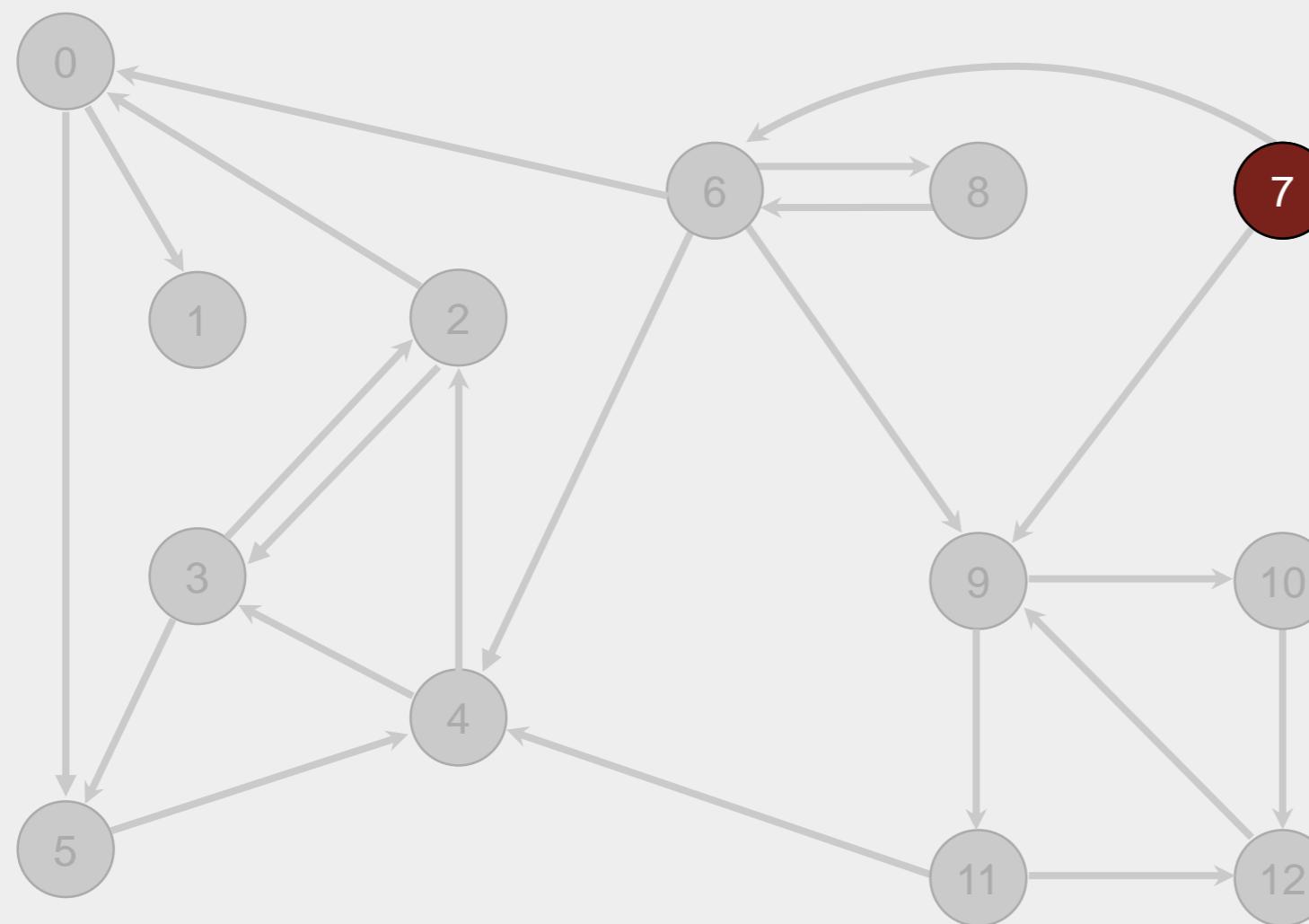
v **scc[v]**

v	scc[v]
0	1
1	0
2	1
3	1
4	1
5	1
6	3
7	4
8	3
9	2
10	2
11	2
12	2

7 done

Phase 2. Run DFS in G , visiting unmarked vertices in reverse postorder of G^R .

1 0 2 4 5 3 11 9 12 10 6 7 8



v **scc[v]**

0 1

1 0

2 1

3 1

4 1

5 1

6 1

7 4

8 3

9 2

10 2

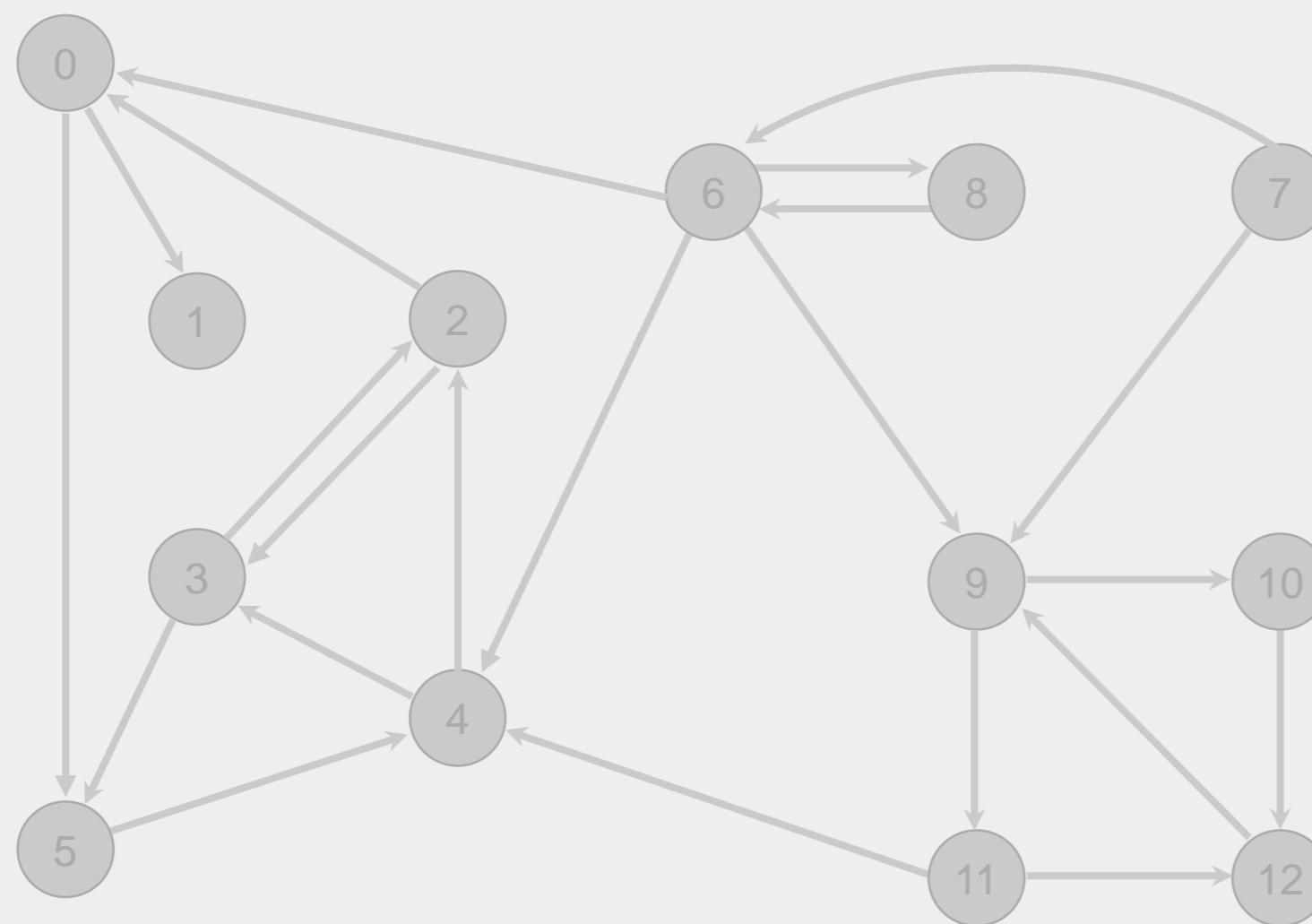
11 2

12 2

strong component: 7

Phase 2. Run DFS in G , visiting unmarked vertices in reverse postorder of G^R .

1 0 2 4 5 3 11 9 12 10 6 7 8



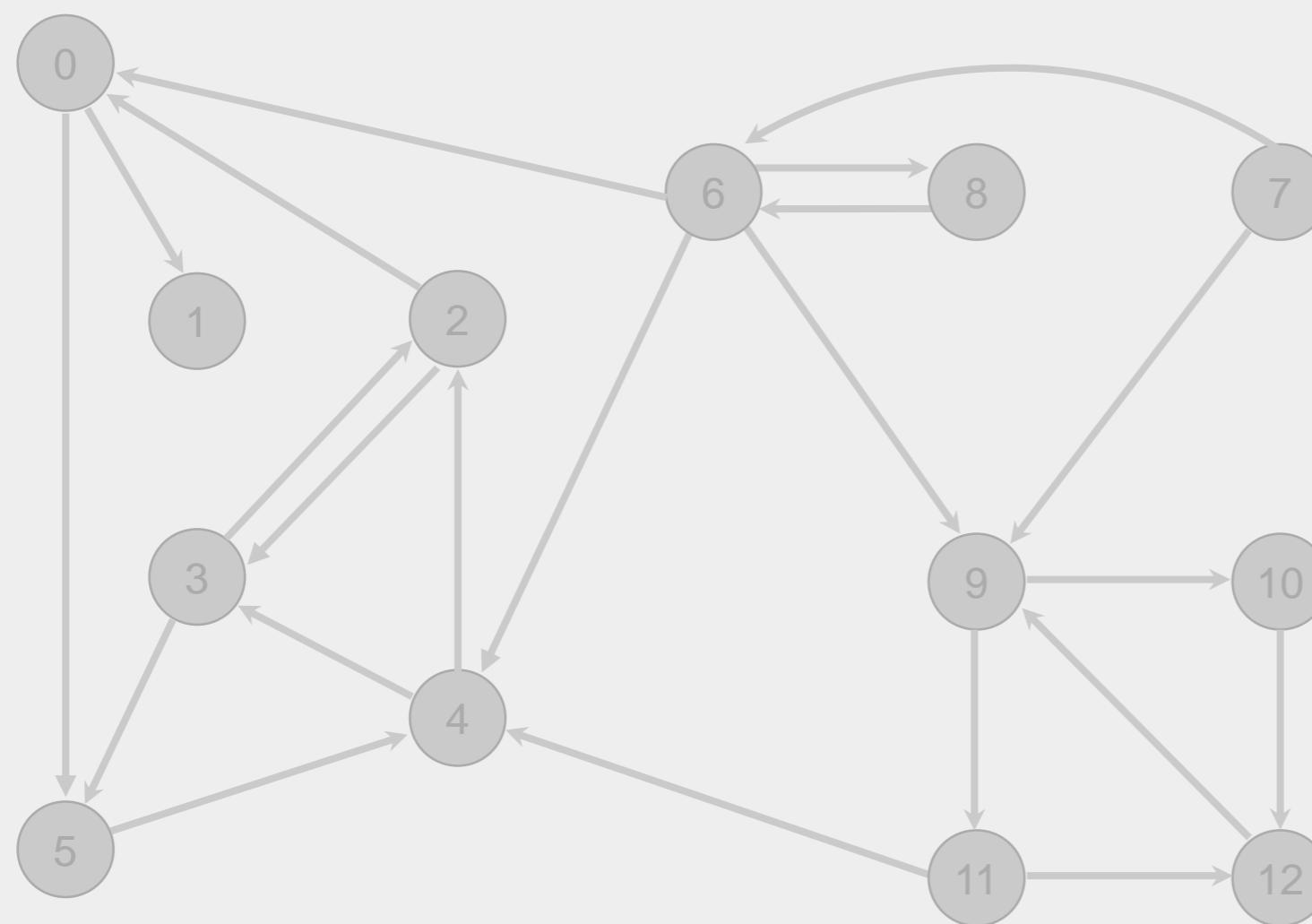
$v \quad scc[v]$

v	$scc[v]$
0	1
1	0
2	1
3	1
4	1
5	1
6	3
7	4
8	3
9	2
10	2
11	2
12	2

check 8

Phase 2. Run DFS in G , visiting unmarked vertices in reverse postorder of G^R .

1 0 2 4 5 3 11 9 12 10 6 7 8



v scc[v]

v	scc[v]
0	1
1	0
2	1
3	1
4	1
5	1
6	3
7	4
8	3
9	2
10	2
11	2
12	2

done

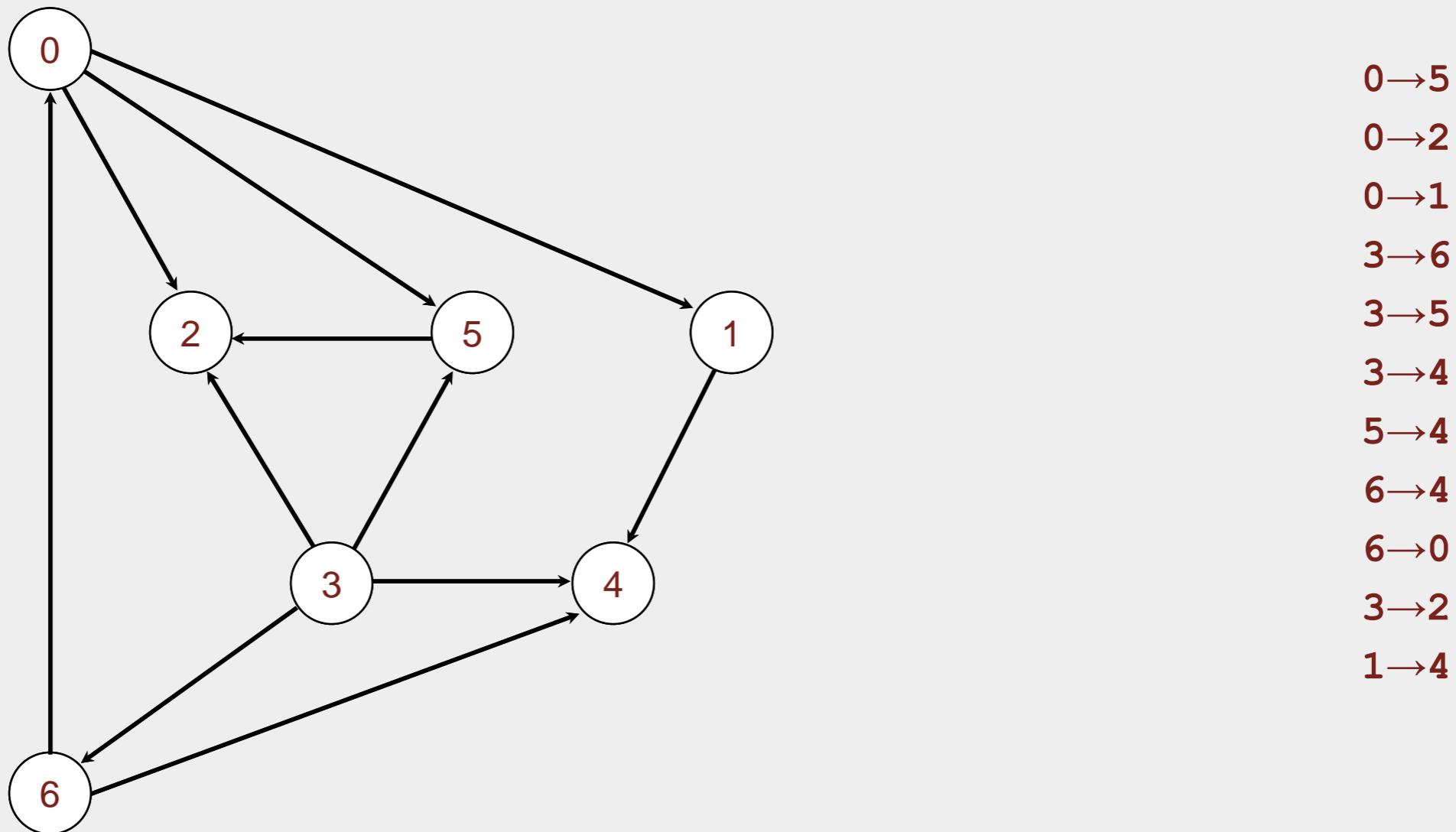
4.2 Topological Sort demo



click to begin demo

Topological sort algorithm

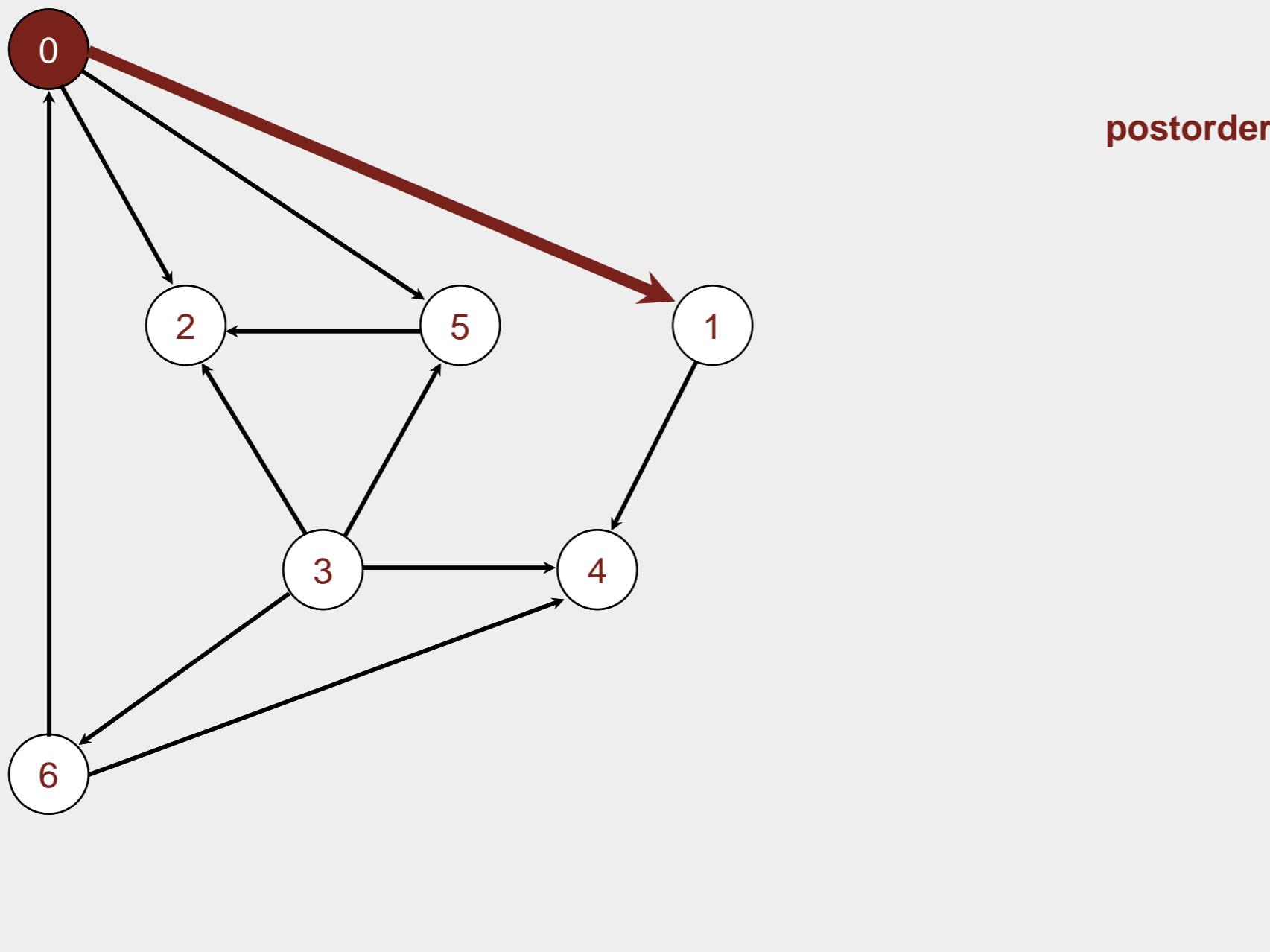
- Run depth-first search.
- Return vertices in reverse postorder.



a directed acyclic graph

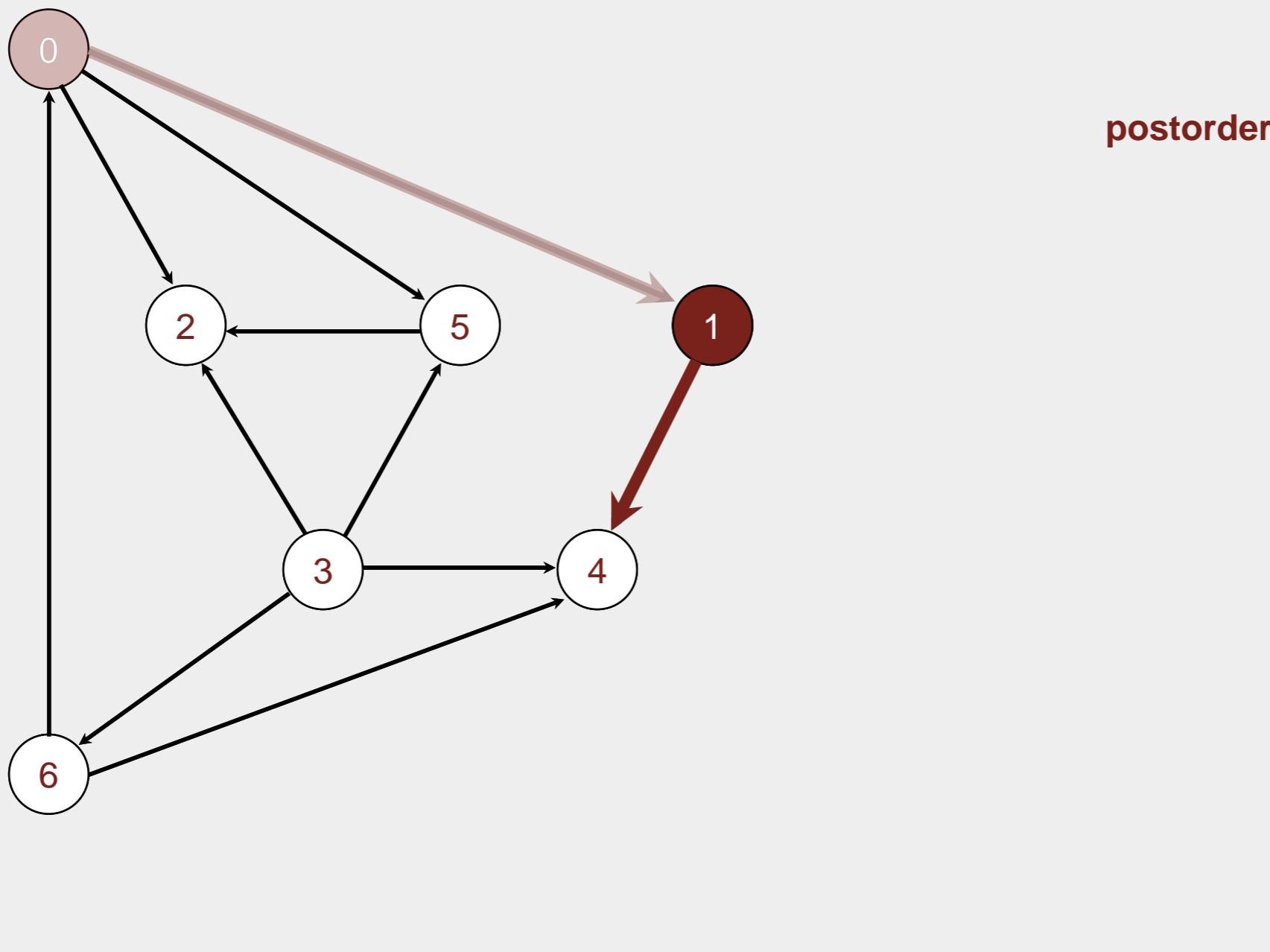
Topological sort algorithm

- Run depth-first search.
- Return vertices in reverse postorder.



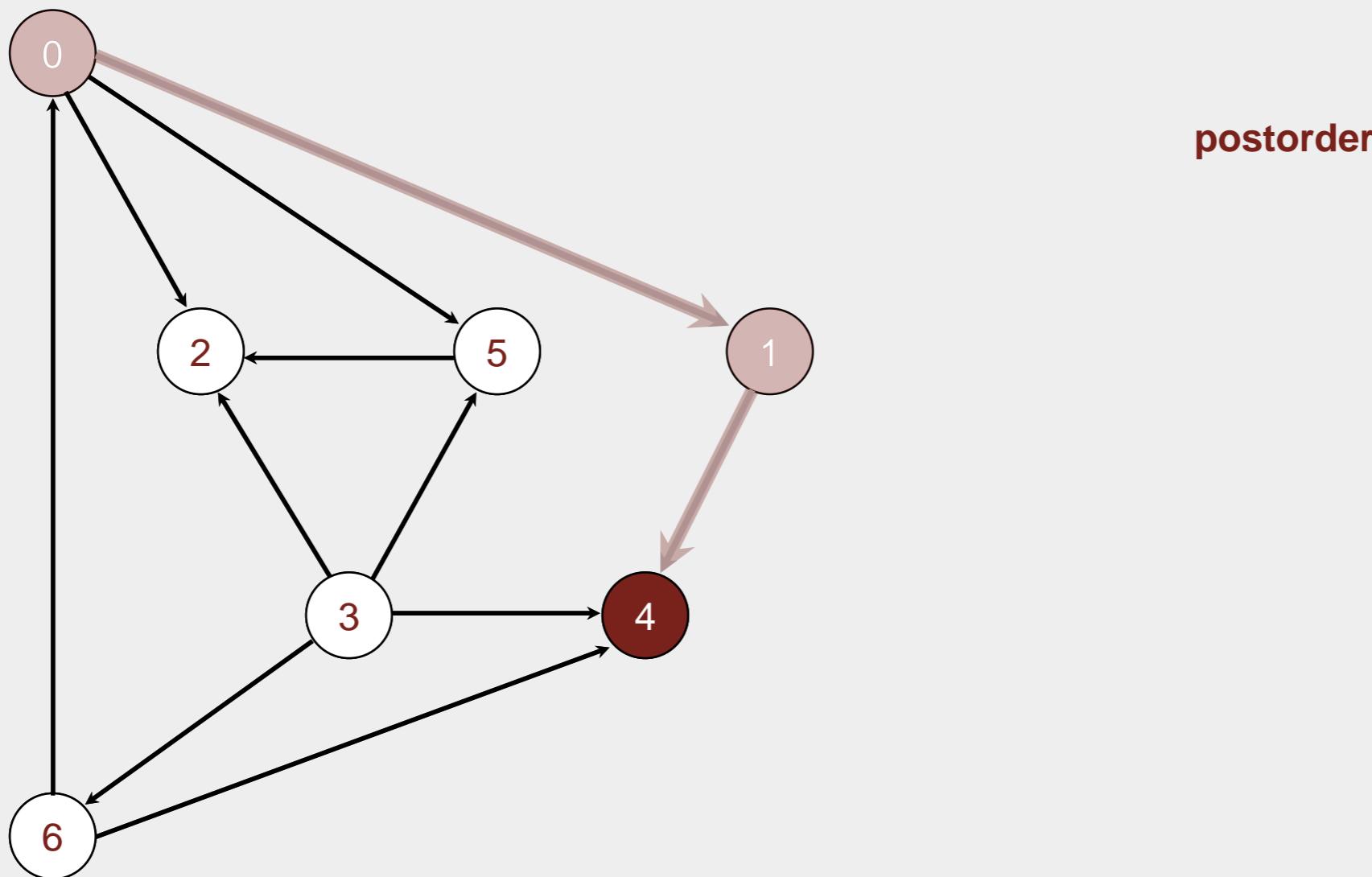
Topological sort algorithm

- Run depth-first search.
- Return vertices in reverse postorder.



Topological sort algorithm

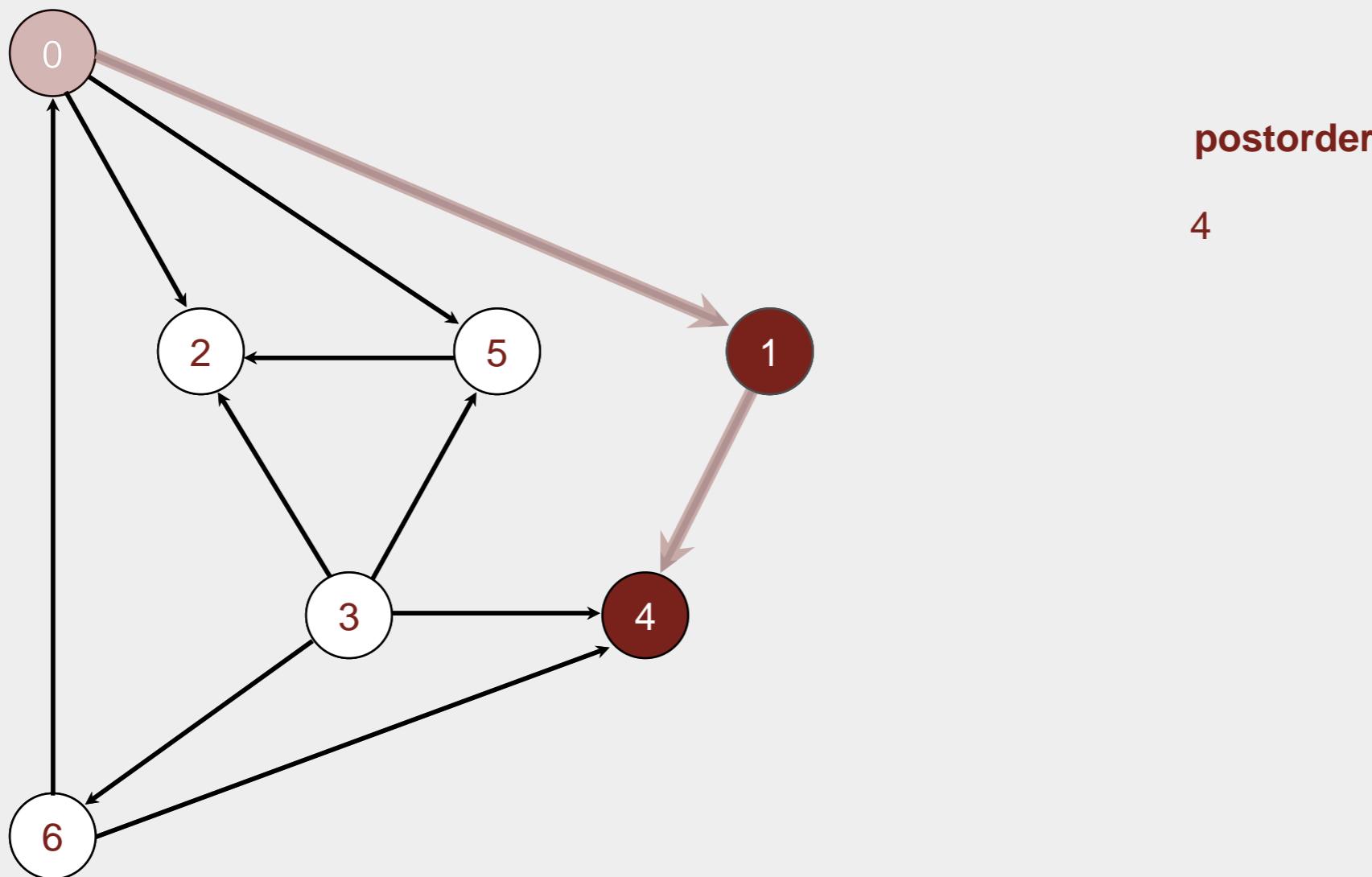
- Run depth-first search.
- Return vertices in reverse postorder.



visit 4

Topological sort algorithm

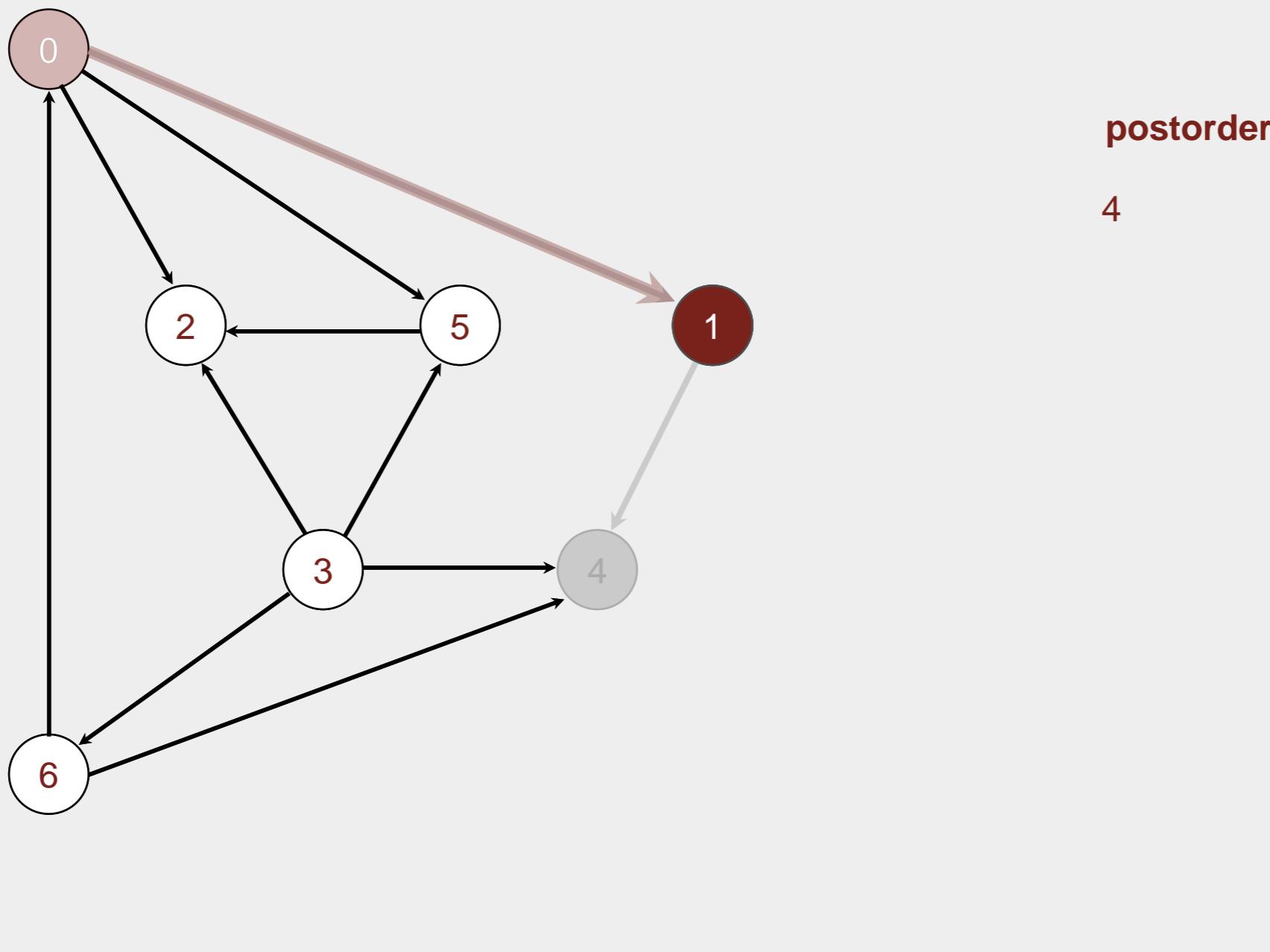
- Run depth-first search.
- Return vertices in reverse postorder.



4 done

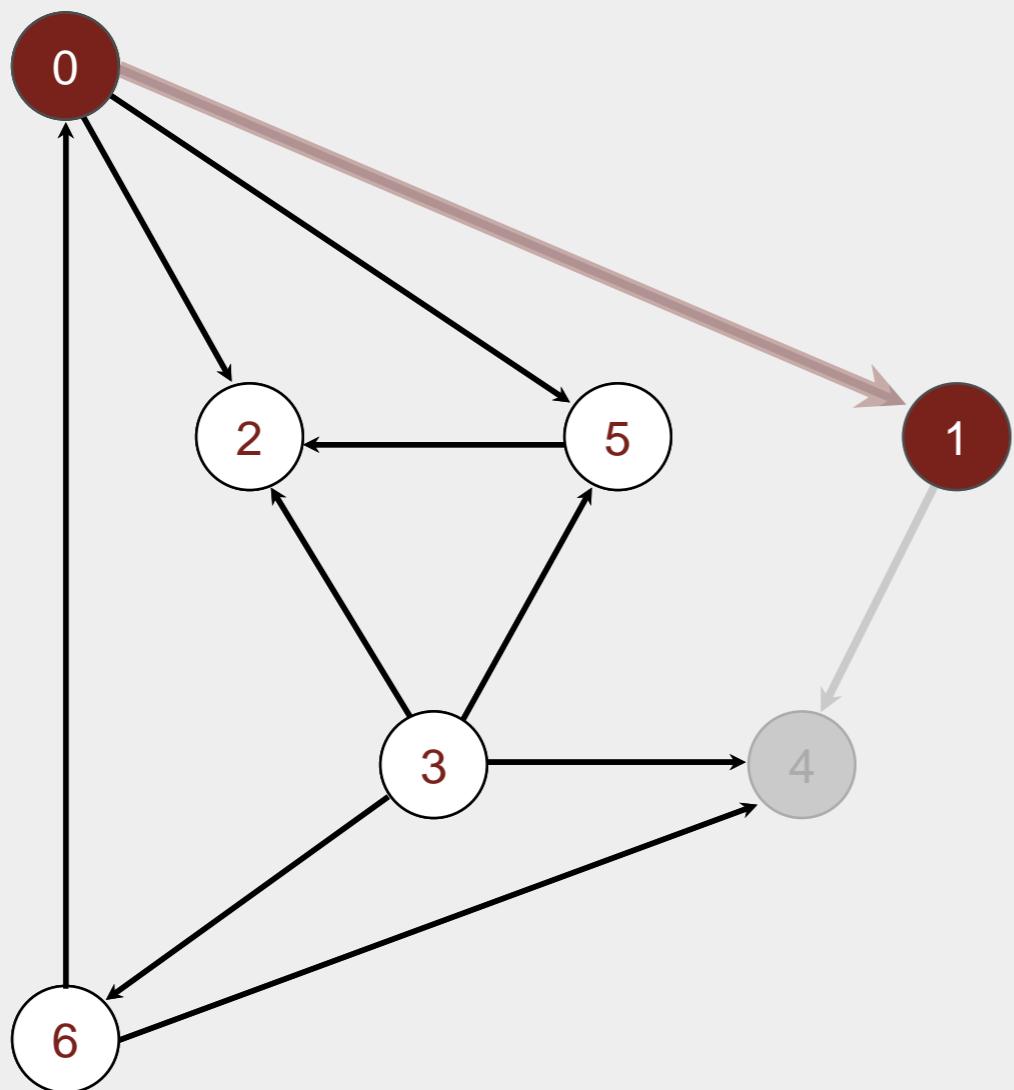
Topological sort algorithm

- Run depth-first search.
- Return vertices in reverse postorder.



Topological sort algorithm

- Run depth-first search.
- Return vertices in reverse postorder.



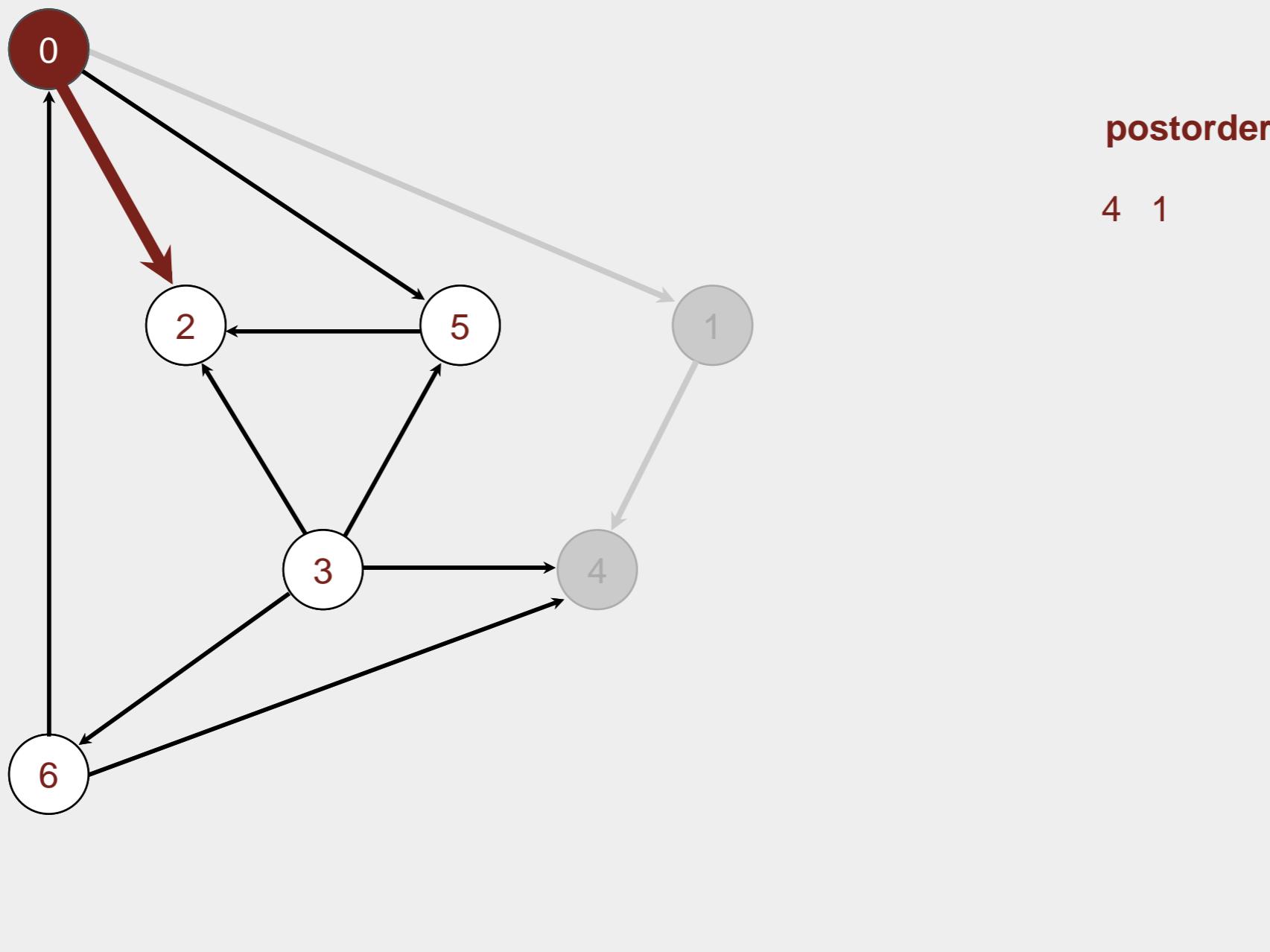
postorder

4 1

1 done

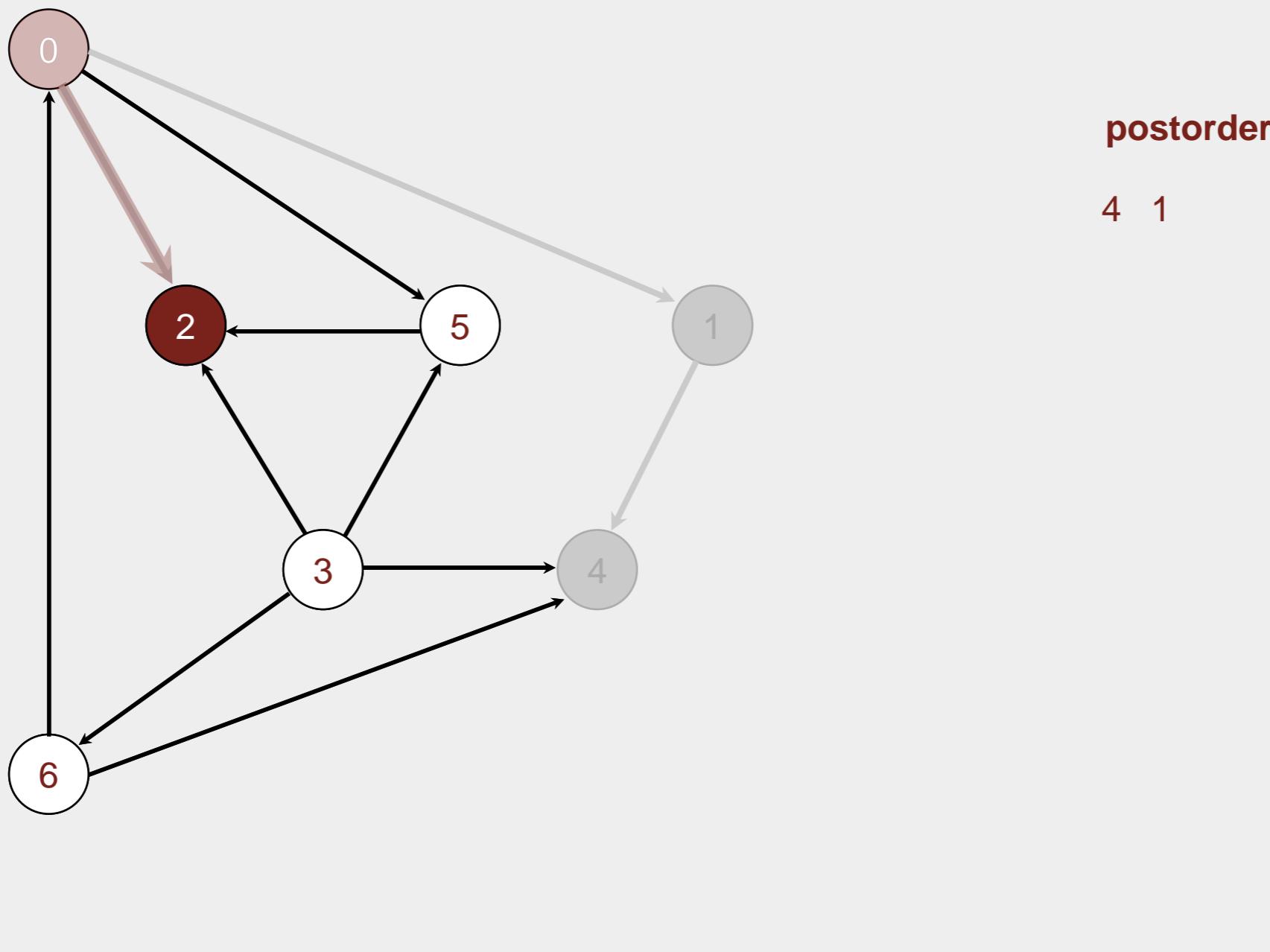
Topological sort algorithm

- Run depth-first search.
- Return vertices in reverse postorder.



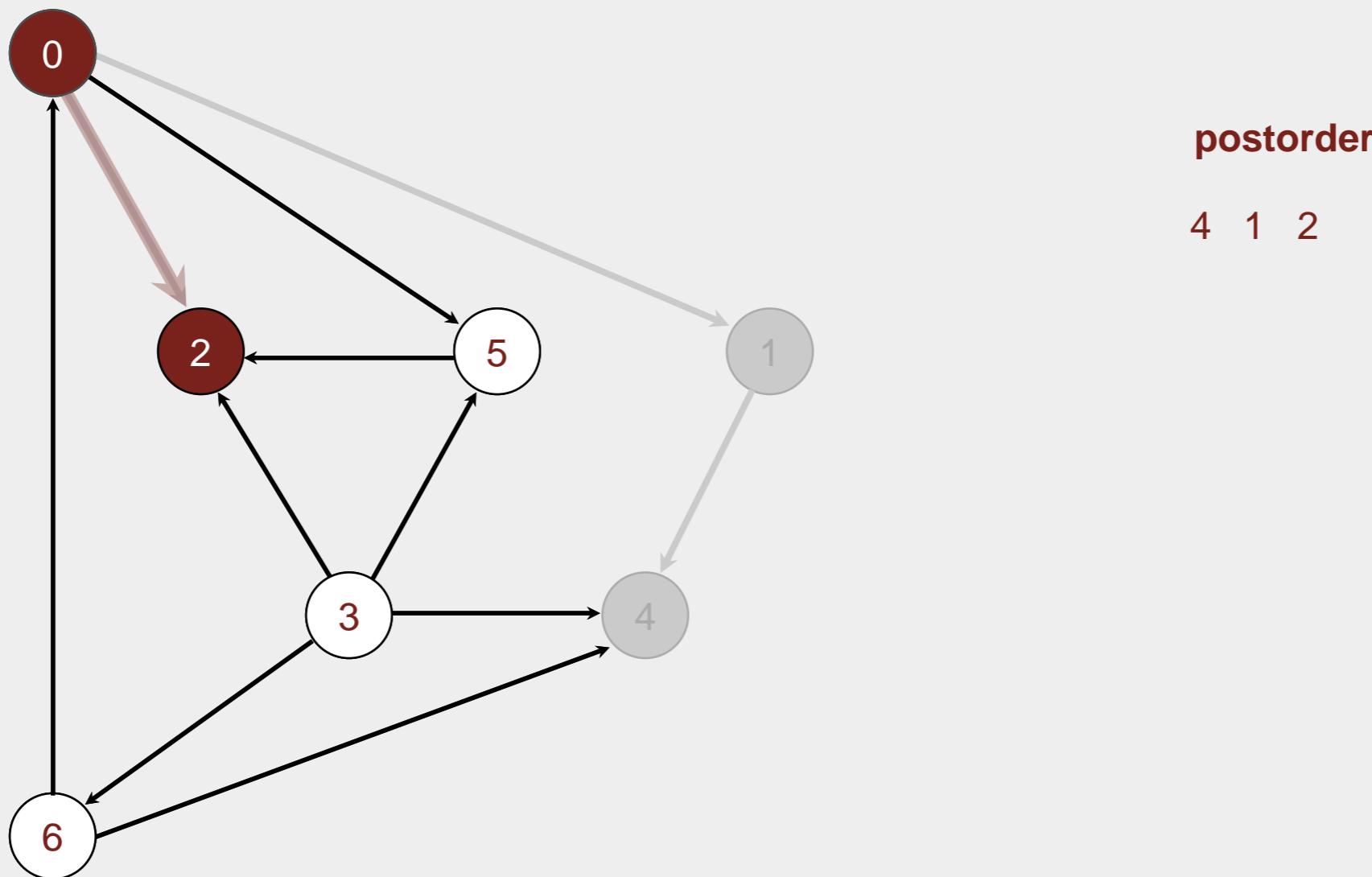
Topological sort algorithm

- Run depth-first search.
- Return vertices in reverse postorder.



Topological sort algorithm

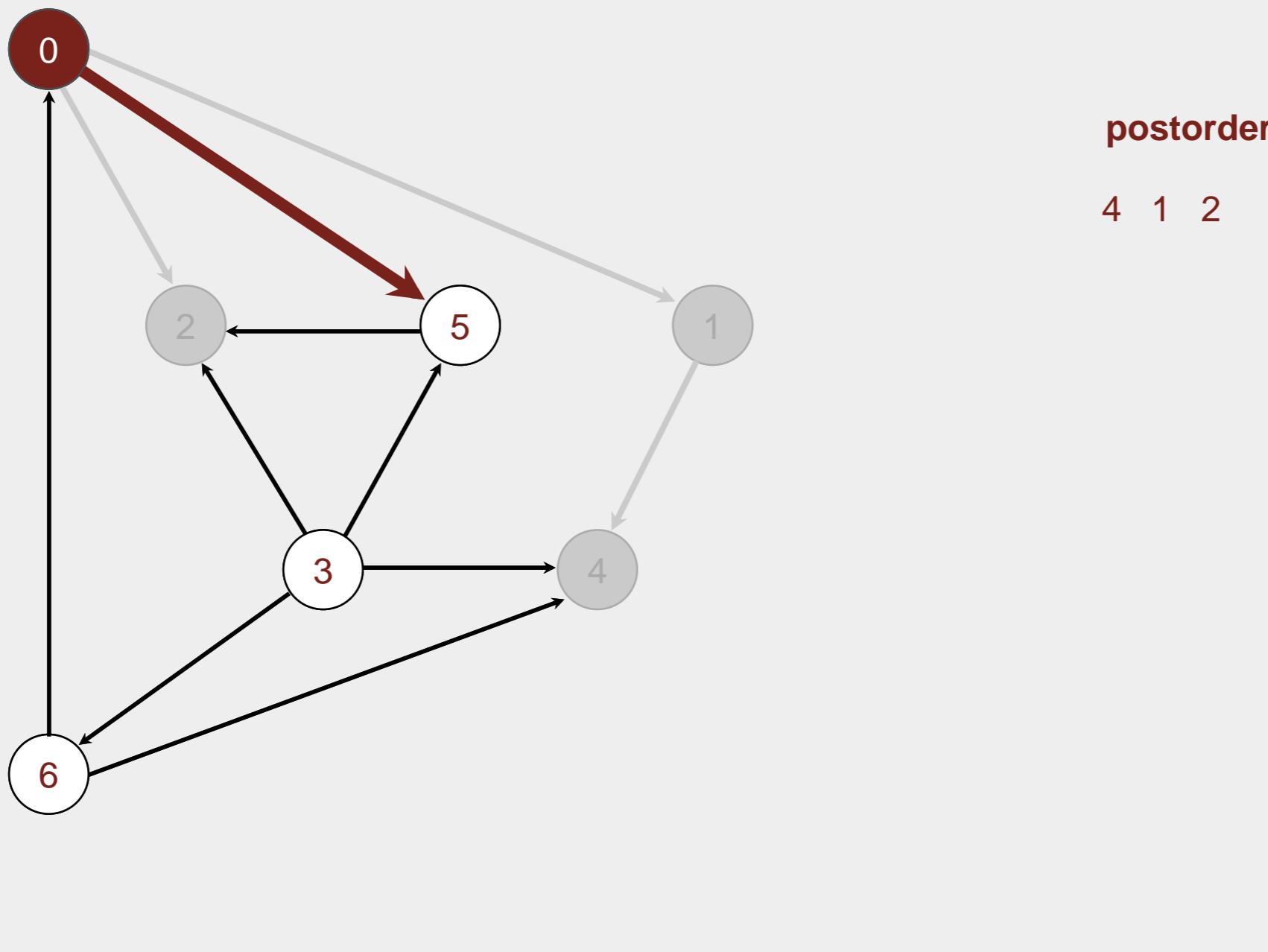
- Run depth-first search.
- Return vertices in reverse postorder.



2 done

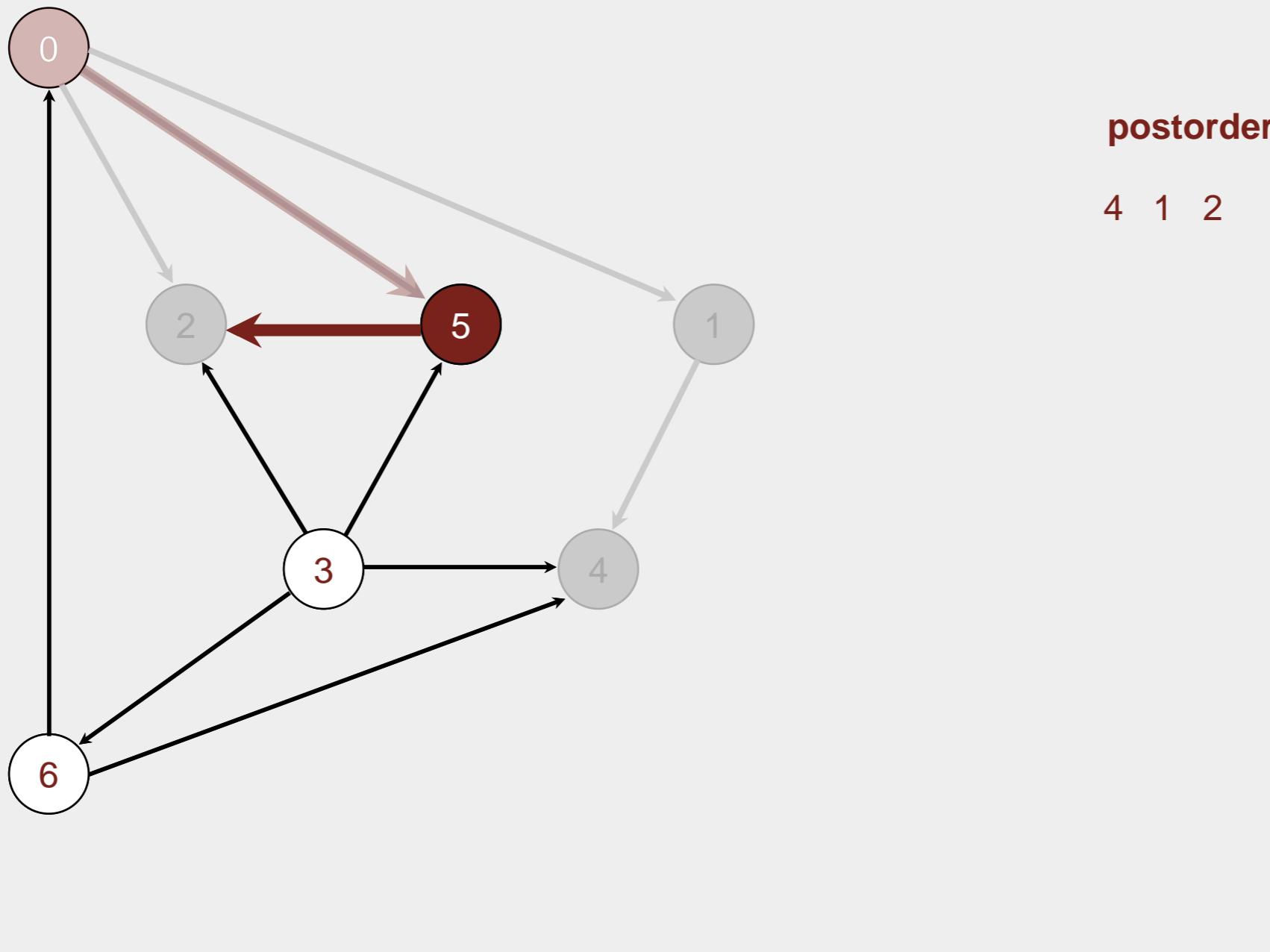
Topological sort algorithm

- Run depth-first search.
- Return vertices in reverse postorder.



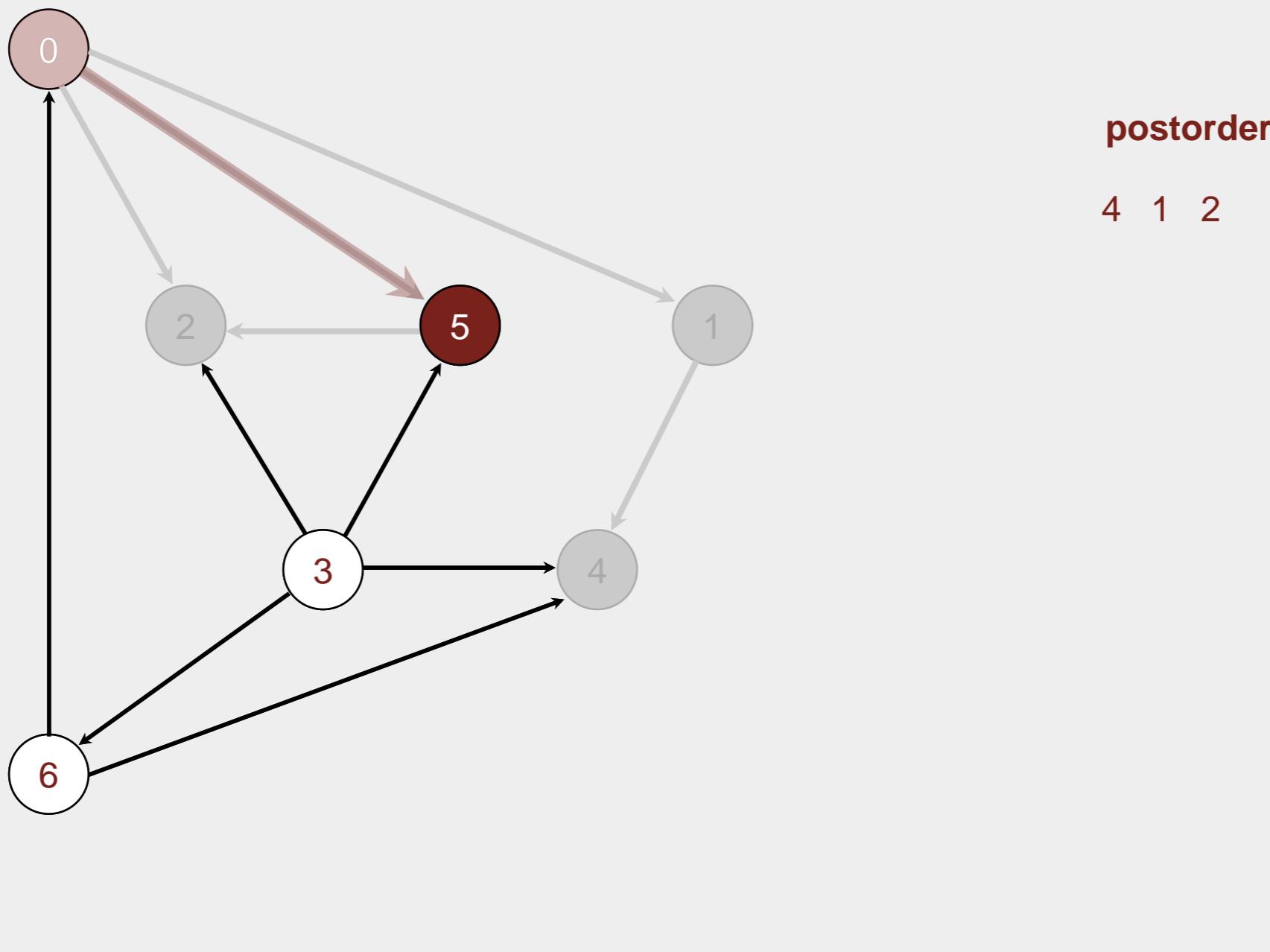
Topological sort algorithm

- Run depth-first search.
- Return vertices in reverse postorder.



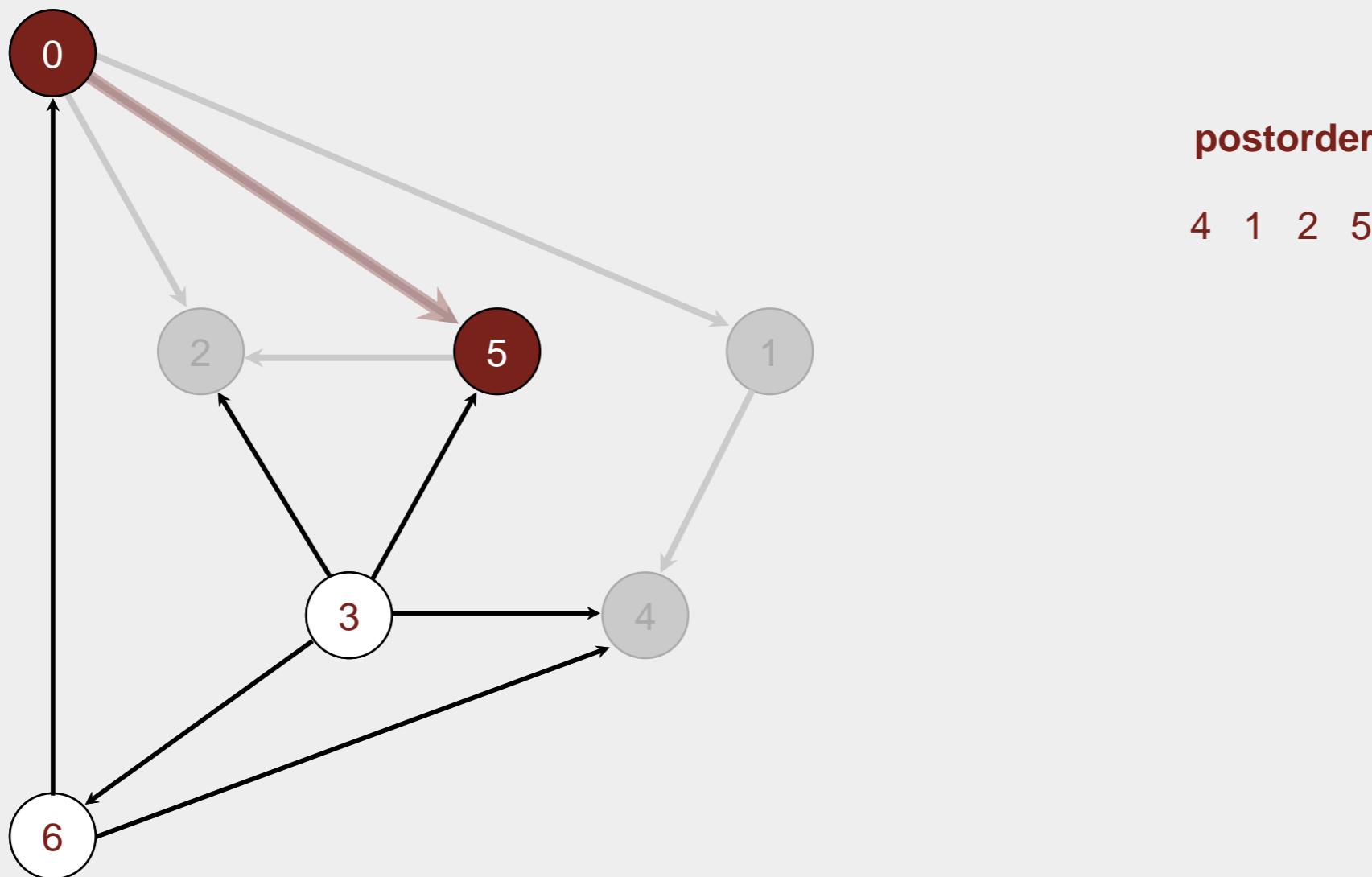
Topological sort algorithm

- Run depth-first search.
- Return vertices in reverse postorder.



Topological sort algorithm

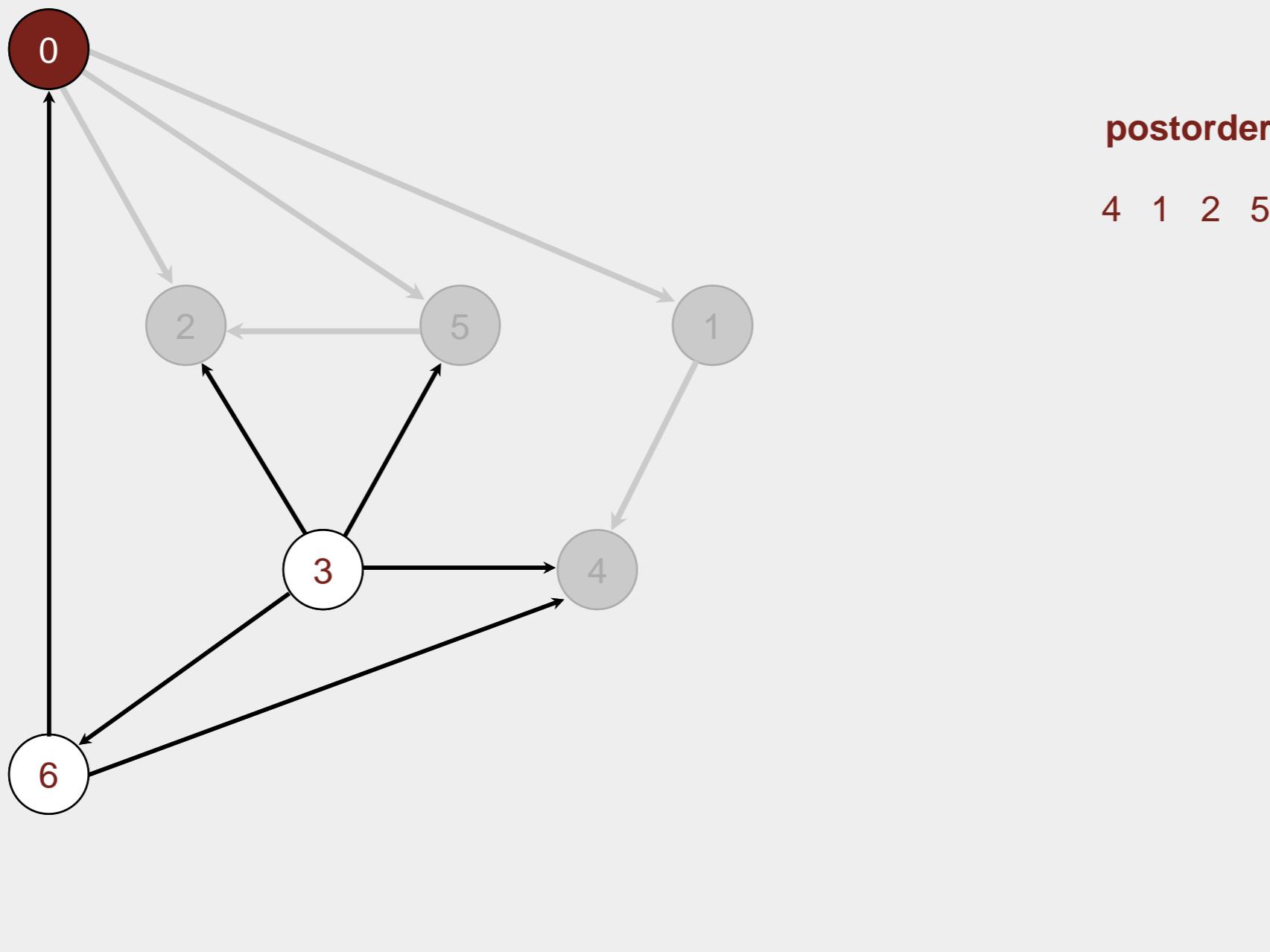
- Run depth-first search.
- Return vertices in reverse postorder.



5 done

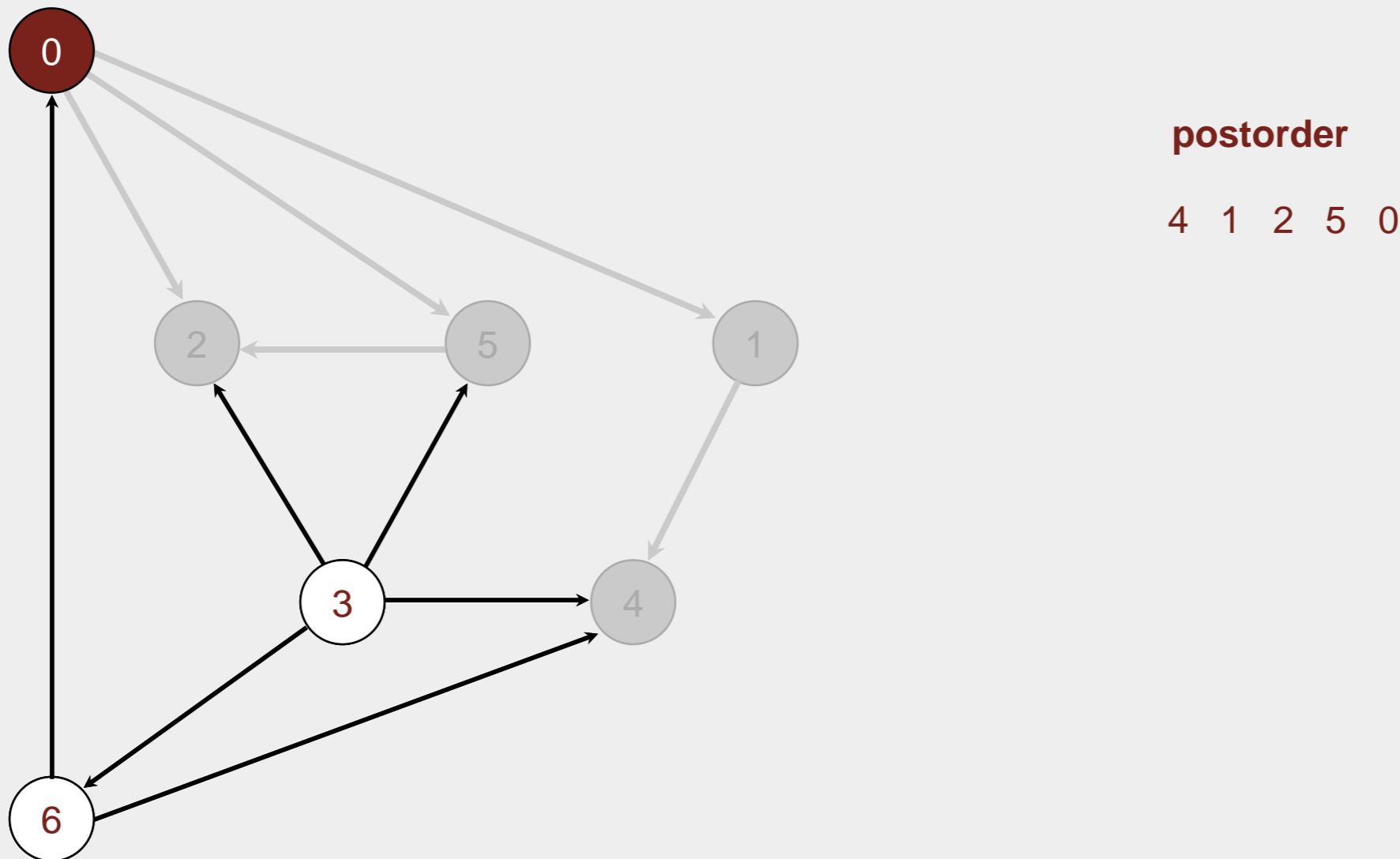
Topological sort algorithm

- Run depth-first search.
- Return vertices in reverse postorder.



Topological sort algorithm

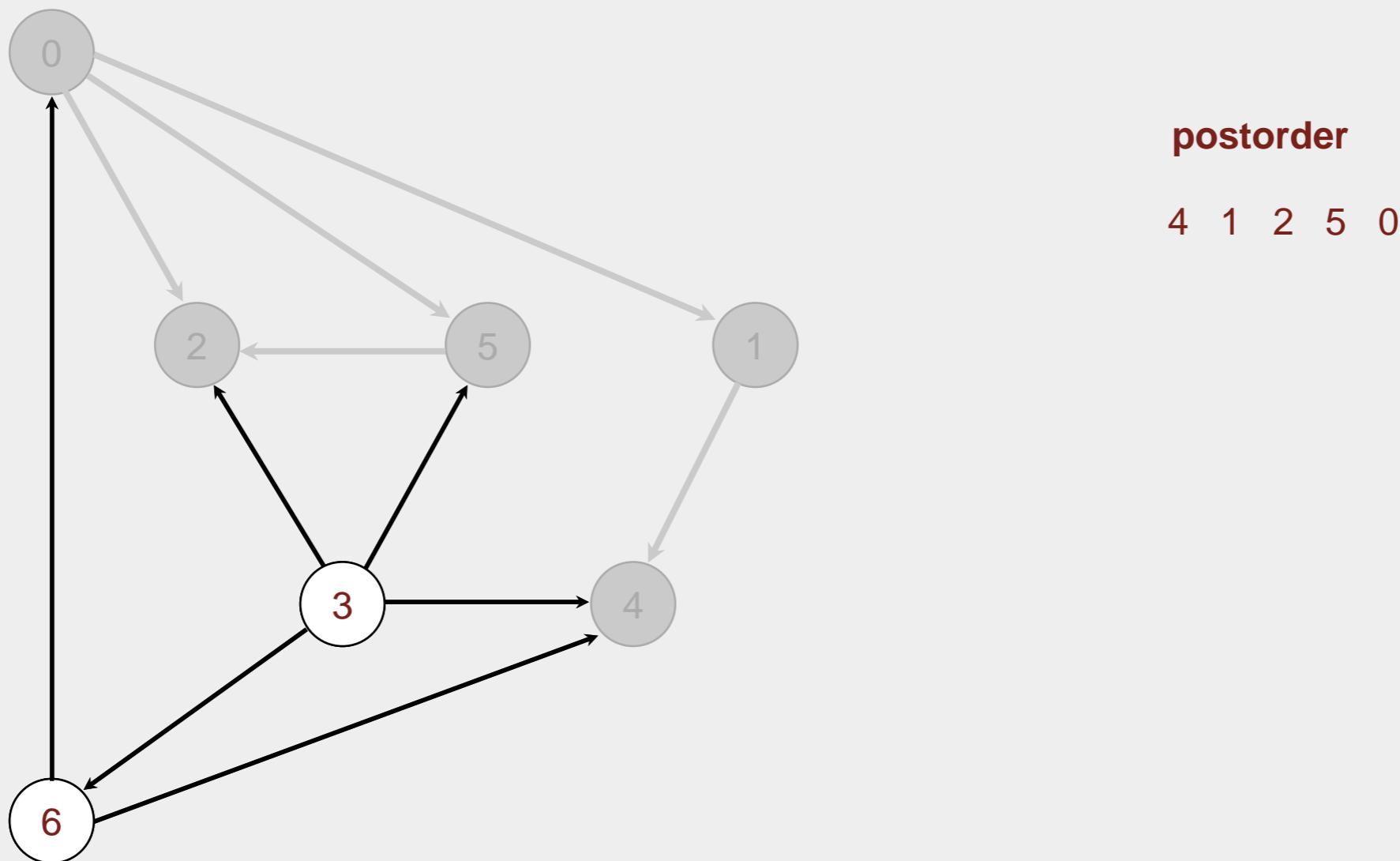
- Run depth-first search.
- Return vertices in reverse postorder.



0 done

Topological sort algorithm

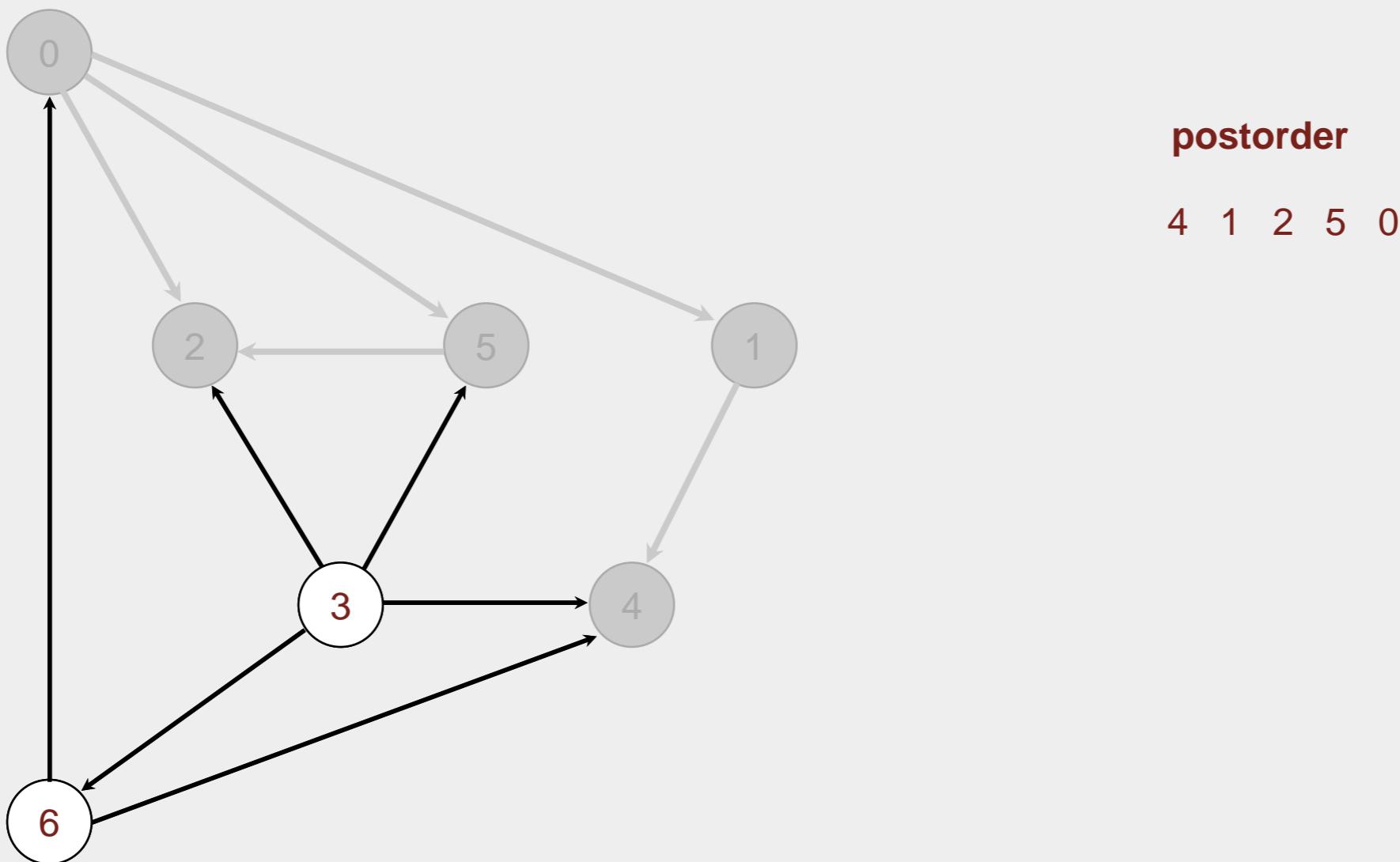
- Run depth-first search.
- Return vertices in reverse postorder.



check 1

Topological sort algorithm

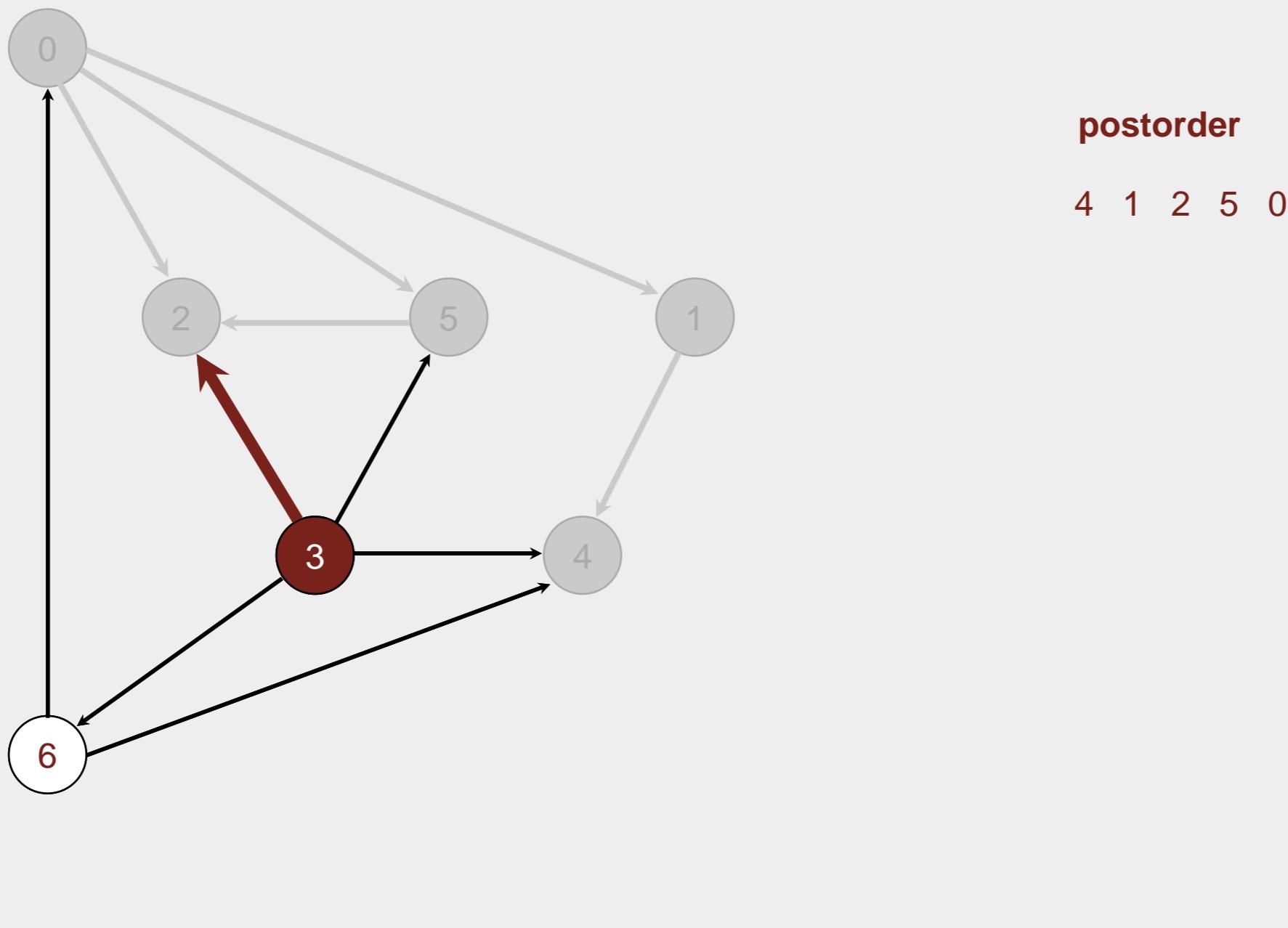
- Run depth-first search.
- Return vertices in reverse postorder.



check 2

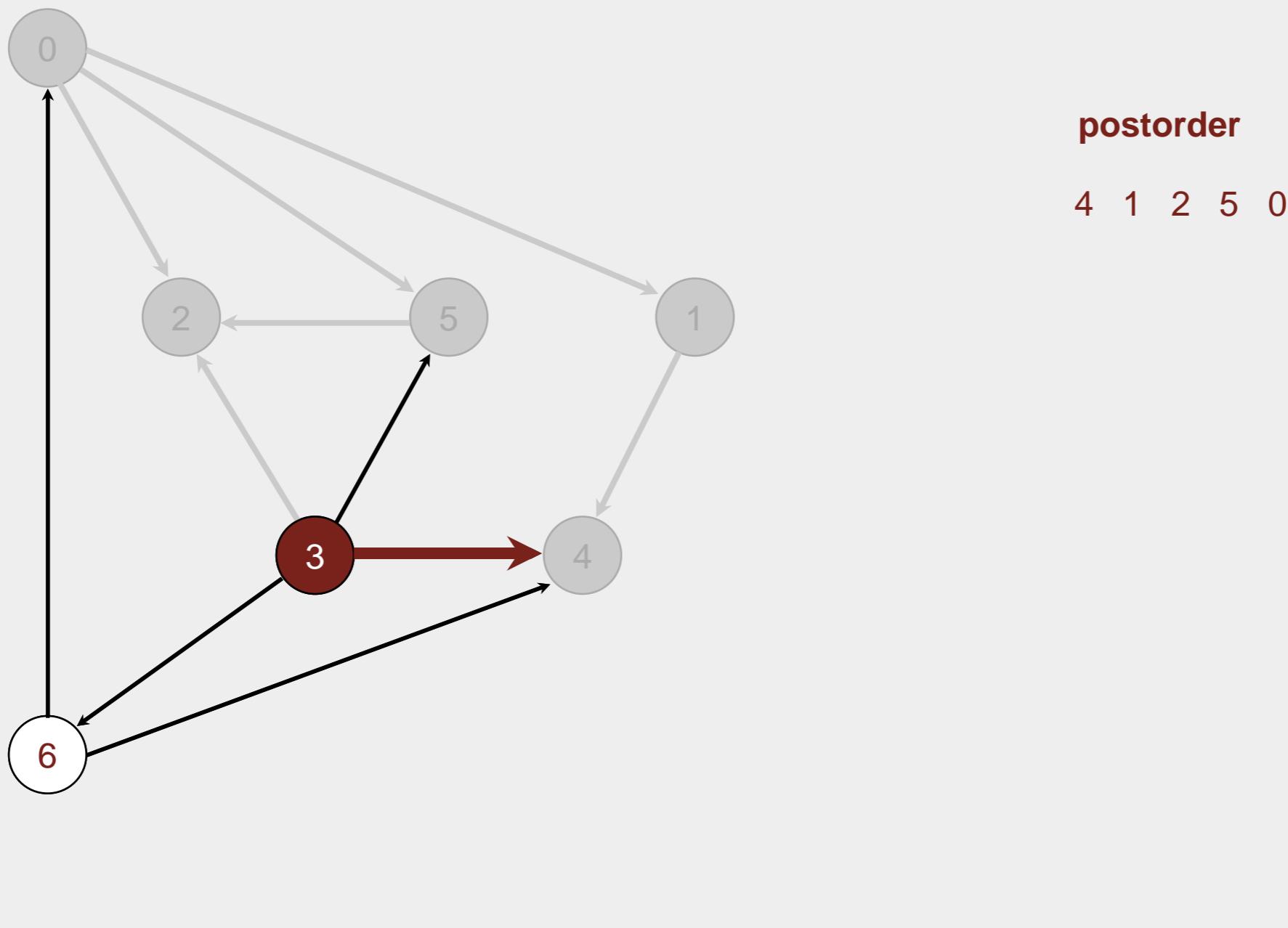
Topological sort algorithm

- Run depth-first search.
- Return vertices in reverse postorder.



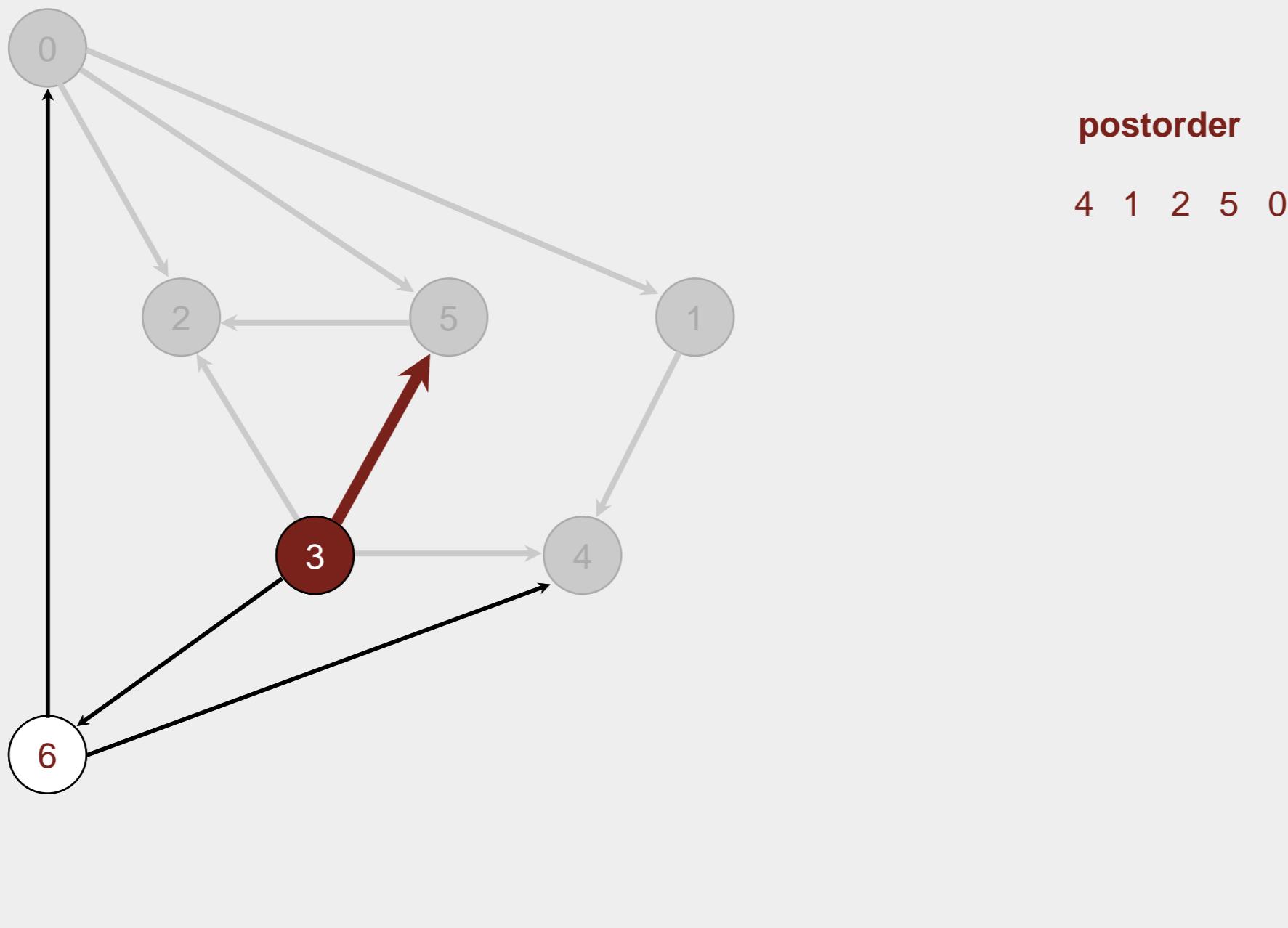
Topological sort algorithm

- Run depth-first search.
- Return vertices in reverse postorder.



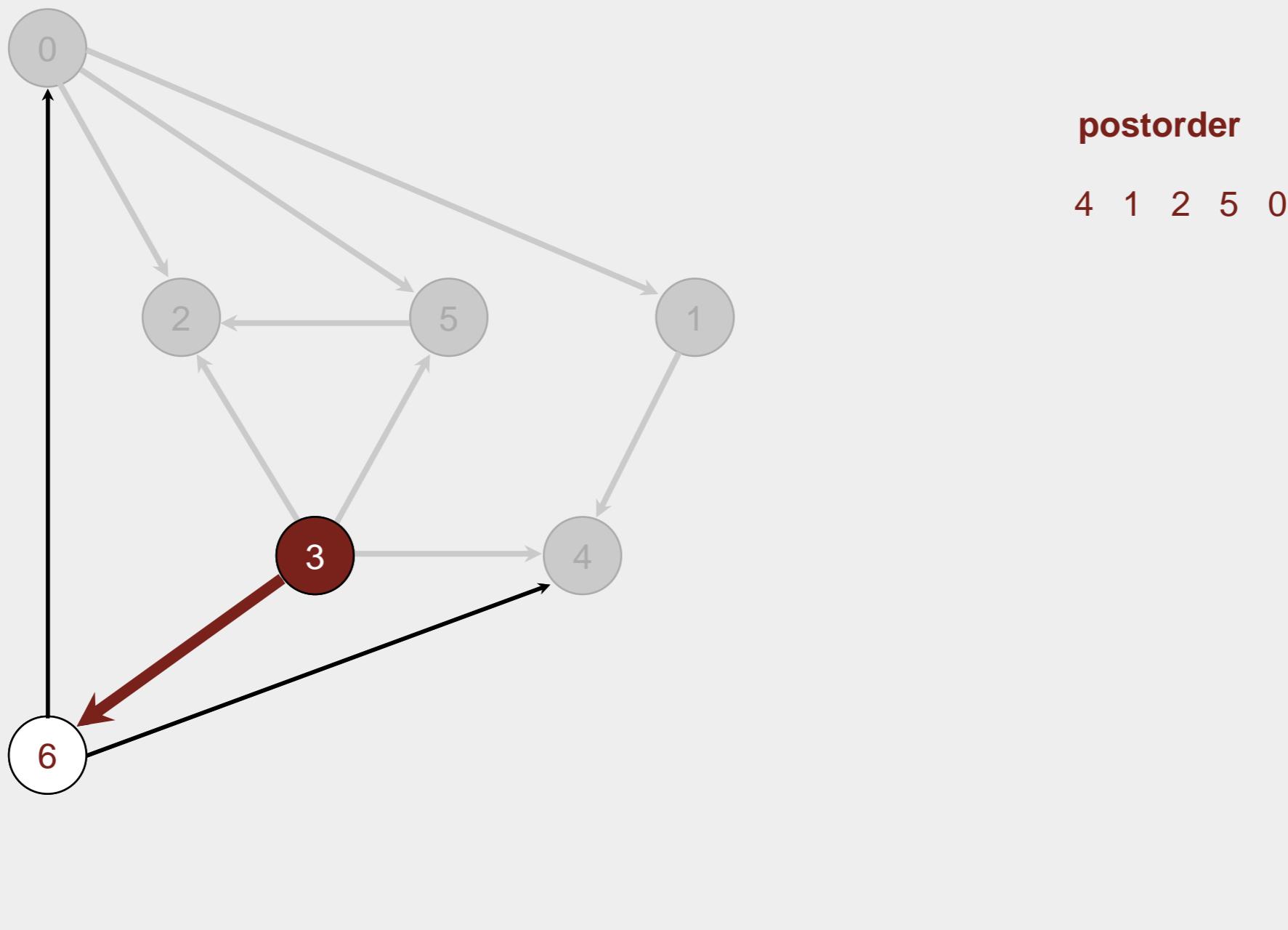
Topological sort algorithm

- Run depth-first search.
- Return vertices in reverse postorder.



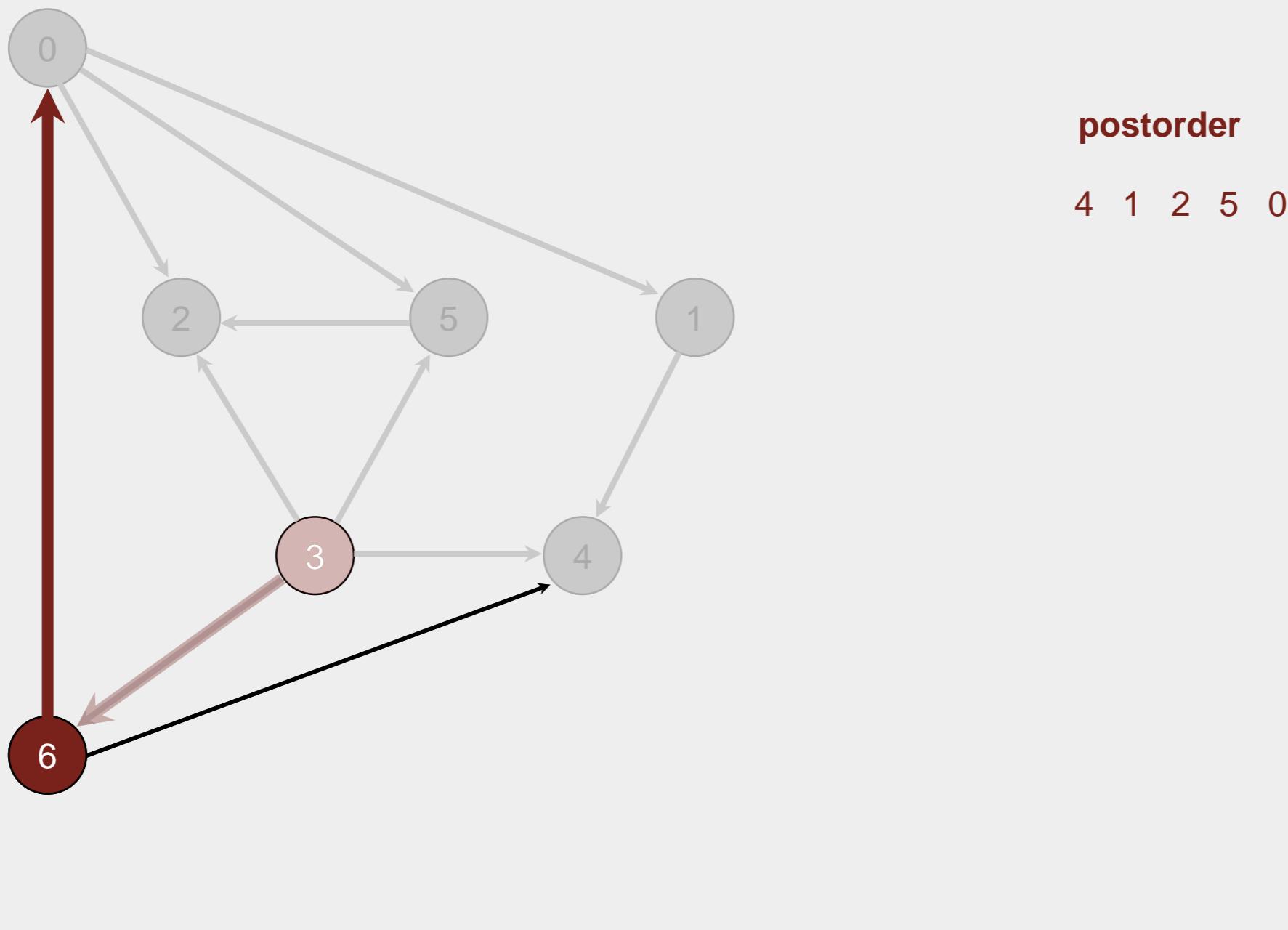
Topological sort algorithm

- Run depth-first search.
- Return vertices in reverse postorder.



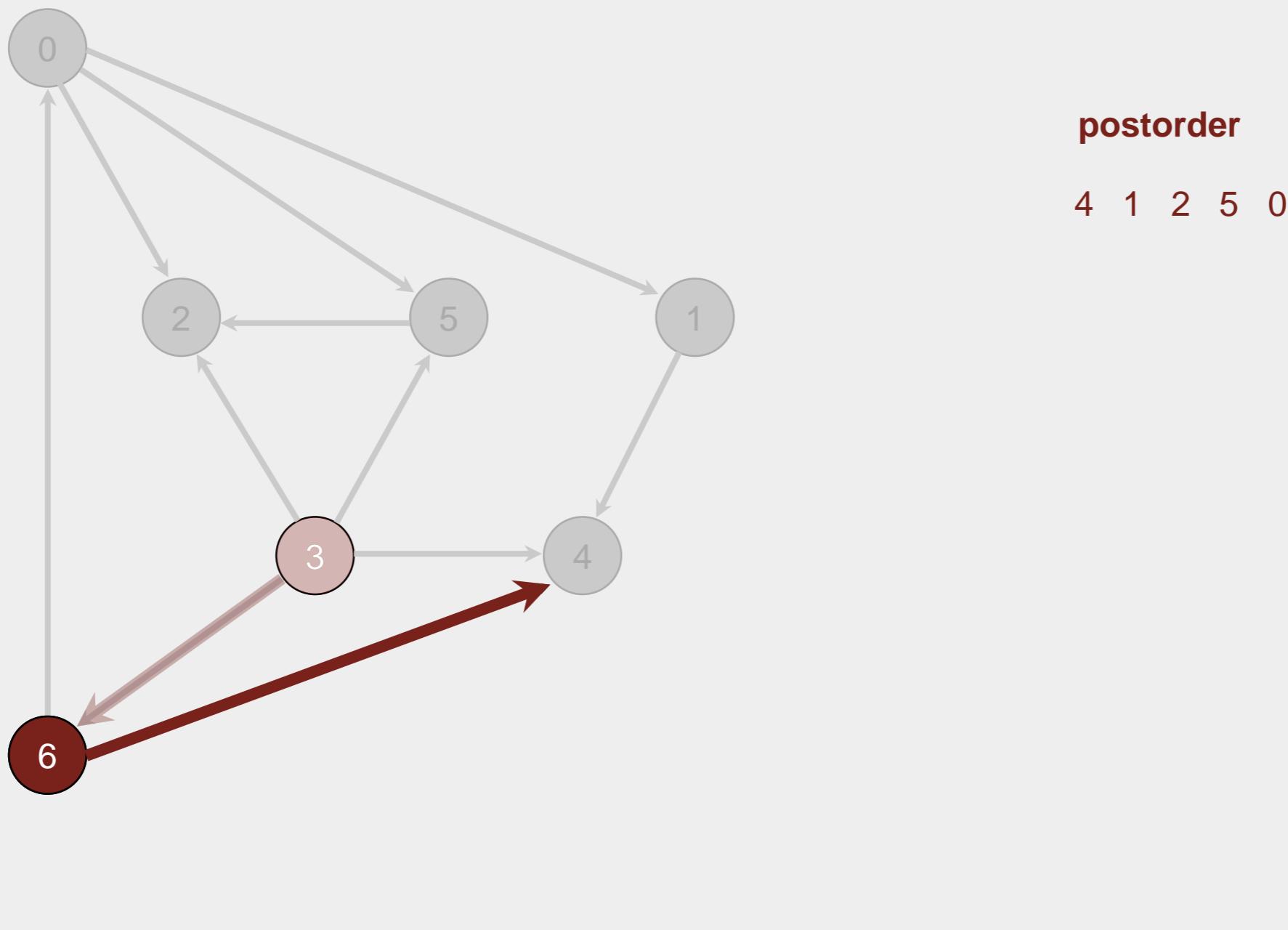
Topological sort algorithm

- Run depth-first search.
- Return vertices in reverse postorder.



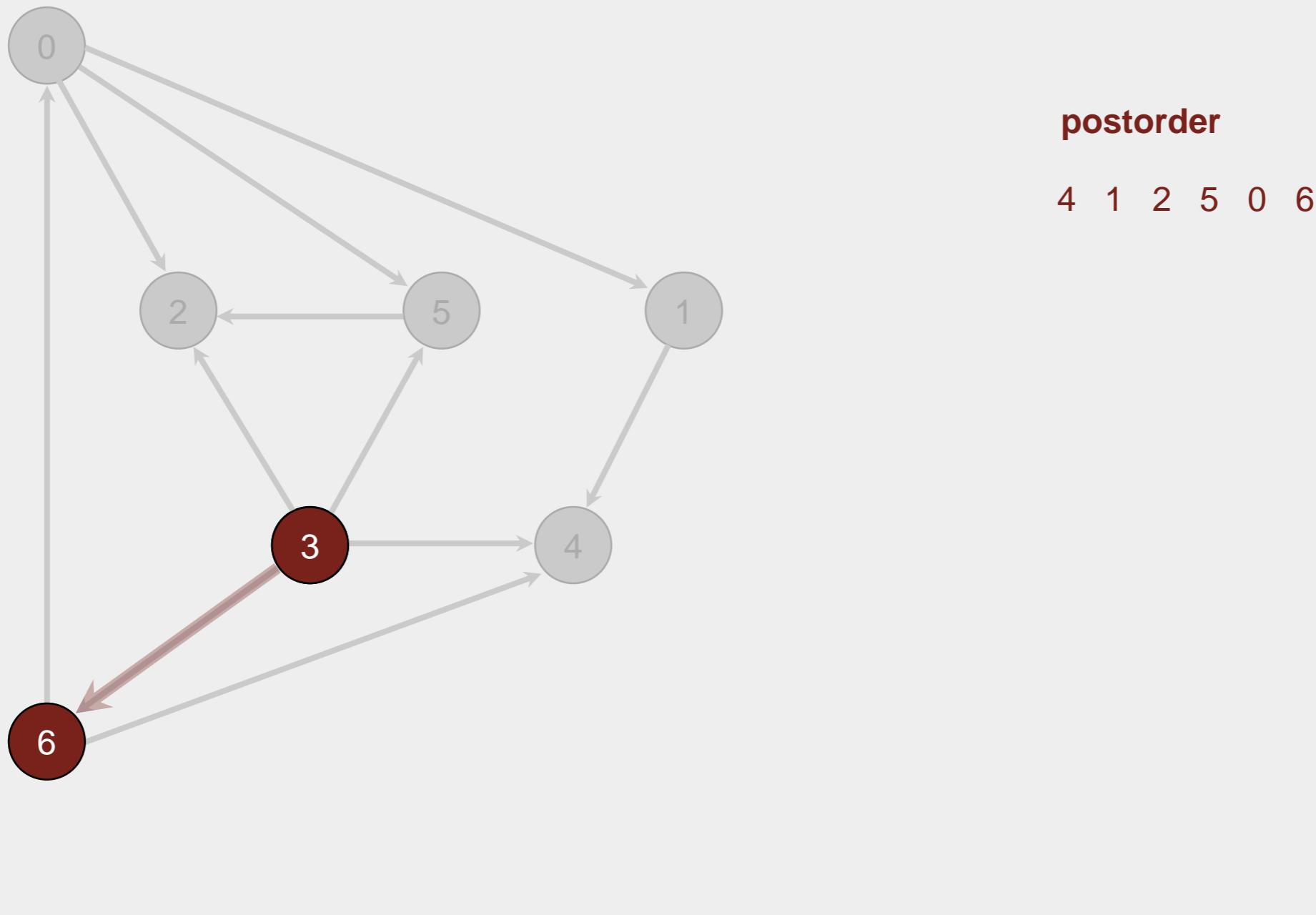
Topological sort algorithm

- Run depth-first search.
- Return vertices in reverse postorder.



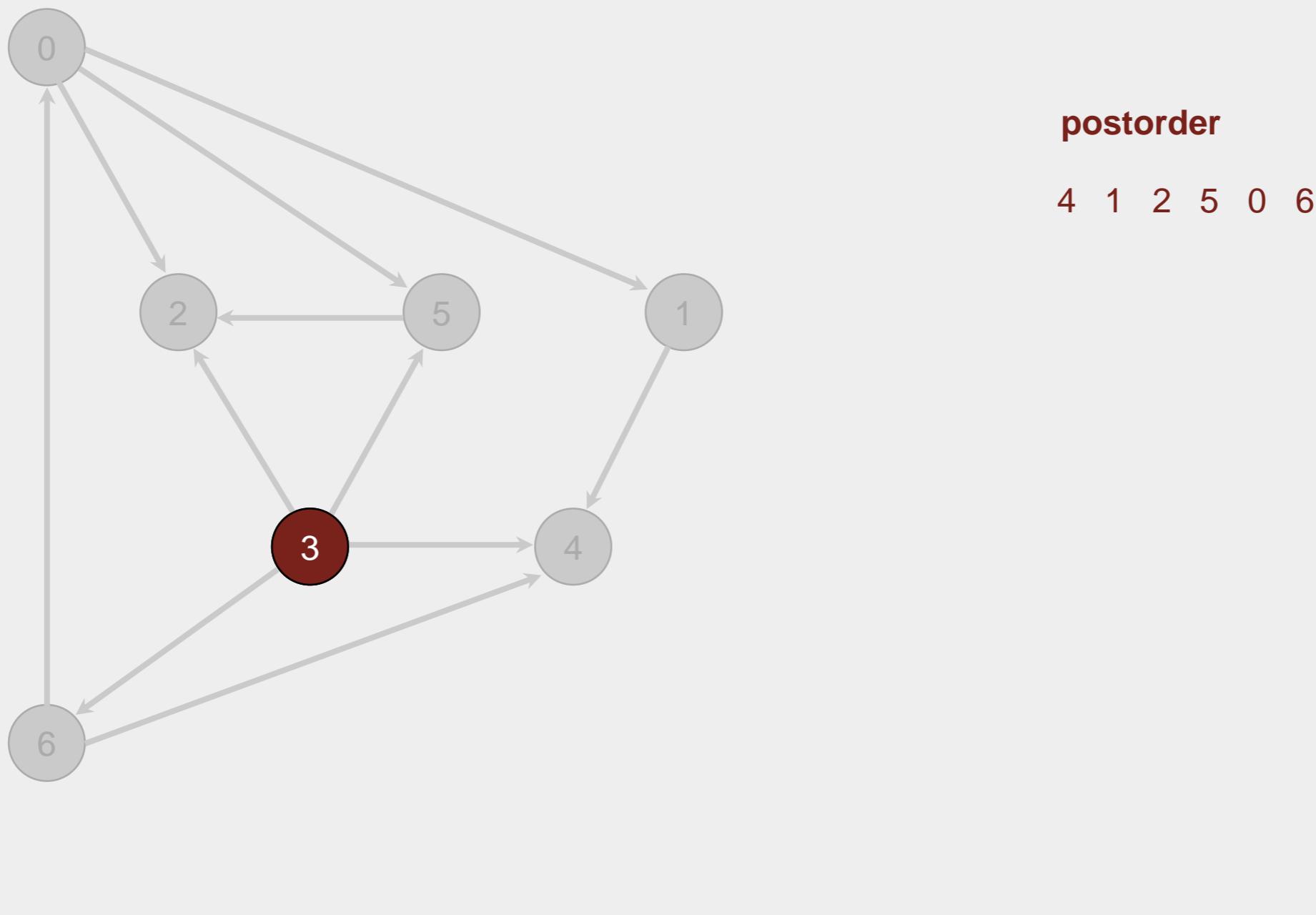
Topological sort algorithm

- Run depth-first search.
- Return vertices in reverse postorder.



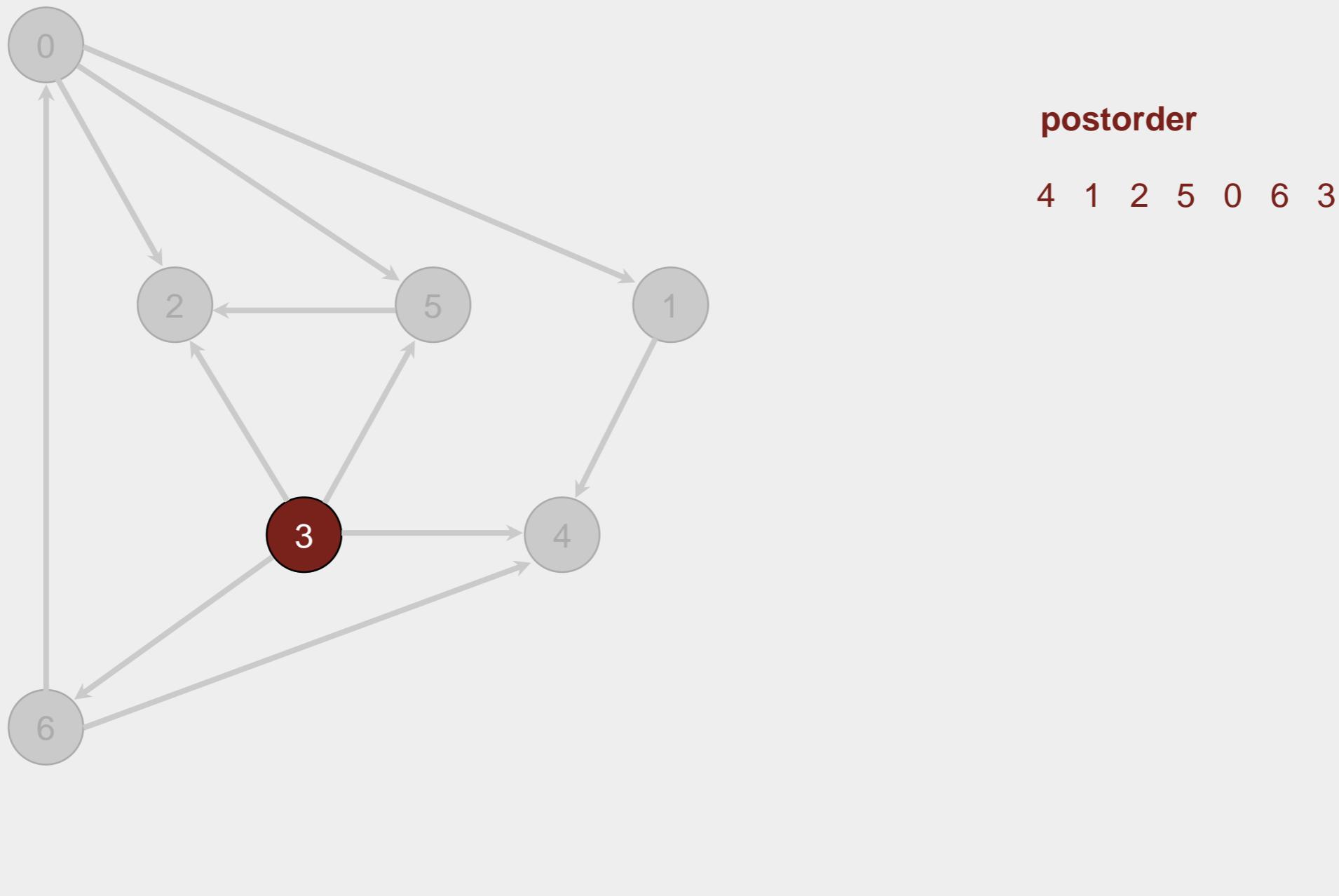
Topological sort algorithm

- Run depth-first search.
- Return vertices in reverse postorder.



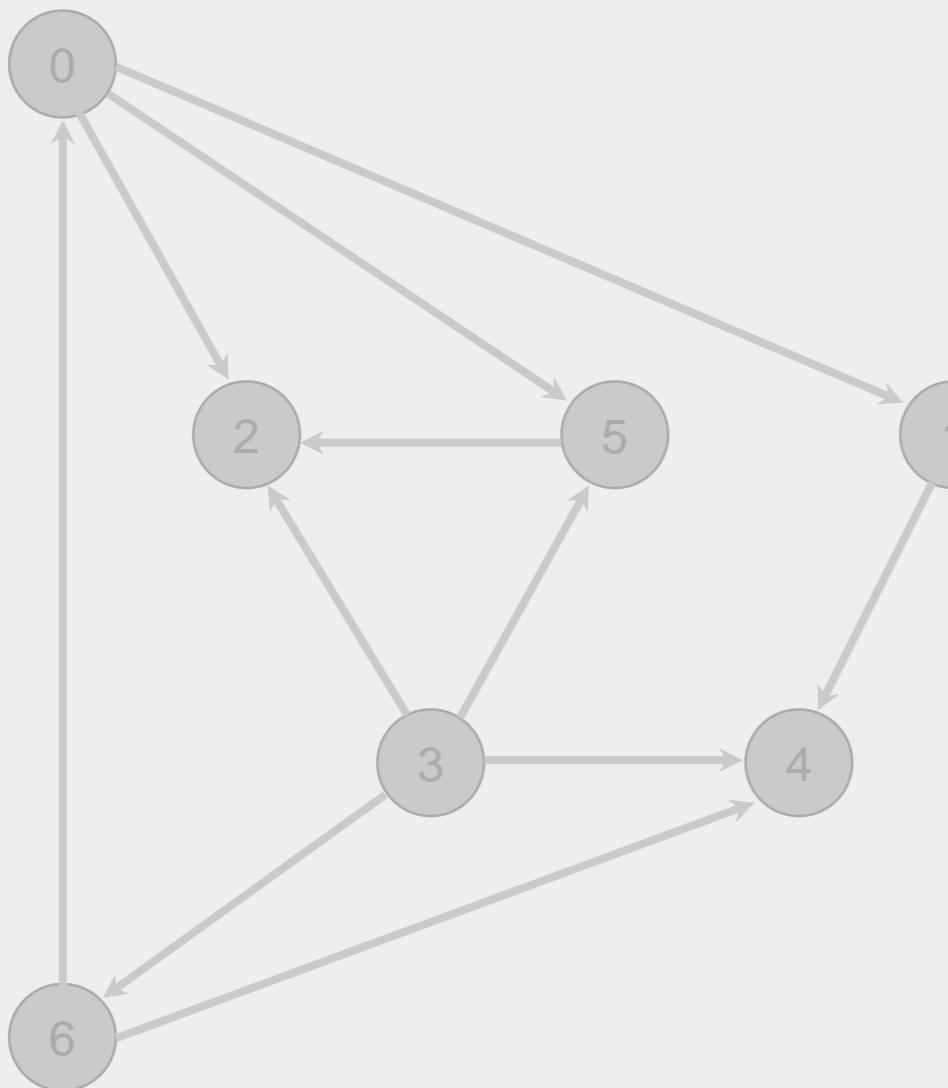
Topological sort algorithm

- Run depth-first search.
- Return vertices in reverse postorder.



Topological sort algorithm

- Run depth-first search.
- Return vertices in reverse postorder.



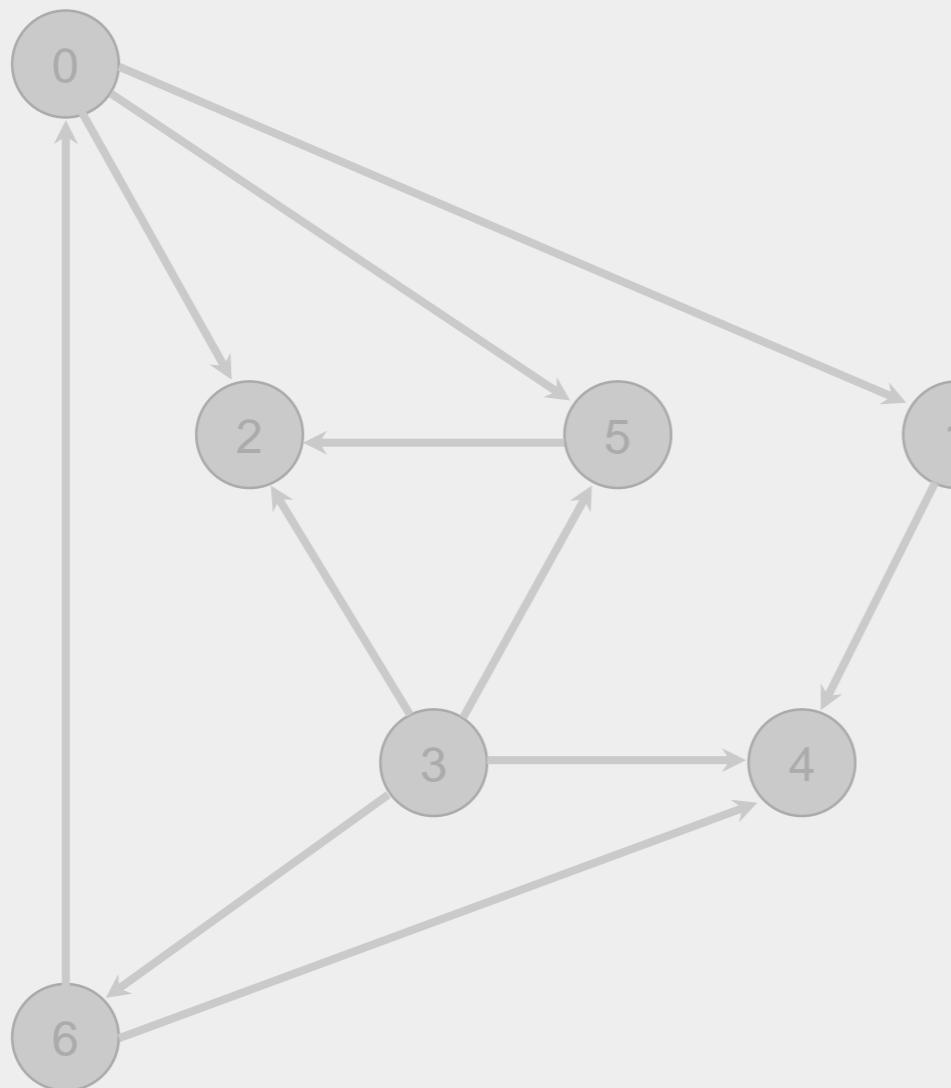
postorder

4 1 2 5 0 6 3

check 4

Topological sort algorithm

- Run depth-first search.
- Return vertices in reverse postorder.



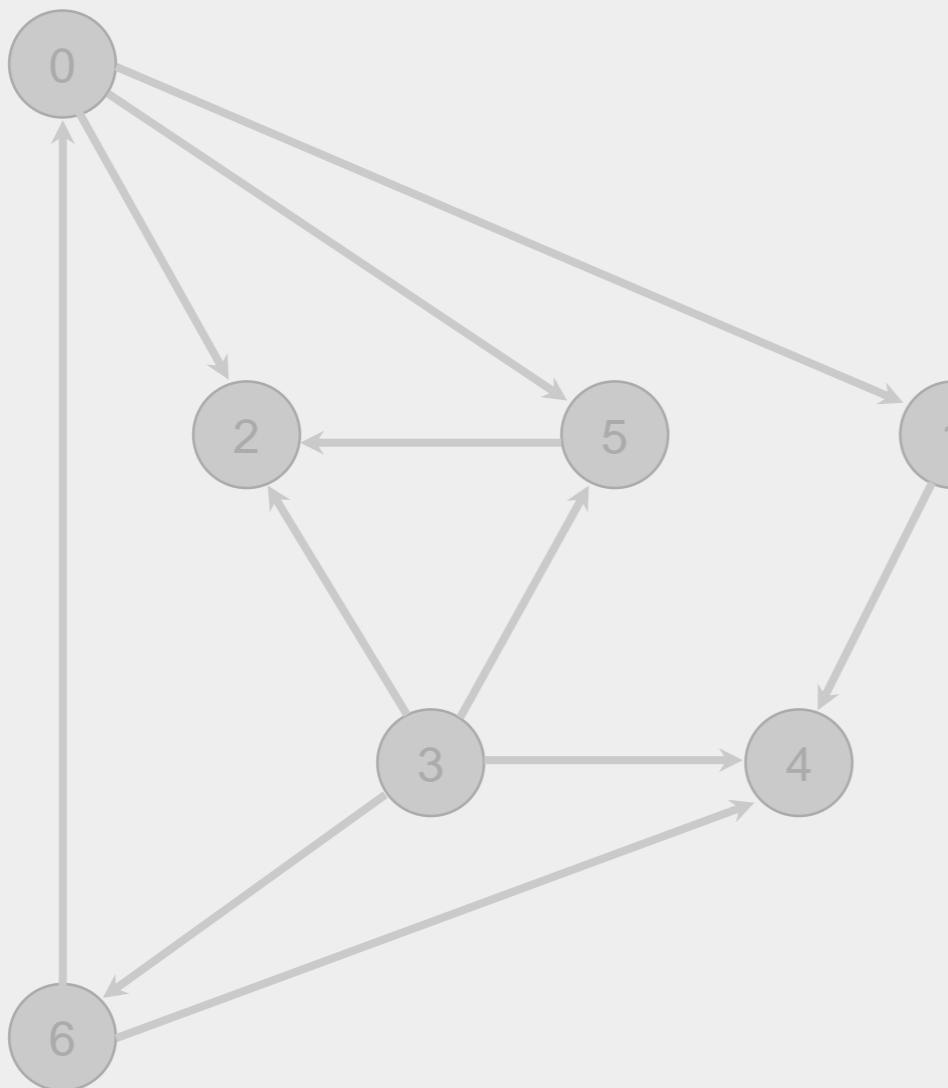
postorder

4 1 2 5 0 6 3

check 5

Topological sort algorithm

- Run depth-first search.
- Return vertices in reverse postorder.



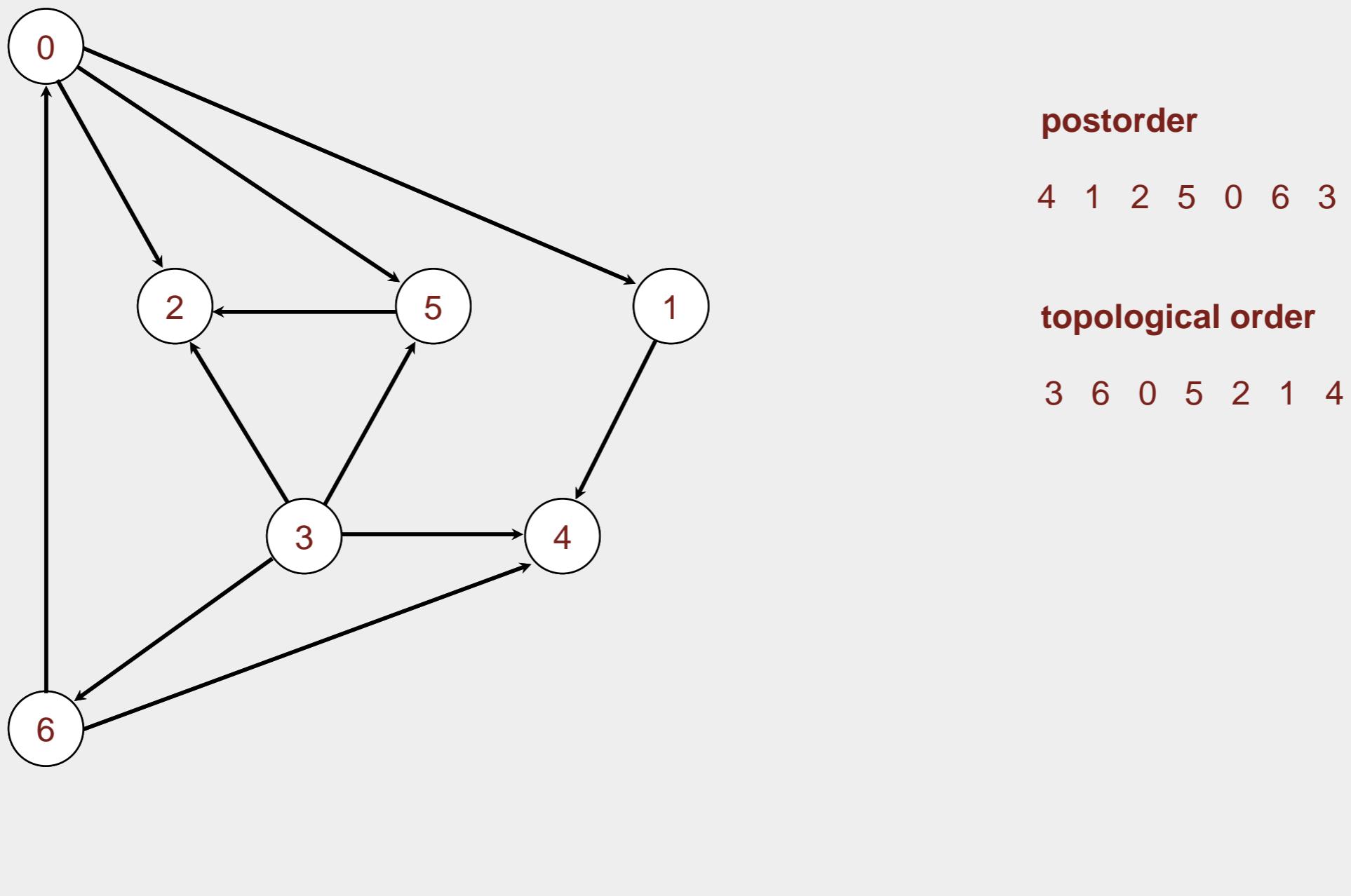
postorder

4 1 2 5 0 6 3

check 6

Topological sort algorithm

- Run depth-first search.
- Return vertices in reverse postorder.



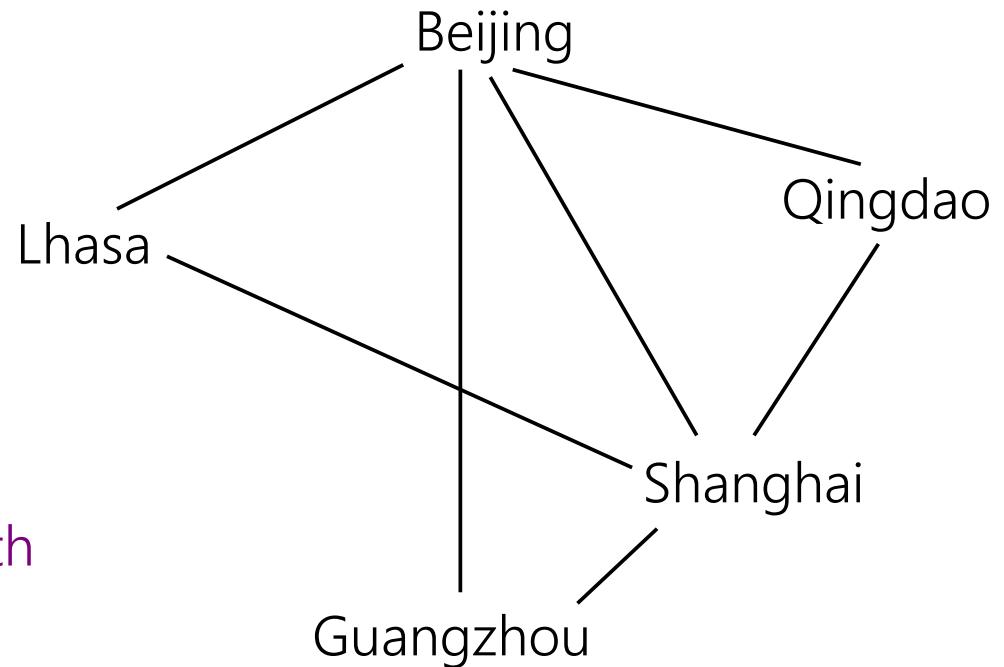
Lecture 8 Shortest Path

- Shortest-path problems
- Single-source shortest path
- All-pair shortest path

Overview

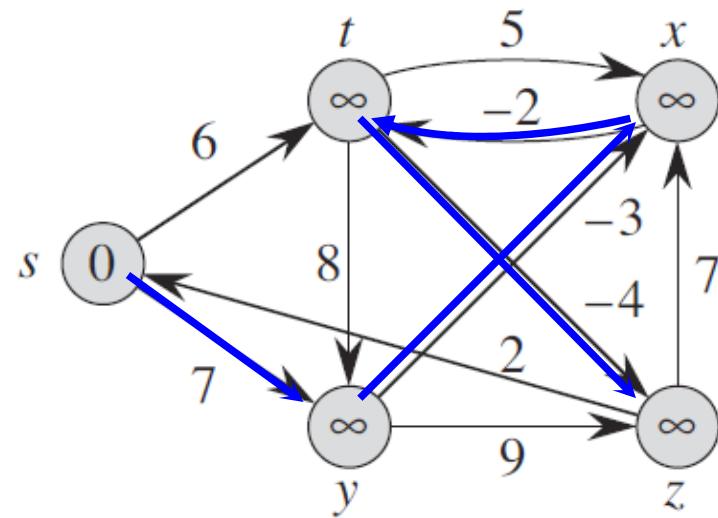
- Shortest-path problems
- Single-source shortest path algorithms
 - Bellman-Ford algorithm
 - Dijkstra algorithm
- All-Pair shortest path algorithms
 - Floyd-Warshall algorithm

Weighted graph



- Single-source shortest path
- All-pair shortest path

Shortest Path

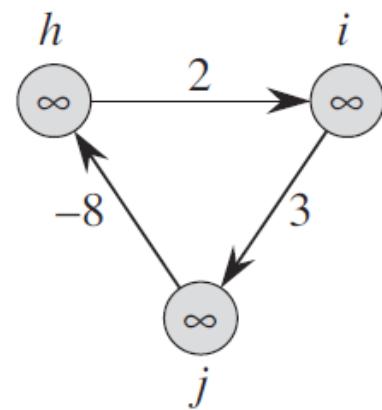


Shortest Path

Optimal substructure:
Subpaths of shortest paths are shortest paths.

Cycles. Can a shortest path contains cycles?

Negative weights.

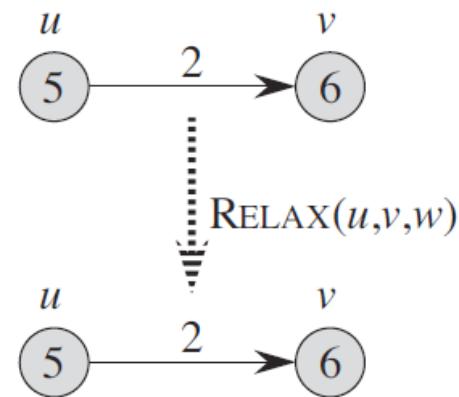
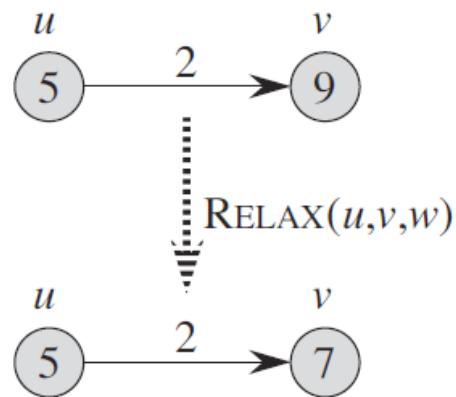


Where are we?

- Shortest-path problems
- Single-source shortest path algorithms
 - Bellman-Ford algorithm
 - Dijkstra algorithm
- All-Pair shortest path algorithms
 - Floyd-Warshall algorithm

Relaxing

The process of *relaxing* an edge (u,v) consists of testing whether we can improve the shortest path to v found so far by going through u .



Shortest-path properties

Notation: We fix the source to be s .

$\lambda[v]$: the length of path computed by our algorithms from s to v .

$\delta[v]$: the length of the shortest path from s to v .

Triangle property

$$\delta[v] \leq \delta[u] + w(u,v) \text{ for any edge } (u,v).$$

Upper-bound property

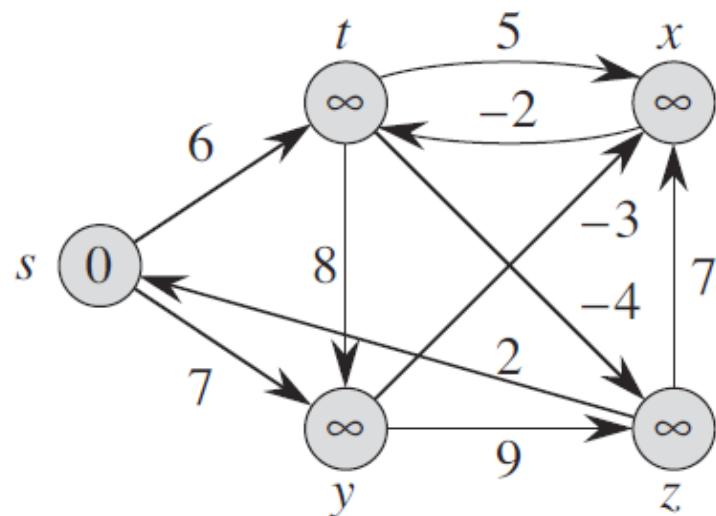
$$\delta[v] \leq \lambda[v] \text{ for any vertex } v.$$

Convergence property

If $s \Rightarrow u \rightarrow v$ is a shortest path, and if $\lambda[u] = \delta[u]$,
then after $\text{relax}(u,v)$, we have $\lambda[v] = \delta[v]$.

Bellman-Ford algorithm

$$E = \{(t,x), (t,y), (t,z), (x,t), (y,x), (y,z), (z,x), (z,s), (s,t), (s,y)\}$$



BELLMAN-FORD(G, w, s)

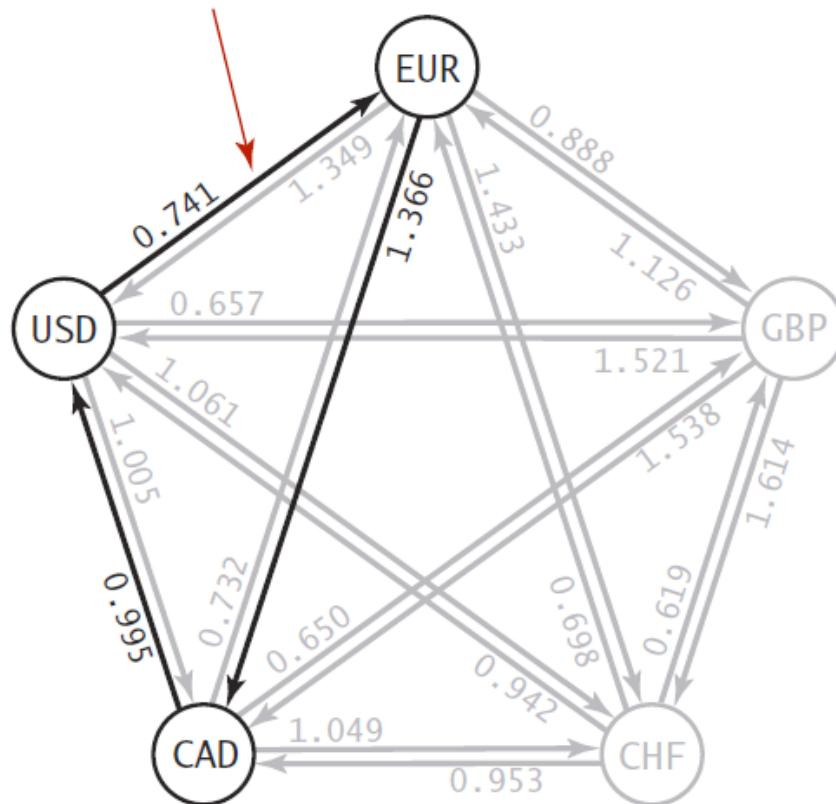
```
1 INITIALIZE-SINGLE-SOURCE( $G, s$ )
2 for  $i = 1$  to  $|G.V| - 1$ 
3   for each edge  $(u, v) \in G.E$ 
4     RELAX( $u, v, w$ )
5   for each edge  $(u, v) \in G.E$ 
6     if  $v.d > u.d + w(u, v)$ 
7       return FALSE
8 return
```



$\Theta(mn)$

Negative cycle

$$0.741 * 1.366 * .995 = 1.00714497$$



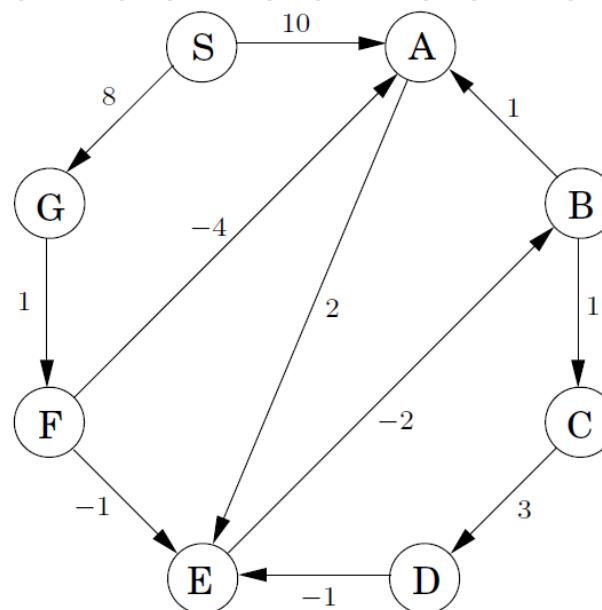
An arbitrage opportunity

Quiz

Run Bellman-Ford algorithm for the following graph,
provided by

$E =$

$\{(S,A),(S,G),(A,E),(B,A),(B,C),(C,D),(D,E),(E,B),(F,A),(F,E),(G,F)\}$



Correctness

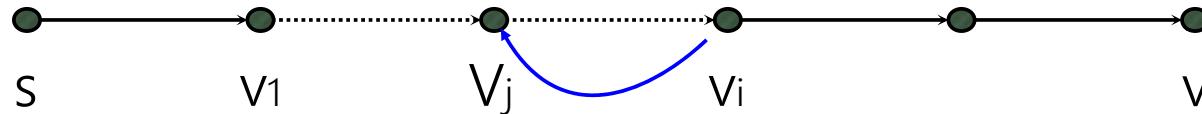
Correctness of Bellman-Ford

If G contains no negative-weight cycles reachable from s , then the algorithm returns TRUE, and for all v , $\lambda[v] = \delta[v]$, otherwise the algorithm returns FALSE.

Proof.

No negative cycle. By the fact that the length of simple paths is bounded by $|V| - 1$.

After i -th iteration of relax, $\lambda[v_i] = \delta[v_i]$.



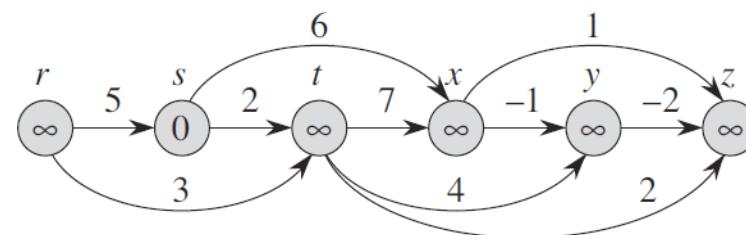
Negative cycle. After i -th iteration of relax,

$$\lambda[v_i] + w(v_i, v_j) < \lambda[v_j].$$

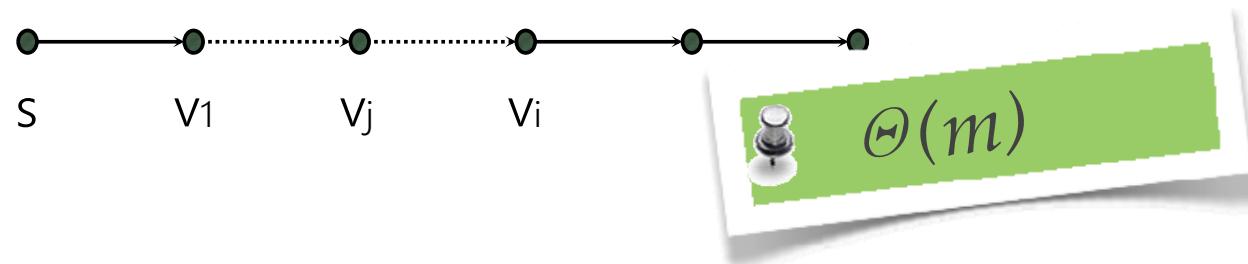
The length of negative cycle is at most $|V|$.

Observation 1

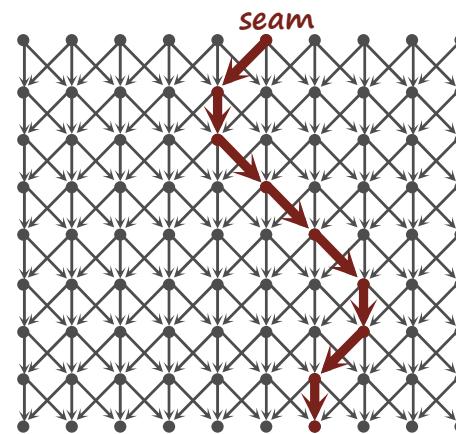
If there is no cycle (DAG), ...



Relax in topological order.

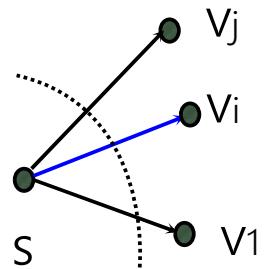


Weighted DAG application



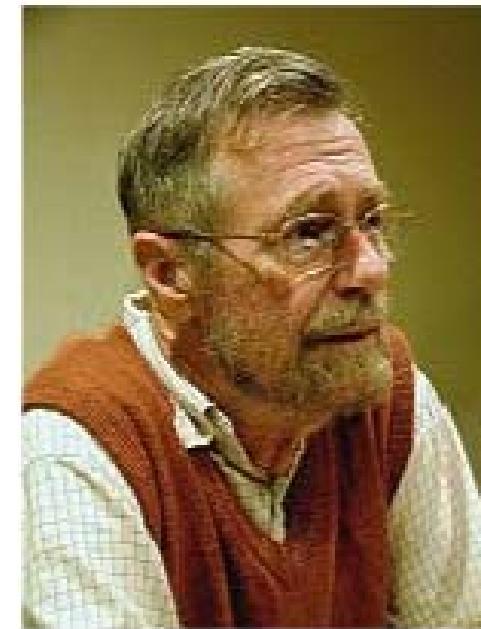
Observation 2

If there is no negative edge, ...



If (s, v_i) is the lightest edge sourcing from s , then $\lambda[v_i] = \delta[v_i]$

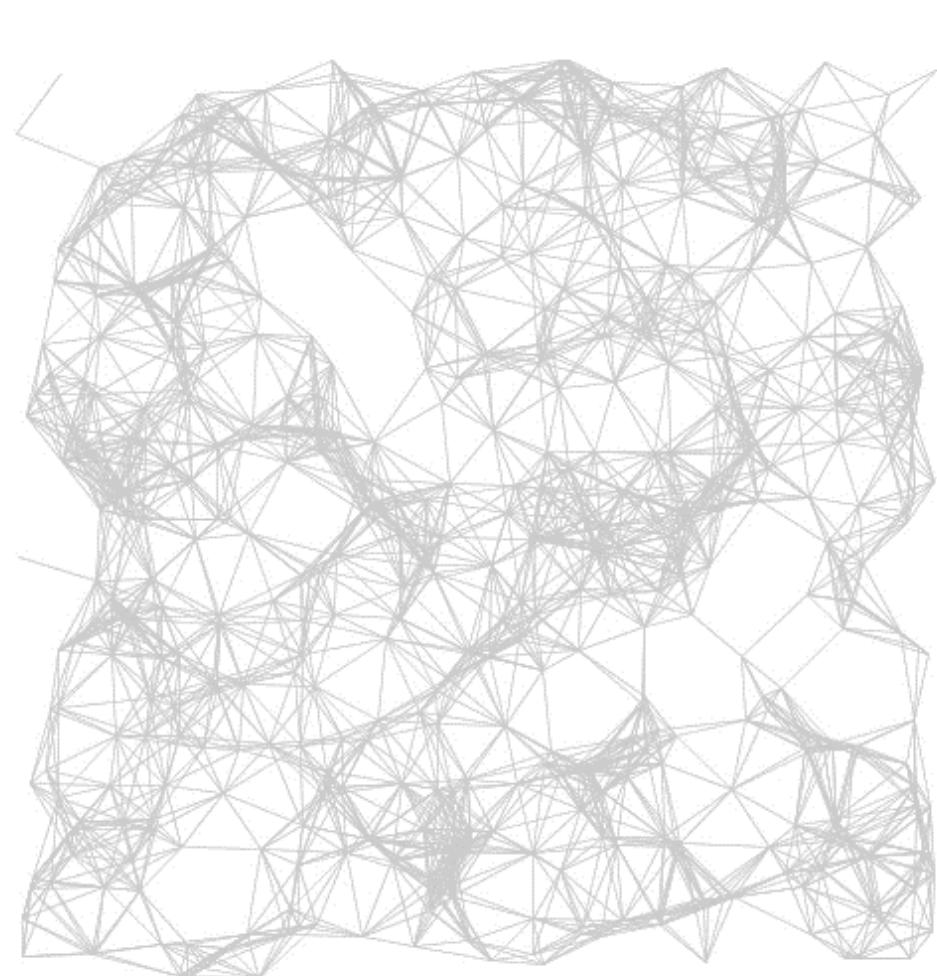
Dijkstra algorithm



Edsger Wybe Dijkstra in 2002

Dijkstra algorithm

```
1.  $X \leftarrow \{1\}$ ;  $Y \leftarrow V \setminus \{1\}$ ;  $\lambda[1] \leftarrow 0$ 
2. for  $y \leftarrow 2$  to  $n$ 
3.   if  $y$  is adjacent to 1 then  $\lambda[y] \leftarrow \text{length}[1, y]$ 
4.   else  $\lambda[y] \leftarrow \infty$  end if
5. end for
6. for  $j \leftarrow 1$  to  $n - 1$ 
7.   Let  $y \in Y$  be such that  $\lambda[y]$  is minimum
8.    $X \leftarrow X \cup \{y\}$ 
9.    $Y \leftarrow Y \setminus \{y\}$ 
10.  for each edge  $(y, w)$ 
11.    if  $w \in Y$  and  $\lambda[y] + \text{length}[y, w] < \lambda[w]$ 
12.      then  $\lambda[w] \leftarrow \lambda[y] + \text{length}[y, w]$ 
13.    end if
14. end for
```



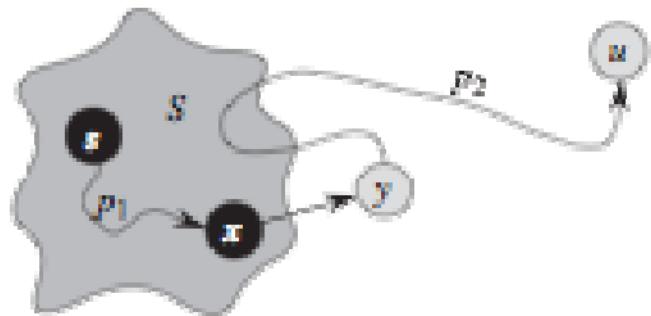
Correctness

Theorem (CorrectnessOfDijkstraAlgorithm)

In Algorithm Dijkstra, when a vertex u is chosen in Step 7, if its label $\lambda[u]$ is finite then $\lambda[u] = \delta[u]$.

Assume u is chosen in Step 7, and

1. $\lambda[u] > \delta[u]$
2. $S \Rightarrow x \rightarrow y \Rightarrow u$ is the shortest path



$$\begin{aligned}\delta[u] &= \delta[x] + w(x,y) + w(p_2) \\ &= \lambda[x] + w(x,y) + w(p_2) \\ &\geq \lambda[x] + w(x,y) \\ &\geq \lambda[y] \geq \lambda[u]\end{aligned}$$

Time complexity

```
1.  $X \leftarrow \{1\}$ ;  $Y \leftarrow V \setminus \{1\}$ ;  $\lambda[1] \leftarrow 0$ 
2. for  $y \leftarrow 2$  to  $n$ 
3.   if  $y$  is adjacent to 1 then  $\lambda[y] \leftarrow \text{length}[1, y]$ 
4.   else  $\lambda[y] \leftarrow \infty$  end if
5. end for
6. for  $i \leftarrow 1$  to  $n - 1$ 
7.   Let  $y \in Y$  be such that  $\lambda[y]$  is minimum
8.    $X \leftarrow X \cup \{y\}$ 
9.    $Y \leftarrow Y \setminus \{y\}$ 
10.  for each edge  $(y, w)$ 
11.    if  $w \in Y$  and  $\lambda[y] + \text{length}[y, w] < \lambda[w]$ 
12.      then  $\lambda[w] \leftarrow \lambda[y] + \text{length}[y, w]$ 
13.    end if
14. end for
```



$\Theta(n^2)$

```
1.  $Y \leftarrow V \setminus \{1\}$ ;  $\lambda[1] \leftarrow 0$ ;  $key(1) \leftarrow \lambda[1]$ 
2. for  $y \leftarrow 2$  to  $n$ 
3.   if  $y$  is adjacent to 1 then
4.      $\lambda[y] \leftarrow length[1, y]$ 
5.      $key(y) \leftarrow \lambda[y]$ 
6.     Insert( $H, y$ )
7.   else
8.      $\lambda[y] \leftarrow \infty$ 
9.      $key(y) \leftarrow \lambda[y]$ 
10.  end if
11. end for
```

```
12. for  $j \leftarrow 1$  to  $n - 1$ 
13.    $y \leftarrow DeleteMin(H)$ 
14.    $Y \leftarrow Y \setminus \{y\}$ 
15.   for each edge  $(y, w)$  such that  $w \in Y$ 
16.     if  $\lambda[y] + length[y, w] < \lambda[w]$  then
17.        $\lambda[w] \leftarrow \lambda[y] + length[y, w]$ 
18.        $key(w) \leftarrow \lambda[w]$ 
19.     end if
20.     if  $w \notin H$  then Insert( $H, w$ )
21.     else SiftUp( $H, H^{-1}(w)$ )
22.   end if
23. end for
24. end for
```



What is the time complexity?
How about use d-heap?

Comparisons

	Array	Binary heap	d-ary heap
Dijkstra	$O(n^2)$	$O(m \log n)$	$O(dn \log_d n + m \log_d n)$

Dense graph

- dense: $m = n^{1+\varepsilon}$, ε is not too small.
- d-heap, $d = m/n$
- complexity: $O(dn\log_dn + m\log_dn) = O(m)$

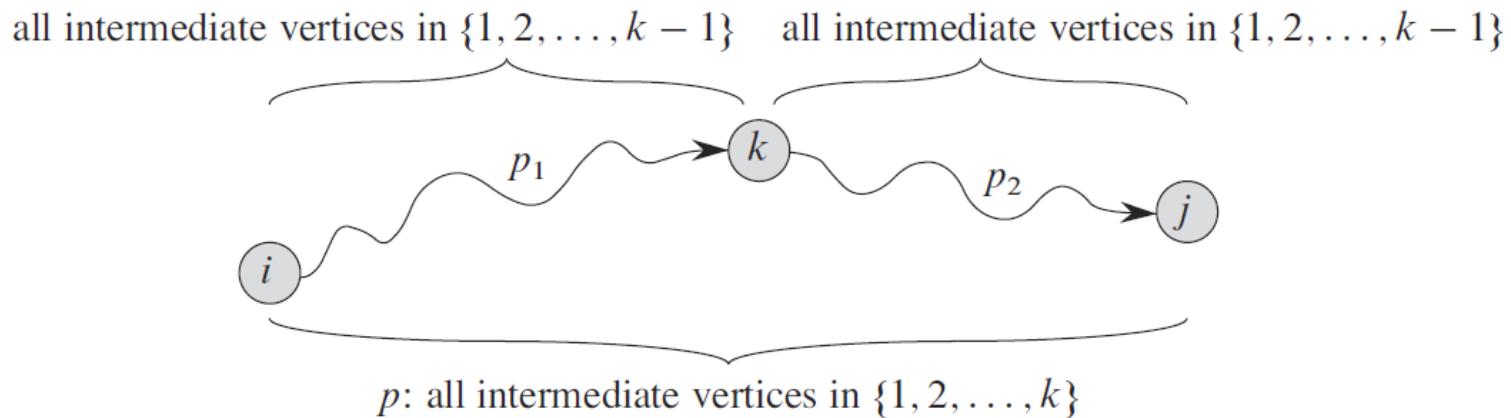
Where are we?

- Shortest-path problems
- Single-source shortest path algorithms
 - Bellman-Ford algorithm
 - Dijkstra algorithm
- All-Pair shortest path algorithms
 - Floyd-Warshall algorithm

All-pairs shortest path

Define $d_{i,j}^k$ to be the length of a shortest path from i to j that does not pass any vertex in $\{k+1, k+2, \dots, n\}$. Clearly

$$d_{i,j}^k = \begin{cases} l[i,j] & \text{if } k = 0 \\ \min\{d_{i,j}^{k-1}, d_{i,k}^{k-1} + d_{k,j}^{k-1}\} & \text{if } 1 \leq k \leq n \end{cases}$$



Floyd-Warshall algorithm



Floyd-Warshall algorithm

Algorithm 7.3 Floyd

Input: An $n \times n$ matrix $I[1..n, 1..n]$ such that $I[i, j]$ is the length of the edge (i, j) in a directed graph $G = (\{1, \dots, n\}, E)$.

Output: A matrix D with $D[i, j] =$ the distance from i to j .

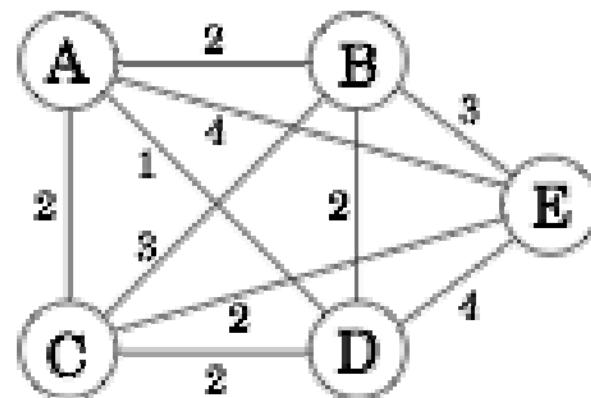
1. $D \leftarrow I$
2. for $k \leftarrow 1$ to n
3. for $i \leftarrow 1$ to n
4. for $j \leftarrow 1$ to n
5. $D[i, j] = \min\{D[i, j], D[i, k] + D[k, j]\}$
6. end for
7. end for
8. end for



$\Theta(n^3)$

Quiz

Run Floyd-Warshall algorithm for the following graph:



Conclusion

Dijkstra's algorithm.

- Nearly linear-time when weights are nonnegative.

Acyclic edge-weighted digraphs.

- Faster than Dijkstra's algorithm.
- Negative weights are no problem.

Negative weights and negative cycles.

- If no negative cycles, can find shortest paths via Bellman-Ford.
- If negative cycles, can find one via Bellman-Ford.

All-pair shortest path.

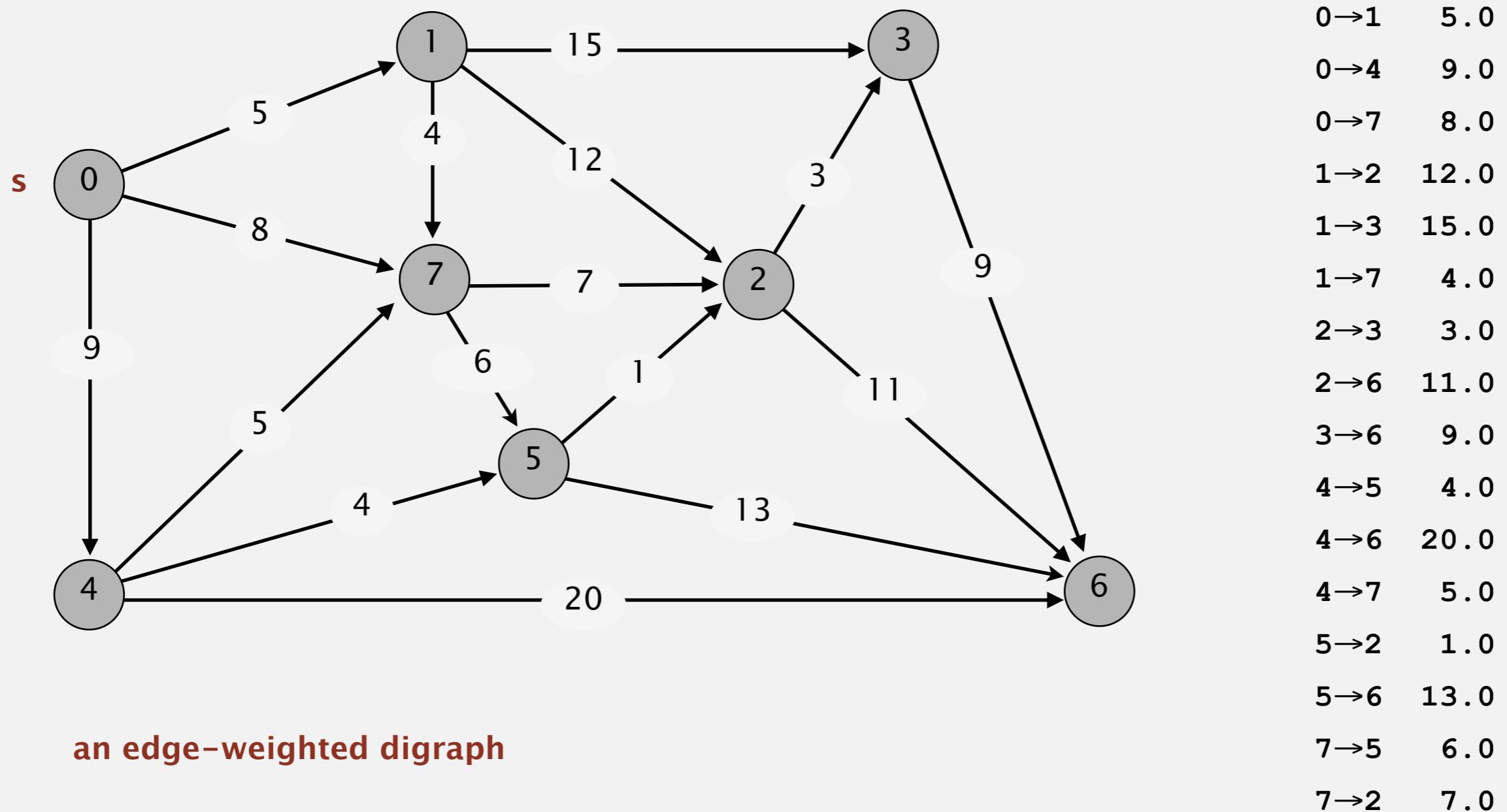
- can be solved via Floyd-Warshall
- Floyd-Warshall can also compute the transitive closure of directed graph.

4.4 BELLMAN-FORD DEMO



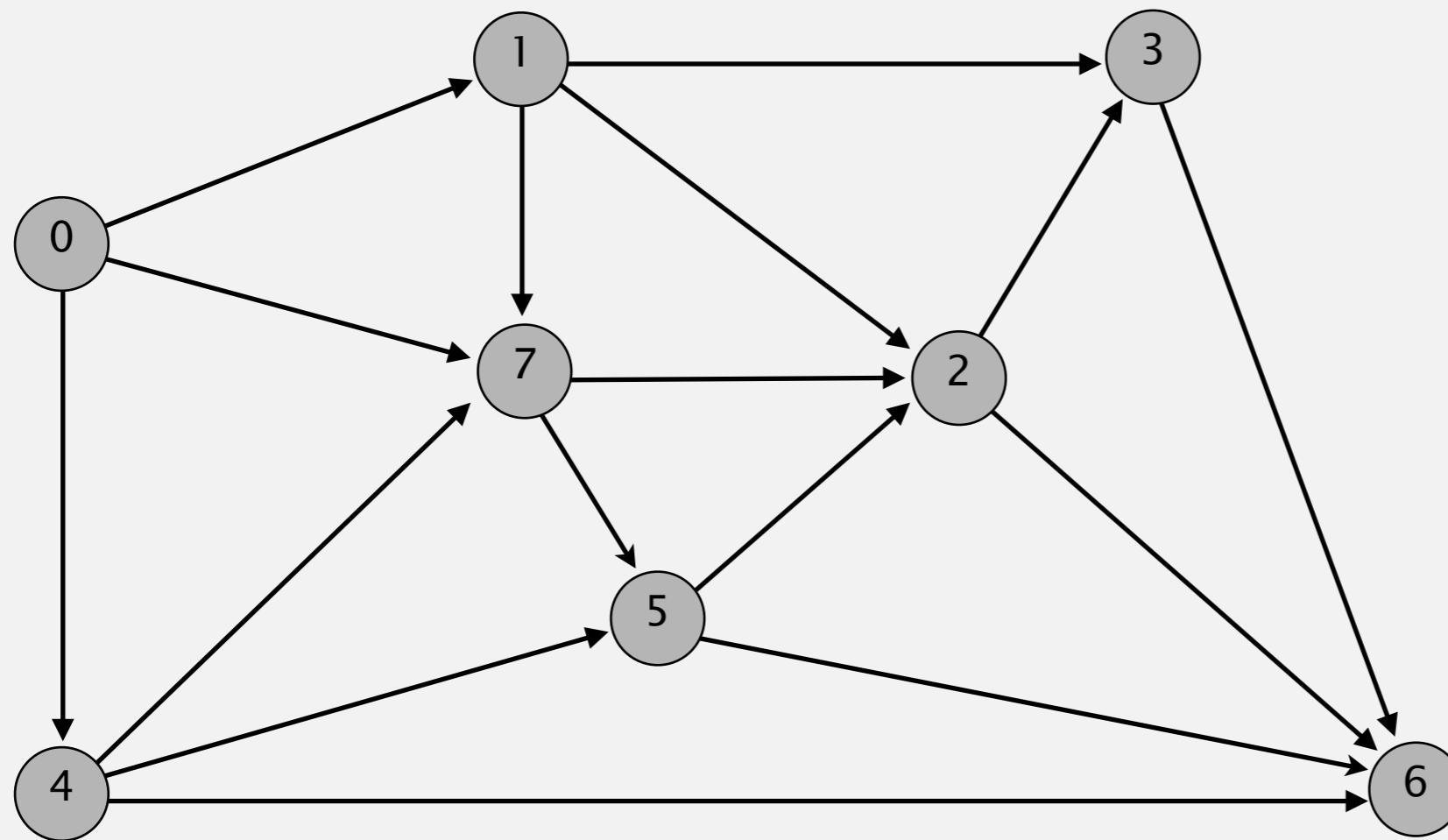
Bellman-Ford algorithm demo

Repeat V times: relax all E edges.



Bellman-Ford algorithm demo

Repeat V times: relax all E edges.

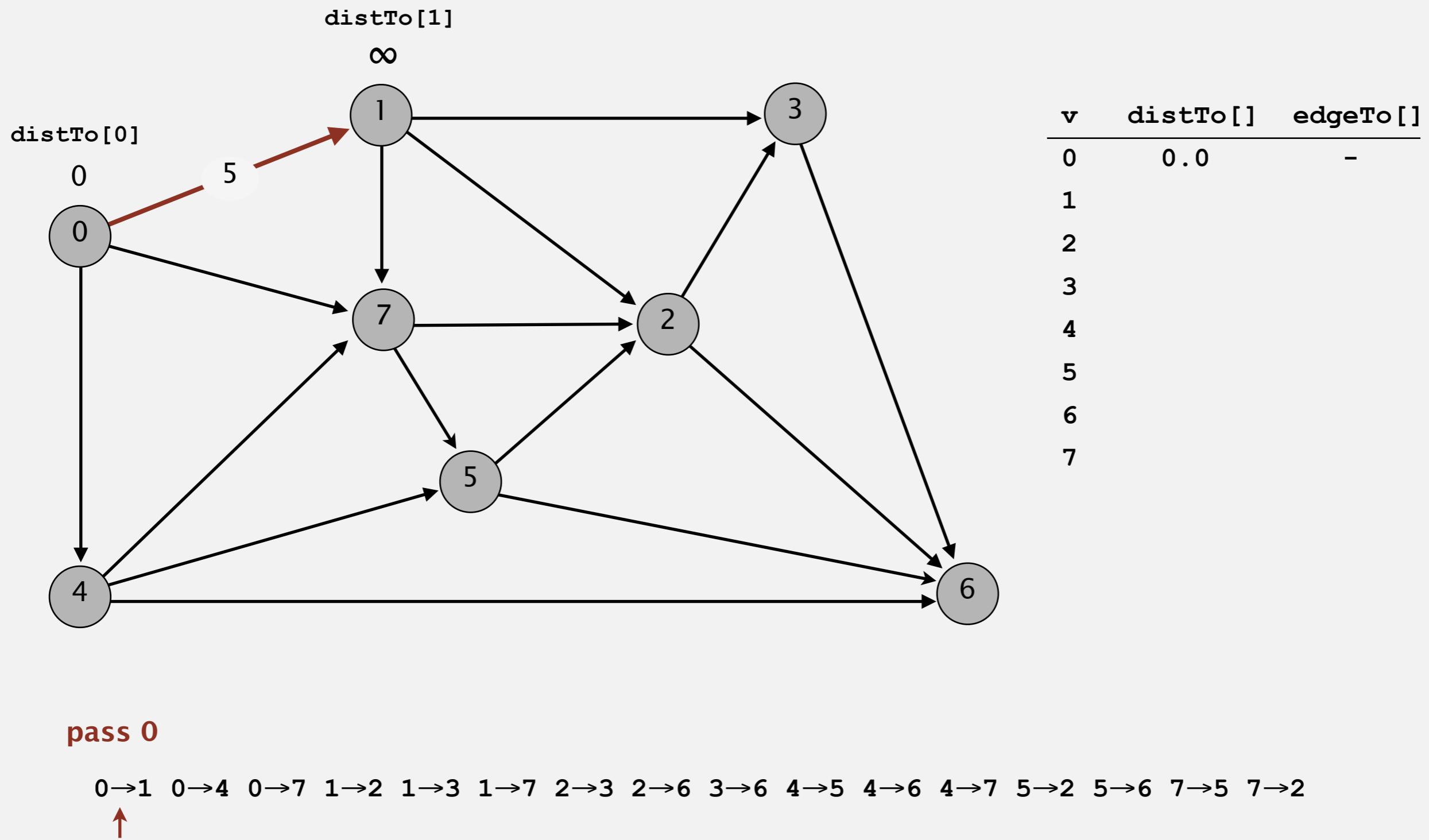


initialize

v	distTo []	edgeTo []
0	0.0	-
1		
2		
3		
4		
5		
6		
7		

Bellman-Ford algorithm demo

Repeat V times: relax all E edges.

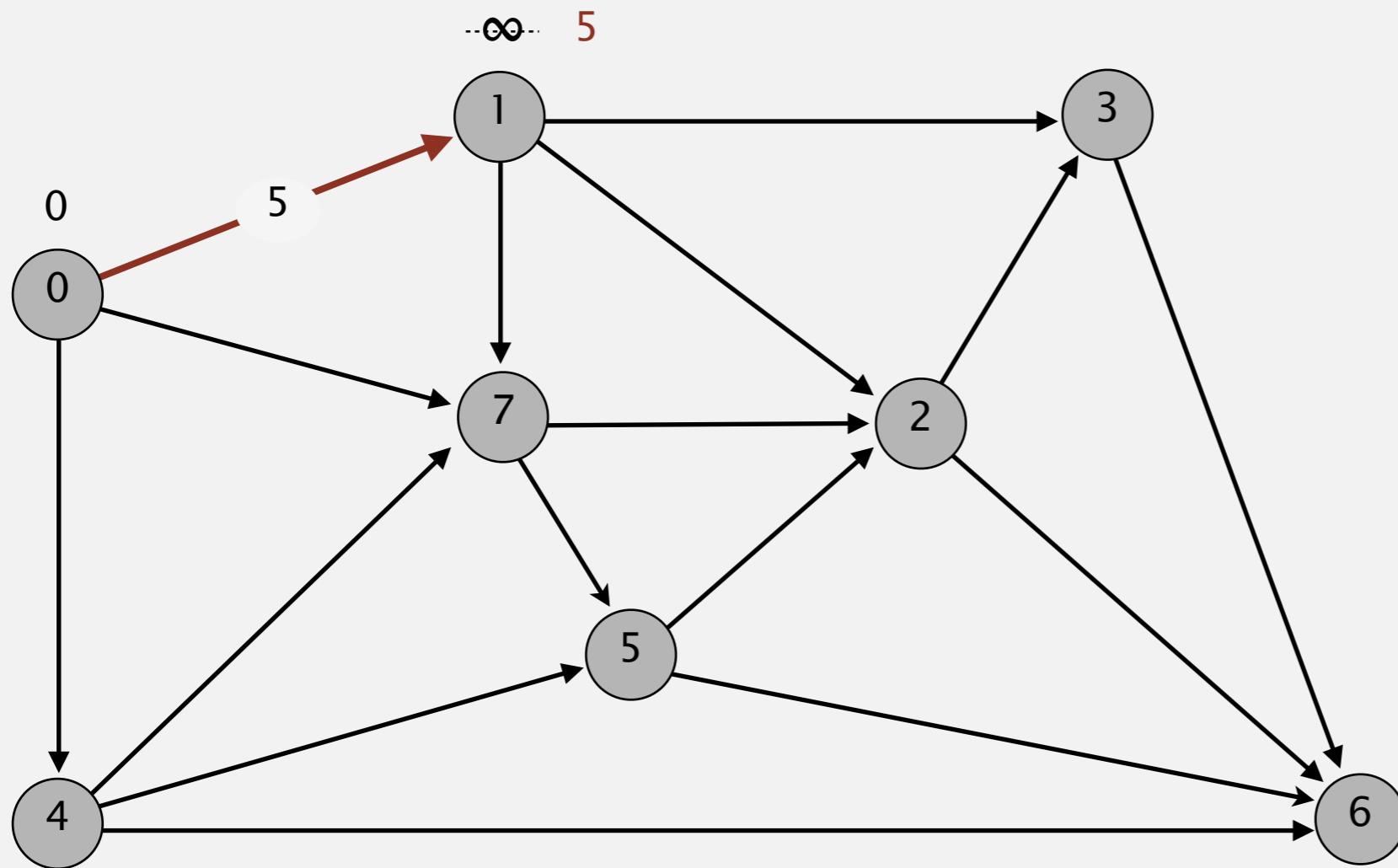


pass 0

$0 \rightarrow 1 \ 0 \rightarrow 4 \ 0 \rightarrow 7 \ 1 \rightarrow 2 \ 1 \rightarrow 3 \ 1 \rightarrow 7 \ 2 \rightarrow 3 \ 2 \rightarrow 6 \ 3 \rightarrow 6 \ 4 \rightarrow 5 \ 4 \rightarrow 6 \ 4 \rightarrow 7 \ 5 \rightarrow 2 \ 5 \rightarrow 6 \ 7 \rightarrow 5 \ 7 \rightarrow 2$

Bellman-Ford algorithm demo

Repeat V times: relax all E edges.



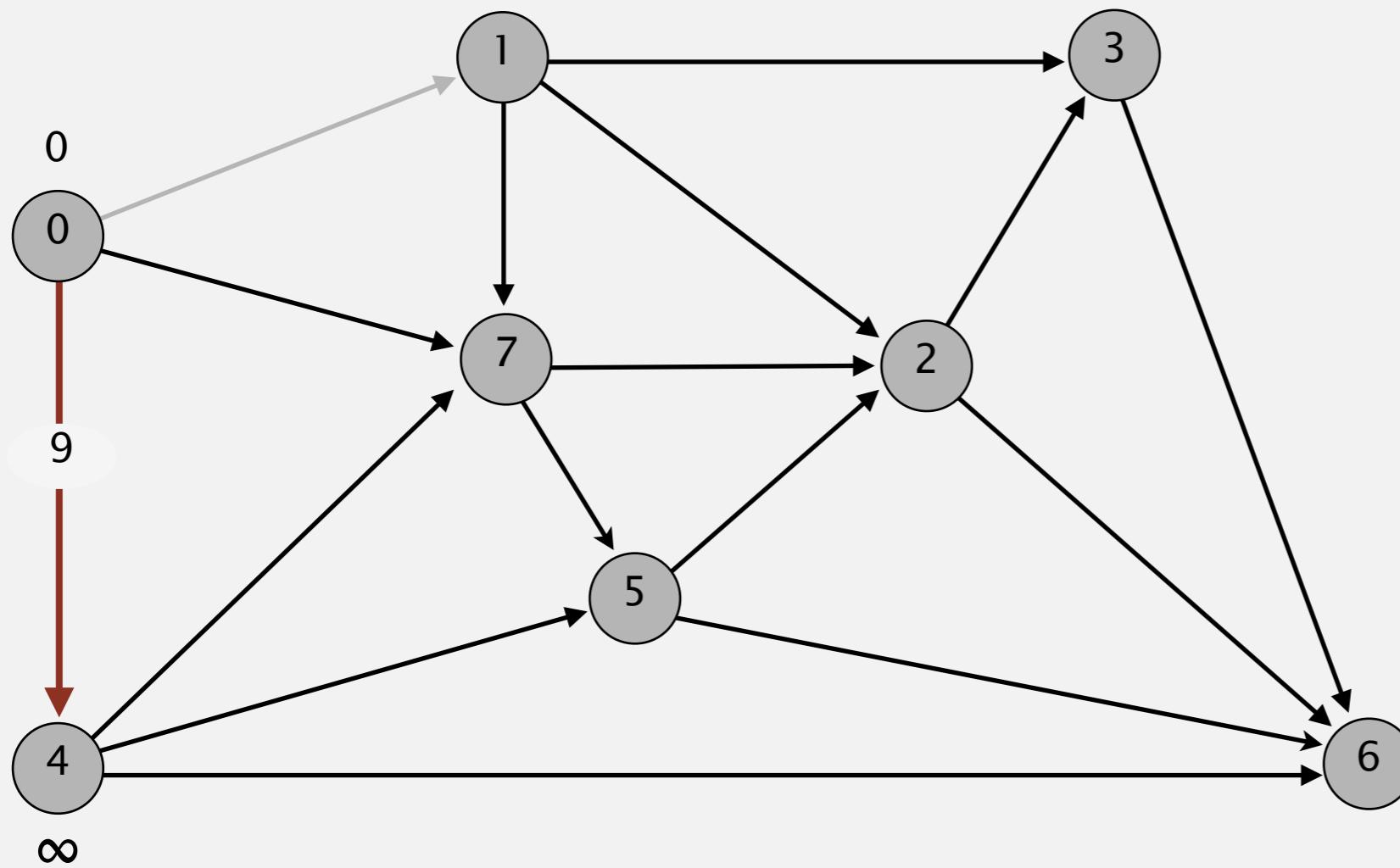
v	distTo []	edgeTo []
0	0.0	-
1	5.0	0→1
2		
3		
4		
5		
6		
7		

pass 0

0→1 0→4 0→7 1→2 1→3 1→7 2→3 2→6 3→6 4→5 4→6 4→7 5→2 5→6 7→5 7→2
↑

Bellman-Ford algorithm demo

Repeat V times: relax all E edges.



v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0→1
2		
3		
4		
5		
6		
7		

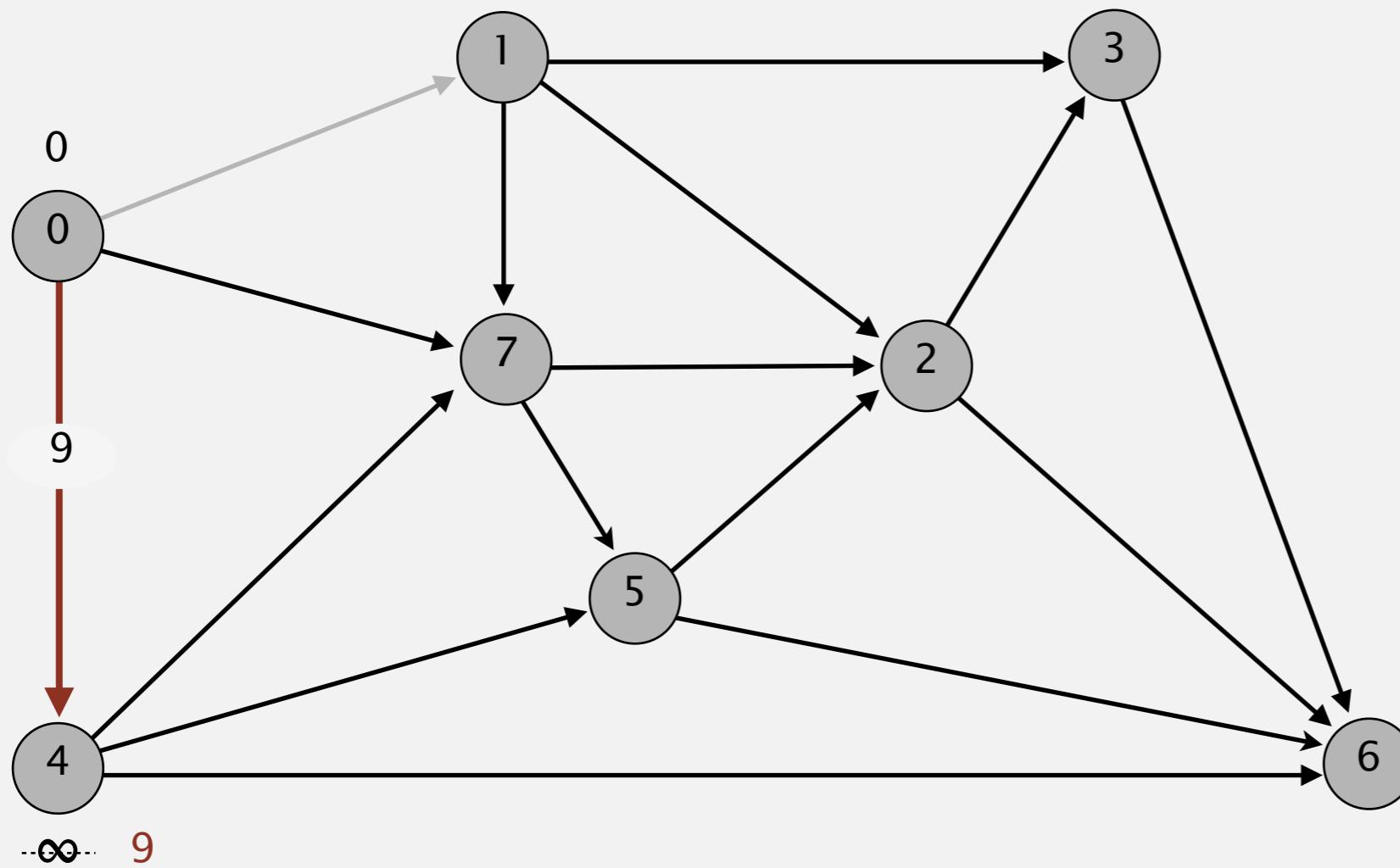
pass 0

0→1 0→4 0→7 1→2 1→3 1→7 2→3 2→6 3→6 4→5 4→6 4→7 5→2 5→6 7→5 7→2



Bellman-Ford algorithm demo

Repeat V times: relax all E edges.



v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0→1
2	∞	
3	∞	
4	9.0	0→4
5	∞	
6	∞	
7	∞	

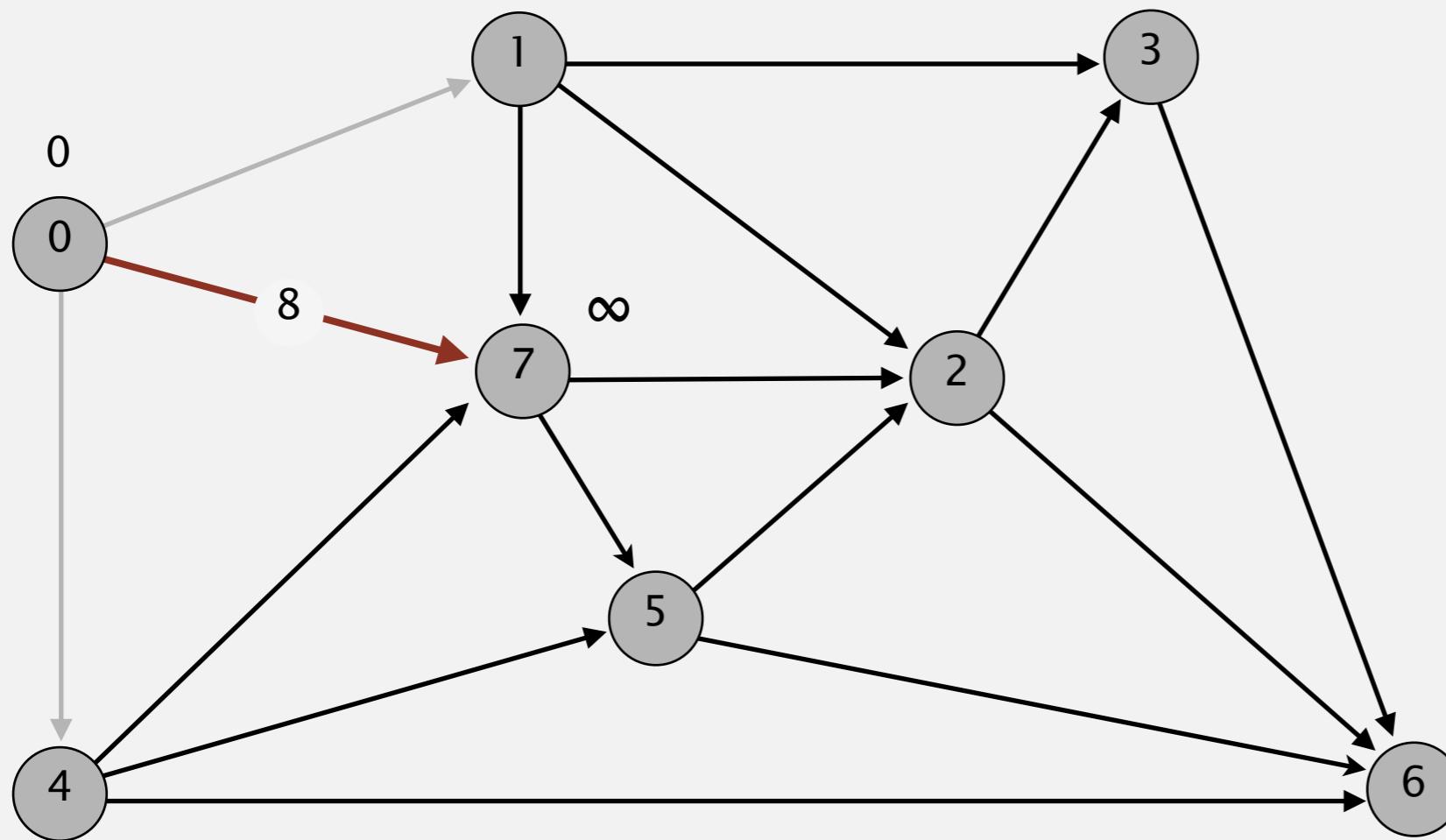
pass 0

0→1 0→4 0→7 1→2 1→3 1→7 2→3 2→6 3→6 4→5 4→6 4→7 5→2 5→6 7→5 7→2



Bellman-Ford algorithm demo

Repeat V times: relax all E edges.



v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0→1
2		
3		
4	9.0	0→4
5		
6		
7		

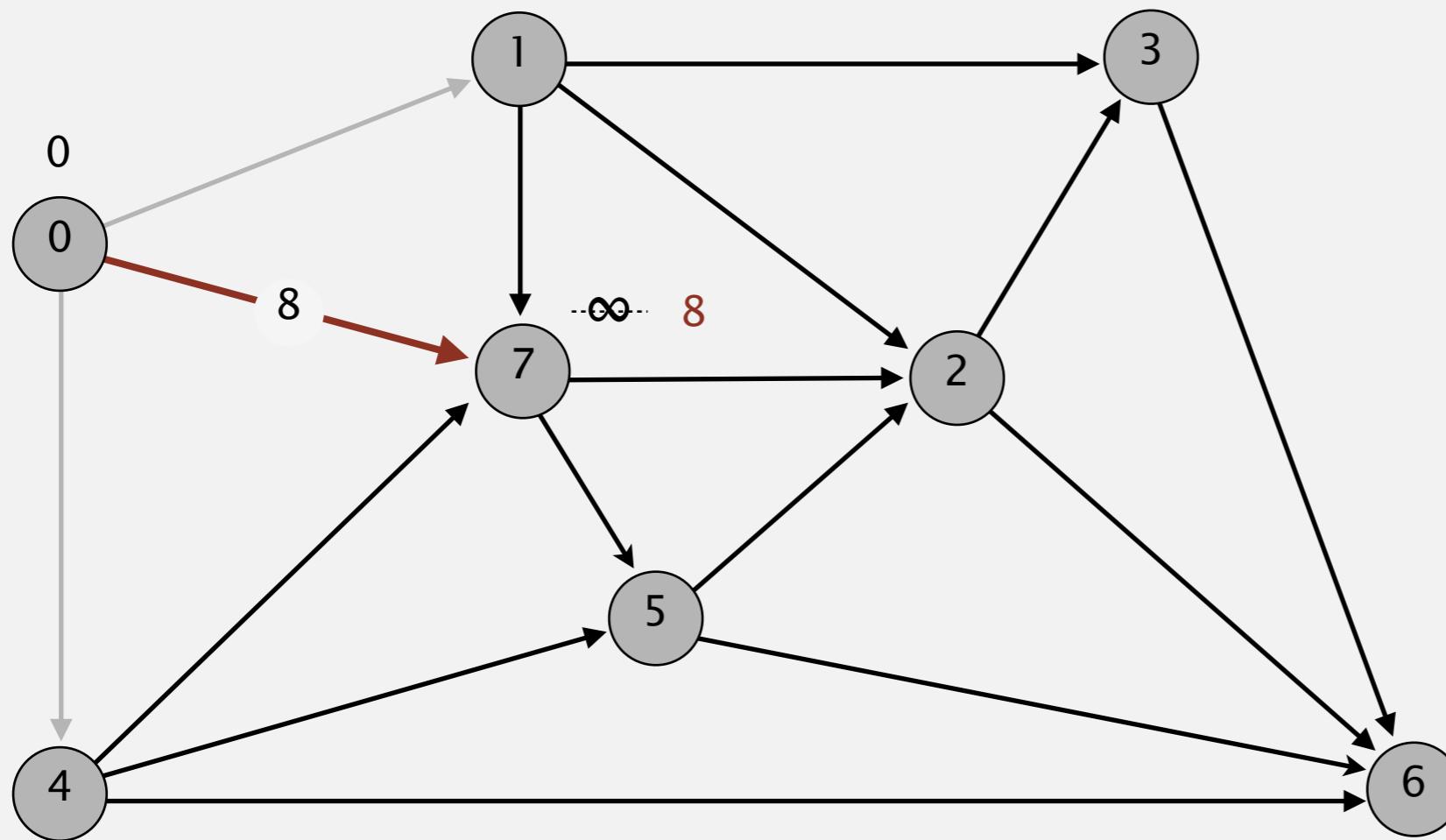
pass 0

0→1 0→4 0→7 1→2 1→3 1→7 2→3 2→6 3→6 4→5 4→6 4→7 5→2 5→6 7→5 7→2



Bellman-Ford algorithm demo

Repeat V times: relax all E edges.



v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0→1
2		
3		
4	9.0	0→4
5		
6		
7	8.0	0→7

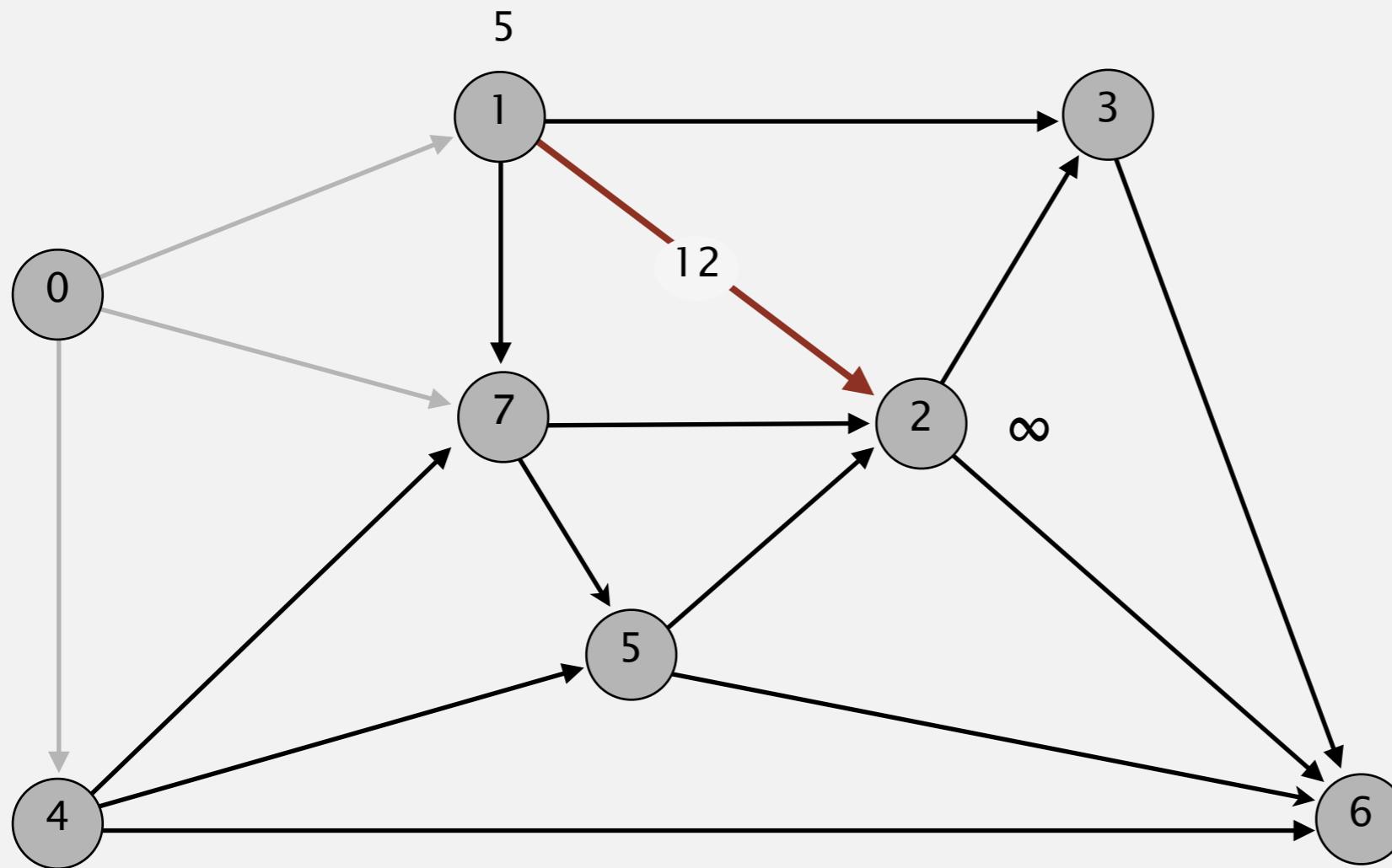
pass 0

0→1 0→4 0→7 1→2 1→3 1→7 2→3 2→6 3→6 4→5 4→6 4→7 5→2 5→6 7→5 7→2



Bellman-Ford algorithm demo

Repeat V times: relax all E edges.



v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0→1
2		
3		
4	9.0	0→4
5		
6		
7	8.0	0→7

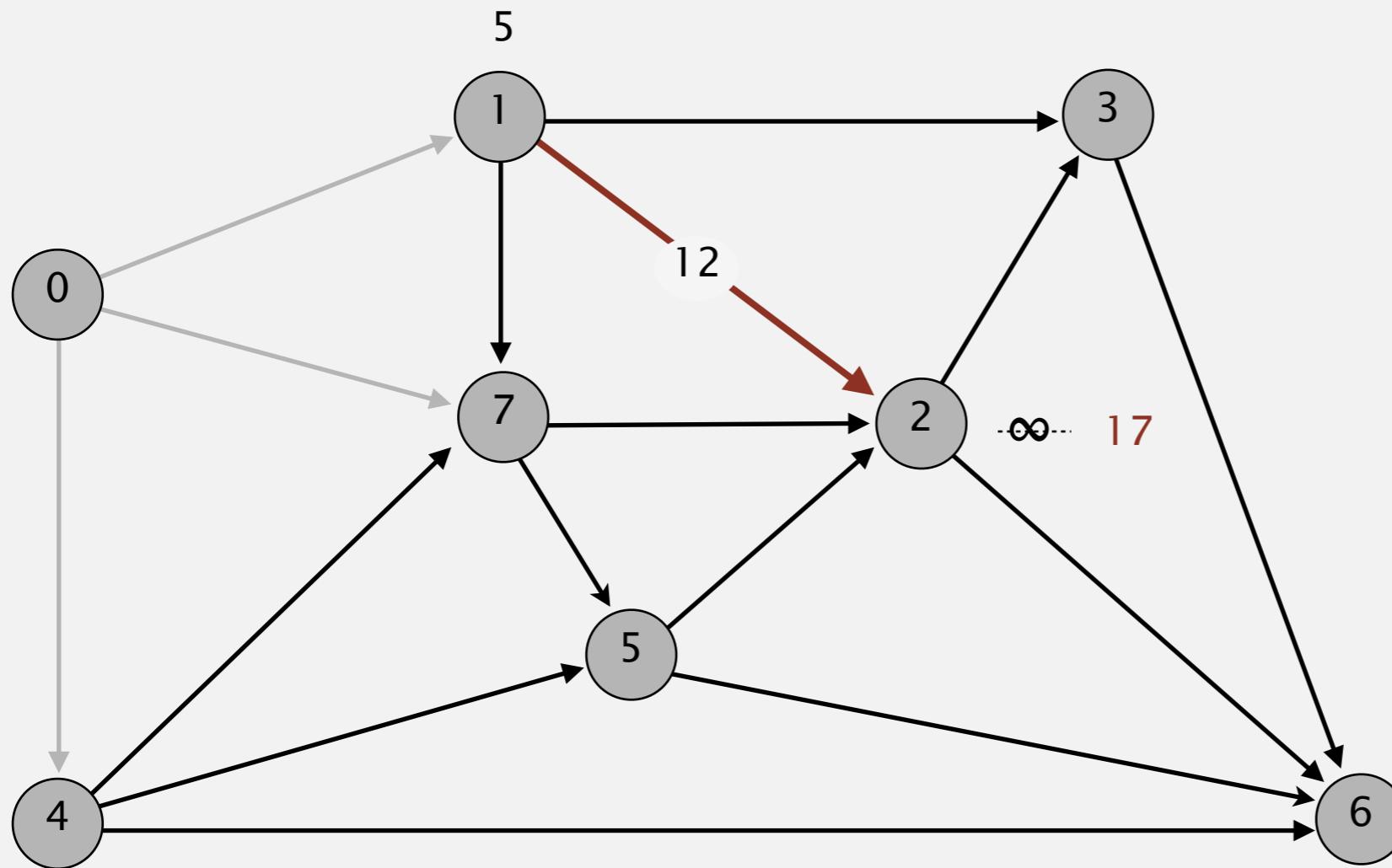
pass 0

0→1 0→4 0→7 1→2 1→3 1→7 2→3 2→6 3→6 4→5 4→6 4→7 5→2 5→6 7→5 7→2



Bellman-Ford algorithm demo

Repeat V times: relax all E edges.



v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0→1
2	17.0	1→2
3		
4	9.0	0→4
5		
6		
7	8.0	0→7

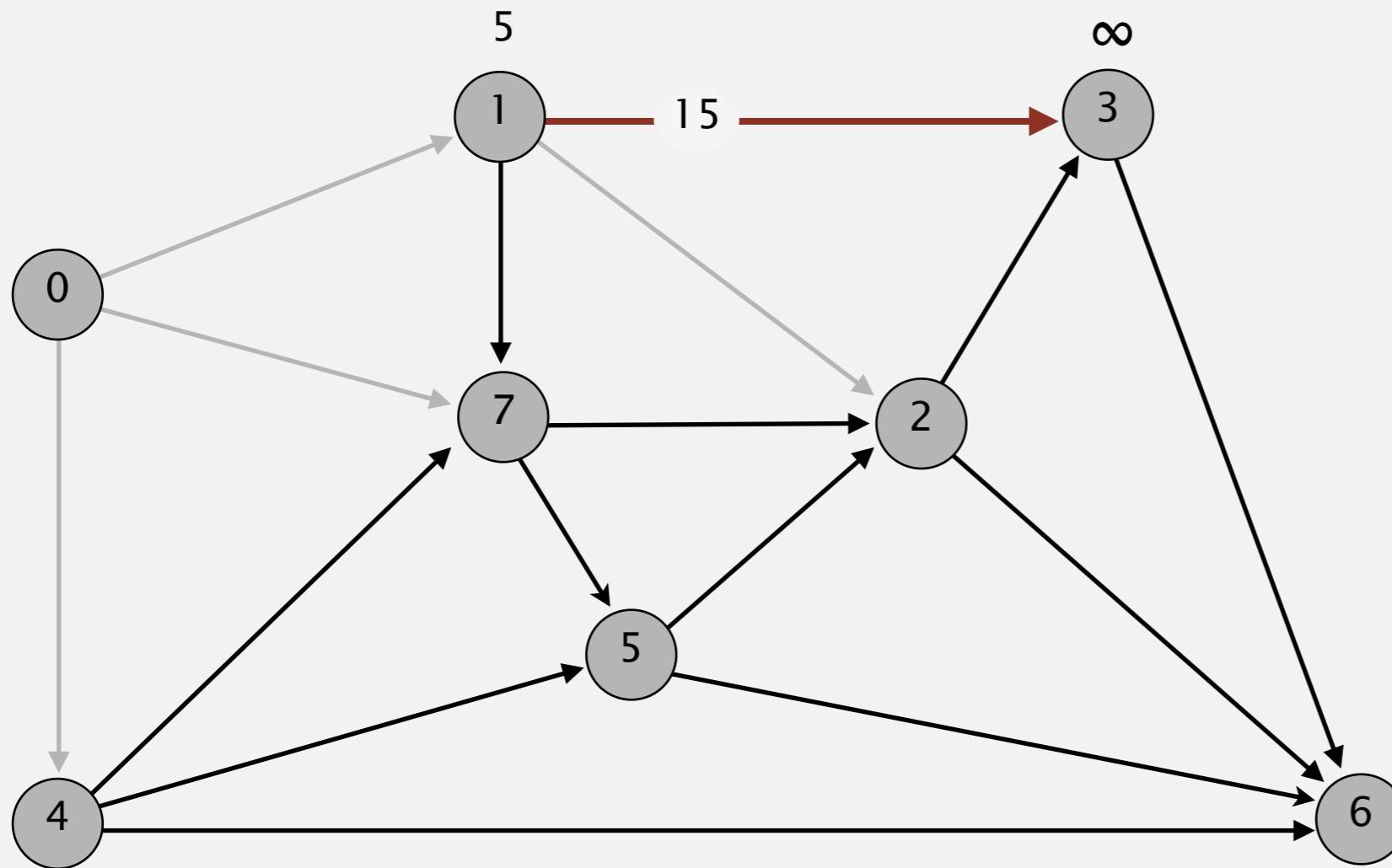
pass 0

0→1 0→4 0→7 1→2 1→3 1→7 2→3 2→6 3→6 4→5 4→6 4→7 5→2 5→6 7→5 7→2



Bellman-Ford algorithm demo

Repeat V times: relax all E edges.



v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0→1
2	17.0	1→2
3		
4	9.0	0→4
5		
6		
7	8.0	0→7

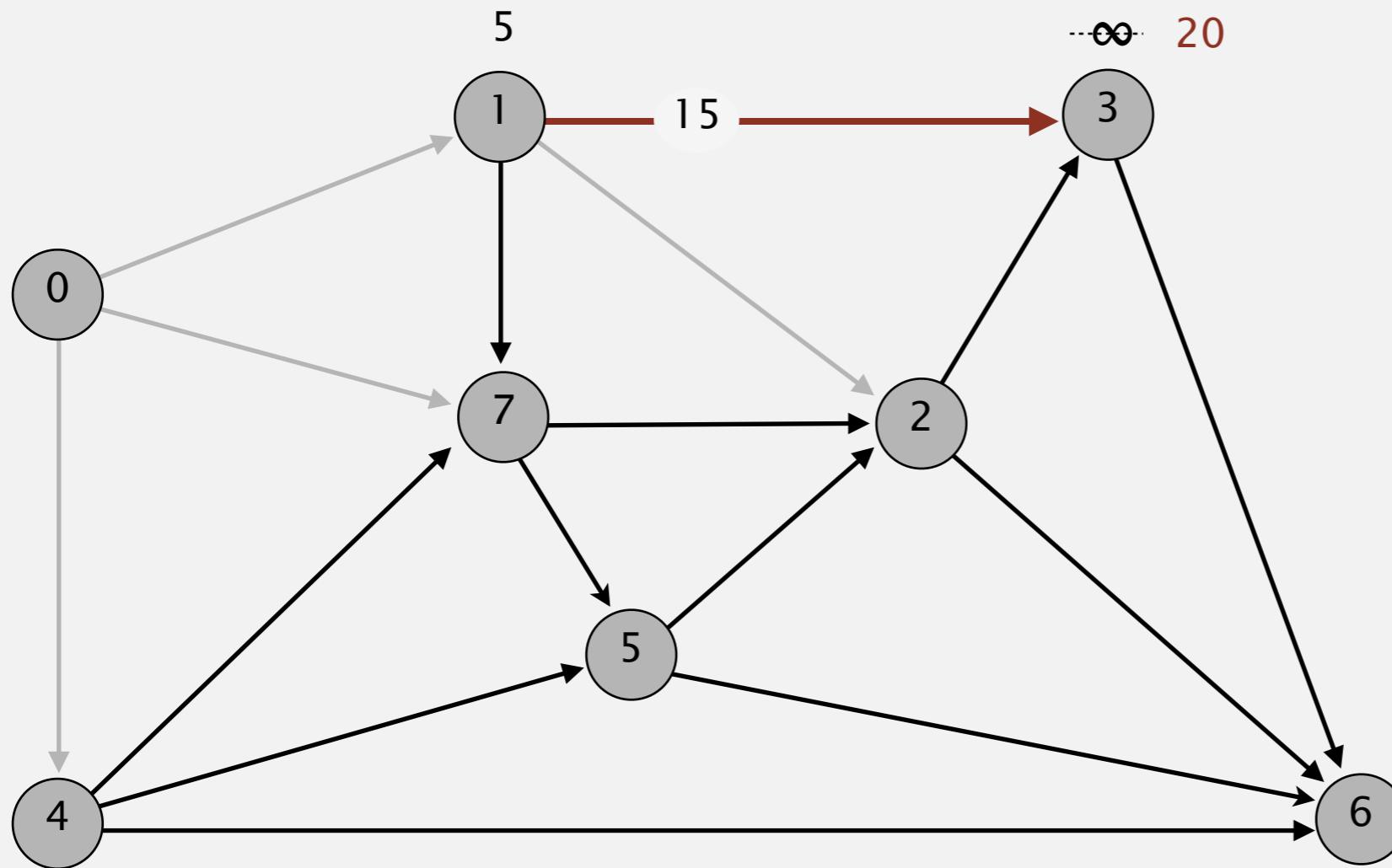
pass 0

0→1 0→4 0→7 1→2 1→3 1→7 2→3 2→6 3→6 4→5 4→6 4→7 5→2 5→6 7→5 7→2



Bellman-Ford algorithm demo

Repeat V times: relax all E edges.



v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0→1
2	17.0	1→2
3	20.0	1→3
4	9.0	0→4
5		
6		
7	8.0	0→7

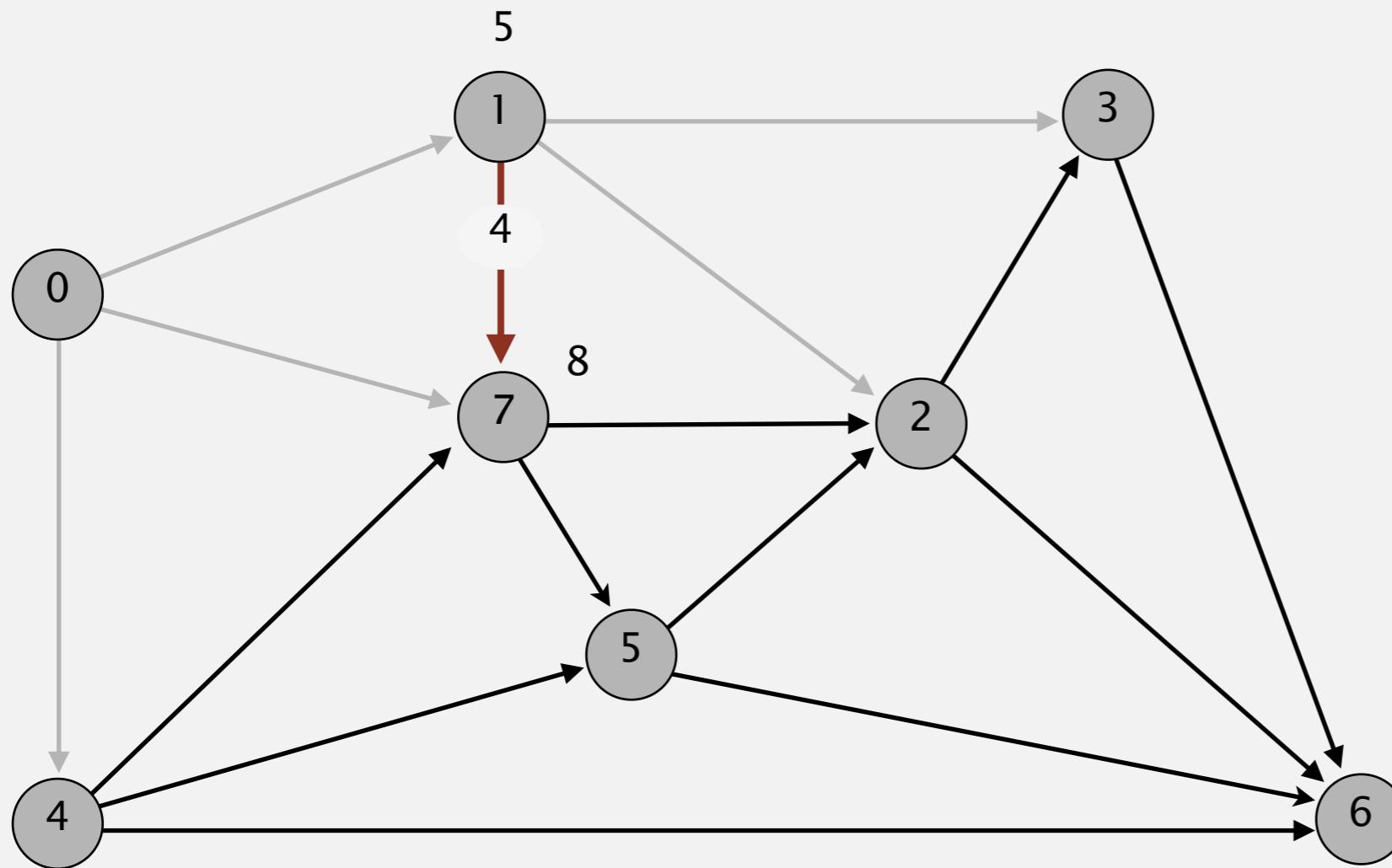
pass 0

0→1 0→4 0→7 1→2 1→3 1→7 2→3 2→6 3→6 4→5 4→6 4→7 5→2 5→6 7→5 7→2



Bellman-Ford algorithm demo

Repeat V times: relax all E edges.



v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0→1
2	17.0	1→2
3	20.0	1→3
4	9.0	0→4
5		
6		
7	8.0	0→7

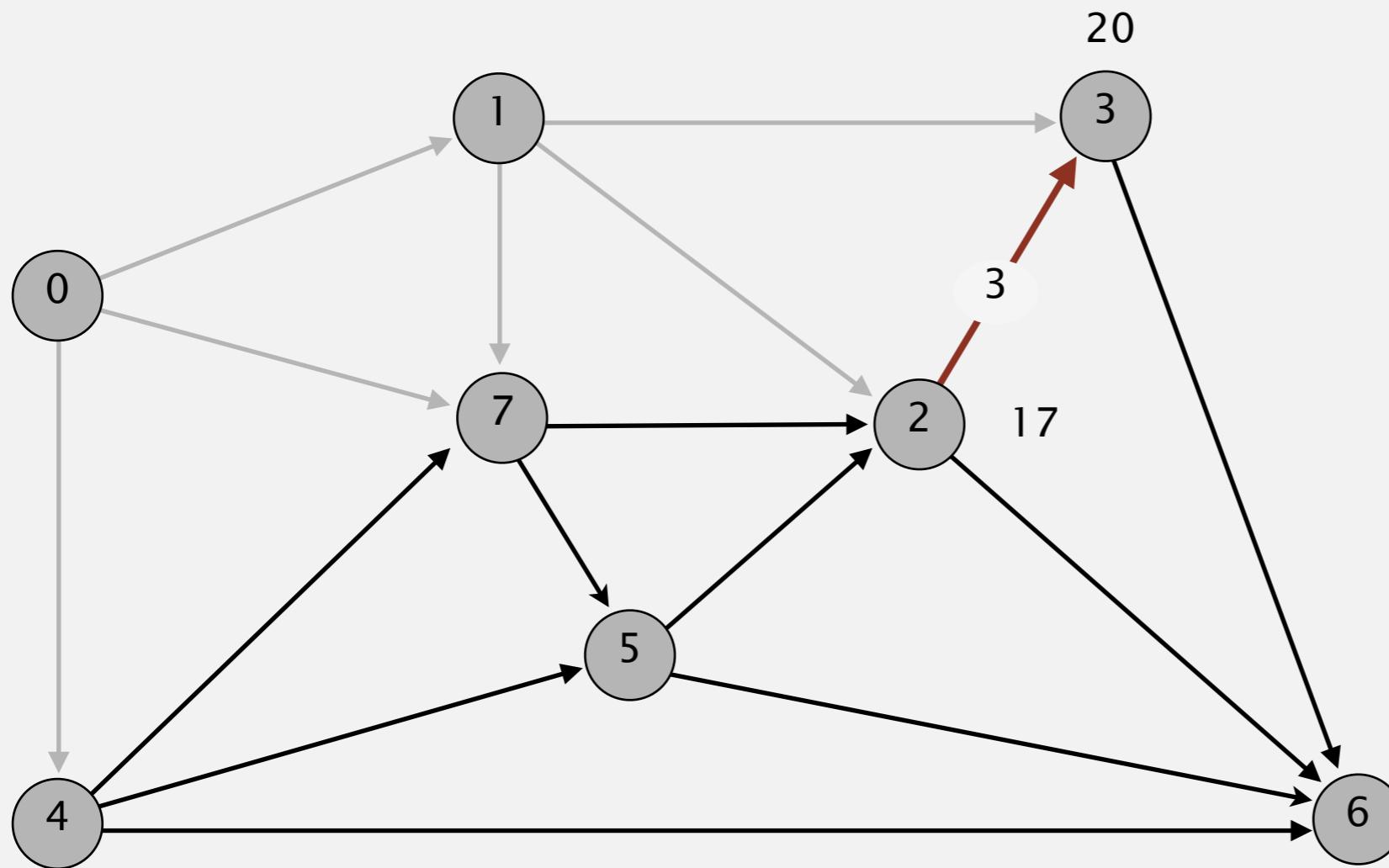
pass 0

0→1 0→4 0→7 1→2 1→3 1→7 2→3 2→6 3→6 4→5 4→6 4→7 5→2 5→6 7→5 7→2



Bellman-Ford algorithm demo

Repeat V times: relax all E edges.



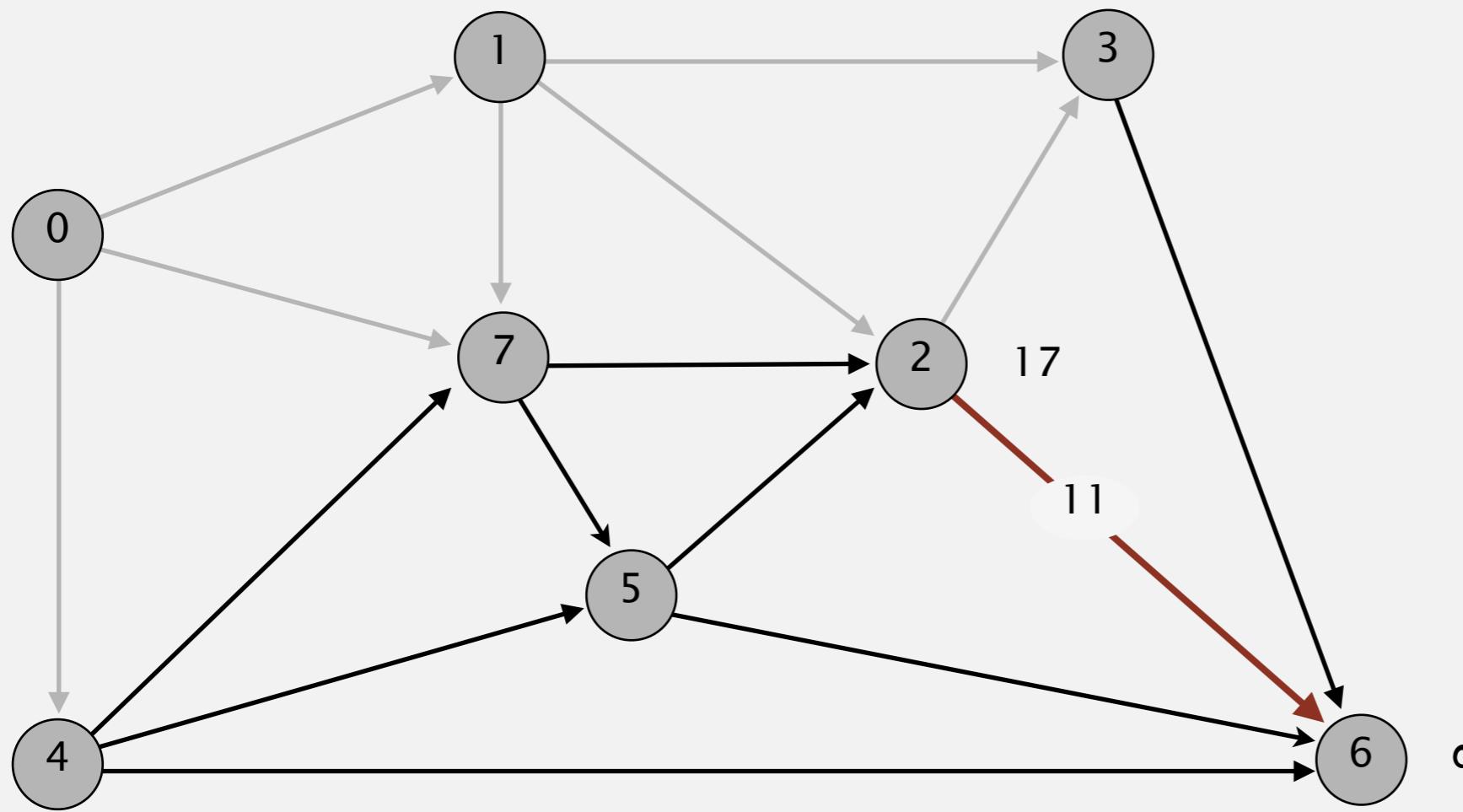
v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0→1
2	17.0	1→2
3	20.0	1→3
4	9.0	0→4
5		
6		
7	8.0	0→7

pass 0

0→1 0→4 0→7 1→2 1→3 1→7 2→3 2→6 3→6 4→5 4→6 4→7 5→2 5→6 7→5 7→2

Bellman-Ford algorithm demo

Repeat V times: relax all E edges.



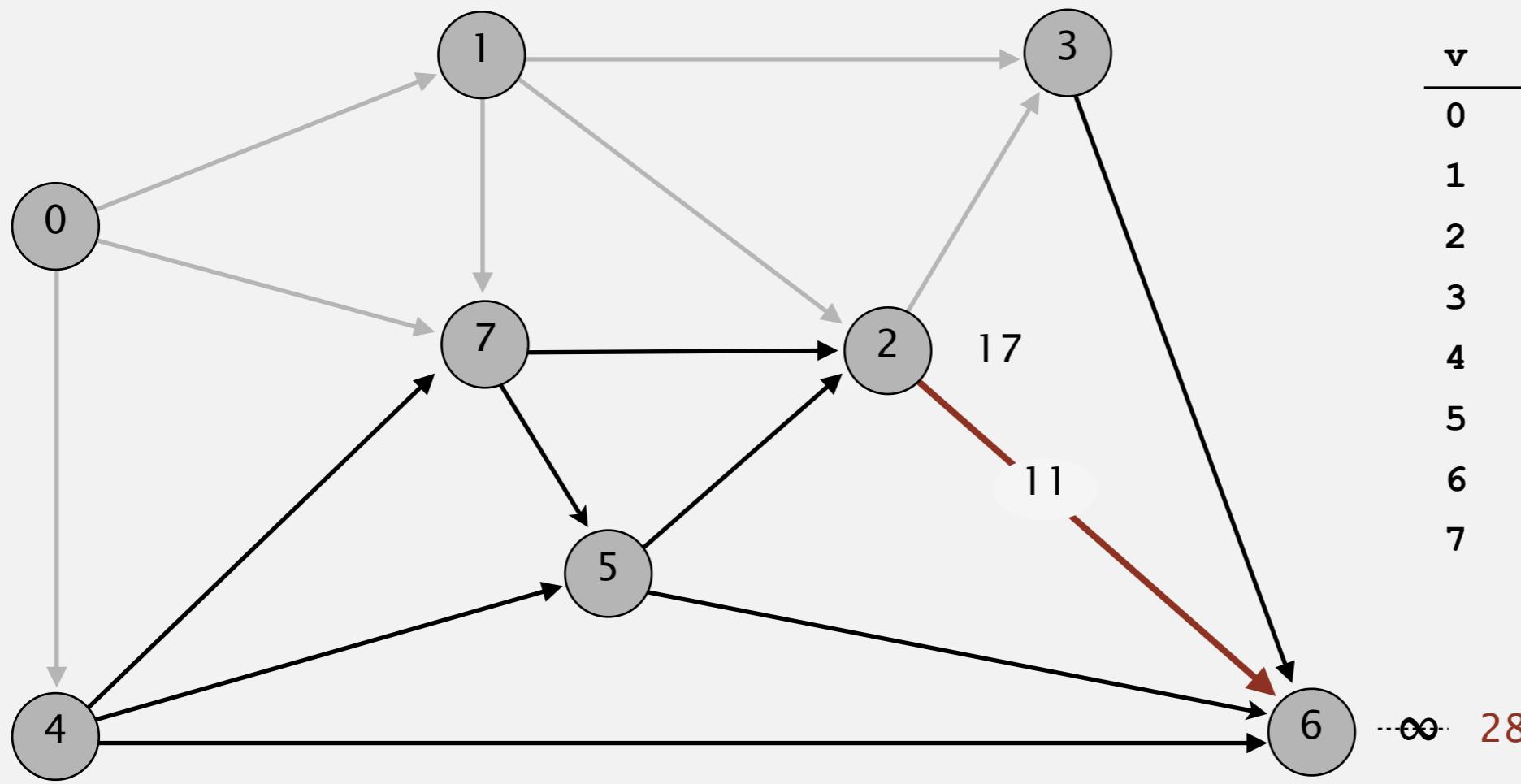
v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0→1
2	17.0	1→2
3	20.0	1→3
4	9.0	0→4
5		
6		
7	8.0	0→7

pass 0

0→1 0→4 0→7 1→2 1→3 1→7 2→3 2→6 3→6 4→5 4→6 4→7 5→2 5→6 7→5 7→2

Bellman-Ford algorithm demo

Repeat V times: relax all E edges.



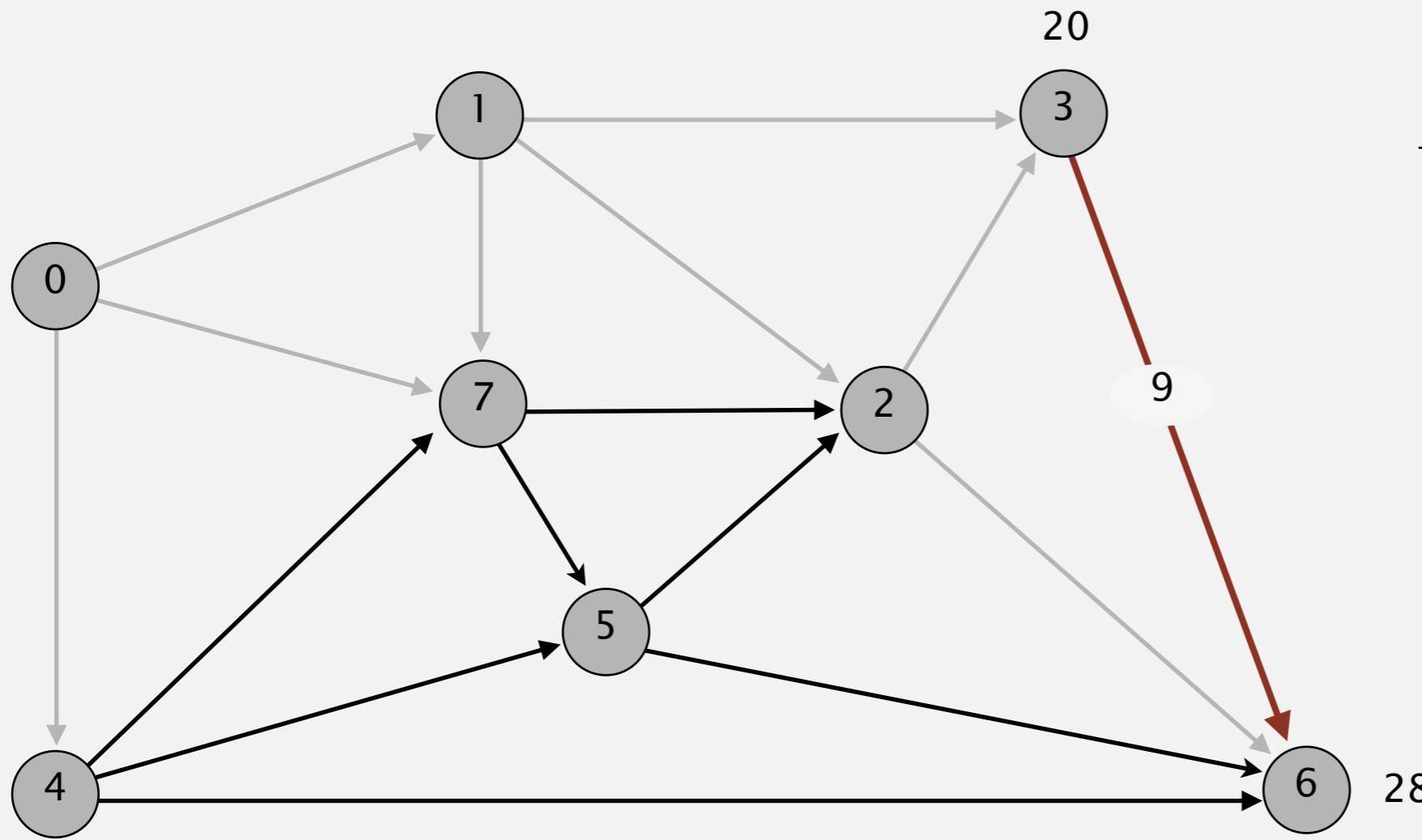
pass 0

0→1 0→4 0→7 1→2 1→3 1→7 2→3 2→6 3→6 4→5 4→6 4→7 5→2 5→6 7→5 7→2



Bellman-Ford algorithm demo

Repeat V times: relax all E edges.



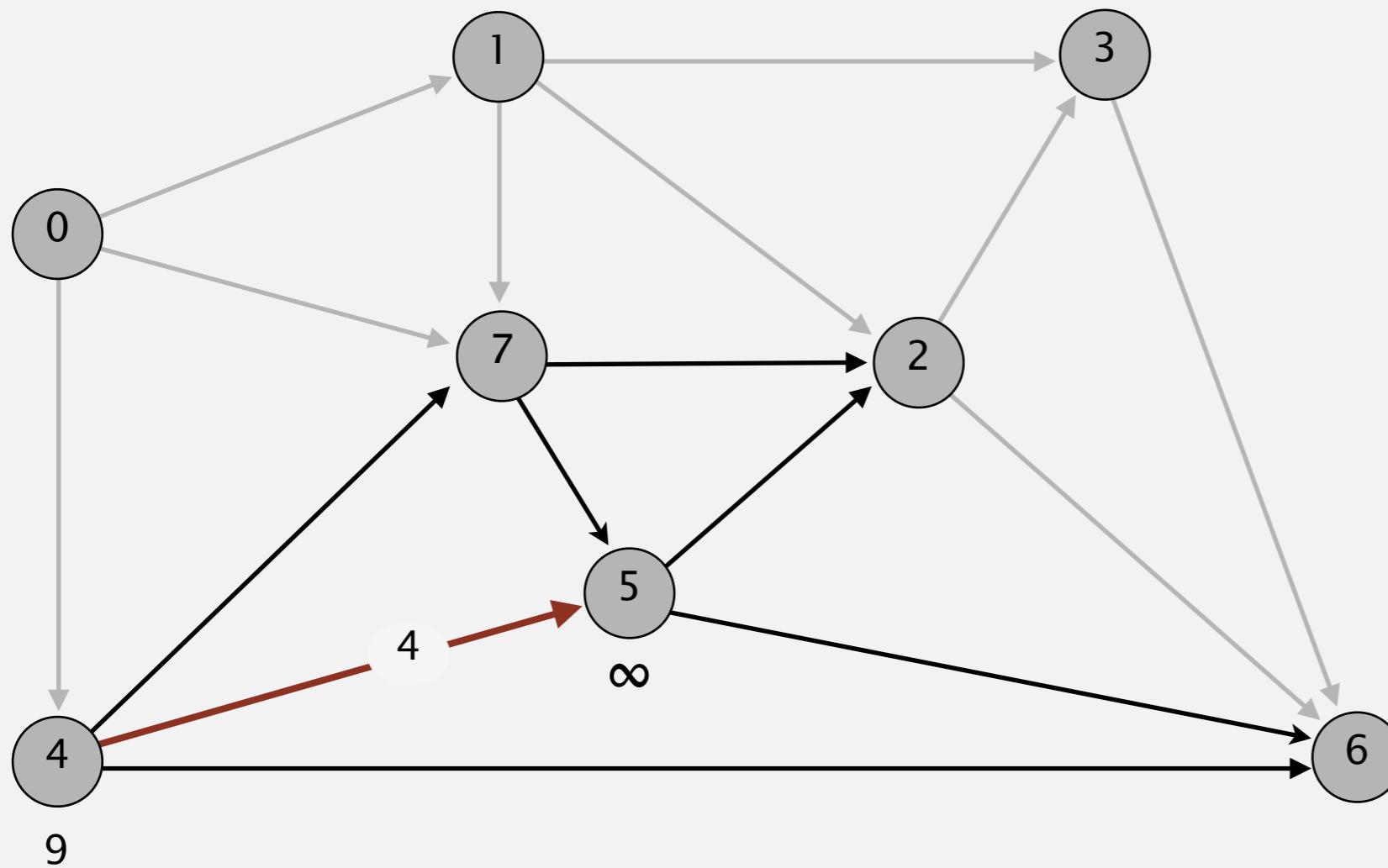
v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0→1
2	17.0	1→2
3	20.0	1→3
4	9.0	0→4
5		
6	28.0	2→6
7	8.0	0→7

pass 0

0→1 0→4 0→7 1→2 1→3 1→7 2→3 2→6 3→6 4→5 4→6 4→7 5→2 5→6 7→5 7→2

Bellman-Ford algorithm demo

Repeat V times: relax all E edges.



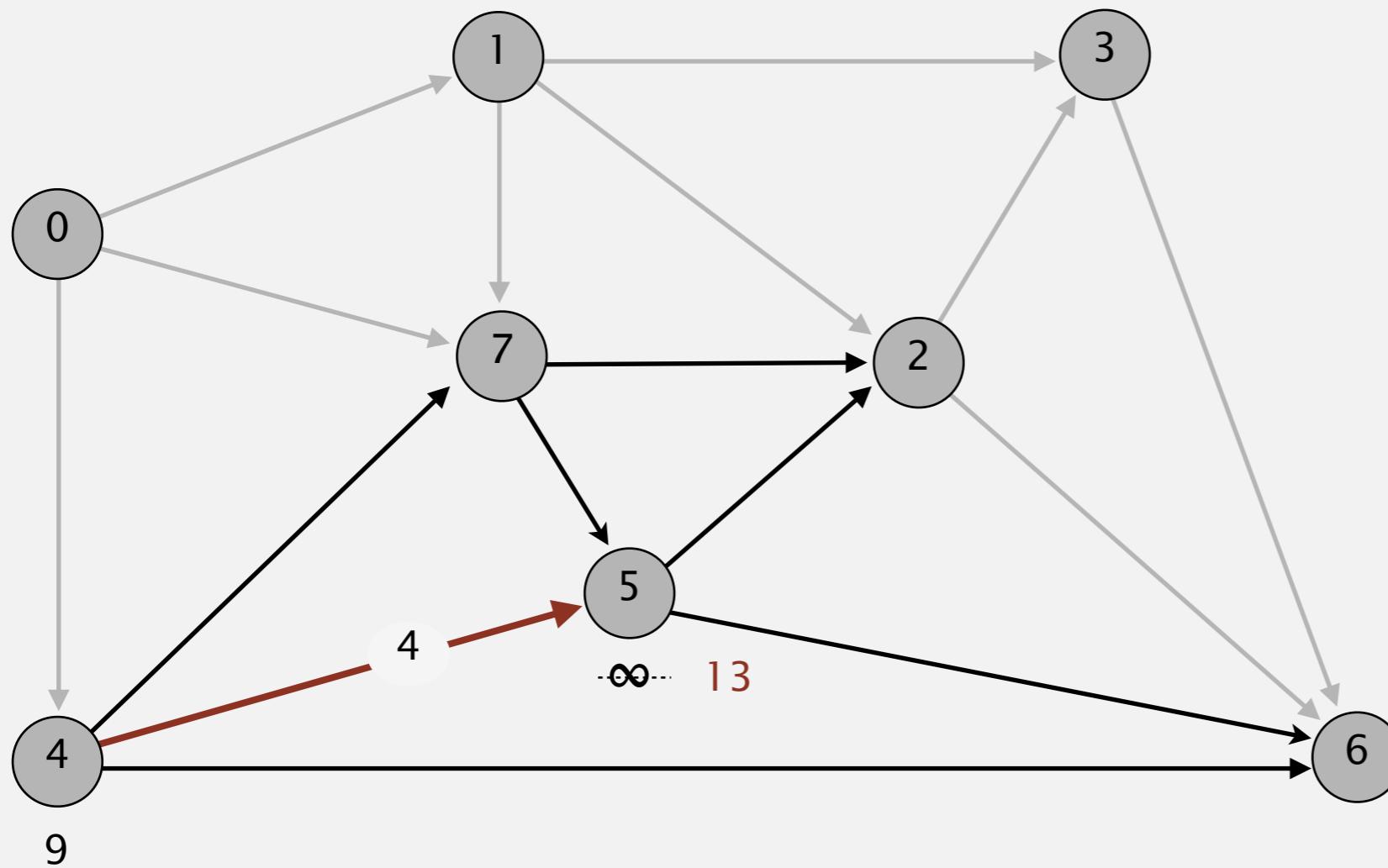
v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0→1
2	17.0	1→2
3	20.0	1→3
4	9.0	0→4
5		
6	28.0	2→6
7	8.0	0→7

pass 0

0→1 0→4 0→7 1→2 1→3 1→7 2→3 2→6 3→6 4→5 4→6 4→7 5→2 5→6 7→5 7→2

Bellman-Ford algorithm demo

Repeat V times: relax all E edges.



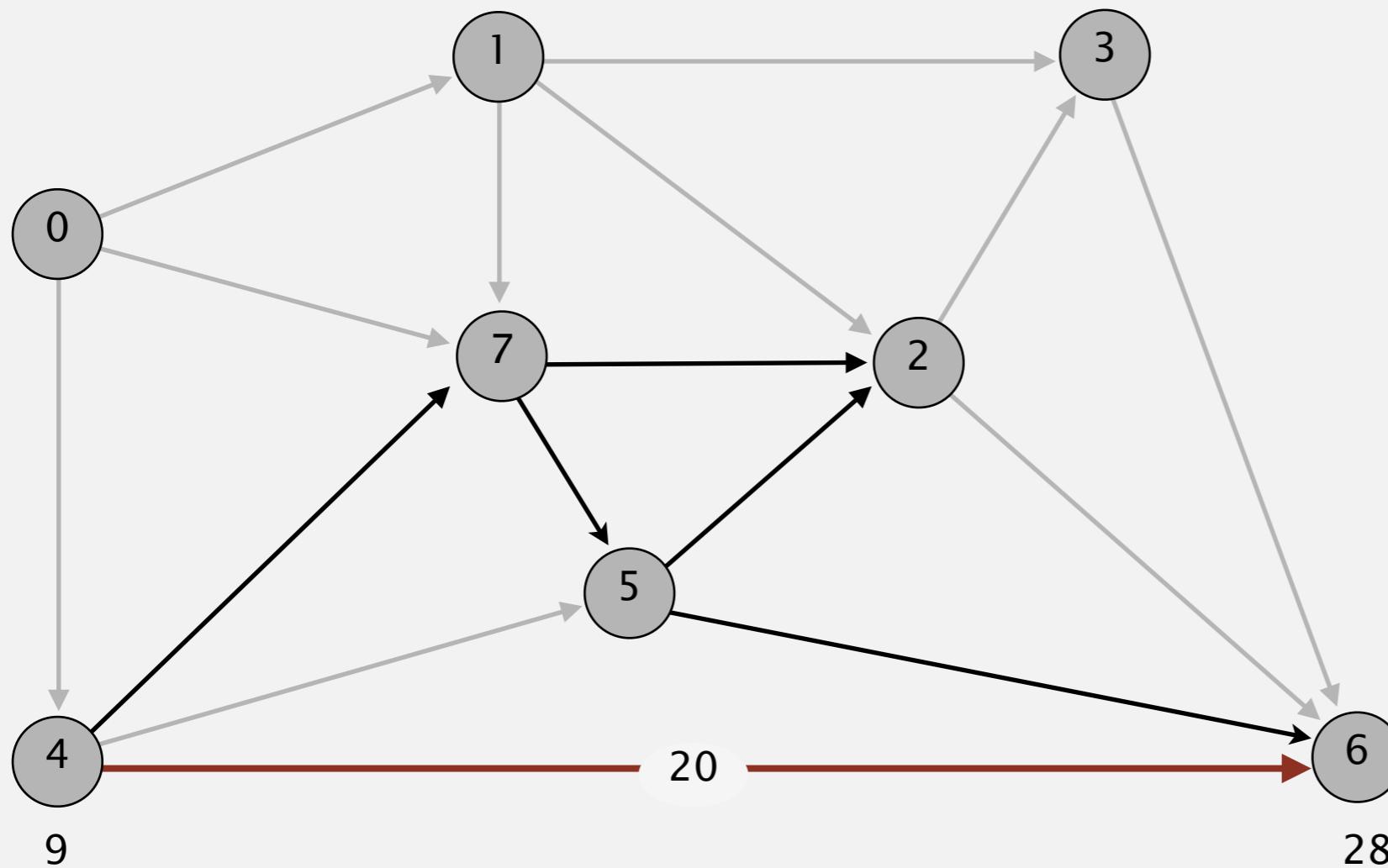
v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0→1
2	17.0	1→2
3	20.0	1→3
4	9.0	0→4
5	13.0	4→5
6	28.0	2→6
7	8.0	0→7

pass 0

0→1 0→4 0→7 1→2 1→3 1→7 2→3 2→6 3→6 4→5 4→6 4→7 5→2 5→6 7→5 7→2

Bellman-Ford algorithm demo

Repeat V times: relax all E edges.



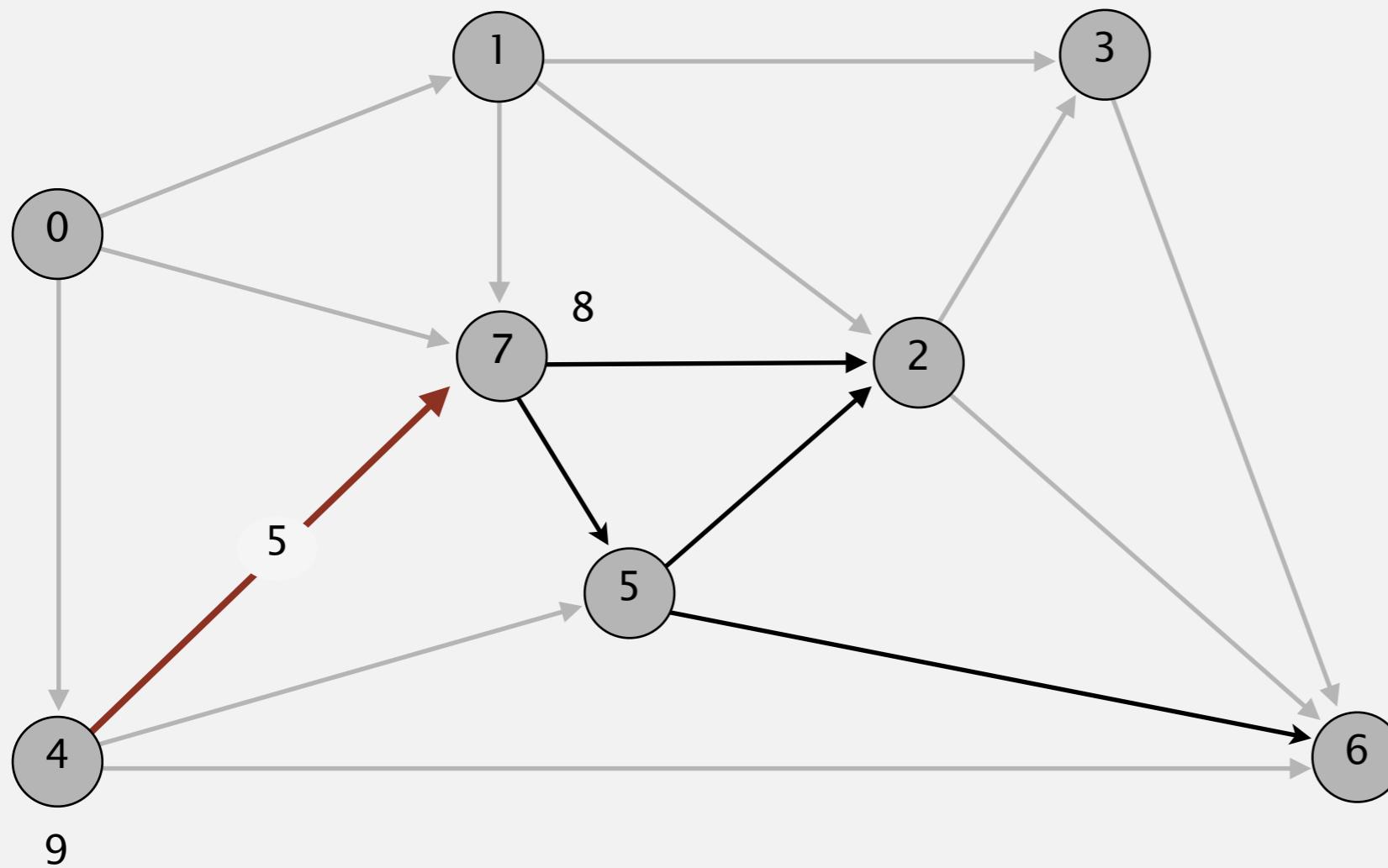
v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0→1
2	17.0	1→2
3	20.0	1→3
4	9.0	0→4
5	13.0	4→5
6	28.0	2→6
7	8.0	0→7

pass 0

0→1 0→4 0→7 1→2 1→3 1→7 2→3 2→6 3→6 4→5 4→6 4→7 5→2 5→6 7→5 7→2

Bellman-Ford algorithm demo

Repeat V times: relax all E edges.



pass 0

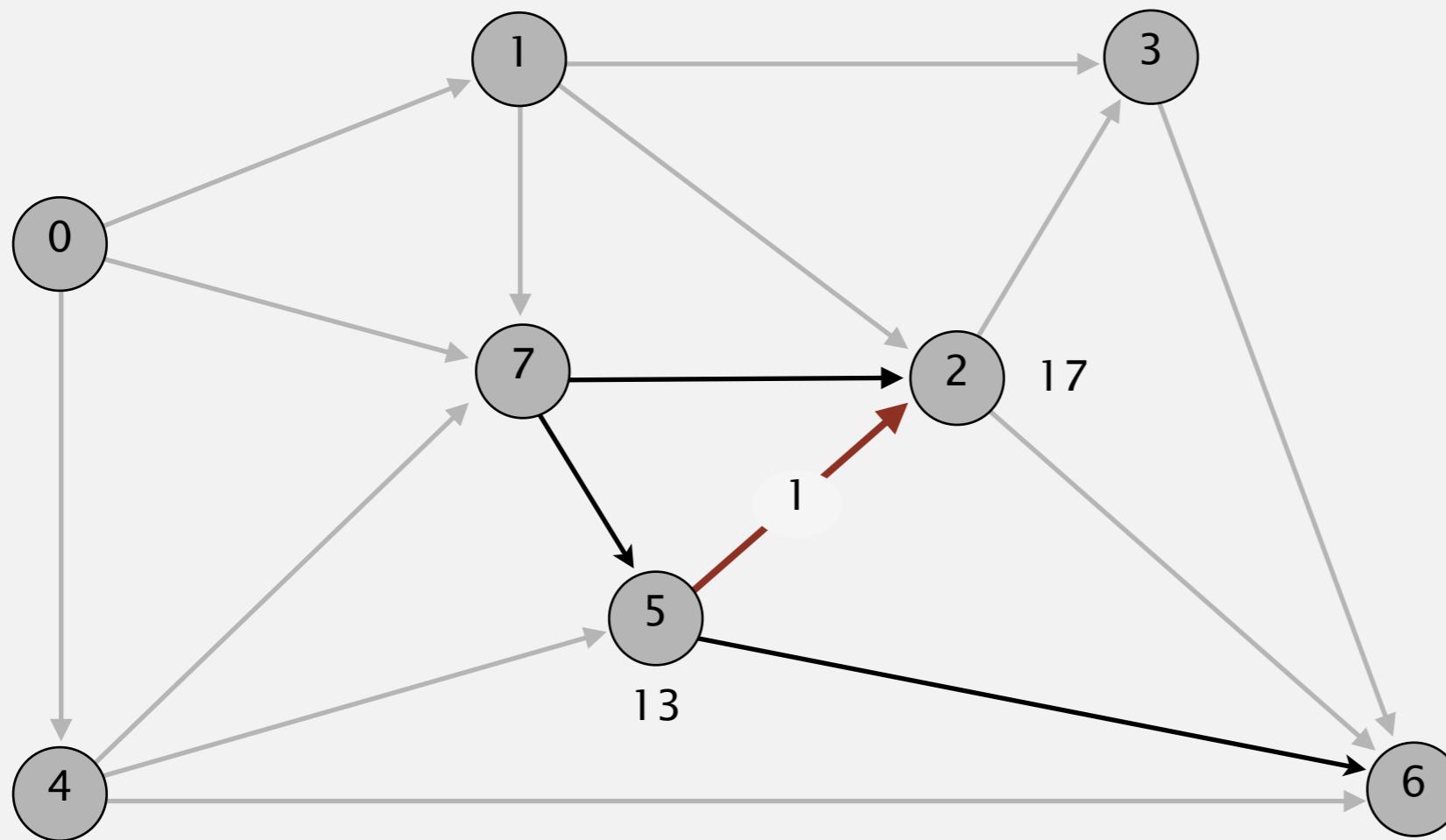
0→1 0→4 0→7 1→2 1→3 1→7 2→3 2→6 3→6 4→5 4→6 4→7 5→2 5→6 7→5 7→2



v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0→1
2	17.0	1→2
3	20.0	1→3
4	9.0	0→4
5	13.0	4→5
6	28.0	2→6
7	8.0	0→7

Bellman-Ford algorithm demo

Repeat V times: relax all E edges.



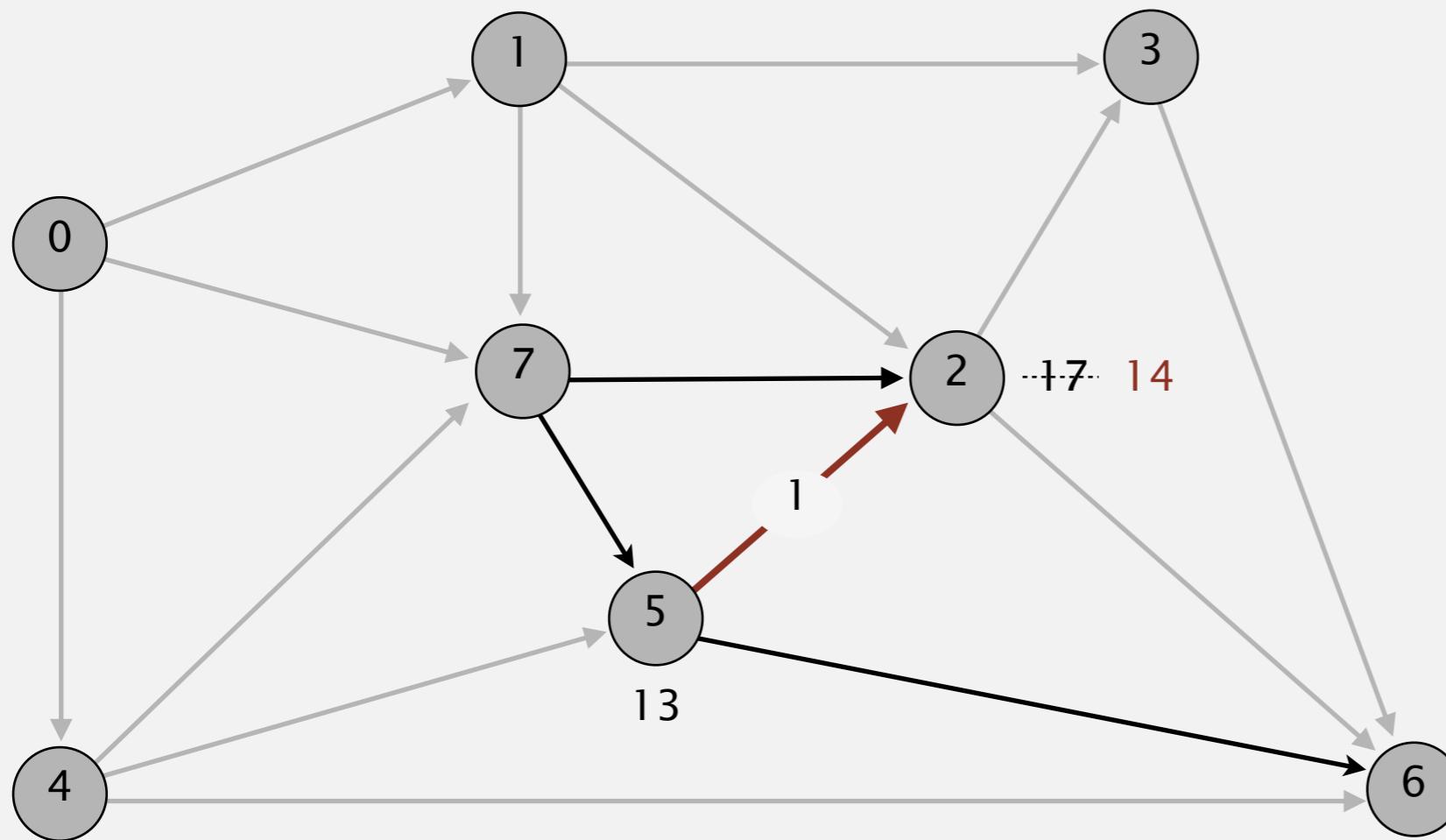
v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0→1
2	17.0	1→2
3	20.0	1→3
4	9.0	0→4
5	13.0	4→5
6	28.0	2→6
7	8.0	0→7

pass 0

0→1 0→4 0→7 1→2 1→3 1→7 2→3 2→6 3→6 4→5 4→6 4→7 5→2 5→6 7→5 7→2

Bellman-Ford algorithm demo

Repeat V times: relax all E edges.



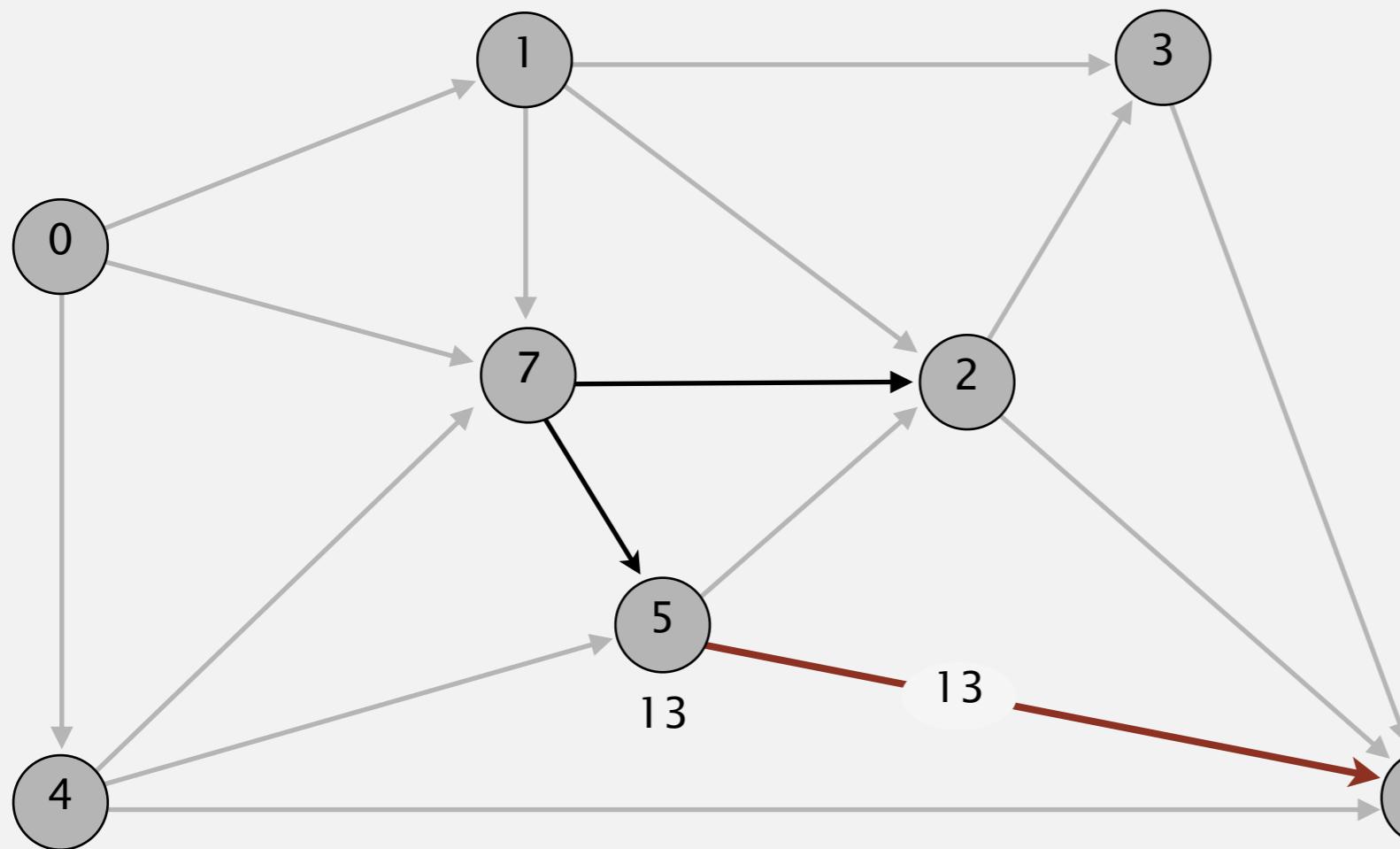
v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0→1
2	14.0	5→2
3	20.0	1→3
4	9.0	0→4
5	13.0	4→5
6	28.0	2→6
7	8.0	0→7

pass 0

0→1 0→4 0→7 1→2 1→3 1→7 2→3 2→6 3→6 4→5 4→6 4→7 5→2 5→6 7→5 7→2

Bellman-Ford algorithm demo

Repeat V times: relax all E edges.



v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0→1
2	14.0	5→2
3	20.0	1→3
4	9.0	0→4
5	13.0	4→5
6	28.0	2→6
7	8.0	0→7

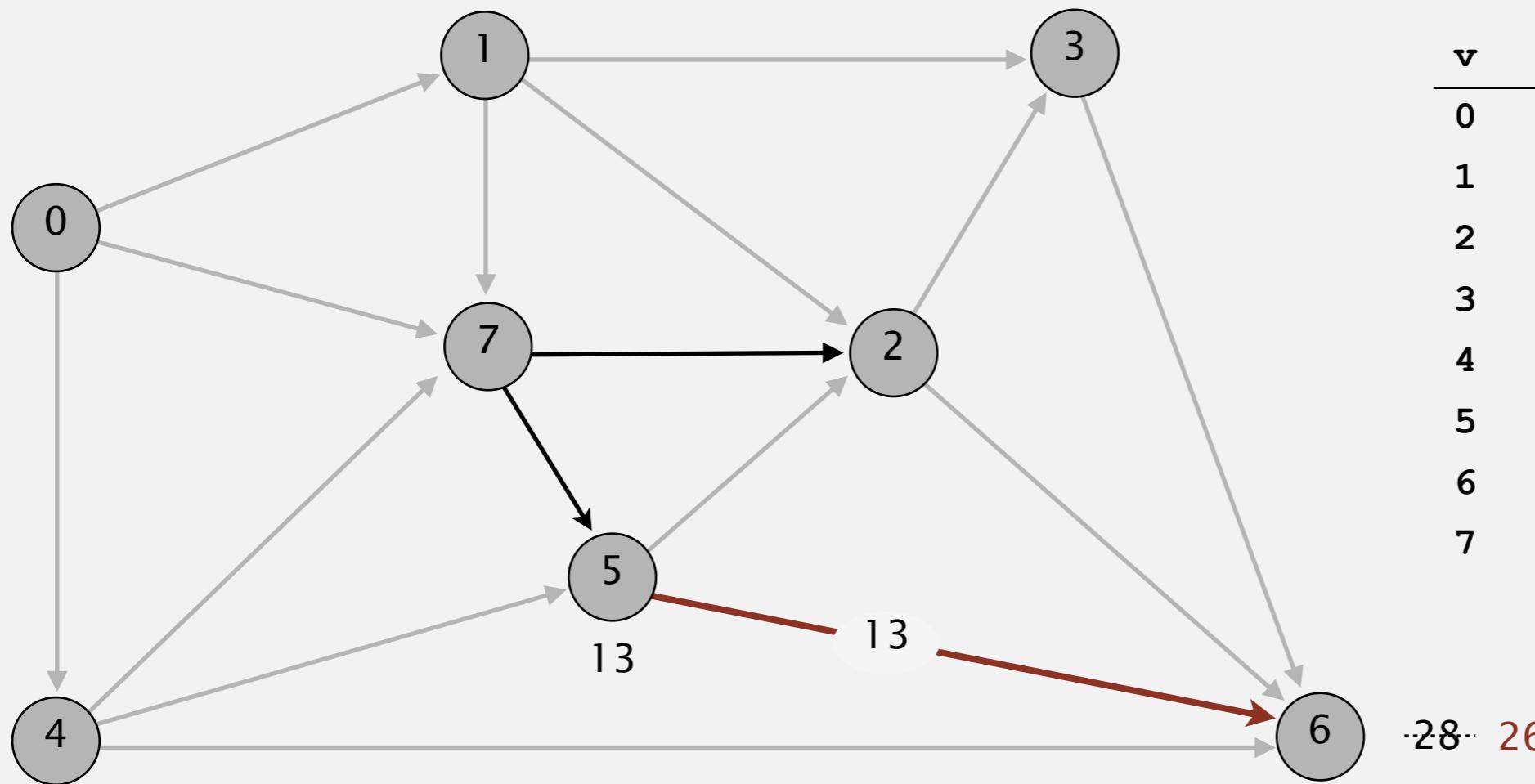
pass 0

0→1 0→4 0→7 1→2 1→3 1→7 2→3 2→6 3→6 4→5 4→6 4→7 5→2 5→6 7→5 7→2



Bellman-Ford algorithm demo

Repeat V times: relax all E edges.



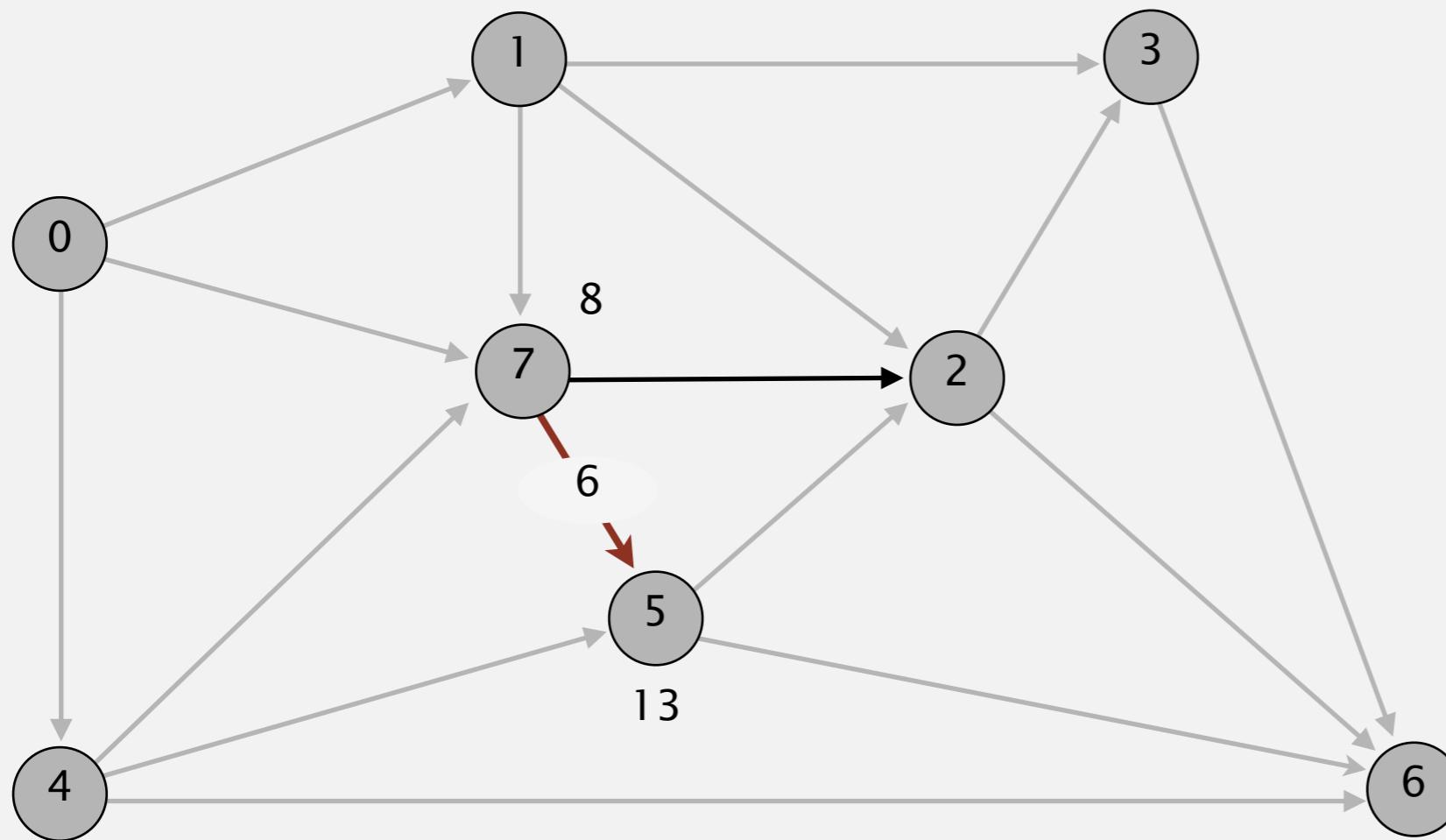
v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0→1
2	14.0	5→2
3	20.0	1→3
4	9.0	0→4
5	13.0	4→5
6	26.0	5→6
7	8.0	0→7

pass 0

0→1 0→4 0→7 1→2 1→3 1→7 2→3 2→6 3→6 4→5 4→6 4→7 5→2 5→6 7→5 7→2

Bellman-Ford algorithm demo

Repeat V times: relax all E edges.



v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0→1
2	14.0	5→2
3	20.0	1→3
4	9.0	0→4
5	13.0	4→5
6	26.0	5→6
7	8.0	0→7

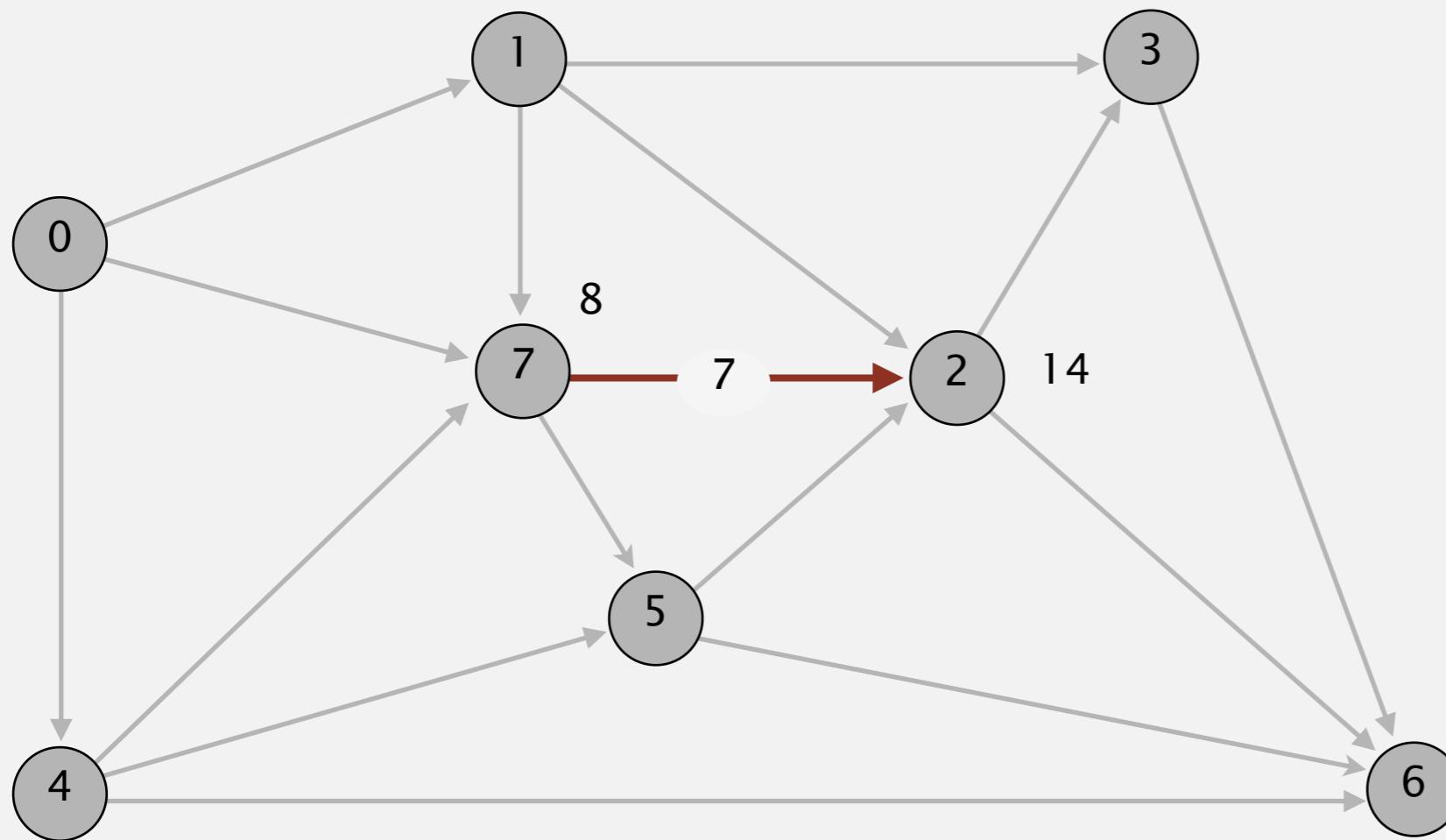
pass 0

0→1 0→4 0→7 1→2 1→3 1→7 2→3 2→6 3→6 4→5 4→6 4→7 5→2 5→6 7→5 7→2



Bellman-Ford algorithm demo

Repeat V times: relax all E edges.



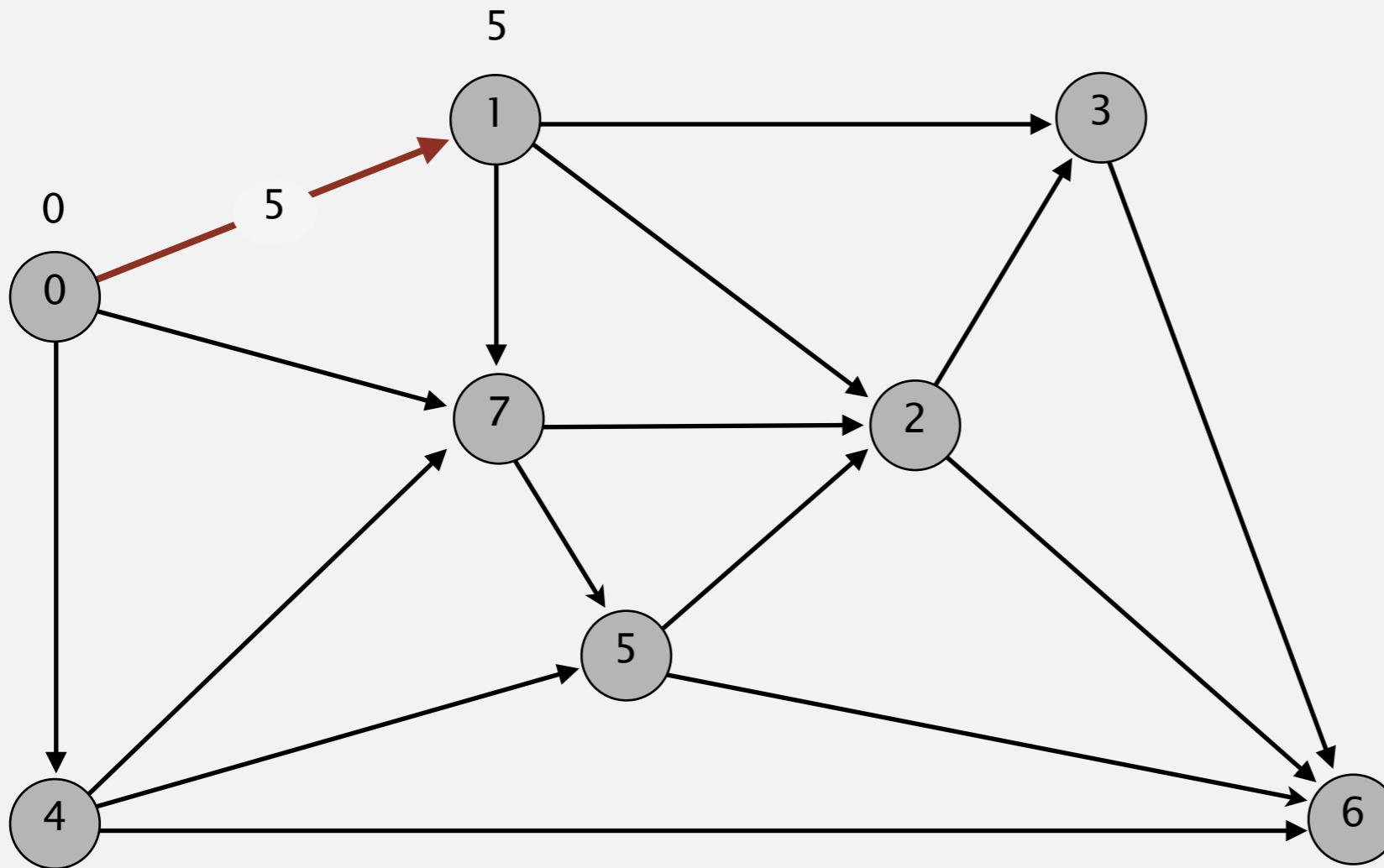
v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0→1
2	14.0	5→2
3	20.0	1→3
4	9.0	0→4
5	13.0	4→5
6	26.0	5→6
7	8.0	0→7

pass 0

0→1 0→4 0→7 1→2 1→3 1→7 2→3 2→6 3→6 4→5 4→6 4→7 5→2 5→6 7→5 7→2

Bellman-Ford algorithm demo

Repeat V times: relax all E edges.



v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0→1
2	14.0	5→2
3	20.0	1→3
4	9.0	0→4
5	13.0	4→5
6	26.0	5→6
7	8.0	0→7

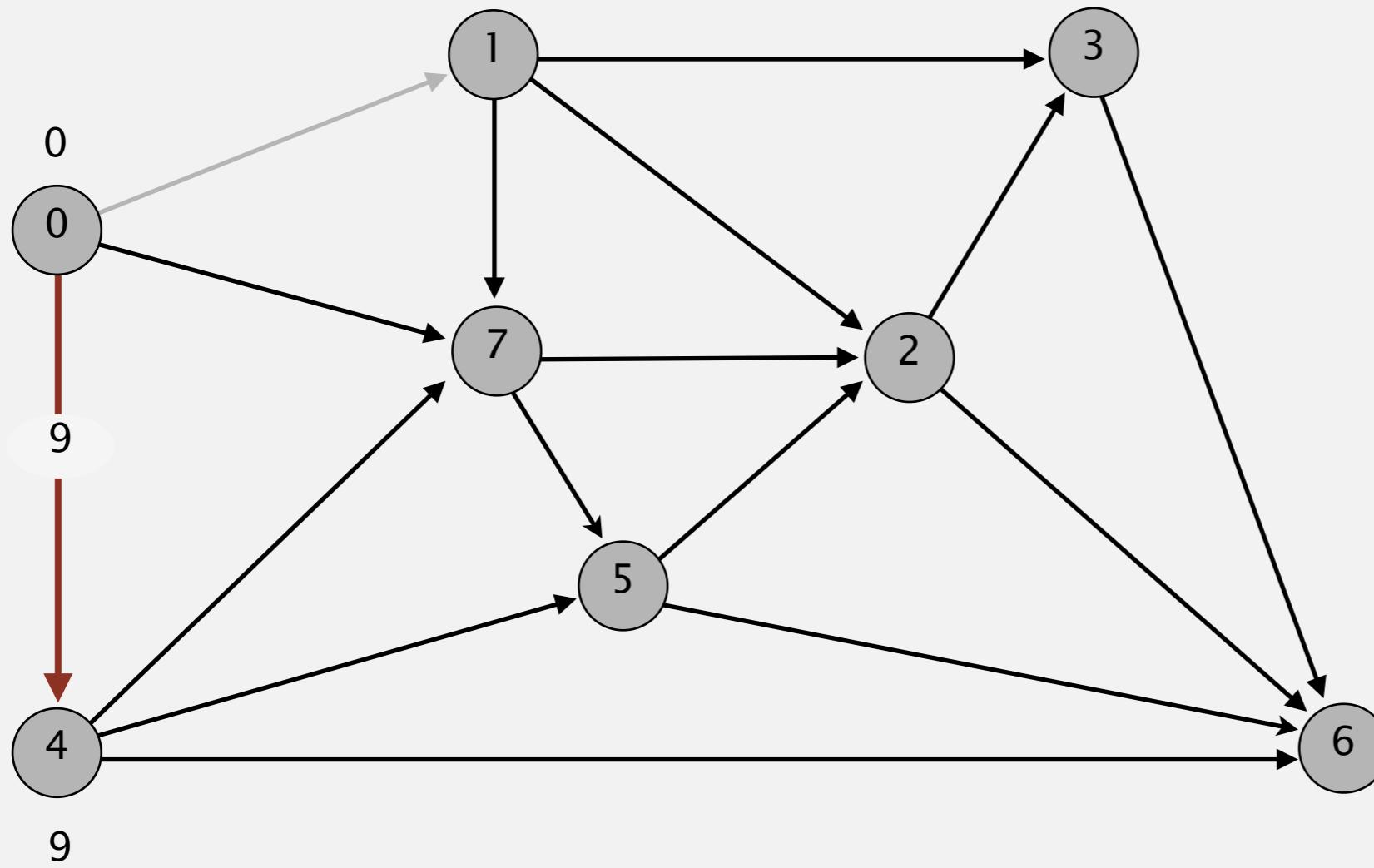
pass 1

0→1 0→4 0→7 1→2 1→3 1→7 2→3 2→6 3→6 4→5 4→6 4→7 5→2 5→6 7→5 7→2



Bellman-Ford algorithm demo

Repeat V times: relax all E edges.



v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0→1
2	14.0	5→2
3	20.0	1→3
4	9.0	0→4
5	13.0	4→5
6	26.0	5→6
7	8.0	0→7

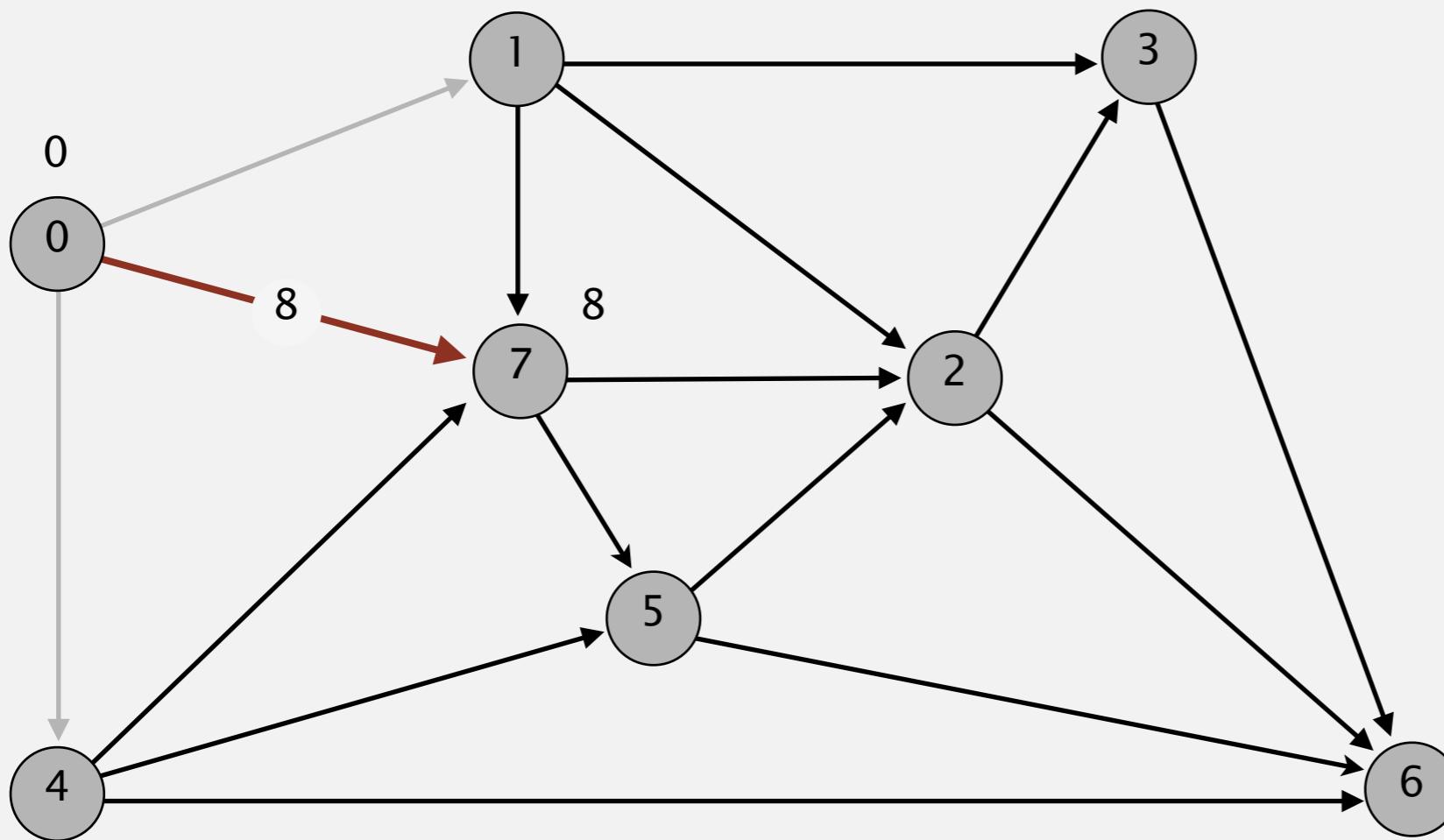
pass 1

0→1 0→4 0→7 1→2 1→3 1→7 2→3 2→6 3→6 4→5 4→6 4→7 5→2 5→6 7→5 7→2



Bellman-Ford algorithm demo

Repeat V times: relax all E edges.



pass 1

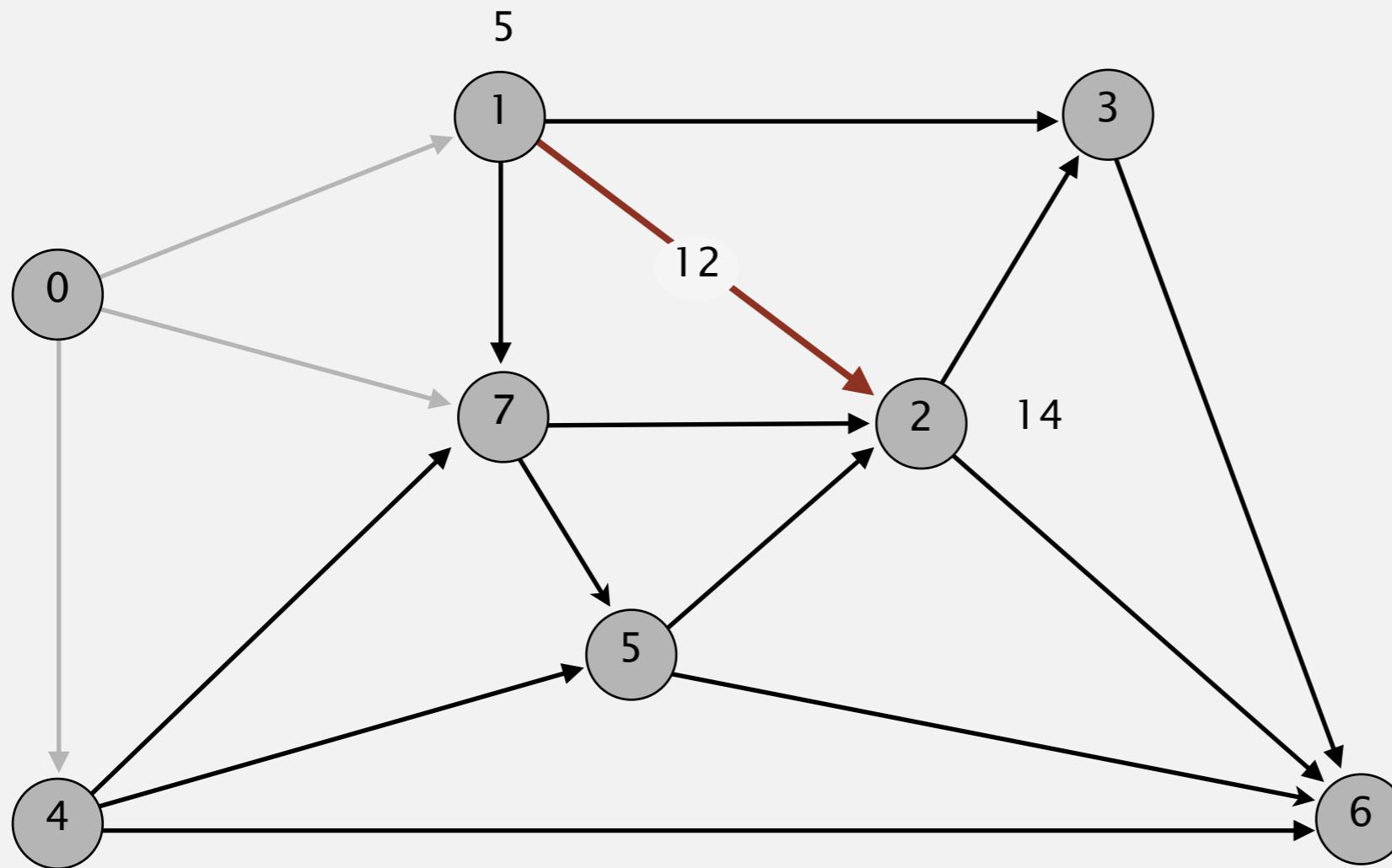
0→1 0→4 0→7 1→2 1→3 1→7 2→3 2→6 3→6 4→5 4→6 4→7 5→2 5→6 7→5 7→2



v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0→1
2	14.0	5→2
3	20.0	1→3
4	9.0	0→4
5	13.0	4→5
6	26.0	5→6
7	8.0	0→7

Bellman-Ford algorithm demo

Repeat V times: relax all E edges.



v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0→1
2	14.0	5→2
3	20.0	1→3
4	9.0	0→4
5	13.0	4→5
6	26.0	5→6
7	8.0	0→7

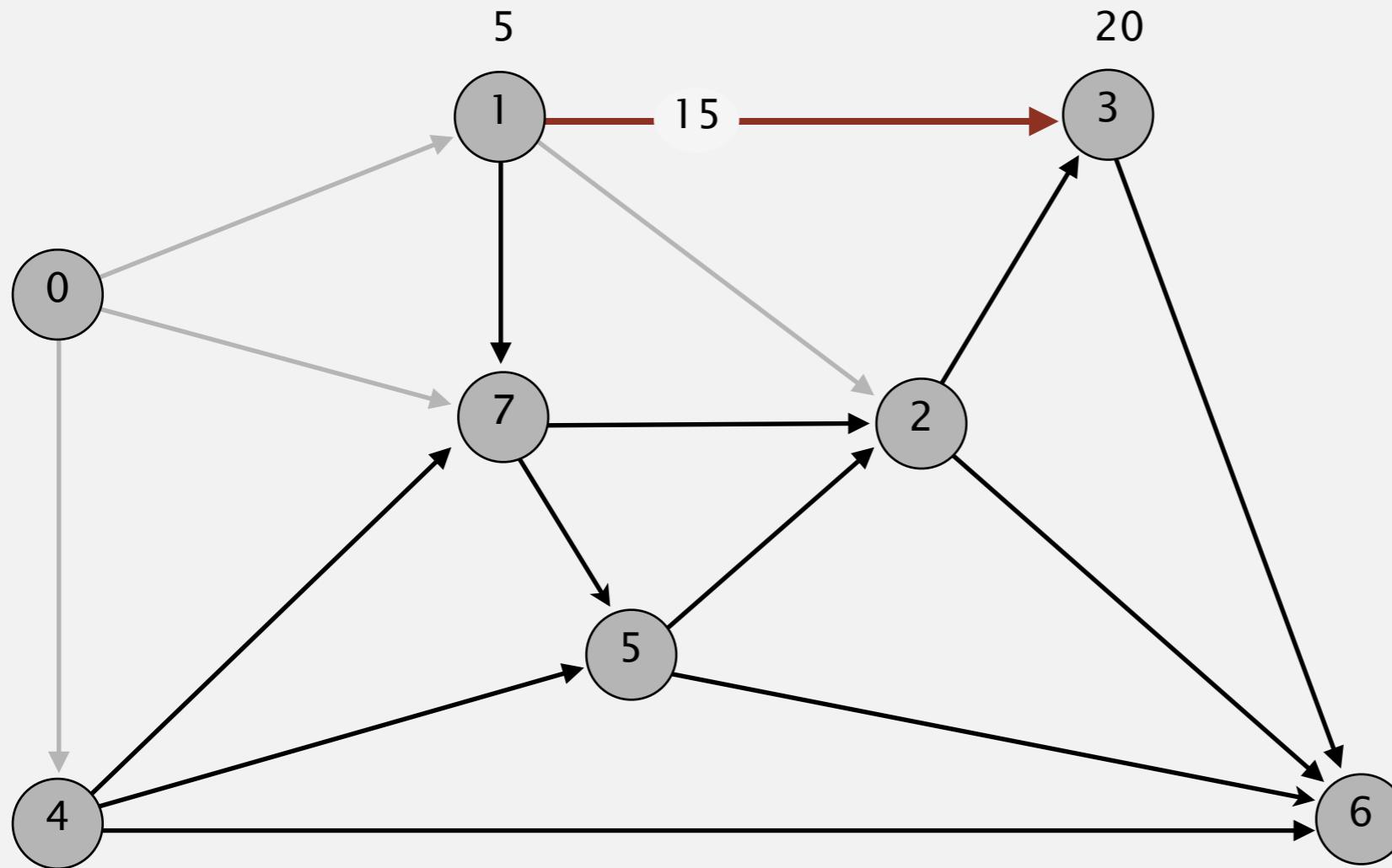
pass 1

0→1 0→4 0→7 1→2 1→3 1→7 2→3 2→6 3→6 4→5 4→6 4→7 5→2 5→6 7→5 7→2



Bellman-Ford algorithm demo

Repeat V times: relax all E edges.



v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0→1
2	14.0	5→2
3	20.0	1→3
4	9.0	0→4
5	13.0	4→5
6	26.0	5→6
7	8.0	0→7

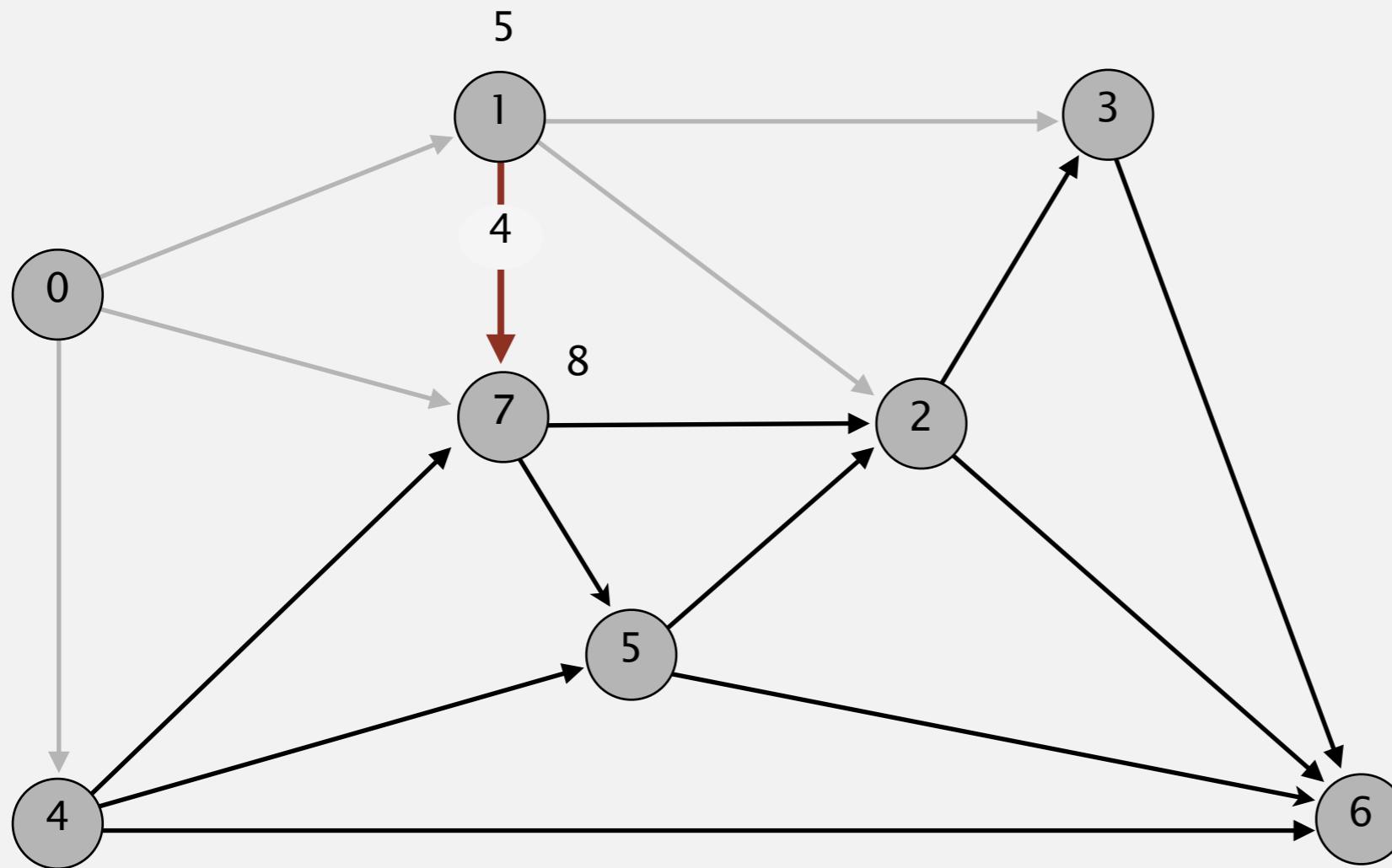
pass 1

0→1 0→4 0→7 1→2 1→3 1→7 2→3 2→6 3→6 4→5 4→6 4→7 5→2 5→6 7→5 7→2



Bellman-Ford algorithm demo

Repeat V times: relax all E edges.



v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0→1
2	14.0	5→2
3	20.0	1→3
4	9.0	0→4
5	13.0	4→5
6	26.0	5→6
7	8.0	0→7

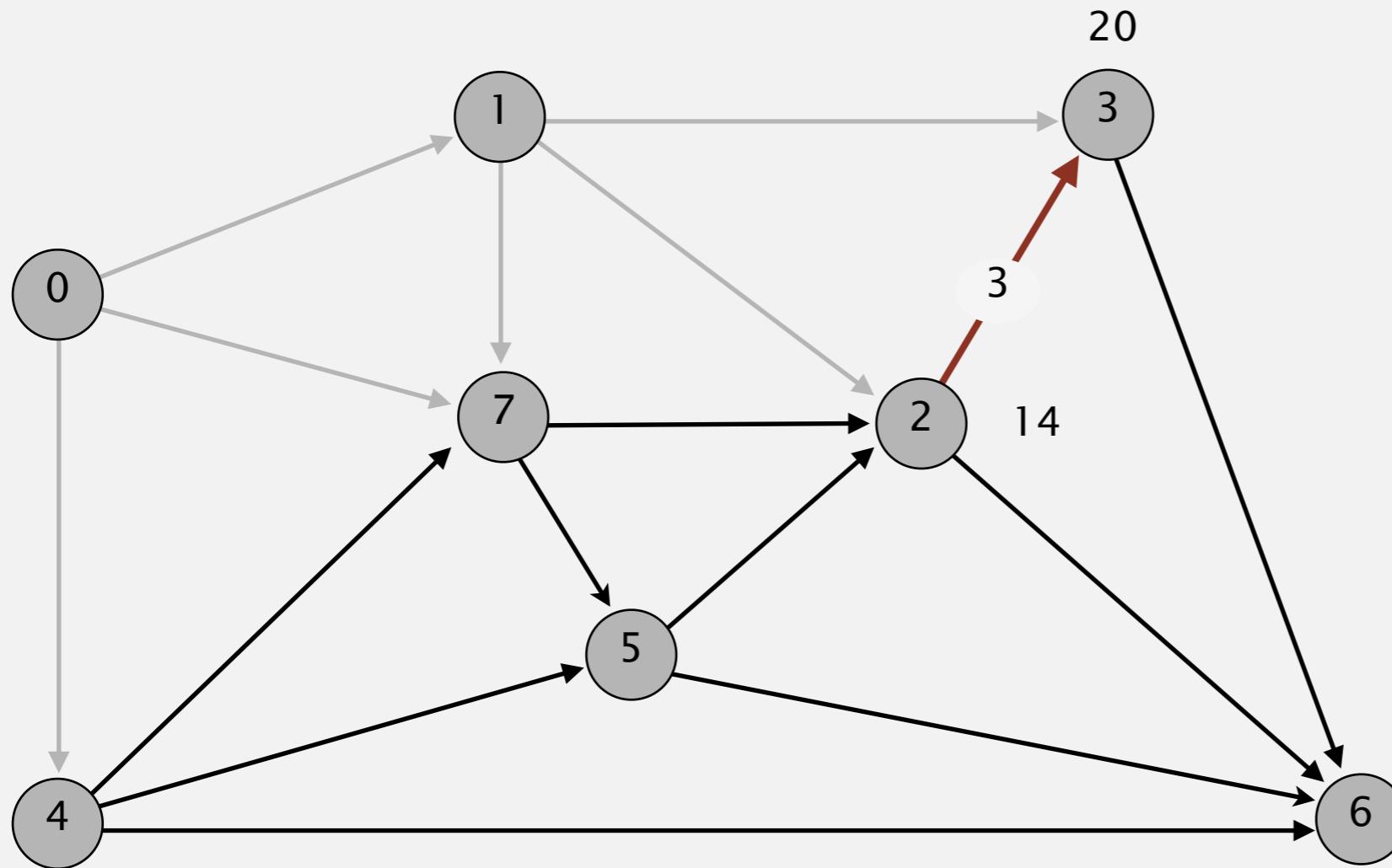
pass 1

0→1 0→4 0→7 1→2 1→3 1→7 2→3 2→6 3→6 4→5 4→6 4→7 5→2 5→6 7→5 7→2



Bellman-Ford algorithm demo

Repeat V times: relax all E edges.



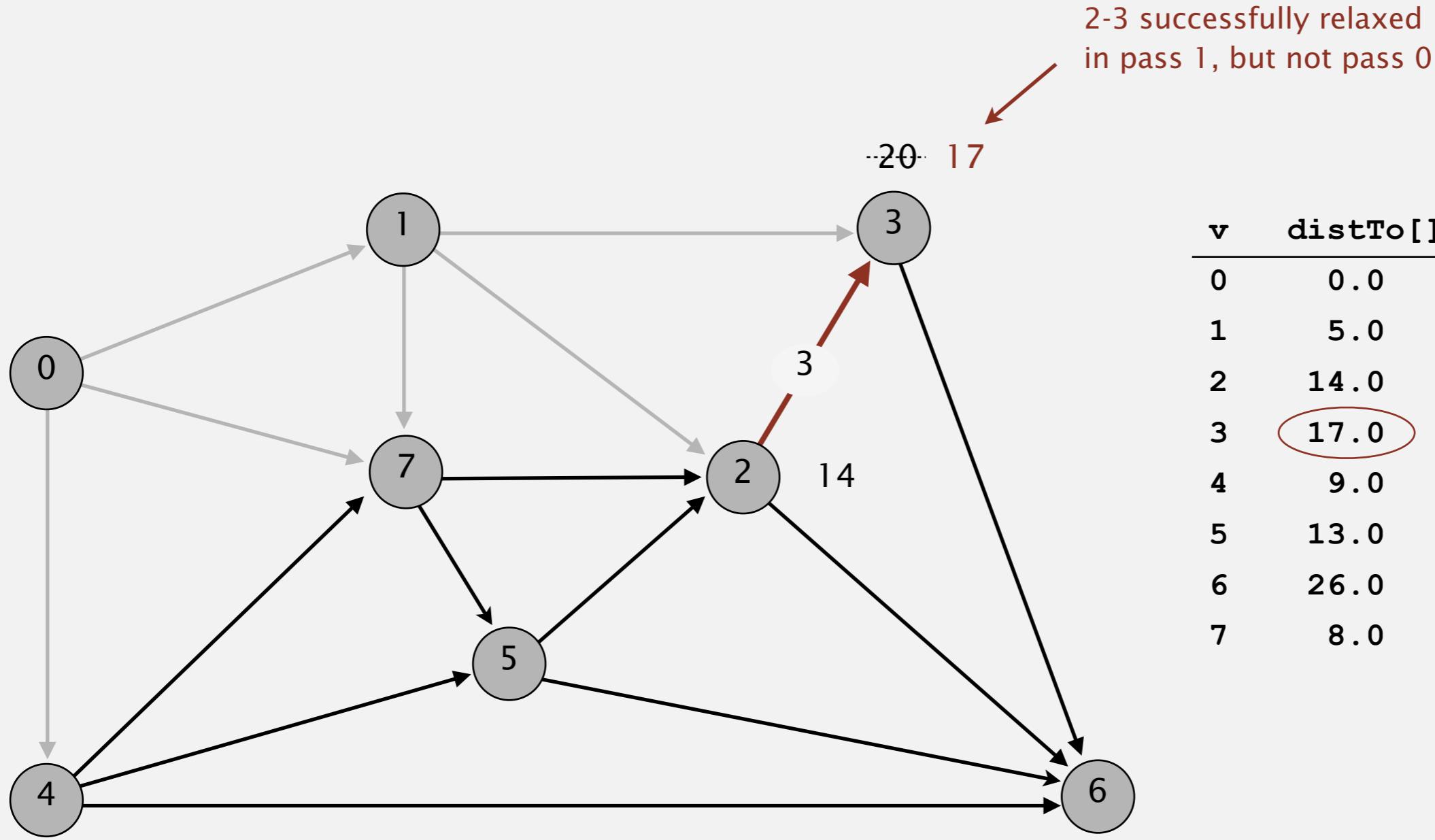
v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0→1
2	14.0	5→2
3	20.0	1→3
4	9.0	0→4
5	13.0	4→5
6	26.0	5→6
7	8.0	0→7

pass 1

0→1 0→4 0→7 1→2 1→3 1→7 2→3 2→6 3→6 4→5 4→6 4→7 5→2 5→6 7→5 7→2

Bellman-Ford algorithm demo

Repeat V times: relax all E edges.



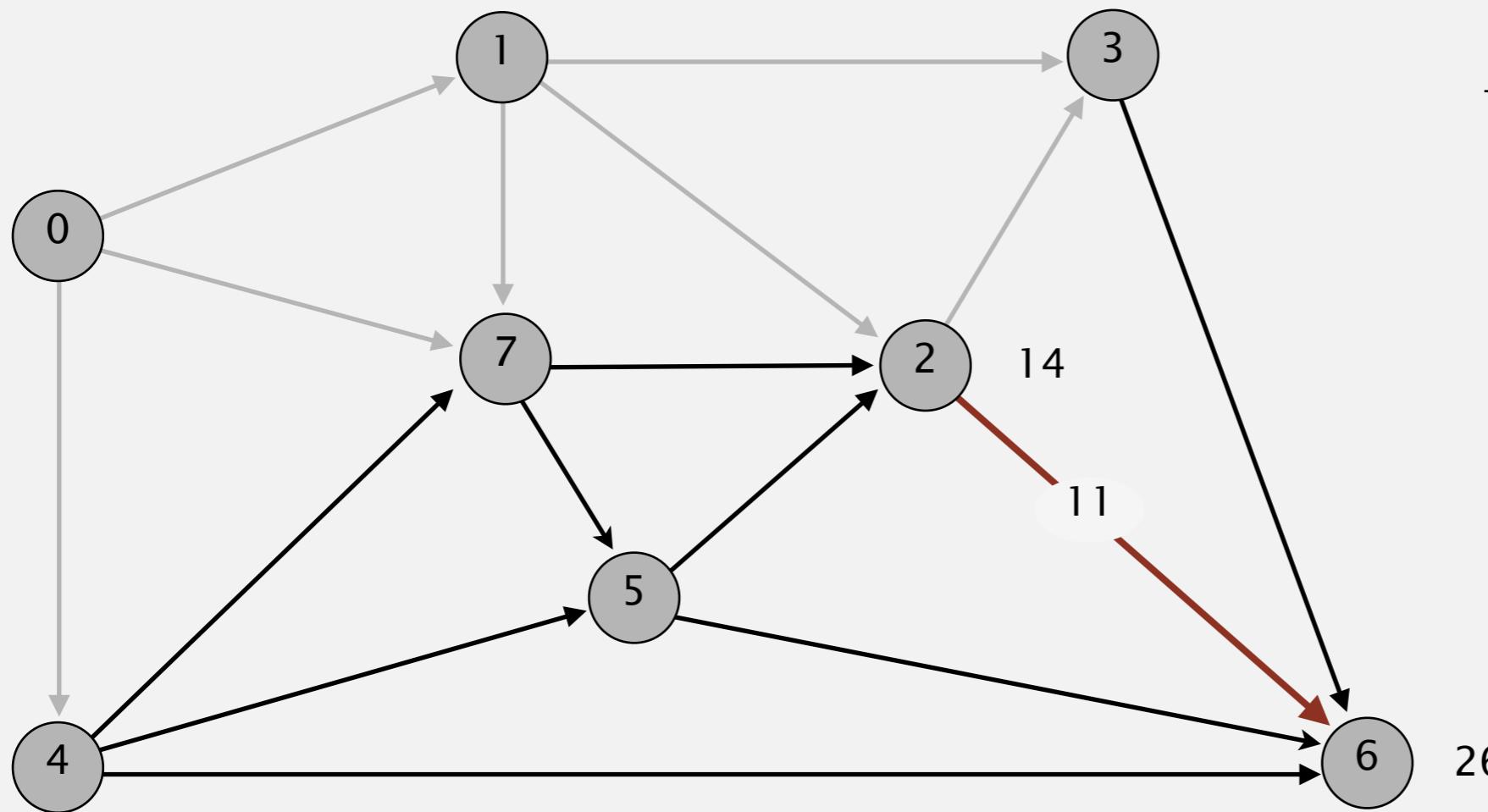
pass 1

0→1 0→4 0→7 1→2 1→3 1→7 2→3 2→6 3→6 4→5 4→6 4→7 5→2 5→6 7→5 7→2



Bellman-Ford algorithm demo

Repeat V times: relax all E edges.



v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0→1
2	14.0	5→2
3	17.0	2→3
4	9.0	0→4
5	13.0	4→5
6	26.0	5→6
7	8.0	0→7

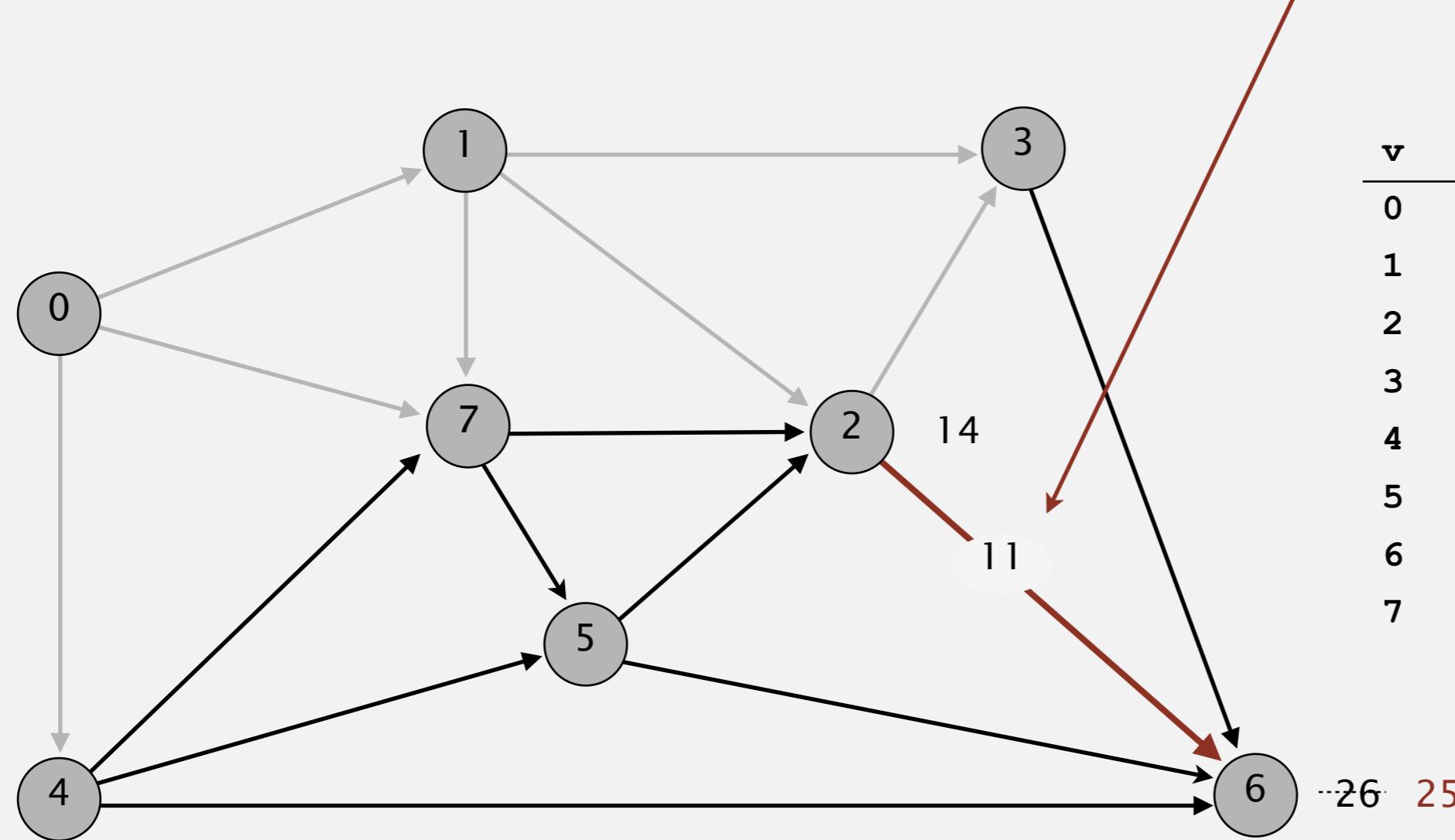
pass 1

0→1 0→4 0→7 1→2 1→3 1→7 2→3 2→6 3→6 4→5 4→6 4→7 5→2 5→6 7→5 7→2



Bellman-Ford algorithm demo

Repeat V times: relax all E edges.



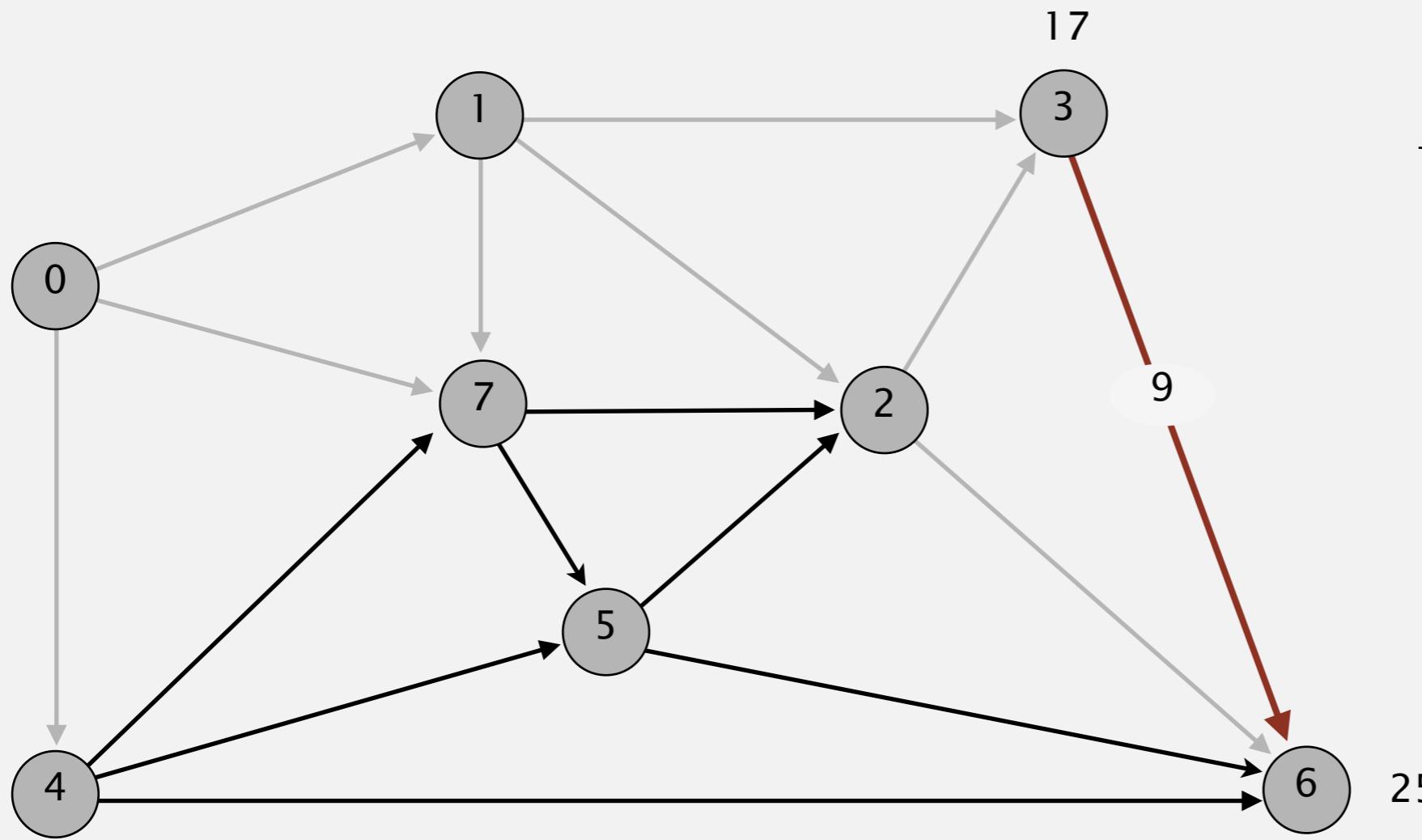
v	distTo []	edgeTo []
0	0.0	-
1	5.0	0→1
2	14.0	5→2
3	17.0	2→3
4	9.0	0→4
5	13.0	4→5
6	25.0	2→6
7	8.0	0→7

pass 1

0→1 0→4 0→7 1→2 1→3 1→7 2→3 2→6 3→6 4→5 4→6 4→7 5→2 5→6 7→5 7→2

Bellman-Ford algorithm demo

Repeat V times: relax all E edges.



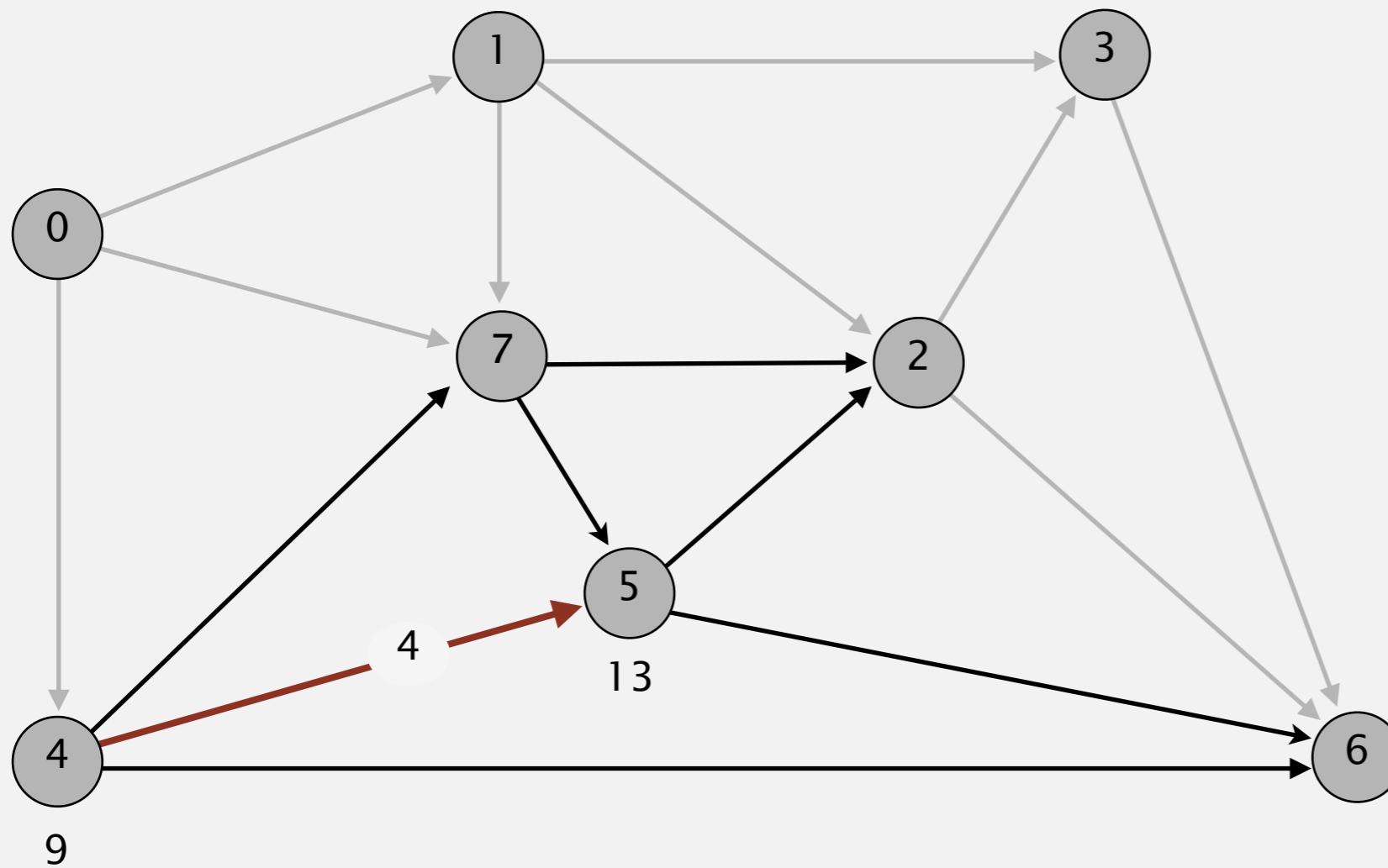
v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0→1
2	14.0	5→2
3	17.0	2→3
4	9.0	0→4
5	13.0	4→5
6	25.0	2→6
7	8.0	0→7

pass 1

0→1 0→4 0→7 1→2 1→3 1→7 2→3 2→6 3→6 4→5 4→6 4→7 5→2 5→6 7→5 7→2

Bellman-Ford algorithm demo

Repeat V times: relax all E edges.



pass 1

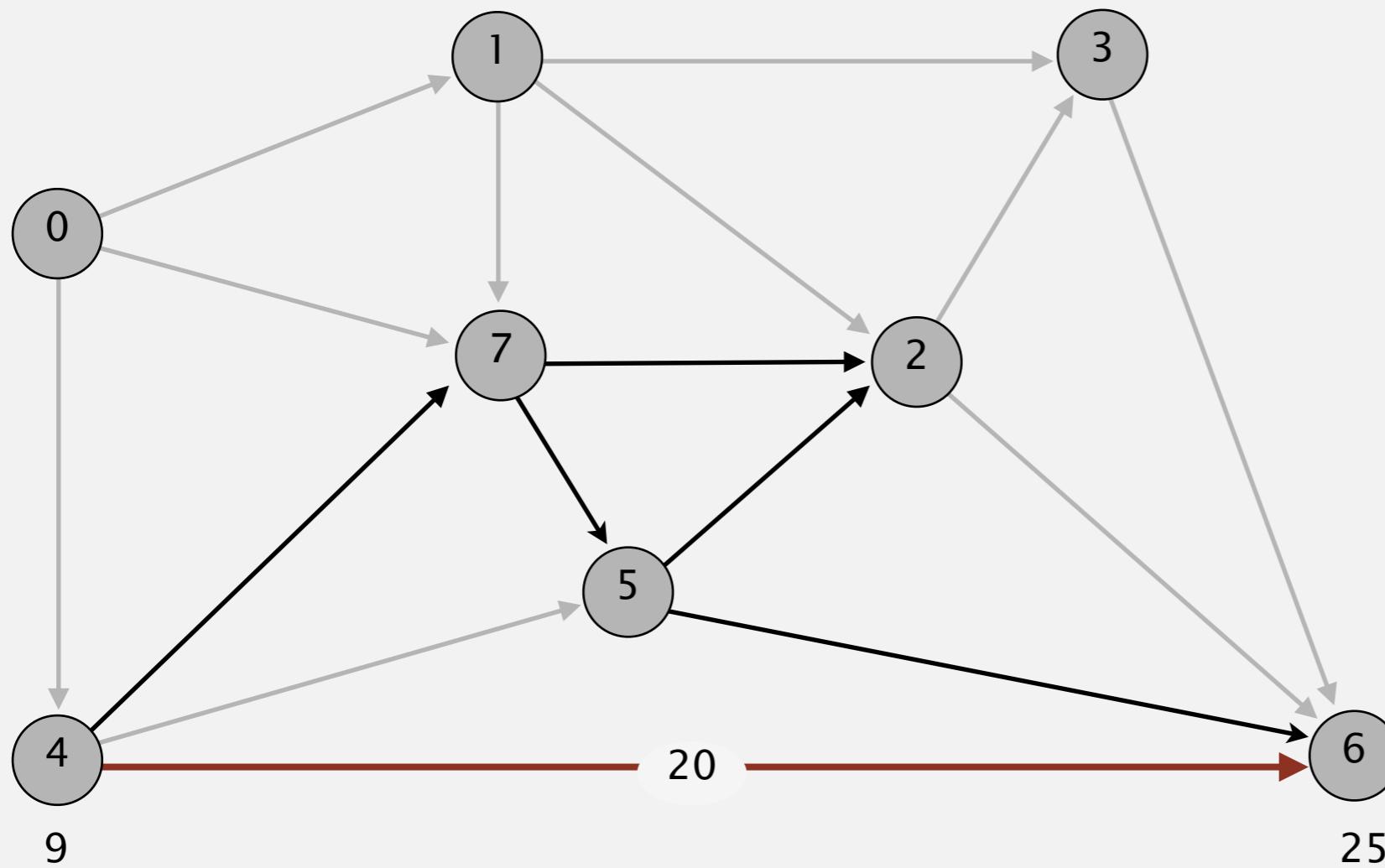
0→1 0→4 0→7 1→2 1→3 1→7 2→3 2→6 3→6 4→5 4→6 4→7 5→2 5→6 7→5 7→2



v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0→1
2	14.0	5→2
3	17.0	2→3
4	9.0	0→4
5	13.0	4→5
6	25.0	2→6
7	8.0	0→7

Bellman-Ford algorithm demo

Repeat V times: relax all E edges.



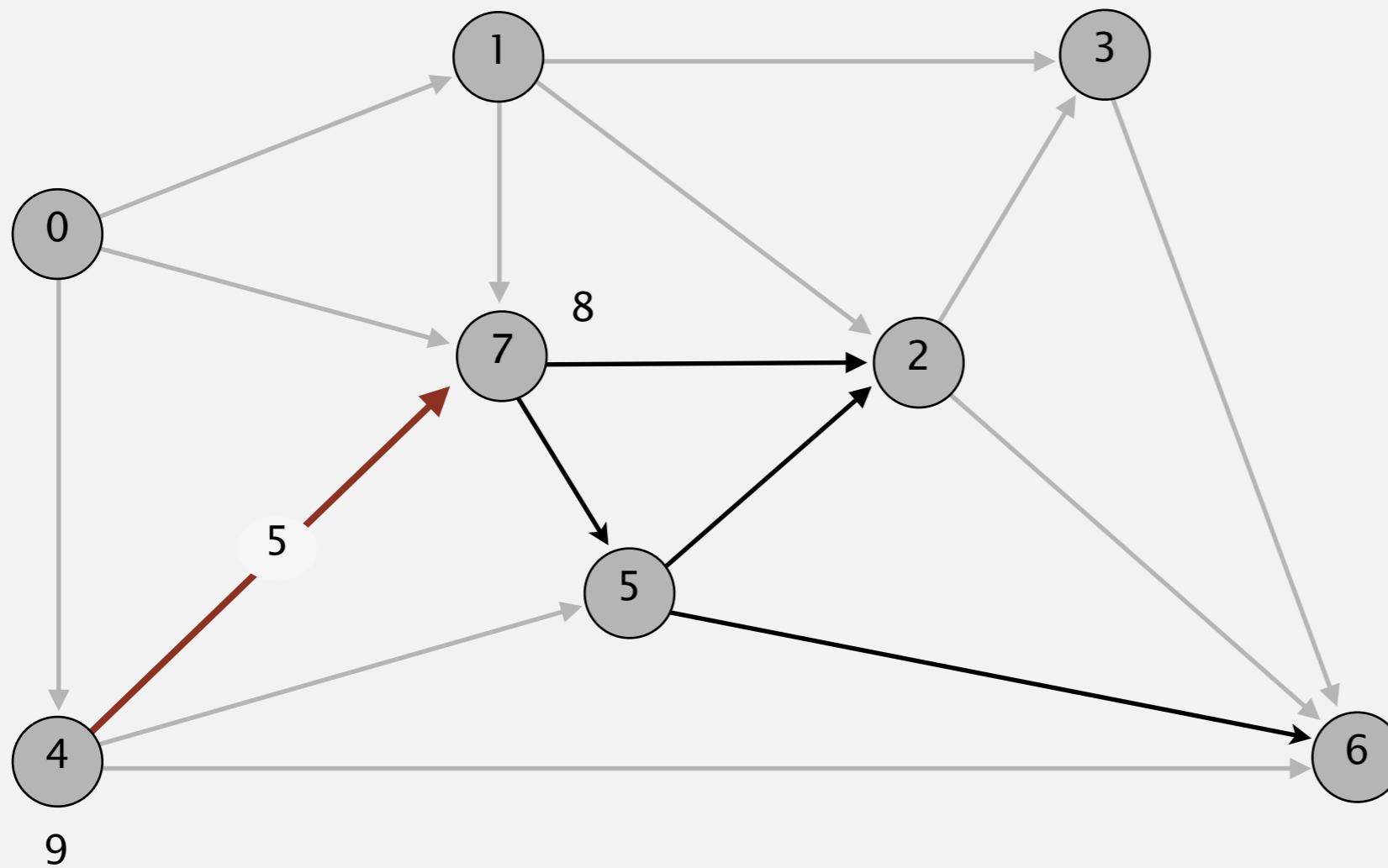
v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0→1
2	14.0	5→2
3	17.0	2→3
4	9.0	0→4
5	13.0	4→5
6	25.0	2→6
7	8.0	0→7

pass 1

0→1 0→4 0→7 1→2 1→3 1→7 2→3 2→6 3→6 4→5 4→6 4→7 5→2 5→6 7→5 7→2

Bellman-Ford algorithm demo

Repeat V times: relax all E edges.



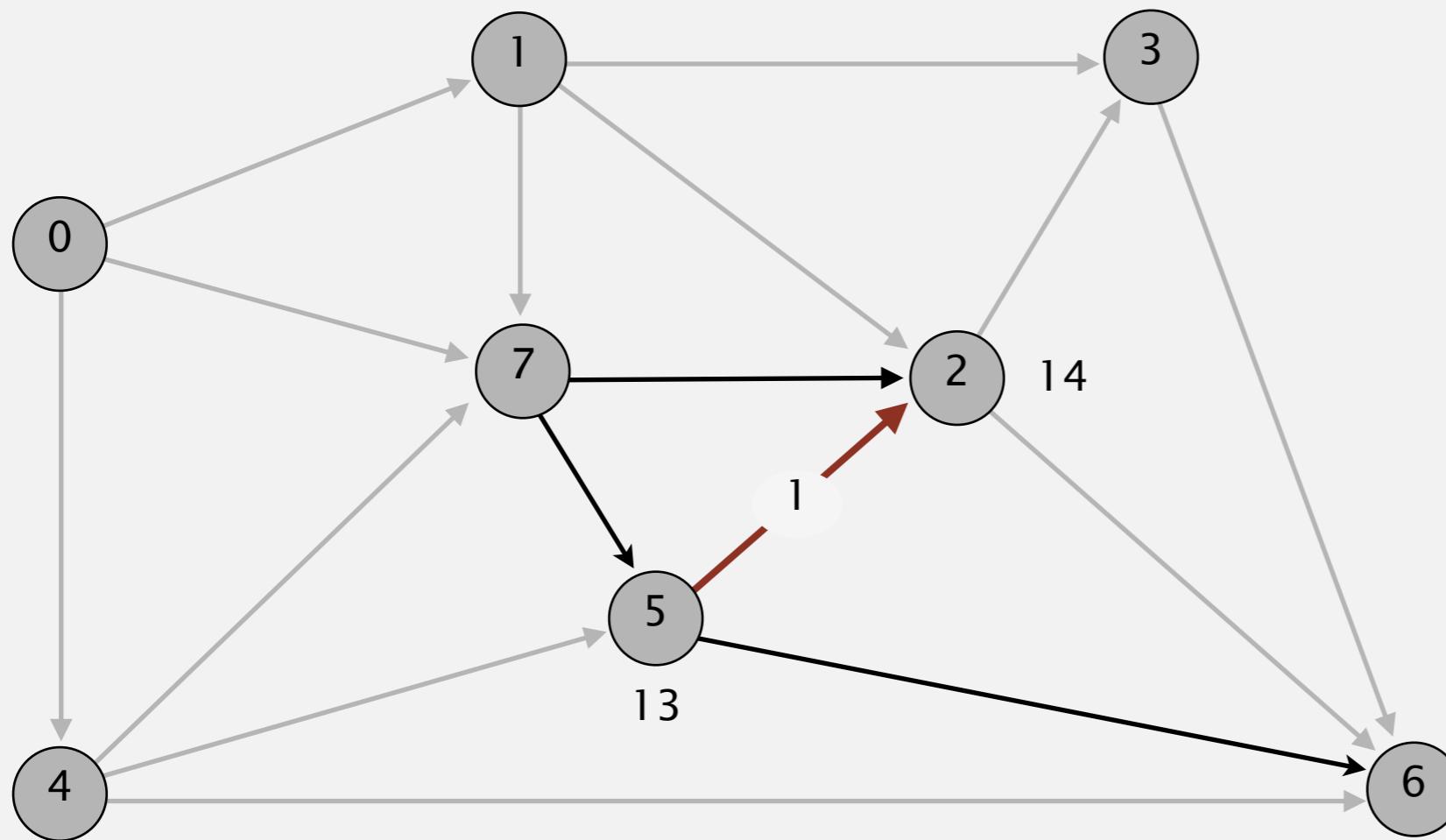
v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0→1
2	14.0	5→2
3	17.0	2→3
4	9.0	0→4
5	13.0	4→5
6	25.0	2→6
7	8.0	0→7

pass 1

0→1 0→4 0→7 1→2 1→3 1→7 2→3 2→6 3→6 4→5 4→6 4→7 5→2 5→6 7→5 7→2

Bellman-Ford algorithm demo

Repeat V times: relax all E edges.



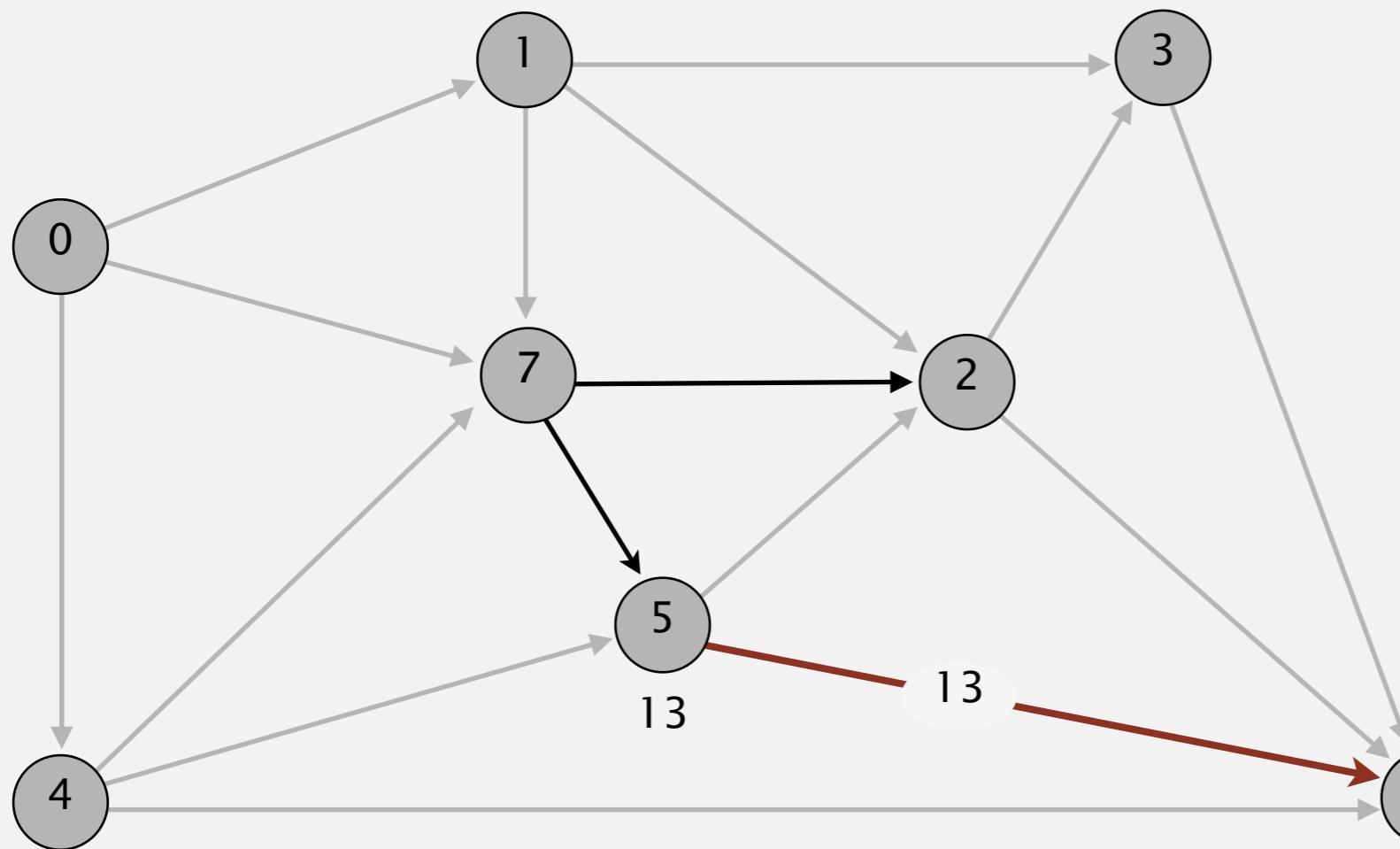
v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0→1
2	14.0	5→2
3	17.0	2→3
4	9.0	0→4
5	13.0	4→5
6	25.0	2→6
7	8.0	0→7

pass 1

0→1 0→4 0→7 1→2 1→3 1→7 2→3 2→6 3→6 4→5 4→6 4→7 5→2 5→6 7→5 7→2

Bellman-Ford algorithm demo

Repeat V times: relax all E edges.



v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0→1
2	14.0	5→2
3	17.0	2→3
4	9.0	0→4
5	13.0	4→5
6	25.0	2→6
7	8.0	0→7

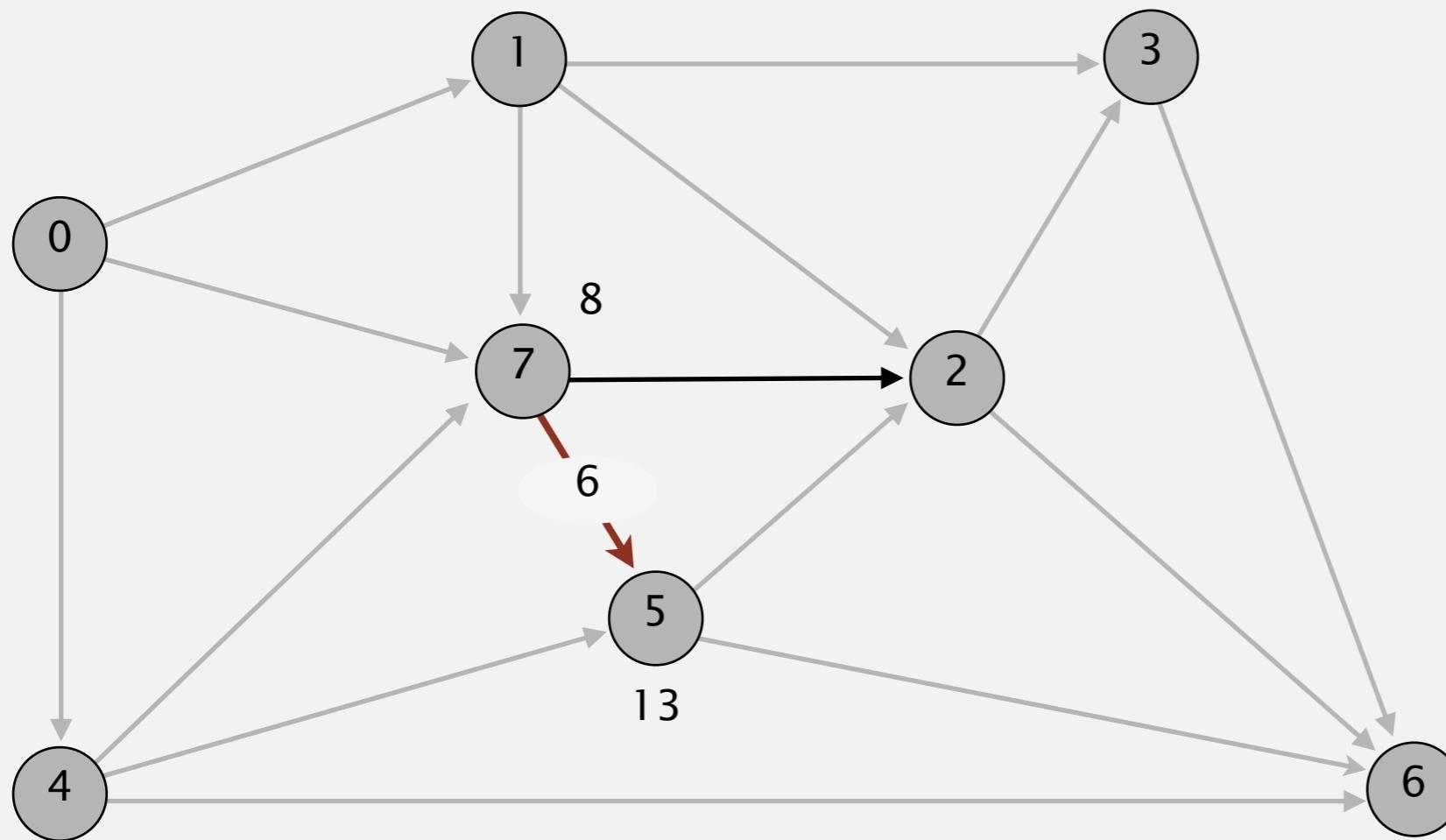
pass 1

0→1 0→4 0→7 1→2 1→3 1→7 2→3 2→6 3→6 4→5 4→6 4→7 5→2 5→6 7→5 7→2



Bellman-Ford algorithm demo

Repeat V times: relax all E edges.



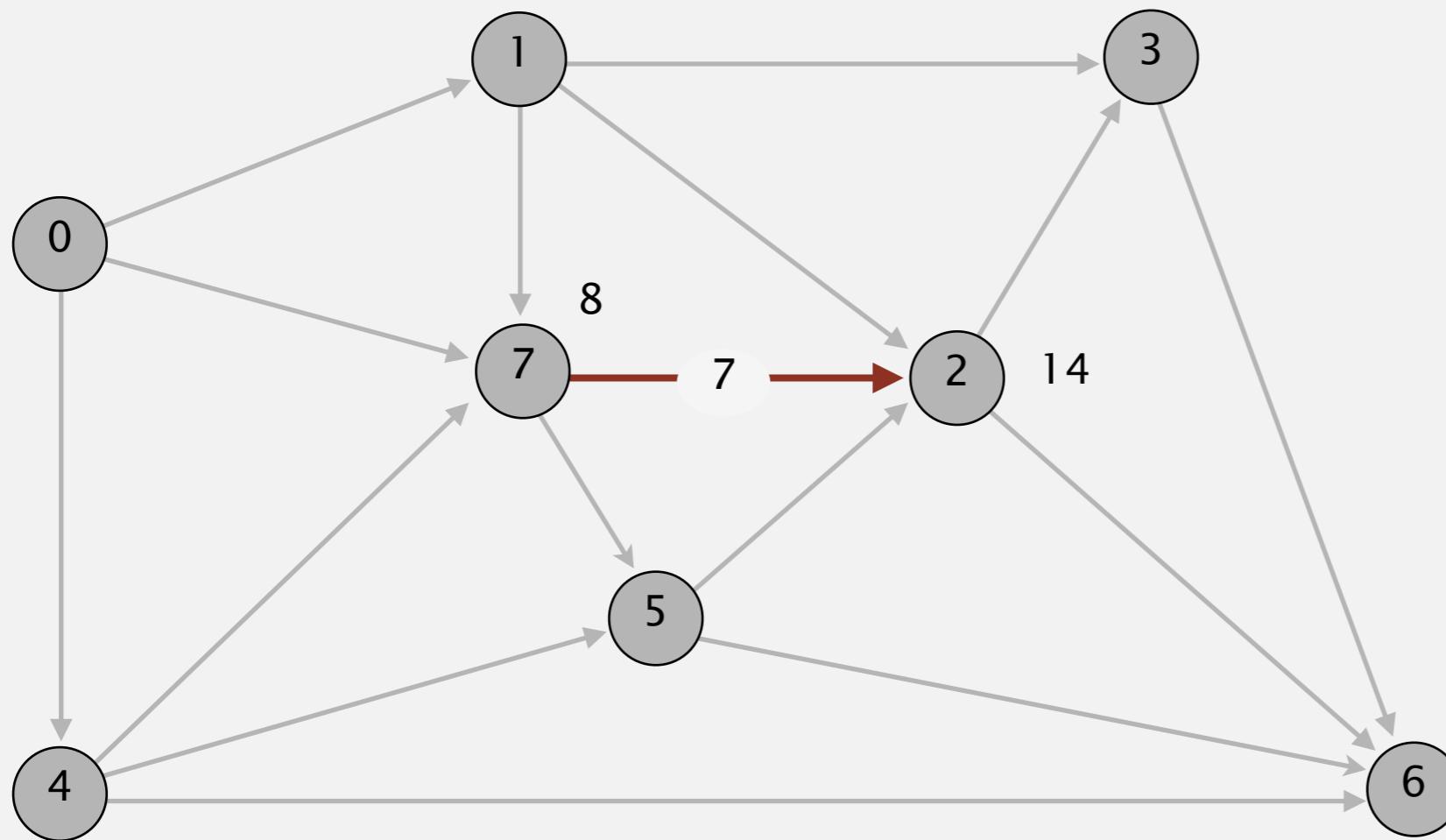
v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0→1
2	14.0	5→2
3	17.0	2→3
4	9.0	0→4
5	13.0	4→5
6	25.0	2→6
7	8.0	0→7

pass 1

0→1 0→4 0→7 1→2 1→3 1→7 2→3 2→6 3→6 4→5 4→6 4→7 5→2 5→6 7→5 7→2

Bellman-Ford algorithm demo

Repeat V times: relax all E edges.



v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0→1
2	14.0	5→2
3	17.0	2→3
4	9.0	0→4
5	13.0	4→5
6	25.0	2→6
7	8.0	0→7

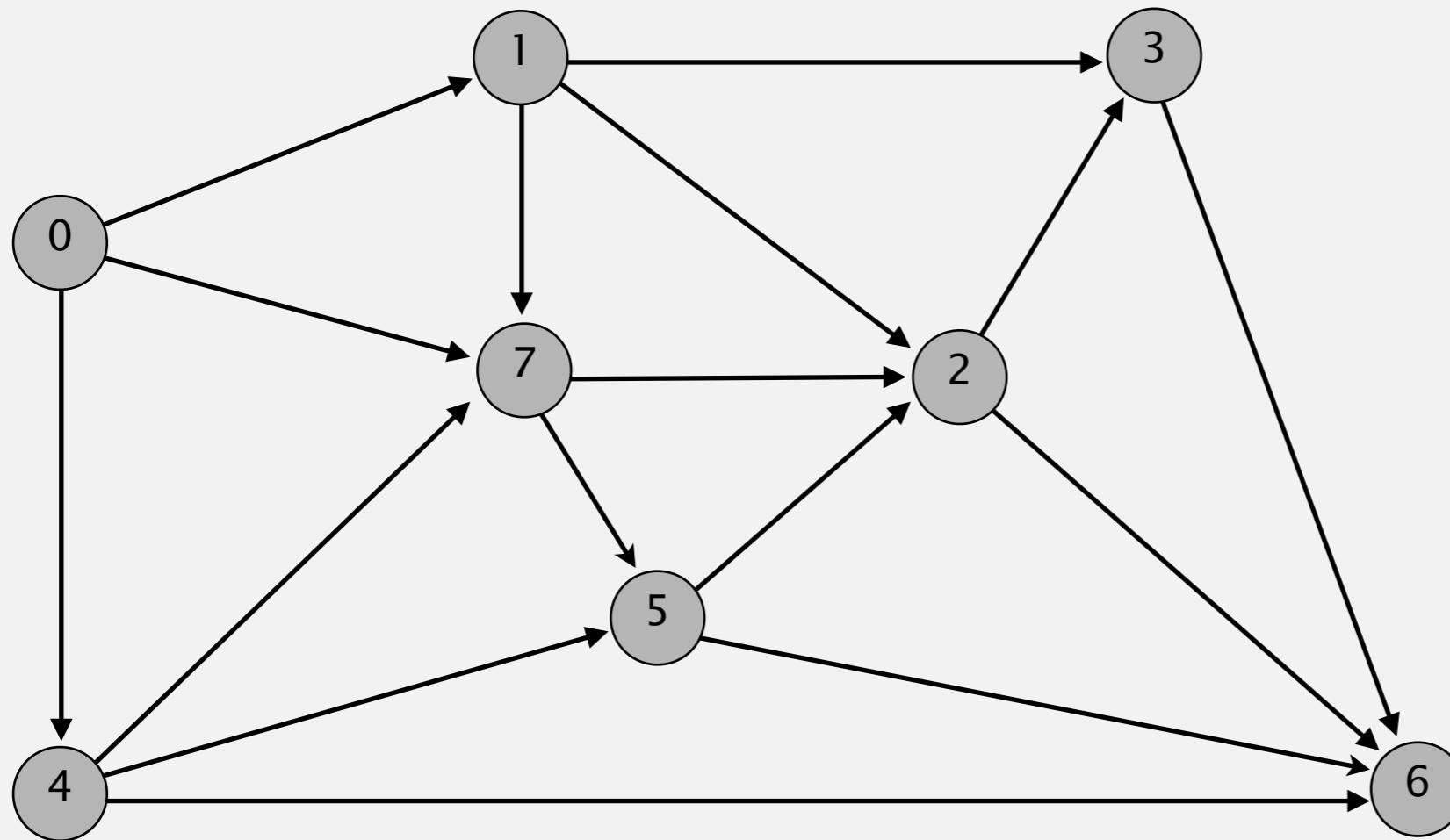
pass 1

0→1 0→4 0→7 1→2 1→3 1→7 2→3 2→6 3→6 4→5 4→6 4→7 5→2 5→6 7→5 7→2



Bellman-Ford algorithm demo

Repeat V times: relax all E edges.



v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0→1
2	14.0	5→2
3	17.0	2→3
4	9.0	0→4
5	13.0	4→5
6	25.0	2→6
7	8.0	0→7

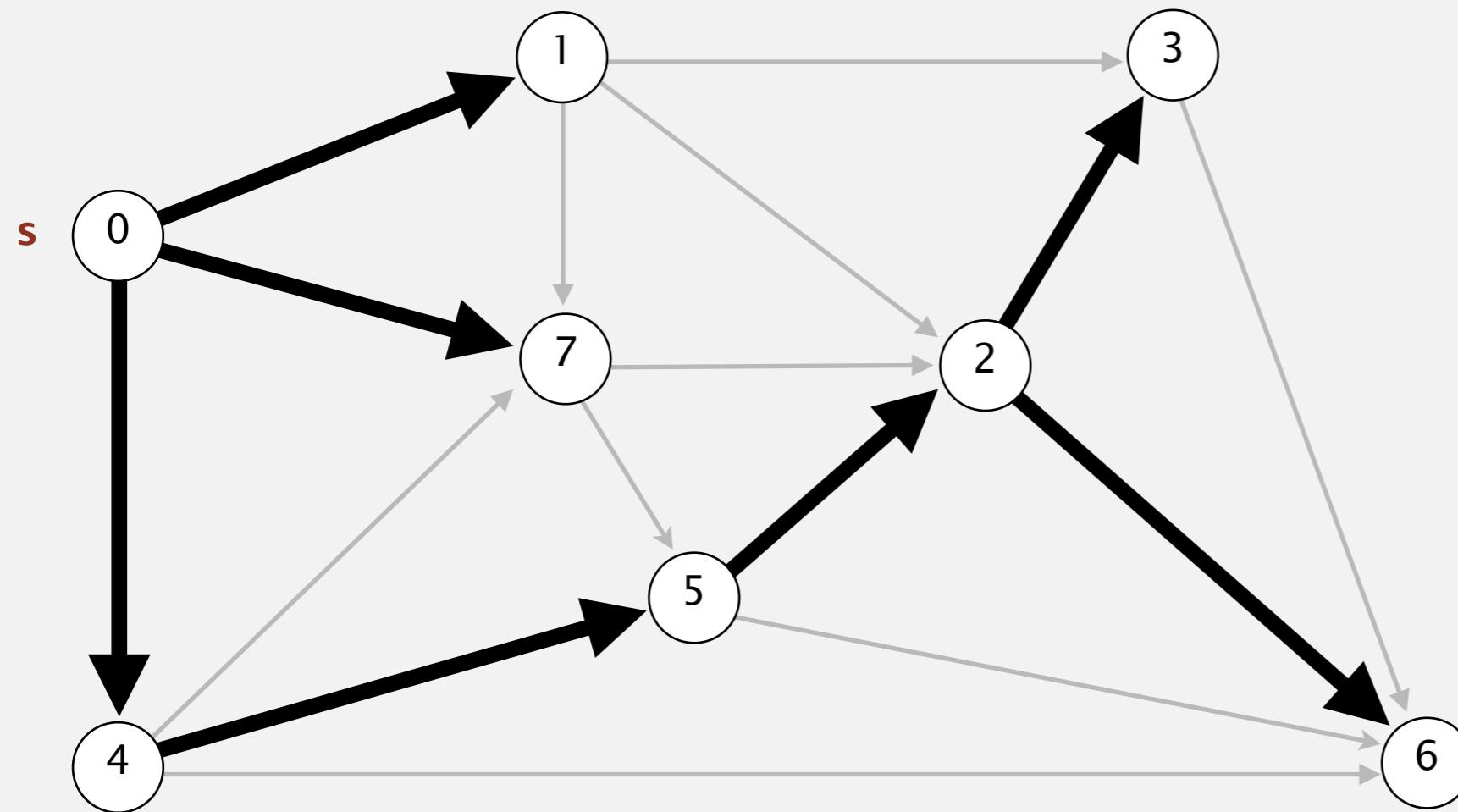
pass 2, 3, 4, ... (no further changes)

0→1 0→4 0→7 1→2 1→3 1→7 2→3 2→6 3→6 4→5 4→6 4→7 5→2 5→6 7→5 7→2



Bellman-Ford algorithm demo

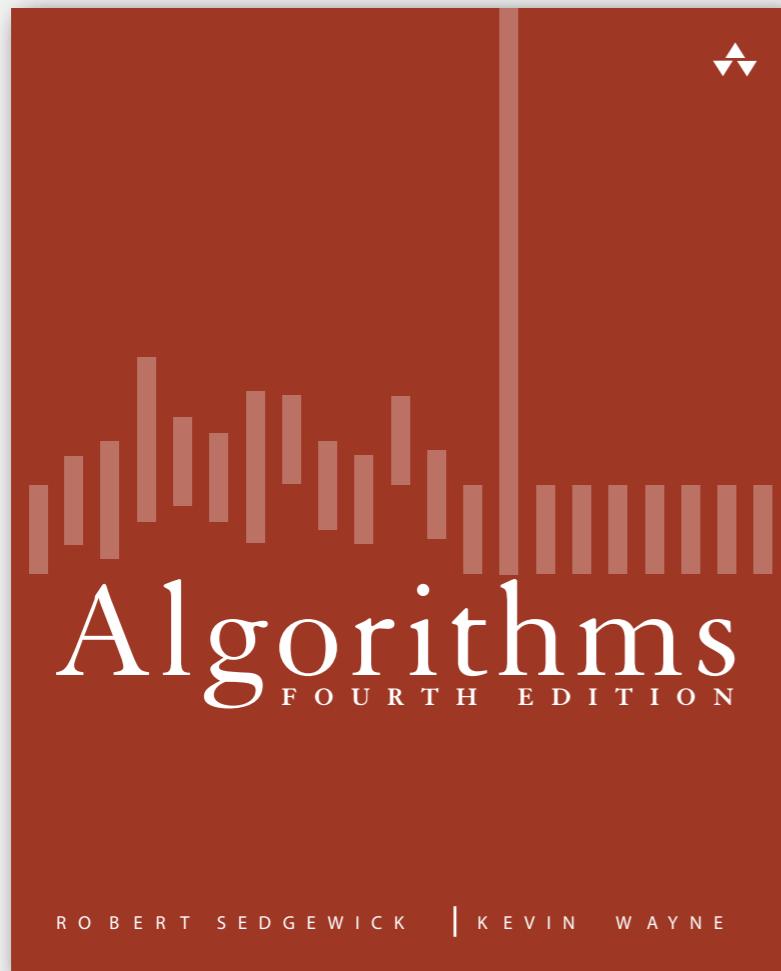
Repeat V times: relax all E edges.



shortest-paths tree from vertex s

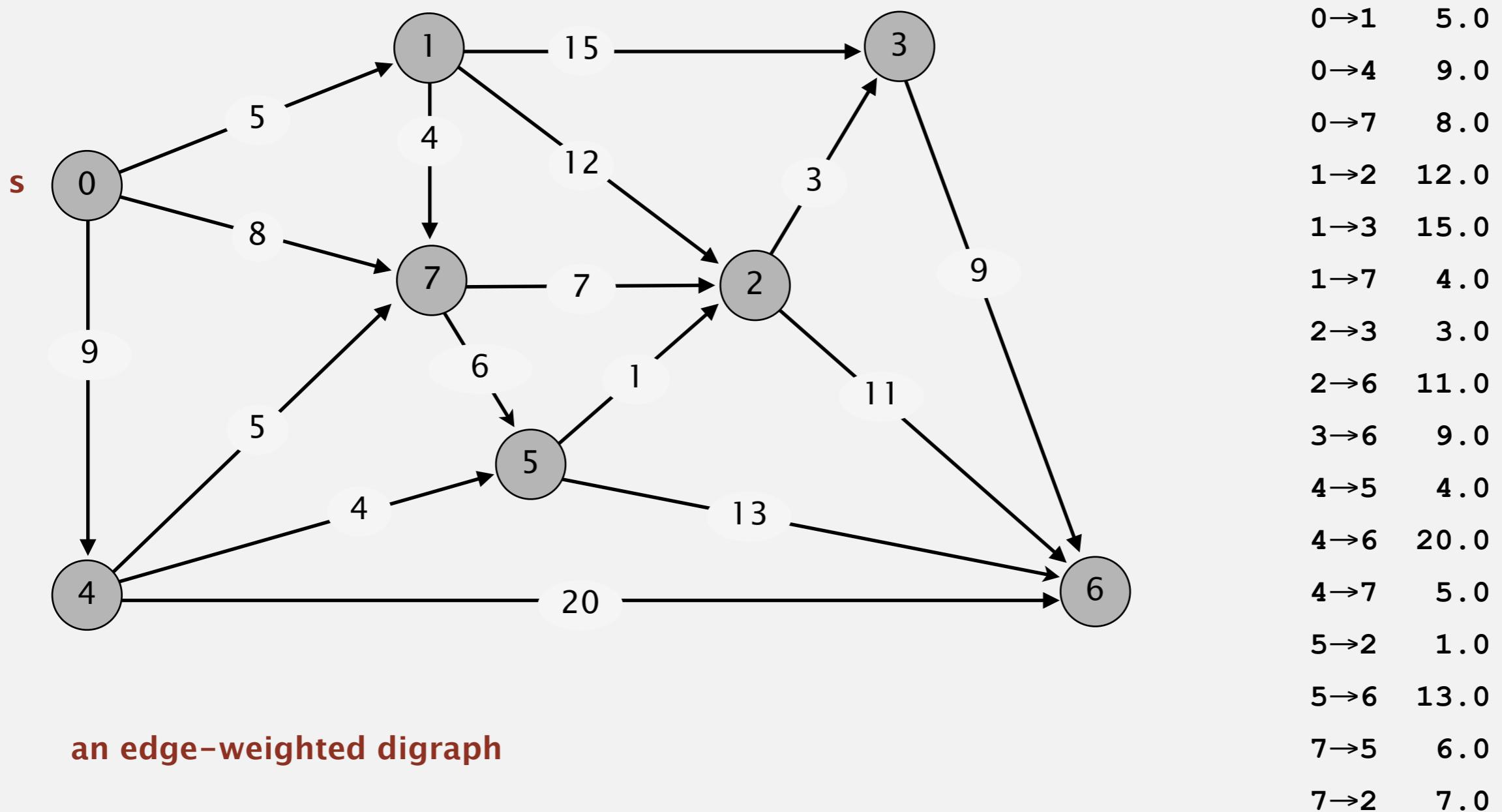
v	distTo []	edgeTo []
0	0.0	-
1	5.0	0→1
2	14.0	5→2
3	17.0	2→3
4	9.0	0→4
5	13.0	4→5
6	25.0	2→6
7	8.0	0→7

4.4 DIJKSTRA's ALGORITHM DEMO



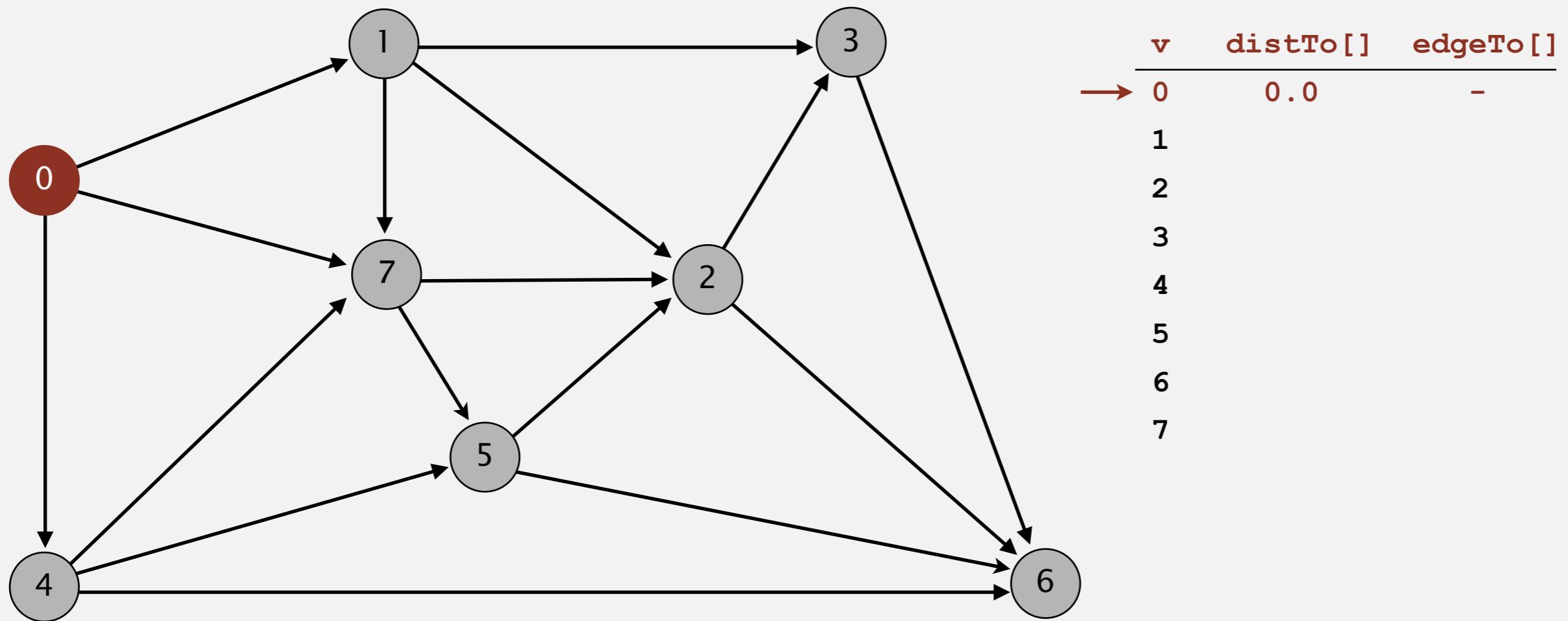
Dijkstra's algorithm

- Consider vertices in increasing order of distance from s (non-tree vertex with the lowest `distTo[]` value).
- Add vertex to tree and relax all edges incident from that vertex.



Dijkstra's algorithm

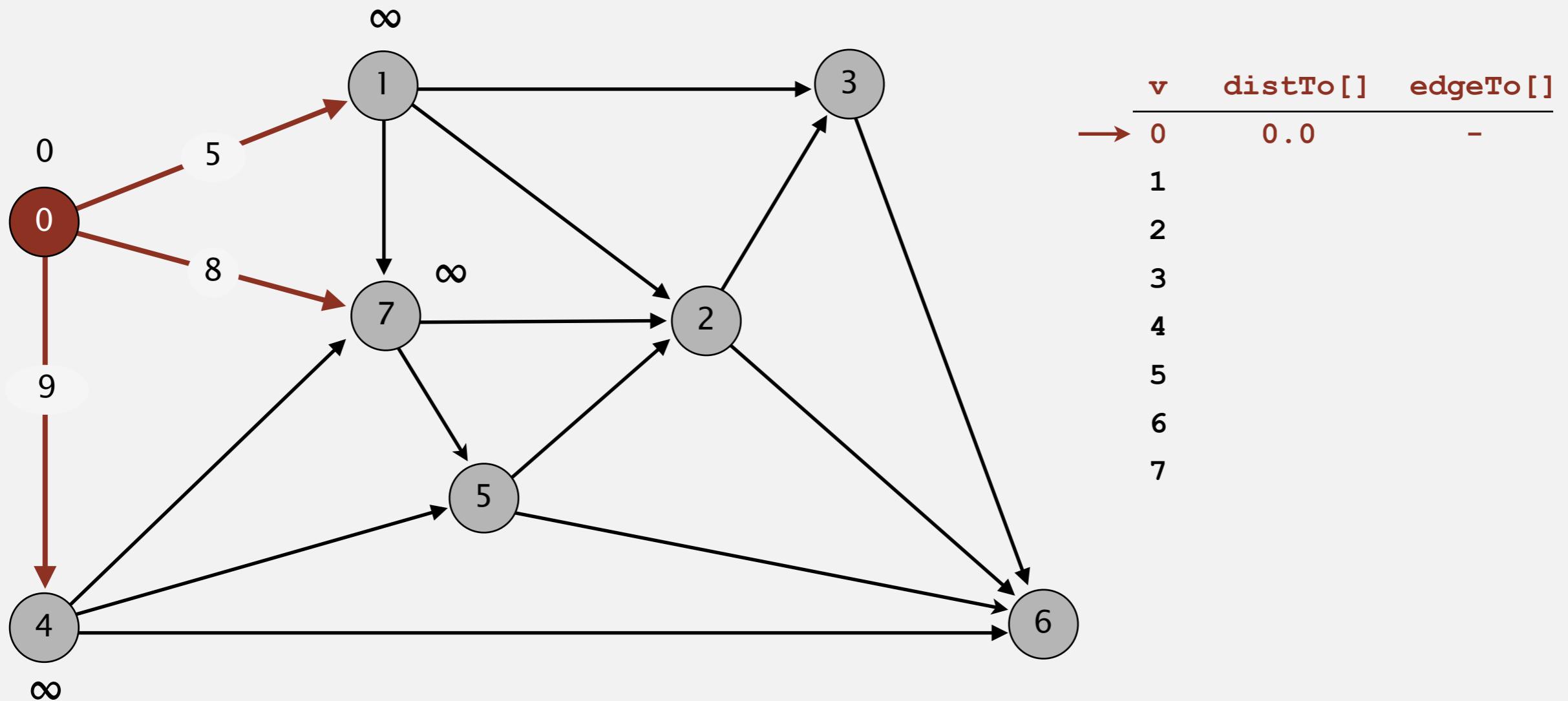
- Consider vertices in increasing order of distance from s (non-tree vertex with the lowest `distTo[]` value).
- Add vertex to tree and relax all edges incident from that vertex.



choose source vertex 0

Dijkstra's algorithm

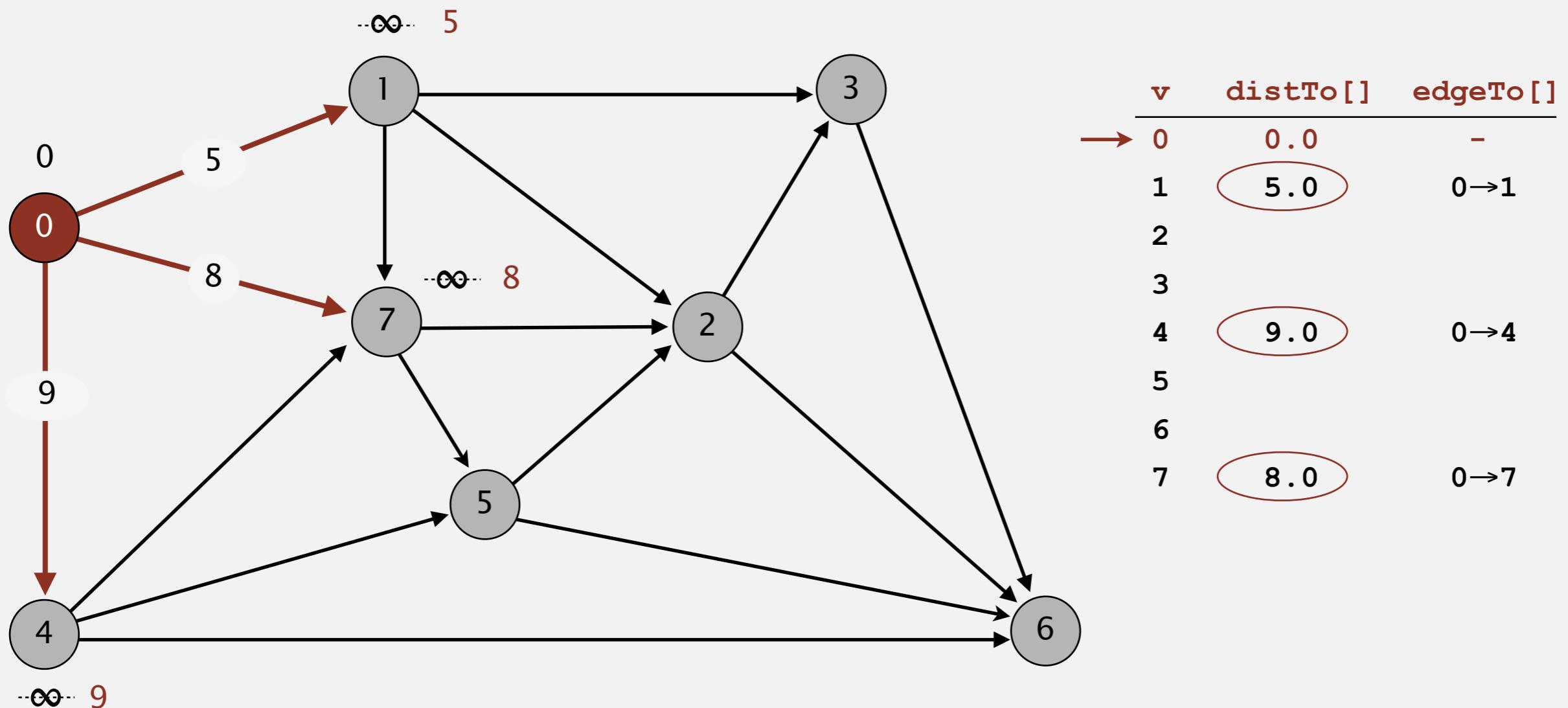
- Consider vertices in increasing order of distance from s (non-tree vertex with the lowest `distTo[]` value).
- Add vertex to tree and relax all edges incident from that vertex.



relax all edges incident from 0

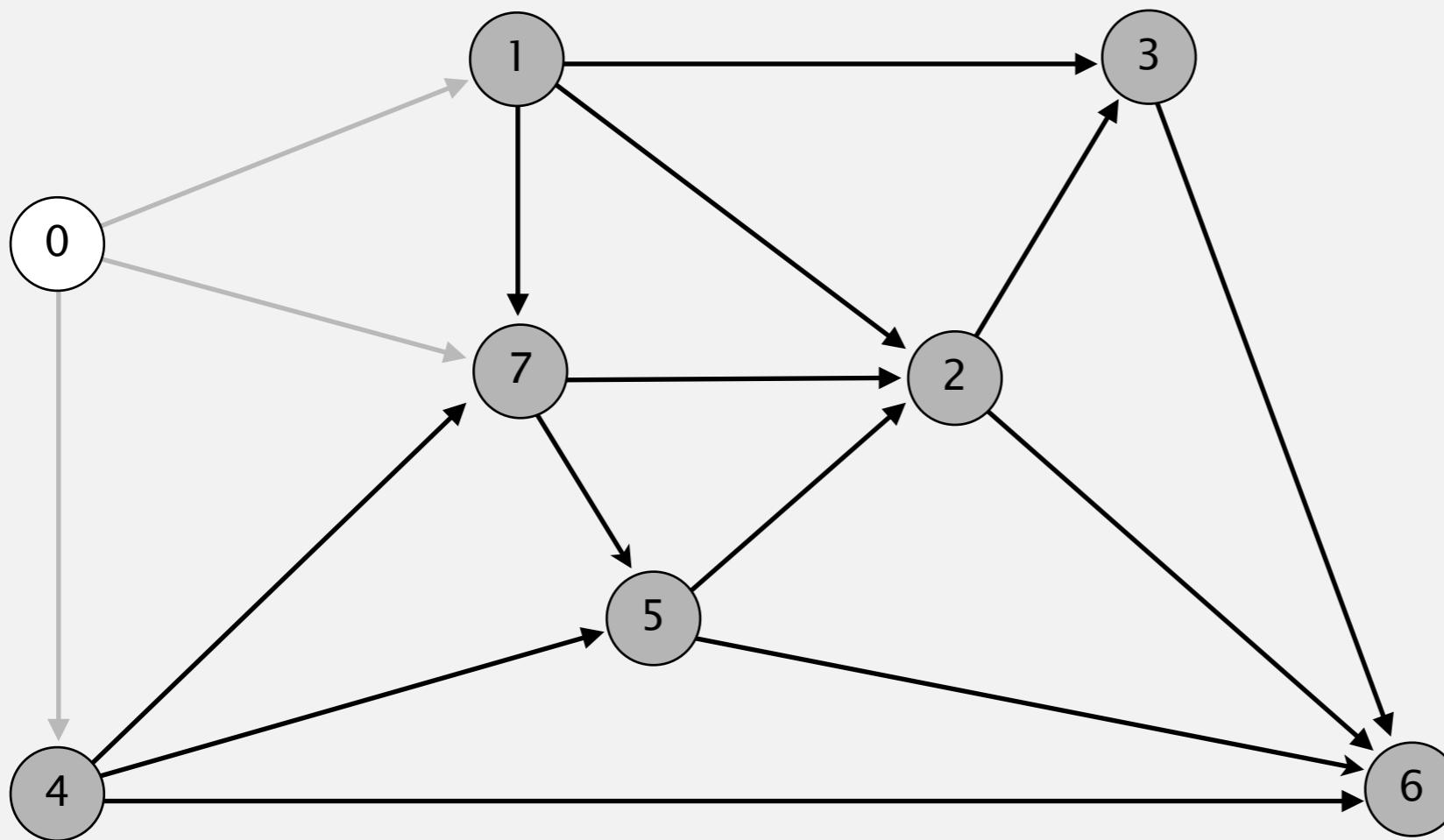
Dijkstra's algorithm

- Consider vertices in increasing order of distance from s (non-tree vertex with the lowest `distTo[]` value).
- Add vertex to tree and relax all edges incident from that vertex.



Dijkstra's algorithm

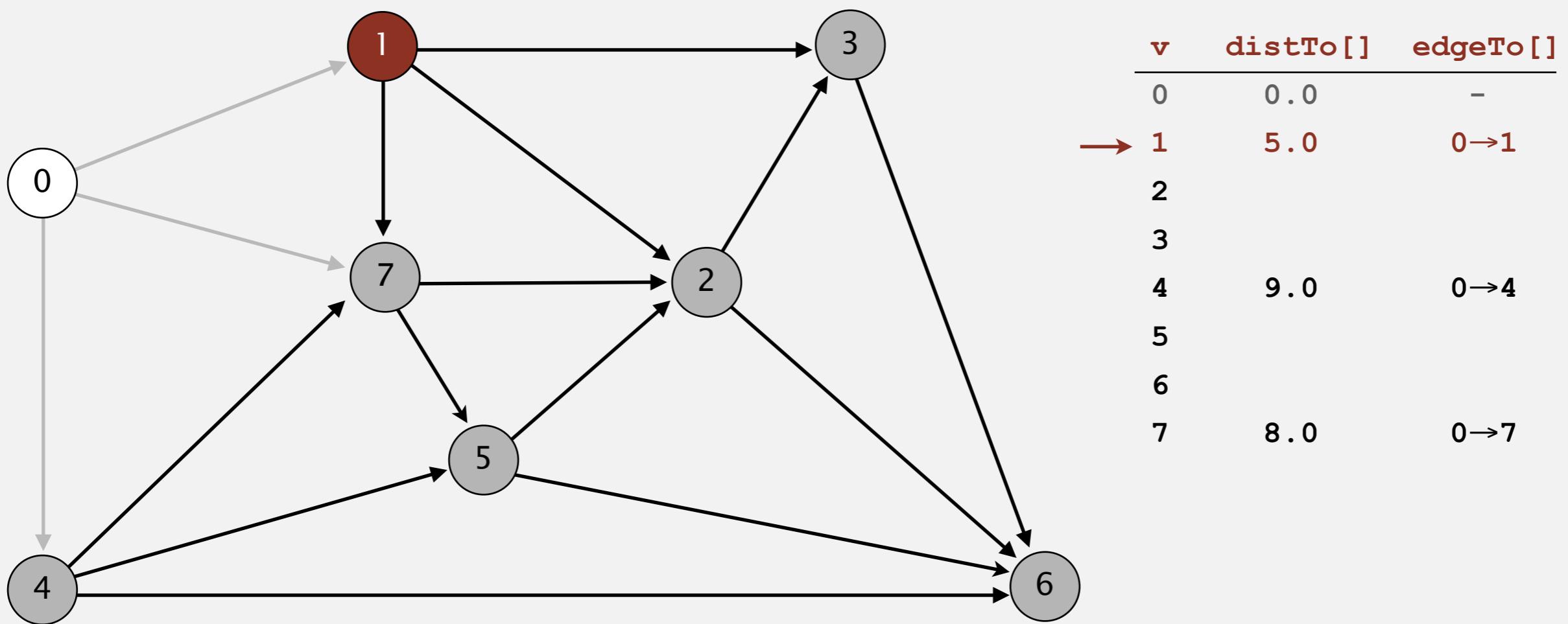
- Consider vertices in increasing order of distance from s (non-tree vertex with the lowest `distTo[]` value).
- Add vertex to tree and relax all edges incident from that vertex.



v	distTo[]	edgeTo []
0	0.0	-
1	5.0	0→1
2		
3		
4	9.0	0→4
5		
6		
7	8.0	0→7

Dijkstra's algorithm

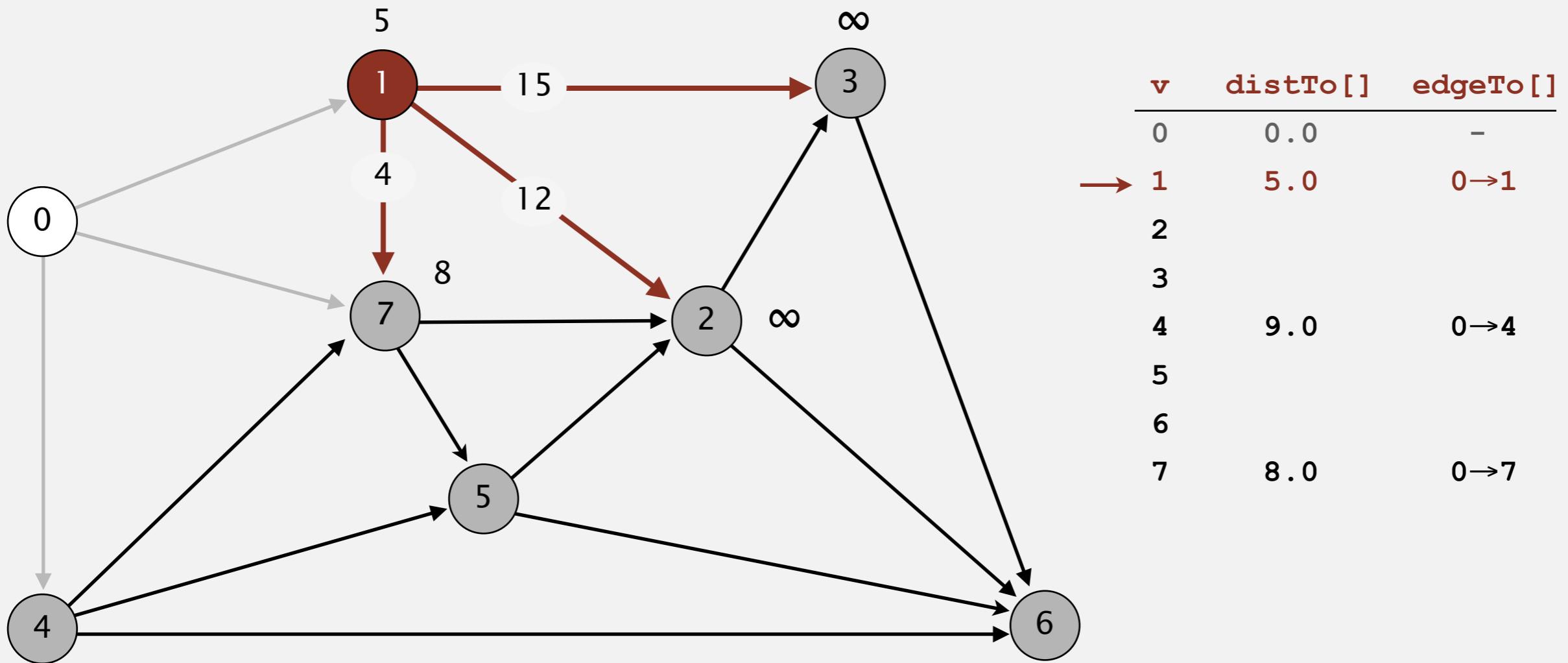
- Consider vertices in increasing order of distance from s (non-tree vertex with the lowest `distTo[]` value).
- Add vertex to tree and relax all edges incident from that vertex.



choose vertex 1

Dijkstra's algorithm

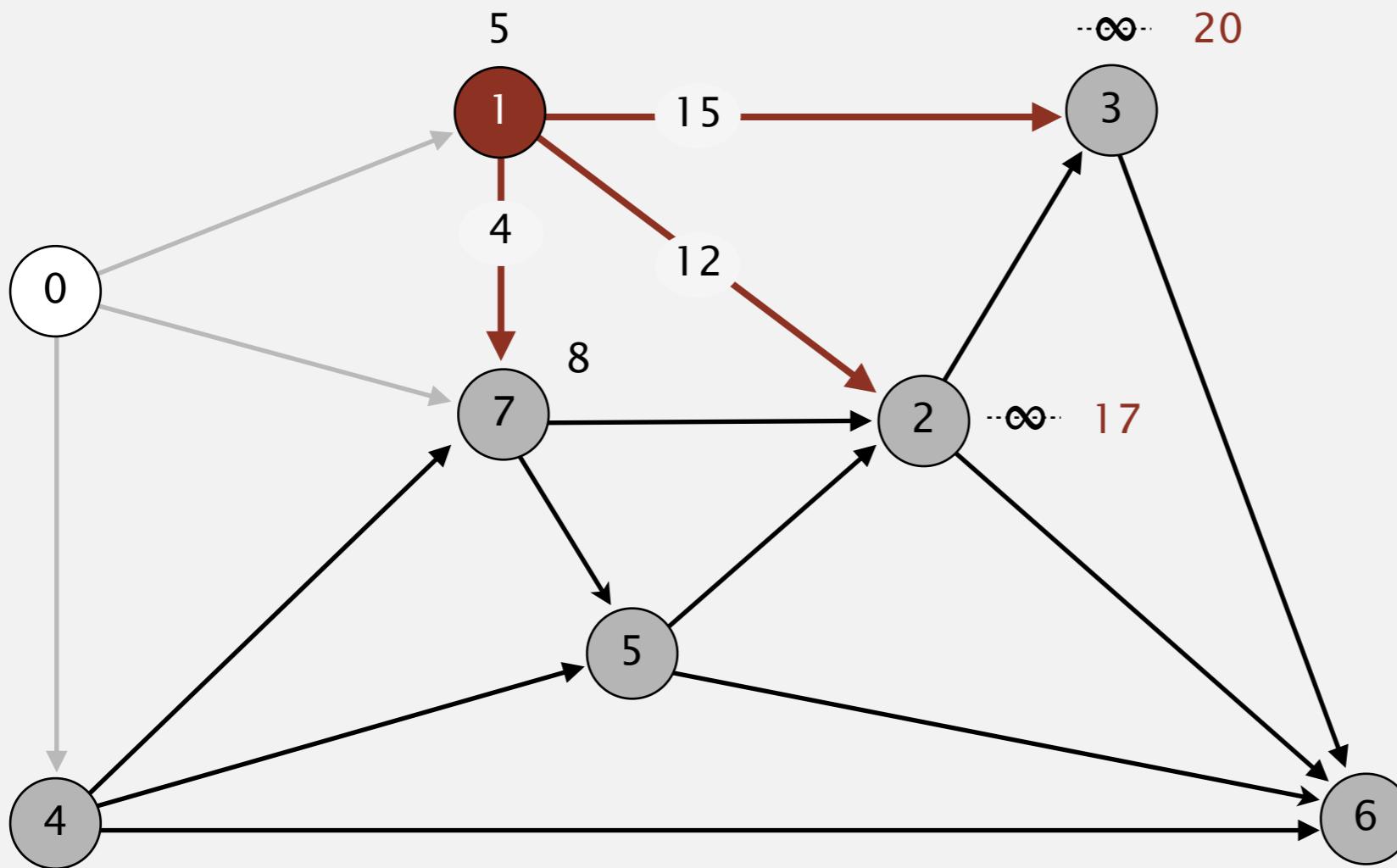
- Consider vertices in increasing order of distance from s (non-tree vertex with the lowest `distTo[]` value).
- Add vertex to tree and relax all edges incident from that vertex.



relax all edges incident from 1

Dijkstra's algorithm

- Consider vertices in increasing order of distance from s (non-tree vertex with the lowest `distTo[]` value).
- Add vertex to tree and relax all edges incident from that vertex.

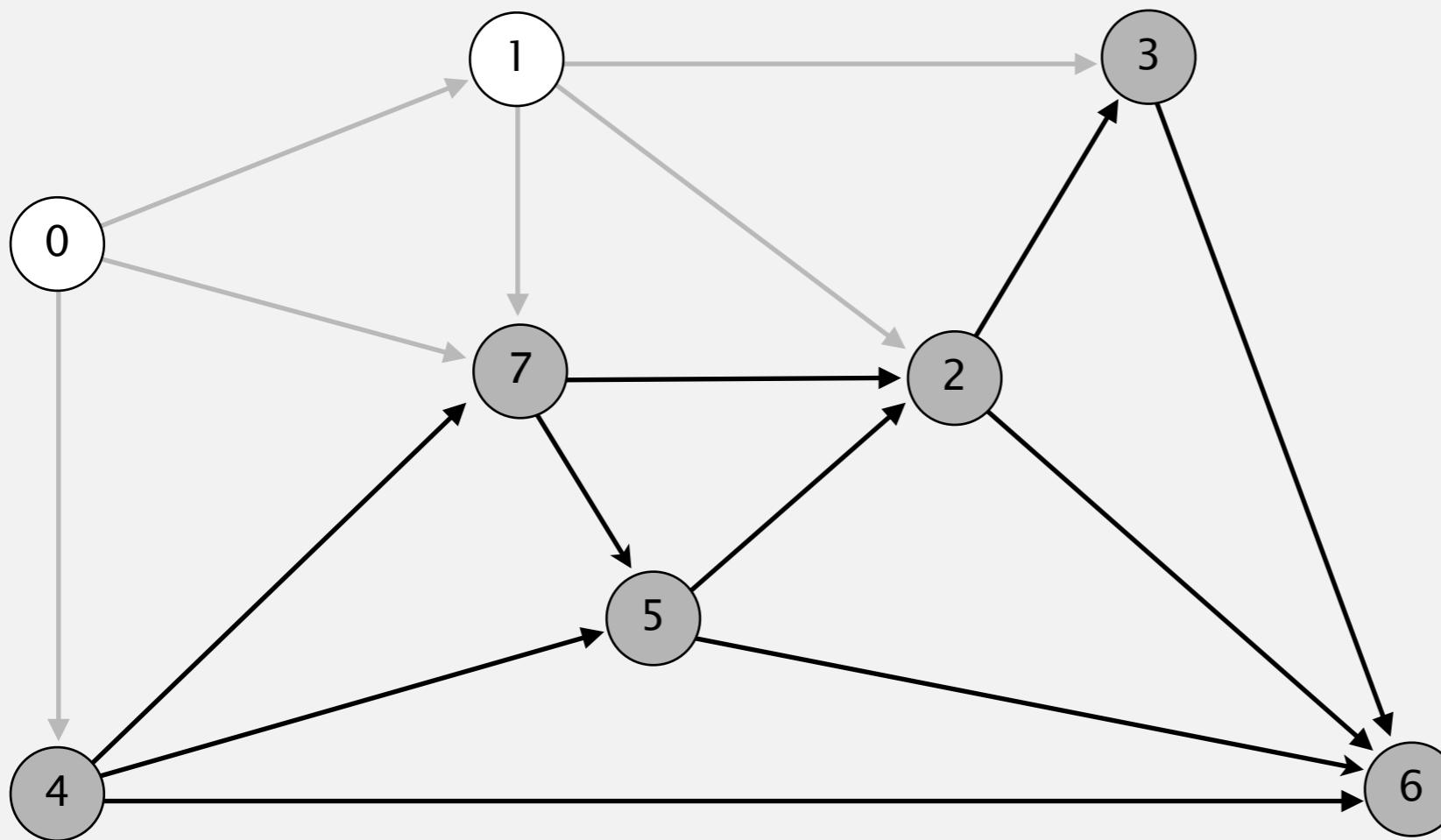


v	distTo []	edgeTo []
0	0.0	-
1	5.0	0→1
2	17.0	1→2
3	20.0	1→3
4	9.0	0→4
5		
6		
7	8.0 ✓	0→7

relax all edges incident from 1

Dijkstra's algorithm

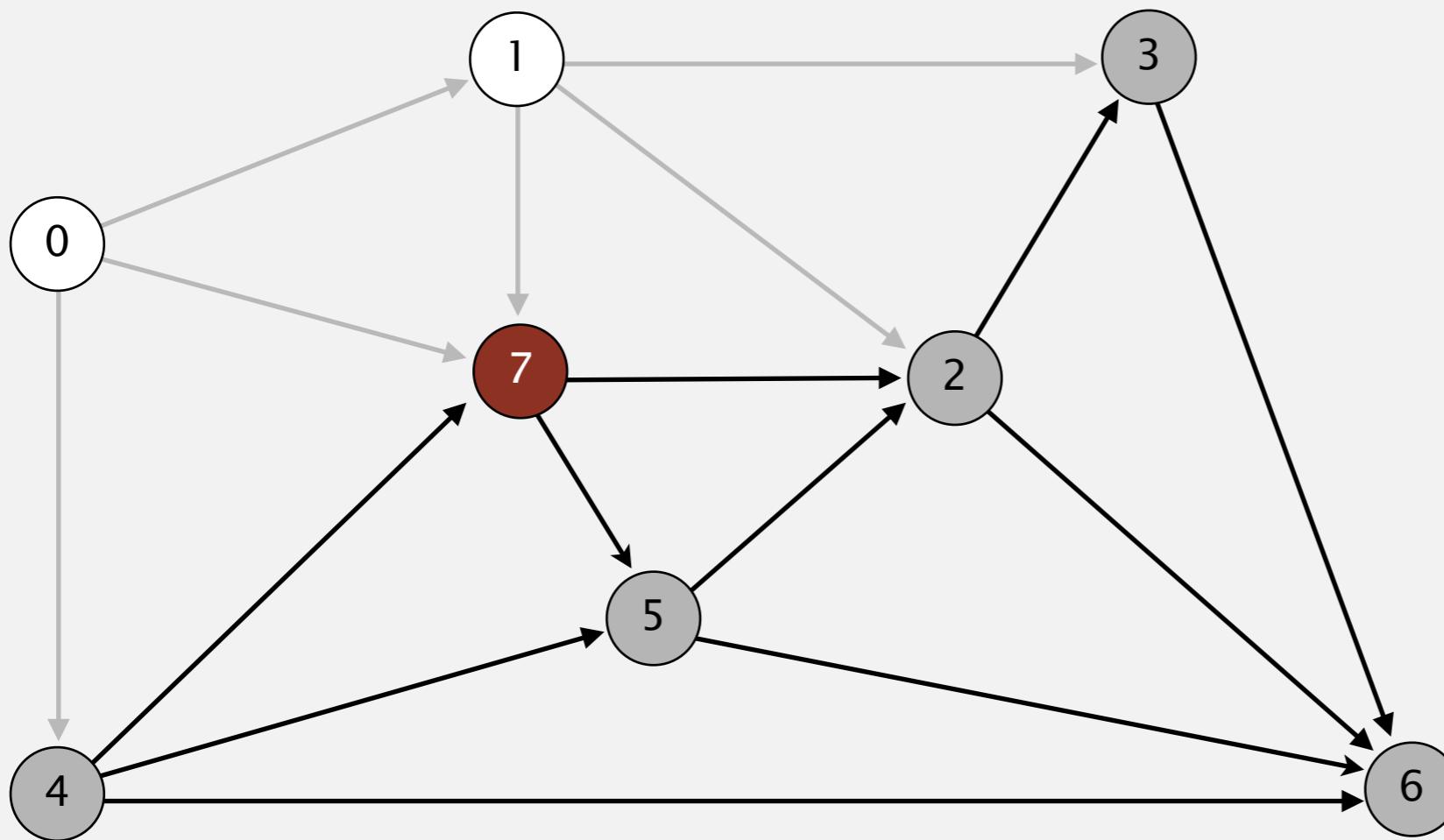
- Consider vertices in increasing order of distance from s (non-tree vertex with the lowest `distTo[]` value).
- Add vertex to tree and relax all edges incident from that vertex.



v	distTo []	edgeTo []
0	0.0	-
1	5.0	0→1
2	17.0	1→2
3	20.0	1→3
4	9.0	0→4
5		
6		
7	8.0	0→7

Dijkstra's algorithm

- Consider vertices in increasing order of distance from s (non-tree vertex with the lowest `distTo[]` value).
- Add vertex to tree and relax all edges incident from that vertex.

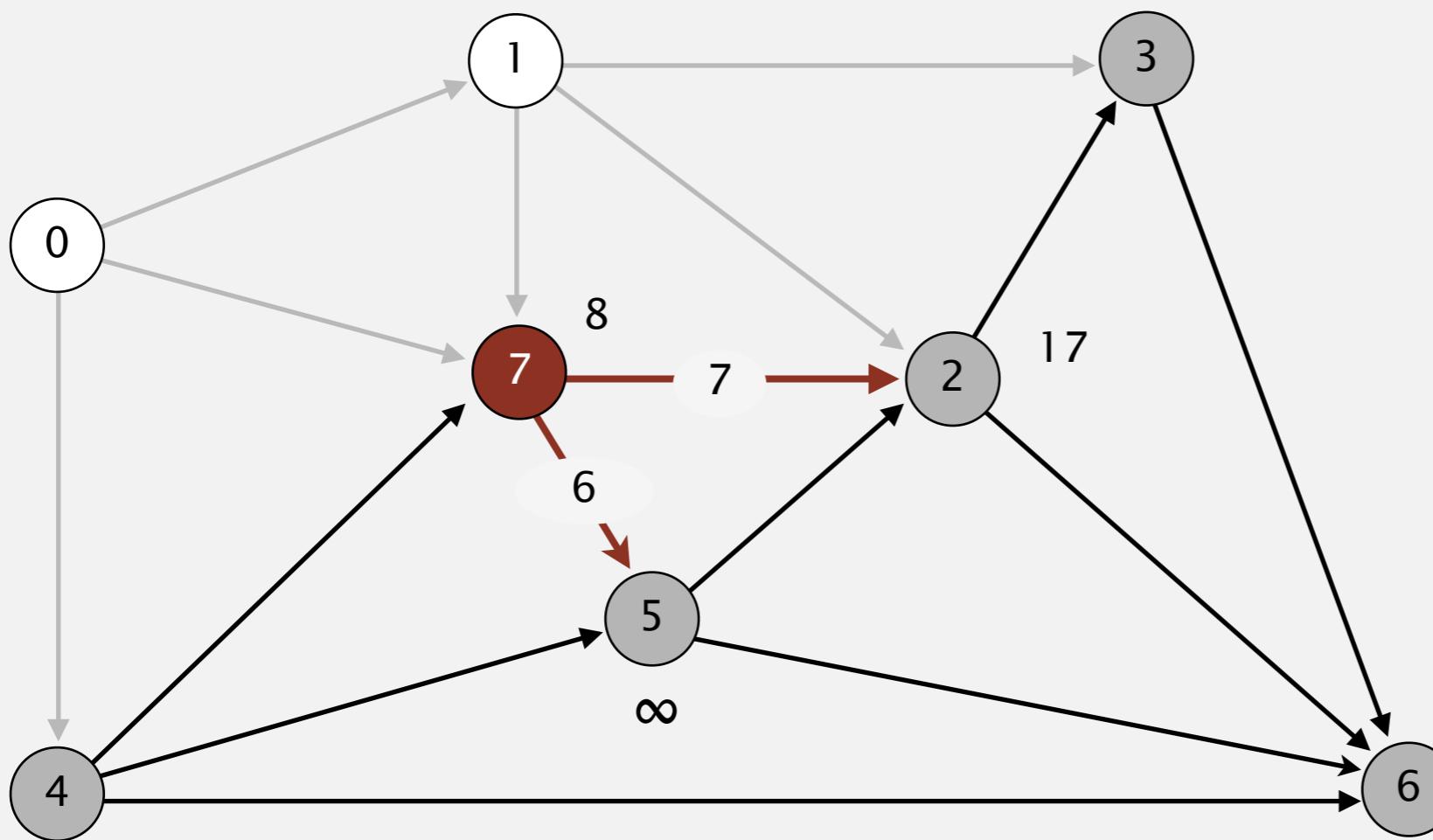


v	distTo []	edgeTo []
0	0.0	-
1	5.0	0→1
2	17.0	1→2
3	20.0	1→3
4	9.0	0→4
5		
6		
7	8.0	0→7

choose vertex 7

Dijkstra's algorithm

- Consider vertices in increasing order of distance from s (non-tree vertex with the lowest `distTo[]` value).
- Add vertex to tree and relax all edges incident from that vertex.

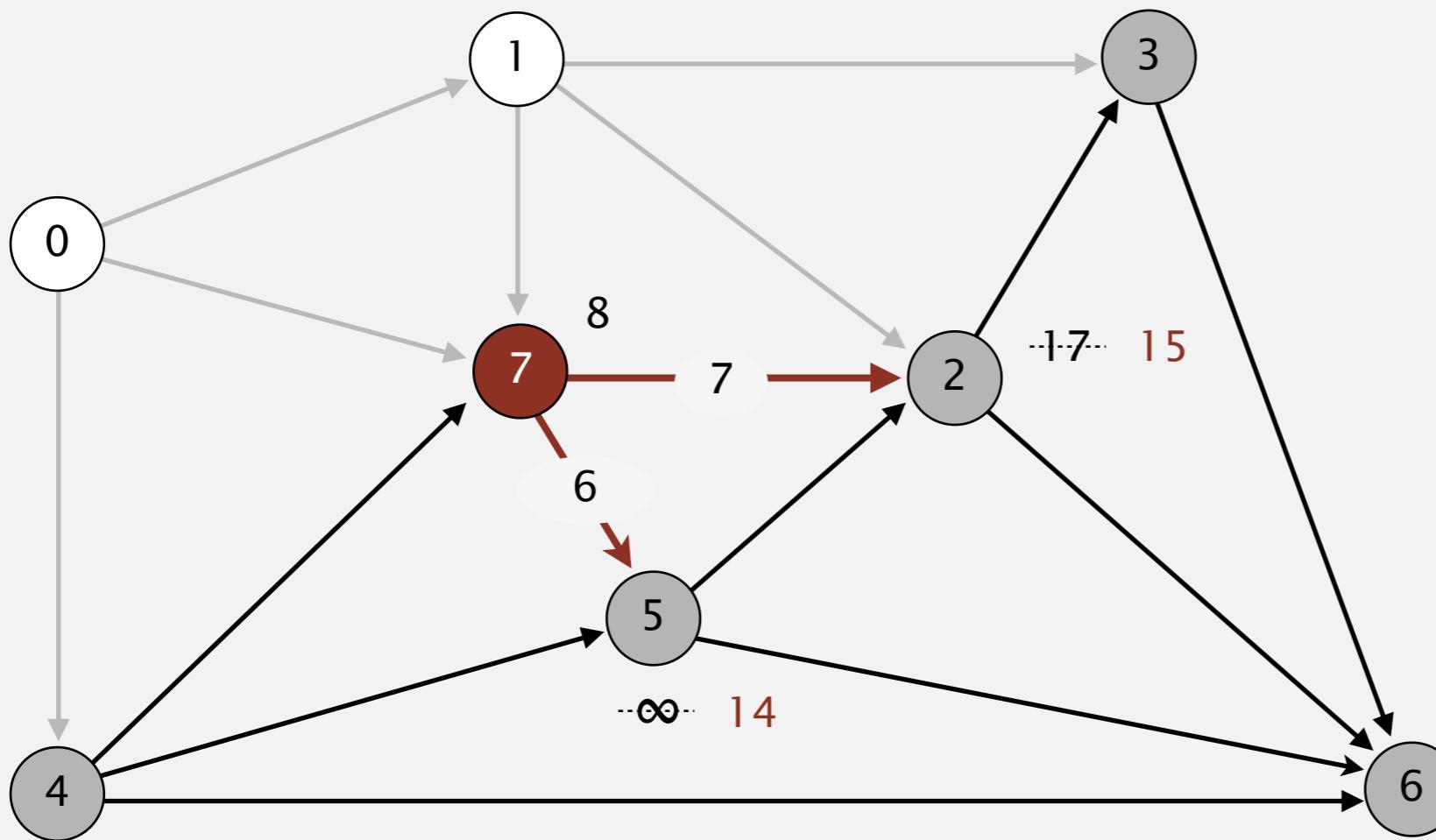


v	distTo []	edgeTo []
0	0.0	-
1	5.0	0→1
2	17.0	1→2
3	20.0	1→3
4	9.0	0→4
5		
6		
7	8.0	0→7

relax all edges incident from 7

Dijkstra's algorithm

- Consider vertices in increasing order of distance from s (non-tree vertex with the lowest `distTo[]` value).
- Add vertex to tree and relax all edges incident from that vertex.

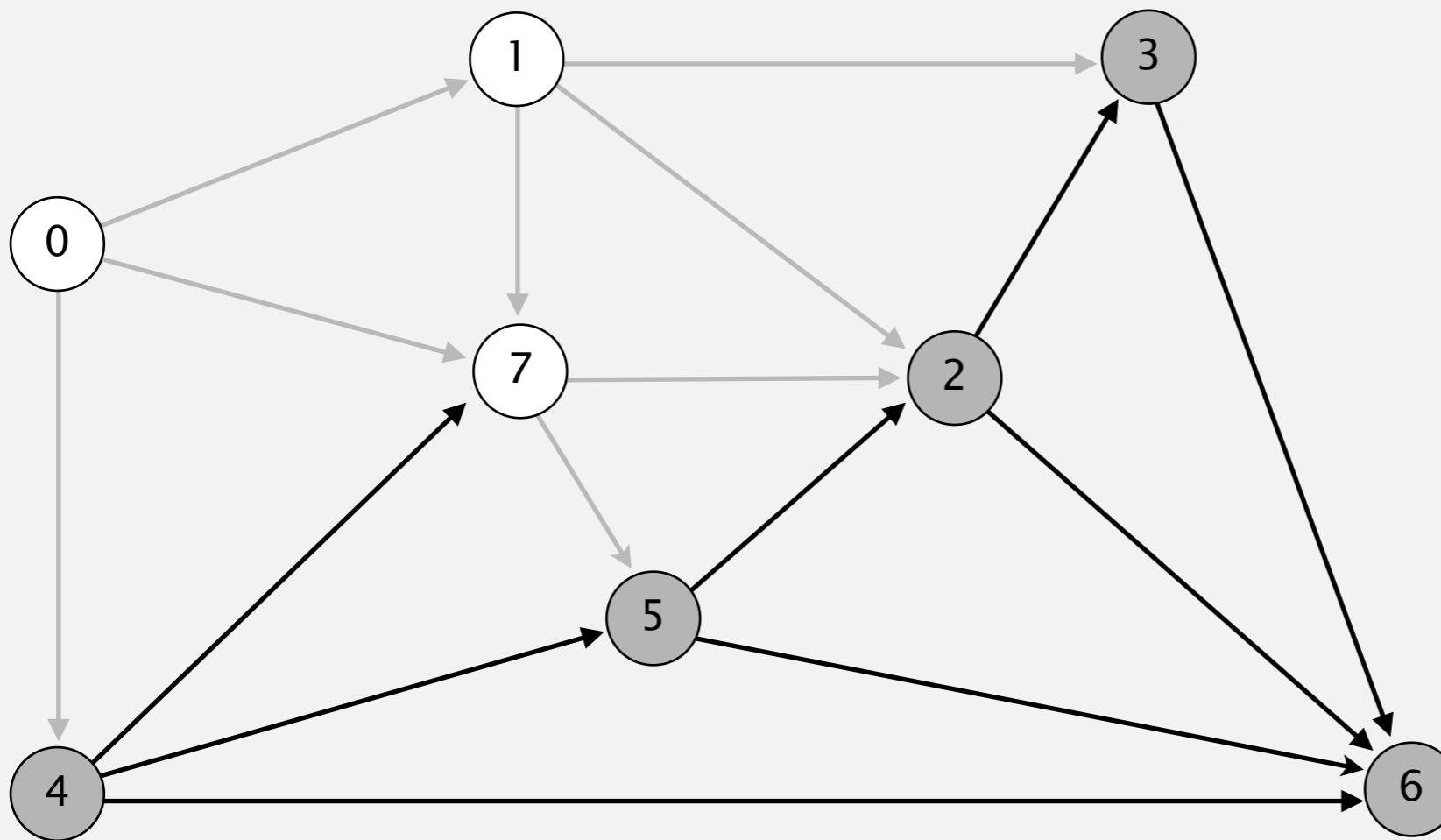


v	distTo []	edgeTo []
0	0.0	-
1	5.0	0→1
2	15.0	7→2
3	20.0	1→3
4	9.0	0→4
5	14.0	7→5
6		
7	8.0	0→7

relax all edges incident from 7

Dijkstra's algorithm

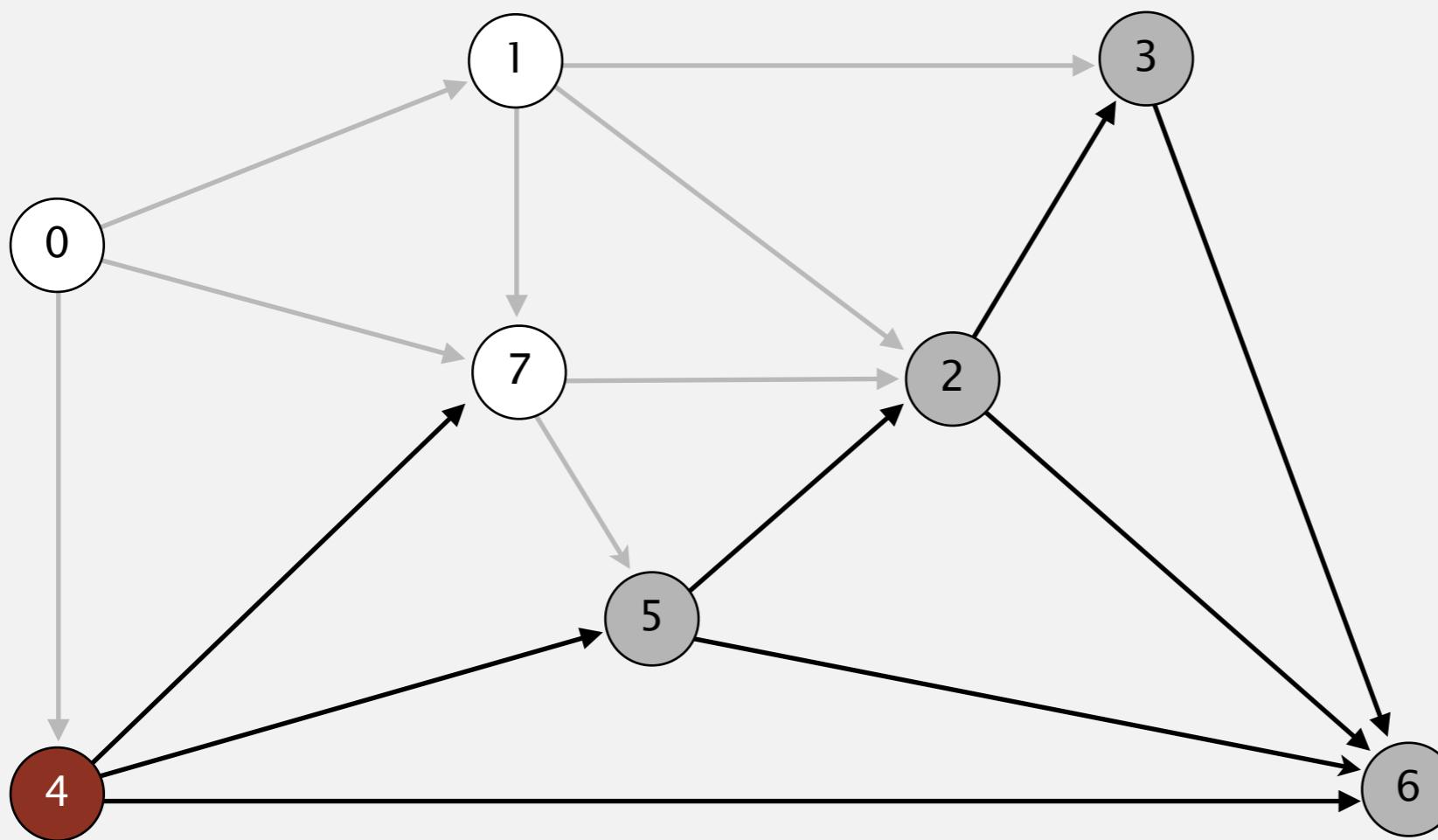
- Consider vertices in increasing order of distance from s (non-tree vertex with the lowest `distTo[]` value).
- Add vertex to tree and relax all edges incident from that vertex.



v	distTo []	edgeTo []
0	0.0	-
1	5.0	0→1
2	15.0	7→2
3	20.0	1→3
4	9.0	0→4
5	14.0	7→5
6		
7	8.0	0→7

Dijkstra's algorithm

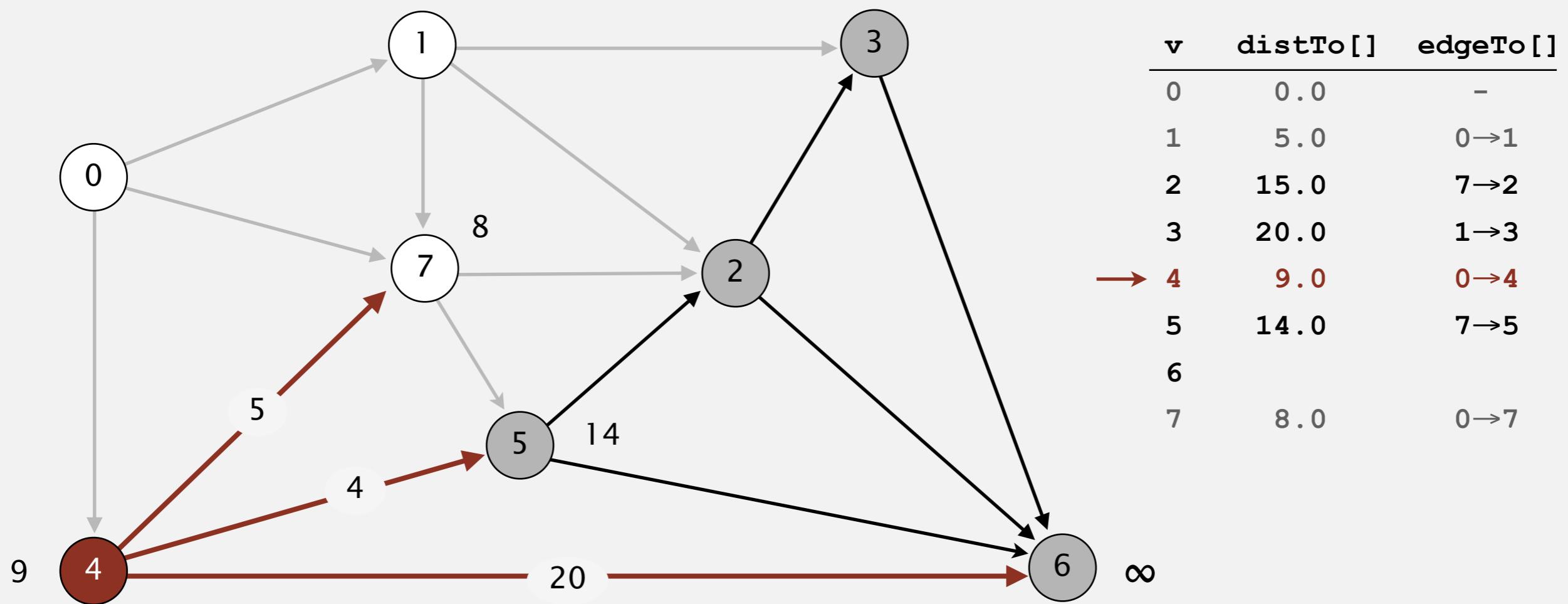
- Consider vertices in increasing order of distance from s (non-tree vertex with the lowest `distTo[]` value).
- Add vertex to tree and relax all edges incident from that vertex.



v	distTo []	edgeTo []
0	0.0	-
1	5.0	0→1
2	15.0	7→2
3	20.0	1→3
4	9.0	0→4
5	14.0	7→5
6		
7	8.0	0→7

Dijkstra's algorithm

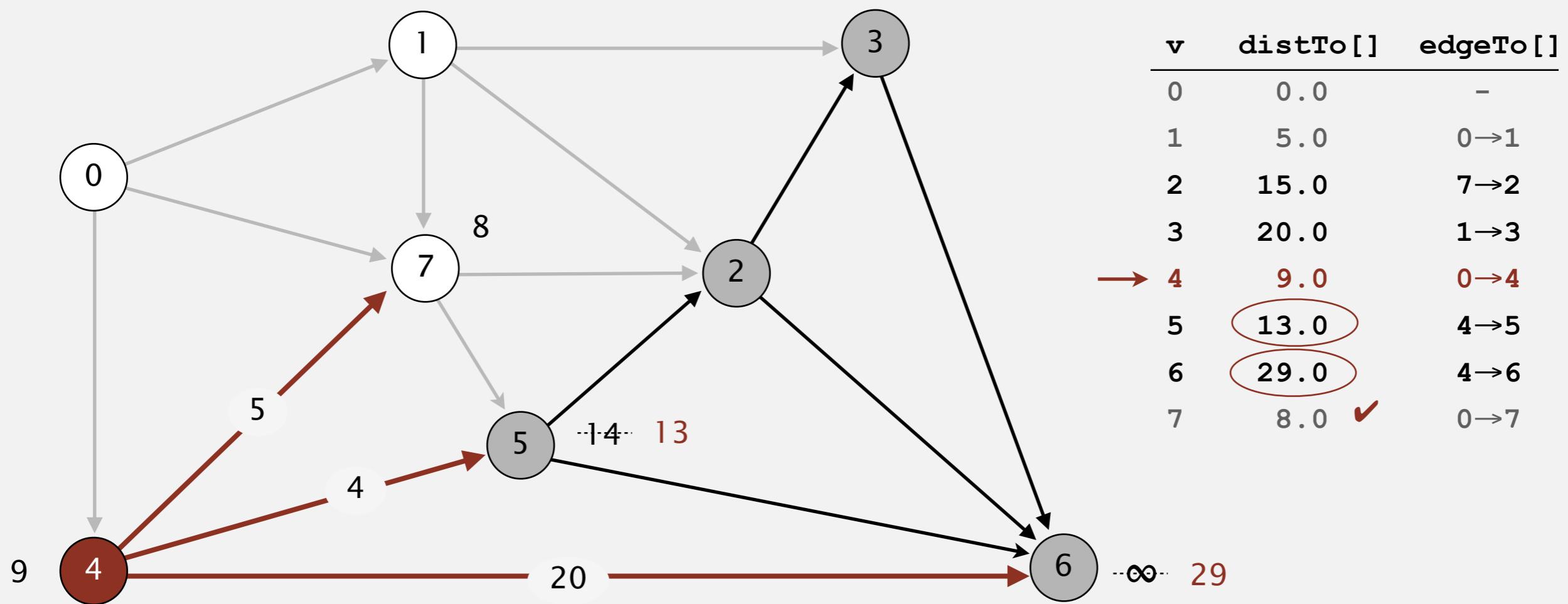
- Consider vertices in increasing order of distance from s (non-tree vertex with the lowest `distTo[]` value).
- Add vertex to tree and relax all edges incident from that vertex.



relax all edges incident from 4

Dijkstra's algorithm

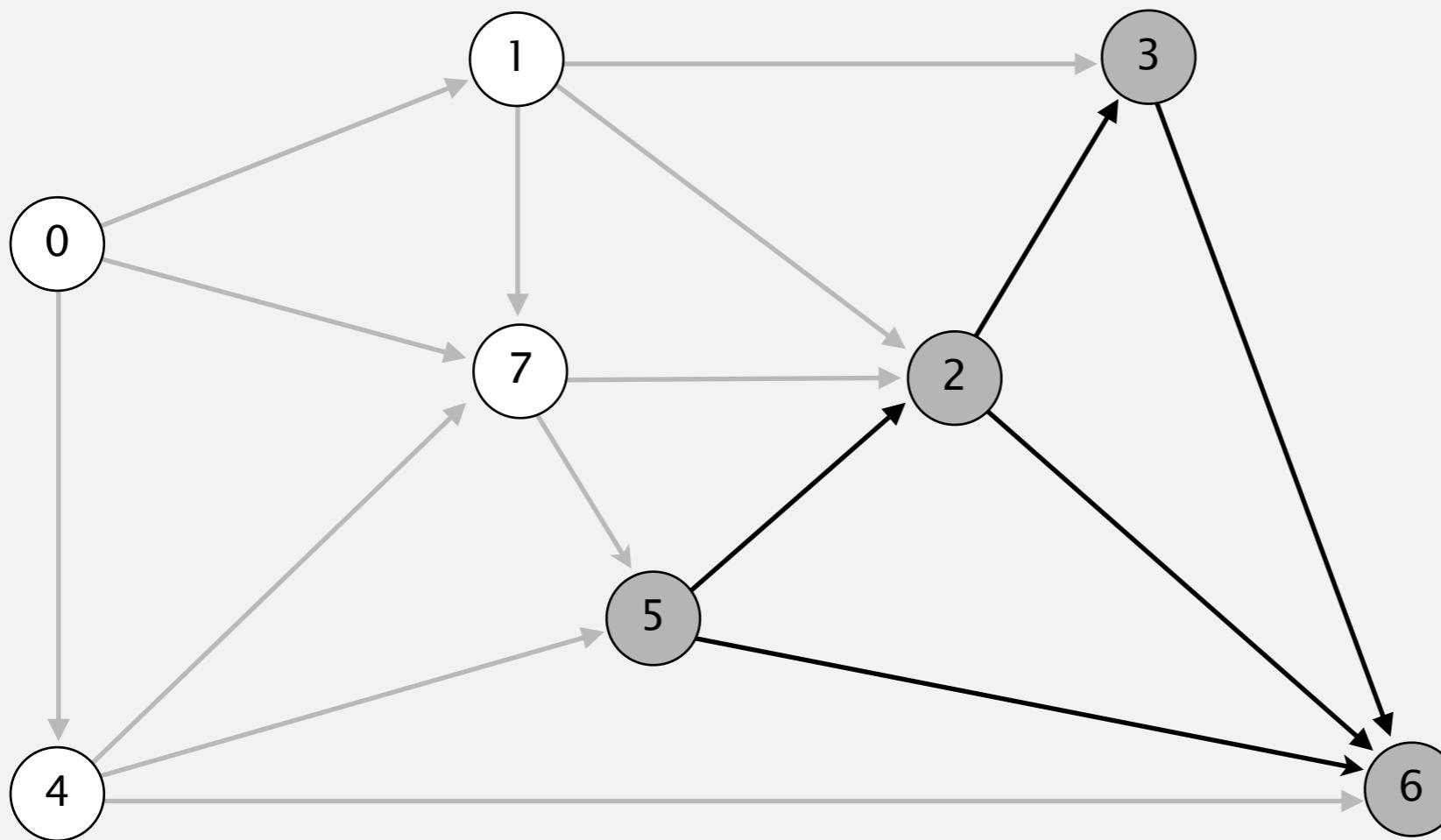
- Consider vertices in increasing order of distance from s (non-tree vertex with the lowest `distTo[]` value).
- Add vertex to tree and relax all edges incident from that vertex.



relax all edges incident from 4

Dijkstra's algorithm

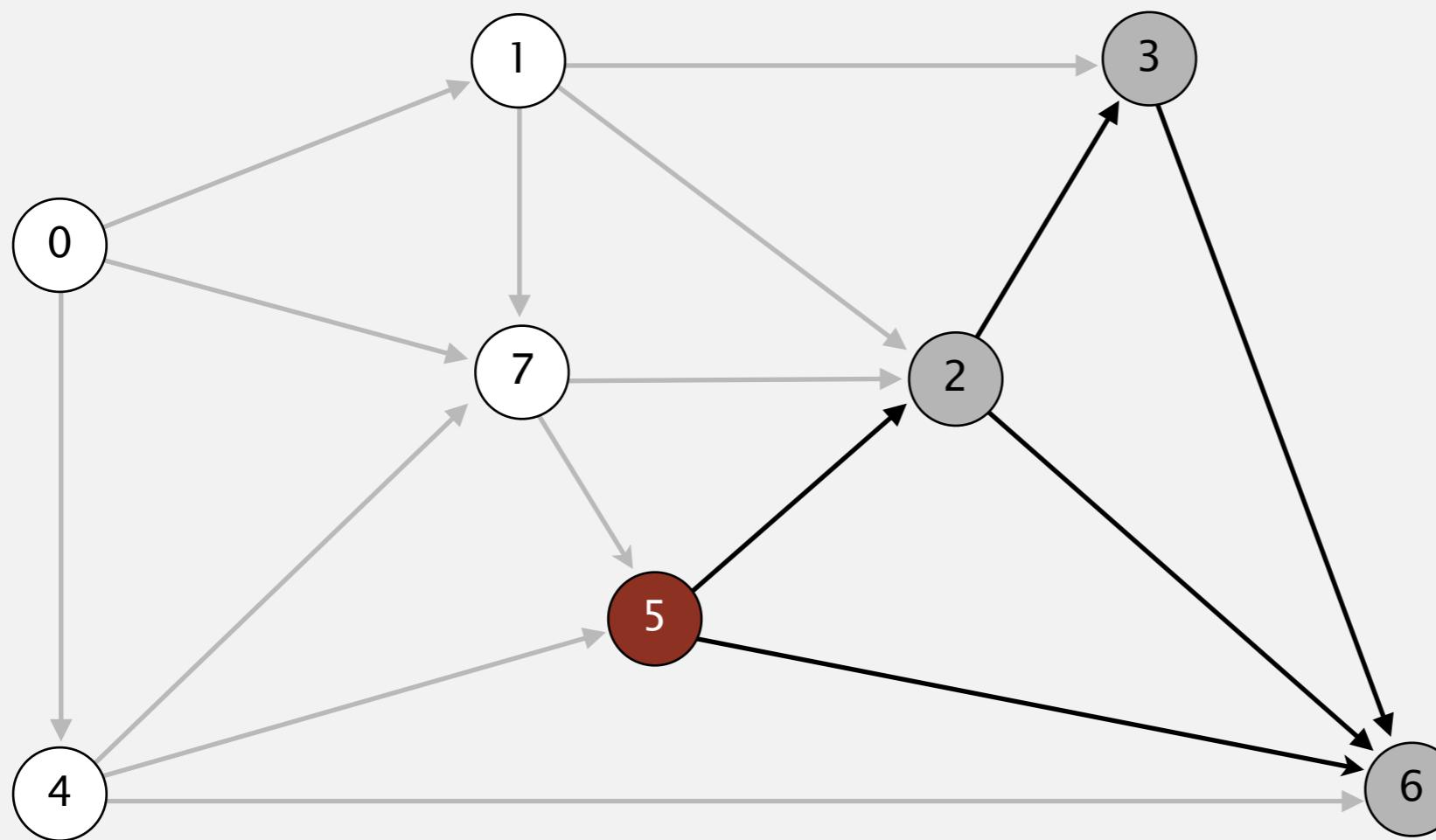
- Consider vertices in increasing order of distance from s (non-tree vertex with the lowest `distTo[]` value).
- Add vertex to tree and relax all edges incident from that vertex.



v	distTo []	edgeTo []
0	0.0	-
1	5.0	0→1
2	15.0	7→2
3	20.0	1→3
4	9.0	0→4
5	13.0	4→5
6	29.0	4→6
7	8.0	0→7

Dijkstra's algorithm

- Consider vertices in increasing order of distance from s (non-tree vertex with the lowest `distTo[]` value).
- Add vertex to tree and relax all edges incident from that vertex.

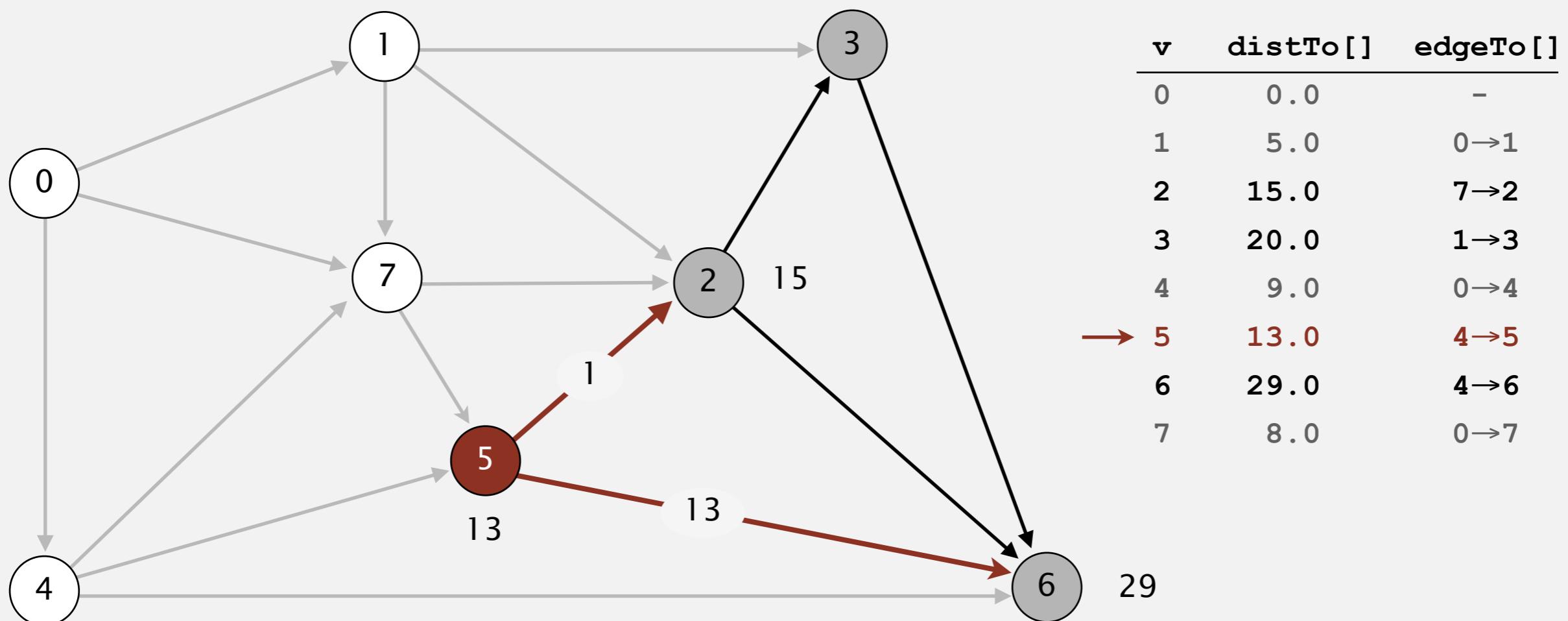


select vertex 5

v	distTo []	edgeTo []
0	0.0	-
1	5.0	0→1
2	15.0	7→2
3	20.0	1→3
4	9.0	0→4
5	13.0	4→5
6	29.0	4→6
7	8.0	0→7

Dijkstra's algorithm

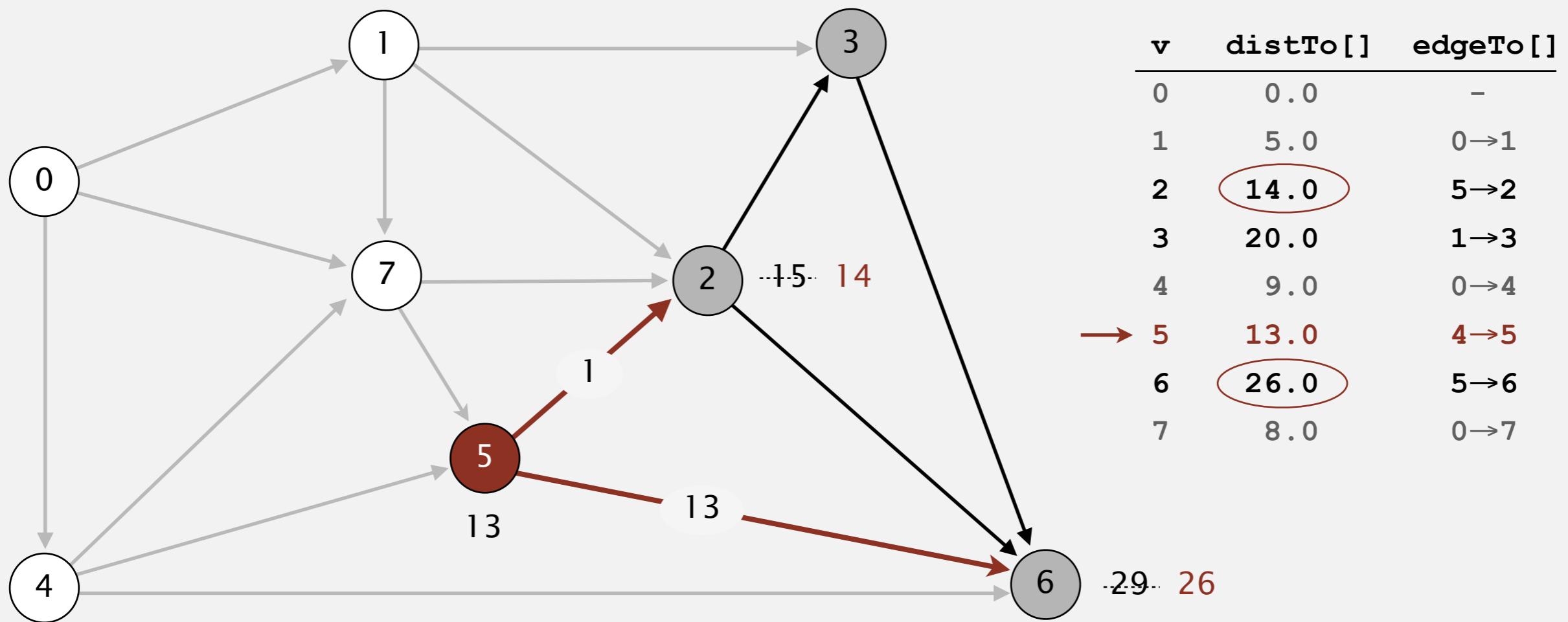
- Consider vertices in increasing order of distance from s (non-tree vertex with the lowest `distTo[]` value).
- Add vertex to tree and relax all edges incident from that vertex.



relax all edges incident from 5

Dijkstra's algorithm

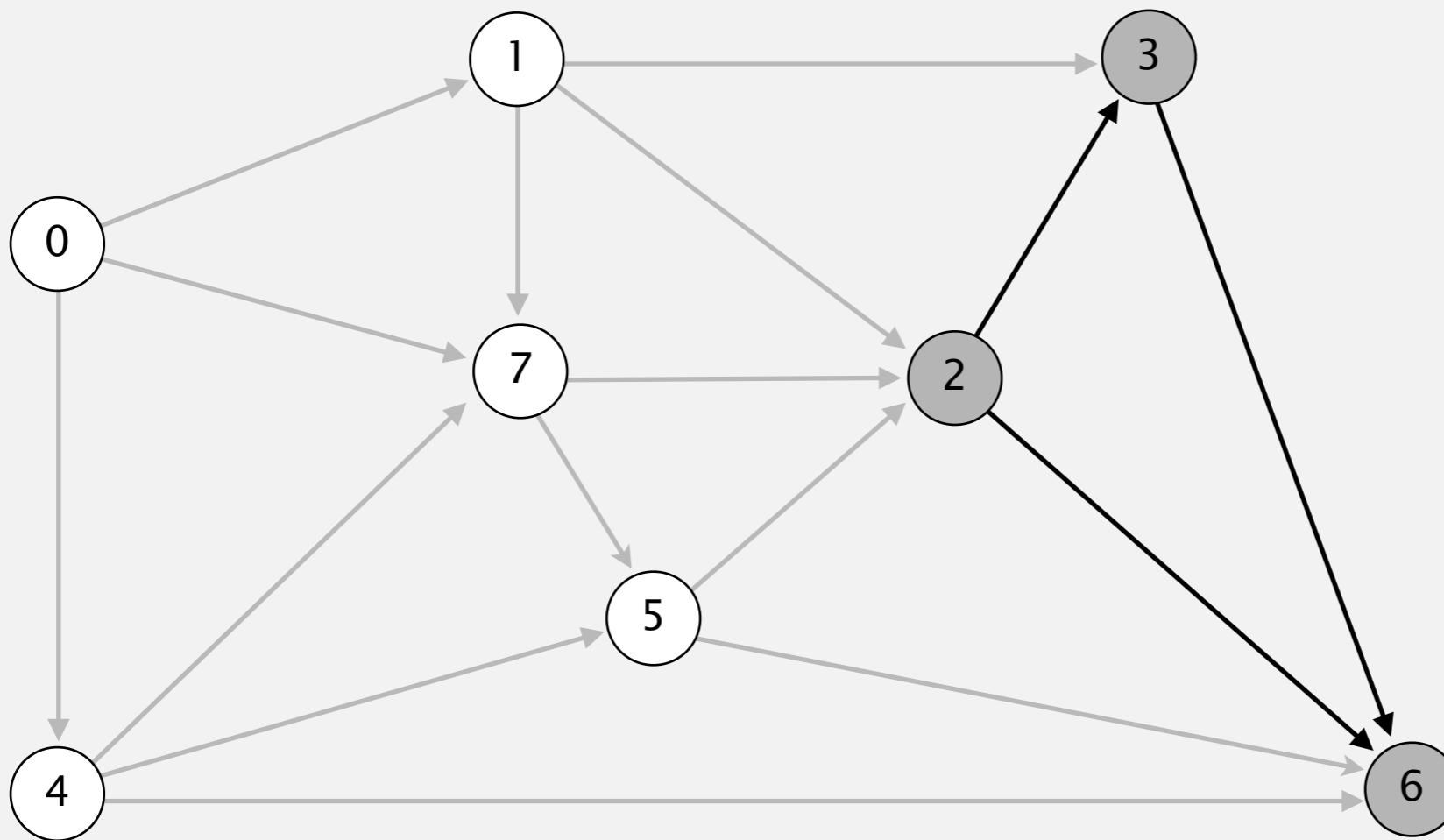
- Consider vertices in increasing order of distance from s (non-tree vertex with the lowest `distTo[]` value).
- Add vertex to tree and relax all edges incident from that vertex.



relax all edges incident from 5

Dijkstra's algorithm

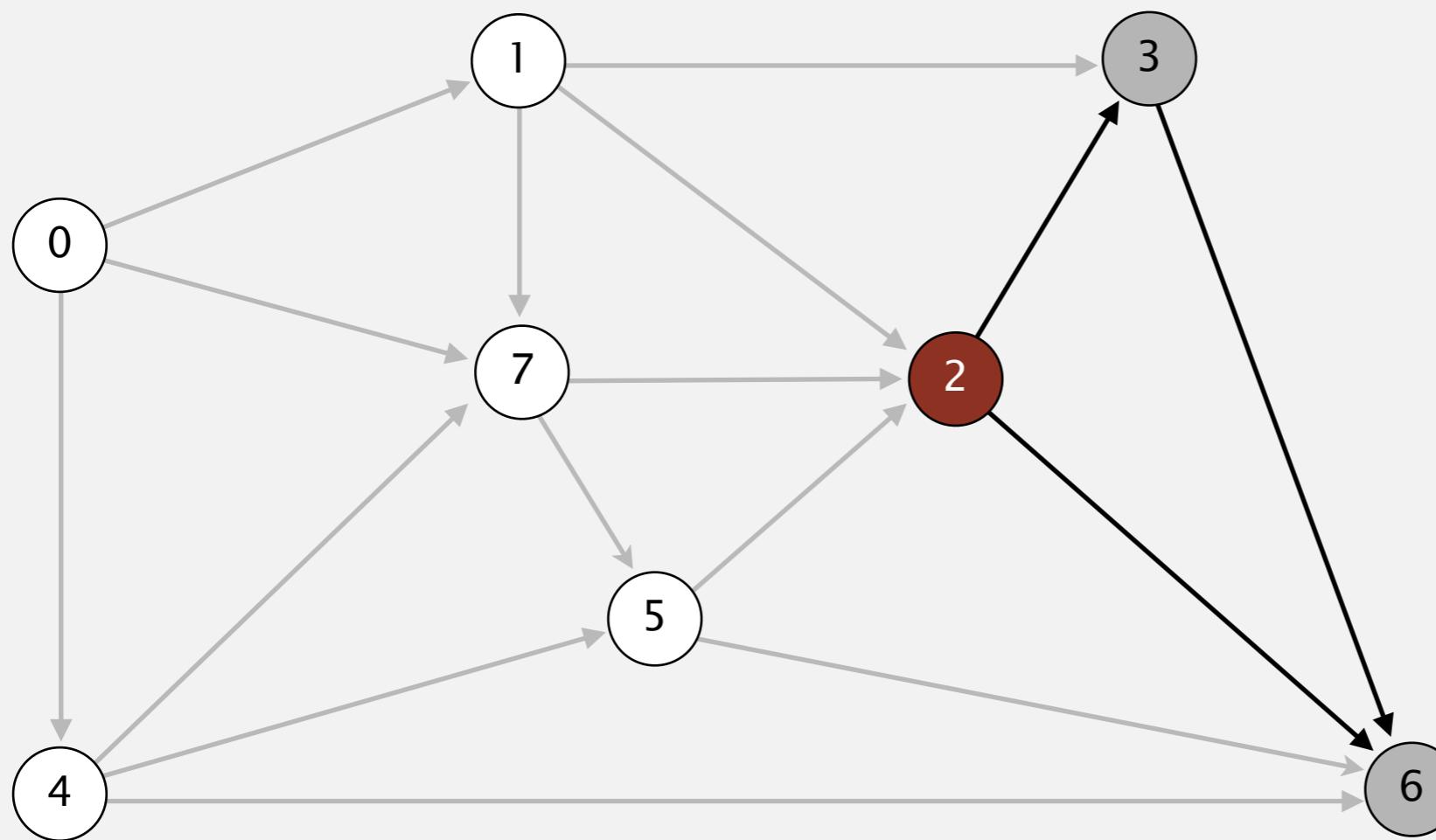
- Consider vertices in increasing order of distance from s (non-tree vertex with the lowest `distTo[]` value).
- Add vertex to tree and relax all edges incident from that vertex.



v	distTo []	edgeTo []
0	0.0	-
1	5.0	0→1
2	14.0	5→2
3	20.0	1→3
4	9.0	0→4
5	13.0	4→5
6	26.0	5→6
7	8.0	0→7

Dijkstra's algorithm

- Consider vertices in increasing order of distance from s (non-tree vertex with the lowest `distTo[]` value).
- Add vertex to tree and relax all edges incident from that vertex.

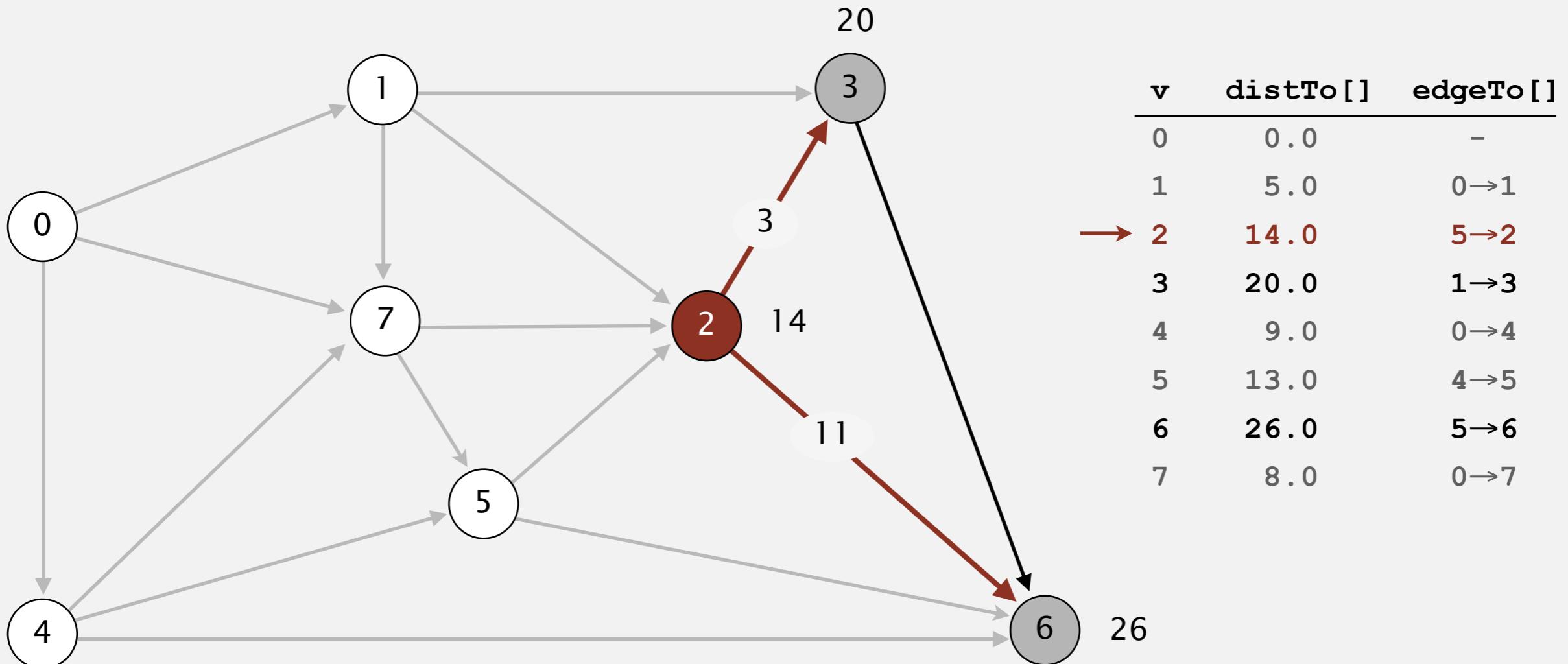


v	distTo []	edgeTo []
0	0.0	-
1	5.0	0→1
2	14.0	5→2
3	20.0	1→3
4	9.0	0→4
5	13.0	4→5
6	26.0	5→6
7	8.0	0→7

select vertex 2

Dijkstra's algorithm

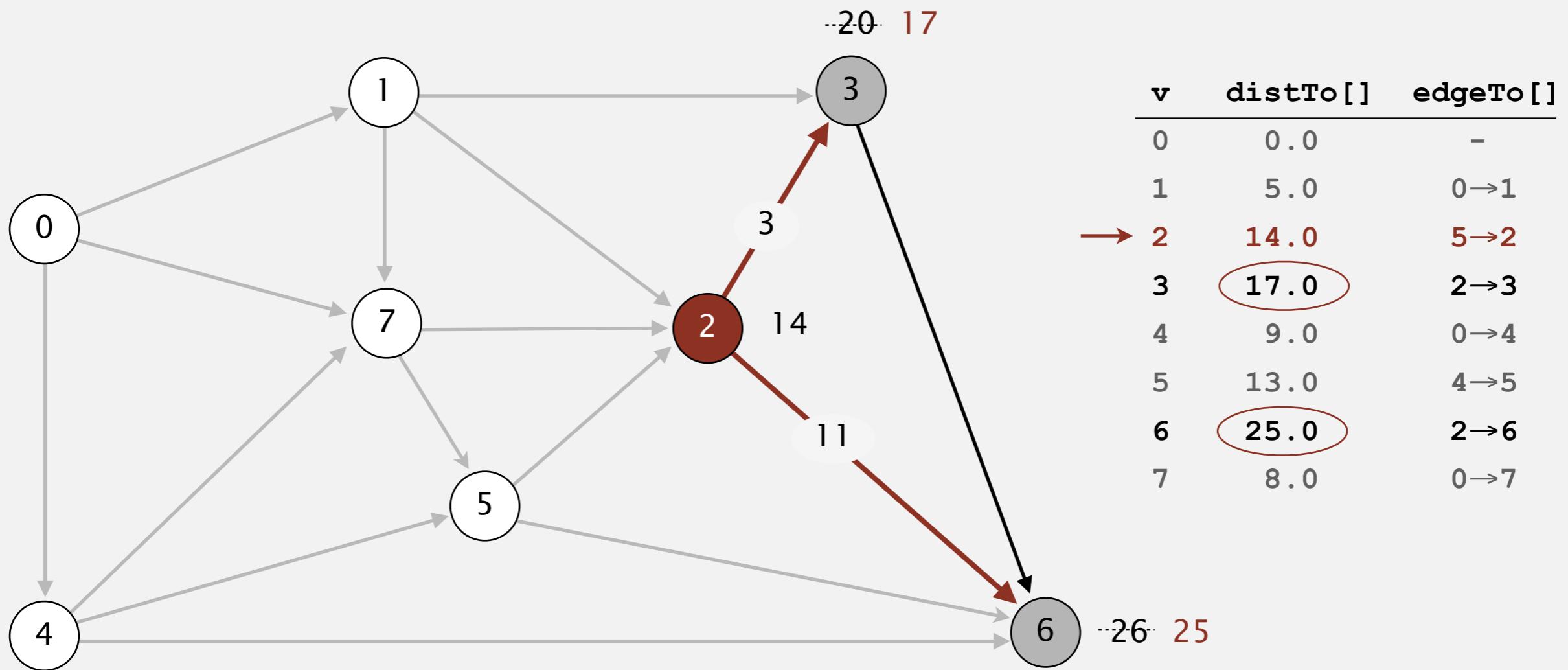
- Consider vertices in increasing order of distance from s (non-tree vertex with the lowest `distTo[]` value).
- Add vertex to tree and relax all edges incident from that vertex.



relax all edges incident from 2

Dijkstra's algorithm

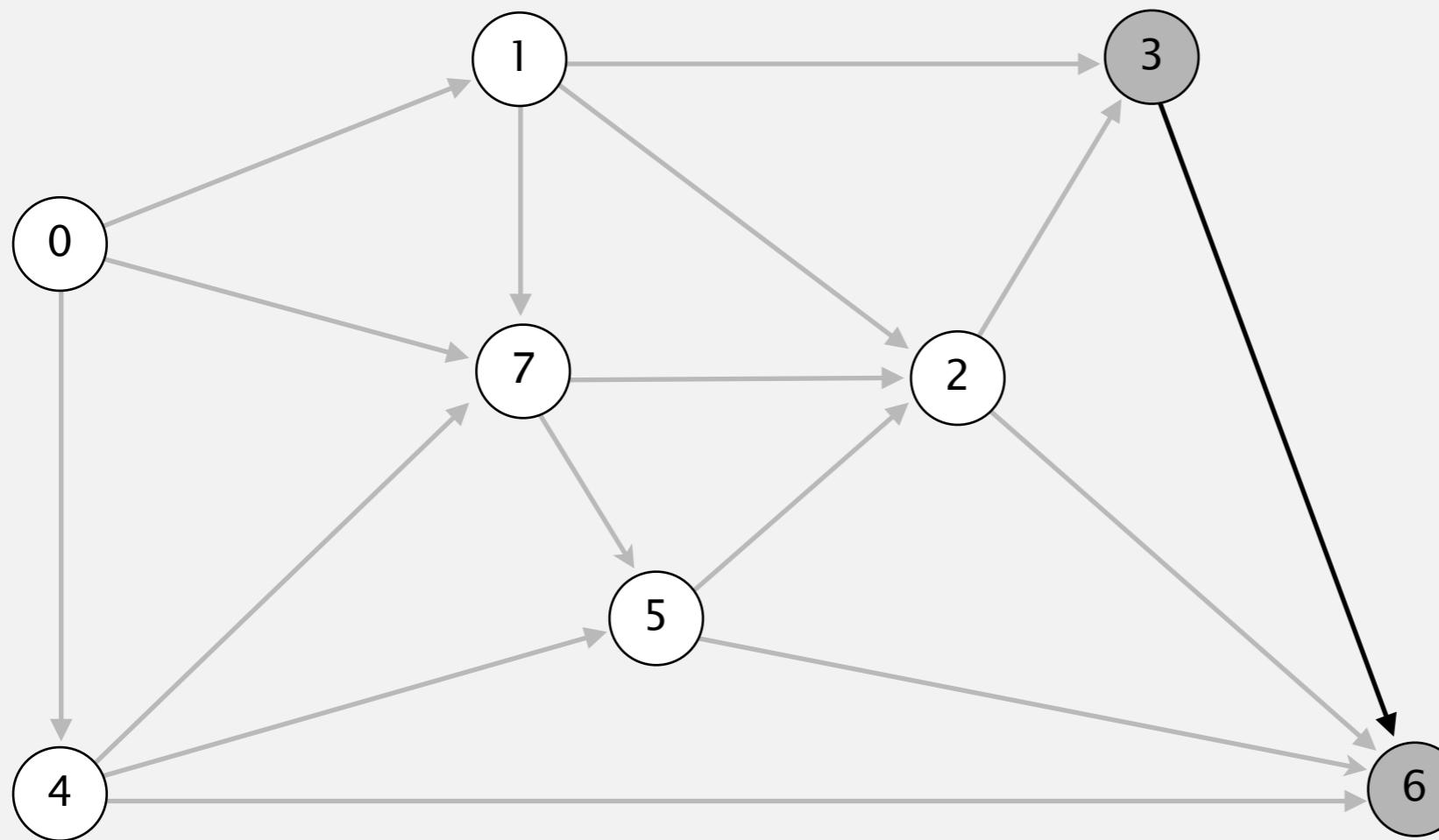
- Consider vertices in increasing order of distance from s (non-tree vertex with the lowest `distTo[]` value).
- Add vertex to tree and relax all edges incident from that vertex.



relax all edges incident from 2

Dijkstra's algorithm

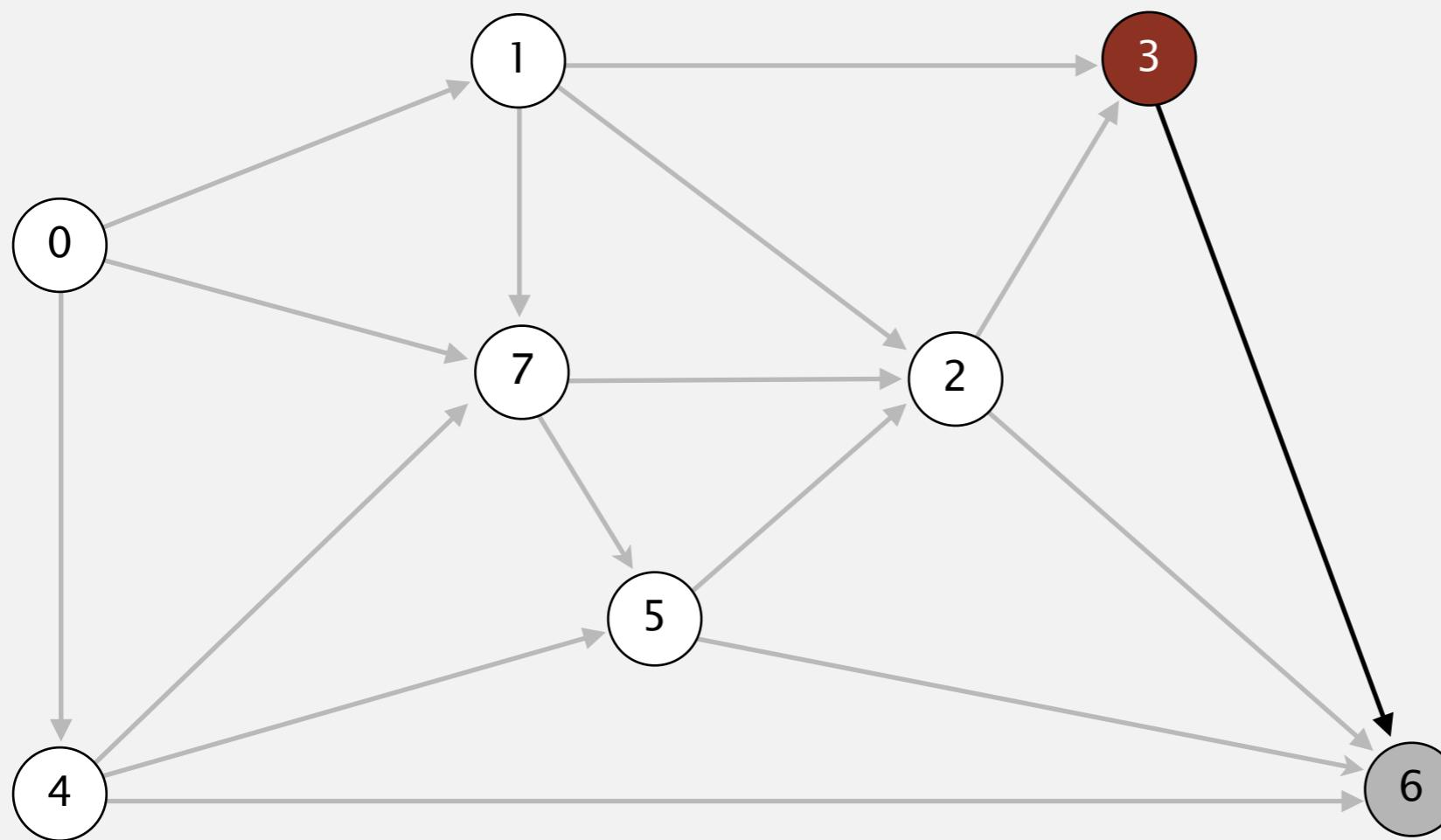
- Consider vertices in increasing order of distance from s (non-tree vertex with the lowest `distTo[]` value).
- Add vertex to tree and relax all edges incident from that vertex.



v	distTo []	edgeTo []
0	0.0	-
1	5.0	0→1
2	14.0	5→2
3	17.0	2→3
4	9.0	0→4
5	13.0	4→5
6	25.0	2→6
7	8.0	0→7

Dijkstra's algorithm

- Consider vertices in increasing order of distance from s (non-tree vertex with the lowest `distTo[]` value).
- Add vertex to tree and relax all edges incident from that vertex.

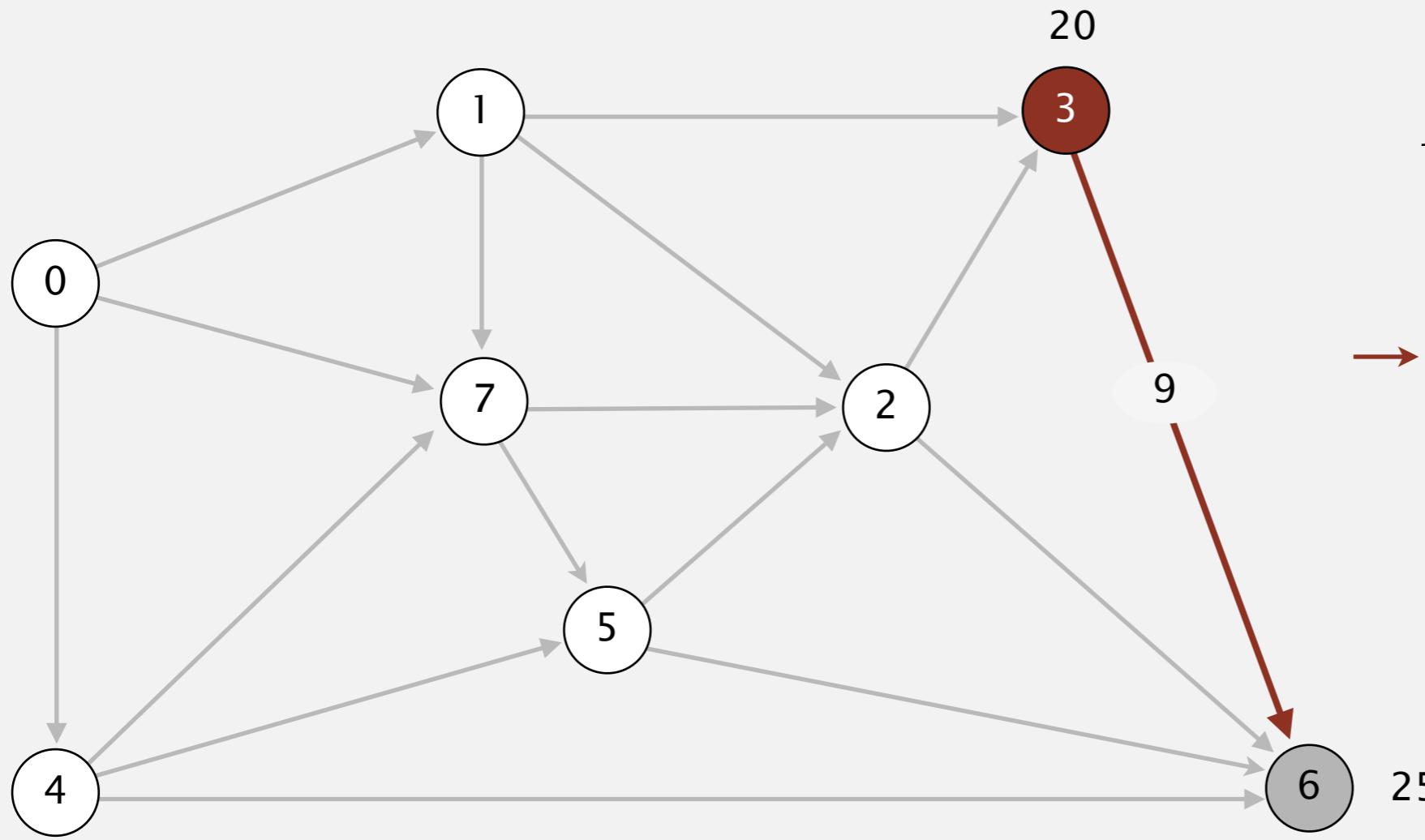


v	distTo []	edgeTo []
0	0.0	-
1	5.0	0→1
2	14.0	5→2
3	17.0	2→3
4	9.0	0→4
5	13.0	4→5
6	25.0	2→6
7	8.0	0→7

select vertex 3

Dijkstra's algorithm

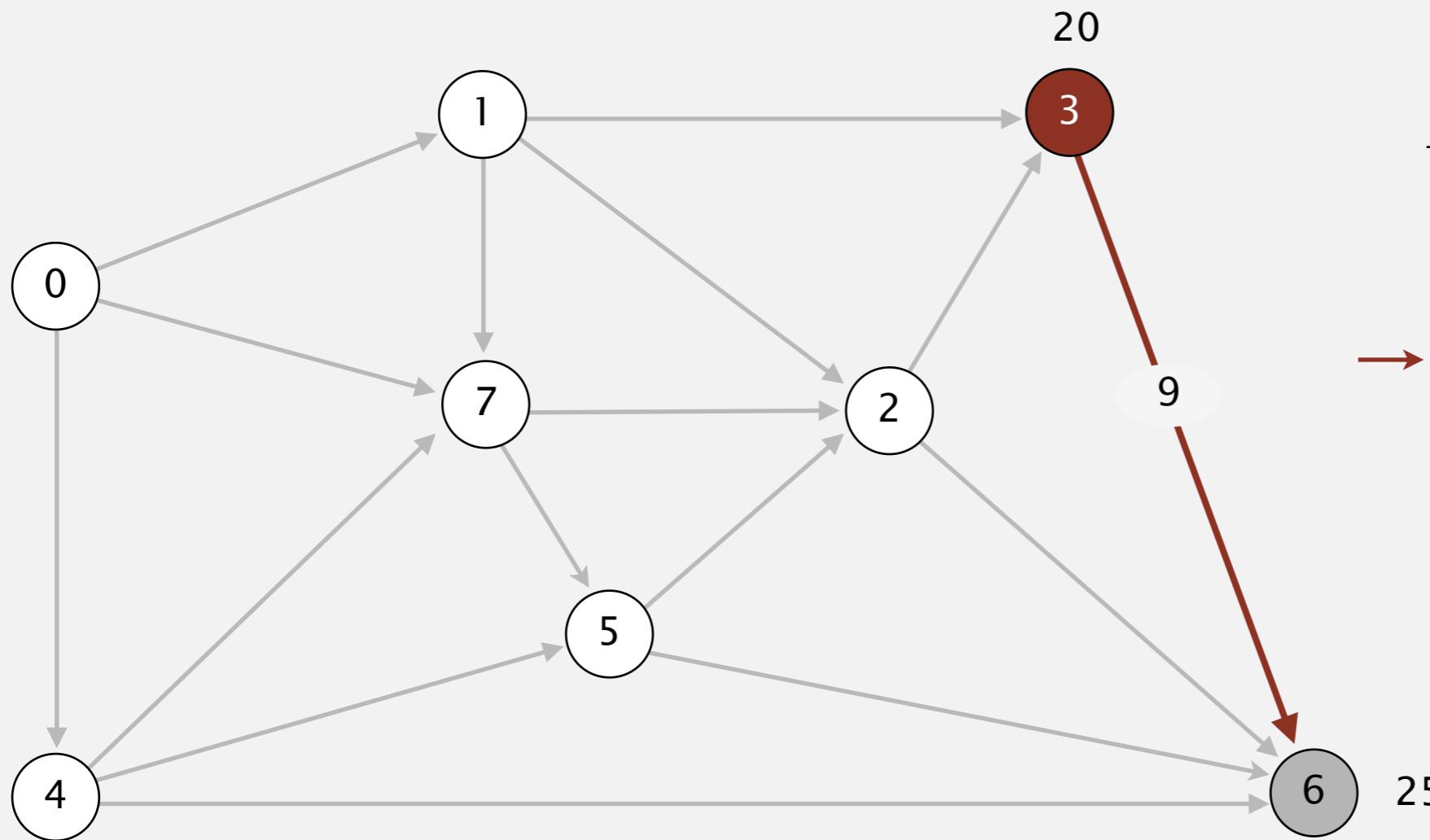
- Consider vertices in increasing order of distance from s (non-tree vertex with the lowest `distTo[]` value).
- Add vertex to tree and relax all edges incident from that vertex.



v	distTo []	edgeTo []
0	0.0	-
1	5.0	0→1
2	14.0	5→2
3	17.0	2→3
4	9.0	0→4
5	13.0	4→5
6	25.0	2→6
7	8.0	0→7

Dijkstra's algorithm

- Consider vertices in increasing order of distance from s (non-tree vertex with the lowest `distTo[]` value).
- Add vertex to tree and relax all edges incident from that vertex.

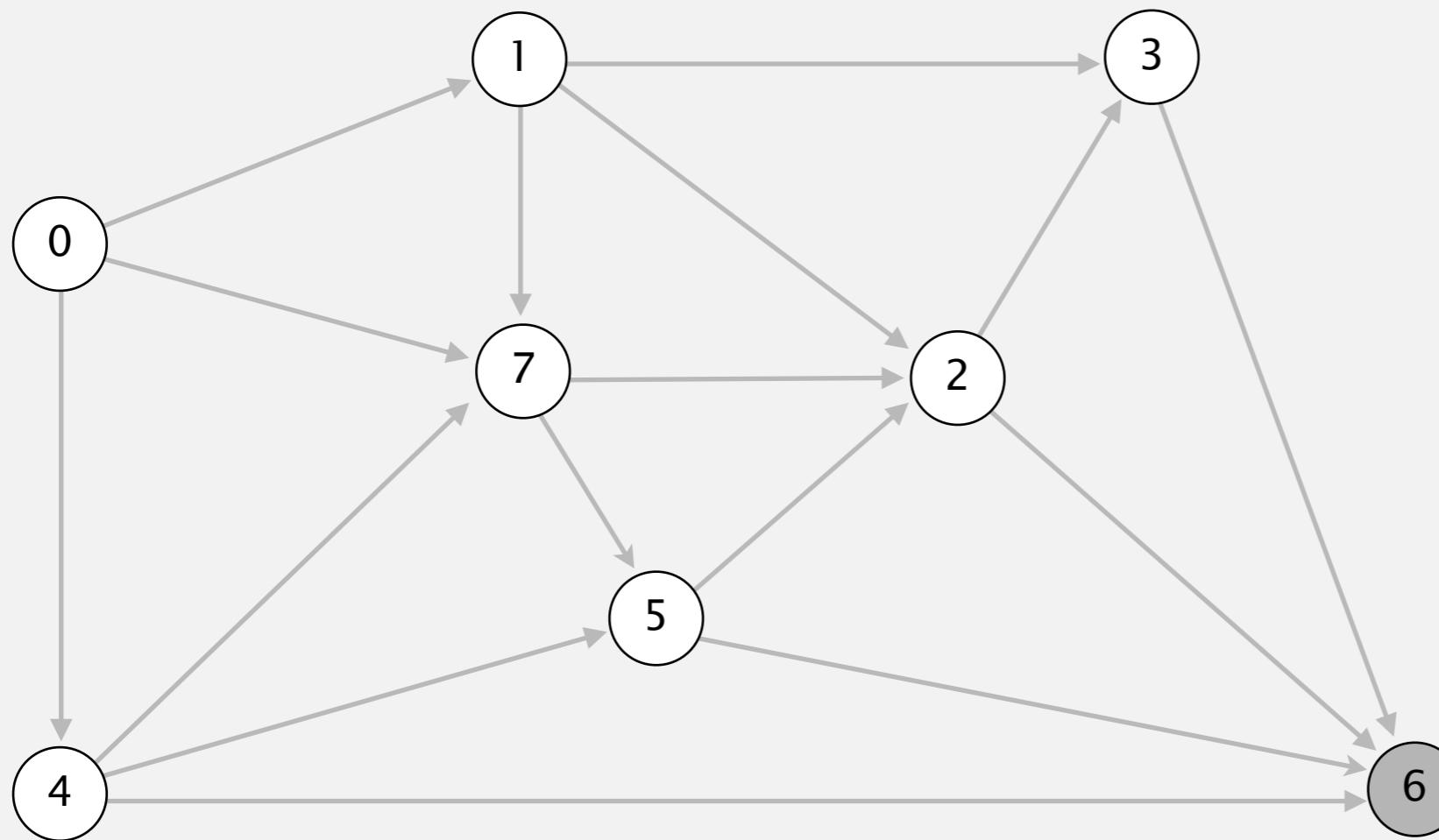


v	distTo []	edgeTo []
0	0.0	-
1	5.0	0→1
2	14.0	5→2
3	17.0	2→3
4	9.0	0→4
5	13.0	4→5
6	25.0 ✓	2→6
7	8.0	0→7

relax all edges incident from 3

Dijkstra's algorithm

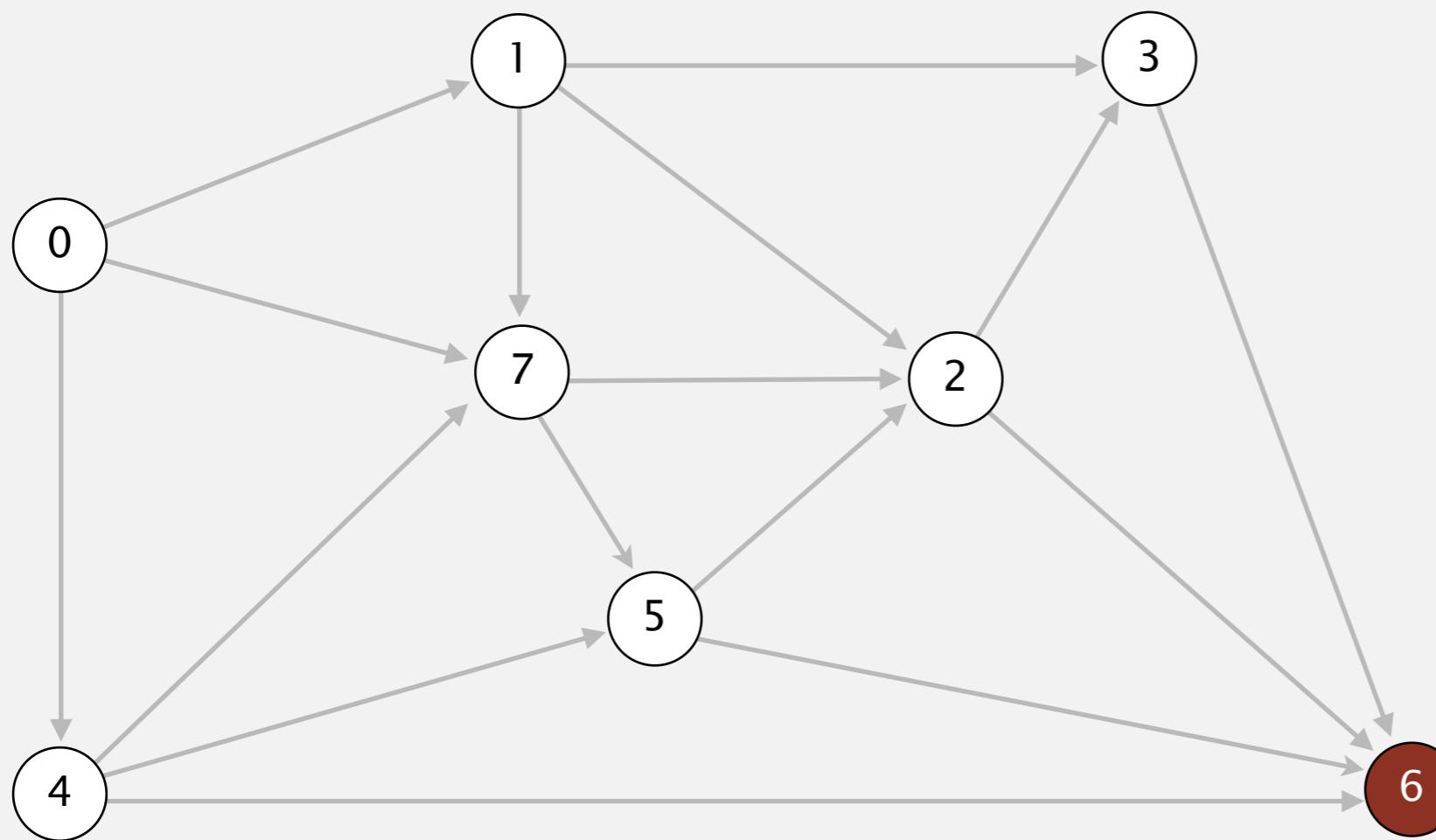
- Consider vertices in increasing order of distance from s (non-tree vertex with the lowest `distTo[]` value).
- Add vertex to tree and relax all edges incident from that vertex.



v	distTo []	edgeTo []
0	0.0	-
1	5.0	0→1
2	14.0	5→2
3	17.0	2→3
4	9.0	0→4
5	13.0	4→5
6	25.0	2→6
7	8.0	0→7

Dijkstra's algorithm

- Consider vertices in increasing order of distance from s (non-tree vertex with the lowest `distTo[]` value).
- Add vertex to tree and relax all edges incident from that vertex.

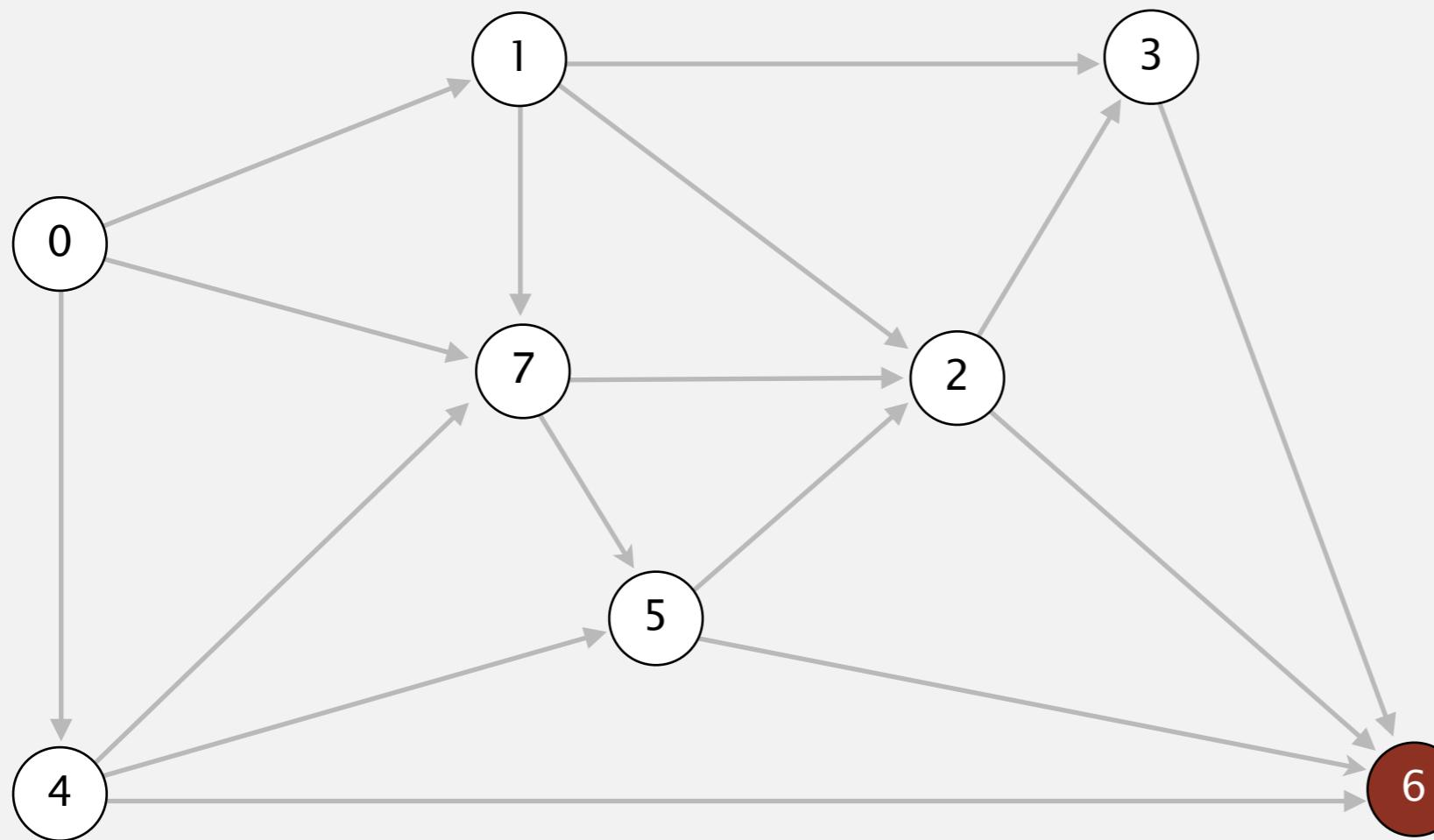


v	distTo []	edgeTo []
0	0.0	-
1	5.0	0→1
2	14.0	5→2
3	17.0	2→3
4	9.0	0→4
5	13.0	4→5
6	25.0	2→6
7	8.0	0→7

select vertex 6

Dijkstra's algorithm

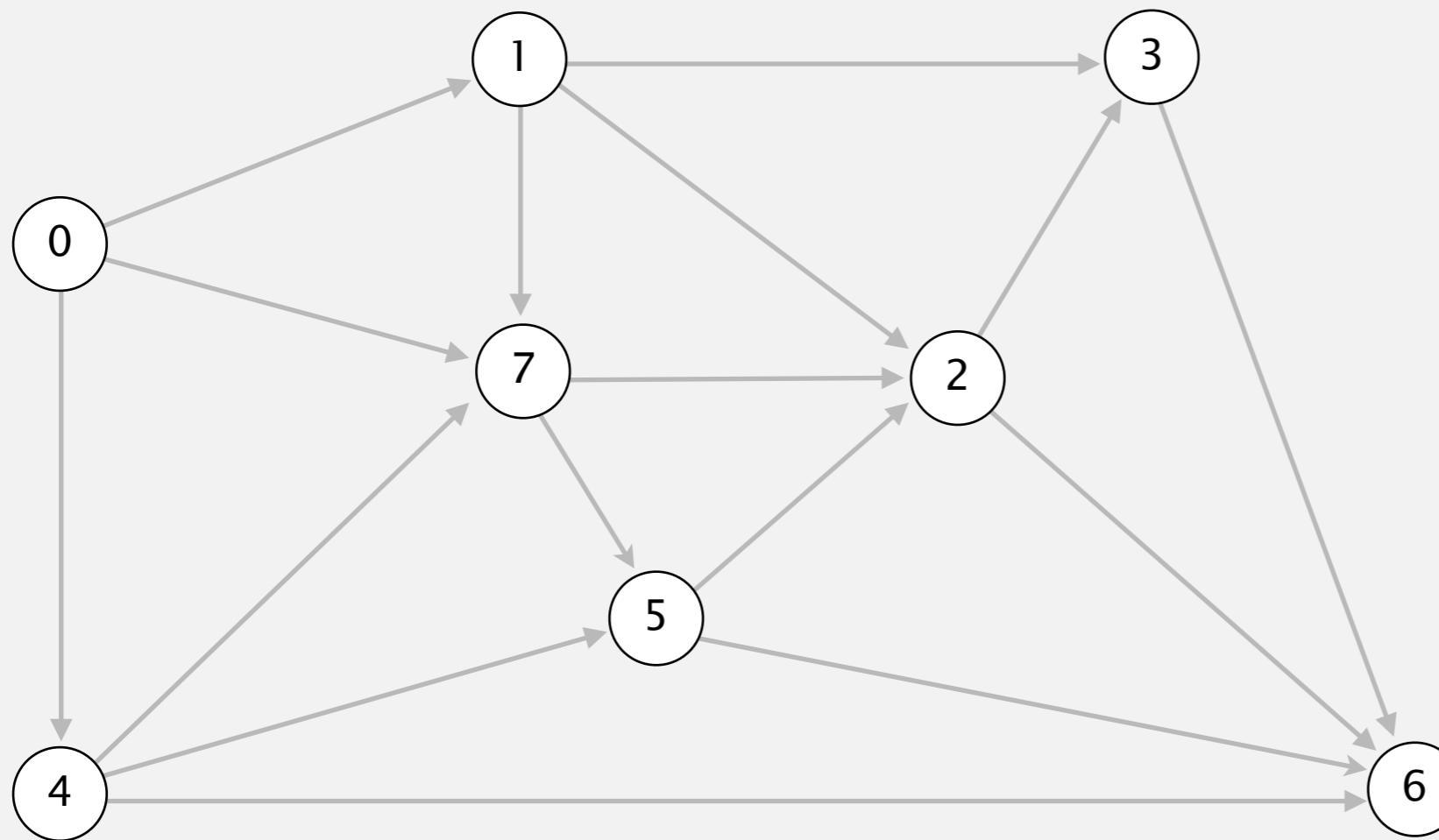
- Consider vertices in increasing order of distance from s (non-tree vertex with the lowest `distTo[]` value).
- Add vertex to tree and relax all edges incident from that vertex.



v	distTo []	edgeTo []
0	0.0	-
1	5.0	0→1
2	14.0	5→2
3	17.0	2→3
4	9.0	0→4
5	13.0	4→5
6	25.0	2→6
7	8.0	0→7

Dijkstra's algorithm

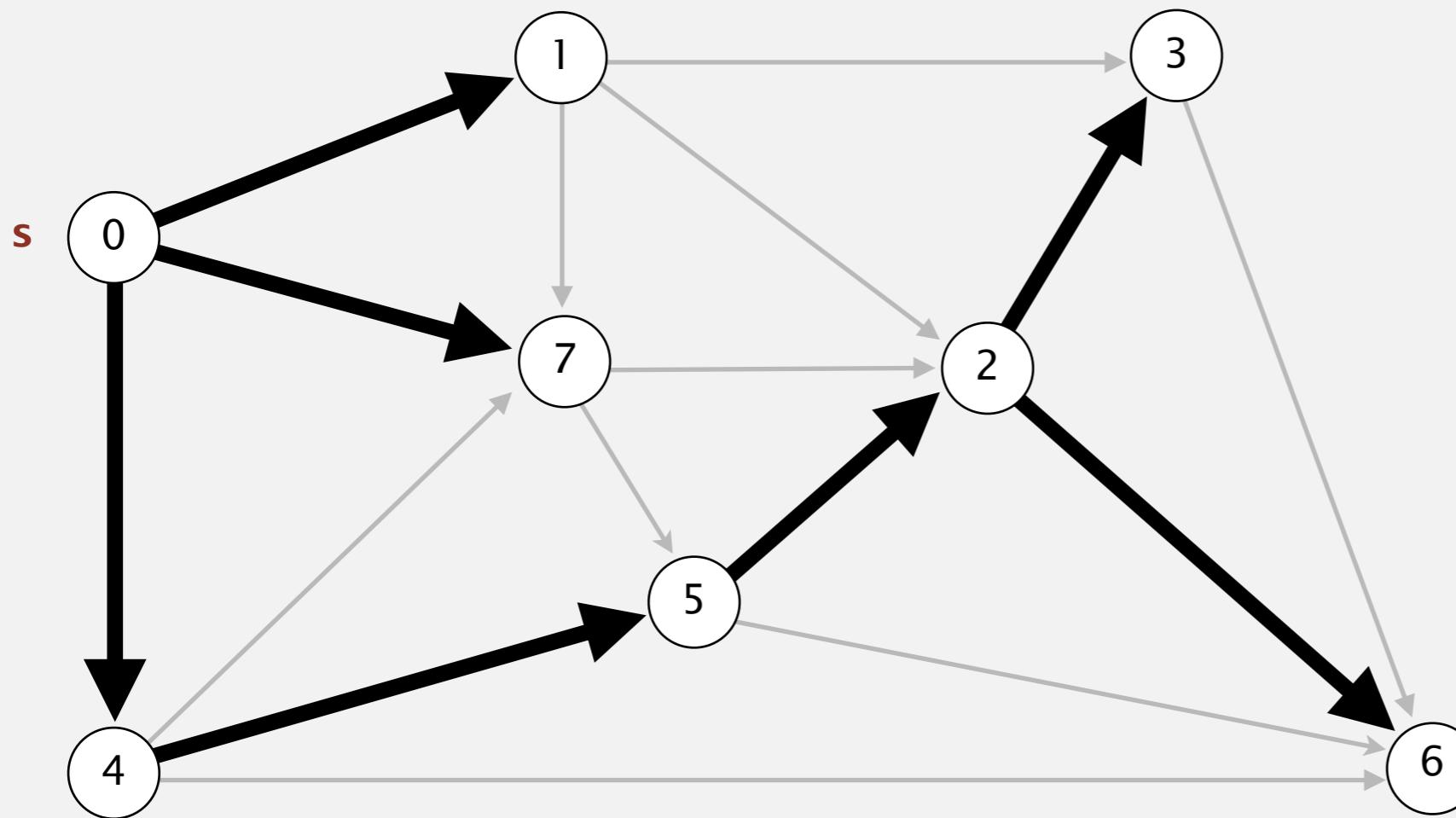
- Consider vertices in increasing order of distance from s (non-tree vertex with the lowest `distTo[]` value).
- Add vertex to tree and relax all edges incident from that vertex.



v	distTo []	edgeTo []
0	0.0	-
1	5.0	0→1
2	14.0	5→2
3	17.0	2→3
4	9.0	0→4
5	13.0	4→5
6	25.0	2→6
7	8.0	0→7

Dijkstra's algorithm

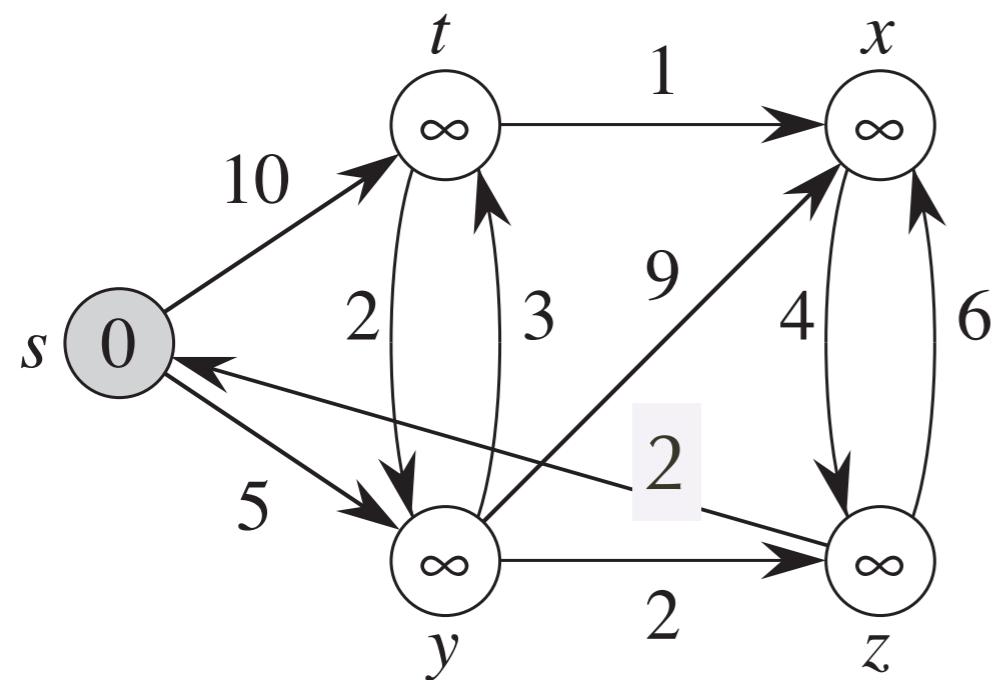
- Consider vertices in increasing order of distance from s (non-tree vertex with the lowest `distTo[]` value).
- Add vertex to tree and relax all edges incident from that vertex.



shortest-paths tree from vertex s

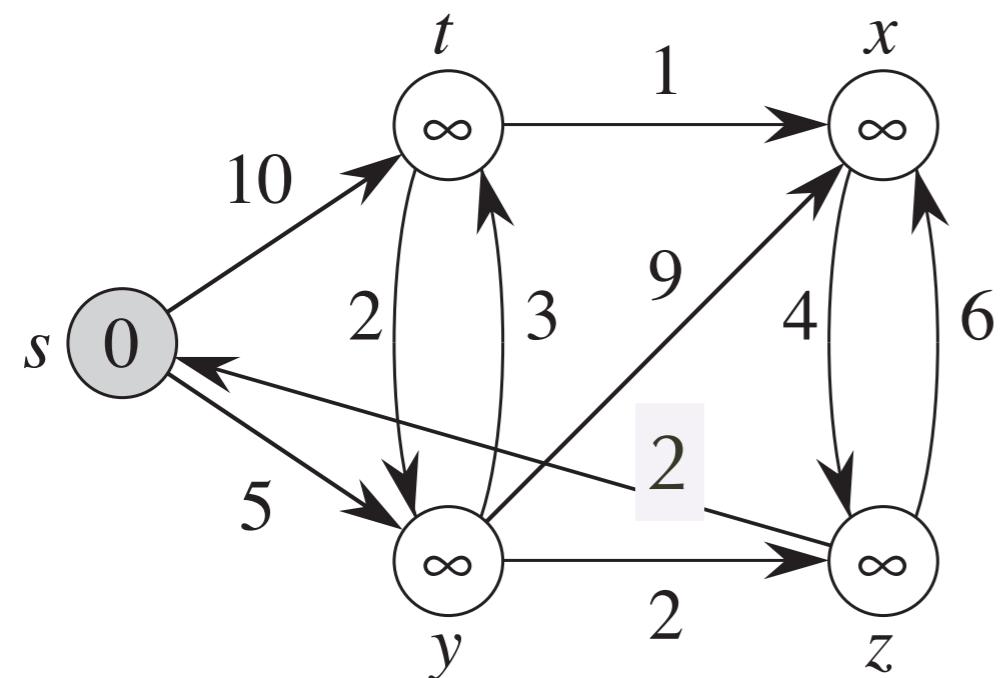
v	distTo []	edgeTo []
0	0.0	-
1	5.0	0→1
2	14.0	5→2
3	17.0	2→3
4	9.0	0→4
5	13.0	4→5
6	25.0	2→6
7	8.0	0→7

Floyd-Warshall algorithm



	s	t	x	y	z
s	0	10	∞	5	∞
t	∞	0	1	2	∞
x	∞	∞	0	∞	4
y	∞	3	9	0	2
z	2	∞	6	∞	0

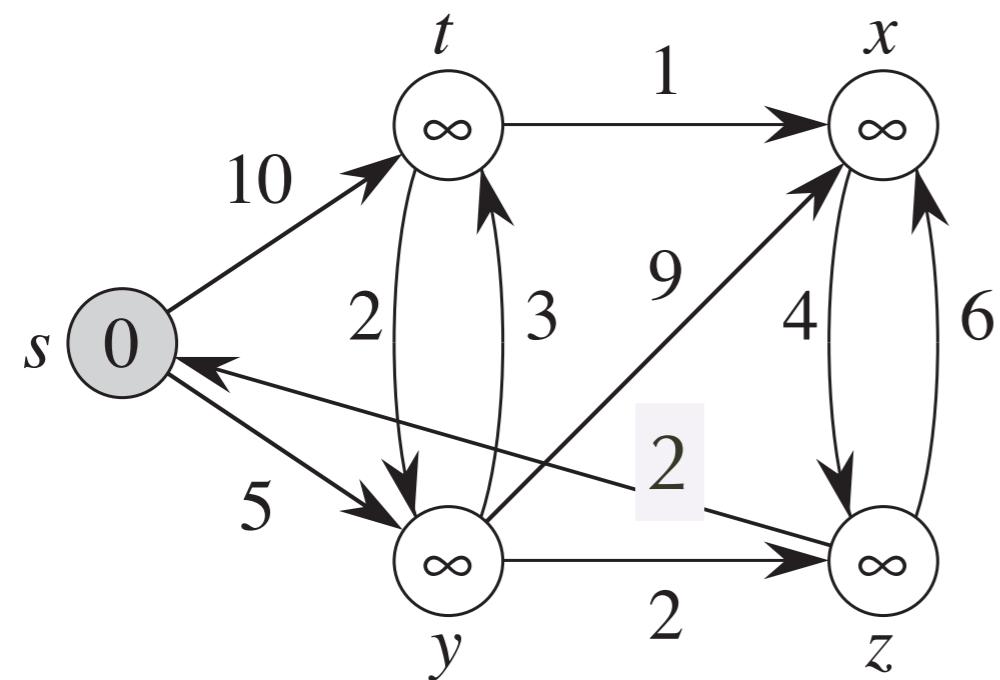
Floyd-Warshall algorithm



	s	t	x	y	z
s	0	10	∞	5	∞
t	∞	0	1	2	∞
x	∞	∞	0	∞	4
y	∞	3	9	0	2
z	2	∞	6	∞	0

The shortest path from u to v that passes none vertex

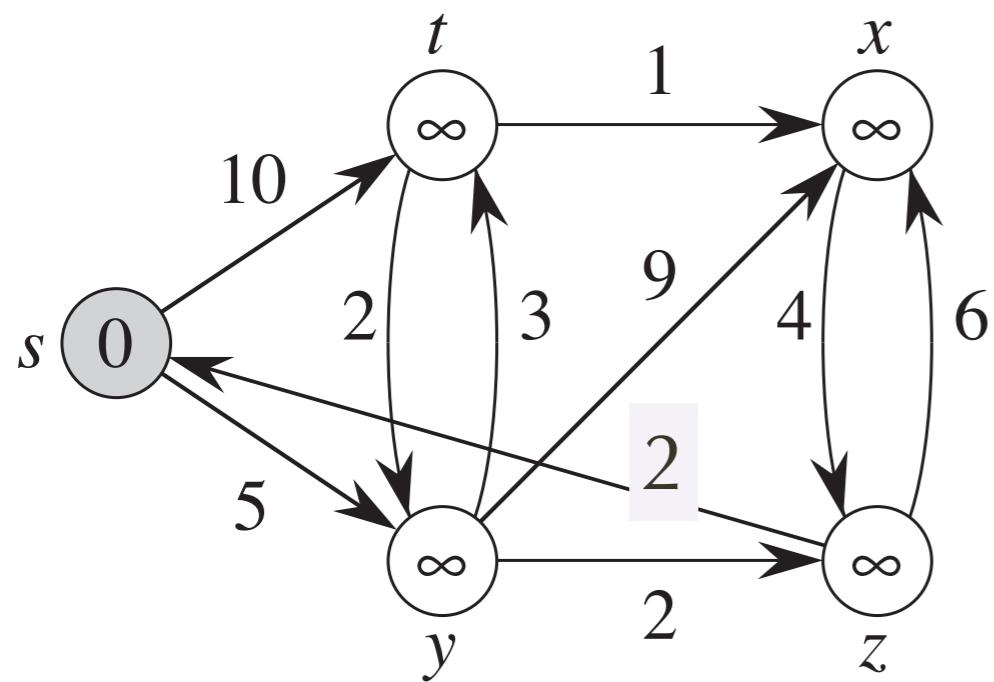
Floyd-Warshall algorithm



	s	t	x	y	z
s	0	10	∞	5	∞
t	∞	0	1	2	∞
x	∞	∞	0	∞	4
y	∞	3	9	0	2
z	∞	6	∞	∞	0

The shortest path from u to v that passes none vertex

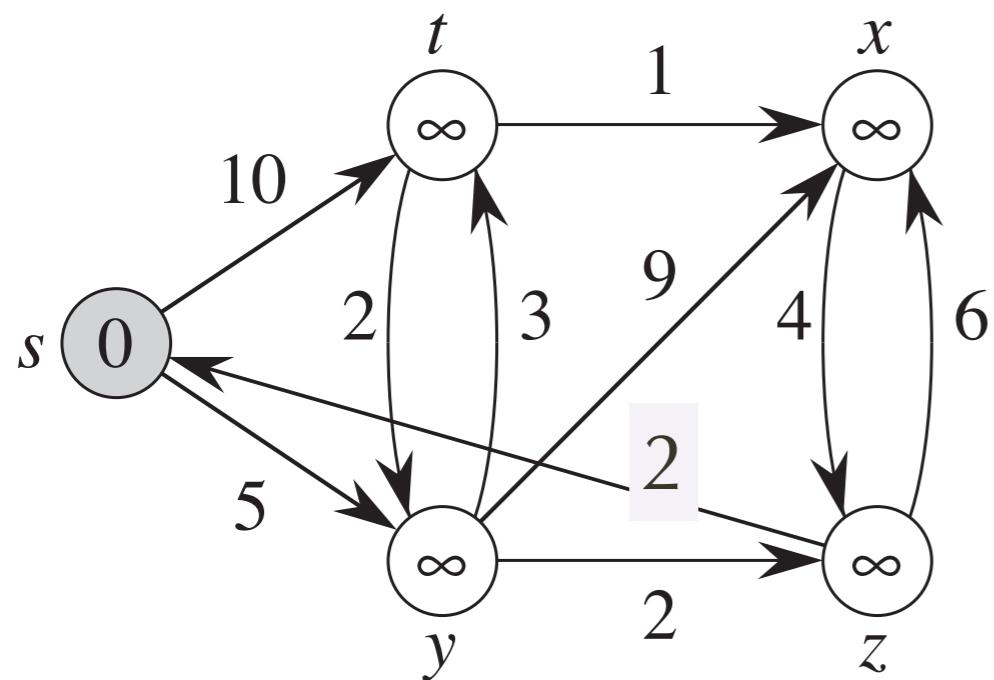
Floyd-Warshall algorithm



	<i>s</i>	<i>t</i>	<i>x</i>	<i>y</i>	<i>z</i>
<i>s</i>	0	10	∞	5	∞
<i>t</i>	∞	0	1	2	∞
<i>x</i>	∞	∞	0	∞	4
<i>y</i>	∞	3	9	0	2
<i>z</i>	2	∞	6	∞	0

The shortest path from u to v that passes none vertex

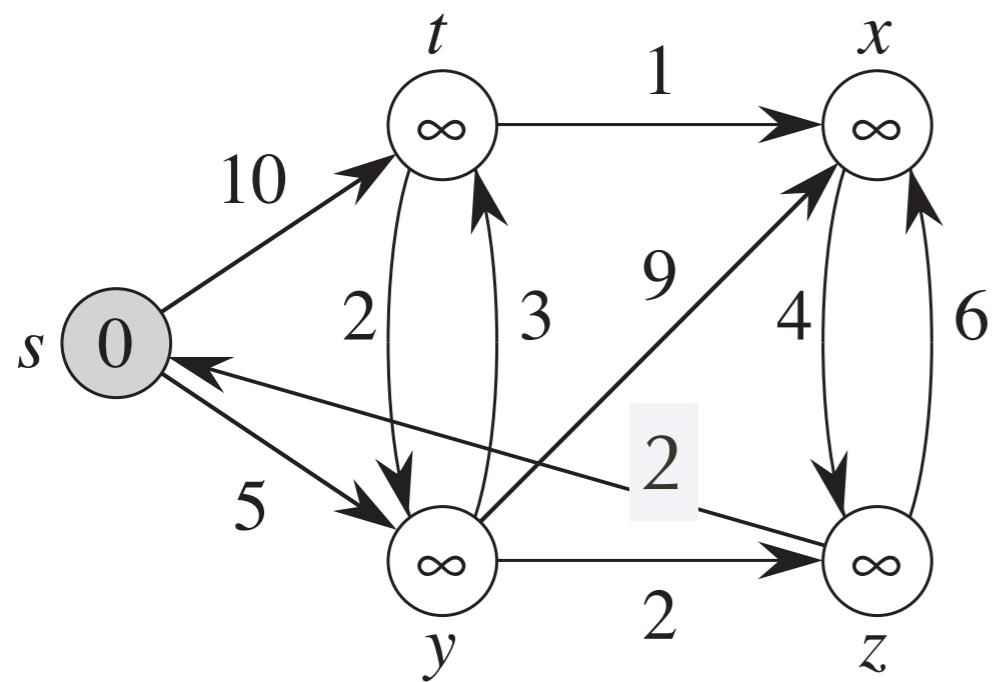
Floyd-Warshall algorithm



	<i>s</i>	<i>t</i>	<i>x</i>	<i>y</i>	<i>z</i>
<i>s</i>	0	10	∞	5	∞
<i>t</i>	∞	0	1	2	∞
<i>x</i>	∞	∞	0	∞	4
<i>y</i>	∞	∞	9	0	2
<i>z</i>	2	∞	6	∞	0

The shortest path from u to v that passes none vertex

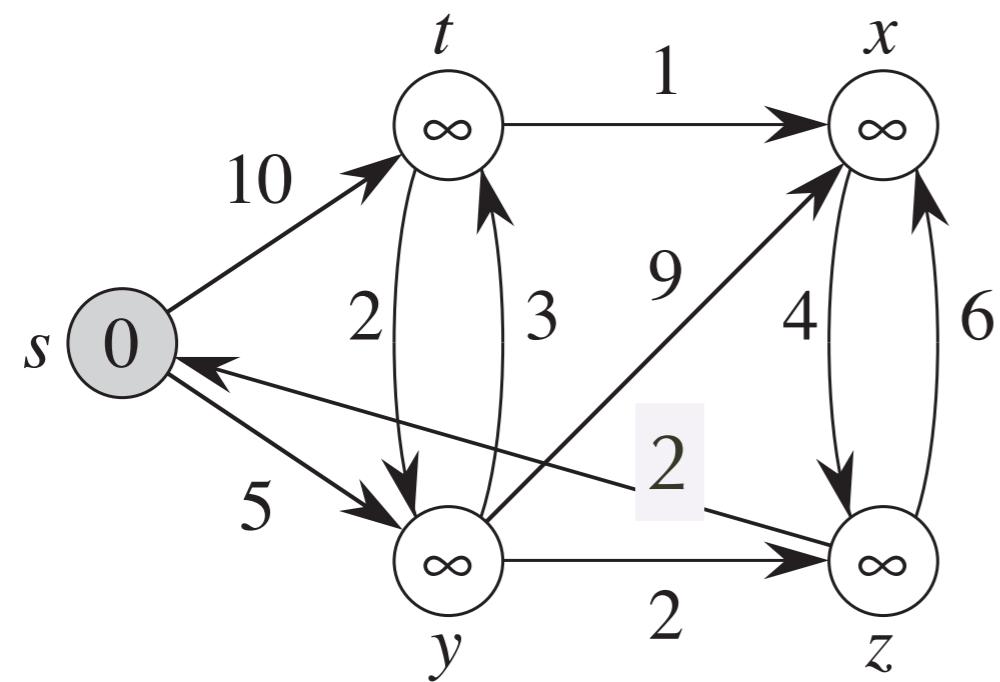
Floyd-Warshall algorithm



	<i>s</i>	<i>t</i>	<i>x</i>	<i>y</i>	<i>z</i>
<i>s</i>	0	10	∞	5	∞
<i>t</i>	∞	0	1	2	∞
<i>x</i>	∞	∞	0	∞	4
<i>y</i>	∞	3	9	0	2
<i>z</i>	2	12	6	∞	0

The shortest path from u to v that passes none vertex

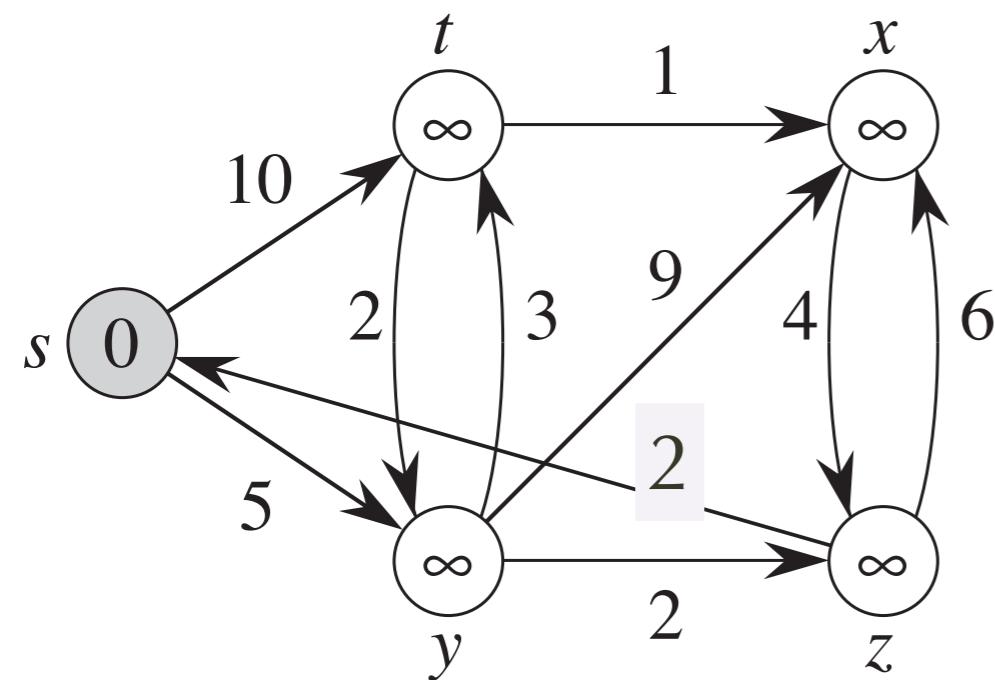
Floyd-Warshall algorithm



	<i>s</i>	<i>t</i>	<i>x</i>	<i>y</i>	<i>z</i>
<i>s</i>	0	10	∞	5	∞
<i>t</i>	∞	0	1	2	∞
<i>x</i>	∞	∞	0	∞	4
<i>y</i>	∞	3	9	0	2
<i>z</i>	2	12	6	∞	0

The shortest path from u to v that passes none vertex

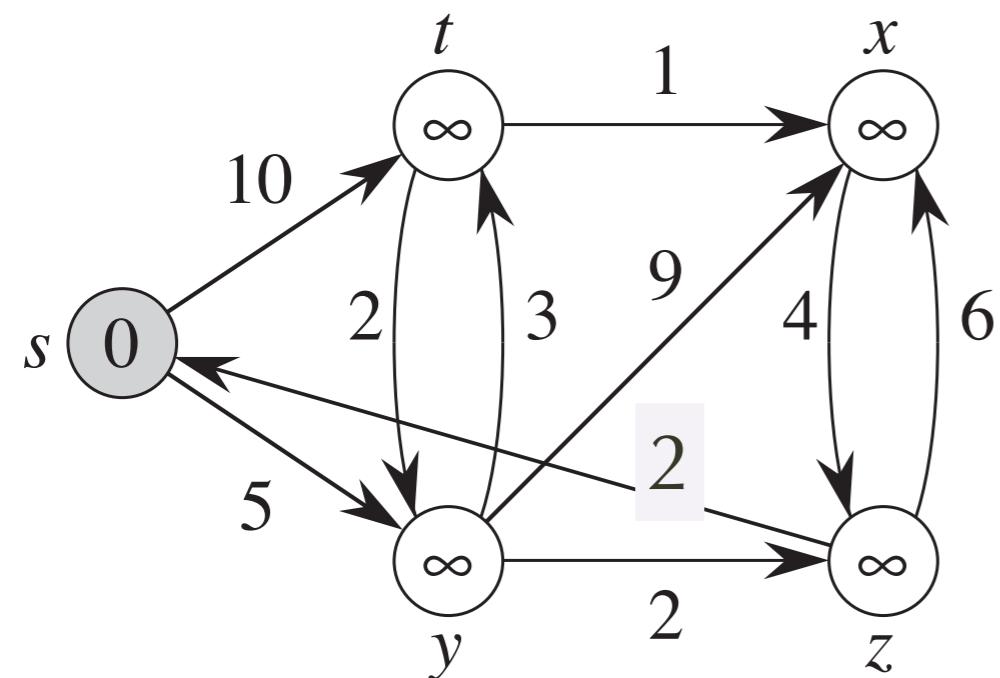
Floyd-Warshall algorithm



	<i>s</i>	<i>t</i>	<i>x</i>	<i>y</i>	<i>z</i>
<i>s</i>	0	10	∞	5	∞
<i>t</i>	∞	0	1	2	∞
<i>x</i>	∞	∞	0	∞	4
<i>y</i>	∞	3	9	0	2
<i>z</i>	2	12	6	7	0

The shortest path from u to v that passes none vertex

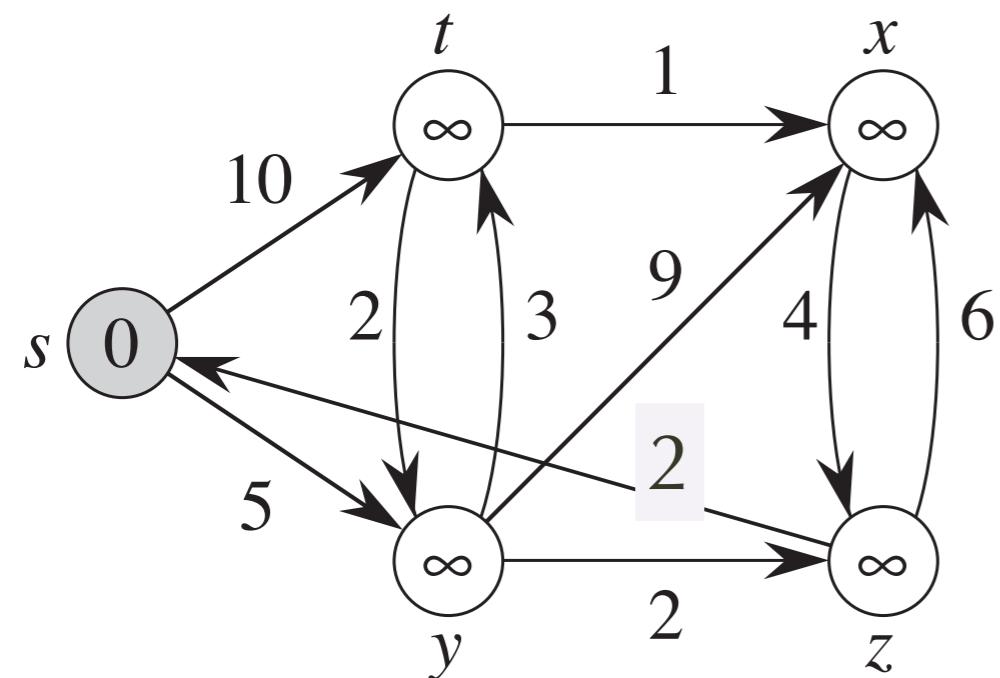
Floyd-Warshall algorithm



	s	t	x	y	z
s	0	10	∞	5	∞
t	∞	0	1	2	∞
x	∞	∞	0	∞	4
y	∞	3	9	0	2
z	2	12	6	7	0

The shortest path from u to v that passes none vertex

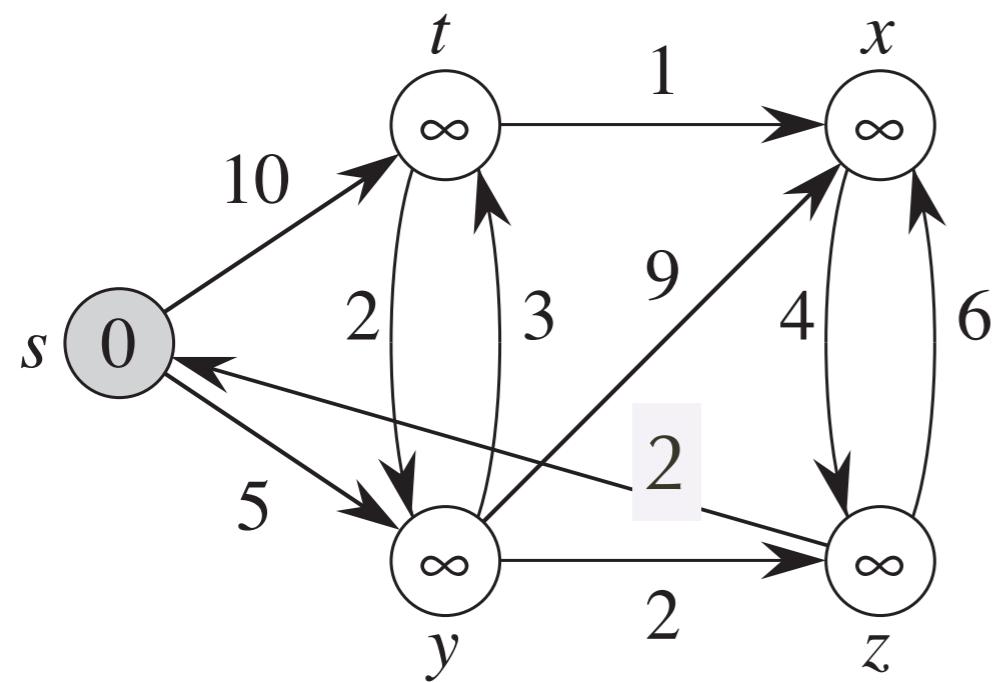
Floyd-Warshall algorithm



	<i>s</i>	<i>t</i>	<i>x</i>	<i>y</i>	<i>z</i>
<i>s</i>	0	10	∞	5	∞
<i>t</i>	∞	0	1	2	∞
<i>x</i>	∞	∞	0	∞	4
<i>y</i>	∞	3	9	0	2
<i>z</i>	2	12	6	7	0

The shortest path from *u* to *v* that may passes *s*

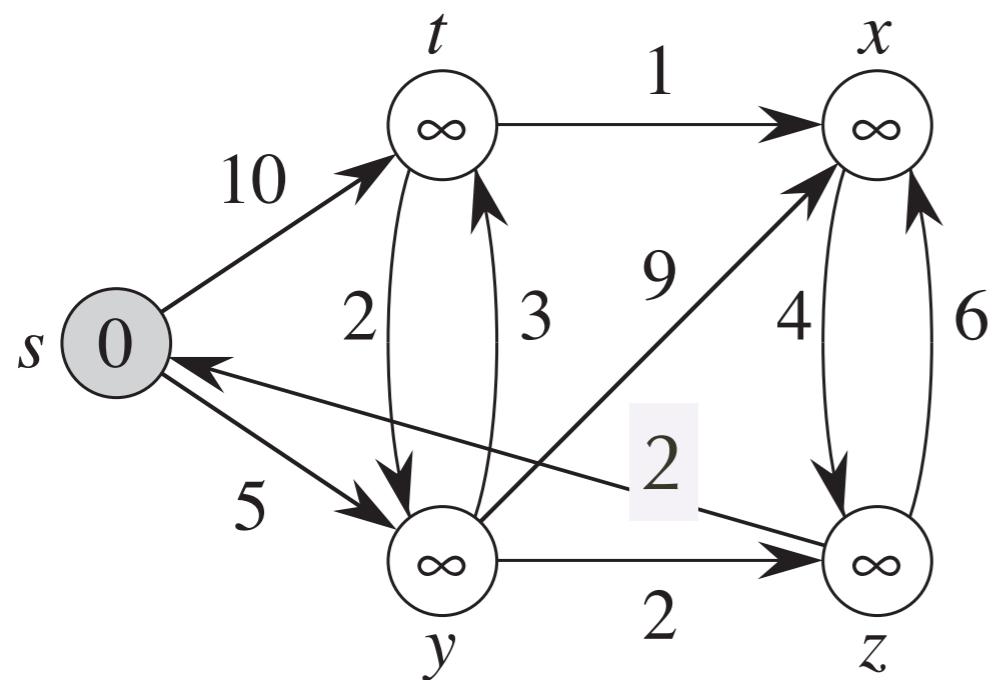
Floyd-Warshall algorithm



	<i>s</i>	<i>t</i>	<i>x</i>	<i>y</i>	<i>z</i>
<i>s</i>	0	10	∞	5	∞
<i>t</i>	∞	0	1	2	∞
<i>x</i>	∞	∞	0	∞	4
<i>y</i>	∞	3	9	0	2
<i>z</i>	2	12	6	7	0

The shortest path from u to v that may passes s

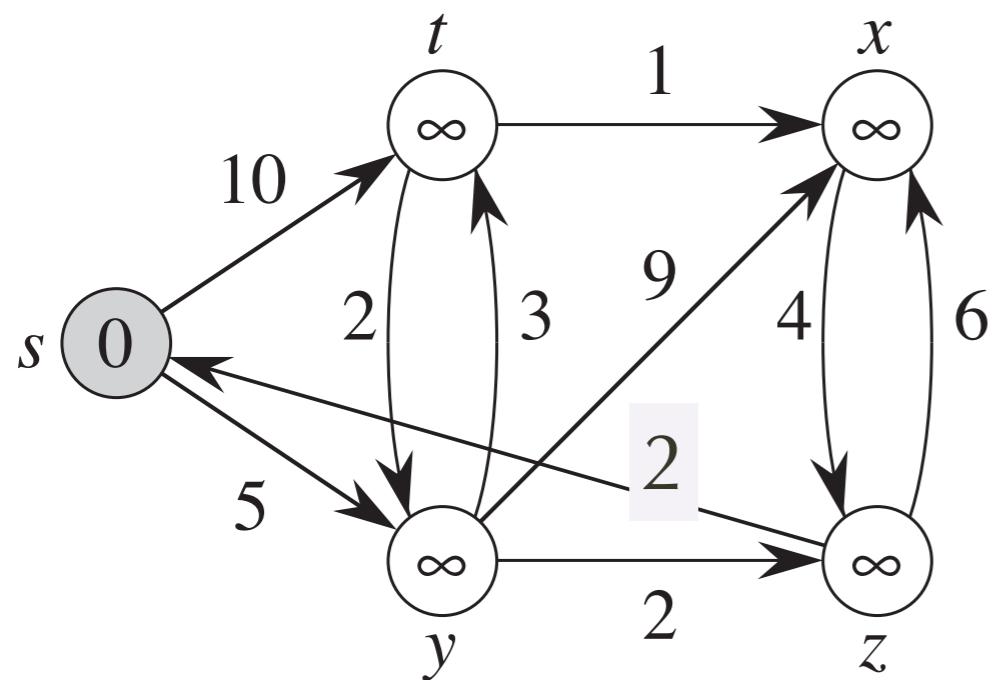
Floyd-Warshall algorithm



	<i>s</i>	<i>t</i>	<i>x</i>	<i>y</i>	<i>z</i>
<i>s</i>	0	10	11	5	∞
<i>t</i>	∞	0	1	2	∞
<i>x</i>	∞	∞	0	∞	4
<i>y</i>	∞	3	9	0	2
<i>z</i>	2	12	6	7	0

The shortest path from u to v that may passes s

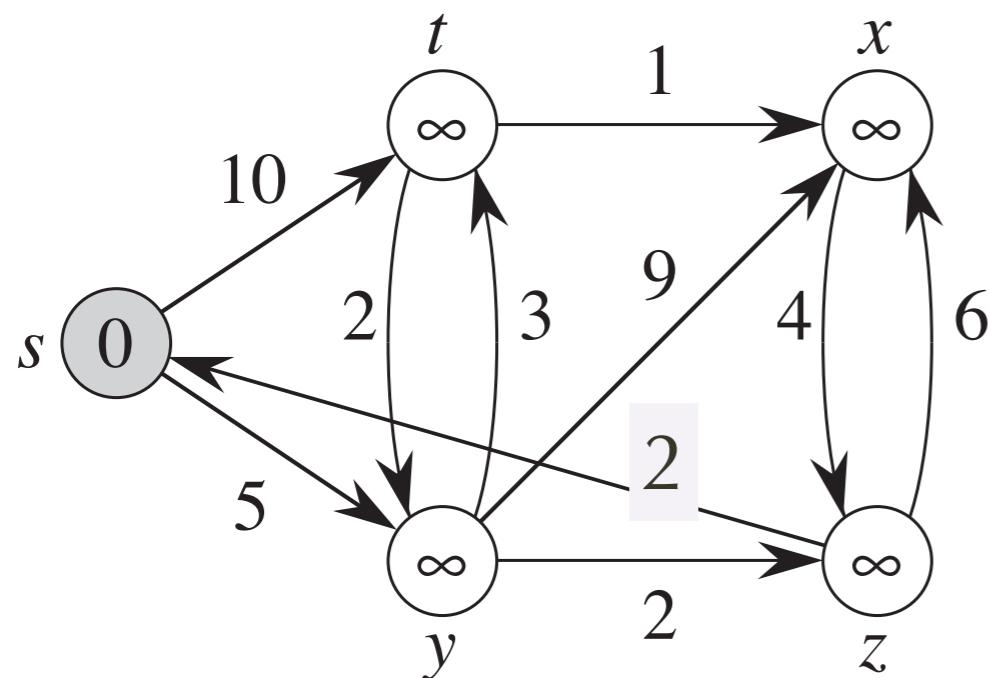
Floyd-Warshall algorithm



	<i>s</i>	<i>t</i>	<i>x</i>	<i>y</i>	<i>z</i>
<i>s</i>	0	10	11	5	∞
<i>t</i>	∞	0	1	2	∞
<i>x</i>	∞	∞	0	∞	4
<i>y</i>	∞	3	4	0	2
<i>z</i>	2	12	6	7	0

The shortest path from u to v that may passes s

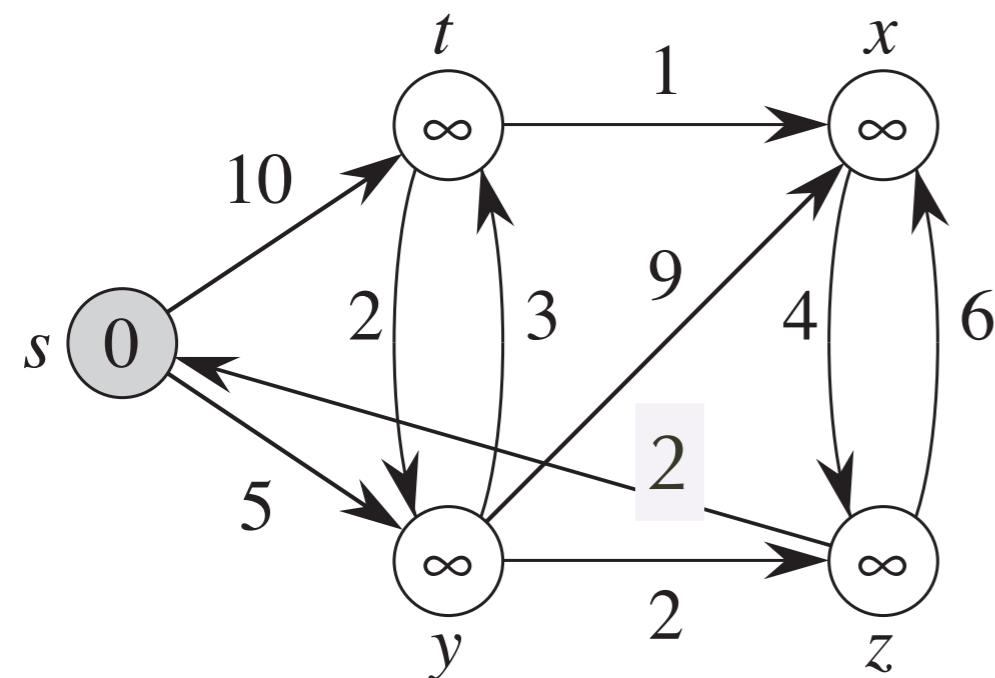
Floyd-Warshall algorithm



	s	t	x	y	z
s	0	10	11	5	8
t	∞	0	1	2	8
x	∞	∞	0	∞	4
y	∞	3	4	0	2
z	2	12	6	7	0

The shortest path from u to v that may passes s

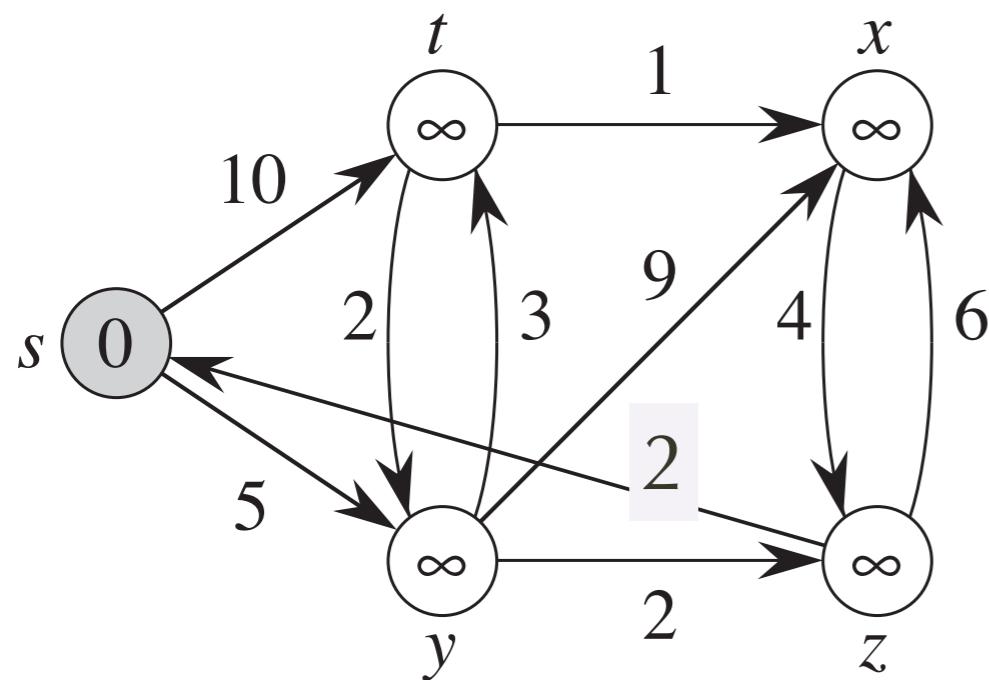
Floyd-Warshall algorithm



s	t	x	y	z	
s	0	10	11	5	8
t	∞	0	1	2	8
x	∞	∞	0	∞	4
y	∞	3	4	0	2
z	2	12	6	7	0

The shortest path from u to v that may passes s, t

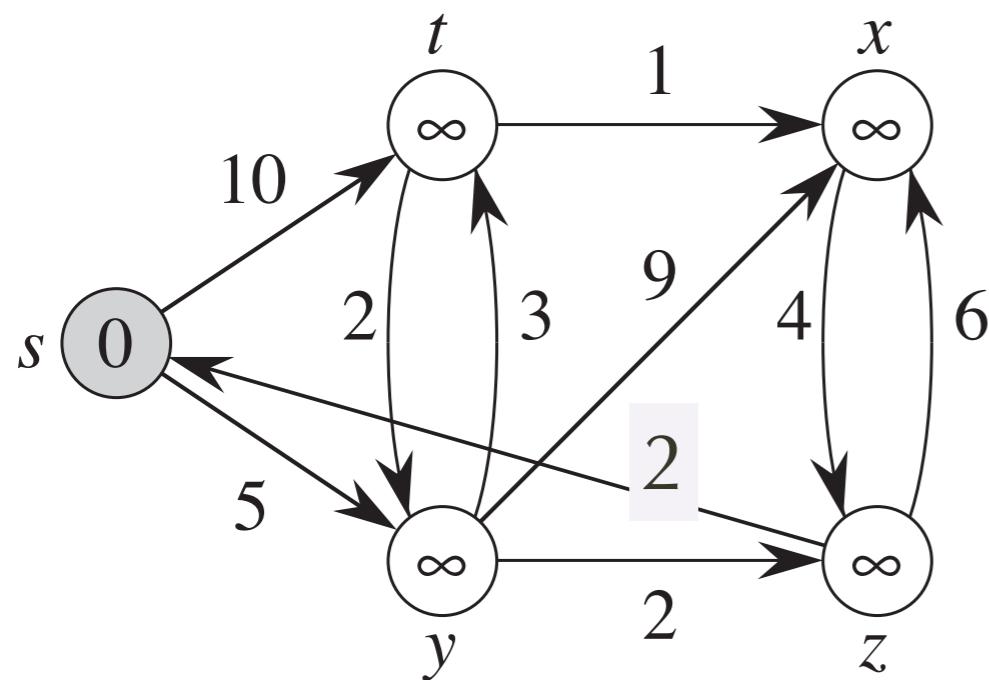
Floyd-Warshall algorithm



	s	t	x	y	z
s	0	10	11	5	∞
t	∞	0	1	2	∞
x	∞	∞	0	∞	4
y	∞	3	4	0	2
z	2	12	6	7	0

The shortest path from u to v that may passes s, t

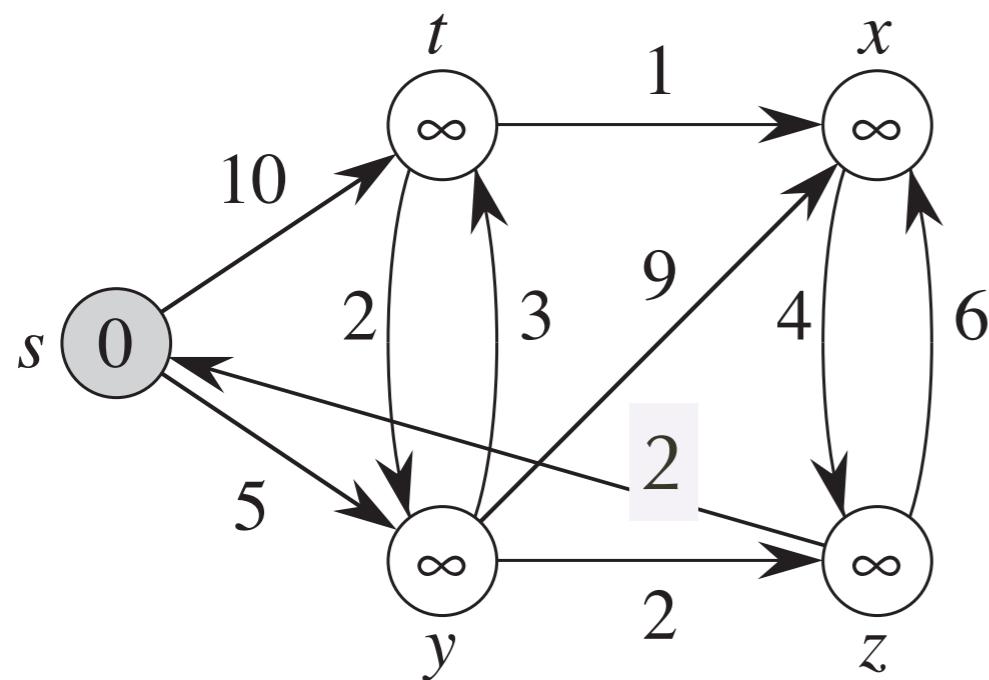
Floyd-Warshall algorithm



	<i>s</i>	<i>t</i>	<i>x</i>	<i>y</i>	<i>z</i>
<i>s</i>	0	10	11	5	15
<i>t</i>	∞	0	1	2	∞
<i>x</i>	∞	∞	0	∞	4
<i>y</i>	∞	3	4	0	2
<i>z</i>	2	12	6	7	0

The shortest path from u to v that may passes s, t

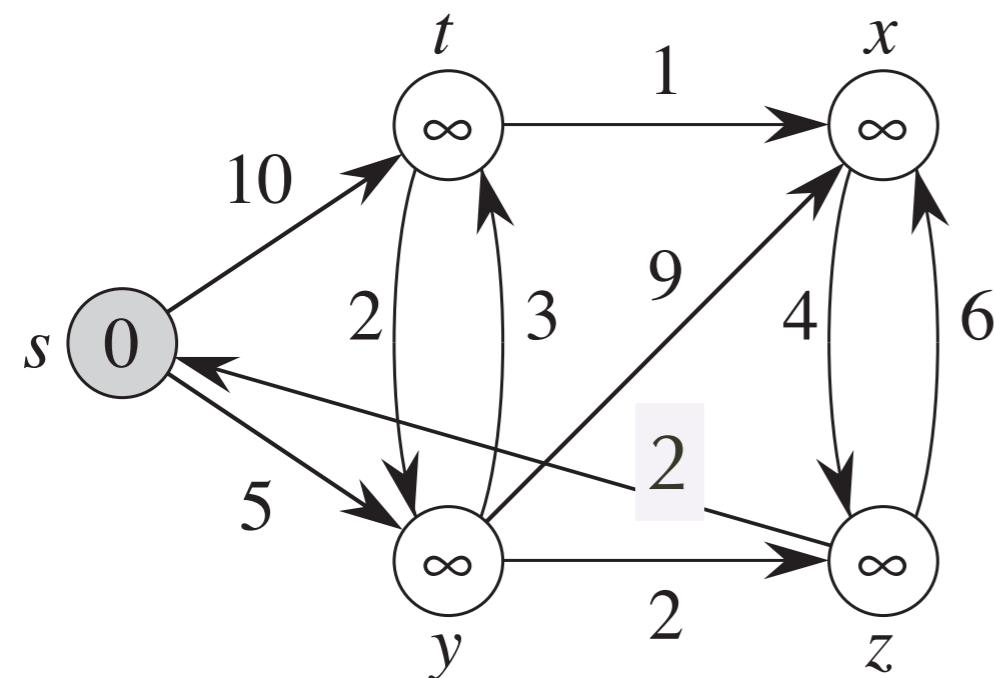
Floyd-Warshall algorithm



	s	t	x	y	z
s	0	10	11	5	15
t	∞	0	1	2	5
x	∞	∞	0	∞	4
y	∞	3	4	0	2
z	2	12	6	7	0

The shortest path from u to v that may passes s, t

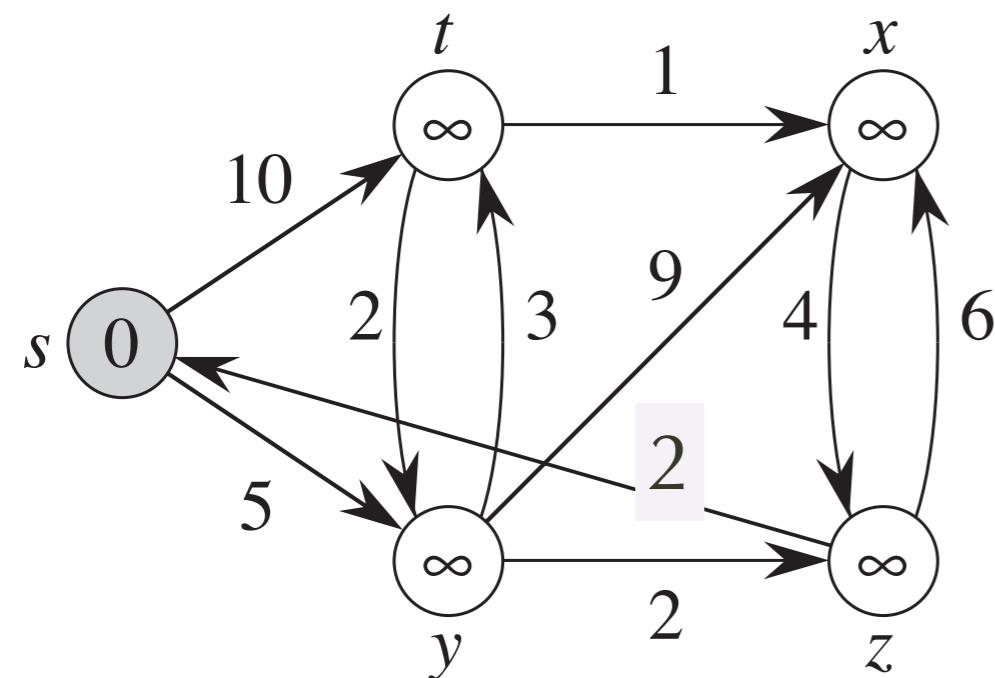
Floyd-Warshall algorithm



s	t	x	y	z	
s	0	10	11	5	15
t	∞	0	1	2	5
x	∞	∞	0	∞	4
y	∞	3	4	0	2
z	2	12	6	7	0

The shortest path from u to v that may passes s, t

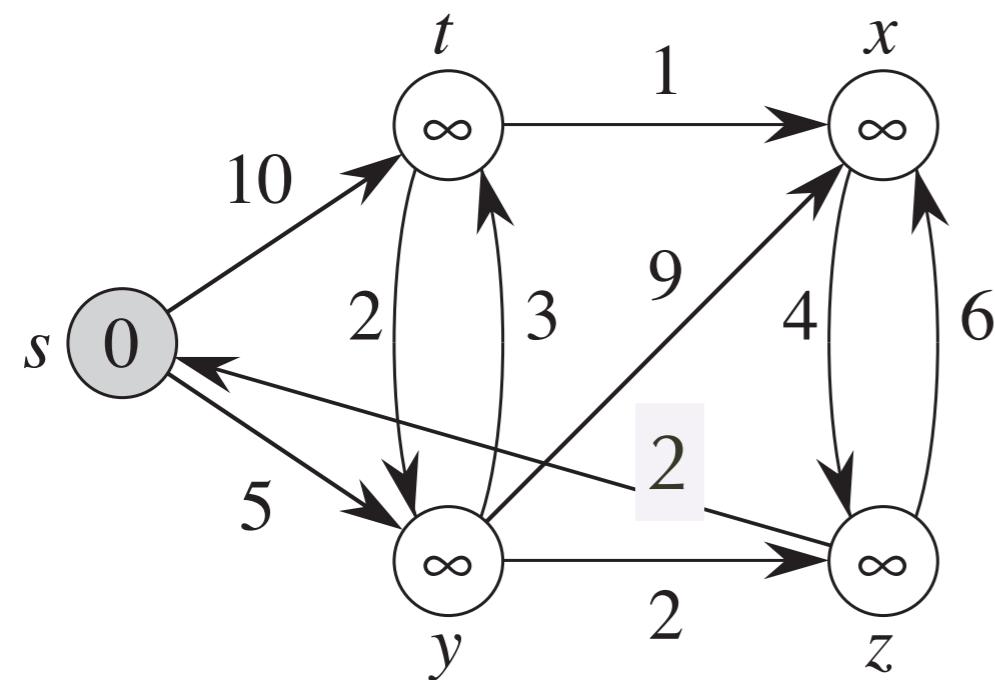
Floyd-Warshall algorithm



	s	t	x	y	z
s	0	10	11	5	15
t	∞	0	1	2	5
x	∞	∞	0	∞	4
y	∞	3	4	0	2
z	2	12	6	7	0

The shortest path from u to v that may passes s, t, x

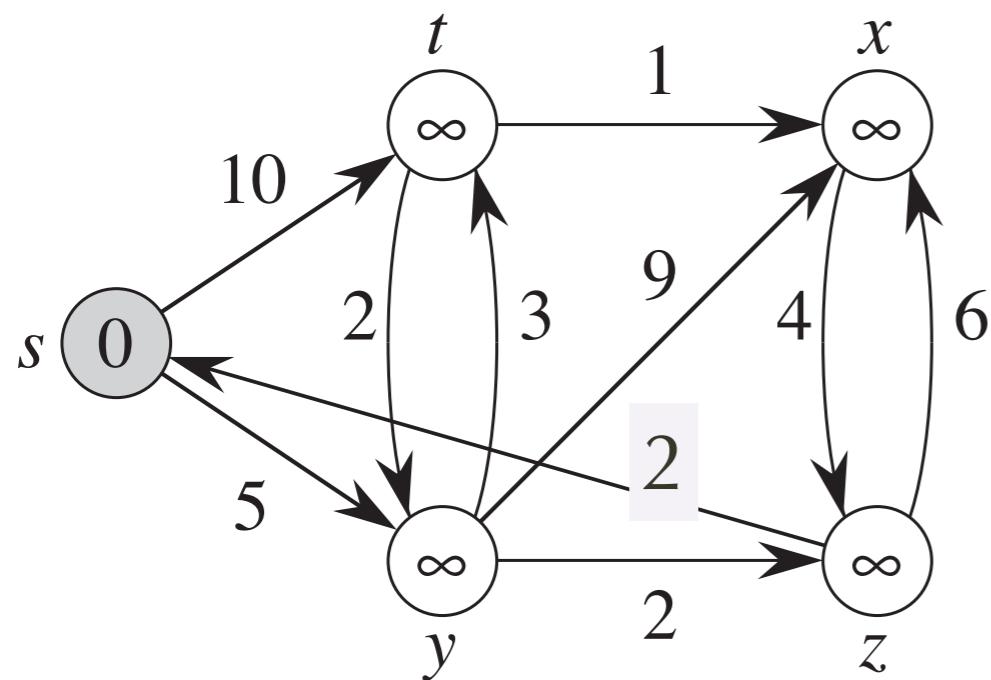
Floyd-Warshall algorithm



<i>s</i>	<i>t</i>	<i>x</i>	<i>y</i>	<i>z</i>
<i>s</i>	0	10	11	15
<i>t</i>	∞	0	1	5
<i>x</i>	∞	∞	0	4
<i>y</i>	∞	3	4	2
<i>z</i>	2	12	6	0

The shortest path from u to v that may passes s, t, x

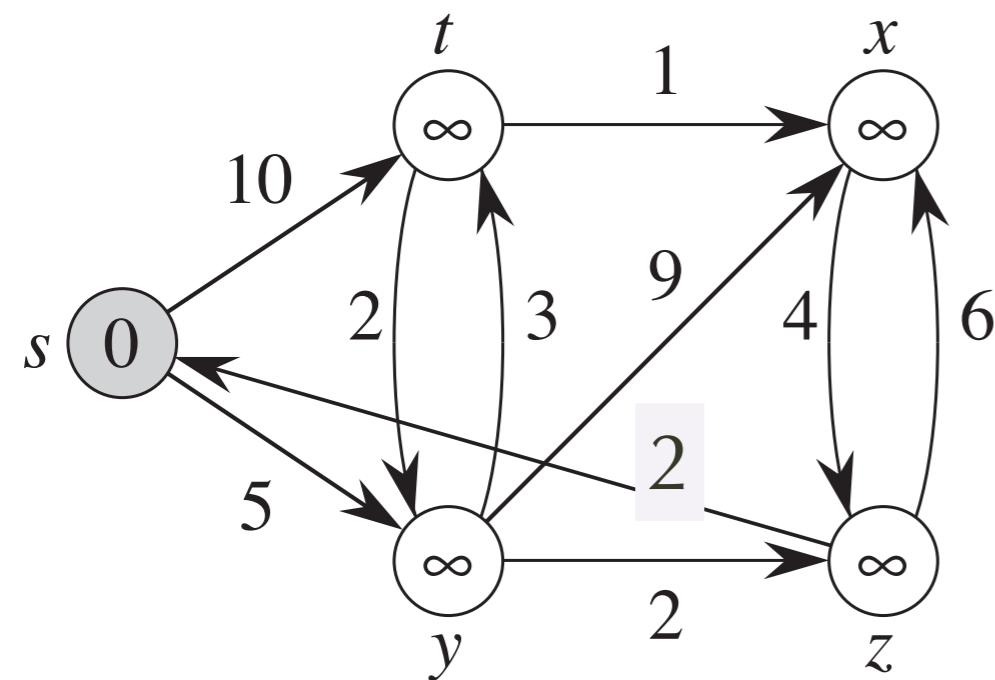
Floyd-Warshall algorithm



	s	t	x	y	z
s	0	10	11	5	7
t	∞	0	1	2	5
x	∞	∞	0	∞	4
y	∞	3	4	0	2
z	2	12	6	7	0

The shortest path from u to v that may passes s, t, x

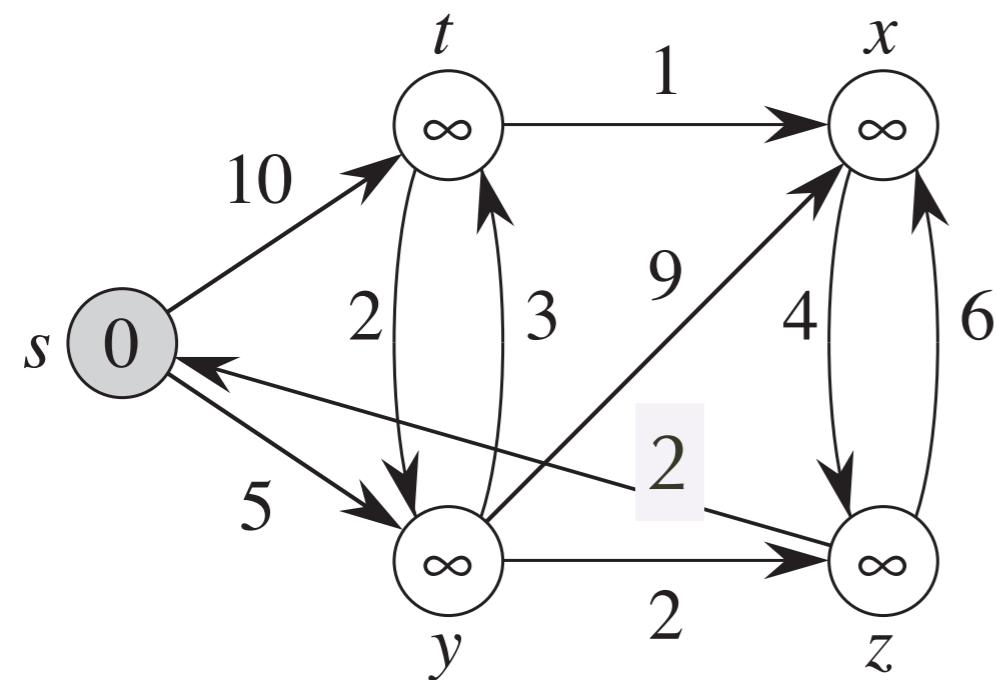
Floyd-Warshall algorithm



s	t	x	y	z
s	0	10	11	7
t	∞	0	1	4
x	∞	∞	0	4
y	∞	3	4	2
z	2	12	6	0

The shortest path from u to v that may passes s, t, x

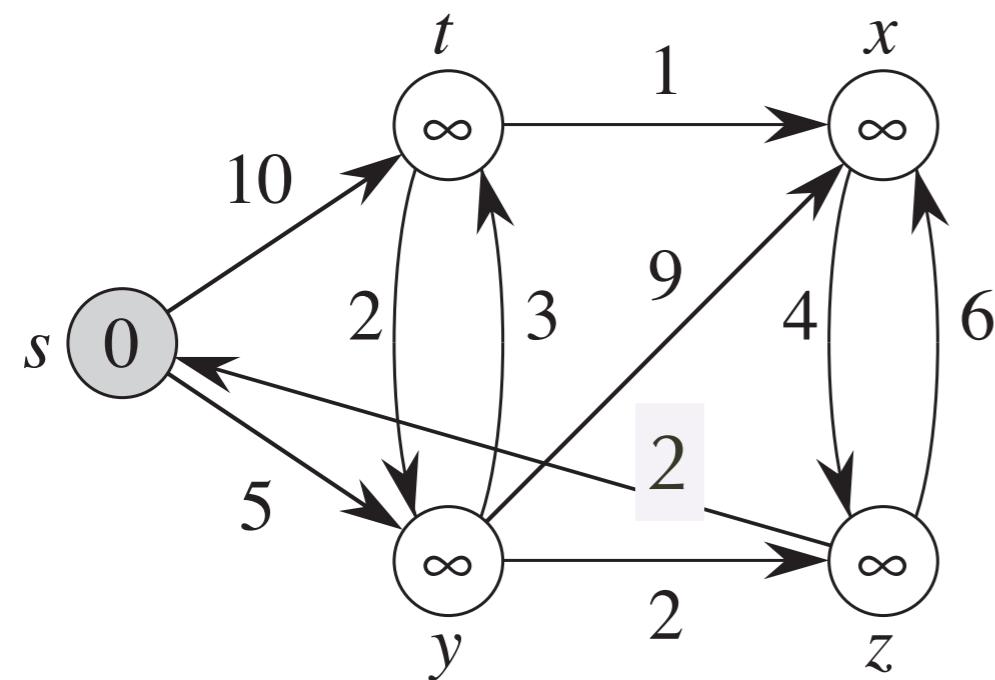
Floyd-Warshall algorithm



	s	t	x	y	z
s	0	10	11	5	7
t	∞	0	1	2	4
x	∞	∞	0	∞	4
y	∞	3	4	0	2
z	2	10	6	7	0

The shortest path from u to v that may passes s, t, x

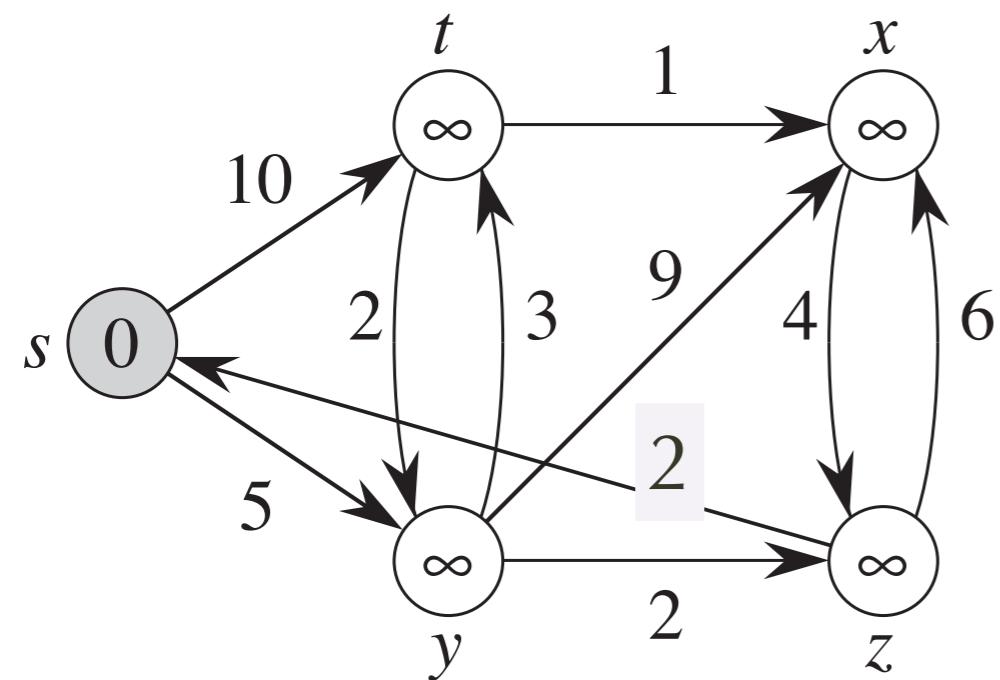
Floyd-Warshall algorithm



<i>s</i>	<i>t</i>	<i>x</i>	<i>y</i>	<i>z</i>
<i>s</i>	0	8	11	7
<i>t</i>	∞	0	1	4
<i>x</i>	∞	∞	0	4
<i>y</i>	∞	3	4	2
<i>z</i>	2	10	6	0

The shortest path from u to v that may passes s, t, x

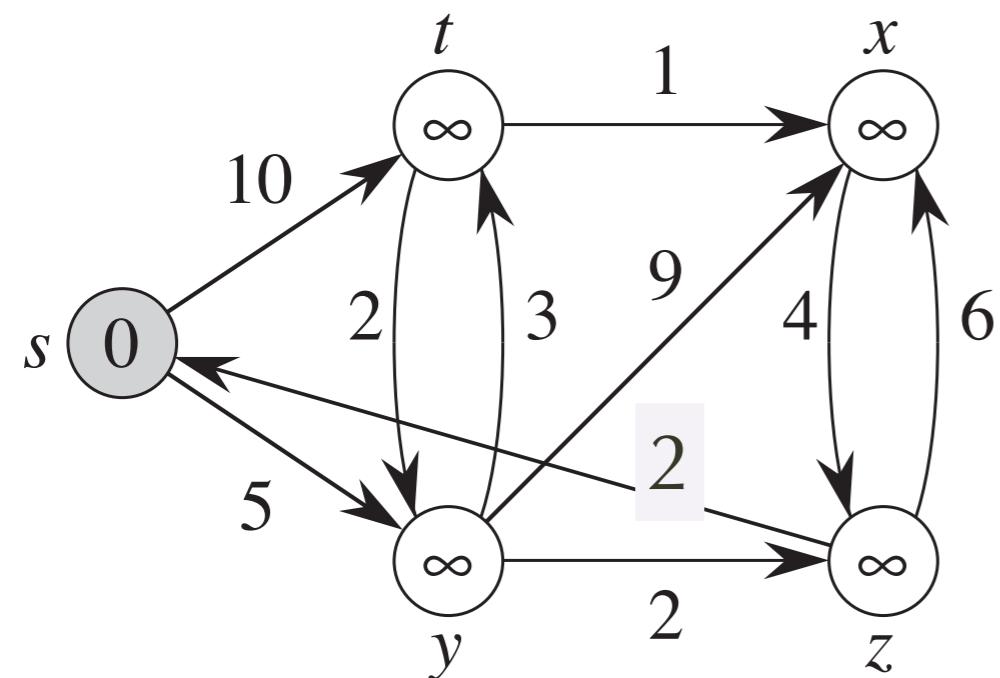
Floyd-Warshall algorithm



<i>s</i>	<i>t</i>	<i>x</i>	<i>y</i>	<i>z</i>
<i>s</i>	0	8	9	7
<i>t</i>	∞	0	1	4
<i>x</i>	∞	∞	0	4
<i>y</i>	∞	3	4	2
<i>z</i>	2	10	6	0

The shortest path from u to v that may passes s, t, x

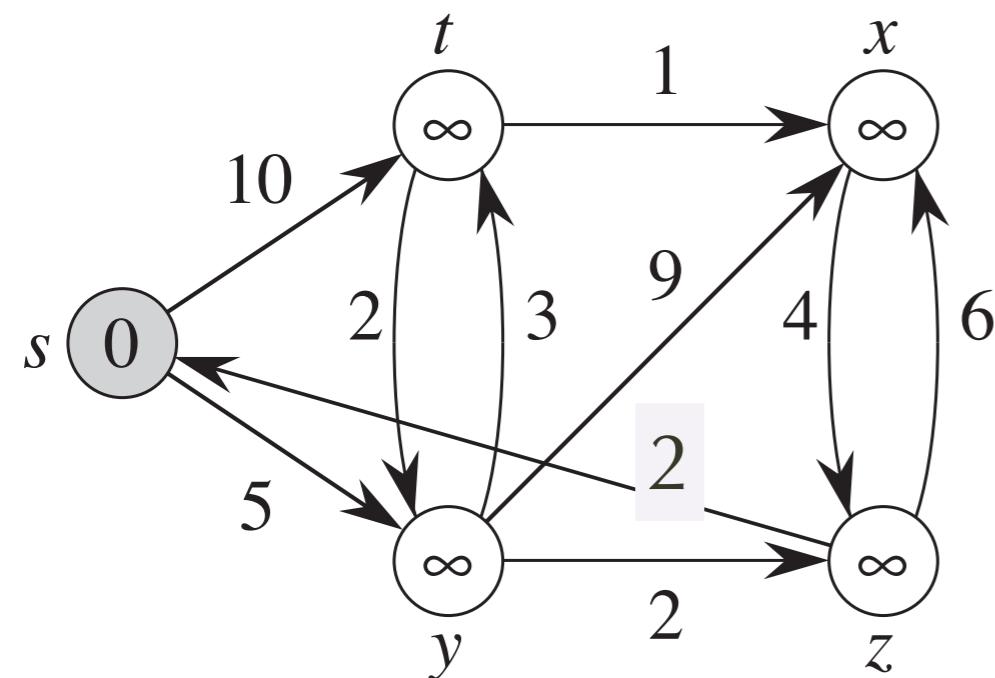
Floyd-Warshall algorithm



s	t	x	y	z	
s	0	8	9	5	7
t	∞	0	1	2	4
x	∞	∞	0	∞	4
y	∞	3	4	0	2
z	2	10	6	7	0

The shortest path from u to v that may passes s, t, x

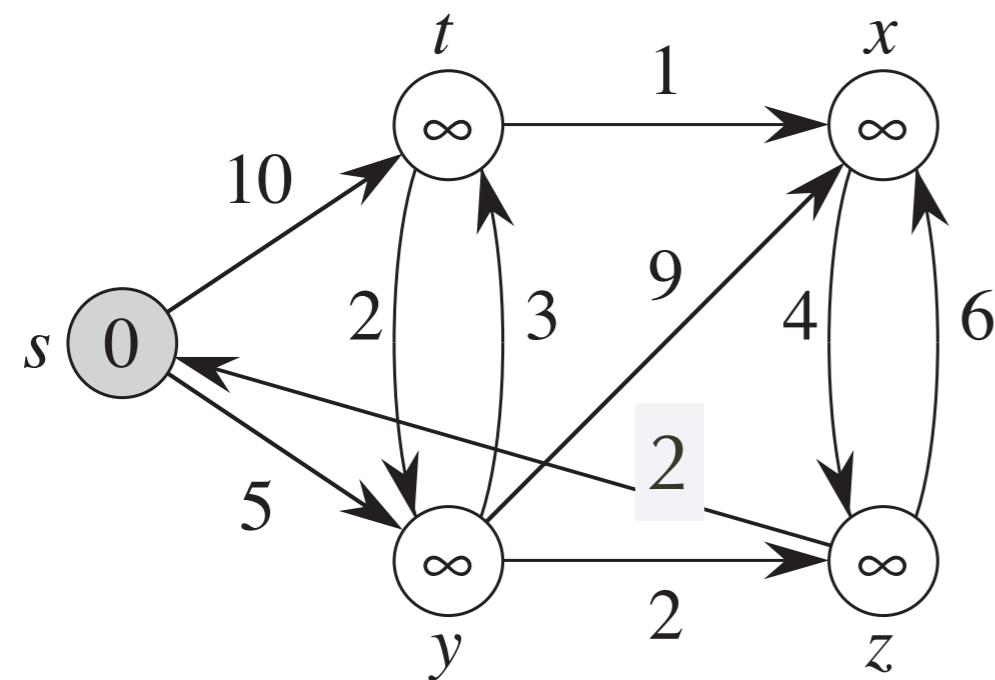
Floyd-Warshall algorithm



<i>s</i>	<i>t</i>	<i>x</i>	<i>y</i>	<i>z</i>	
<i>s</i>	0	8	9	5	7
<i>t</i>	∞	0	1	2	4
<i>x</i>	∞	∞	0	∞	4
<i>y</i>	∞	3	4	0	2
<i>z</i>	2	10	6	7	0

The shortest path from *u* to *v* that may passes *s, t, x, y*

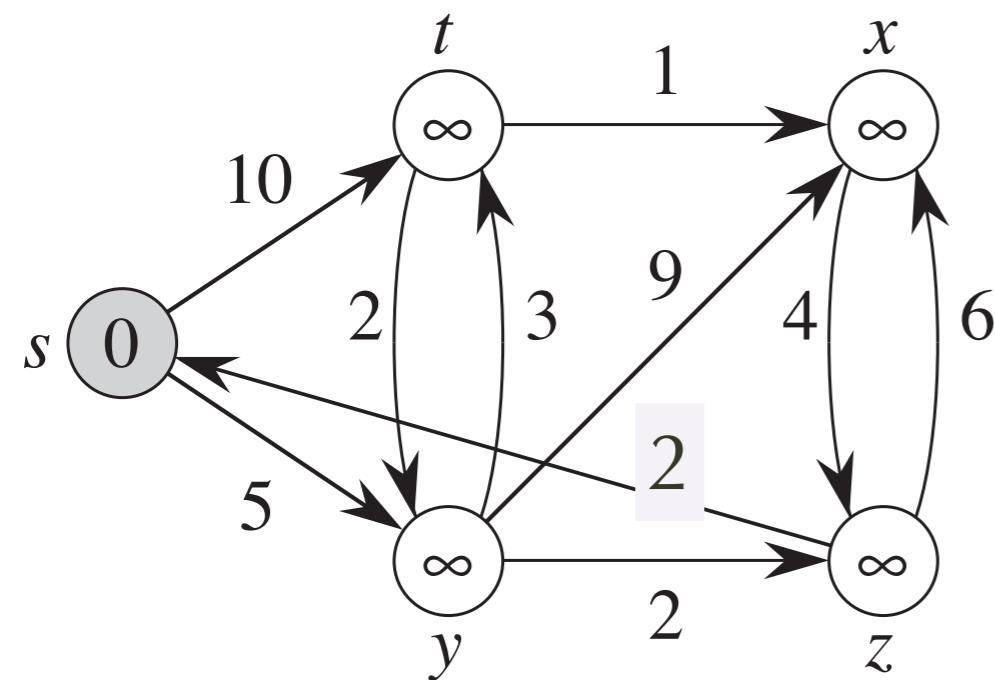
Floyd-Warshall algorithm



<i>s</i>	<i>t</i>	<i>x</i>	<i>y</i>	<i>z</i>
<i>s</i>	0	8	9	5
<i>t</i>	∞	0	1	2
<i>x</i>	∞	∞	0	∞
<i>y</i>	∞	3	4	0
<i>z</i>	2	10	6	7
				0

The shortest path from u to v that may passes s, t, x, y

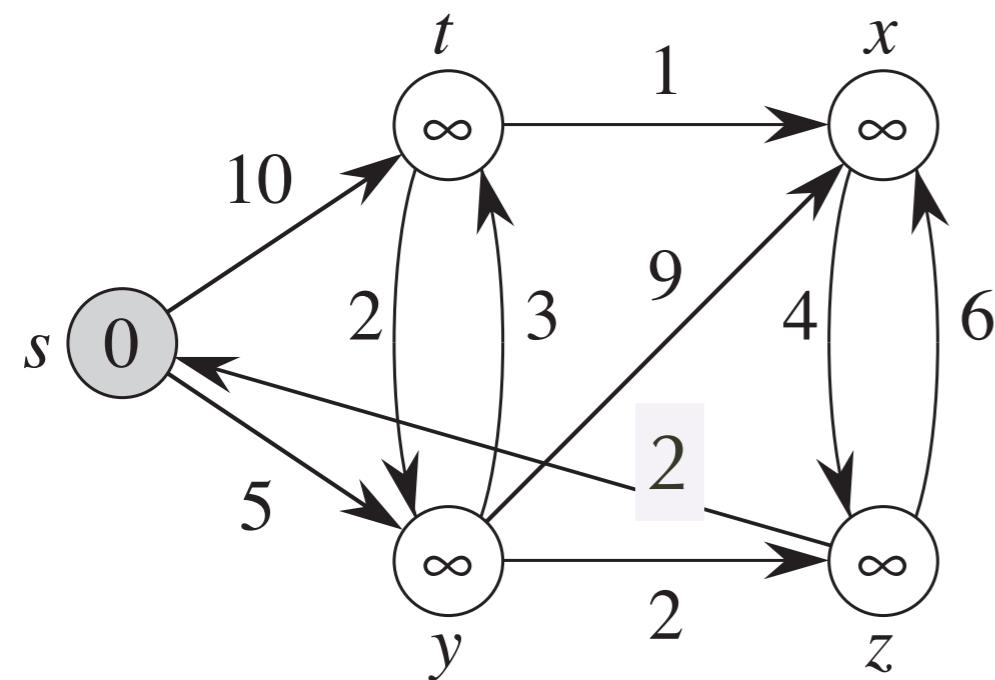
Floyd-Warshall algorithm



s	t	x	y	z
s	0	8	9	5
t	∞	0	1	2
x	∞	∞	0	11
y	∞	3	4	0
z	2	10	6	7
				0

The shortest path from u to v that may passes s, t, x, y

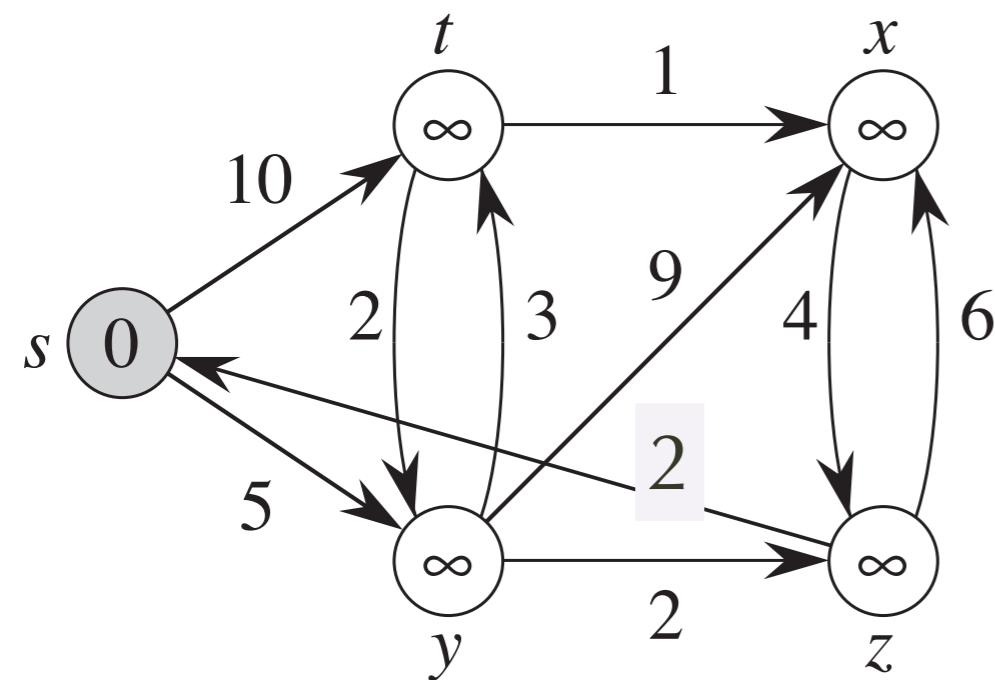
Floyd-Warshall algorithm



s	t	x	y	z
s	0	8	9	5
t	∞	0	1	2
x	∞	∞	0	11
y	4	3	4	0
z	2	10	6	7
				0

The shortest path from u to v that may passes s, t, x, y

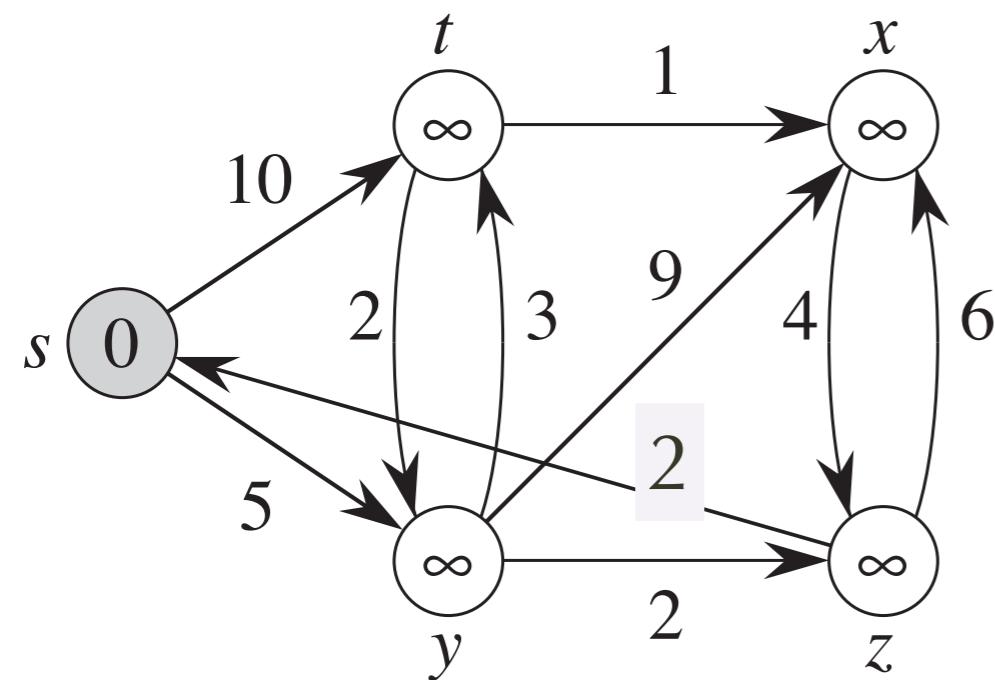
Floyd-Warshall algorithm



s	t	x	y	z
s	0	8	9	5
t	8	0	1	2
x	8	14	0	11
y	4	3	4	0
z	2	10	6	7
				0

The shortest path from u to v that may passes s, t, x, y

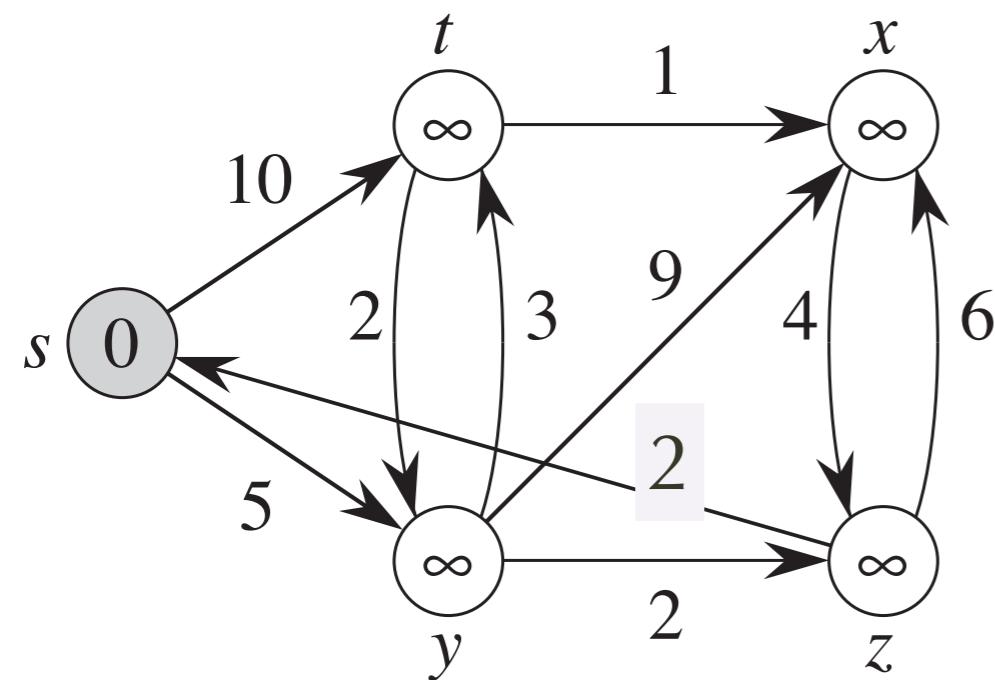
Floyd-Warshall algorithm



s	t	x	y	z
s	0	8	9	5
t	∞	0	1	2
x	6	14	0	11
y	4	3	4	0
z	2	10	6	7
				0

The shortest path from u to v that may passes s, t, x, y

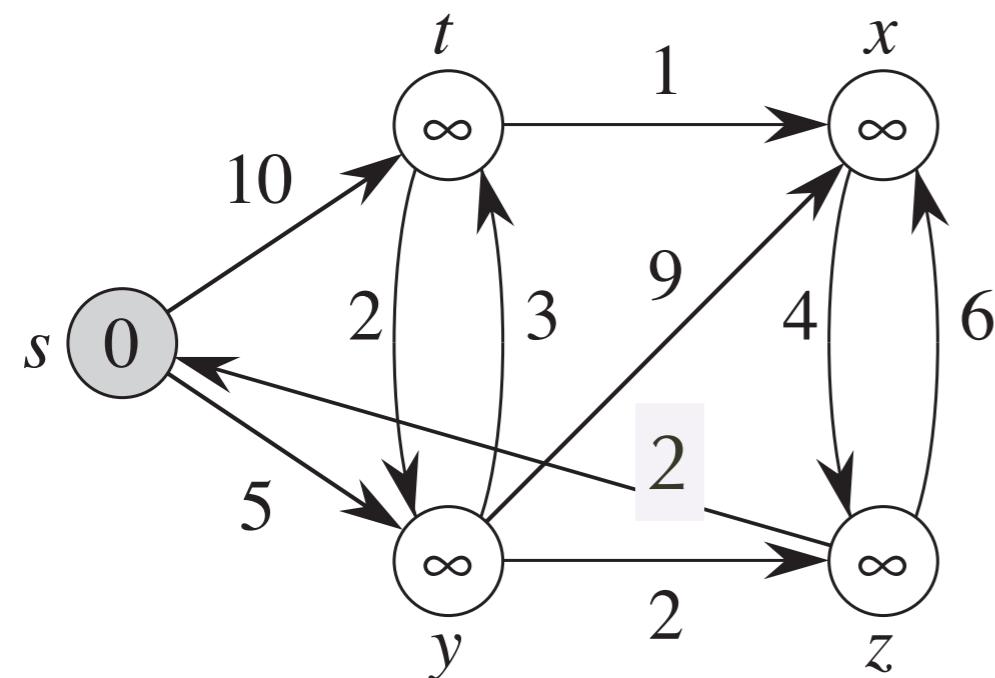
Floyd-Warshall algorithm



s	t	x	y	z
s	0	8	9	5
t	6	0	1	2
x	6	14	0	11
y	4	3	4	0
z	2	10	6	7
				0

The shortest path from u to v that may passes s, t, x, y

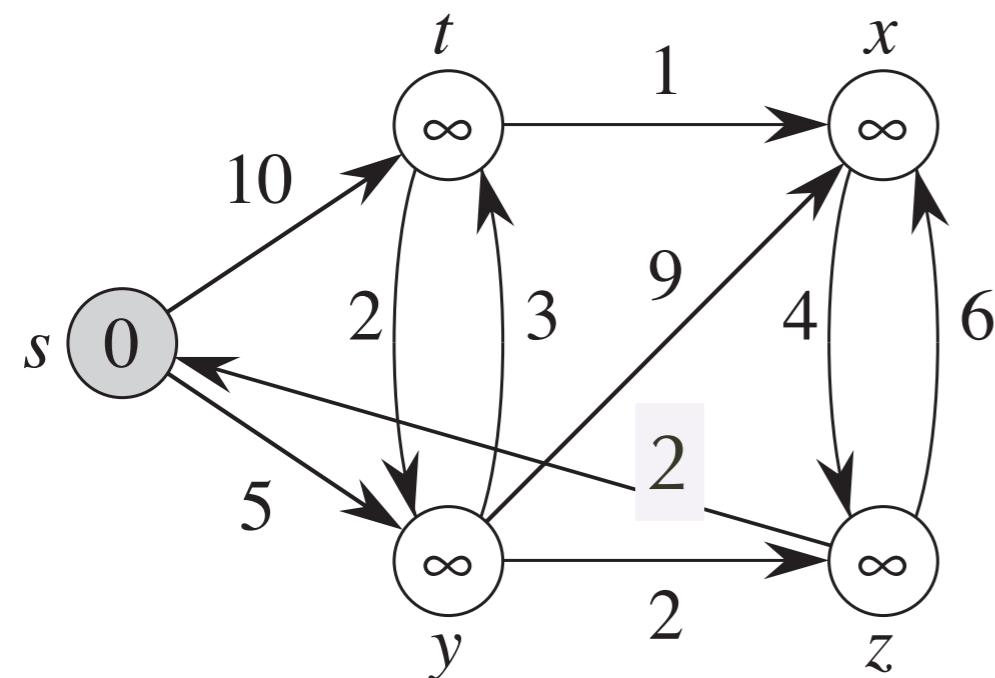
Floyd-Warshall algorithm



s	t	x	y	z	
s	0	8	9	5	7
t	6	0	1	2	4
x	6	14	0	11	4
y	4	3	4	0	2
z	2	10	6	7	0

The shortest path from u to v that may passes s, t, x, y

Floyd-Warshall algorithm



<i>s</i>	<i>t</i>	<i>x</i>	<i>y</i>	<i>z</i>	
<i>s</i>	0	8	9	5	7
<i>t</i>	6	0	1	2	4
<i>x</i>	6	14	0	11	4
<i>y</i>	4	3	4	0	2
<i>z</i>	2	10	6	7	0

The shortest path from *u* to *v* that may passes *s, t, x, y, z*

Lecture 9 Greedy Approach

- Minimum spanning tree
- How to design greedy algorithms

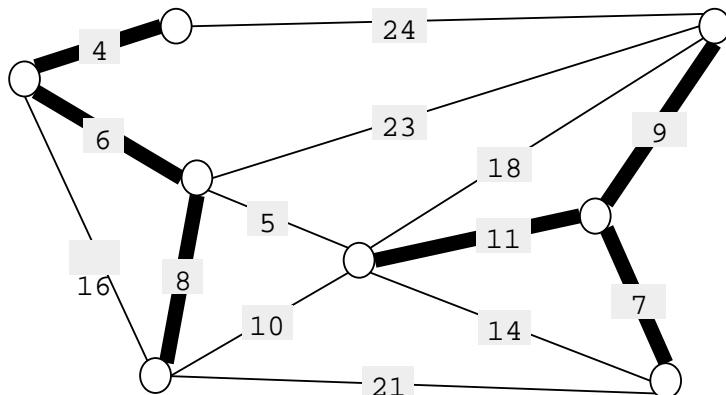
Roadmap

- Minimum spanning tree
- How to design greedy algorithms
 - Change-making problem
 - Activity selection problem
 - Huffman code
 - Knapsack problem

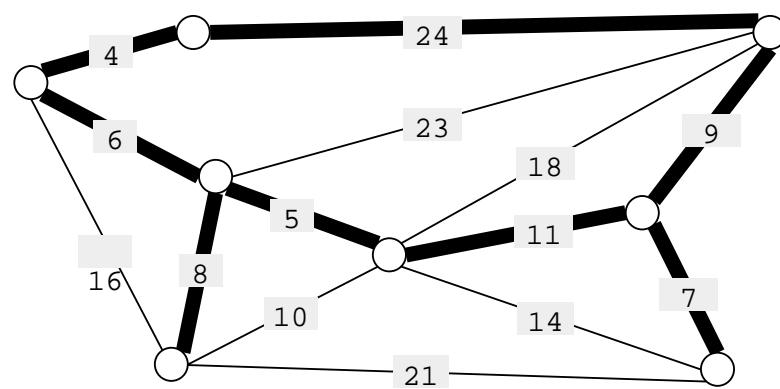
MST

Given. Undirected graph G with positive edge weights (connected).

Def. A **spanning tree** of G is a subgraph T that is connected and acyclic.



not connected



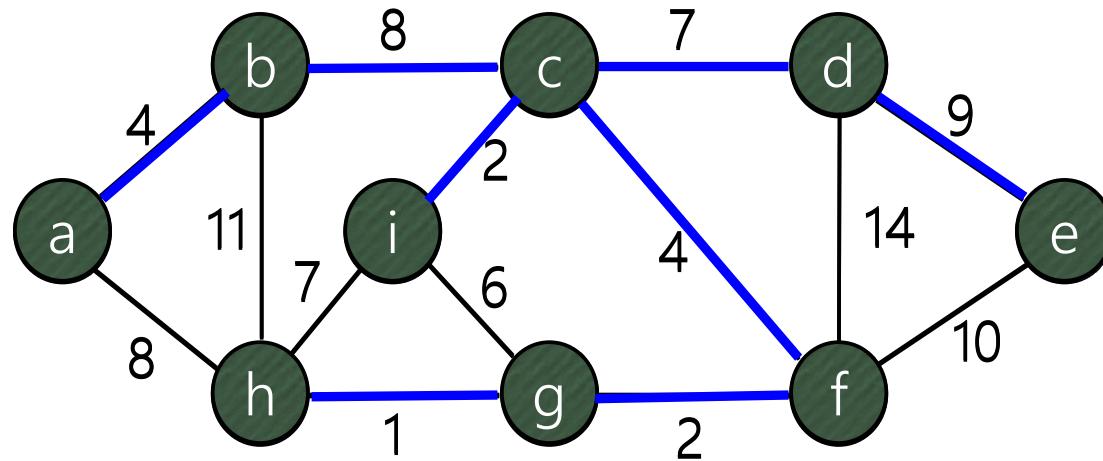
not acyclic

Minimum spanning tree

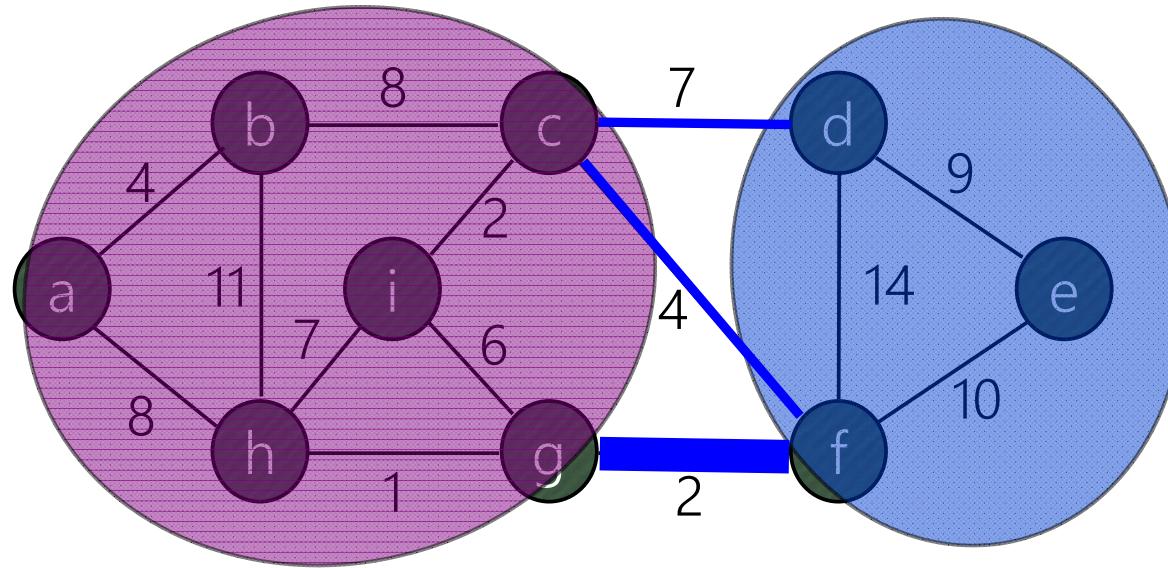
Problem: MinSpanning

Input: A connected undirected graph $G = (V, E)$ in which each edge has a weighted length

Output: A spanning tree of G that has minimum cost



Cut property



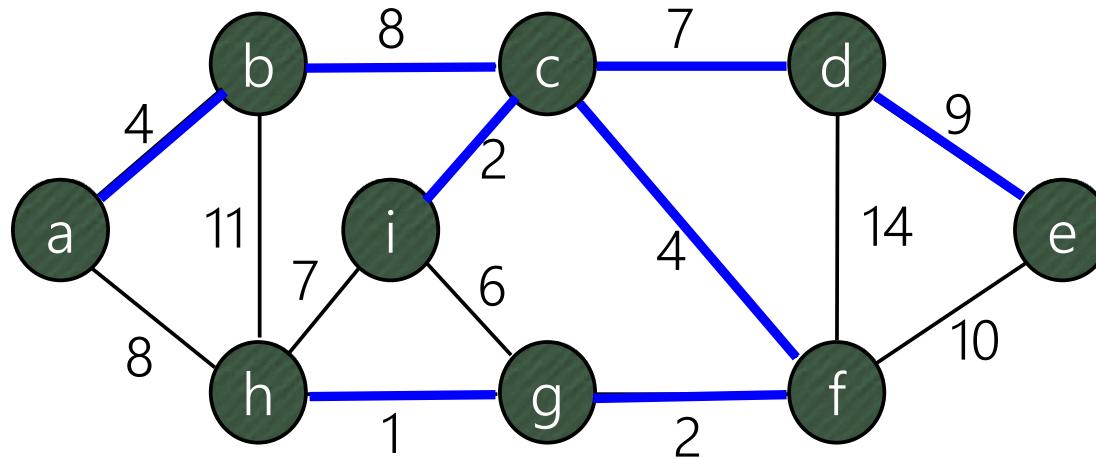
Definition: Cut

A *cut* $\{S, T\}$ is a partition of the vertex set V into two subsets S and T .

Theorem (Cut property) Given any *cut*, the crossing edge of min weight is in some MST.

Kruskal algorithm

(h,g), (c,i), (g,f), (a,b), (c,f), (c,d), (i,g), (i,h), (b,c),
(a,h), (d,e), (e,f), (b,h)



Kruskal algorithm

Algorithm 8.3 Kruskal

Input: A weighted connected undirected graph $G = (V, E)$ with n vertices.

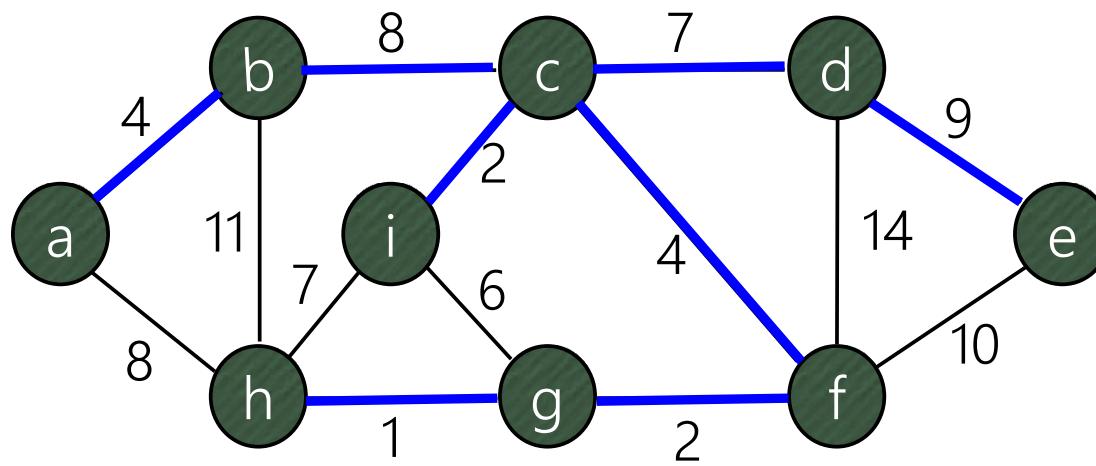
Output: The set of the edges T of a minimum cost spanning tree for G .

1. Sort the edges in E by nondecreasing weight.
2. for each vertex $v \in V$ do *Makeset($\{v\}$)* end for
3. $T = \{\}$
4. while $|T| < n - 1$
5. Let (x, y) be the next edge in E
6. if *Find(x) ≠ Find(y)* then
7. *Add(x, y) to T*
8. *Union(x, y)*
9. end if
10. end while



$\Theta(m \log m)$

Prim algorithm



Algorithm 8.4 Prim

Input: A weighted connected undirected graph $G = (V, E)$, where $V = \{1, 2, \dots, n\}$.

Output: The set of edges T of minimum cost spanning tree for G .

1. $T \leftarrow \{\}$; $X \leftarrow \{1\}$; $Y \leftarrow V \setminus \{1\}$
2. **for** $y \leftarrow 2$ to n
3. **if** y is adjacent to 1 **then**
4. $N[y] \leftarrow 1$
5. $C(y) \leftarrow c[1, y]$
6. **else** $C[y] \leftarrow \infty$
7. **end if**
8. **end for**
9. **for** $j \leftarrow 1$ to $n - 1$
10. Let y be such that $C[y]$ is minimum
11. $T \leftarrow T \cup \{(y, N[y])\}$
12. $X \leftarrow X \cup \{y\}$
13. $Y \leftarrow Y \setminus \{y\}$
14. **for each edge** (y, w) **such that** $w \in Y$
15. **if** $c[y, w] < C[w]$ **then**
16. $N[w] \leftarrow y$
17. $C[w] \leftarrow c[y, w]$
18. **end if**
19. **end for**
20. **end for**



$\Theta(n^2)$
• Heap: $\Theta(m \log n)$

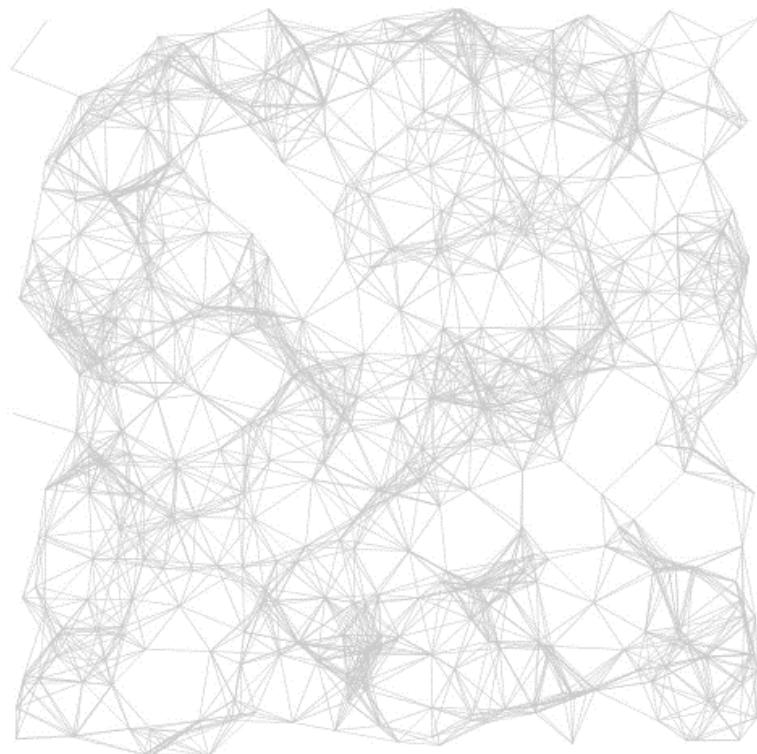
Dense graph

- dense: $m = n^{1+\varepsilon}$, ε is not too small.
- d-heap, $d = m/n$
- complexity: $O(nd\log_dn + m\log_dn) = O(m)$

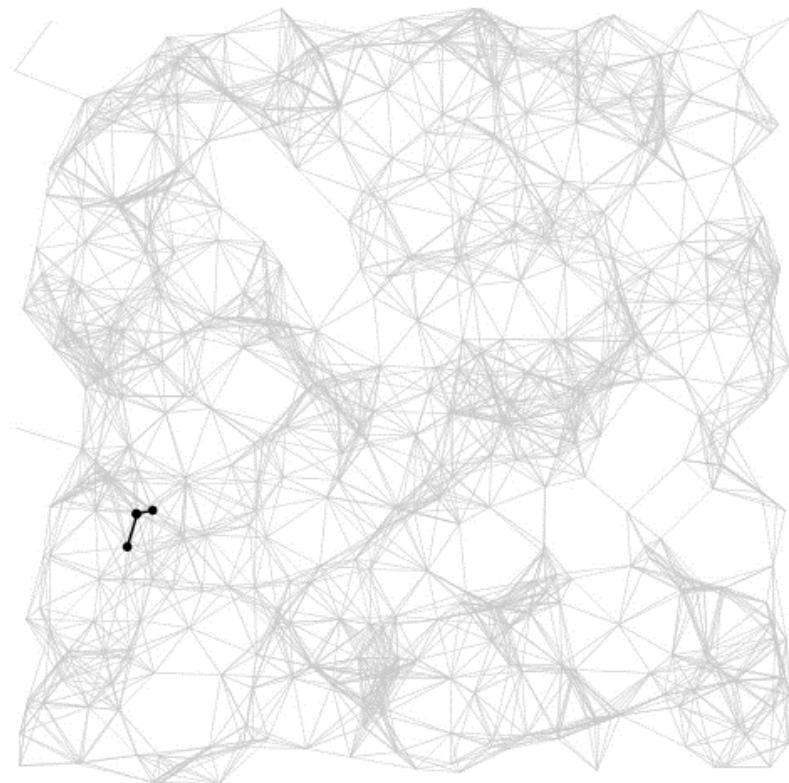
Comparison

	General	Dense
Prim	$O(m \log n)$	$O(m)$
Kruskal	$O(m \log m)$	$O(m \log m)$
Dijkstra	$O(m \log n)$	$O(m)$

Comparison



Kruskal demo



Prim demo

Correctness

Theorem (General MST Algorithm)

Let A be a subset of E that is included in some MST for G ,
let $\{V, S-V\}$ be any cut of G that respects A , and
let (u,v) be a lightest edge crossing $V, S-V$.
Then, $A \cup \{(u,v)\}$ is included in some MST.

Theorem

Kruskal algorithm and Prim algorithm correctly find a minimum cost spanning tree.

Greed is good.

Greed, for lack of a better word, is good. Greed is right. Greed works. Greed clarifies, cuts through, and captures, the essence of the evolutionary spirit. Greed, in all of its forms; greed for life, for money, for love, knowledge, has marked the upward surge of mankind and greed, you mark my words, will not only save Teldar Paper, but that other malfunctioning corporation called the U.S.A.

----- Michael Douglas, Wall Street

Greedy approach

A **greedy algorithm** always makes the choice that looks best *at the moment*.

locally optimal choices --> a globally optimal solution.

Greedy algorithms DO NOT always yield optimal solutions, but for many problems they do.

Dijkstra algorithm, Prim algorithm, Kruskal algorithm

Where are we?

- Minimum spanning tree
- How to design greedy algorithms
 - Change-making problem
 - Activity selection problem
 - Huffman code
 - Knapsack problem

Change making

Problem: ChangeMaking

Input: an integer m and a coin system a_1, a_2, \dots, a_n

Output: the minimum number of coins needed



1 2 5

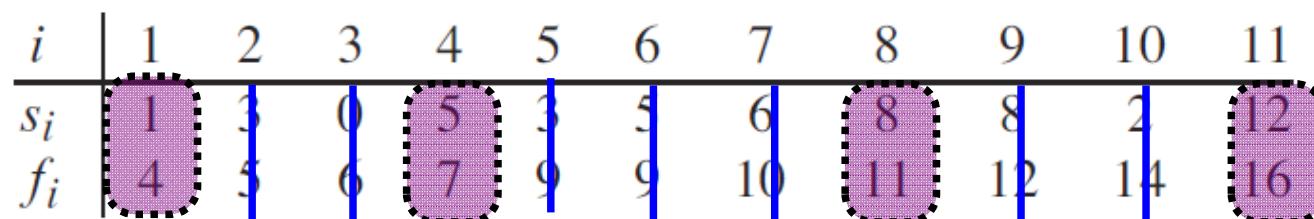
1 2 7 10 16?

Activity selection problem

Problem: ActivitySelection

Input: a set $S = \{a_1, a_2, \dots, a_n\}$ of n proposed activities, each a_i has a starting time s_i and a finishing time f_i with $0 \leq s_i < f_i < \infty$

Output: A maximum-size subset of compatible activities



Activity-selection

GREEDY-ACTIVITY-SELECTOR(s, f)

```
1   $n = s.length$ 
2   $A = \{a_1\}$ 
3   $k = 1$ 
4  for  $m = 2$  to  $n$ 
5      if  $s[m] \geq f[k]$ 
6           $A = A \cup \{a_m\}$ 
7           $k = m$ 
8  return  $A$ 
```

Theorem

Consider any nonempty subproblem S_k , and let a_m be an activity in S_k with the **earliest** finish time. Then a_m is included in some maximum-size subset of mutually compatible activities of S_k .

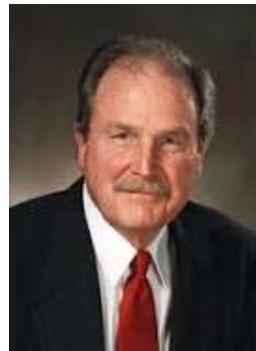
Huffman code



Robert Fano



Claude Shannon



David Huffman

Ambiguity

Morse code:

SOS ?

V7 ?

IAMIE ?

EEWNI ?

Letters	Numbers
A •—	1 • —————
B —•••	2 •• ———
C —•—•	3 ••• ——
D —••	4 •••• •—
E •	5 •••••
F ••—•	6 —••••
G ——•	7 ——•••
H ••••	8 —— —••
I ..	9 —— ——•
J •— —	0 —— —— —
K —•—	
L •—••	
M ——	
N —•	
O —— —	
P •— —•	
Q —— •—	
R •— •	
S •••	
T —	
U ••—	
V ••• —	
W •— —	
X —•• —	
Y —•— —	
Z —— ••	

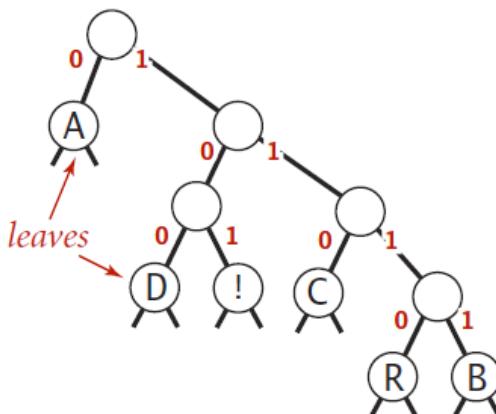
Prefix-free code

Prefix code: no codeword is a prefix of some other codeword.

codeword table

key	value
!	101
A	0
B	1111
C	110
D	100
R	1110

trie representation



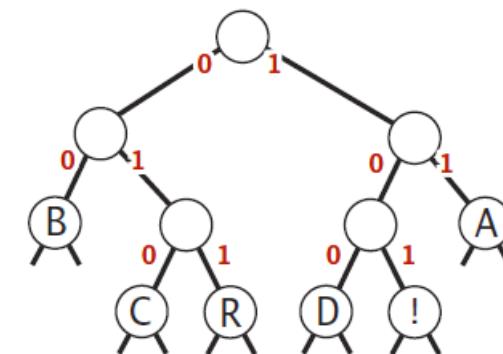
compressed bitstring

01111111001100100011111100101 ← 30 bits
A B RA CA DA B RA !

codeword table

key	value
!	101
A	11
B	00
C	010
D	100
R	011

trie representation



compressed bitstring

110001111010111100110001111101 ← 29 bits
A B R A C A D A B R A !

Use a binary tree to represent a prefix-free code.

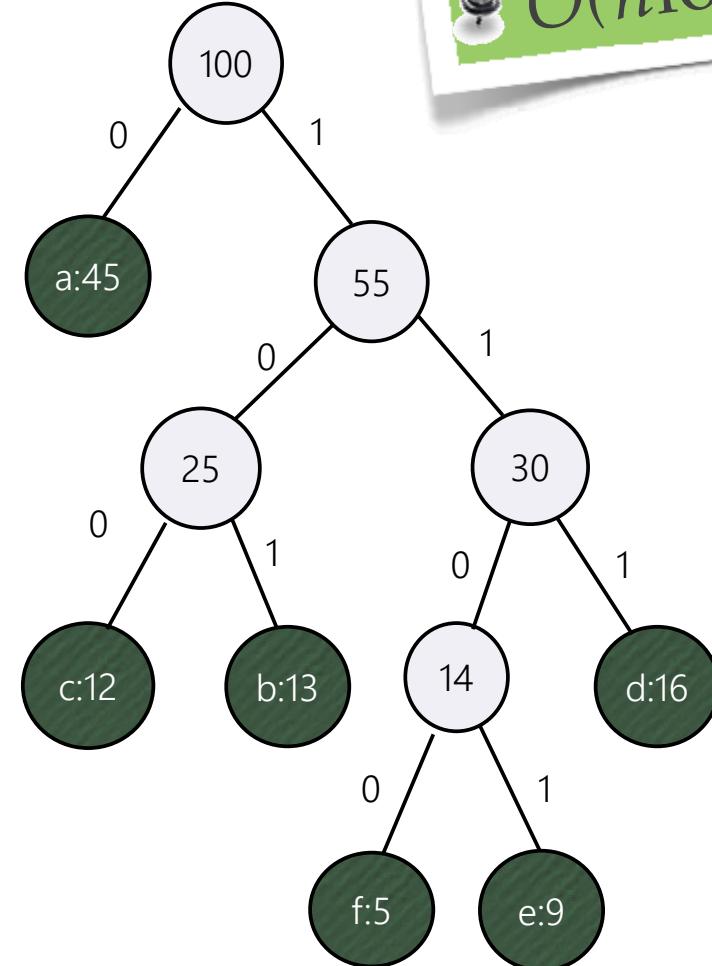
Huffman code

	a	b	c	d	e	f
Frequency	45	13	12	16	9	5
Fixed-length	000	001	010	011	100	101
Huffman code	0	101	100	111	1101	1100

100,000 characters, Fixed length code: 300,000 bits
Huffman code: 224,000 bits

Huffman code

a	45
b	13
c	12
d	16
e	9
f	5



O(nlogn)

Huffman code

Algorithm 8.6 Huffman

Input: A set $C = \{c_1, c_2, \dots, c_n\}$ of n characters and their frequencies $\{f(c_1), f(c_2), \dots, f(c_n)\}$.

Output: A Huffman tree (V, T) for C .

1. Insert all characters into a min-heap H according to their frequencies.
2. $V \leftarrow C; T \leftarrow \emptyset$
3. for $j \leftarrow 1$ to $n - 1$
4. $c \leftarrow \text{DeleteMin}(H)$
5. $c' \leftarrow \text{DeleteMin}(H)$
6. $f(v) \leftarrow f(c) + f(c')$
7. $\text{Insert}(H, v)$
8. $V \leftarrow V \cup \{v\}$
9. $T \leftarrow T \cup \{(v, c), (v, c')\}$
10. end for

Correctness

Lemma

Let x and y be two characters in C having the lowest frequencies.

Then there exists an *optimal prefix code* for C in which x and y have the same length and differ only in the last bit.

Lemma

Let x and y be two characters in C having the lowest frequencies.

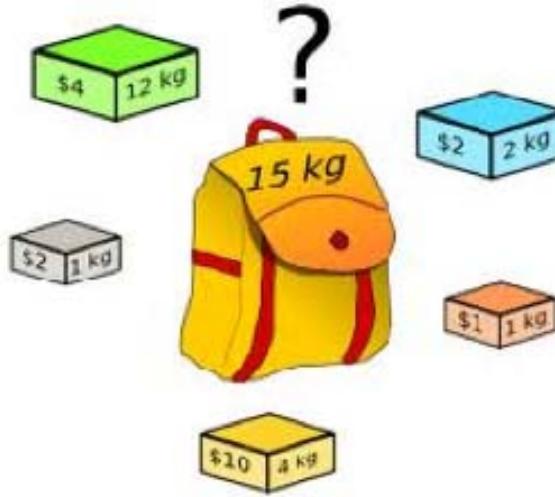
$$C' = C - \{x, y\} + \{z\}, f_z = f_x + f_y.$$

If T' is an optimal prefix code for C' ,
then $T = T' + \{(z, x), (z, y)\}$ is an optimal prefix code for C .

Theorem

Procedure HUFFMAN produces an optimal prefix code.

Knapsack problem



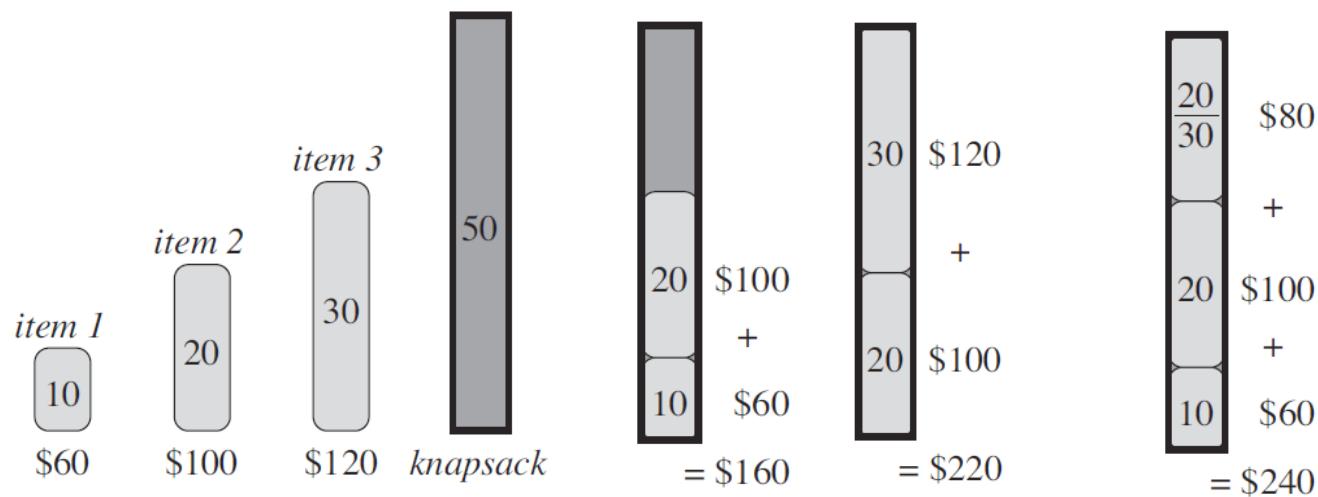
Problem: Knapsack

Input: A set of items $U = \{u_1, \dots, u_n\}$ with sizes s_1, s_2, \dots, s_n and values v_1, v_2, \dots, v_n and a knapsack capacity C

Output: The maximum value that can be put into the knapsack

Knapsack problem

What if the thief can take fractions of items?



0-1 knapsack

Horn formula

Boolean variables:

x = the murder took place in the kitchen

y = the butler is innocent

z = the colonel was asleep at 8 pm

Horn formula:

Implication $(z \wedge w) \Rightarrow u$

Negative clause $(\bar{u} \vee \bar{v} \vee \bar{y})$

Horn formula

Problem: HornFormula

Input: Horn formula

Output: *a satisfying assignment, if one exists*

$$\begin{aligned} & (w \wedge y \wedge z) \Rightarrow x, \quad (x \wedge z) \Rightarrow w, \quad x \Rightarrow y, \\ & \Rightarrow x, \quad (x \wedge y) \Rightarrow w, \quad (\bar{w} \vee \bar{x} \vee \bar{y}), \quad (\bar{z}). \end{aligned}$$

1. set all variables to **false**
2. *while* there is an **implication** that is not satisfied:
 3. set the right-hand variable of the implication to **true**
 4. *if* all pure **negative** clauses are satisfied:
 5. *return* the assignment
 6. *else*:
 7. *return* "formula is not satisfiable"

Correctness

Theorem

If a certain set of variables is set to true, then they must be true in any satisfying assignment.

Conclusion

- Minimum spanning tree
- How to design greedy algorithms
 - Change-making problem
 - Activity selection problem
 - Huffman code
 - Knapsack problem

Lecture 10 Dynamic programming

- What is Dynamic Programming
- How to write Dynamic Programming paradigm
- When to apply Dynamic Programming

Roadmap

- What is Dynamic Programming
- How to write DP paradigm
 - Knapsack problem
 - Longest common subsequence
 - Matrix chain multiplication
- When to apply Dynamic Programming
- Practice

Techniques Based on Recursion

- Induction or Tail recursion
- Non-overlapping subproblems
- Overlapping subproblems

An Introductory Example

$$\text{Fib}(0) = 0$$

$$\text{Fib}(1) = 1$$

$$\text{Fib}(n) = \text{Fib}(n - 1) + \text{Fib}(n - 2)$$



19th century statue of Fibonacci in Camposanto, Pisa.
(Source from [Wikipedia](#))

Problem: Fibonacci

Input: A positive integer n

Output: $\text{Fib}(n)$

Quiz

Problem: Fibonacci

Input: A positive integer n

Output: $Fib(n)$

Give your solution of Fibonacci and explain the time/space complexity.

Recursion implementation

```
int fib_rec (int n){  
    if (n==1 || n==2) {  
        return 1;  
    }  
    else {  
        return fib_rec(n-1) + fib_rec(n-2);  
    }  
}
```

$$\begin{aligned} T(n) &= T(n-1) + T(n-2) + 2, \quad T(1), T(2) < 3 \\ T(n) &> \text{Fib}_n \\ S(n) &= O(n) \end{aligned}$$

$O(n)$ time, $O(n)$ space

```
int fib (int n){  
    int* array = new int[n];  
    array[0] = 0;  
    array[1] = 1;  
    for (int i = 2; i <= n; i++)  
        array[i] = array[i-1] + array[i-2];  
    return array[n];  
}
```

$$\begin{aligned} T(n) &= n+1 \\ S(n) &= n+1 \end{aligned}$$

$O(n)$ time, $O(1)$ space

```
int fib (int n){  
    if (n < 2) return n;  
  
    int f0 = 0;  
    int f1 = 1;  
    int i = 2;  
    while (i <= n) {  
        f1 = f1 + f0;  
        f0 = f1 - f0;  
        i++;  
    }  
    return f1;  
}
```

$$T(n) < 3n$$

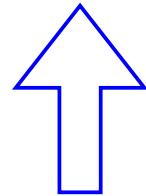
$$S(n) = 3$$

We can do even better: $T(n) = O(\log n)$

$O(\log n)$ time, $O(1)$ space

$$\begin{pmatrix} f_n & f_{n-1} \\ f_{n-1} & f_{n-2} \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \times \begin{pmatrix} f_{n-1} & f_{n-2} \\ f_{n-2} & f_{n-3} \end{pmatrix}$$

$$\begin{pmatrix} f_n & f_{n-1} \\ f_{n-1} & f_{n-2} \end{pmatrix} = \left(\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n \right)^{n-2} \times \begin{pmatrix} f_2 & f_1 \\ f_1 & f_0 \end{pmatrix}$$



$$T(n) = T\left(\frac{n}{2}\right) + \Theta(1)$$

$$T(n) = \Theta(\log n)$$

Use Divide-and-conquer

Fibonacci

$$\text{Fib}(0) = 0$$

$$\text{Fib}(1) = 1$$

$$\text{Fib}(n) = \text{Fib}(n - 1) + \text{Fib}(n - 2)$$

Overlapping subproblems



19th century statue of Fibonacci in Camposanto, Pisa.
(Source from *Wikipedia*)



Recursion? No, thanks.

Dynamic programming

Non-recursively handle recursive problems

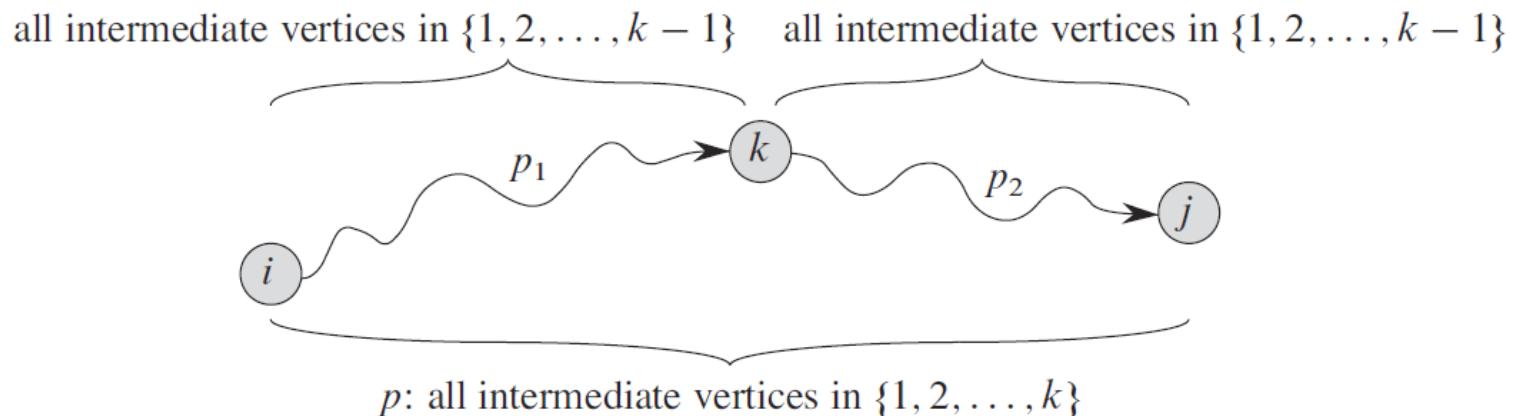
Time-Memory tradeoff

1. Find smaller subproblems
2. Write the relation expression
3. Use an array to save the intermediate results
4. Update the array iteratively.

All-pairs shortest path

Define $d_{i,j}^k$ to be the length of a shortest path from i to j that does not pass any vertex in $\{k+1, k+2, \dots, n\}$. Clearly

$$d_{i,j}^k = \begin{cases} l[i,j] & \text{if } k = 0 \\ \min\{d_{i,j}^{k-1}, d_{i,k}^{k-1} + d_{k,j}^{k-1}\} & \text{if } 1 \leq k \leq n \end{cases}$$



Where are we?

- What is Dynamic Programming
- How to write DP paradigm
 - Knapsack problem
 - Longest common subsequence
 - Matrix chain multiplication
- When to apply Dynamic Programming
- Practice

Knapsack problem



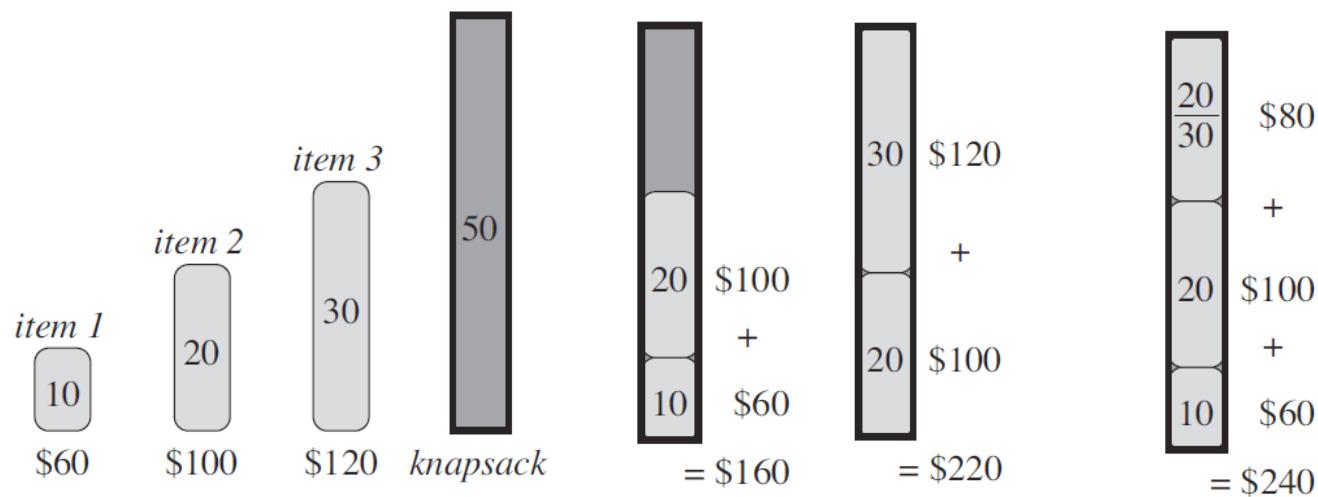
Problem: Knapsack

Input: A set of items $U = \{u_1, \dots, u_n\}$ with sizes s_1, s_2, \dots, s_n and values v_1, v_2, \dots, v_n and a knapsack capacity C

Output: The maximum value that can be put into the knapsack

Knapsack problem

What if the thief can take fractions of items?



0-1 knapsack

Knapsack problem

$V[i,j]$: the **maximum** value obtained by filling a knapsack of size j with items taken from the first i items $\{u_1, \dots, u_i\}$.

$$V[i, j] = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \\ V[i - 1, j] & \text{if } j < s_i \\ \max\{V[i - 1, j], V[i - 1, j - s_i] + v_i\} & \text{if } i > 0 \text{ and } j \geq s_i \end{cases}$$

Knapsack problem

Algorithm 7.4 Knapsack

```
1. for  $i \leftarrow 0$  to  $n$  do  $V[i, 0] \leftarrow 0$ 
2. for  $j \leftarrow 0$  to  $C$  do  $V[0, j] \leftarrow 0$ 
3. for  $i \leftarrow 1$  to  $n$ 
4.   for  $j \leftarrow 1$  to  $C$ 
5.      $V[i, j] \leftarrow V[i - 1, j]$ 
6.     if  $s_i \leq j$  then  $V[i, j] \leftarrow \max\{V[i, j], V[i - 1, j - s_i] + v_i\}$ 
7.   end for
8. end for
9. return  $V[n, C]$ 
```



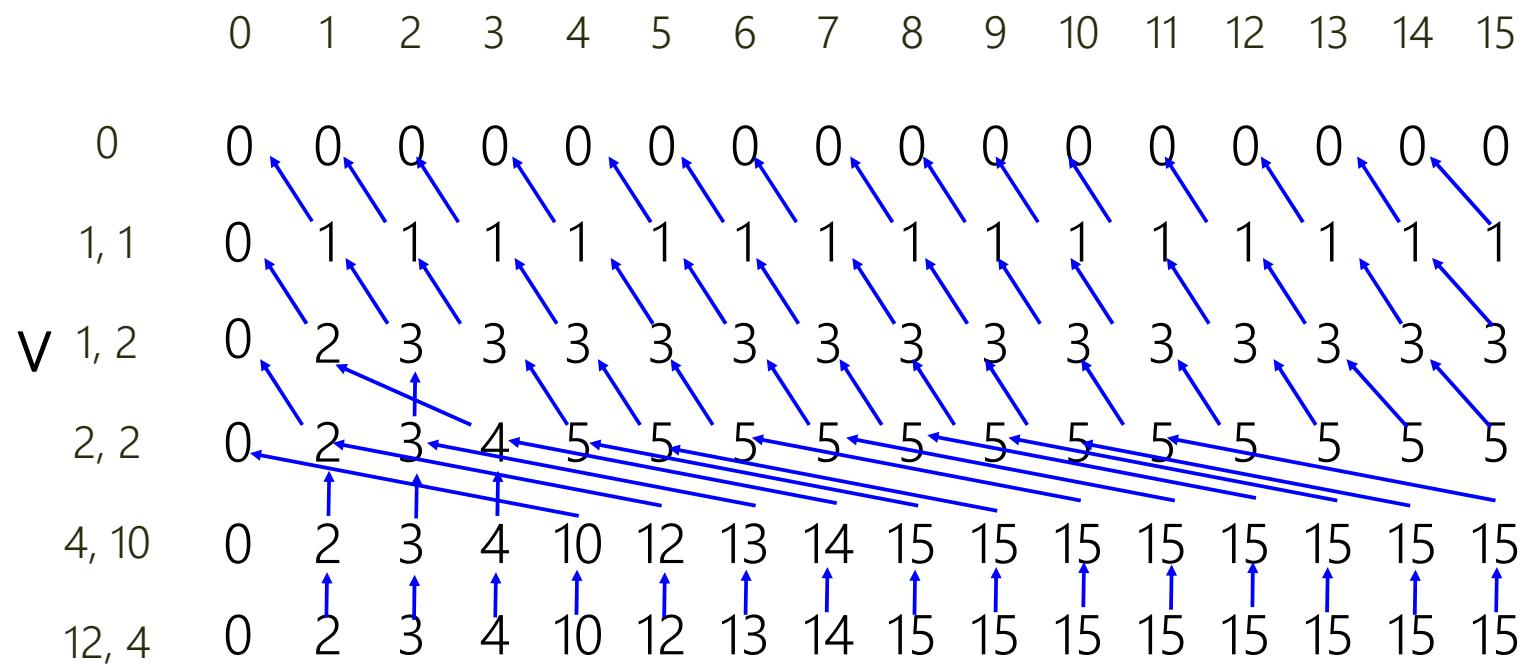
Time: $\Theta(nC)$



Space: $\Theta(nC)$

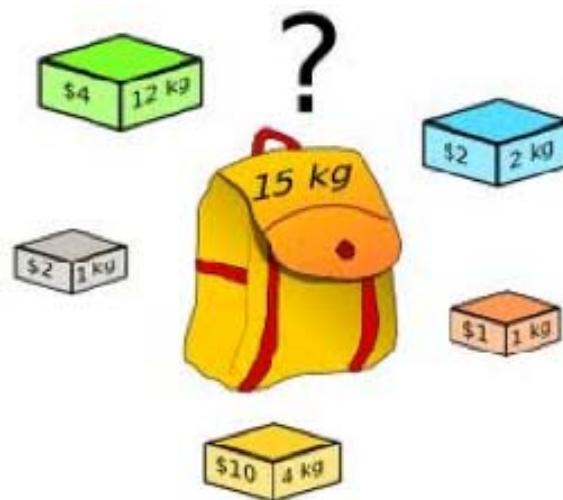
Pseudo-polynomial algorithm

How to construct the solution?



Knapsack problem

What if the thief robs a super-market? Knapsack with repetition.



Longest common subsequence

Problem: LCS

Input: Two strings $A=a_1a_2\dots a_n$ and $B=b_1b_2\dots b_m$

Output: The longest common subsequence of A and B

$L[i,j]$: length of the longest
common subsequence of
 $A[1..i]$ and $B[1..j]$

ababe

abcabc

Solution: $L[n,m]$

$$L[i,j] = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \\ L[i-1, j-1] + 1 & \text{if } i > 0, j > 0 \text{ and } a_i = b_j \\ \max\{L[i-1, j], L[i, j-1]\} & \text{if } i > 0, j > 0 \text{ and } a_i \neq b_j \end{cases}$$

LCS

$$L[i, j] = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \\ L[i - 1, j - 1] + 1 & \text{if } i > 0, j > 0 \text{ and } a_i = b_j \\ \max\{L[i - 1, j], L[i, j - 1]\} & \text{if } i > 0, j > 0 \text{ and } a_i \neq b_j \end{cases}$$

	0	a	b	c	a	b	c
0	0	0	0	0	0	0	0
a	0	1 ↘	1 ↘	1 ↘	1 ↘	1 ↘	1 ↘
b	0	1 ↗	2 ↗	2 ↗	2 ↗	2 ↗	2 ↗
a	0	1 ↗	2 ↗	2 ↗	3 ↗	3 ↗	3 ↗
b	0	1 ↗	2 ↗	2 ↗	3 ↗	4 ↗	4 ↗
e	0	1 ↗	2 ↗	2 ↗	3 ↗	4 ↗	4 ↗

LCS

Algorithm 7.1 LCS

Input: Two strings A and B of length n and m over Σ .

Output: The length of the longest common subsequence of A, B .

1. **for** $i \leftarrow 0$ **to** n **do** $L[i, 0] \leftarrow 0$ **end for**
2. **for** $j \leftarrow 0$ **to** m **do** $L[0, j] \leftarrow 0$ **end for**
3. **for** $i \leftarrow 1$ **to** n
4. **for** $j \leftarrow 1$ **to** m
5. **if** $a_i = b_j$ **then** $L[i, j] \leftarrow L[i - 1, j - 1] + 1$
6. **else** $L[i, j] \leftarrow \max\{L[i - 1, j], L[i, j - 1]\}$
7. **end if**
8. **end for**
9. **end for**
10. **return** $L[n, m]$



Time: $\Theta(mn)$

Space: $\Theta(\max\{m, n\})$

Longest common substring?

$L[i,j]$: length of longest common **sub-postfix** of $a[1..i]$ and $b[1..j]$.

$$L[i, j] = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \\ L[i - 1, j - 1] + 1 & \text{if } i > 0, j > 0 \text{ and } a_i = b_j \\ 0 & \text{if } i > 0, j > 0 \text{ and } a_i \neq b_j \end{cases}$$

Solution: the **maximum** $L[i,j]$ for all $i>0$ and $j>0$

Matrix chain multiplication

What is the most **efficient** way of computing the following multiplication?

$$\begin{pmatrix} a_{11} \dots a_{15} \\ \vdots \\ a_{61} \dots a_{65} \end{pmatrix} \begin{pmatrix} b_{11} \dots b_{14} \\ \vdots \\ b_{51} \dots b_{54} \end{pmatrix} \begin{pmatrix} c_{11} \dots c_{13} \\ \vdots \\ c_{41} \dots c_{43} \end{pmatrix}$$

The order of multiplications matters.

Quiz

How many different ways to compute

$$M_1 * M_2 * \dots * M_n$$

Matrix chain multiplication

Problem: MCM

Input: A matrix chain $M_1M_2\dots M_n$ with rank $r_1r_2\dots r_nr_{n+1}$

Output: Minimal cost of the multiplication

ABCD
50,20,1,10,100

Parenthesization	Cost computation	Cost
$A \times ((B \times C) \times D)$	$20 \cdot 1 \cdot 10 + 20 \cdot 10 \cdot 100 + 50 \cdot 20 \cdot 100$	120,200
$(A \times (B \times C)) \times D$	$20 \cdot 1 \cdot 10 + 50 \cdot 20 \cdot 10 + 50 \cdot 10 \cdot 100$	60,200
$(A \times B) \times (C \times D)$	$50 \cdot 20 \cdot 1 + 1 \cdot 10 \cdot 100 + 50 \cdot 1 \cdot 100$	7,000

Dynamic programming

$C[i,j]$: the minimal cost of the multiplication $M_i M_{i+1} \dots M_j$

$$C[i, j] = \min_{i < k \leq j} \{ C[i, k-1] + C[k, j] + r_i r_k r_{j+1} \}$$

Memoization

	A	B	C	D
A	0	1000	1500	7000
B		0	200	3000
C			0	1000
D				0

ABCD
50,20,1,10,100

MCM

Algorithm 7.2 MatChain

```
1. for  $i \leftarrow 1$  to  $n$ 
2.    $C[i, i] \leftarrow 0$ 
3. end for
4. for  $d \leftarrow 1$  to  $n - 1$ 
5.   for  $i \leftarrow 1$  to  $n - d$ 
6.      $j \leftarrow i + d$ 
7.      $C[i, j] \leftarrow \infty$ 
8.     for  $k \leftarrow i + 1$  to  $j$ 
9.        $C[i, j] \leftarrow \min\{C[i, j], C[i, k-1] + C[k, j] + r[i]r[k]r[j+1]\}$ 
10.    end for
11.   end for
12. end for
13. return  $C[1, n]$ 
```

$$\sum_{d=1}^{n-1} \sum_{i=1}^{n-d} \sum_{k=1}^d c = \frac{cn^3 - cn}{6} = \Theta(n^3)$$

Dynamic programming

$C[i, j]$: the minimal cost of the multiplication $M_i M_{i+1} \dots M_j$

$$C[i, j] = \min_{i < k \leq j} \{ C[i, k-1] + C[k, j] + r_i r_k r_{j+1} \}$$

ABCD
50,20,1,10,100

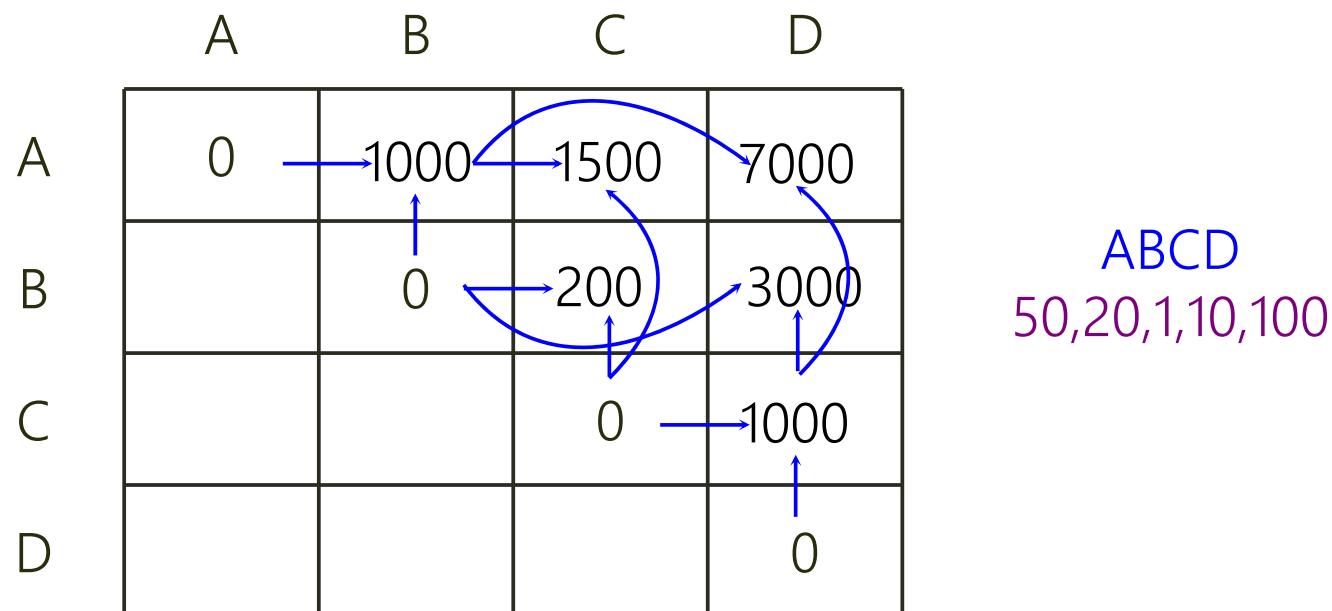
	A	B	C	D
A	0	1000	1500	7000
B		0	200	3000
C			0	1000
D				0

	A	B	C	D
A	1	2	3	3
B		2	3	4
C			3	4
D				4

Dynamic programming

$C[i,j]$: the minimal cost of the multiplication $M_i M_{i+1} \dots M_j$

$$C[i, j] = \min_{i < k \leq j} \{ C[i, k-1] + C[k, j] + r_i r_k r_{j+1} \}$$



Where are we?

- What is Dynamic Programming
- How to write DP paradigm
 - Knapsack problem
 - Longest common subsequence
 - Matrix chain multiplication
- When to apply Dynamic Programming
- Practice

Elements of DP

- Optimal substructure
 - A problem exhibits **optimal substructure** if an optimal solution to the problem contains within it optimal solutions to subproblems.
- Overlapping subproblems

Optimal Substructure

Given a graph $G = \langle V, E \rangle$, which problem has optimal substructure, **longest** simple path problem or **shortest** simple path problem?

Optimal Binary search tree

Problem: BinarySearching

Input: A sorted integer array $K[1\dots n]$, and an integer x

Output: Does x exist in A ?

Binary Search Tree (BST)

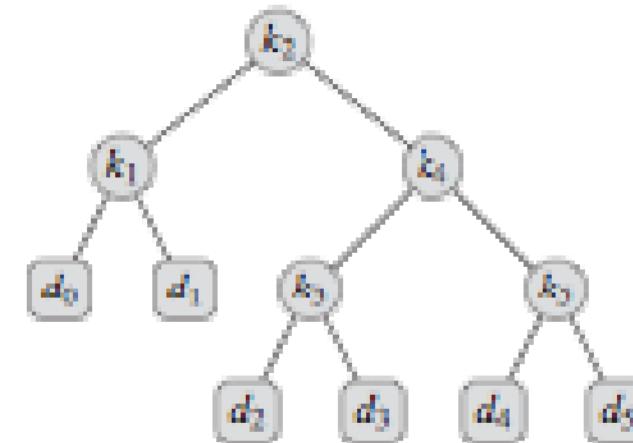
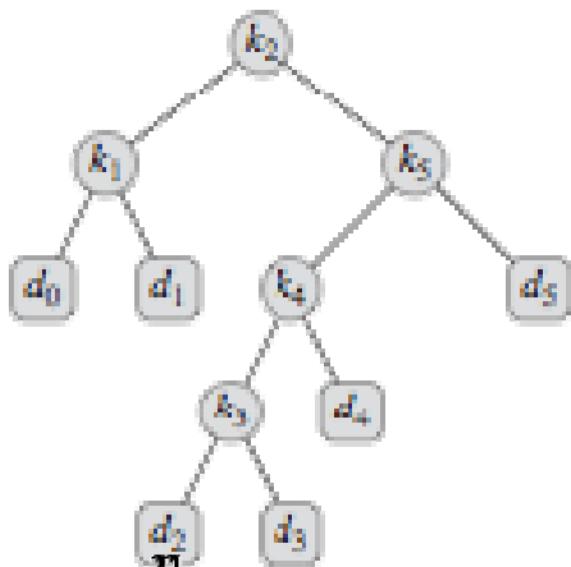
Given the possibilities that a search will lie in the intervals:

$< k_1$	k_1	$>k_1, < k_2$	k_2	$>k_2, < k_3$...	k_n	$>k_n$
q_0	p_1	q_1	p_2	q_2	...	p_n	q_n

$$\sum_{i=0}^n q_i + \sum_{i=1}^n p_i = 1$$

Search cost

d ₀	k ₁	d ₁	k ₂	d ₂	k ₃	d ₃	k ₄	d ₄	k ₅	d ₅
0.05	0.15	0.10	0.10	0.05	0.05	0.05	0.10	0.05	0.20	0.10



$$\text{cost} = \sum_{i=0}^n q_i \cdot (\text{depth}(d_i) + 1) + \sum_{i=1}^n p_i \cdot (\text{depth}(k_i) + 1)$$

OPT BST

Optimal substructure & overlapping subproblems

$e[i,j]$: The cost of optimal binary search tree for k_i, k_{i+1}, \dots, k_j

d_0	k_1	d_1	k_2	d_2	k_3	d_3	k_4	d_4	k_5	d_5
0.05	0.15	0.10	0.10	0.05	0.05	0.05	0.10	0.05	0.20	0.10

$$e[i,j] = q_j \text{ if } j = i-1$$

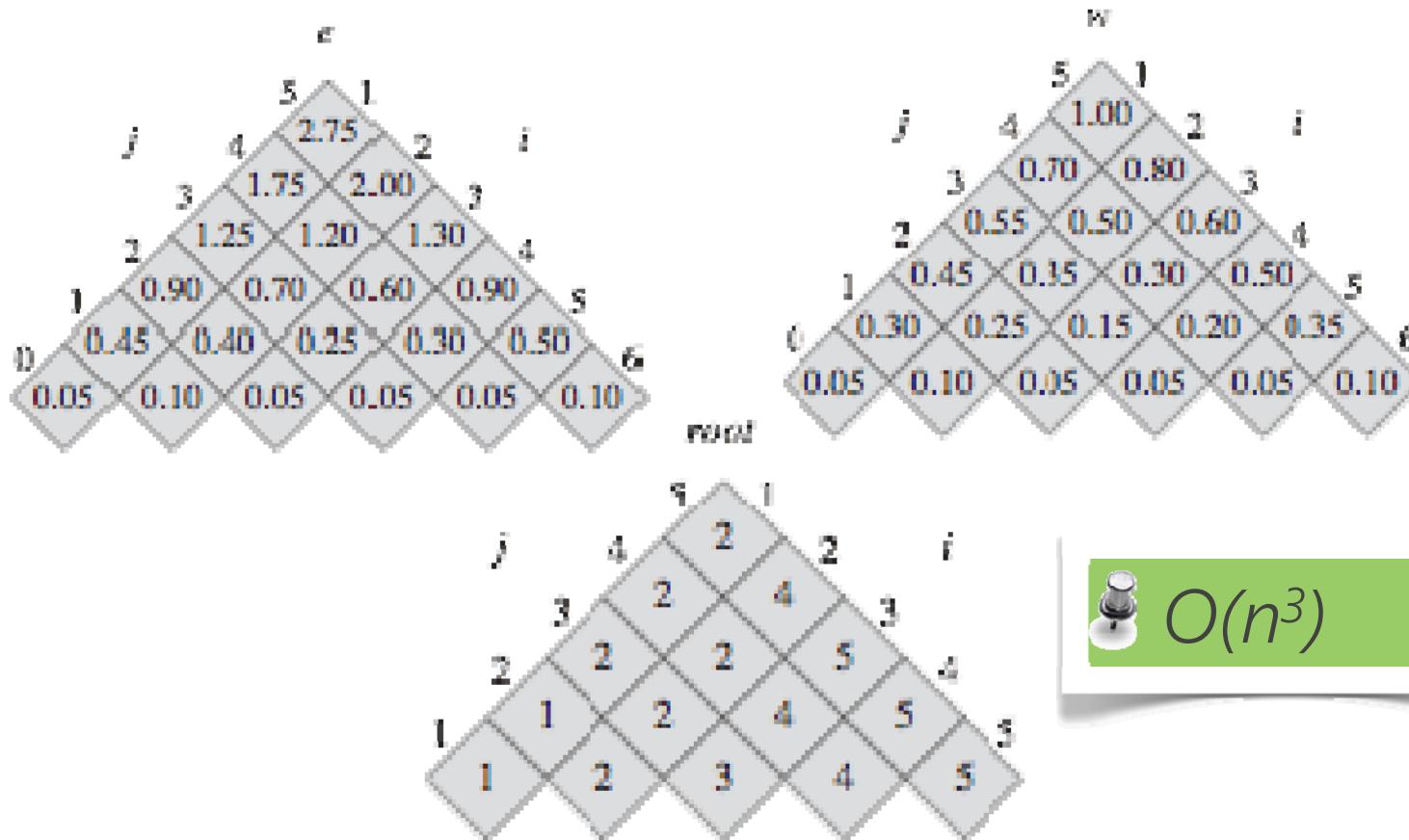
$$e[i,j] = \min_{i \leq r \leq j} \{ e[i,r-1] + e[r+1,j] + w[i,j] \} \text{ if } j \geq i$$

where

$$w[i,j] = q_j \text{ if } j = i-1$$

$$w[i,j] = w[i,r-1] + w[r+1,j] + p_r \text{ if } j \geq i$$

Memoization



Practice

- Subset sum
- Longest palindrome sequence
- Editing distance
- Longest increasing subsequence
- Planning a company party

Subset sum problem

Problem: SubsetSum

Input: A set of numbers $A = \{a_1, a_2, \dots, a_n\}$, and a sum s

Output: Yes, if there exists a subset $B \subseteq A$ such that the sum of B equals to s ; No, otherwise.

$L[i, j]$: Does there exist a subset B of $\{a_1, \dots, a_i\}$
such that the sum of B equals to j

$$L[i, j] = \begin{cases} 0 & \text{if } i = 0 \\ 1 & \text{if } j = 0 \\ L[i - 1, j] & \text{if } i > 0, j > 0, j < a_i \\ L[i - 1, j - a_i] \vee L[i - 1, j] & \text{if } i > 0, j > 0, j \geq a_i \end{cases}$$

Solution: $L[n, s]$

Longest palindrome subsequence

A **palindrome** is a nonempty string over some alphabet that reads the same forward and backward.

civic, racecar

Problem: PalindromeSubsequence

Input: A string $A = a_1a_2\dots a_n$

Output: Length of the longest subsequence of A that is a palindrome



We can also reduce this problem to the *Longest Common Subsequence* problem of $a_1a_2\dots a_n$ and $a_n\dots a_2a_1$.

(by Qiu Zhe.)

$$L[i,j] = \max\{L[i+1,j], L[i,j-1]\} \text{ O.W}$$

Editing distance

snowy --> sunny

S	-	N	O	W	Y
S	U	N	N	-	Y
Cost: 3					

-	S	N	O	W	-	Y
S	U	N	-	-	N	Y
Cost: 5						

The *edit distance* between two strings is the cost of their best possible alignment.

$E[i,j]$: edit distance of $a_1 \dots a_i$ and $b_1 \dots b_j$

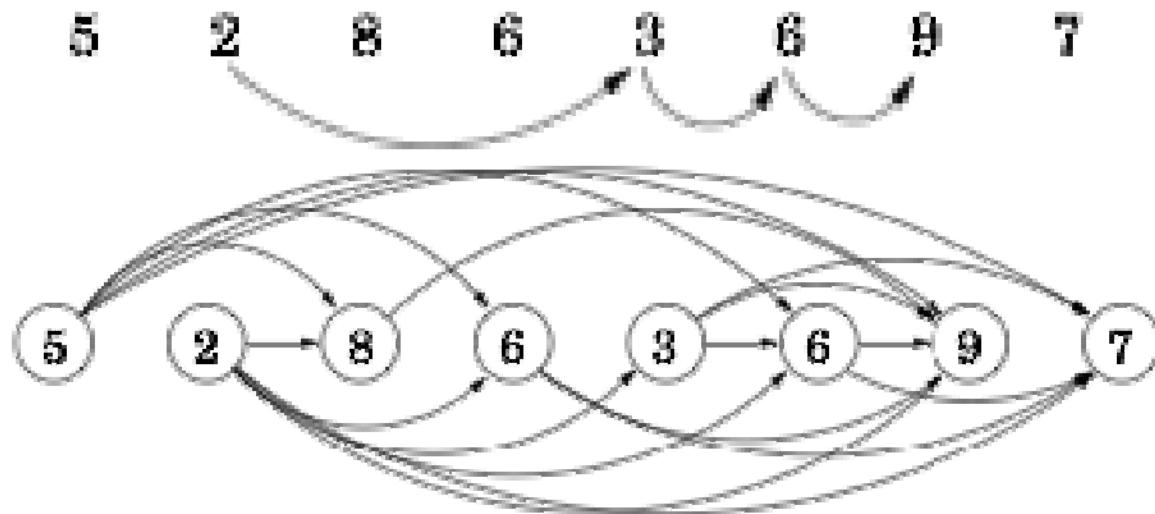
$$E[i,j] = \begin{cases} j & \text{if } i = 0 \\ i & \text{if } j = 0 \\ \min E[i-1, j-1], E[i-1, j] + 1, E[i, j-1] + 1 & \text{if } i, j > 0 \text{ and } a_i = b_j \\ \min E[i-1, j-1] + 1, E[i-1, j] + 1, E[i, j-1] + 1 & \text{if } i, j > 0 \text{ and } a_i \neq b_j \end{cases}$$

Editing distance

$$E[i] = \begin{cases} j & \text{if } i = 0 \\ i & \text{if } j = 0 \\ \min\{E[i-1, j-1], E[i-1, j] + 1, E[i, j-1] + 1\} & \text{if } i > 0, j > 0 \text{ and } a_i = b_j \\ \min\{E[i-1, j-1] + 1, E[i-1, j] + 1, E[i, j-1] + 1\} & \text{if } i > 0, j > 0 \text{ and } a_i \neq b_j \end{cases}$$

	0	s	n	o	w	y
0	0	1	2	3	4	5
s	1	→ 0	1	2	3	4
u	2	1	1	2	3	4
n	3	2	1	2	3	4
n	4	3	2	2	3	4
y	5	4	3	3	3	3

Longest increasing subsequence

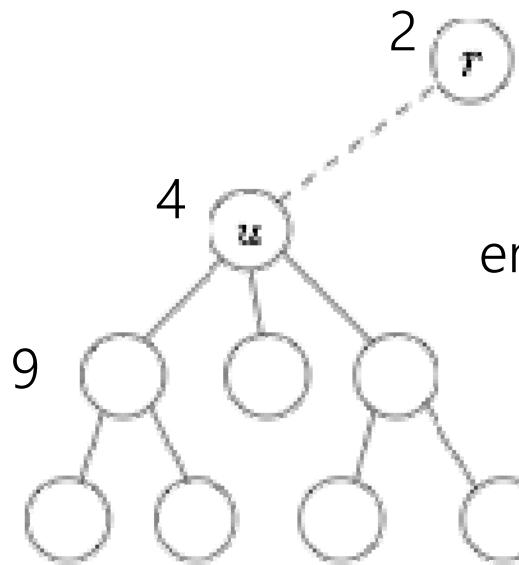


$L[j]$: length of the longest increasing subsequence in $A[1..j]$

$$L(j) = 1 + \max\{L(i) : (i, j) \in E\}.$$

Solution: $L[n]$

Planning a company party



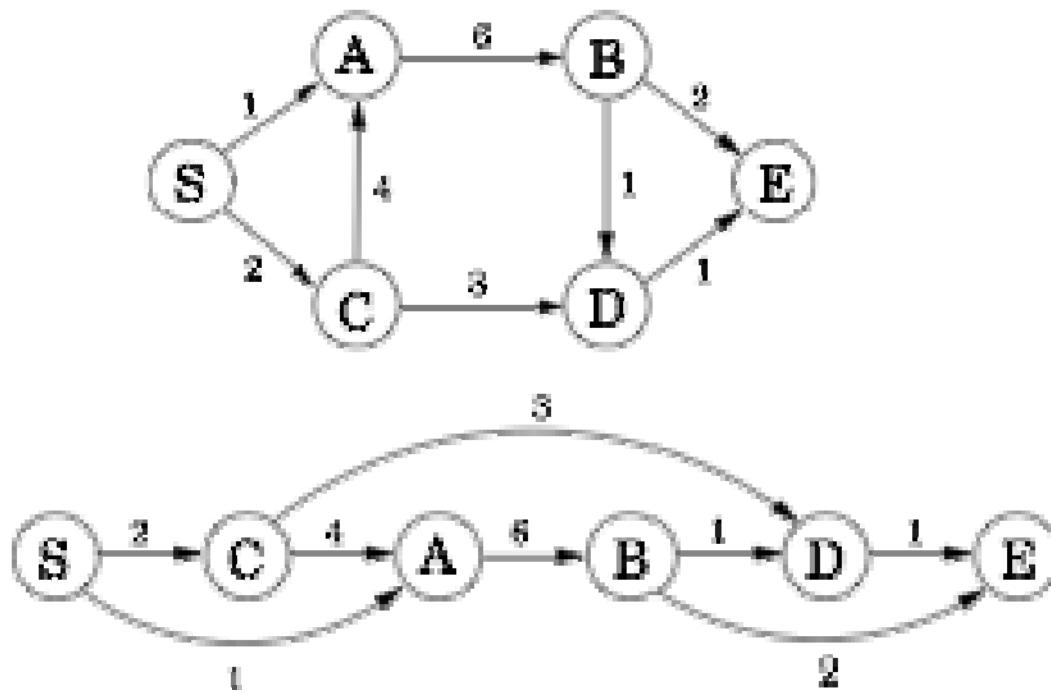
We like convivial friends.

And we will not invite both the employee and his/her direct supervisor.

How to maximize the conviviality?

$$C[u] = \max \{ c[u] + \sum_{\text{grandchildren } w \text{ of } u} C[w], \sum_{\text{children } w \text{ of } u} C[w] \}$$

Single-source longest path in DAG

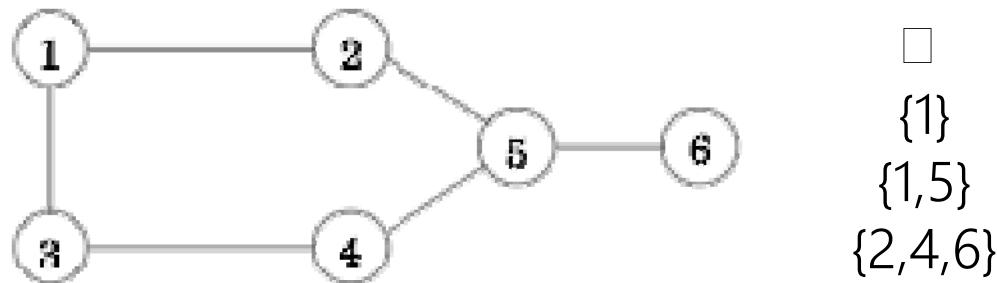


$$D[s] = 0$$

$$D[v] = \max_{(u,v) \in E} \{D[u] + w(u,v)\}$$

Independent set

An *independent set* of graph $G = (V,E)$ if there are no edges between them.

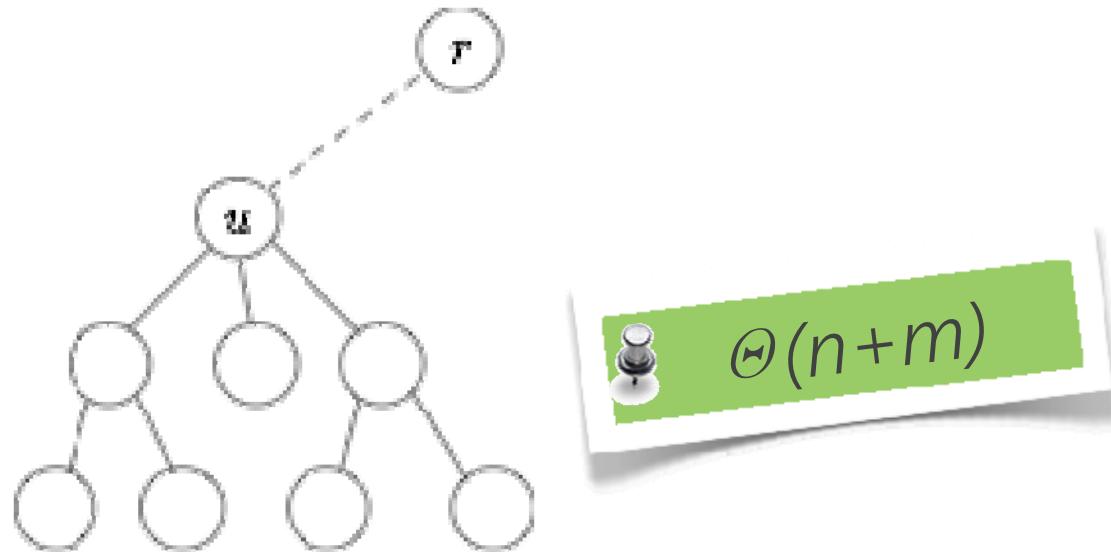


Problem: IndSetInTree

Input: A tree $T=(V,E)$

Output: The size of Maximum independent set in T

Independent set of tree



$I(u)$: size of largest independent set of subtree hanging from u

$$I(u) = \max \left\{ 1 + \sum_{\text{grandchildren } w \text{ of } u} I(w), \sum_{\text{children } w \text{ of } u} I(w) \right\}.$$

Subset sum

	0	1	2	3	4	5	6
L[i,j]	0	0	0	0	0	0	0
	3	1	0	0	1	0	0
	5	1	0	0	1	0	1
	7	1	0	0	1	0	1
	2	1	0	1	1	0	1
	3	1	0	1	1	0	1

Conclusion

- What is Dynamic Programming
- How to write DP paradigm
 - Knapsack problem
 - Longest common subsequence
 - Matrix chain multiplication
- When to apply Dynamic Programming
- Practice

Lecture 11 Network flow

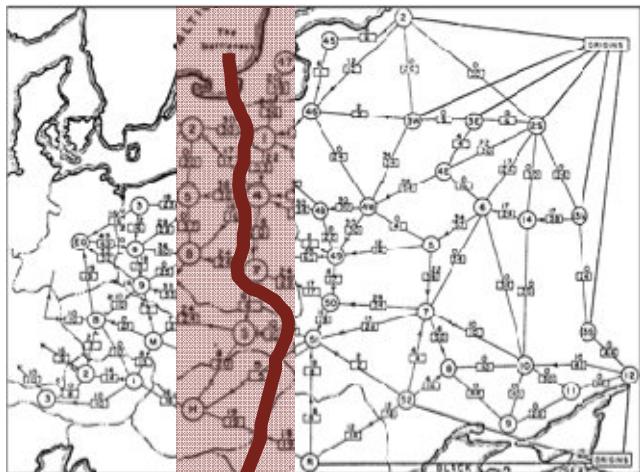
- Max-flow and Min-cut
- Ford-Fulkerson method
- Applications

Roadmap

- Max-flow and Min-cut
- Ford-Fulkerson methods
- Applications

History

First formulated in 1954 by T.E. Harris as a simplified model of *Soviet Railway traffic flow*.



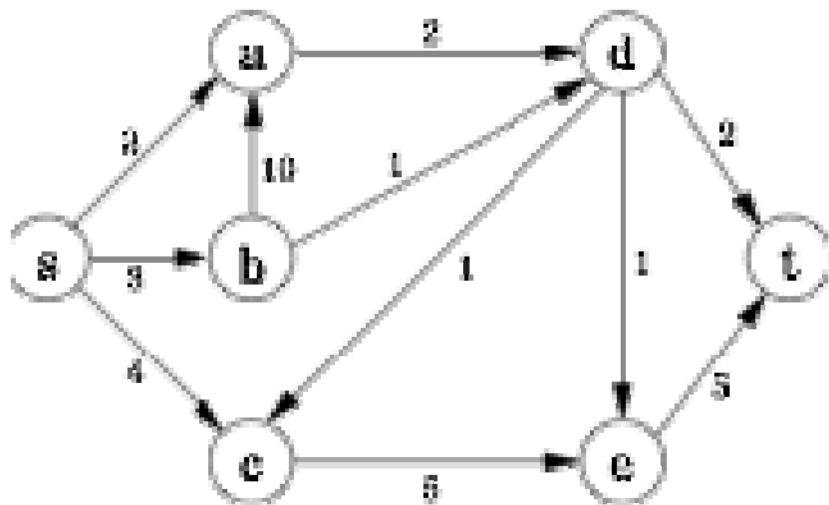
rail network connecting Soviet Union
with Eastern European countries
(map declassified by Pentagon in 1999)

In 1955, L. Ford and D. Fulkerson created the first known algorithm --- *Ford-Fulkerson algorithm*.

Over the years, various improved solutions to the maximum flow problem were discovered:
Edmonds and Karp
Dinitz (Dinic)
Goldberg and Tarjan

.....

Flow Networks



A *flow network* is

1. a directed graph $G < V, E >$,
2. Weights on edges denote the *capacities*.
3. Two distinguished vertex: s and t , denoting *source* and *sink*.

Definition: st-Flow A function f :

$V \times V \rightarrow R$ is a *st-flow (flow)* if satisfying:

$$\forall u, v \in V. f(u, v) = -f(v, u).$$

$$\forall u, v \in V. f(u, v) \leq c(u, v).$$

$$\forall u \in V \setminus \{s, t\}. \sum_{v \in V} f(u, v) = 0.$$

$$\forall v \in V. f(v, v) = 0.$$

The *value* of a flow f :

$$|f| = f(s, V) = \sum_{v \in V} f(s, v)$$

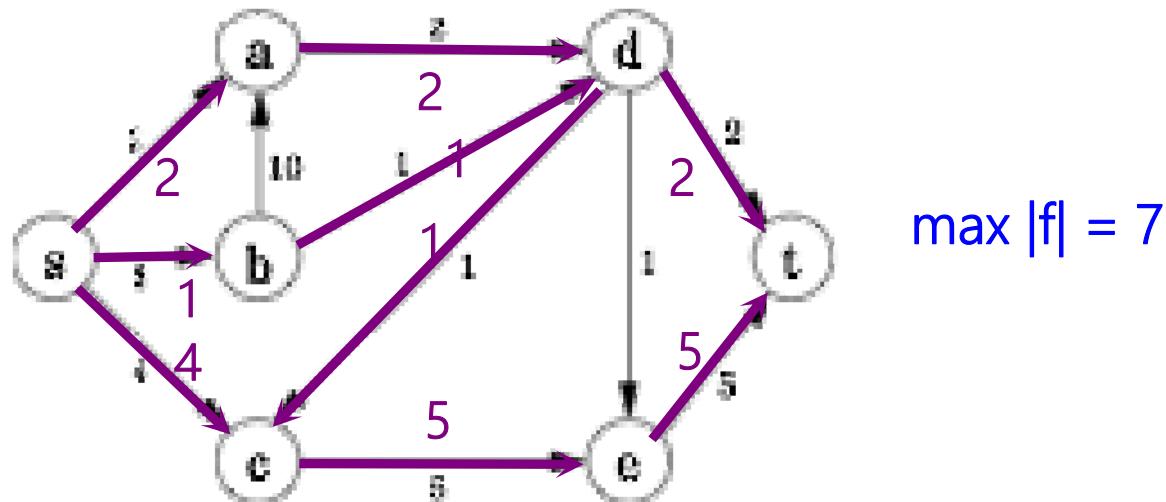
$$|f| \geq 0$$

Maximum flow

Problem: Maxflow

Input: A network $G(V, E)$, s, t , and capacity function c .

Output: The flow with maximum value.



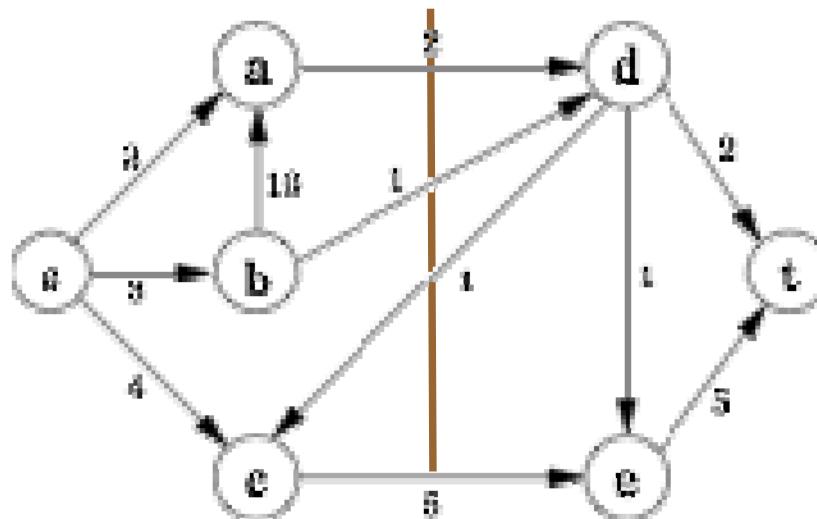
Minimum Cut

Definition: Cut

A **cut** $\{S, T\}$ is a partition of the vertex set V into two subsets S and T such that $s \in S$ and $t \in T$.

The **capacity** of the cut $\{S, T\}$, denoted by $c(S, T)$, is

$$c(S, T) = \sum_{u \in S, v \in T} c(u, v)$$

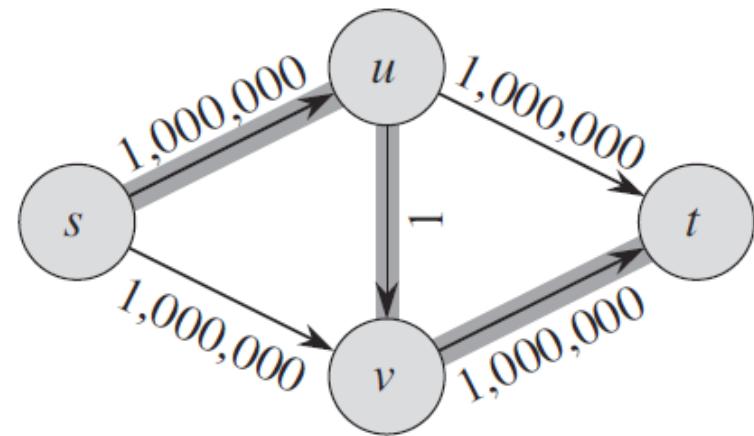
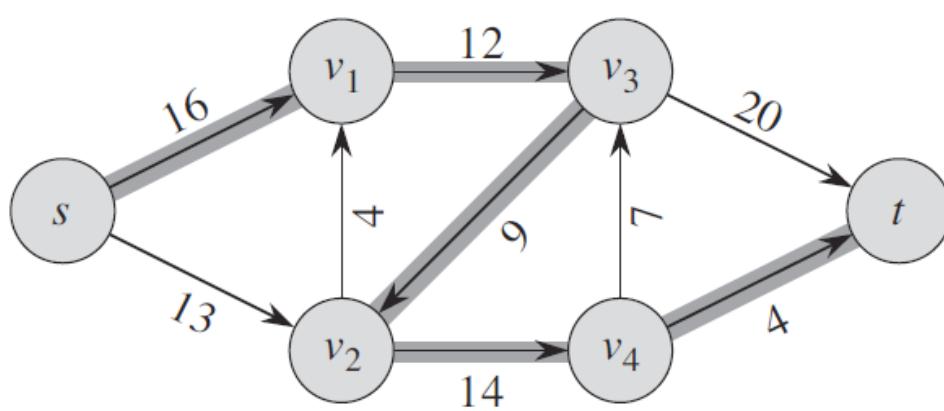


Lemma

For any cut $\{S, T\}$ and a flow f , $|f| = f(S, T) \leq c(S, T)$.

Quiz

Compute the maximum flow and minimum cut
of the following graphs.



- A. 22
- B. 23
- C. 24
- D. 2,000,000

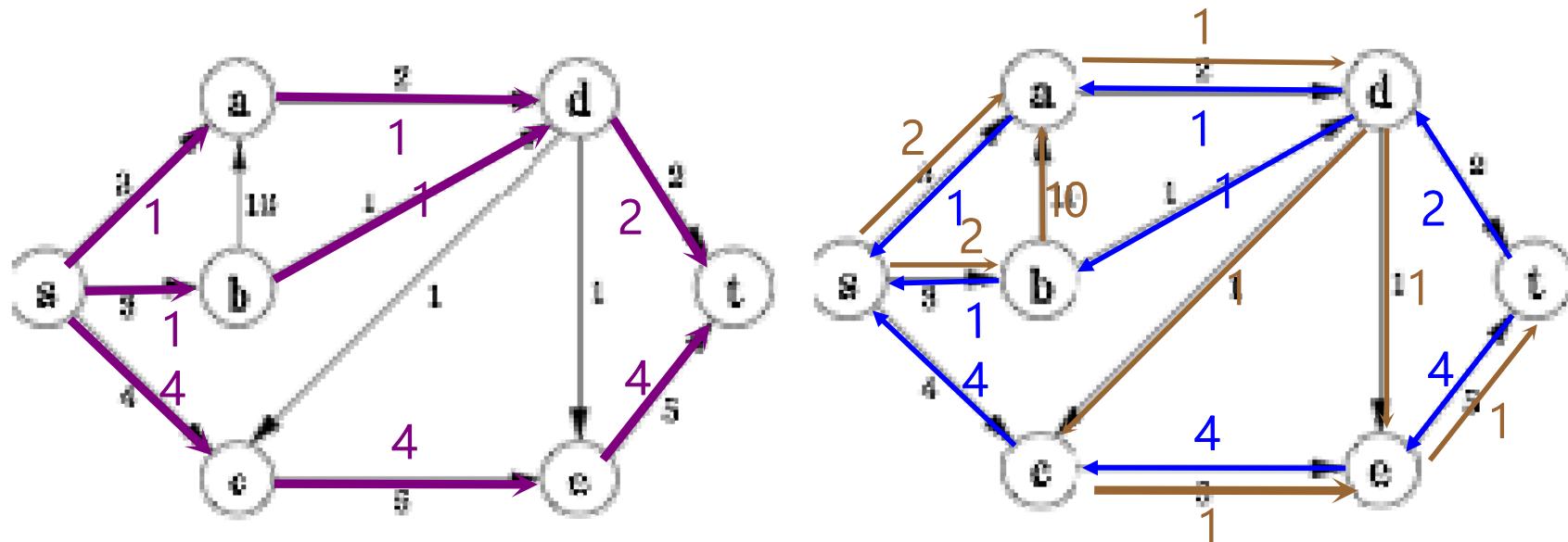
Where are we?

- Max-flow and Min-cut
- Ford-Fulkerson methods
- Applications

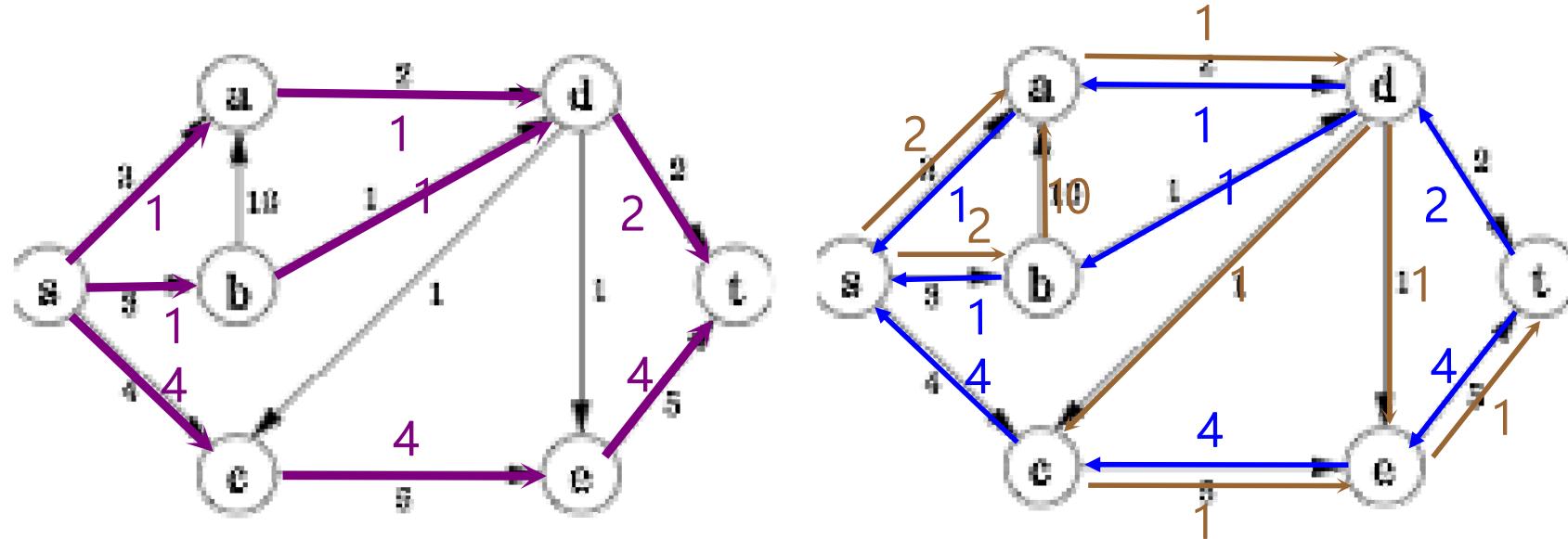
Residual graph

The *residual graph* for the flow f is the directed graph $R = (V, E_f)$, with capacities defined by r and

$$E_f = \{(u,v) \mid r(u,v) > 0\} \text{ where } r(u,v) = c(u,v) - f(u,v)$$



Flow in residual graph



Lemma

Let f be a flow in G and f' the flow in the residual graph R for f .
Then the function $f + f'$ is a flow in G of value

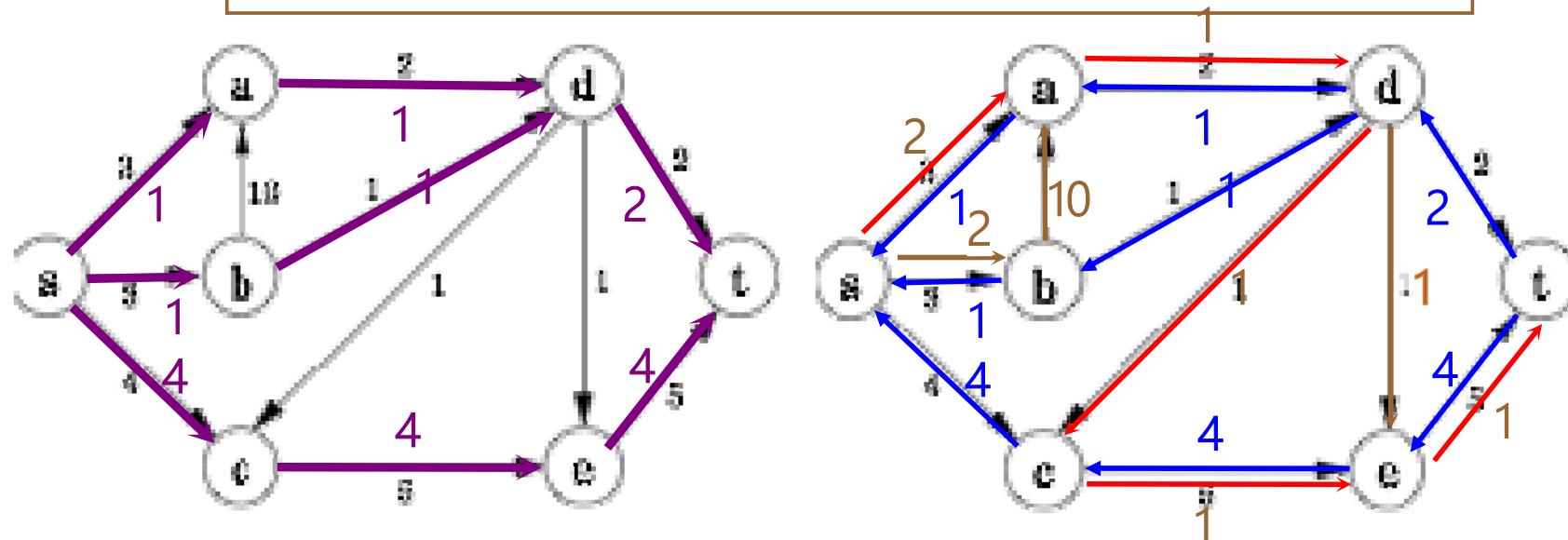
$$|f + f'| = |f| + |f'| > |f|.$$

Augmenting path

Definition: AugmentingPath

Given a flow in G , an *augmenting path* p is a directed path from s to t in the residual graph R .

The *bottleneck capacity* of p is the minimum residual capacity along p . The number of edges in p will be denoted by $|p|$.



Basic Ford-Fulkerson

Algorithm 16.1 Ford-Fulkerson

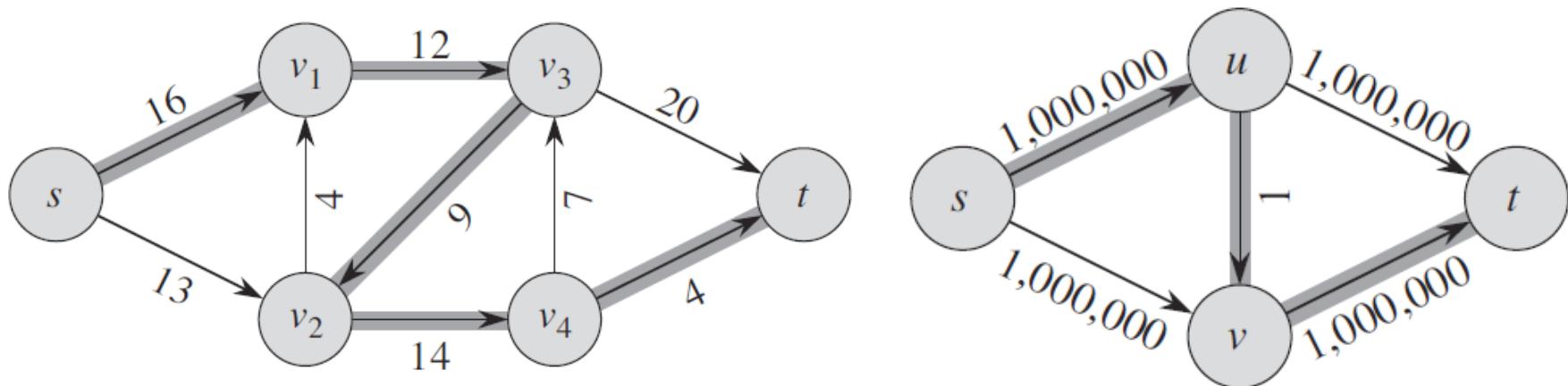
Input: A network (G, s, t, c) .

Output: A flow in G .

1. Initialize the residual graph by setting $R = G$.
2. for each edge $(u, v) \in E$ do $f(u, v) \leftarrow 0$
3. while there is an augmenting path $p = s, \dots, t$ in R do
4. Let Δ be the bottleneck capacity of p
5. for each edge (u, v) in p do $f(u, v) \leftarrow f(u, v) + \Delta$
6. Update the residual graph R
7. end while

Quiz

Apply FF to the following graph to compute the maximum flow.



Questions

- How to compute a min-cut?
- How to find an augmenting path?
- If FF terminates, does it always compute a maxflow?
- Does FF always terminate? If so, after how many augmentations?

Max-flow-Min-cut theorem

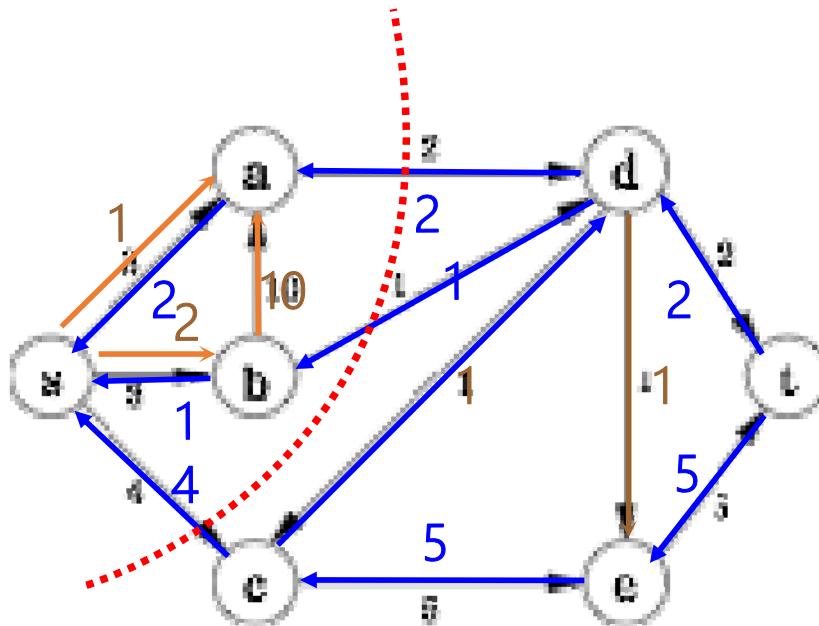
Max-flow-Min-cut theorem

Let (G, s, t, c) be a network and f a flow in G . The following three statements are equivalent:

1. f is a maximum flow in G .

2. There is no augmenting path.
3. There is a cut $\{S, T\}$ with $c(S, T) = |f|$.

Compute Min-cut (S, T)



- Compute the maximum network flow using FF
- There is no augmenting paths in final residual graph G^r .
- Compute $S = \text{set of vertices connected to } s \text{ in } G^r$
- Compute $T = V \setminus S$.

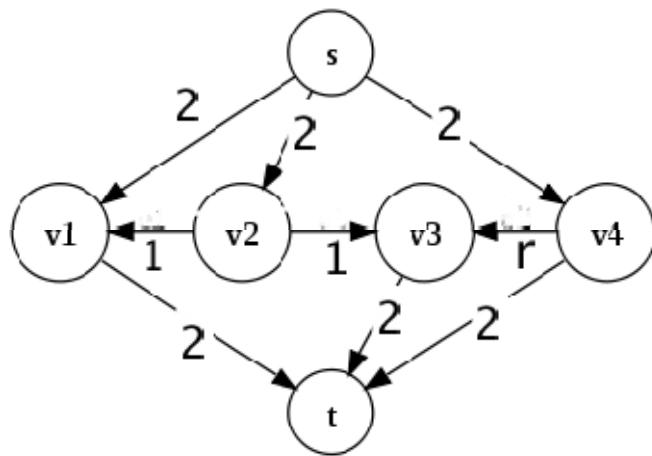
Questions

- How to compute a min-cut? ✓
- How to find an augmenting path? BFS works well. ✓
- If FF terminates, does it always compute a maxflow? By Max-flow-Min-cut theorem. ✓
- Does FF always terminate? If so, after how many augmentations?

Ford-Fulkerson method

For integer capacities, $\Theta(m|f^*|)$, where f^* is the maximum flow.

Ford-Fulkerson may **NOT** halt if the capacities are irrational, or may not converge to a value that is maximum.



$$r = (\sqrt{5} - 1)/2$$

$$p_0 = \{s, v_2, v_3, t\}$$

$$p_1 = \{s, v_4, v_3, v_2, v_1, t\}$$

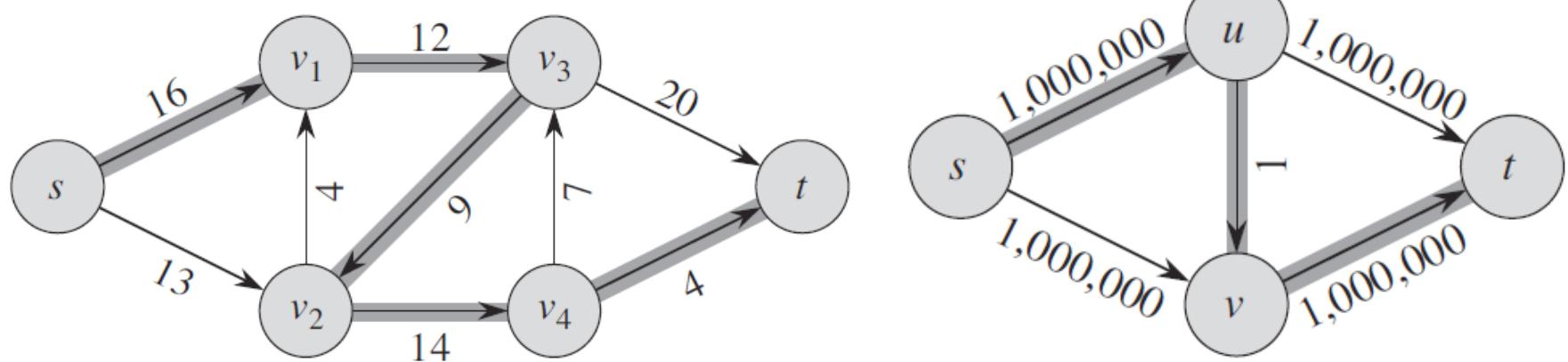
$$p_2 = \{s, v_2, v_3, v_4, t\}$$

$$p_3 = \{s, v_1, v_2, v_3, t\}$$

$$p_0, p_1, p_2, p_1, p_3, p_1, p_2, \dots$$

Improvements by Edmonds and Karp

- Maximum capacity augmentation: $O(V^2 E \log c^*)$
- Shortest path augmentation: $O(VE^2)$



Solutions

Method	Complexity
Ford–Fulkerson	$O(E \max f)$
Edmonds–Karp	$O(V^2 E \log c^*)$, $O(VE^2)$
Dinitz blocking flow	$O(V^2 E)$
General push-relabel maximum flow	$O(V^2 E)$
Push-relabel with FIFO vertex selection rule	$O(V^3)$
Dinitz blocking flow with dynamic trees	$O(VE \log(V))$
Push-relabel with dynamic trees	$O(VE \log(V^2/E))$

Where are we?

- Max-flow and Min-cut
- Ford-Fulkerson methods
- Applications
 - maximum bipartite matching
 - edge disjoint path
 - vertex disjoint path
 - network connectivity

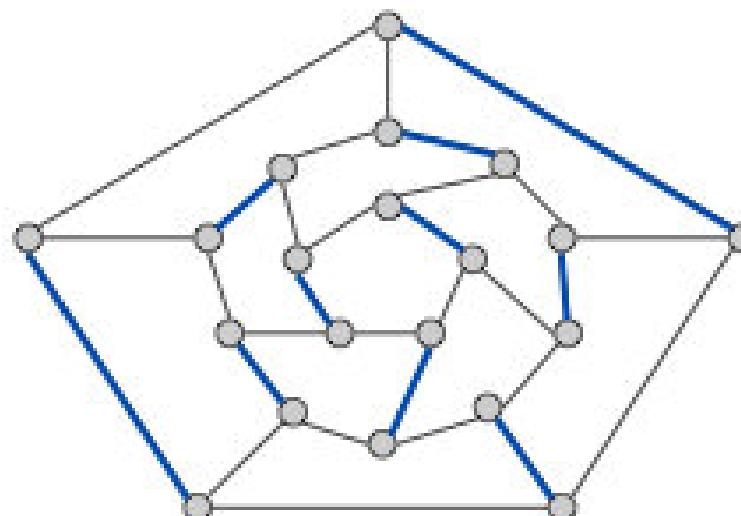
App: Bipartite match

Problem: MaxMatch

Input: An undirected graph $G(V, E)$

A *matching* is a subset $M \subseteq E$ such that NO two edges in M have a vertex in common.

Output: finding a max cardinality matching



App: Bipartite match

N students apply for N jobs.



Each gets several offers.



Is there a way to match all students to jobs?



bipartite matching problem

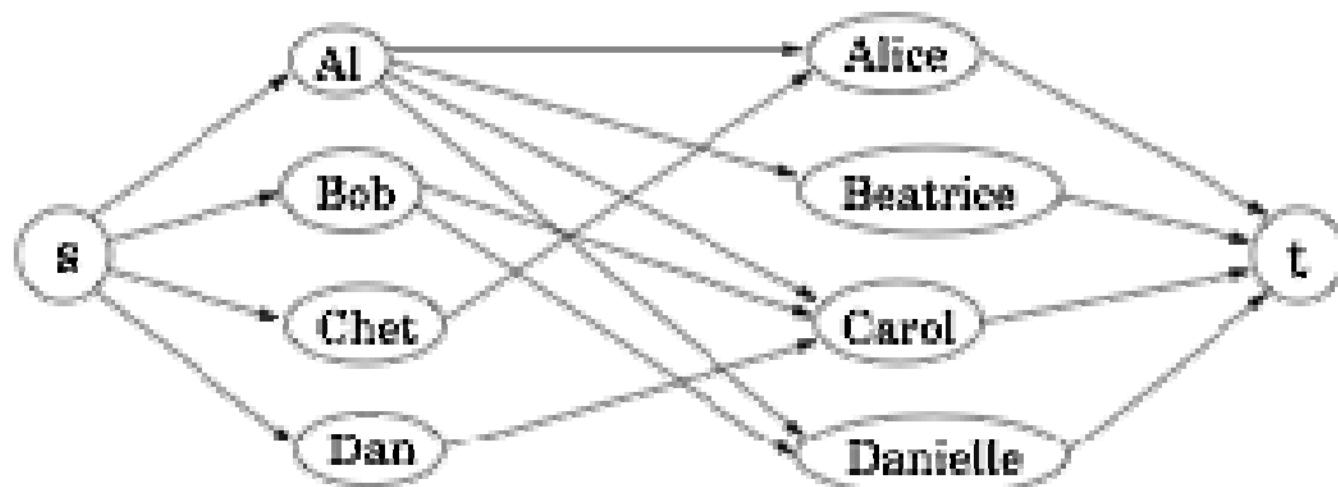
1	Alice	6	Adobe
	Adobe		Alice
	Amazon		Bob
	Google		Carol
2	Bob	7	Amazon
	Adobe		Alice
	Amazon		Bob
3	Carol		Dave
	Adobe		Eliza
	Facebook	8	Facebook
	Google		Carol
4	Dave	9	Google
	Amazon		Alice
	Yahoo		Carol
5	Eliza	10	Yahoo
	Amazon		Dave
	Yahoo		Eliza

App: Bipartite match

Problem: BipartiteMatch

Input: A bipartite graph $G(V, E)$

Output: The largest matchings in G .



Stable matching

An unmatched pair m-w is **unstable** if man m and woman w prefer each other to current partners.

A stable matching is a perfect matching without unstable pairs.

	favorite ↓	least favorite ↑	
	1 st	2 nd	3 rd
Xavier	Amy	Bertha	Clare
Yancey	Bertha	Amy	Clare
Zeus	Amy	Bertha	Clare

Men's Preference Profile

	favorite ↓	least favorite ↑	
	1 st	2 nd	3 rd
Amy	Yancey	Xavier	Zeus
Bertha	Xavier	Yancey	Zeus
Clare	Xavier	Yancey	Zeus

Women's Preference Profile

X-C, Y-B, Z-A
X-A, Y-B, Z-C

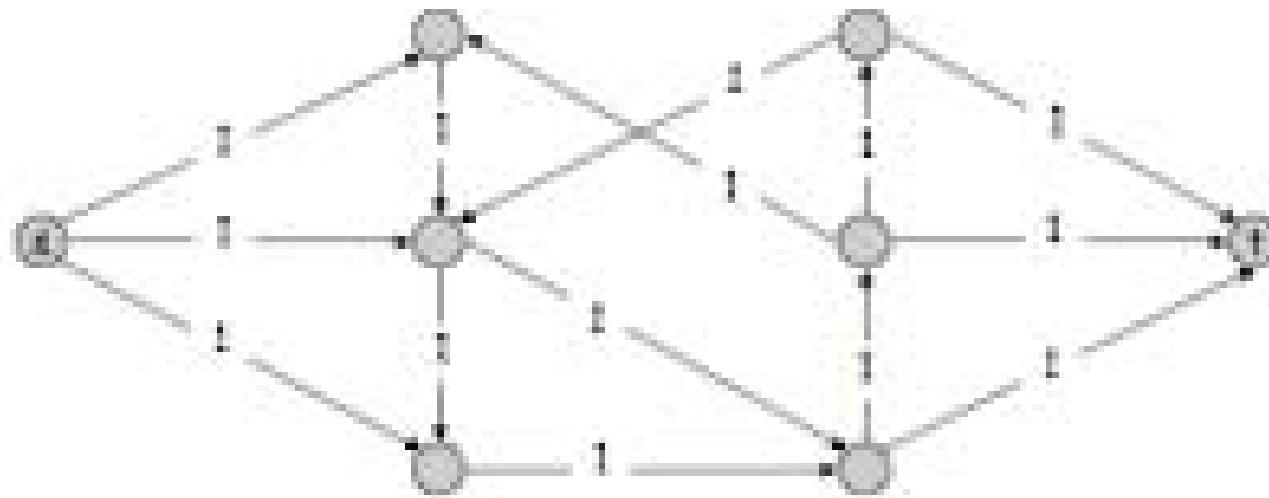
unstable
stable

App: Edge-Disjoint path

Problem: EdgeDisjointPath

Input: A graph $G(V, E)$, and two vertex s, t

Output: The max number of edge disjoint paths
between s and t

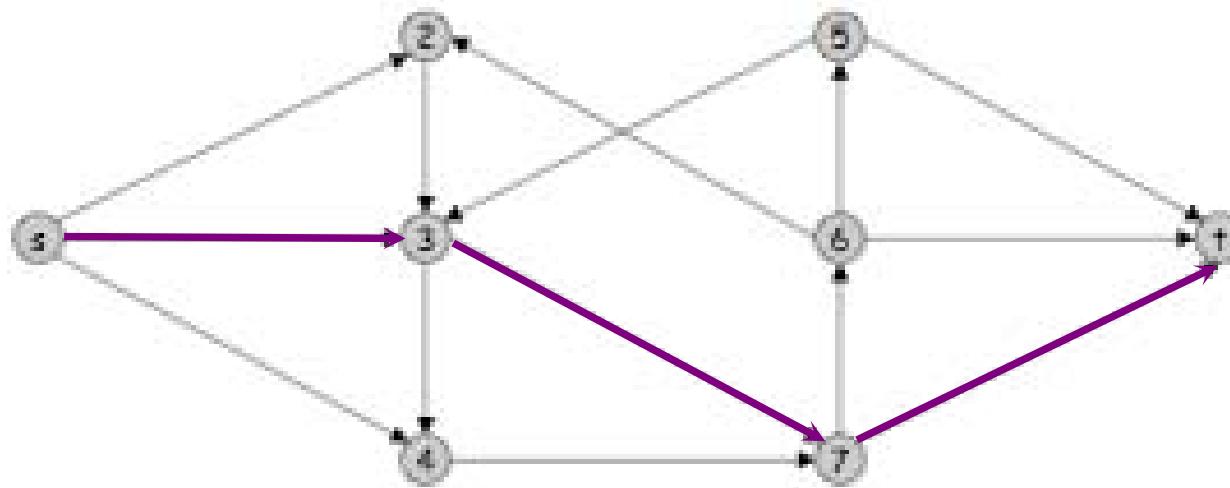


App: Vertex-Disjoint path

Problem: VertexDisjointPath

Input: A graph $G(V, E)$, and two vertex s, t

Output: The max number of vertex disjoint paths
between s and t

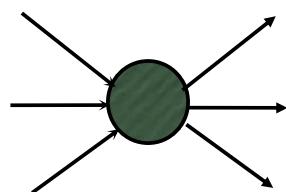


App: Vertex-Disjoint path

Problem: VertexDisjointPath

Input: A graph $G(V, E)$, and two vertex s, t

Output: The max number of vertex disjoint paths between s and t

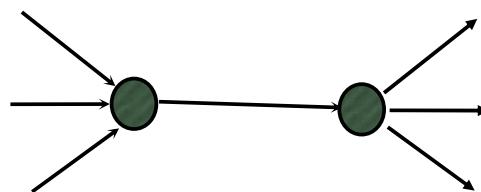


App: Vertex-Disjoint path

Problem: VertexDisjointPath

Input: A graph $G(V, E)$, and two vertex s, t

Output: The max number of vertex disjoint paths between s and t



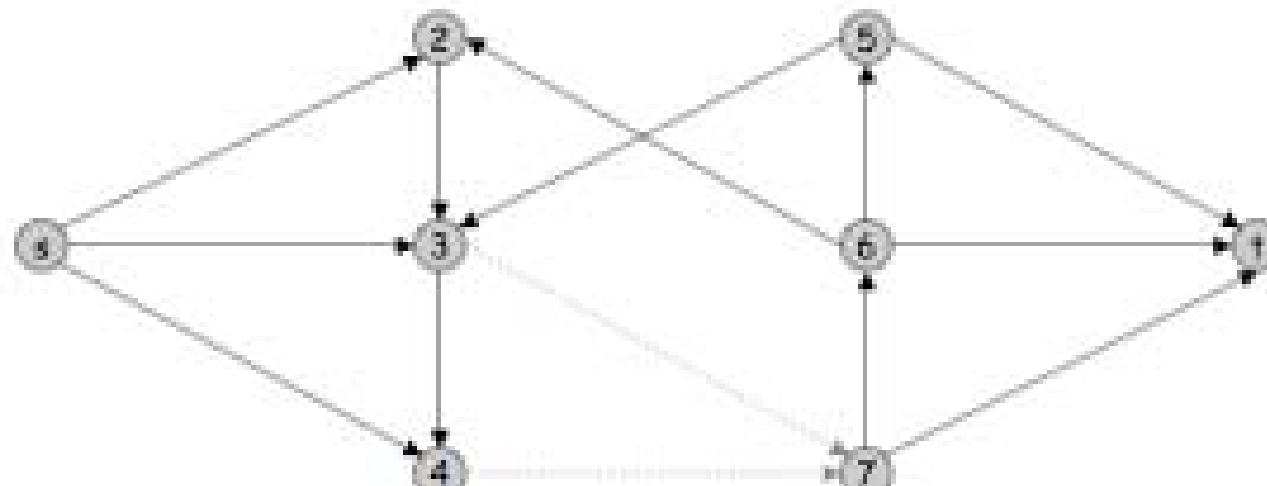
Reduce to Edge Disjoint Path problem

App: Network Connectivity

Problem: NetworkConnectivity

Input: A graph $G(V, E)$, and two vertex s, t

Output: The min number of edge whose removal disconnects t from s



Find the min-cut

Conclusion

- **Mincut problem.** Find an st-cut of minimum capacity.
Maxflow problem. Find an st-flow of maximum value.
Duality. Value of the maxflow = capacity of mincut.
- Proven *successful approaches*.
Ford-Fulkerson (various augmenting-path strategies).
Preflow-push (various versions).
- Open research challenges.
Practice: solve real-word maxflow/mincut problems
in linear time.
Theory: prove it for worst-case inputs.

Lecture 12 P and NP

- Introduction to intractability
- Class P and NP
- Class NPC (NP-complete)

Roadmap

- Introduction to intractability
- Class P and NP
- Class NPC

Questions on Computation

- What is a general-purpose computer?
- Are there limits on the power of digital computers?



David Hilbert



Kurt Gödel



Alan Turing

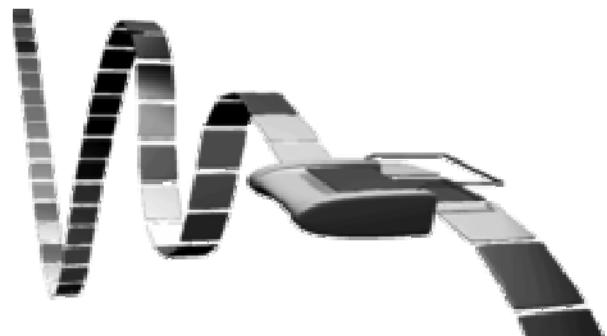


Alonzo Church



John von Neumann

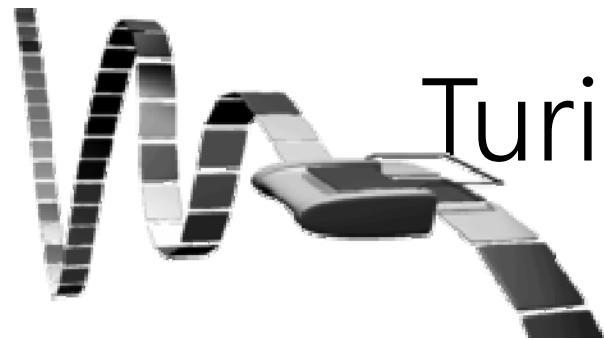
Turing machine



Artistic representation of a Turing Machine
(Source from *Wikipedia*)



Alan Turing,
23 June 1912 – 7 June 1954
(Source from *Wikipedia*)



Turing machine

Definition: TuringMachine

A (one-tape) *Turing machine* is a 6-tuple $\langle Q, \Gamma, b, \delta, q_0, F \rangle$ where

- Q : is a finite, non-empty set of states
- Γ : is a finite, non-empty set of the tape alphabet/symbols
- b : is the blank symbol
- q_0 : is the initial state
- F : is the set of final or accepting states.
- δ : is the transition function $Q \setminus F \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$

QUIZ

Construct a Turing machine to solve the Palindrome problem.

□	1	0	1	0	1	b	b	b	b	...
---	---	---	---	---	---	---	---	---	---	-----

Palindrome



$$M = \langle \{s, f, q_0, q_1, q_0', q_1', r, a\}, \{0, 1\}, b, \delta, s, \{r, a\} \rangle$$

$s, b \rightarrow a, b, R$ $q_0, 0 \rightarrow q_0, 0, R$ $q_1, 0 \rightarrow q_1, 0, R$

$s, 0 \rightarrow q_0, b, R$ $q_0, 1 \rightarrow q_0, 1, R$ $q_1, 1 \rightarrow q_1, 1, R$

$s, 1 \rightarrow q_1, b, R$ $q_0, b \rightarrow q_0', b, L$ $q_1, b \rightarrow q_1', b, L$
 $q_0', 0 \rightarrow f, b, L$ $q_1', 0 \rightarrow r, _, _$

$f, 0 \rightarrow f, 0, L$ $q_0', 1 \rightarrow r, _, _$ $q_1', 1 \rightarrow f, b, L$

$f, 1 \rightarrow f, 1, L$

$f, b \rightarrow s, b, R$

Nondeterministic Turing machine

$$M = \langle \{s, f, q_0, q_1, r, a\}, \{0, 1\}, b, \delta, s, \{r, a\} \rangle$$

$s, b \rightarrow a, b, R$	$q_0, 0 \rightarrow q_0, 0, R$	$q_1, 0 \rightarrow q_1, 0, R$
$s, 0 \rightarrow q_0, b, R$	$q_0, 1 \rightarrow q_0, 1, R$	$q_1, 1 \rightarrow q_1, 1, R$
$s, 1 \rightarrow q_1, b, R$	$q_0, 0 \rightarrow f, b, L$ $q_0, 1 \rightarrow r, _, _$	$q_1, 0 \rightarrow r, _, _$ $q_1, 1 \rightarrow f, b, L$ $q_1, b \rightarrow r, _, _$
$f, 0 \rightarrow f, 0, L$	$q_0, b \rightarrow r, _, _$	
$f, 1 \rightarrow f, 0, L$		
$f, b \rightarrow s, b, R$		

Church-Turing Thesis

ChurchTuringThesis.

Every *effectively calculable* function is a *computable function*. (can be computed by a turing machine.)

Remark. "Thesis" is not a mathematical theorem because it's a statement about the physical world and not subject to proof.

Models equivalent: simulation.

- Android simulator on iPhone.
- iPhone simulator on Android.

Questions on Algorithms

Q: Which algorithms are useful in practice?

A: Useful in practice ("efficient") = polynomial time for all inputs.

(efficient) Sorting N items takes $N \log N$ compares using mergesort.

(inefficient) Finding best TSP tour on N points takes $N !$ steps using brute search.

Exponential growth

Suppose you have a giant parallel computing device...

With as many processors as electrons in the universe...
And each processor has power of today's supercomputers...
And each processor works for the life of the universe...

quantity	value
electrons in universe †	10^{79}
supercomputer instructions per second †	10^{13}
age of universe in seconds †	10^{17}

† estimated

Will not help solve 1,000 city TSP problem via brute force.

$$1000! >> 10^{1000} >> 10^{\{79+13+17\}}$$

Exponential growth



Sessa, the inventor of chess
(Source from *Wikipedia*)

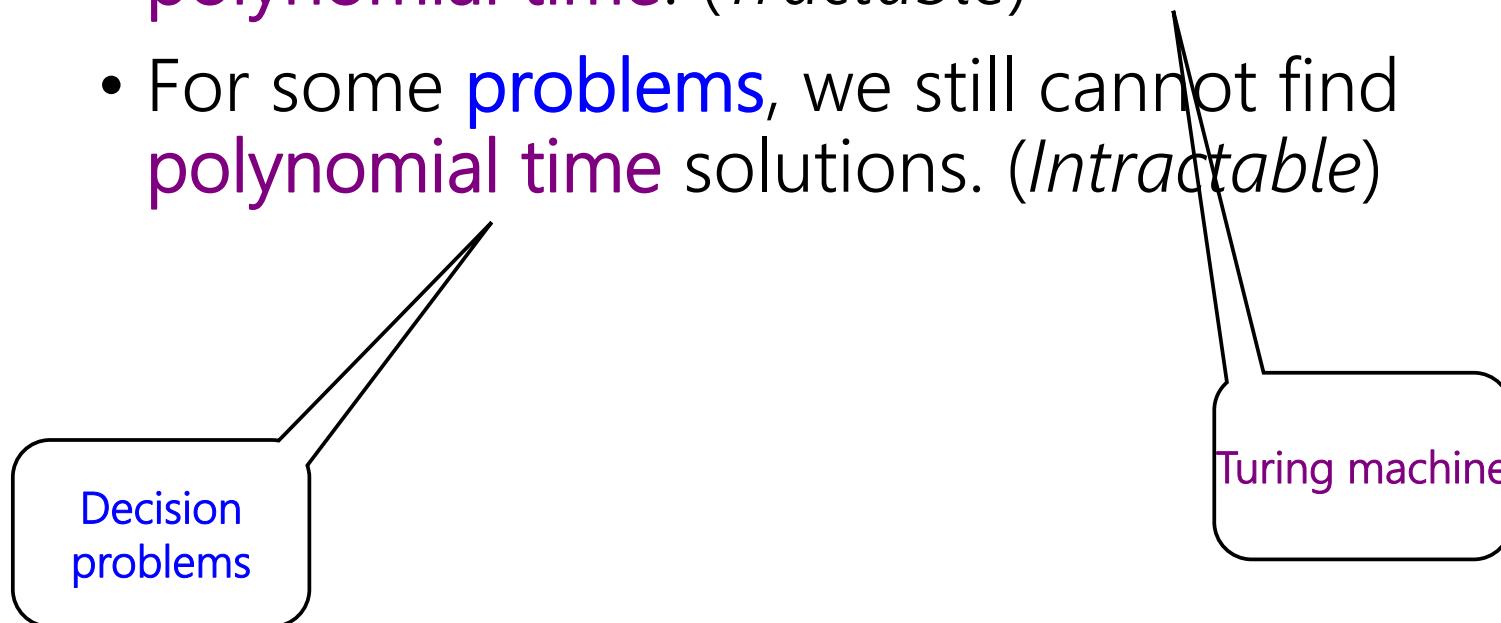


(Source from *Wikipedia*)

$$2^{64} - 1 = 18,446,744,073,709,551,615$$

Tractability

- Some **problems** can be solved in **polynomial time**. (*Tractable*)
- For some **problems**, we still cannot find **polynomial time** solutions. (*Intractable*)



Decision problems

Decision problems are those whose solutions have only two possible outcomes: Yes or No.

Decision problems v.s. *optimization problem*

Decision v.s. Optimization

DecisionProblem: Coloring

Input: An undirected graph $G = (V, E)$ and a positive integer k

Output: Is G k -colorable?

OptimizationProblem: Coloring

Input: An undirected graph $G = (V, E)$

Output: The chromatic number of G

Decision v.s. Optimization

DecisionProblem: Clique

Input: An undirected graph $G = (V, E)$ and a positive integer k

Output: Does G have a clique of size k ?

OptimizationProblem: MaxClique

Input: An undirected graph $G = (V, E)$

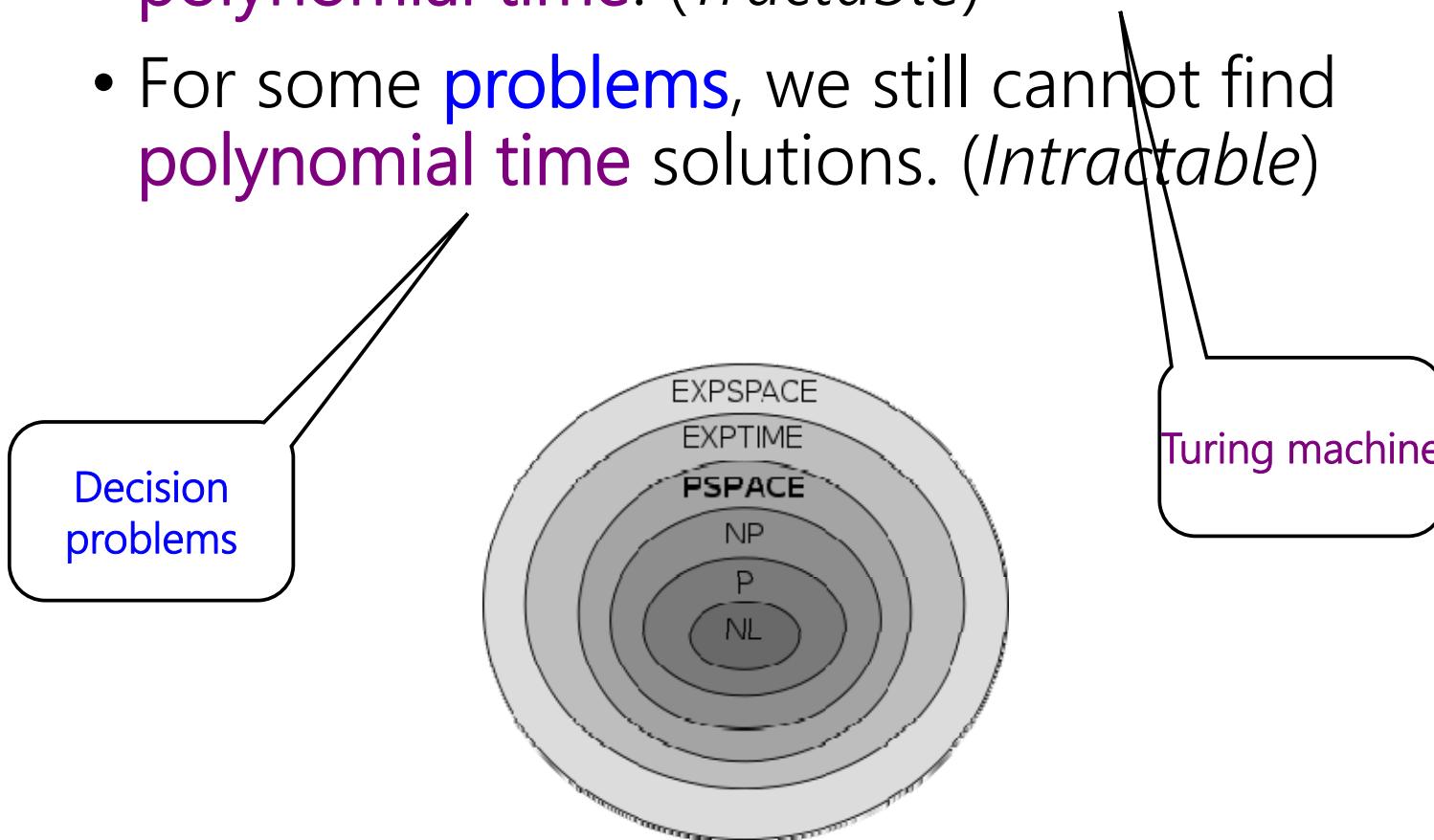
Output: The maximum clique size of G



Given an algorithm for *decision problem*, how to solve its corresponding *optimization problem*?

Efficiency

- Some **problems** can be solved in **polynomial time**. (*Tractable*)
- For some **problems**, we still cannot find **polynomial time** solutions. (*Intractable*)



Where are we?

- Introduction to intractability
- Class P and NP
- Class NPC

Class P

Definition: Class P

A problem π is in class **P** if and only if there exists a *deterministic* Turing machine M, such that

- M runs in **polynomial time** on all inputs
- For all instance with *Yes* answer, M terminates with *accept* state
- For all instance with *No* answer, M terminates with *reject* state

Class NP

Definition: ClassNP

A problem π is in class **NP** if and only if there exists a *nondeterministic* Turing machine M , such that

- M runs in **polynomial time** on all inputs
- For all instance with **Yes** answer, there exists a path of M that terminates with **accept** state
- For all instance with **No** answer, all paths terminates with **reject** state

Guess a witness --> verify the witness is valid

Examples

DecisionProblem: 2-Coloring

Input: A graph $G = \langle V, E \rangle$

Output: Is G 2-colorable?

DecisionProblem: 3-Coloring

Input: A graph $G = \langle V, E \rangle$

Output: Is G 3-colorable?

Examples

DecisionProblem: ShortestPath

Input: A weighted graph $G = \langle V, E \rangle$, vertex s, t and a number k

Output: Does the shortest path between s and t have length k ?

DecisionProblem: LongestPath

Input: A weighted graph $G = \langle V, E \rangle$, vertex s, t and a number k

Output: Does the longest path between s and t have length k ?

Examples

DecisionProblem: EulerianTour

Input: A graph $G = \langle V, E \rangle$

Output: Does G have a Eulerian tour?

DecisionProblem: HamiltonianTour

Input: A graph $G = \langle V, E \rangle$

Output: Does G have a Hamiltonian tour?

P is in NP

Lemma. $P \subseteq NP$.



$NP = P?$ or $P \subseteq NP?$

Class co-P and co-NP

The *complement* of a decision problem is the decision problem resulting from reversing the **yes** and **no** answers.

If problem π is in P (NP), then its complement is in co-P (co-NP).

$$co\text{-}P = P$$

$$co\text{-}NP = NP?$$

Examples

DecisionProblem:2-Coloring

Input: A graph $G = \langle V, E \rangle$

Output: Is G 2-colorable?

DecisionProblem: Co-2-Coloring

Input: A graph $G = \langle V, E \rangle$

Output: Is G not 2-colorable?

DecisionProblem:3-Coloring

Input: A graph $G = \langle V, E \rangle$

Output: Is G 3-colorable?

DecisionProblem: Co-3-Coloring

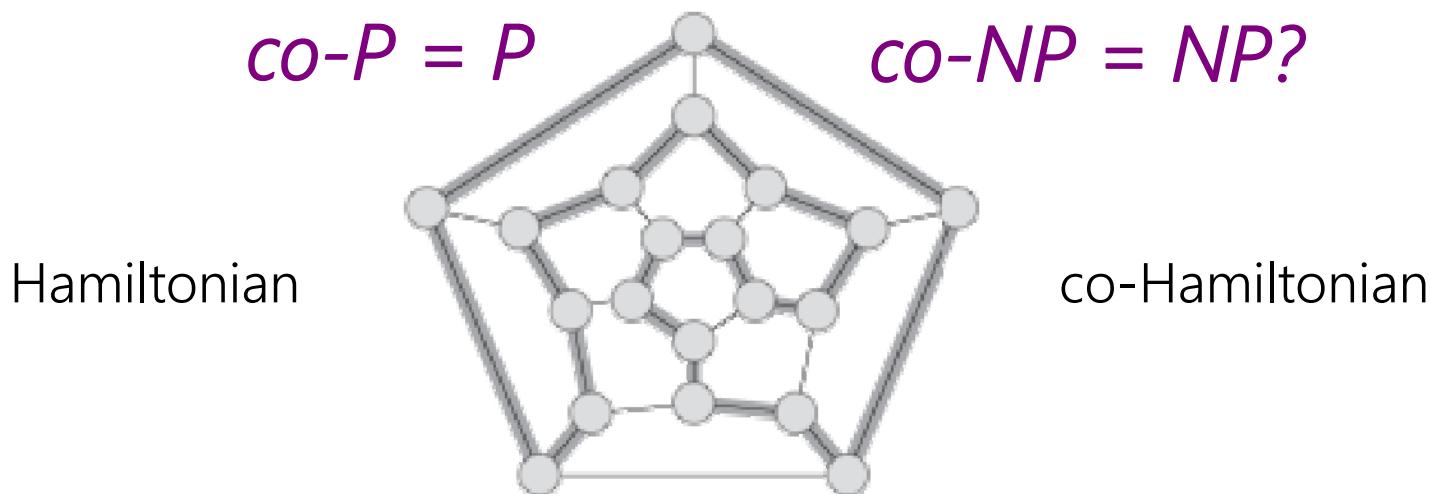
Input: A graph $G = \langle V, E \rangle$

Output: Is G not 3-colorable?

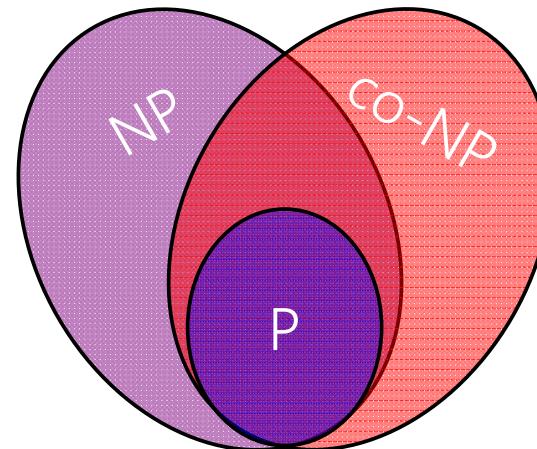
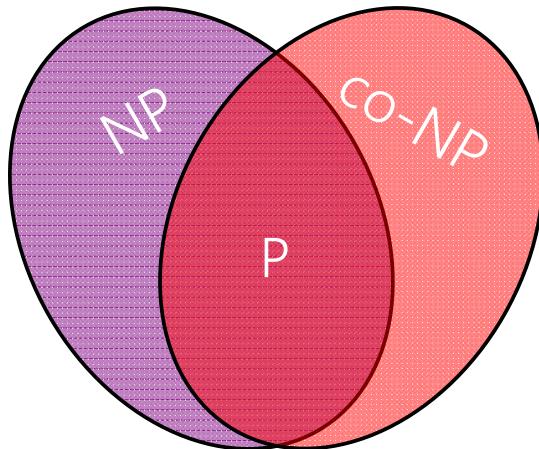
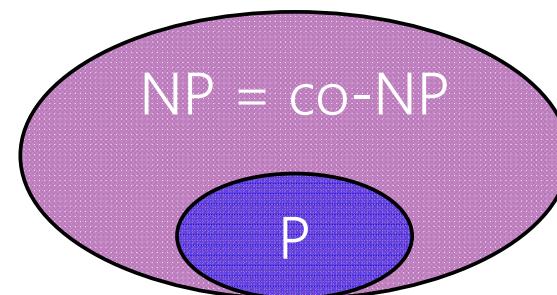
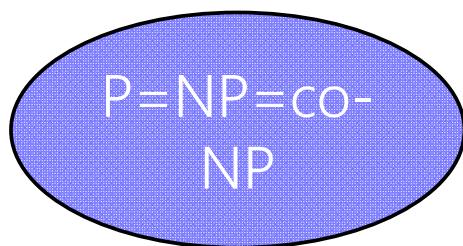
Class co-P and co-NP

The *complement* of a decision problem is the decision problem resulting from reversing the **yes** and **no** answers.

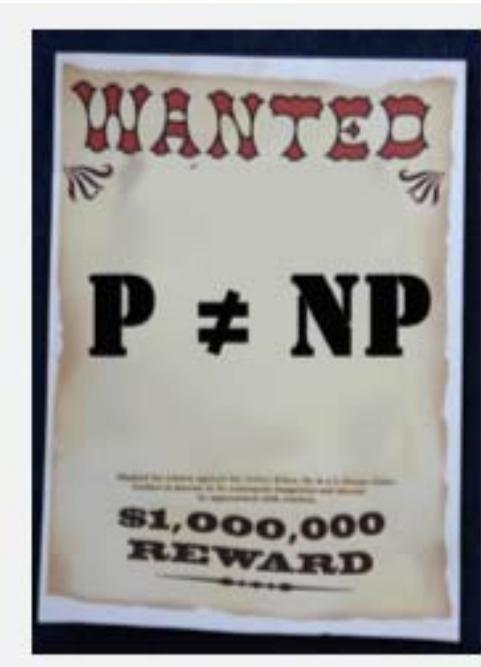
If problem π is in P (NP), then its complement is in co-P (co-NP).



Possible pictures



P = NP ?



Clay Mathematics Institute
Dedicated to increasing and disseminating mathematical knowledge

HOME | ABOUT CMI | PROGRAMS | NEWS & EVENTS | AWARDS | SCHOLARS | PUBLICATIONS

Millennium Problems

In order to celebrate mathematics in the new millennium, The Clay Mathematics Institute of Cambridge, Massachusetts (CMI) has named seven *Prize Problems*. The Scientific Advisory Board of CMI selected these problems, focusing on important classic questions that have resisted solution over the years. The Board of Directors of CMI designated a \$7 million prize fund for the solution to these problems, with \$1 million allocated to each. During the Millennium Meeting held on May 24, 2000 at the Collège de France, Timothy Gowers presented a lecture entitled *The Importance of Mathematics*, aimed for the general public, while John Tate and Michael Atiyah spoke on the problems. The CMI invited specialists to formulate each problem.

- Birch and Swinnerton-Dyer Conjecture
- Hodge Conjecture
- Navier-Stokes Equations
- Exotic RP³
- Poincaré Conjecture
- Riemann Hypothesis
- Yang-Mills Theory
- Bures
- Millennium Meeting Videos

Where are we?

- Introduction to intractability
- Class P and NP
- Class NPC

Reduction

" Give me a lever long enough and a fulcrum on which to place it, **and I shall move the world.** "



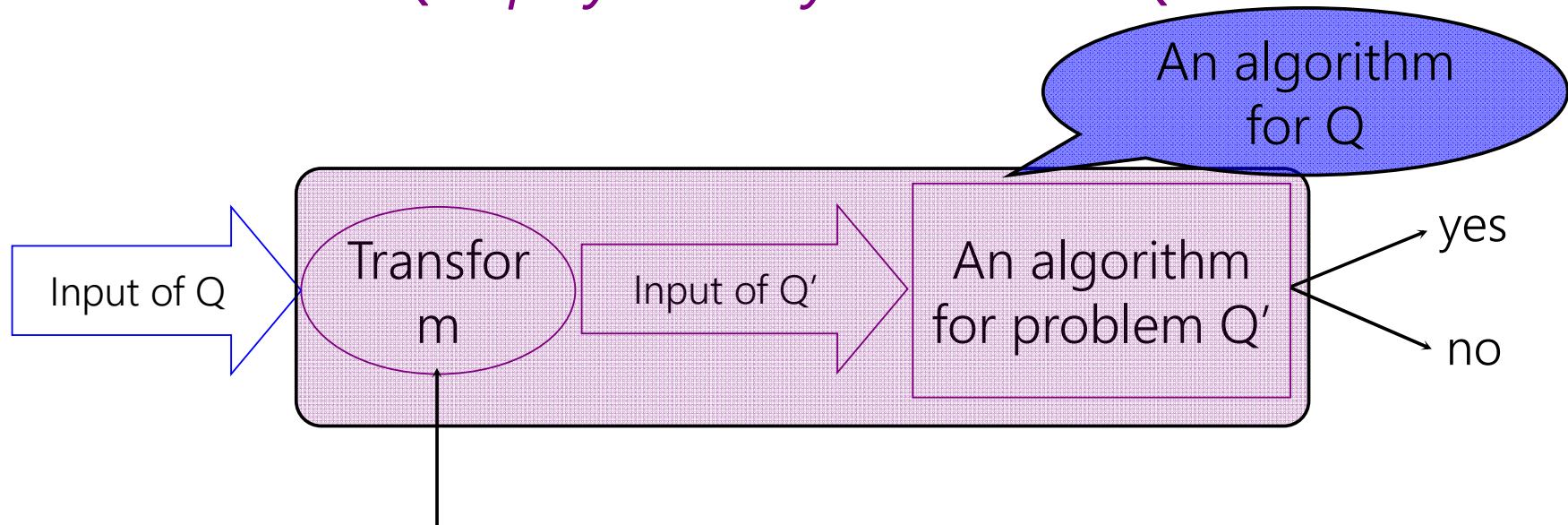
-- *Archimedes*

I shall move the world reduce to ∞

Give me a lever long enough and a fulcrum on which to place it

Polynomial-time reduction

Q is polynomially reduced to Q'



Polynomial-time

$Q \propto_p Q'$: Q' is harder than Q

Examples

Longest palindrome subsequence

\propto_p

Longest common subsequence

Bipartite matching \propto_p *Max flow*

Median \propto_p *Sorting*

Uniqueness \propto_p *Sorting*

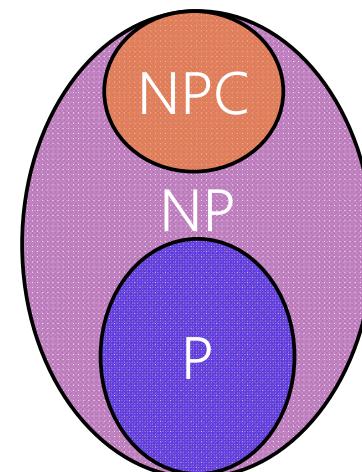
.....

NP complete: The hardest problem in NP

Definition: NP-Complete

A decision problem π is said to be *NP-complete* if the following two properties hold:

1. $\pi \in \text{NP}$, and
2. for every problem $\pi' \in \text{NP}$, $\pi' \leq_p \pi$.



The first NPC problem

DecisionProblem: SAT

Input: A boolean formula

Output: Is this formula satisfiable?

2-SAT: $(x \vee y) \wedge (x \vee \bar{z}) \wedge (y \vee z)$

3-SAT: $(x \vee y \vee z) \wedge (x \vee \bar{y} \vee \bar{z}) \wedge (\bar{x} \vee y \vee z) \wedge (y \vee \bar{y} \vee \bar{z})$

Cook-Levin Theorem [Cook, 1971; Levin, 1973]

SAT is NP-complete



How to find the second NPC problem?

The 2nd, 3rd, ..., NPC problems

Lemma

For any two decision problem A, B in NP,
if

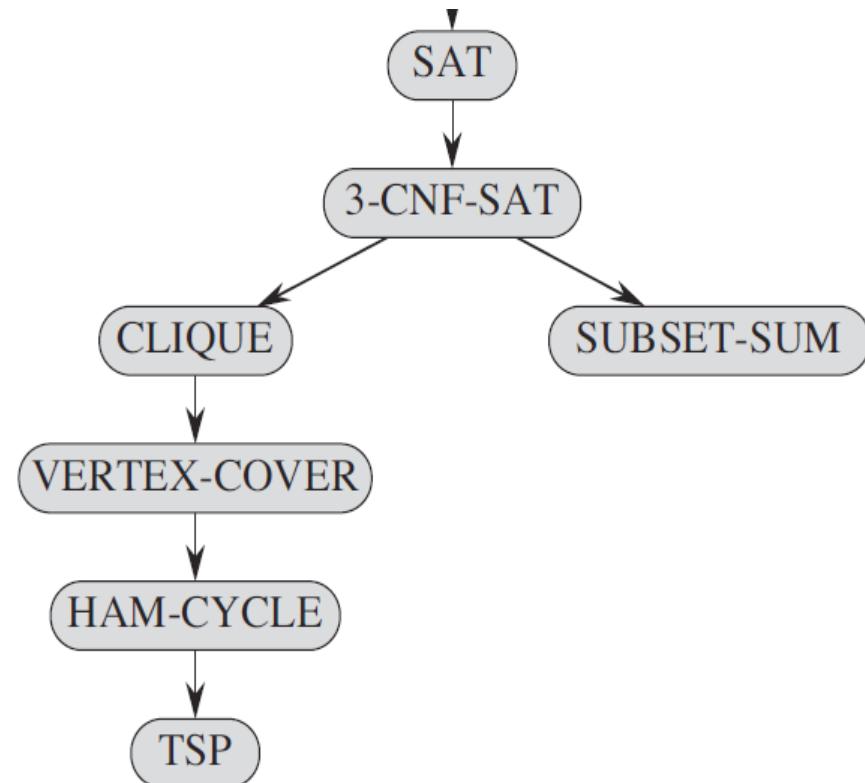
A is NP-complete

and

$A \leq_p B$,

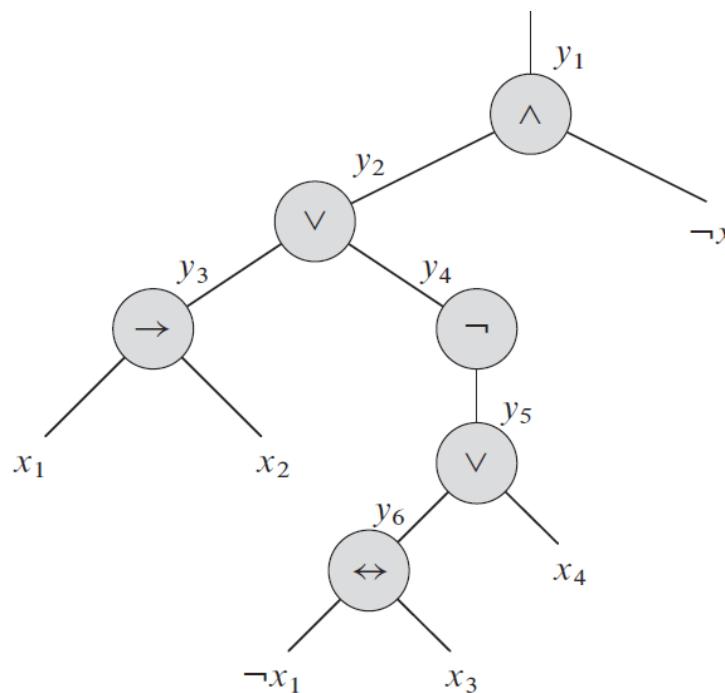
then

B is NP-complete.



SAT --> 3-SAT

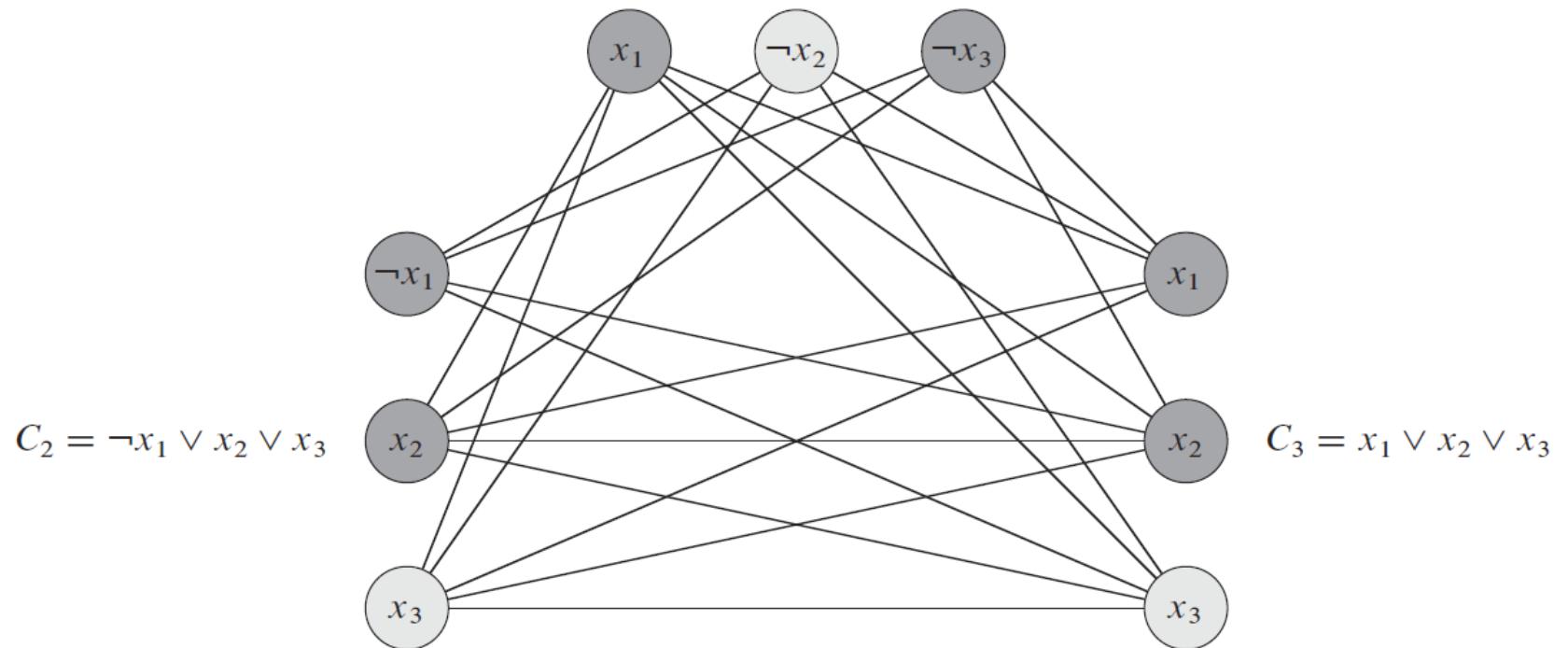
$$\phi = ((x_1 \rightarrow x_2) \vee \neg((\neg x_1 \leftrightarrow x_3) \vee x_4)) \wedge \neg x_2$$



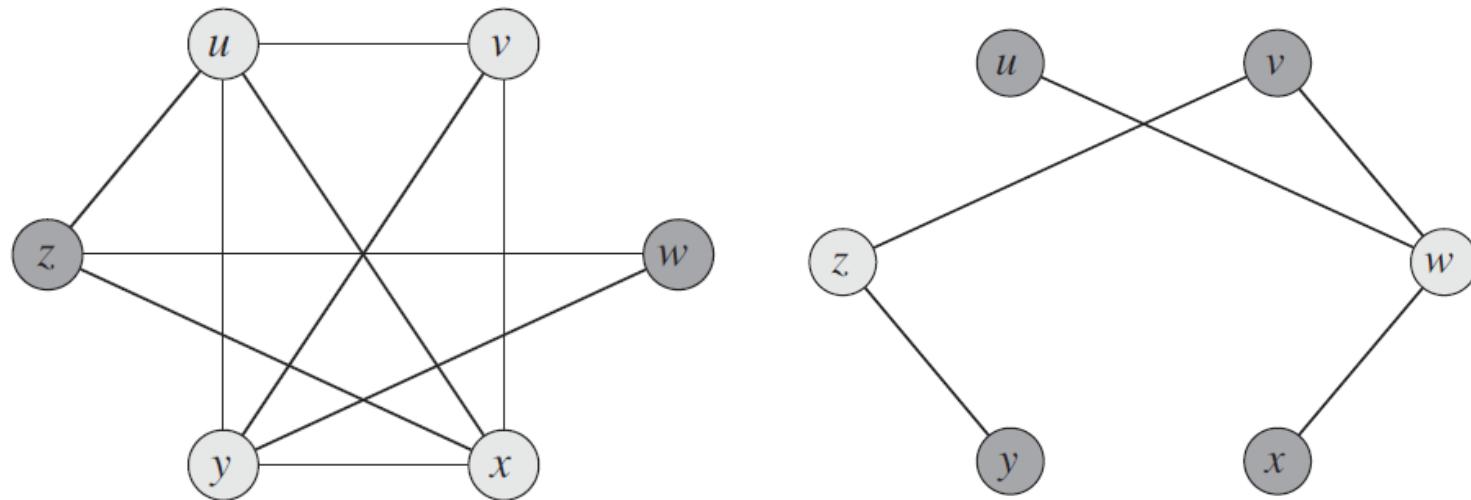
$$\begin{aligned}\phi' = & y_1 \wedge (y_1 \leftrightarrow (y_2 \wedge \neg y_2)) \\ & \wedge (y_2 \leftrightarrow (y_3 \vee y_4)) \\ & \wedge (y_3 \leftrightarrow (x_1 \rightarrow x_2)) \\ & \wedge (y_4 \leftrightarrow \neg y_5) \\ & \wedge (y_5 \leftrightarrow (y_6 \vee x_4)) \\ & \wedge (y_6 \leftrightarrow (\neg x_1 \leftrightarrow x_3))\end{aligned}$$

3-SAT --> Clique

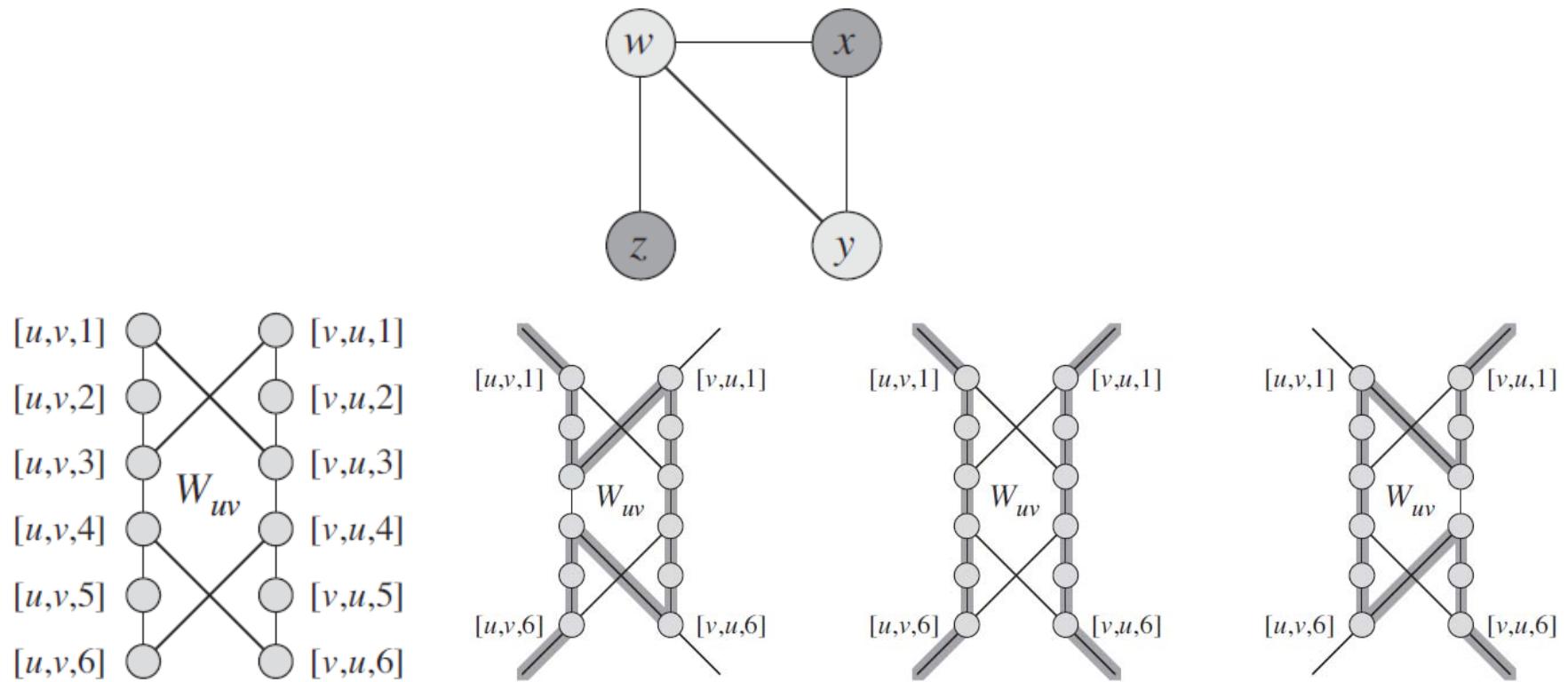
$$C_1 = x_1 \vee \neg x_2 \vee \neg x_3$$



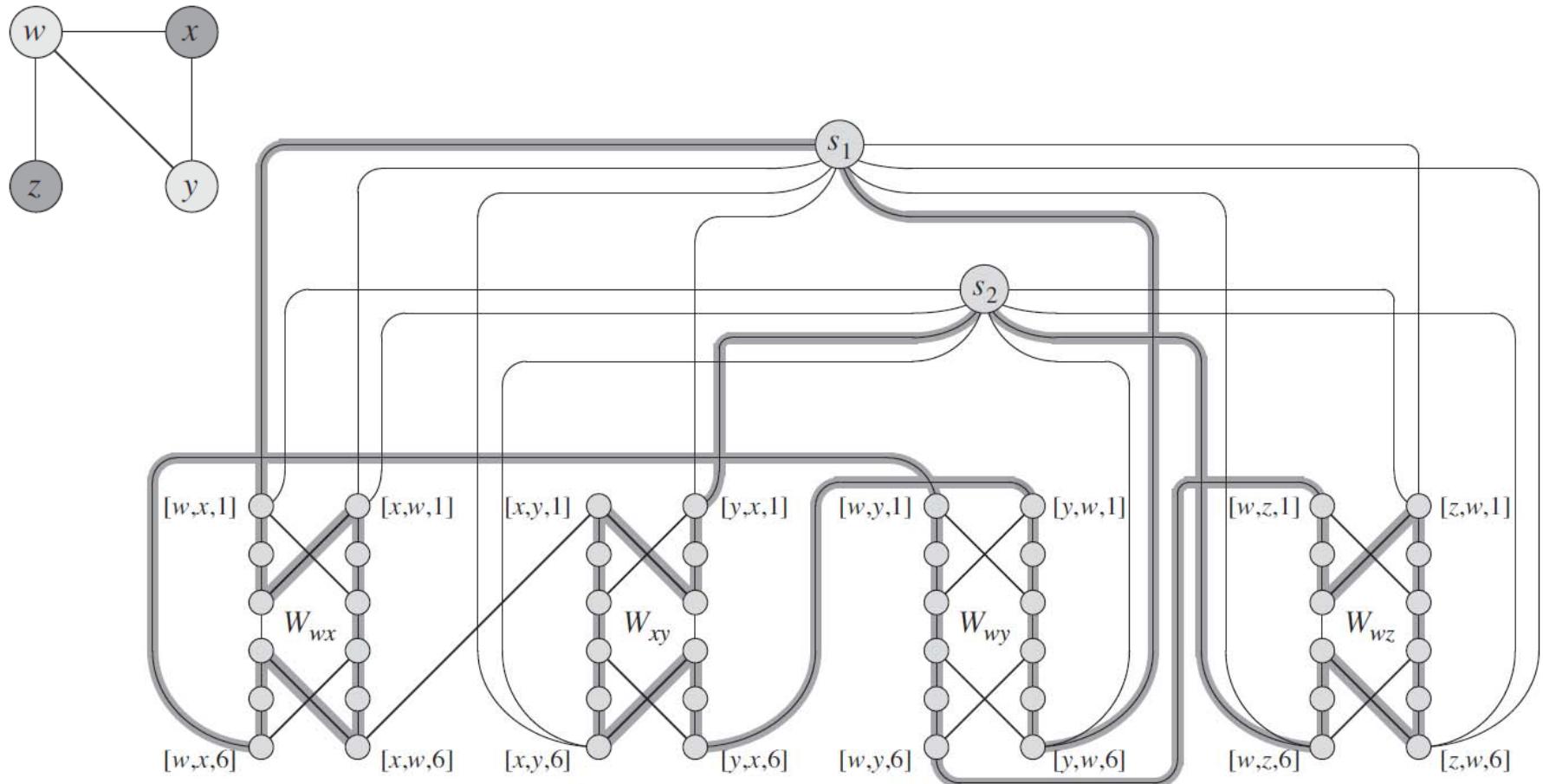
Clique \rightarrow Vertex Cover



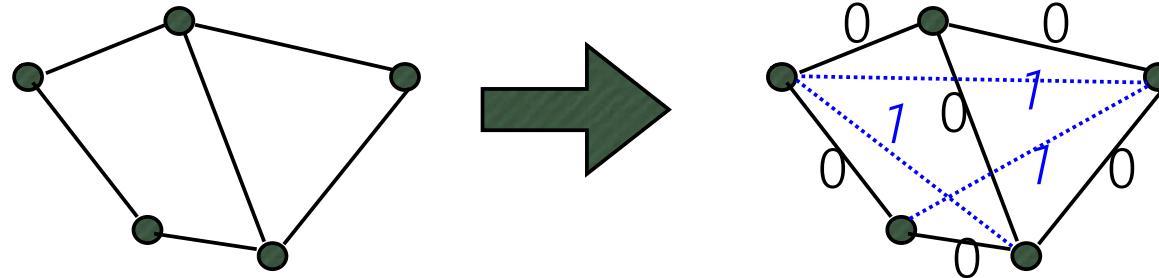
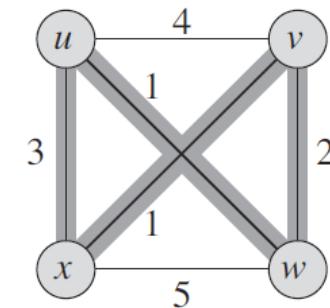
Vertex Cover --> Hamiltonian



Vertex Cover --> Hamiltonian



Hamiltonian \rightarrow TSP



3-SAT --> Subset Sum

$$C_1 \wedge C_2 \wedge C_3 \wedge C_4$$

$$C_1 = (x_1 \vee \neg x_2 \vee \neg x_3)$$

$$C_2 = (\neg x_1 \vee \neg x_2 \vee \neg x_3)$$

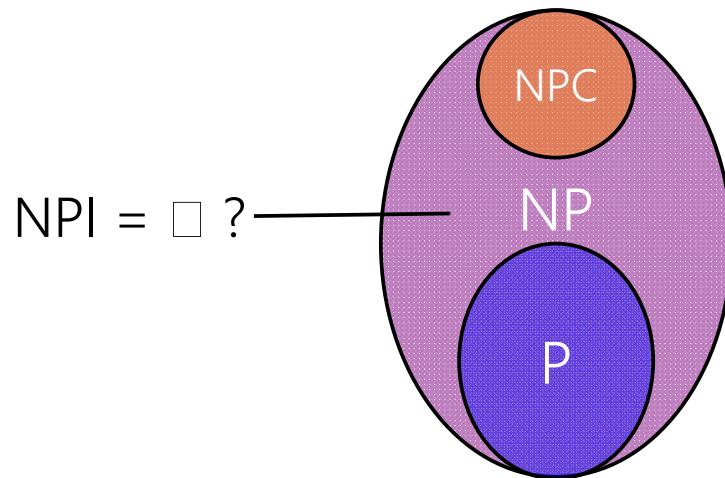
$$C_3 = (\neg x_1 \vee \neg x_2 \vee x_3)$$

$$C_4 = (x_1 \vee x_2 \vee x_3)$$

		x_1	x_2	x_3	C_1	C_2	C_3	C_4
v_1	=	1	0	0	1	0	0	1
v'_1	=	1	0	0	0	1	1	0
v_2	=	0	1	0	0	0	0	1
v'_2	=	0	1	0	1	1	1	0
v_3	=	0	0	1	0	0	1	1
v'_3	=	0	0	1	1	1	0	0
s_1	=	0	0	0	1	0	0	0
s'_1	=	0	0	0	2	0	0	0
s_2	=	0	0	0	0	1	0	0
s'_2	=	0	0	0	0	2	0	0
s_3	=	0	0	0	0	0	1	0
s'_3	=	0	0	0	0	0	2	0
s_4	=	0	0	0	0	0	0	1
s'_4	=	0	0	0	0	0	0	2
t	=	1	1	1	4	4	4	4

Conclusion

1. What is **P**, **NP**, and **NPC**?
2. How to **prove** a problem is NPC?
3. What is the **possible picture** of NP theory?



Possible candidates:
• Factoring
• Graph isomorphism

Ladner's Theorem [Ladner,1975]

If P ≠ NP, then NPI is not empty;

Class P

A is an *algorithm* solving a decision problem π , if given any instance I of the problem π , the answer of I is exactly the output of A.

A is a *deterministic algorithm* solving a problem π if A is a algorithm solving π , and when given any instance of the problem π , A has only one choice in each step throughout its execution.

A problem π has a deterministic algorithm which runs in polynomial time of the input size, if and only if π is in Class P.

Class NP

1. On input x , a *nondeterministic algorithm* consists of two phases: **The guessing phase**. Guess a solution in polynomial time. **The verification phase**. Use a polynomial-time deterministic algorithm to verify whether the guess is really a solution or not.

Let A be a nondeterministic algorithm for a problem π . We say that A *accepts* an instance I of π iff on input I there is a guess that leads to a 'yes' answer.

3-color, k-clique

Class NP

The *class NP* consists of those decision problems for which there exists a *nondeterministic* algorithm that run in polynomial time.

Lecture 14 Lower Bounds

- Decision tree model
- Linear-time reduction

Lower & Upper Bounds

Upper bounds of a problem A: there is an algorithm for the problem that is $O(f(n))$.

Lower bounds of a problem A: ALL algorithms for the problem that are $\Omega(f(n))$.

An optimal algorithm for some problem is an algorithm with its upper bound *asymptotically equivalent* to the lower bound of this problem.

Trivial lower bounds

- Finding the maximum element of a list of n elements: $\Omega(n)$.
- Multiplying two $n \times n$ matrix: $\Omega(n^2)$.

Roadmap

- Decision tree model
- Linear-time reduction

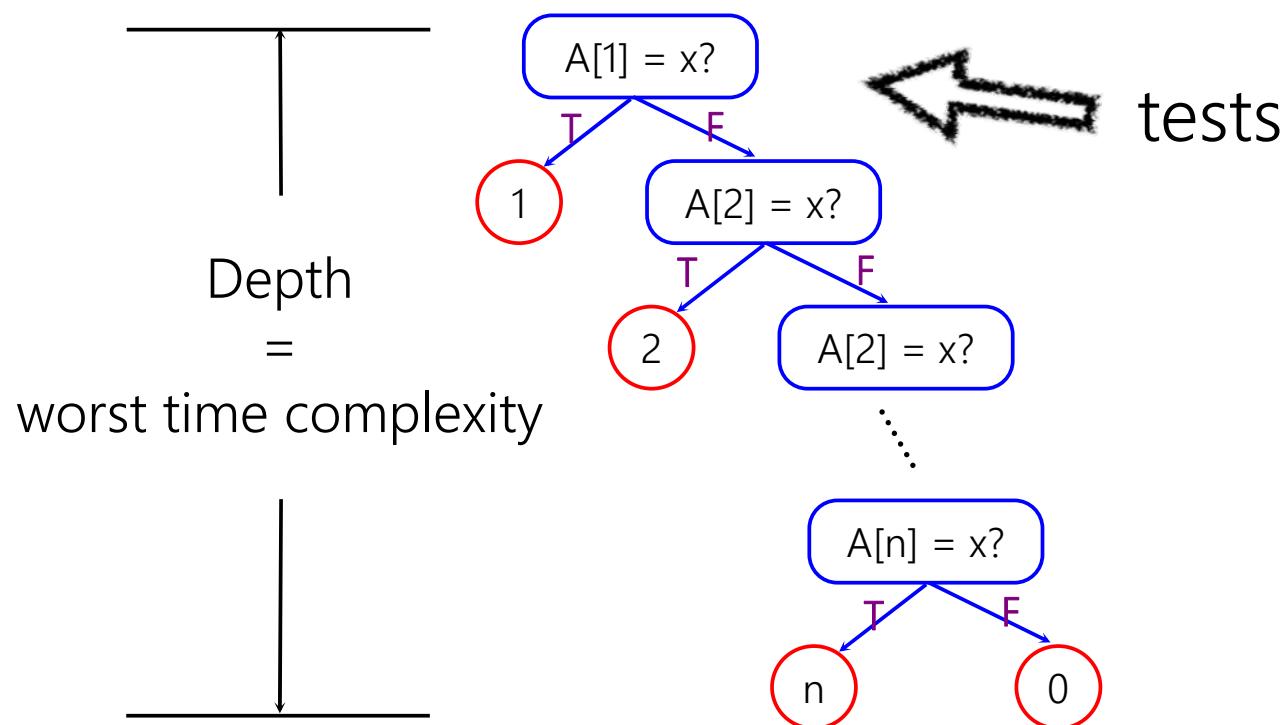
Decision tree model

Definition: Decision Tree Model

Decision tree model is the model of computation in which an algorithm is considered to be a **decision tree**, i.e., a sequence of branching operations based on comparisons of some quantities.

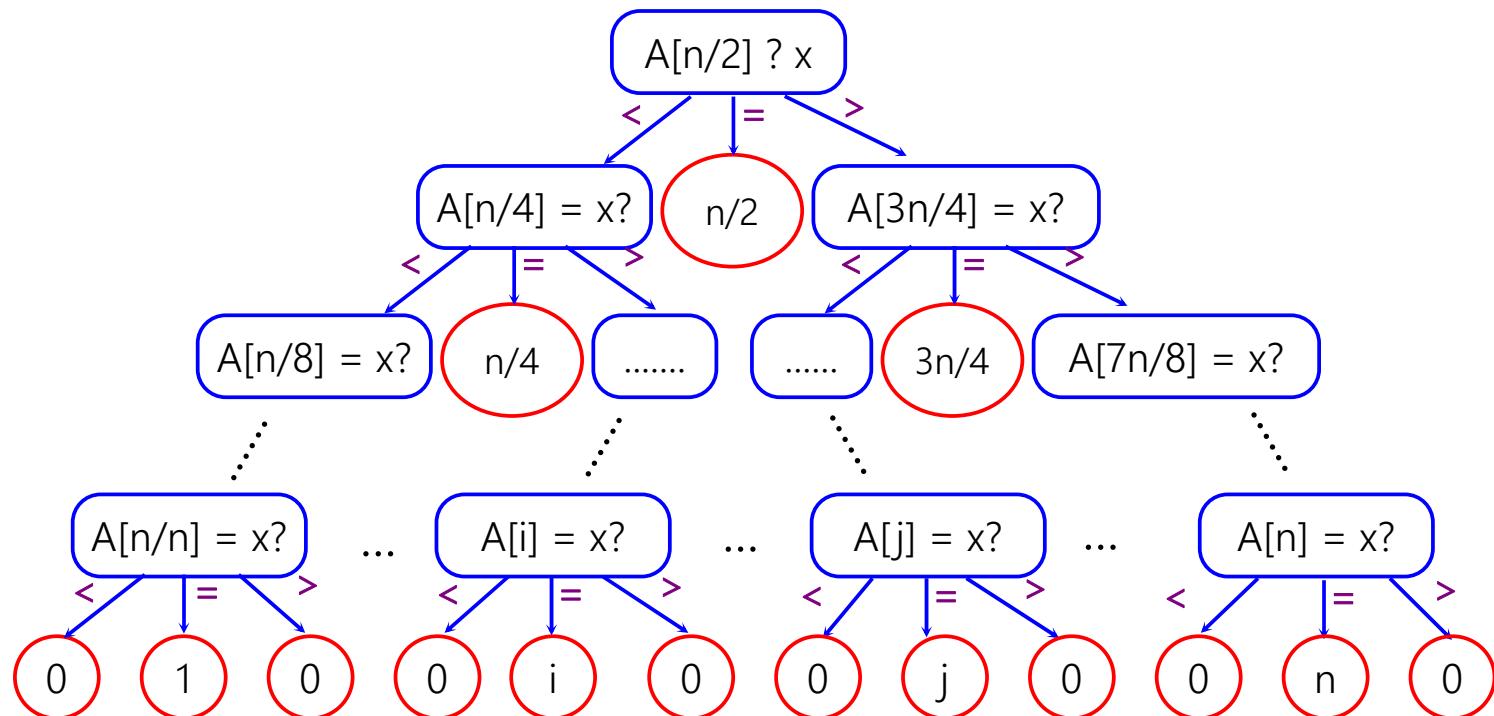
Algorithm \Leftrightarrow Decision tree

Linear Search



Algorithm \Leftrightarrow Decision tree

Binary Search



Lower bounds based on Decision Tree Model

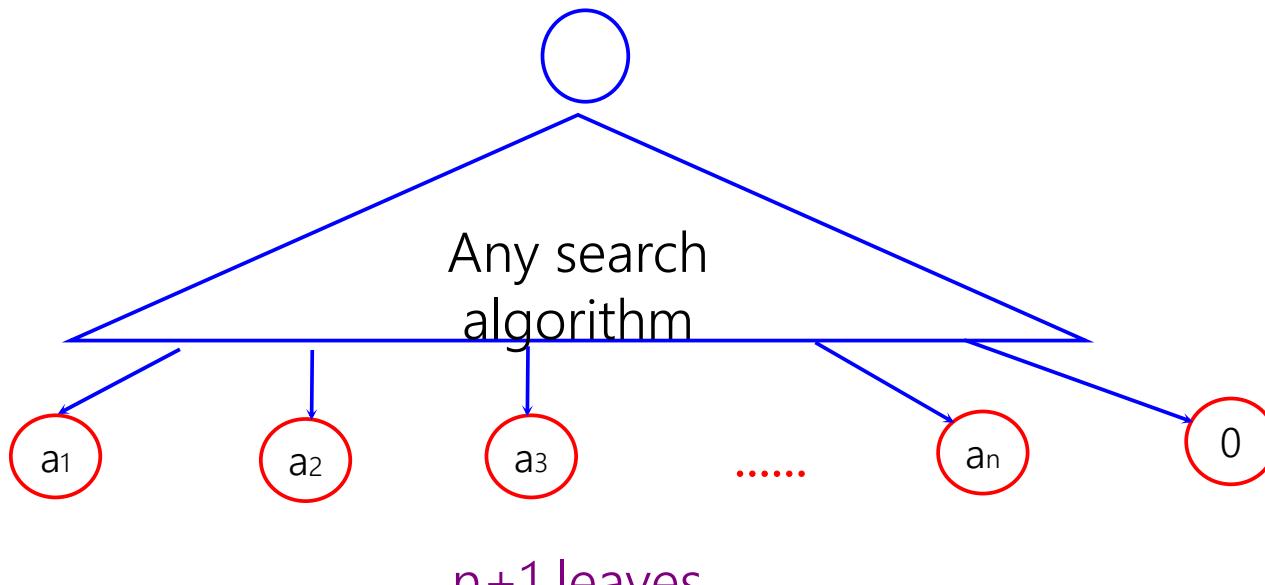
- How many leaves?
- Lower bound = the smallest depth

Searching problem

Problem: Search

Input: A array $A[1..n]$ of n elements, and an element a

Output: The index i if $A[i]=a$, 0 otherwise



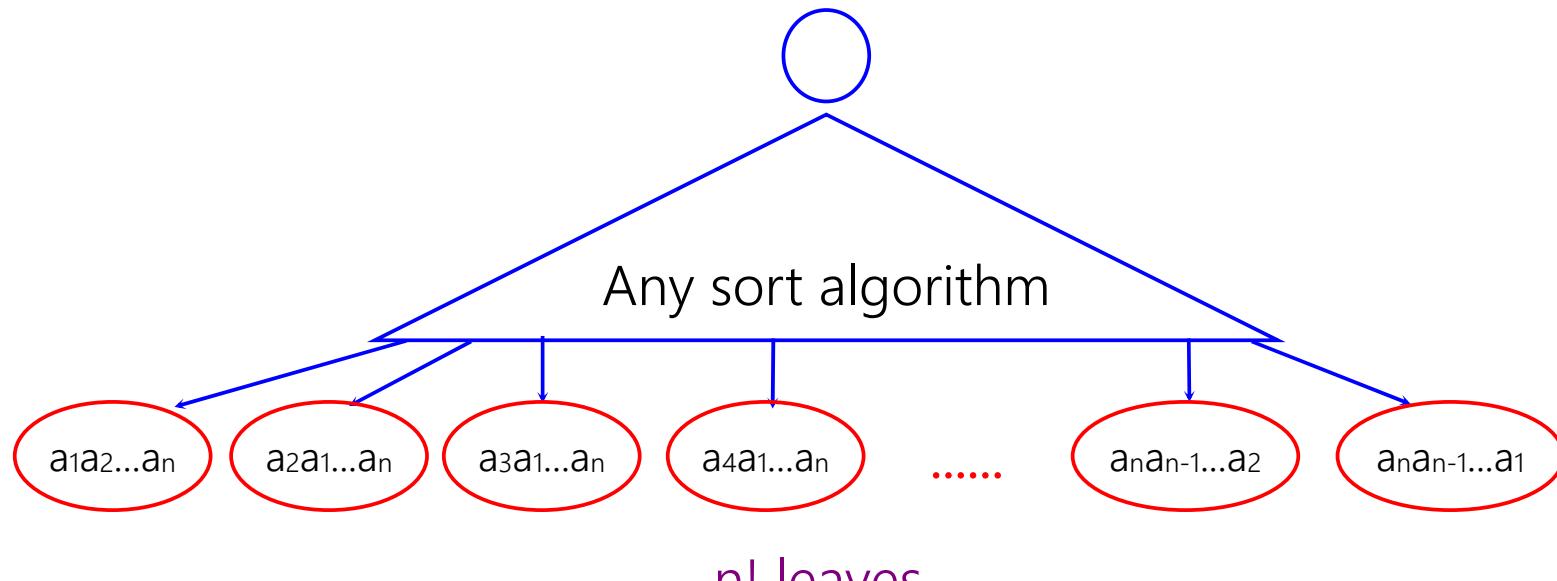
$$\lceil \log(n+1) \rceil = \Theta(\log n)$$

Sorting problem

Problem: Sorting

Input: An array $A[1..n]$ of n elements

Output: The sorted array of $A[1..n]$



$n!$ leaves

$$\square \log(n!) \square = \Theta(n \log n)$$

Decision trees

Definition: Simple Decision tree model

Every decision is based on the **comparison** of two numbers within constant time

Definition: Algebra Decision tree model

Every decision is based on some polynomial function:

$$f(x_1, x_2, \dots, x_n) = 0?$$

Element uniqueness

Problem: ElementUniqueness

Input: A set of n numbers

Output: yes, if there does not exist two of them that are equal; no, otherwise

Theorem

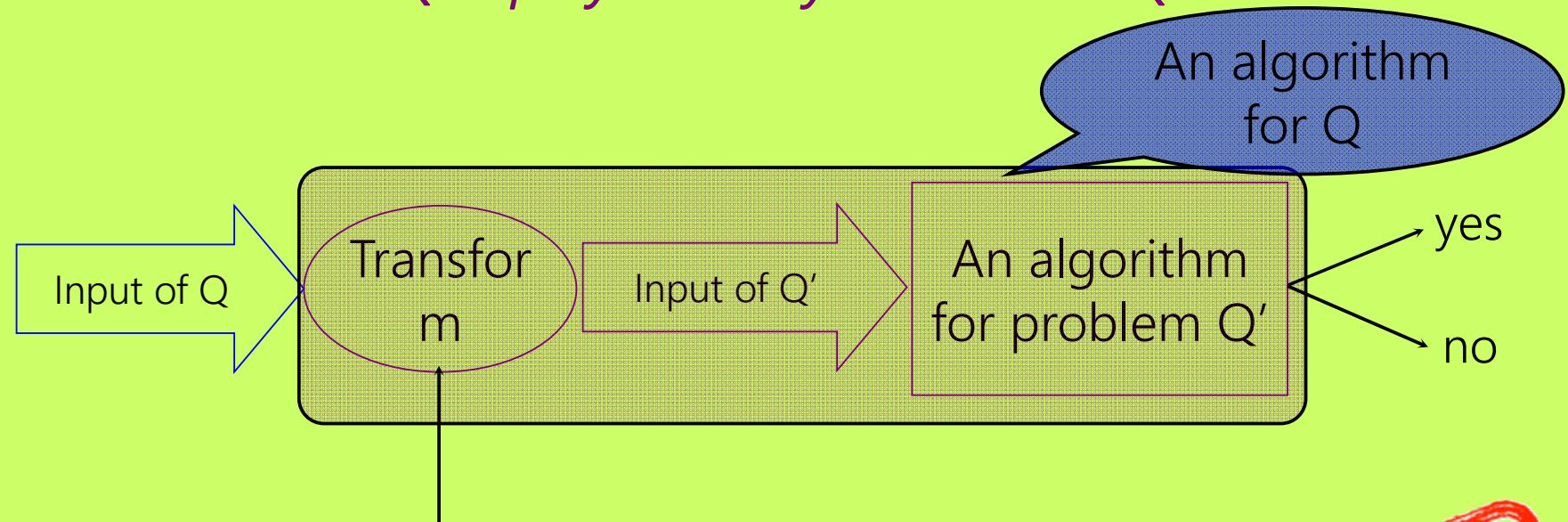
Any algorithm base on *algebraic decision tree model* to determine element uniqueness requires $\Omega(n \log n)$ comparisons in the worst case.

Roadmap

- Decision tree model
- Linear-time reduction

Polynomial-time reduction

Q is polynomially reduced to Q'

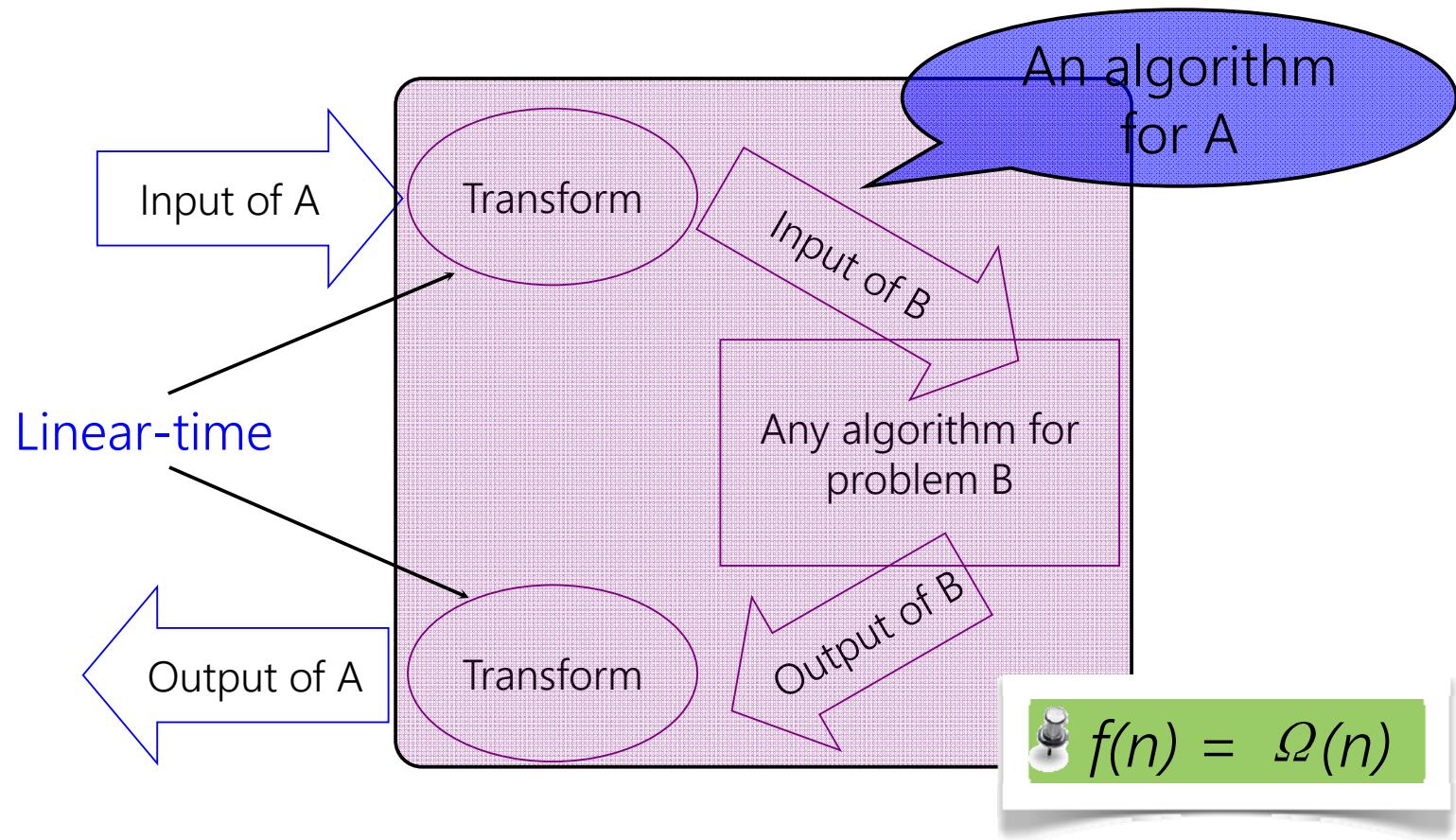


Polynomial-time

$Q \propto_p Q'$: Q' is harder than Q

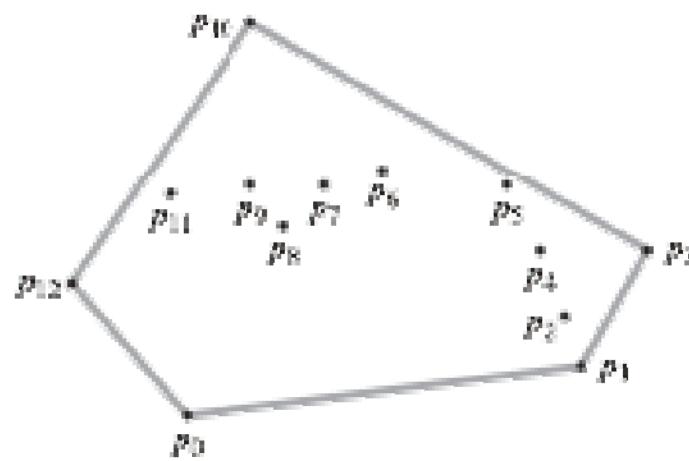
Review
(P v.s. NP)

Linear-time reductions



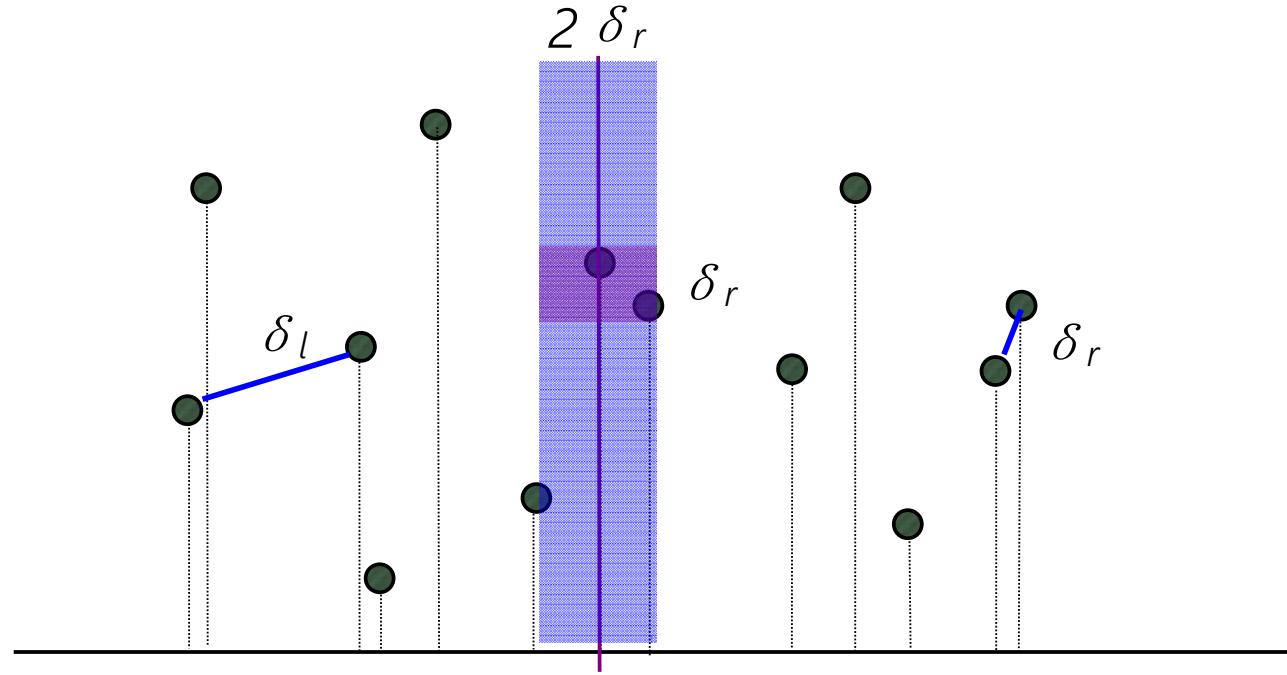
$A \propto_n B$: A has lower bound $\Omega(f(n))$ implies
B has lower bound $\Omega(f(n))$

The convex hull problem



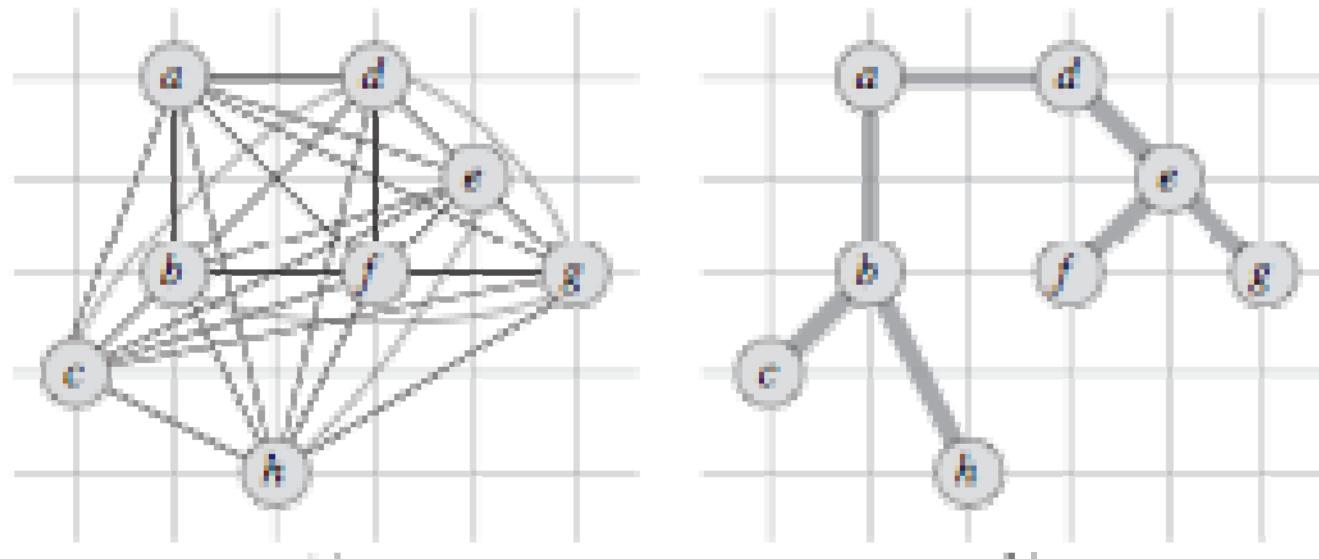
Sorting ∞_n ConvexHull

The closest pair problem



ElementUniqueness \propto_n *ClosestPair*

The Euclidean minimum spanning tree problem



Sorting ∞_n EuclideanMinSpan

Conclusion

Theorem

In algebraic decision tree model for computation, any algorithm that solves

- *sorting problem*, or
- *element uniqueness problem*, or
- *convex hull problem*, or
- *the closest pair problem*, or
- *the Euclidean minimum spanning tree problem*

requires $\Omega(n \log n)$ operations in the worst case.

Lecture 14 Approximation algorithms

- Approximation algorithms
- Hardness result: Traveling sales man problem
- Easiness result: Subset Sum problem

Optimization problem

Optimization Problems are a type of problems of finding the optimum solution.

- Knapsack
- Minimum spanning tree
- Change making
- Min Vertex cover
- Max-Clique
- Traveling-Sales-Person

Approximation algorithm

An approximation algorithm for an optimization problem π is a (polynomial time) algorithm such that given an instance I , it outputs some solution.

Approximation ratio

$$\frac{|A(I) - OPT(I)|}{OPT(I)}$$

or

$$A(I) / OPT(I)$$

Reason1.

Planar graph coloring problem

$$|A(I) - OPT(I)| < 4$$

Reason2.

Knapsack problem

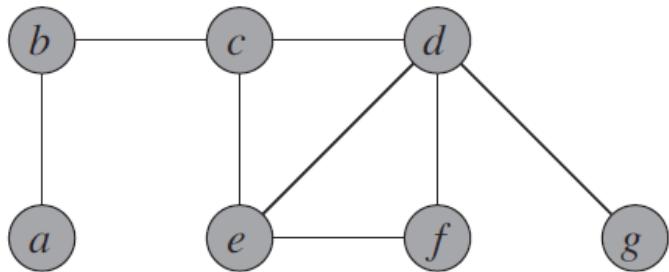
for any k . there DOES NOT exist A , s.t.

$$|A(I) - OPT(I)| \leq k$$

Where are we?

- Approximation algorithms:
 - Vertex cover problem
 - Knapsack problem
 - Euclidean TSP problem
- Hardness result: Traveling Sales Person problem
- Easiness result: Subset Sum problem

Vertex cover problem



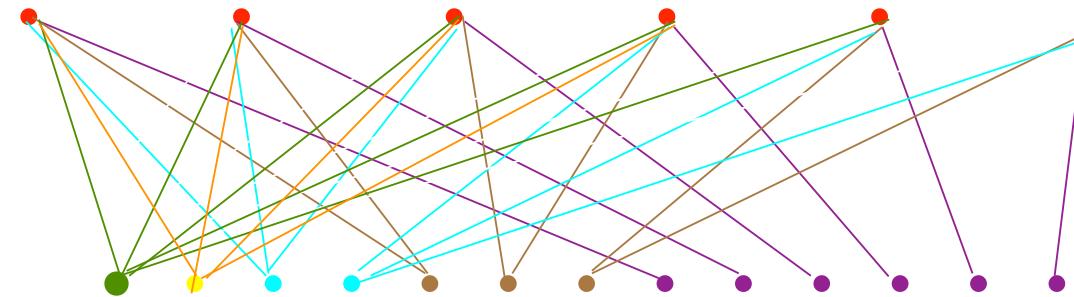
- None is correct.
- Which one has the best approx. ratio?

Algorithm1: Pick up a node $v \rightarrow$ delete v and all edges incident to v . $R_1 = \infty$

Algorithm2: Pick up an edge $e(u,v) \rightarrow$ add u, v to the cover \rightarrow delete all edges incident to u, v . $R_2 \leq 2$

Algorithm3(Greedy): Pick up a node v with max degree \rightarrow delete v and all edges incident to v . $R_3 = \infty$

VC - Greedy



$$R \geq \ln n$$

Greedy for Knapsack

Algorithm1: Repeatedly choose the item with the largest value/size ratio. $R_1 = \infty$

What about $U=\{u_1, u_2\}$, $v_1=2$, $s_1=1$; $v_2=s_2=C>2$?

Algorithm2: Repeatedly choose the item with the largest value/size ratio. Then compare the total value with the item with maximum value. $R_2 \leq 2$

Greedy for Knapsack

Algorithm 15.3 KNAPSACKGREEDY

Input: $2n + 1$ positive integers corresponding to item sizes $\{s_1, s_2, \dots, s_n\}$, item values $\{v_1, v_2, \dots, v_n\}$ and the knapsack capacity C .

Output: A subset Z of the items whose total size is at most C .

1. Renumber the items so that $v_1/s_1 \geq v_2/s_2 \geq \dots \geq v_n/s_n$.
2. $j \leftarrow 0; K \leftarrow 0; V \leftarrow 0; Z \leftarrow \{\}$
3. **while** $j < n$ and $K < C$
4. $j \leftarrow j + 1$
5. **if** $s_j \leq C - K$ **then**
6. $Z \leftarrow Z \cup \{u_j\}; K \leftarrow K + s_j; V \leftarrow V + v_j$
7. **end if**
8. **end while**



$\forall I, OPT(I)/A(I) \leq 2$

R <= 2

Let $X = \{u_1, u_2, \dots, u_r\}$ be the set of items of an optimal solution, assuming

$$v_i/s_i \geq v_{i+1}/s_{i+1}. \quad (1)$$

If $|X| \leq 1$ then the solution of the algorithm is optimal. So suppose $|X| > 1$.

Let u_m be the first item of X not included into the knapsack by the algorithm. If no such items exists then the output of the algorithm is optimal. So assume that u_m exists.

The optimal solution can be rewritten as

$$OPT(I) = \sum_{j=1}^{m-1} v_j + \sum_{j=m}^r v_j \quad (2)$$

Let W denote the set of items packed by the algorithm before u_m but not in $\{u_1, u_2, \dots, u_{m-1}\}$. In other words, $u_j \in W$ iff $u_j \notin \{u_1, u_2, \dots, u_{m-1}\}$ and

$$v_j/s_j \geq v_m/s_m \quad (3)$$

Now $A(I)$ can be written as

$$A(I) \geq \sum_{j=1}^{m-1} v_j + \sum_{j \in W} v_j \quad (4)$$

Let

$$C' = C - \sum_{j=1}^{m-1} s_j \quad (5)$$

and

$$C'' = C' - \sum_{u_j \in W} s_j \quad (6)$$

By the definition of u_m , we have

$$C'' < s_m \quad (7)$$

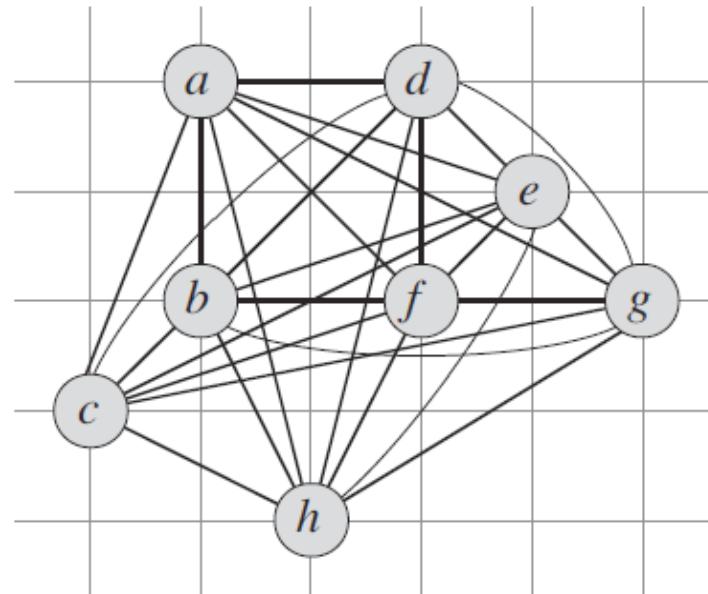
$$R \leq 2$$

$$\begin{aligned} OPT(I) &\leq \sum_{j=1}^{m-1} v_j + C' \frac{v_m}{s_m} \\ &= \sum_{j=1}^{m-1} v_j + \left(\sum_{v_j \in W} s_j + C'' \right) \frac{v_m}{s_m} \\ &< \sum_{j=1}^{m-1} v_j + \sum_{v_j \in W} v_j + v_m \\ &< A(I) + A(I) \end{aligned}$$

$$\frac{OPT(I)}{A(I)} < 2.$$

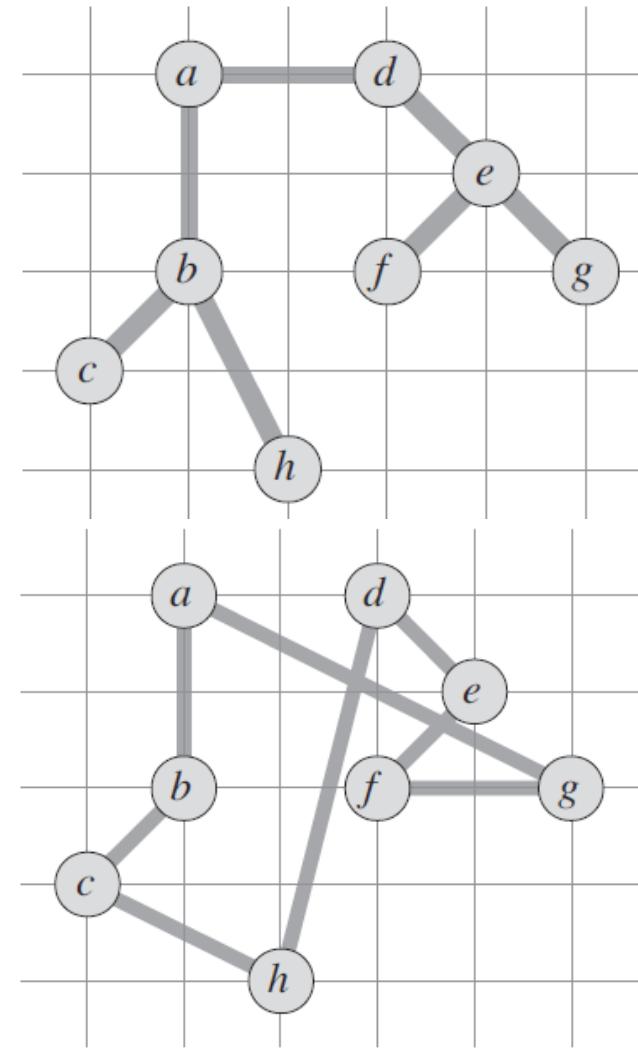
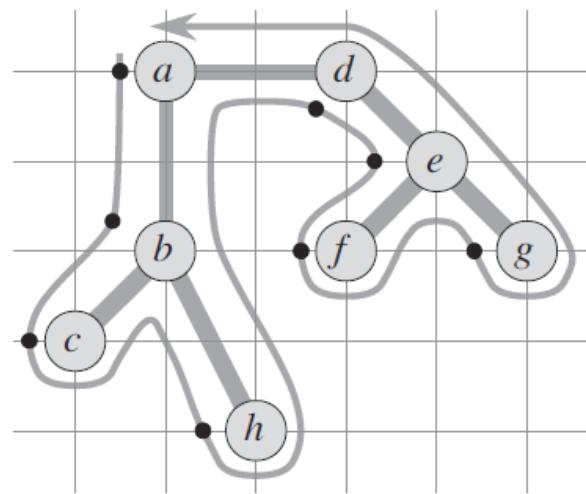
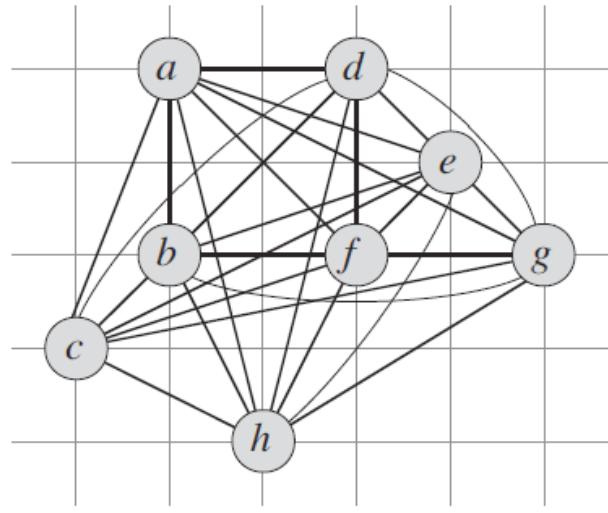
Euclidean TSP problem

TSP with the triangle inequality: $ab + ad \geq bd$



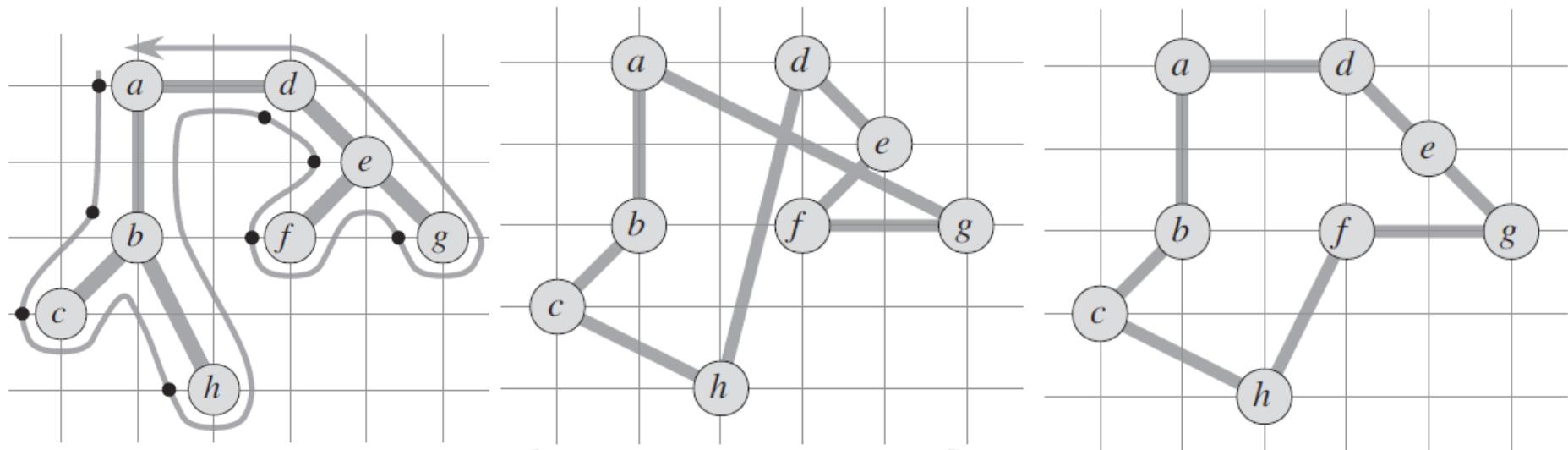
A simple approximation

APP-ETSP-MST



A simple approximation

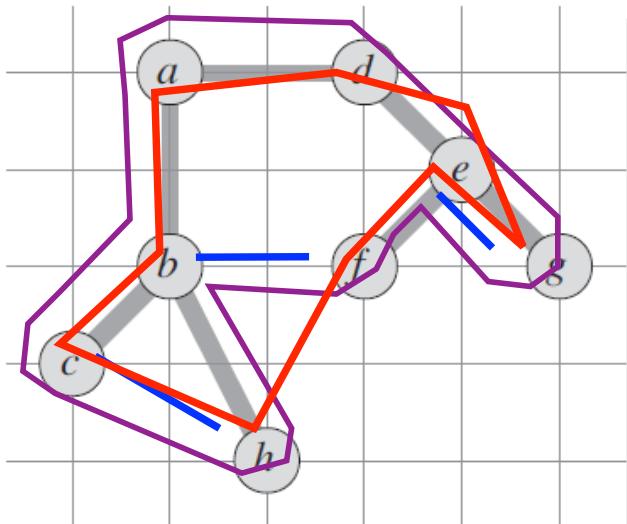
APP-ETSP-MST



Theorem. $R < 2$.

$$2\text{OPT}(\mathcal{I}) > 2\text{MST}(\mathcal{I}) > A(\mathcal{I})$$

An improved heuristic



Algorithm 15.1 ETSPAPPROX

Input: A set S of n points in the plane.

Output: An Eulerian tour τ of S .

1. Construct a minimum spanning tree T of S
2. Identify the set X of odd degree vertices in T
3. Find a minimum weight matching M on X
4. Find a Eulerian tour τ_e in $T \cup M$
5. Traverse τ_e edge by edge and bypass every previously visited vertex.
6. Let τ be the resulting tour

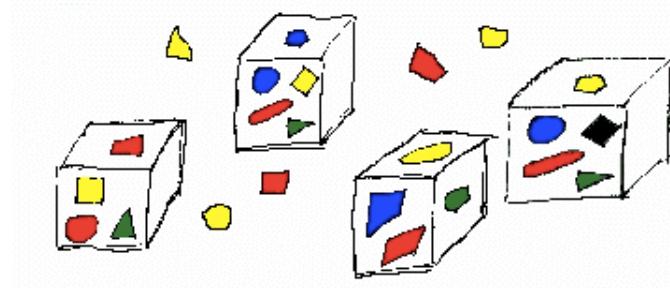
Theorem $R < 3/2$.

$$\text{OPT}(I) > \text{MST}(I)$$

$$\text{OPT}(I) \geq 2\text{Matching}(I)$$

$$\text{MST}(I) + \text{Matching}(I) > \text{A}(I)$$

Bin packing



Bin: 20

Item: 15,14,9,8,7,6,5,4

Roadmap

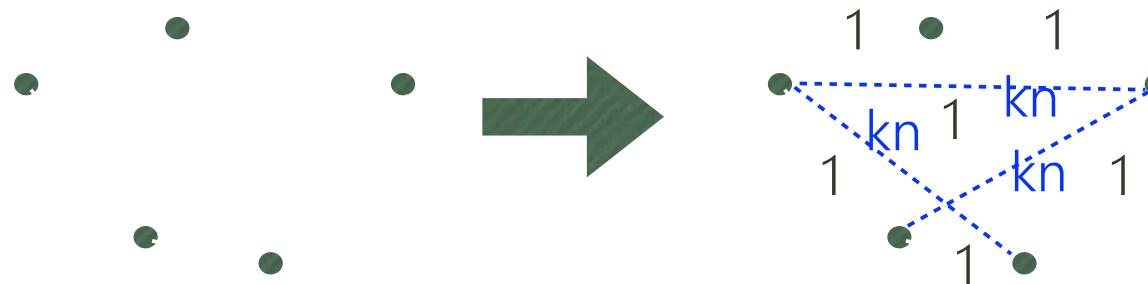
- Approximation algorithms:
 - Vertex cover problem
 - Knapsack problem
 - Euclidean traveling sales man problem
- Hardness result: Traveling sales man problem
- Easiness result: Subset Sum problem

Traveling sales person problem

Theorem

There is no approximation algorithm A for the problem Traveling Sales person with $R_A < \infty$ unless $NP = P$.

Hamiltonian cycle \Rightarrow Approximation TSP with $R < k$



Roadmap

- Approximation algorithms:
 - Vertex cover problem
 - Knapsack problem
 - Euclidean traveling sales man problem
- Hardness result: Traveling sales man problem
- Easiness result: Subset Sum problem

Subset sum

Problem: SubsetSum

Input: A set of numbers $A=\{a_1, a_2, \dots, a_n\}$, and a sum s

Output: Yes, if there exists a subset $B \subseteq A$ such that the sum of B equals to s ; No, otherwise.

Problem: SubsetSumOpt

Input: A set of numbers $A=\{a_1, a_2, \dots, a_n\}$, and a sum s

Output: a subset of the items that maximizes the total sum of their sizes without exceeding the sum s .

APP_SUBSETSUM

We construct a new instance I' and apply the subset-sum algorithm to it:

$$\begin{aligned} K &= C/(2(k+1)n) \\ C' &\stackrel{\text{def}}{=} \lfloor C/K \rfloor \\ s'_j &\stackrel{\text{def}}{=} \lfloor s_j/K \rfloor \end{aligned}$$

Time complexity: $O(nC/K) = O(kn^2)$

S={104,102,201,101}, C=308, k=5/2

S={109,109,208,109}, C=308, k=5/2

FPTAS for Subset sum

Theorem

Let $\epsilon = 1/k$ for some $k \geq 1$. Then the running time of APP_SUBSETSUM is $O(kn^2)$, and its performance ratio is $1 + \epsilon$.

Easiness results

Polynomial Approximation Scheme (PAS, or PTAS)

Given any $\epsilon = 1/k$ for some positive integer k , we can construct an approximation algorithm for Knapsack problem with $R < 1+\epsilon$, and the algorithm runs in time that is polynomial in the length of the input instance.

Fully Polynomial Approximation Scheme (FPTAS)

Given any $\epsilon = 1/k$ for some positive integer k , we can construct an approximation algorithm for Subset sum problem with $R < 1+\epsilon$, and the algorithm runs in time that is polynomial in the length of the input instance and k .

PAS for Knapsack

APP_KNAPSACK

1. Choose a subset of at most k items and put them in the knapsack;
2. Run Algorithm KNAPSACKGREEDY on the set of remaining items to complete the packing.

Repeat $\sum_{j=0}^k \binom{n}{j}$ times once for a subset of size j

$$\sum_{j=0}^k \binom{n}{j} = O(kn^k)$$

Time complexity: $O(kn^{k+1})$

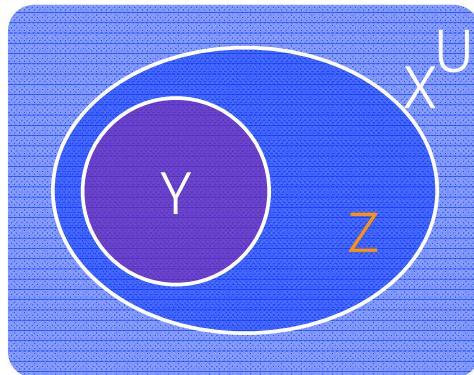
PAS for Knapsack

Theorem

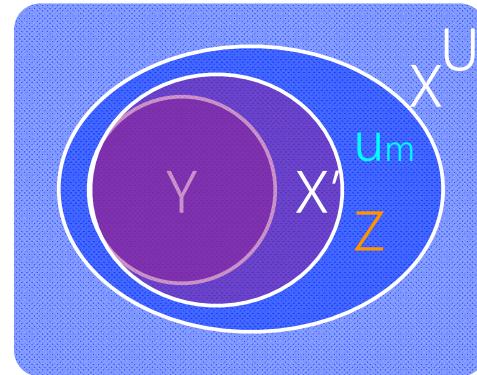
Let $\epsilon = 1/k$ for some $k \geq 1$. Then the running time of APP_KNAPSACK is $O(kn^{k+1})$, and its performance ratio is $1 + \epsilon$.

Proof

1. Let $U = \{u_1, u_2, \dots, u_n\}$ and C being the knapsack capacity.
2. Let X be the set of items of an optimal solution. If $|X| \leq k$ then the solution of the algorithm is optimal. So suppose $|X| > k$.
3. Let $Y = \{u_1, u_2, \dots, u_k\}$ be the set of k largest valued items in X .
4. Let $Z = \{u_{k+1}, u_{k+2}, \dots, u_r\}$ be the set of remaining items in X , assuming $v_{k+1}/s_{k+1} \geq v_{k+2}/s_{k+2} \geq \dots \geq v_r/s_r$.



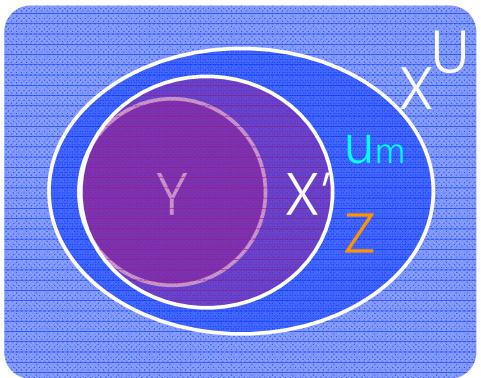
Proof



$$v_j \leq \frac{\text{OPT}(I)}{k+1} \quad \text{for } j = k+1, k+2, \dots, r$$

There exists an iteration such that:

1. Choose and put $\{u_1, \dots, u_k\}$ into the knapsack
2. Run greedy algorithm and u_m is the first item of Z that is not chosen by the greedy algorithm.
3. $W = \{u_j \mid u_j \text{ selected by } A, \text{ and } u_j \not\in \{u_1, \dots, u_m\} \text{ but } v_j/s_j \geq v_m/s_m\}$
4. X' includes $\{u_1, \dots, u_{m-1}\}$ and W .



Proof

$$\text{OPT}(I) = \sum_{j=1}^k v_j + \sum_{j=k+1}^{m-1} v_j + \sum_{j=m}^r v_j$$

$$C' = C - \sum_{j=1}^k s_j - \sum_{j=k+1}^{m-1} s_j \quad A(I) \geq \sum_{j=1}^k v_j + \sum_{j=k+1}^{m-1} v_j + \sum_{j \in W} v_j$$

$$C'' = C' - \sum_{u_j \in W} s_j \quad \text{OPT}(I) \leq \sum_{j=1}^k v_j + \sum_{j=k+1}^{m-1} v_j + C' \frac{v_m}{s_m}$$

$$C'' < s_m$$

$$v_j \leq \frac{\text{OPT}(I)}{k+1} \text{ for } k < j \leq r$$

PAS for Knapsack

Theorem

Let $\epsilon = 1/k$ for some $k \geq 1$. Then the running time of APP_KNAPSACK is $O(kn^{k+1})$, and its performance ratio is $1 + \epsilon$

Running time is exponential to k.

Conclusion

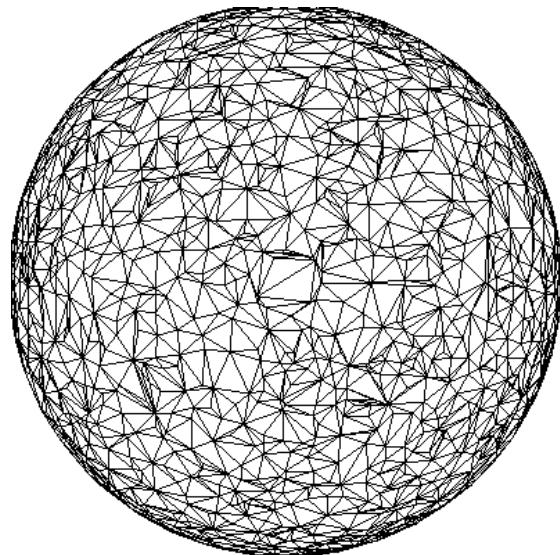
- Approximation algorithms are **polynomial** algorithms.
- The trade-off between **precision** and **time**
- Hardness result: Traveling sales man problem
- Easiness result: Subset Sum problem

Lecture 15 Computational Geometry

- Geometry sweeping
- Geometric preliminaries
- Some basics geometry algorithms

Computational geometry

- Computational geometry is devoted to the study of algorithms which can be stated in terms of geometry.



Roadmap

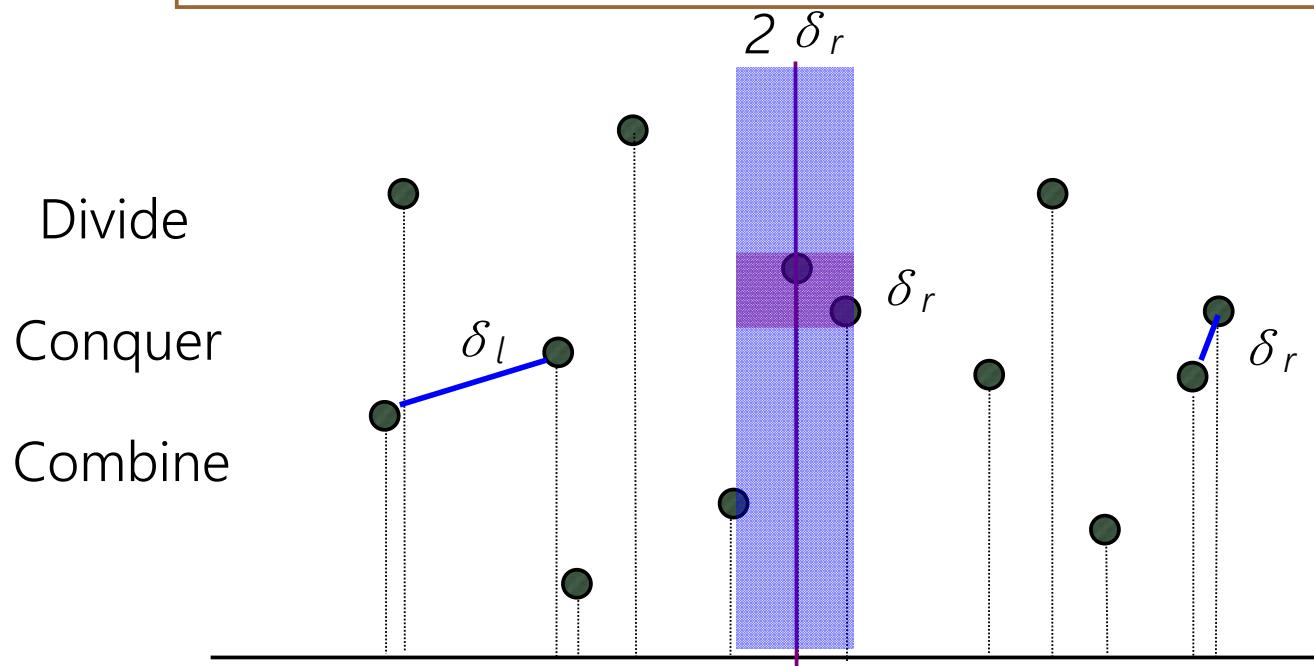
- Closest pair
- Geometry sweeping
- Geometric preliminaries
- Some basics geometry algorithms
 - Convex hull problem
 - Diameter of a set of points
 - Intersection of line segments

Closest pair problem

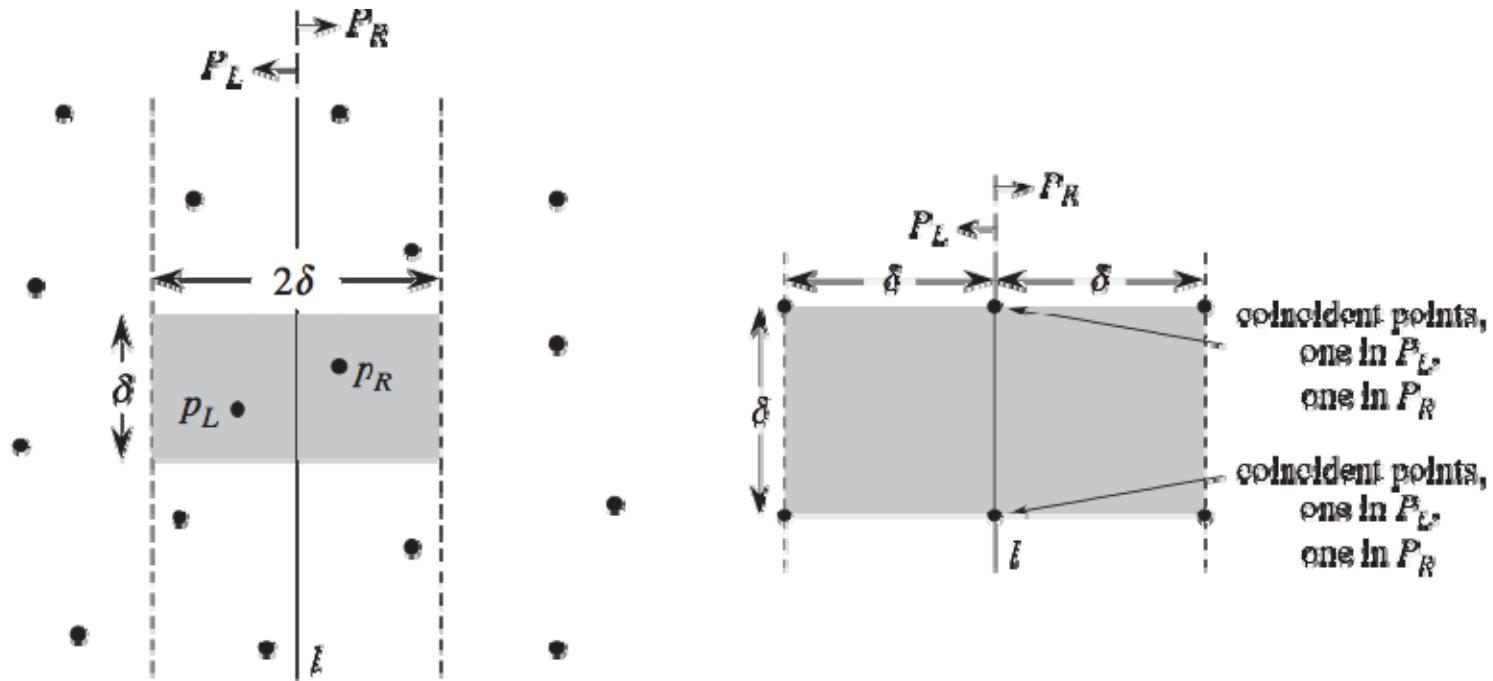
Problem: ClosestPair

Input: A set S of n points in the plane

Output: The pair $p1 = (x_1, y_1)$ and $p2 = (x_2, y_2)$ such that the Euclidean distance between $p1$ and $p2$ is minimal.



Closest pair problem



At most 8 points of P can reside within this $R \times 2R$ rectangle.

Closest pair problem

Algorithm 6.7 ClosestPair

Input: A set of n points in the plane.

Output: The minimum separation realized by two points in S .

1. Sort the points in S in nondecreasing order of their x -coordinates.
2. $Y \leftarrow$ the points in S sorted in nondecreasing order of their y -coordinates.
3. $\delta \leftarrow cp(1, n)$

Procedure $cp(low, high)$

1. $p \leftarrow high - low + 1$
2. **if** $high - low + 1 \leq 3$ **then** compute δ
3. **else**
4. $mid \leftarrow \lfloor (low + high)/2 \rfloor$
5. $x_0 \leftarrow x(S[mid])$
6. $\delta_l \leftarrow cp(low, mid)$
7. $\delta_r \leftarrow cp(mid + 1, high)$
8. $\delta \leftarrow \min\{\delta_l, \delta_r\}$

9. $k \leftarrow 0$
10. **for** $i \leftarrow 1$ **to** n
11. **if** $|x(Y[i]) - x_0| \leq \delta$ **then**
12. $k \leftarrow k + 1$; $T[k] \leftarrow Y[i]$ **end if**
13. **end for**
14. $\delta' \leftarrow \delta$
15. **for** $i \leftarrow 1$ **to** $k - 1$
16. **for** $j \leftarrow i + 1$ **to** $\min\{i + 7, k\}$
17. **if** $d(T[i], T[j]) < \delta'$ **then** $\delta' \leftarrow d(T[i], T[j])$
18. **end for**
19. **end for**
20. **end if**
21. **return** δ'

Closest pair problem

$$T(n) = \begin{cases} 1 & \text{if } n = 2 \\ 3 & \text{if } n = 3 \\ 2T(n/2) + \Theta(n) & \text{if } n > 3 \end{cases}$$

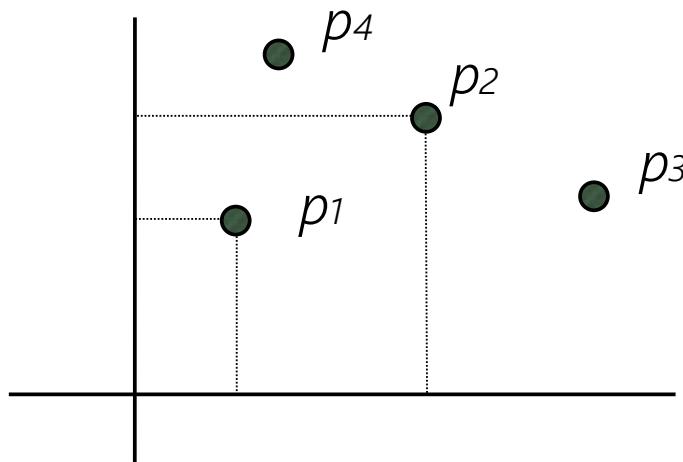
$$T(n) = \Theta(n \log n)$$

Where are we?

- Closest pair
- Geometry sweeping
- Geometric preliminaries
- Some basics geometry algorithms
 - Convex hull problem
 - Diameter of a set of points
 - Intersection of line segments

Maximal points problem

The point $p_2 = (x_2, y_2)$ dominates the point $p_1 = (x_1, y_1)$, denoted by $p_1 \square p_2$, if $x_1 \leq x_2$ and $y_1 \leq y_2$.



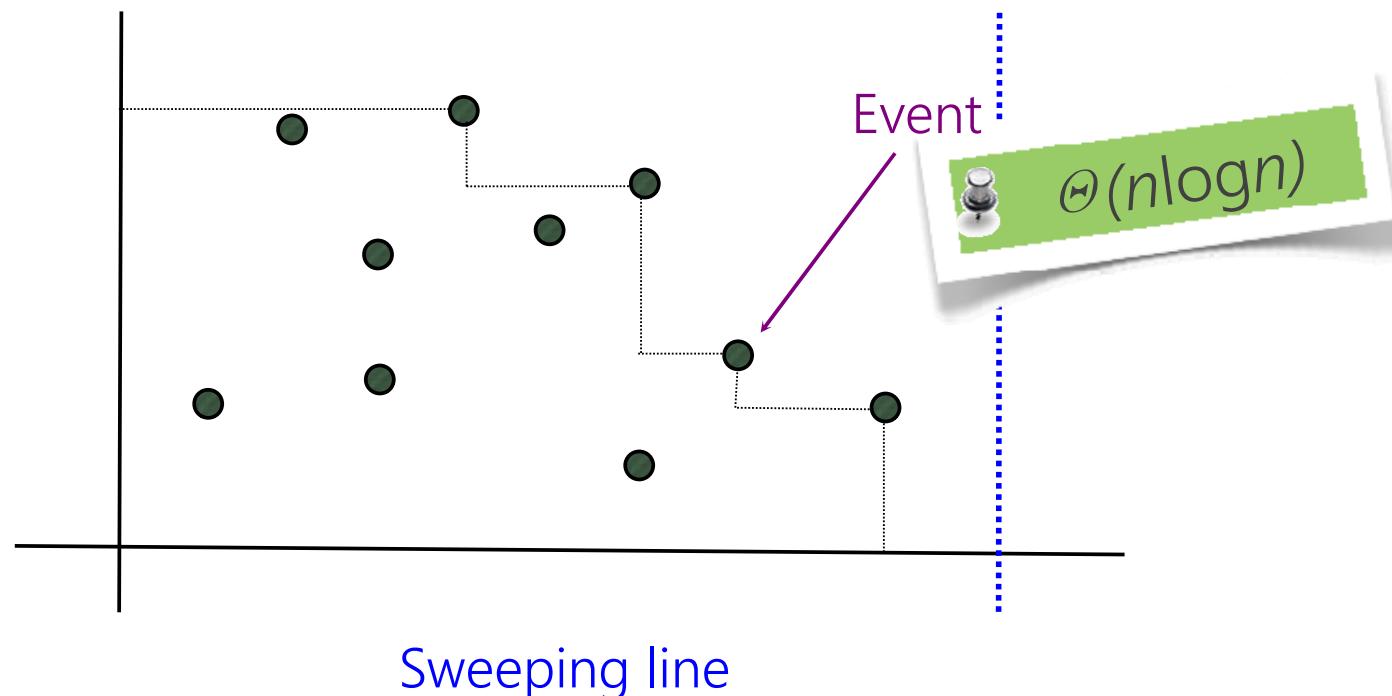
A point p is maximal point if there does NOT exist a point q such that $p \neq q$ and $p \square q$.

Maximal points problem

Problem: MaximalPoints

Input: a set of points with their coordinates (x_i, y_i) .

Output: All the maximal points in the set.

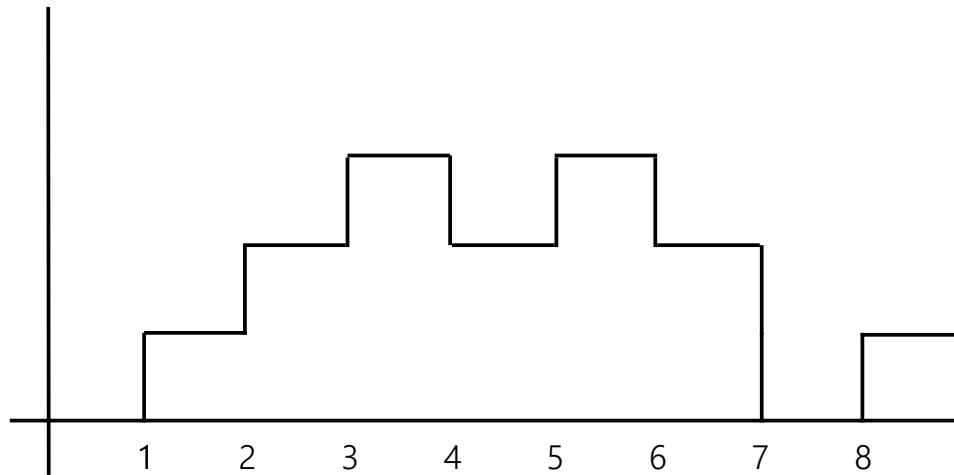


Geometric sweeping

- Determine the sweeping line and Events
- Handle events
- Solve the problem

City Skyline

8
(1, 1)
(2, 2)
(3, 3)
(4, 2)
(5, 3)
(6, 2)
(7, 0)
(8, 1)

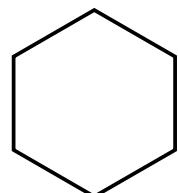


Where are we?

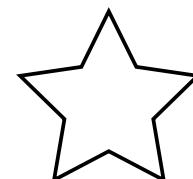
- Closest pair
- Geometry sweeping
- Geometric preliminaries
- Some basics geometry algorithms
 - Convex hull problem
 - Diameter of a set of points
 - Intersection of line segments

Geometry preliminaries

- Point: $p(x,y)$
- Line segment: $((x_1,y_1),(x_2,y_2))$
- **Convex** polygon: A polygon P is **convex** if the line segment connecting any two points in P lies ENTIRELY in P .

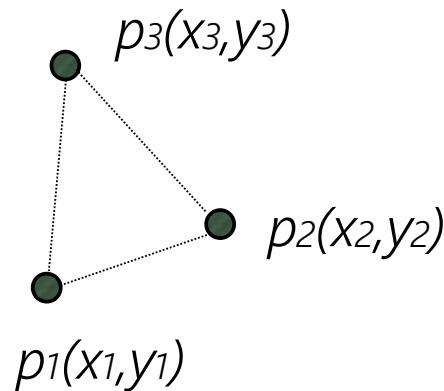


convex

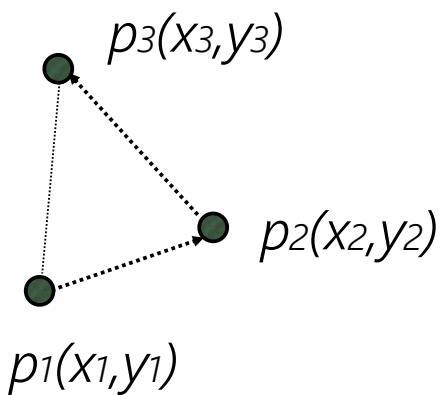


concave

Left turn and right turn

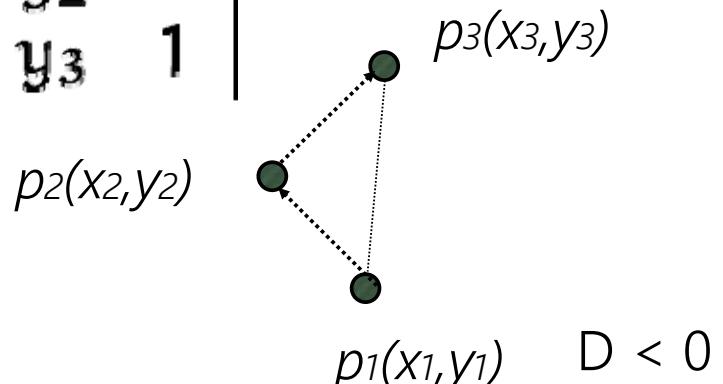


Area = ?



$$D = \begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix}$$

$D > 0$



$p_1(x_1, y_1) \quad D < 0$

Left turn and right turn

- To determine whether a point is under or above a line segment.
- To determine if two line segments intersect.

Where are we?

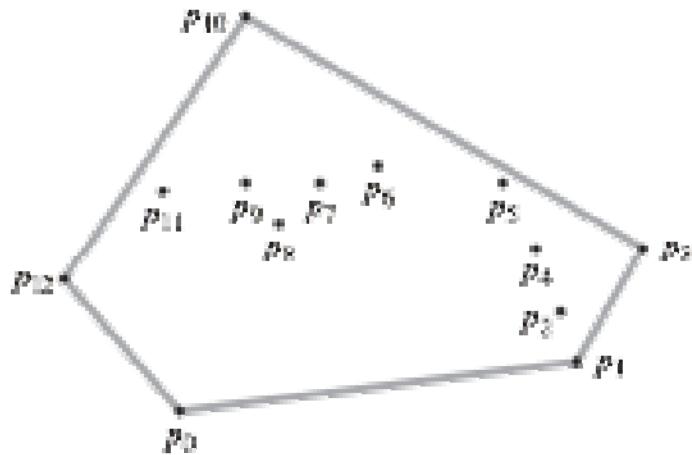
- Closest pair
- Geometry sweeping
- Geometric preliminaries
- Some basics geometry algorithms
 - Convex hull problem
 - Diameter of a set of points
 - Intersection of line segments

Convex hull

Problem: ConvexHull

Input: a set $S=\{p_1, p_2, \dots, p_n\}$ of n points in the plane,

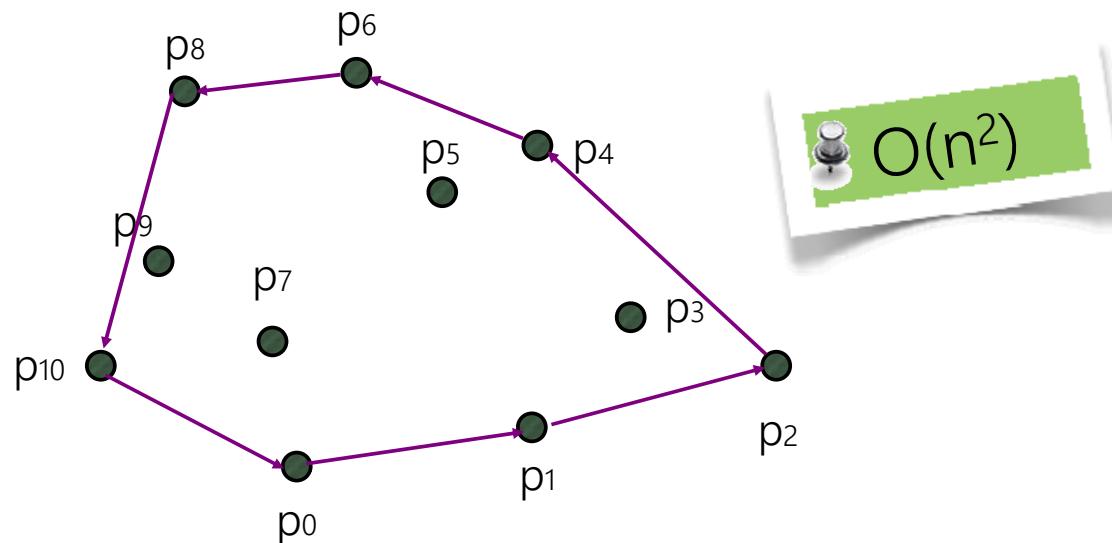
Output: $\text{CH}(S)$



- Graham scan
- Jarvis' march (gift-wrapping algorithm)
- Quick-hull

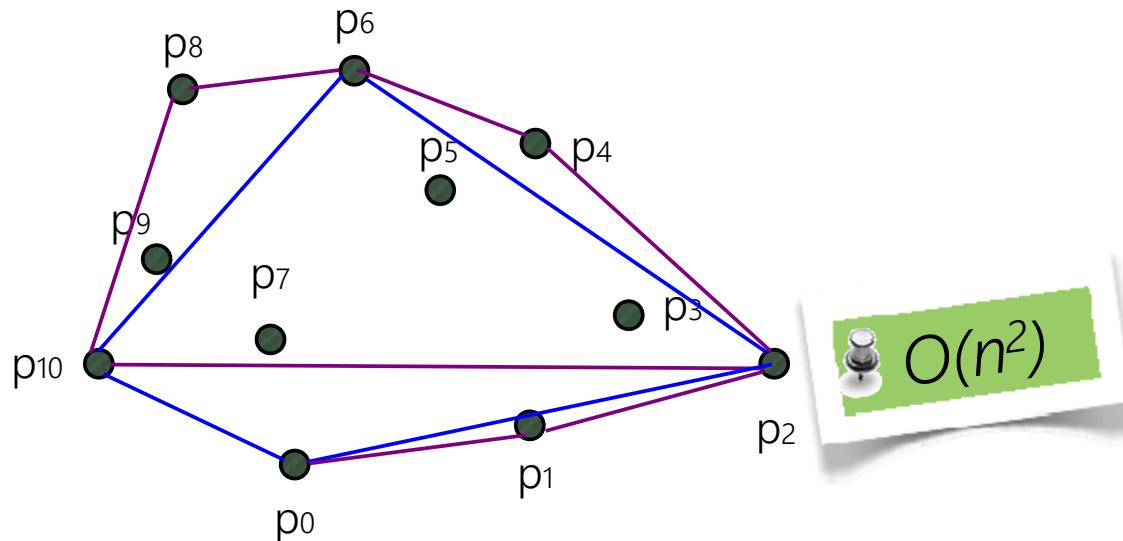
Jarvis' march

- Start at some *extreme point*, which is guaranteed to be on the hull.
- At each step, test each of the points, and find the one *which makes the largest right-hand turn*. That point has to be the next one on the hull.



Quick hull

- Given a chord $p_i p_j$ of the convex hull
- Find the farthest one p_k from the chord
- The points inside the triangle $p_i p_j p_k$ cannot be on the hull. Put the points which lie outside edge $p_i p_k$ in set s_1 , and points outside edge $p_j p_k$ in set s_2 .
- Recursively solve the problem of s_1 and s_2 .

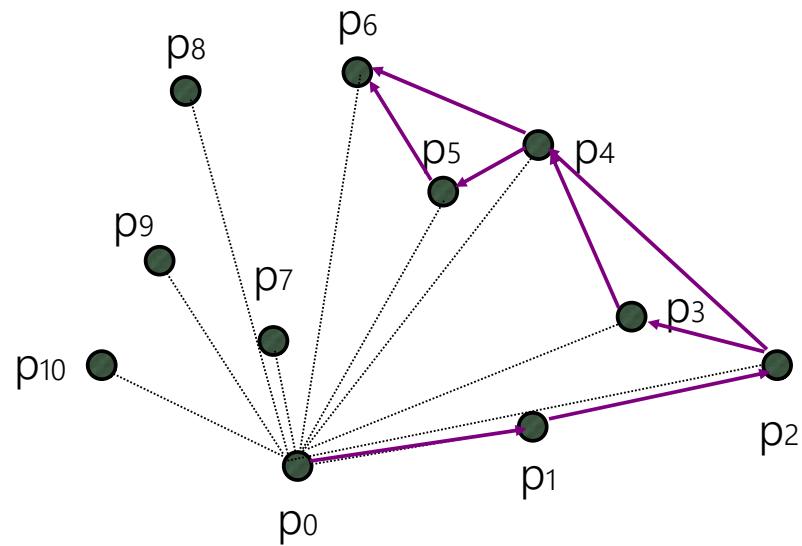


<http://www.cs.princeton.edu/courses/archive/fall10/cos226/demo/ah/ConvexHull.html>

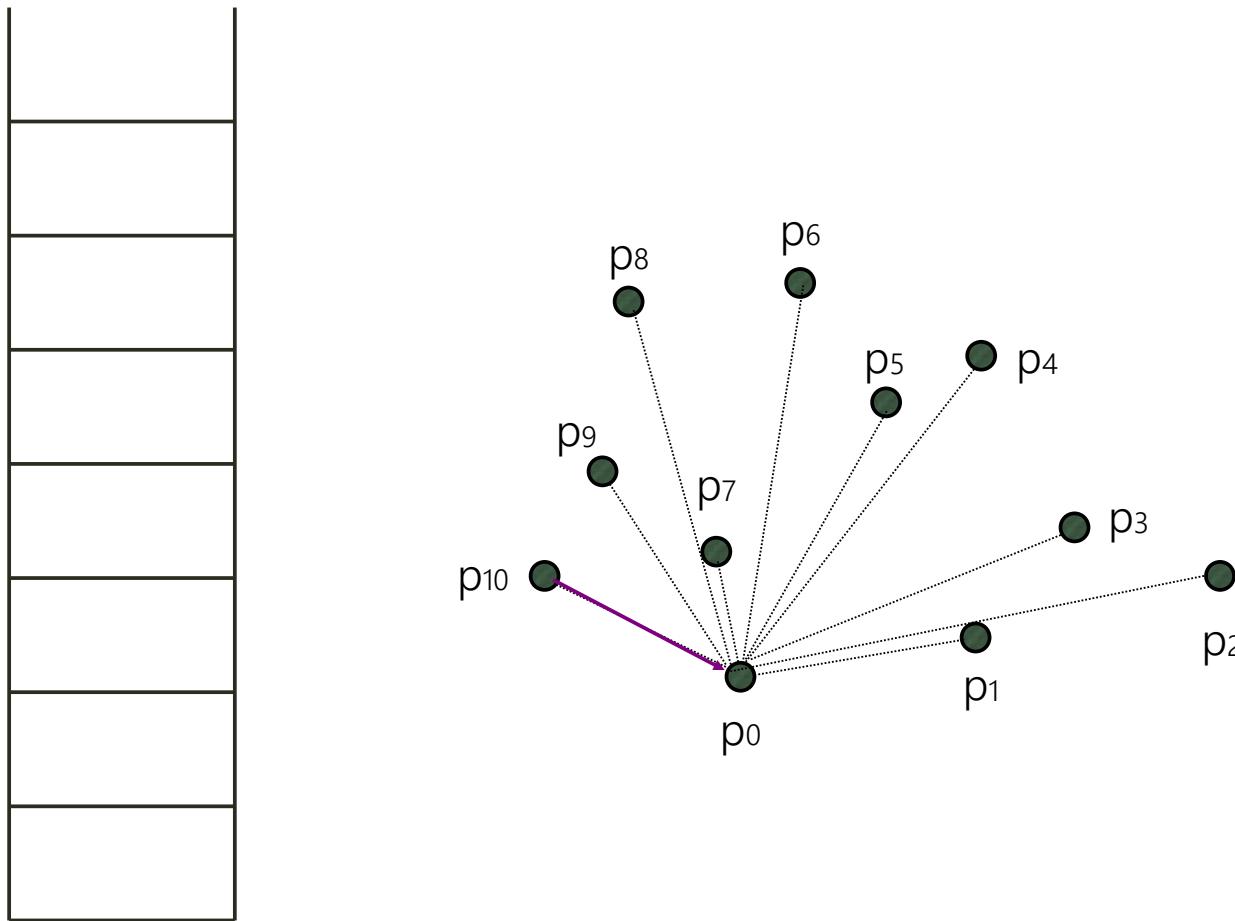
Graham Scan



Ronald Graham
(source from Wikipedia)

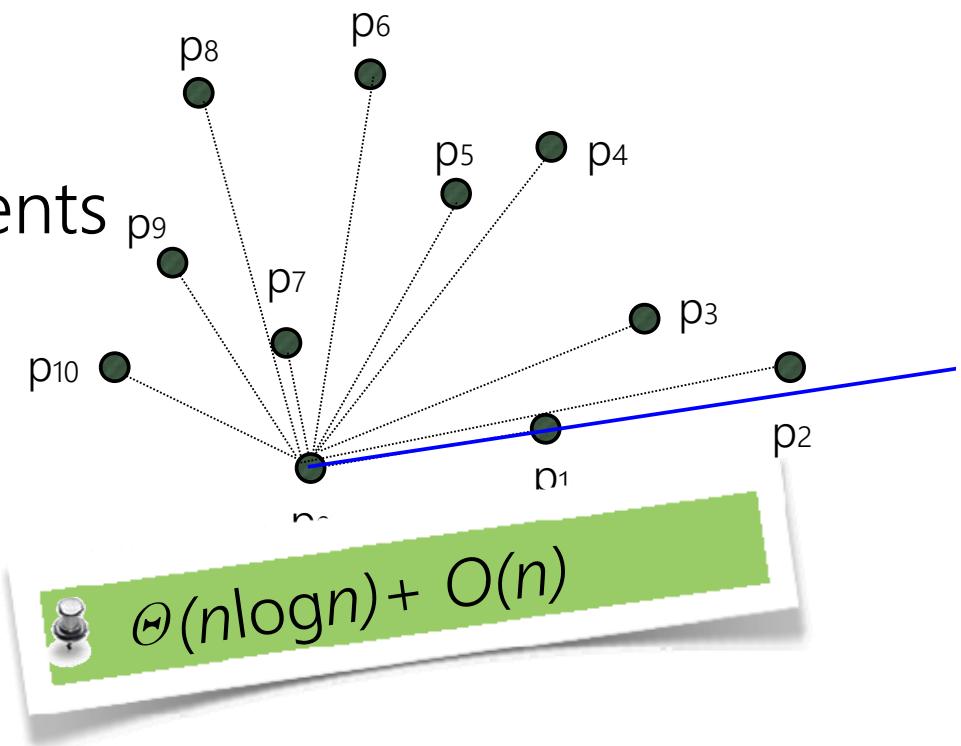


Graham scan



Geometric sweeping

- Sweeping line and Events
- How to handle events
- Solve the problem



Algorithm 18.3 CONVEXHULL

Input: A set of n points in the plane.

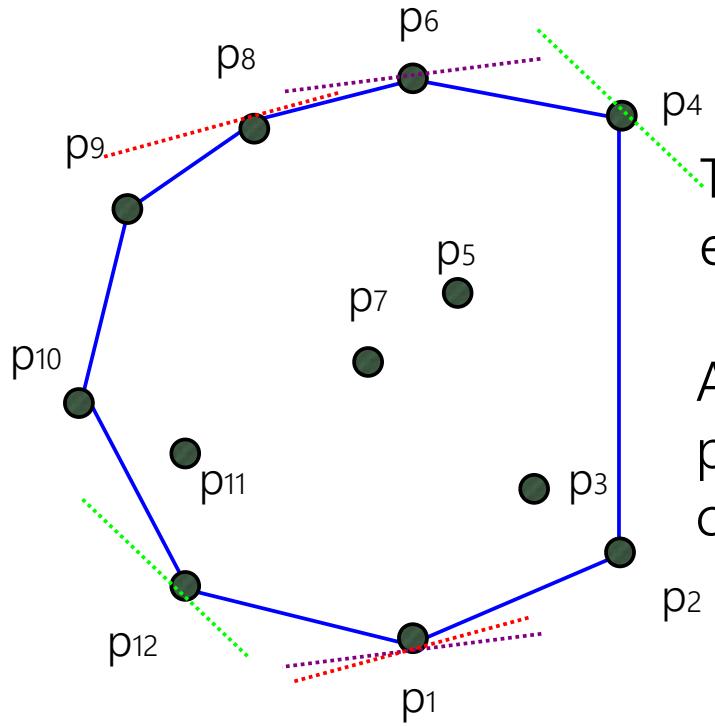
Output: $CH(S)$, the convex hull of S stored in a stack St .

1. Let p_0 be the rightmost point with minimum y -coordinate.
2. $E[0] \leftarrow p_0$
3. Let $E[1\dots n-1]$ be the points in $S \setminus \{p_0\}$ stored in increasing polar angle about p_0 . If two points p_i and p_j form the same angle with p_0 , then the one that is closer to p_0 precedes the other in the ordering.
4. push ($St, E[n-1]$); push ($St, E[0]$)
5. $k \leftarrow 1$
6. while $k < n-1$
7. Let $St = (E[n-1], \dots, E[i], E[j])$, $E[j]$ is on top of the stack
8. if $E[i], E[j], E[k]$ is a left turn then
9. push ($St, E[k]$)
10. $k \leftarrow k + 1$
11. else pop (St)
12. end if
13. end while

Where are we?

- Closest pair
- Geometry sweeping
- Geometric preliminaries
- Some basics geometry algorithms
 - Convex hull problem
 - Diameter of a set of points
 - Intersection of line segments

The diameter of a set of points



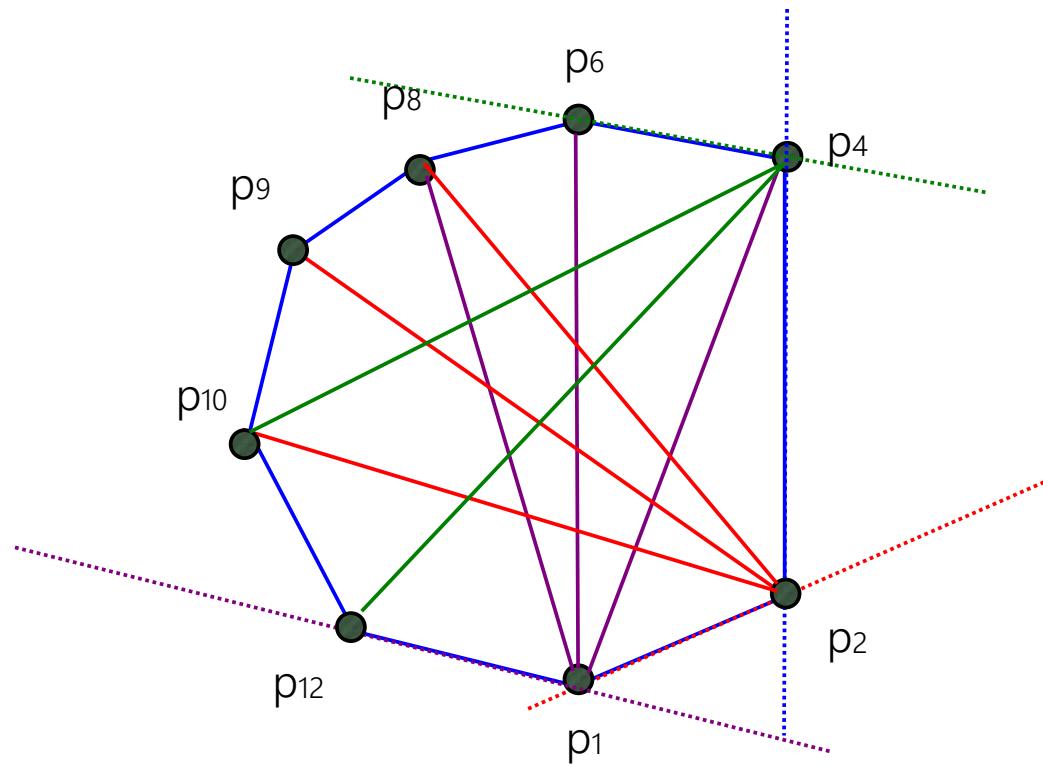
The diameter of a set P of points equals to the diameter of $CH(P)$.

Any two points that admit two parallel supporting lines are called *antipodal pair*.

Any pair of vertices realizing the diameter in a convex polygon is *an antipodal pair*.

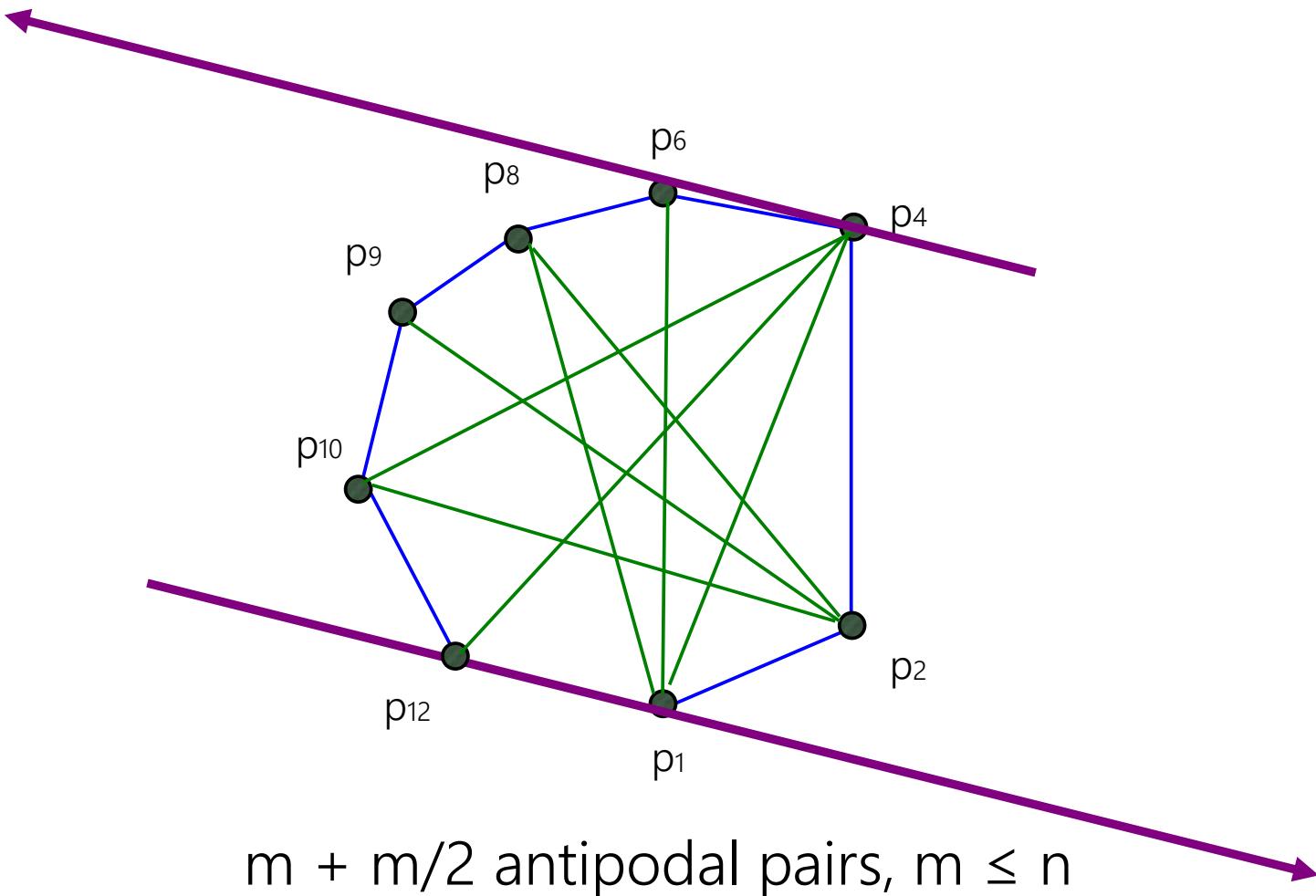
How to find antipodal pairs?

The diameter of a set of points



How many antipodal pairs?

Rotating caliper



Algorithm 18.4 DIAMETER

Input: A set of n points in the plane.

Output: $\text{Diam}(S)$, the diameter of S .

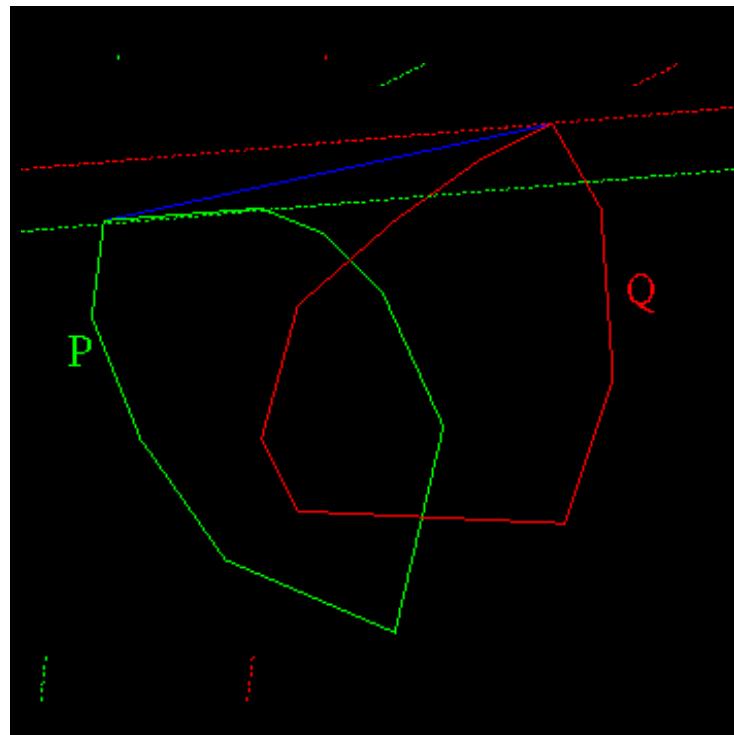
1. $\{p_1, p_2, \dots, p_m\} \leftarrow \text{CH}(S)$
2. $A \leftarrow \{\}$
3. $k \leftarrow 2$
4. **while** $\text{dist}(p_m, p_1, p_{k+1}) > \text{dist}(p_m, p_1, p_k)$
5. $k \leftarrow k + 1$
6. **end while**
7. $i \leftarrow 1; j \leftarrow k$

8. **while** $i \leq k$ and $j \leq m$
9. $A \leftarrow A \cup \{(p_i, p_j)\}$
10. **while** $\text{dist}(p_i, p_{i+1}, p_{j+1}) > \text{dist}(p_i, p_{i+1}, p_j)$ and $j < m$
11. $A \leftarrow A \cup \{(p_i, p_j)\}$
12. $j \leftarrow j + 1$
13. **end while**
14. $i \leftarrow i + 1$
15. **end while**
16. Scan A to obtain an antipodal pair (p_r, p_s) with maximum separation
17. **return** the distance between p_r and p_s



$\Theta(n \log n)$

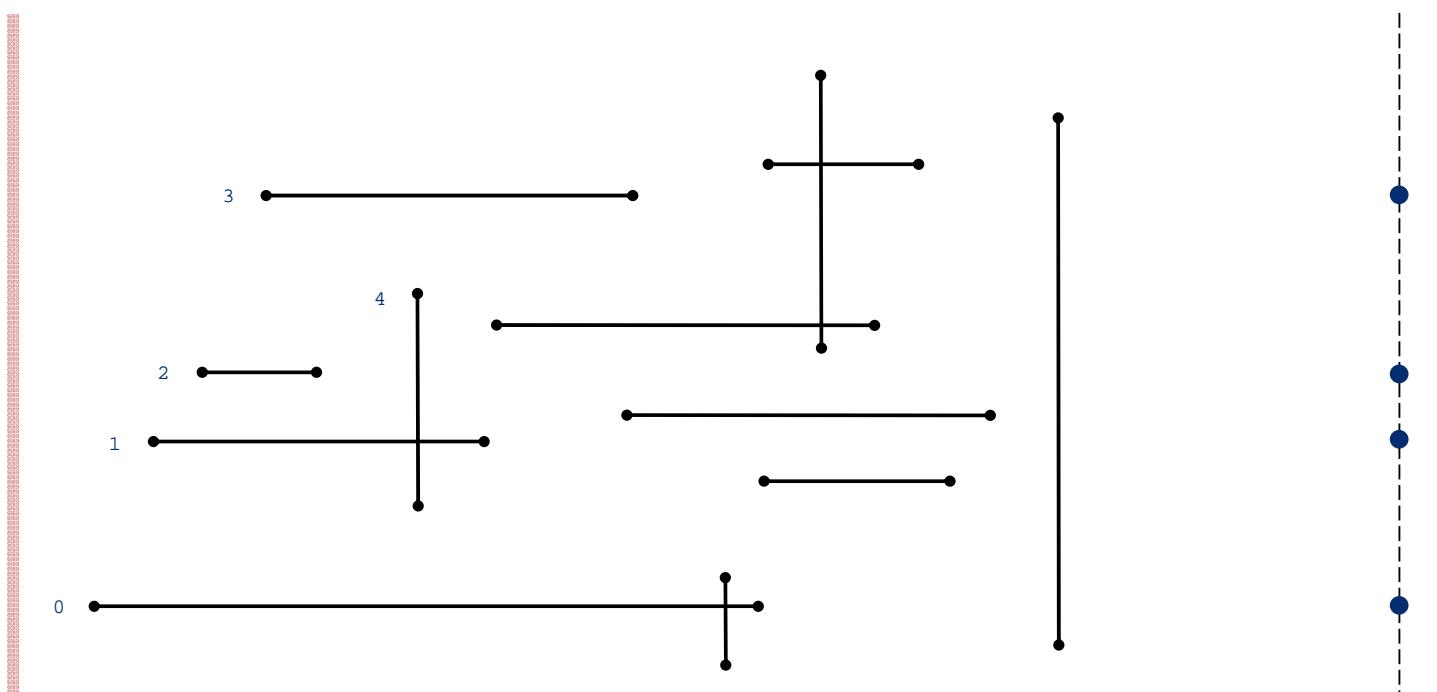
Union of 2 convex polygons



Intersection of orthogonal line segments

Sweep vertical line from left to right.

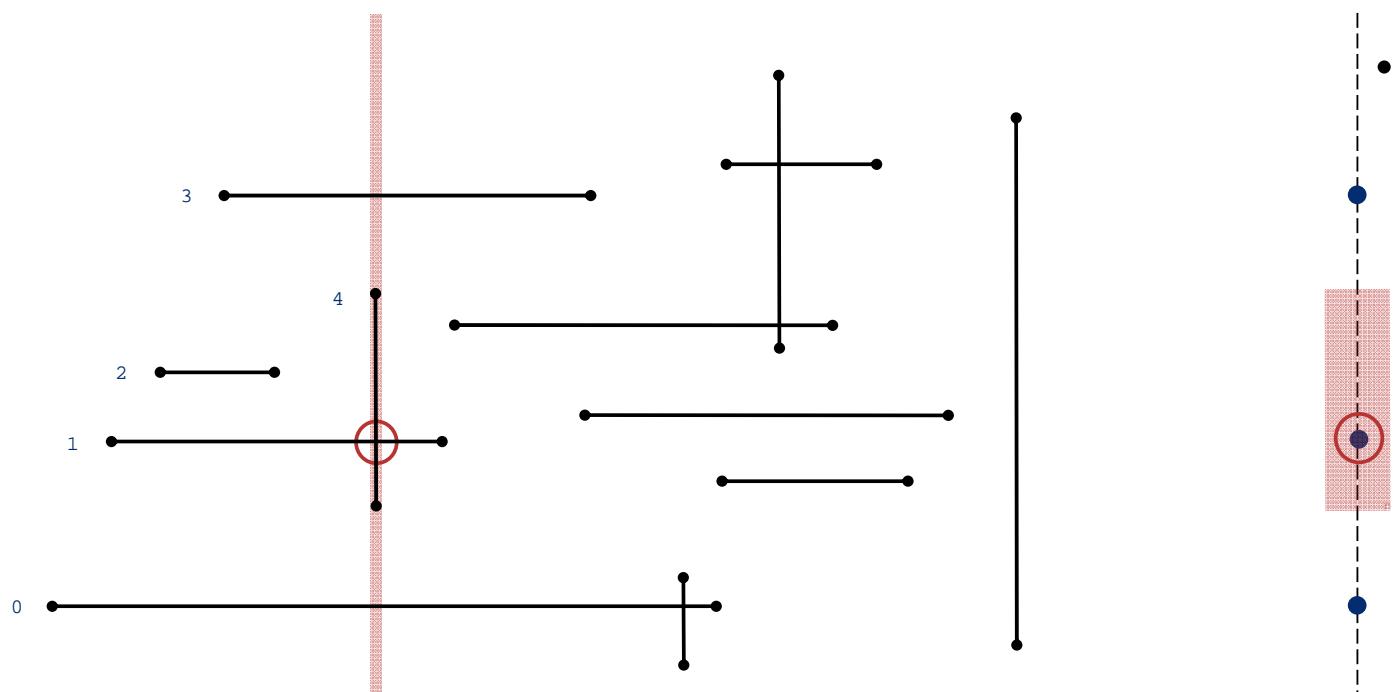
- x -coordinates define events.
- h -segment (left endpoint): insert y -coordinate into BST.
- h -segment (right endpoint): remove y -coordinate from BST.
- v -segment: range search for interval of y -endpoints.



Intersection of orthogonal line segments

Sweep vertical line from left to right.

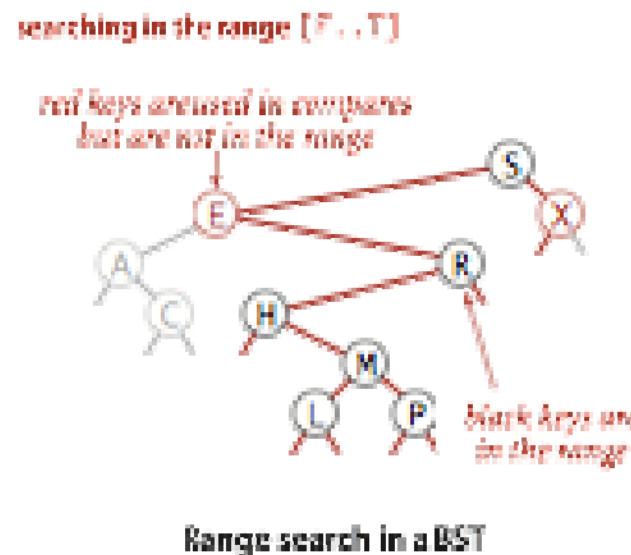
- x -coordinates define events.
- h -segment (left endpoint): insert y -coordinate into BST.
- h -segment (right endpoint): remove y -coordinate from BST.
- v -segment: range search for interval of y -endpoints.



Intersection of orthogonal line segments

Review: Binary Search Tree

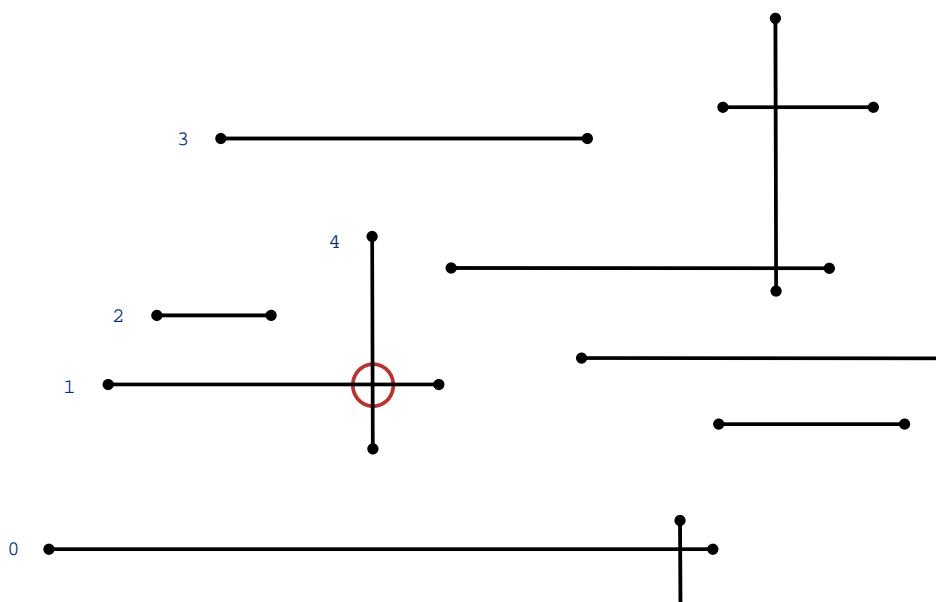
1d range search. Find all keys between lo and hi .



Proposition. Running time is proportional to $R + \log N$ (assuming BST is balanced).

Pf. Nodes examined = search path to lo + search path to hi + matching keys.

Intersection of orthogonal line segments

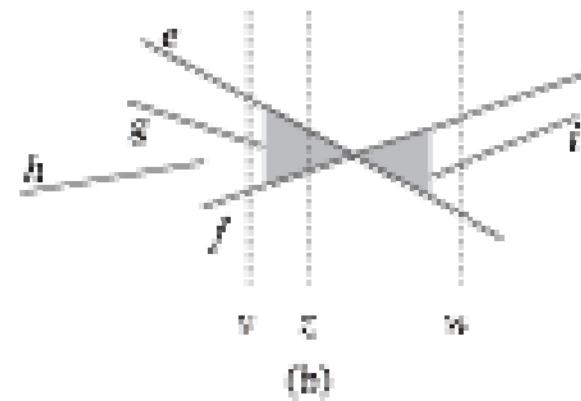
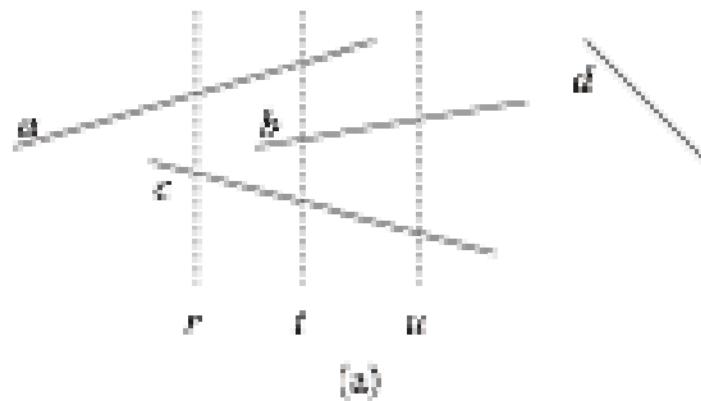


$\Theta(n \log n + m)$

Intersections of line segments

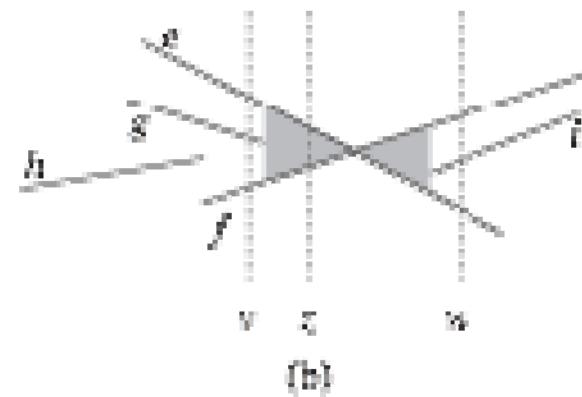
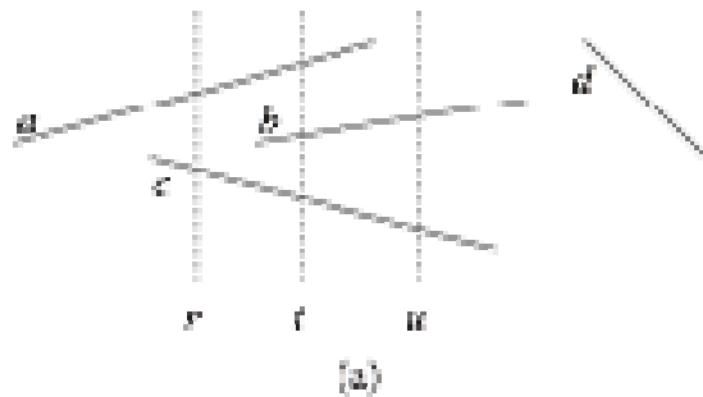
Problem: Intersections

Input: a set $L = \{l_1, l_2, \dots, l_n\}$ of n line segments in the plane,
Output: the set of points in which they intersect.



Assume that NO three line segments intersect at the same point.

Intersections of line segments

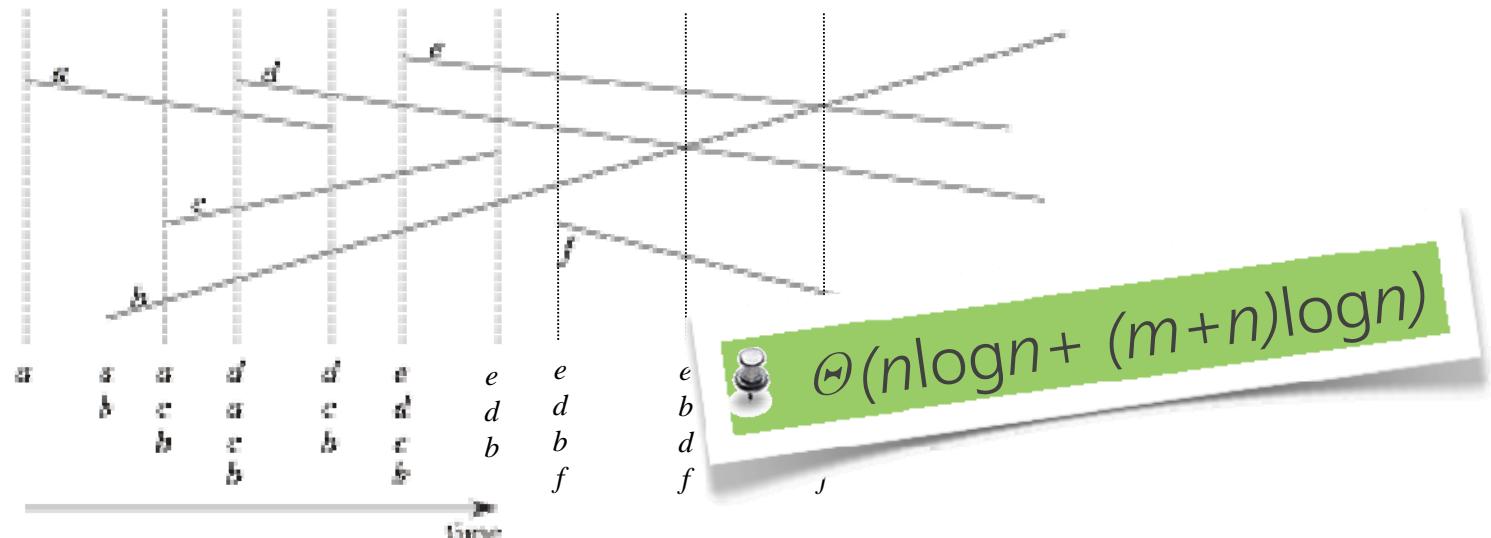


1. If a and b intersects, then at some time (before the intersection point), a and b are adjacent on the sweeping line .
2. When the sweeping line passes the intersection point of a,b, then the order of a, b will be reversed.

Intersections of line segments

Sweep endpoints and intersect points from left to right

- left endpoints of l , *insert l*, and see whether l intersects with $\text{above}(l)$ and $\text{below}(l)$; if so, add the *intersect points*.
- right endpoints of l , *remove l*, and see whether $\text{above}(l)$ and $\text{below}(l)$ intersect; if so, add the intersect point;
- intersect point of (l, l') ; see whether l and $\text{below}(l')$; l' and $\text{above}(l)$ intersect; if so, add the intersect point; *swap (l, l')* .



Intersections of line segments

Algorithm 18.2 INTERSECTIONSLS

Input: A set $L = \{l_1, \dots, l_n\}$ of n line segments in the plane.

Output: The intersection points of the line segments in L .

1. Sort the endpoints in nondecreasing order of their x -coordinates and insert them into a heap E (the event point schedule). Let S be the empty heap.
2. while E is not empty
3. $p \leftarrow \text{deletemin}(E)$
4. if p is a left endpoint then
5. let l be the line segment whose left endpoint is p
6. $\text{insert}(l, S)$
7. $l_1 \leftarrow \text{above}(l, S)$
8. $l_2 \leftarrow \text{below}(l, S)$
9. if l intersects l_1 at point q_1 then $\text{process}(q_1)$
10. if l intersects l_2 at point q_2 then $\text{process}(q_2)$

Intersections of line segments

```
11. else if  $p$  is a right endpoint then
12.   let  $l$  be the line segment whose right endpoint is  $p$ 
13.    $h_1 \leftarrow \text{above}(l, S)$ 
14.    $h_2 \leftarrow \text{below}(l, S)$ 
15.    $\text{delete}(l, S)$ 
16.   if  $h_1$  intersects  $h_2$  at  $q$  to the left of  $p$ 
17.     else  $\{p$  is an intersection point
18.       Let the two intersecting line segments at  $p$  be  $l_1$  and  $l_2$ 
19.       where  $l_1$  is above  $l_2$  to the left of  $p$ 
20.        $h_3 \leftarrow \text{above}(l_1, S)$ 
21.        $h_4 \leftarrow \text{below}(l_2, S)$ 
22.       if  $l_2$  intersects  $h_3$  at point  $q_1$  then  $\text{process}(q_1)$ 
23.       if  $l_1$  intersects  $h_4$  at point  $q_2$  then  $\text{process}(q_2)$ 
24.       interchange the ordering of  $l_1$  and  $l_2$  in  $S$ 
25.     end if
26.   end while
```

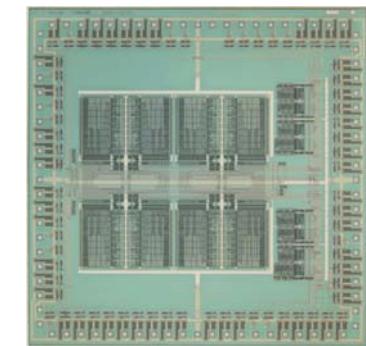
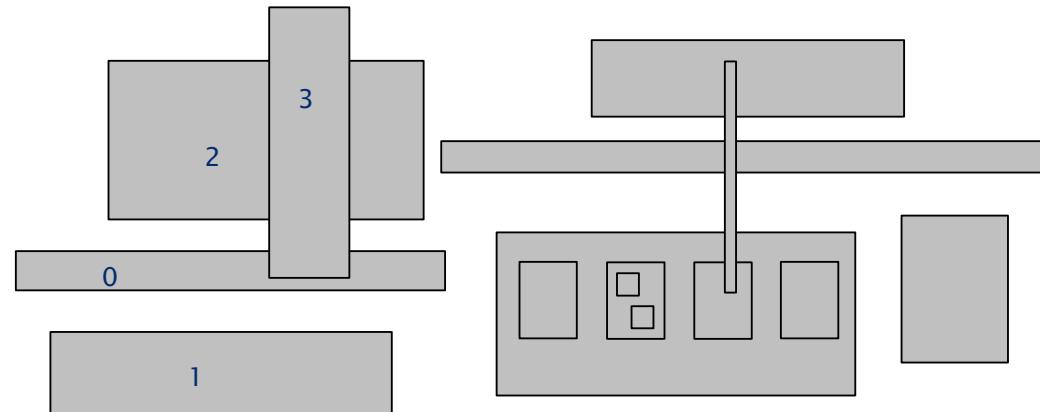


$\Theta(n \log n + (m+n) \log n)$

Where are we?

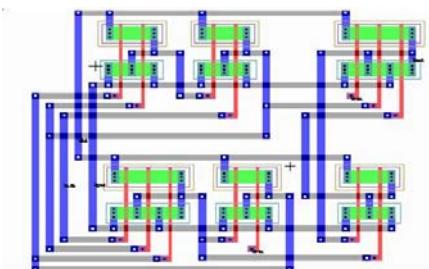
- Diameter of a set of points
- Intersection of line segments
- Intersection of rectangles
- Nearest neighbor search

Intersection of rectangles



Microprocessor design became a **geometric** problem.

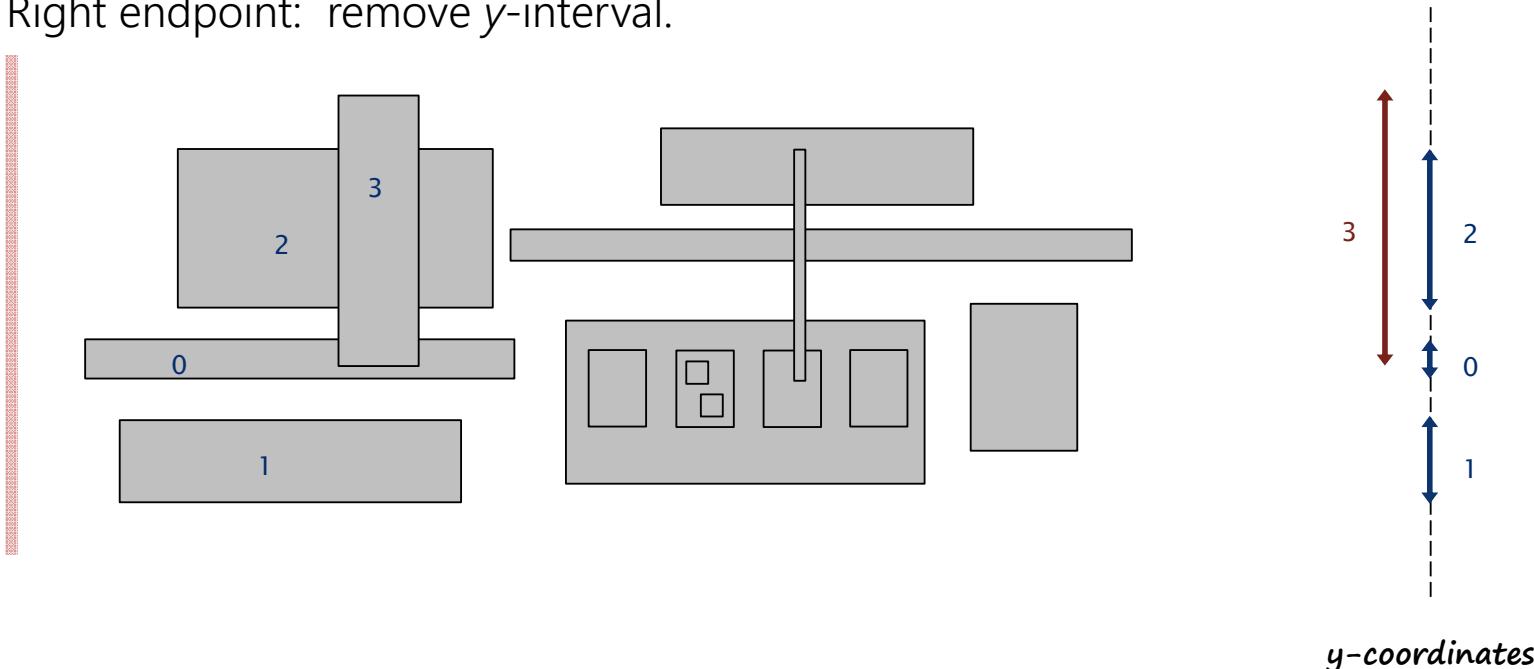
- Certain wires cannot intersect.
- Certain spacing needed between different types of wires.
- Debugging = orthogonal rectangle intersection search.



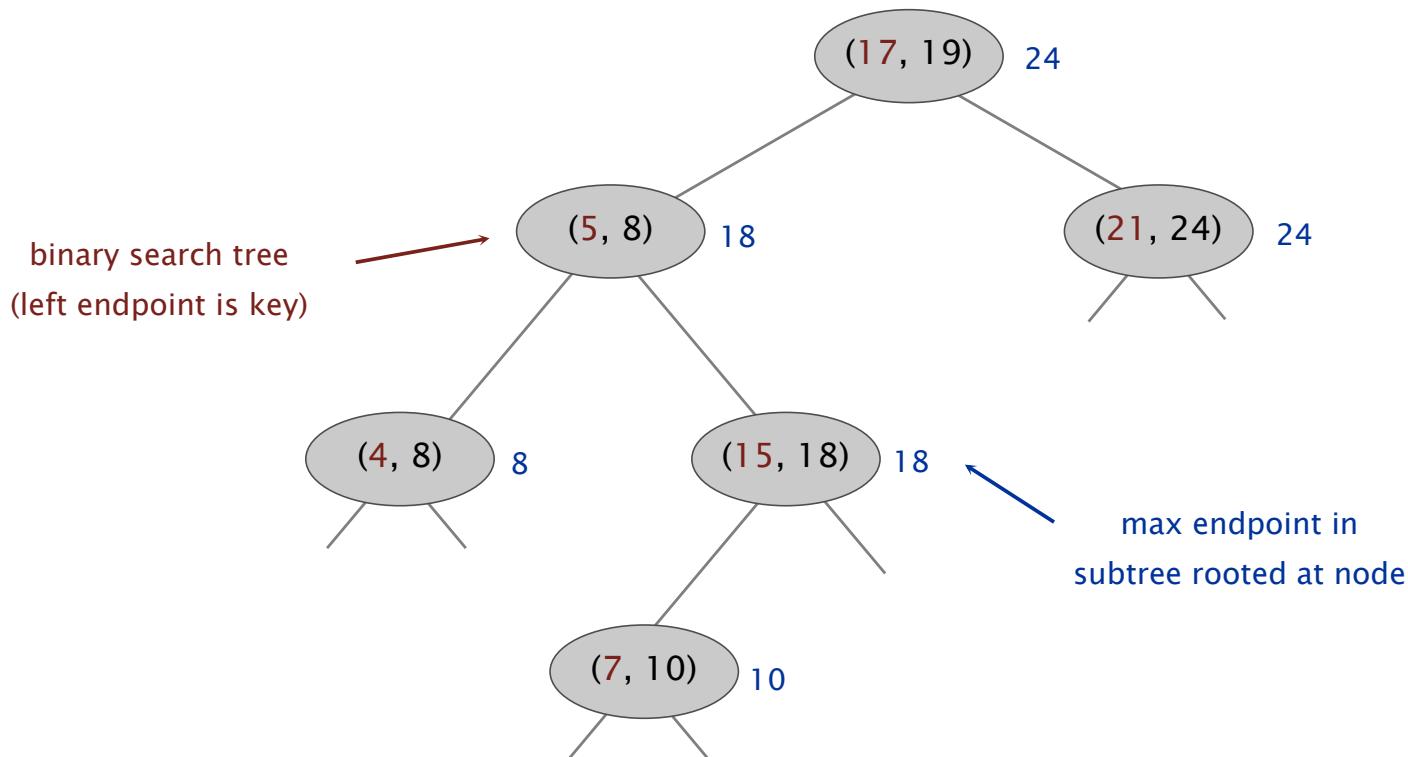
Intersection of rectangles

Sweep vertical line from left to right.

- x-coordinates of left and right endpoints define events.
- Maintain set of rectangles that intersect the sweep line in an interval search tree (using y-intervals of rectangle).
- Left endpoint: interval search for y-interval of rectangle; insert y-interval.
- Right endpoint: remove y-interval.



Interval search tree



Insertion = BST insertion

Interval search tree

Search (lo , hi) :

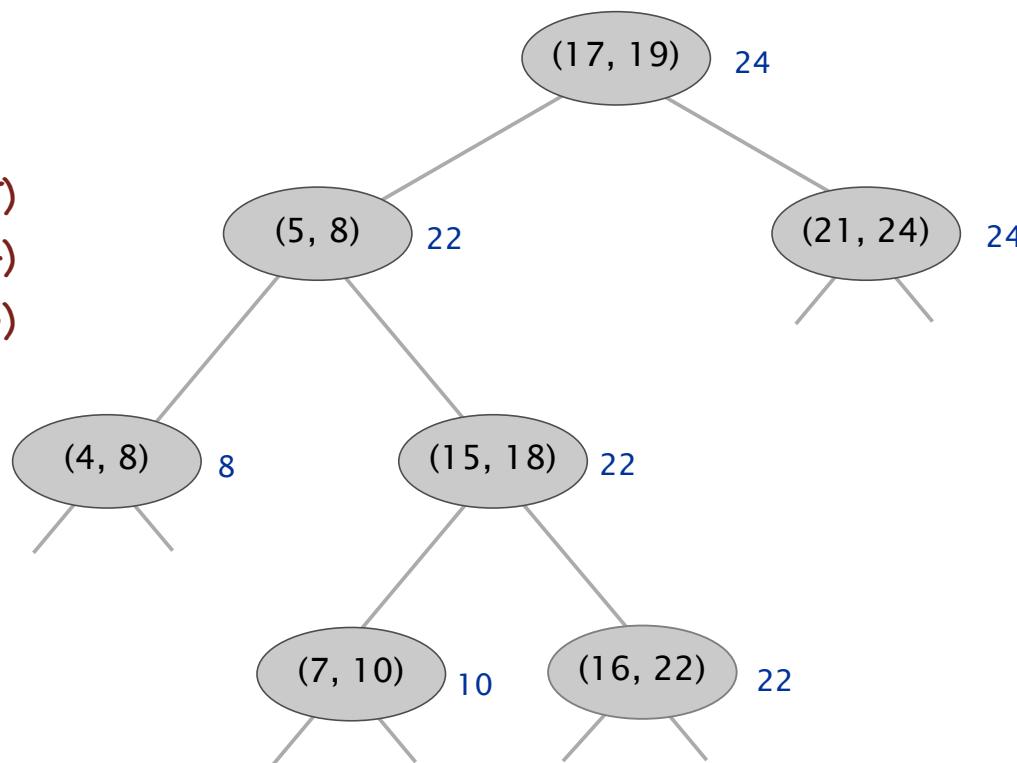
- If interval in node intersects query interval, return it.
- If left subtree is null, go right.
- If max endpoint in left subtree is less than lo , go right.
- Else go left.

interval intersection

search for $(23, 25)$

$(12, 14)$

$(21, 23)$

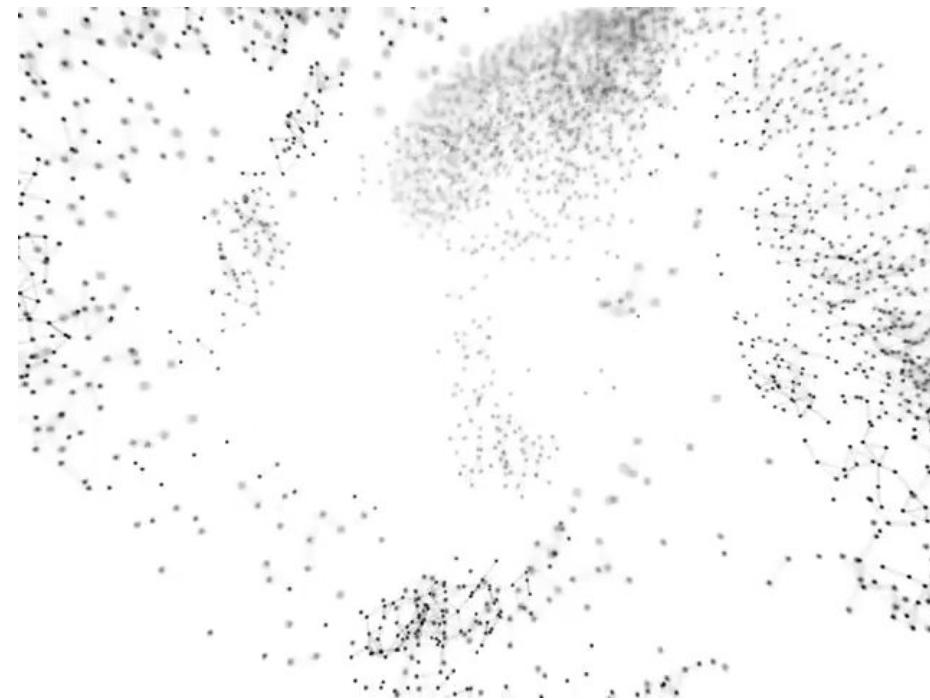


Flocking birds



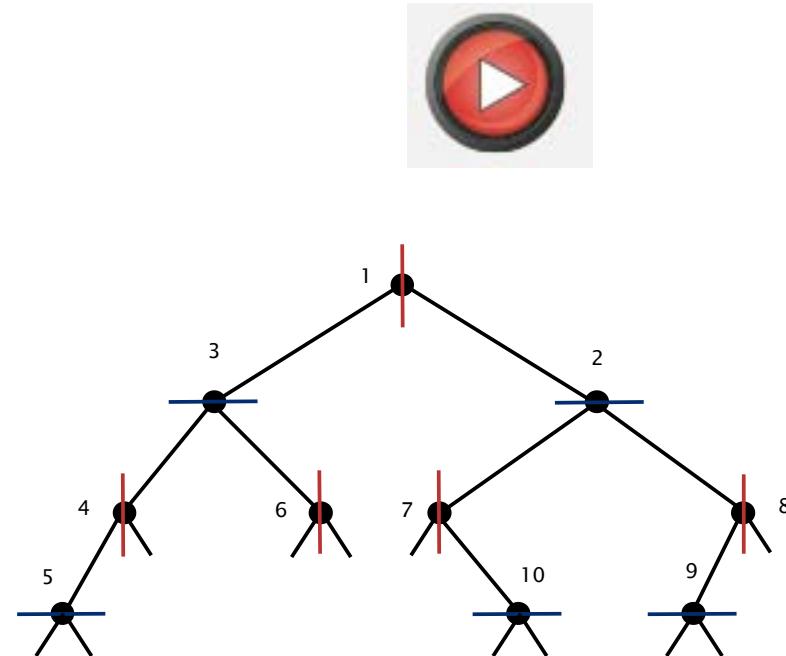
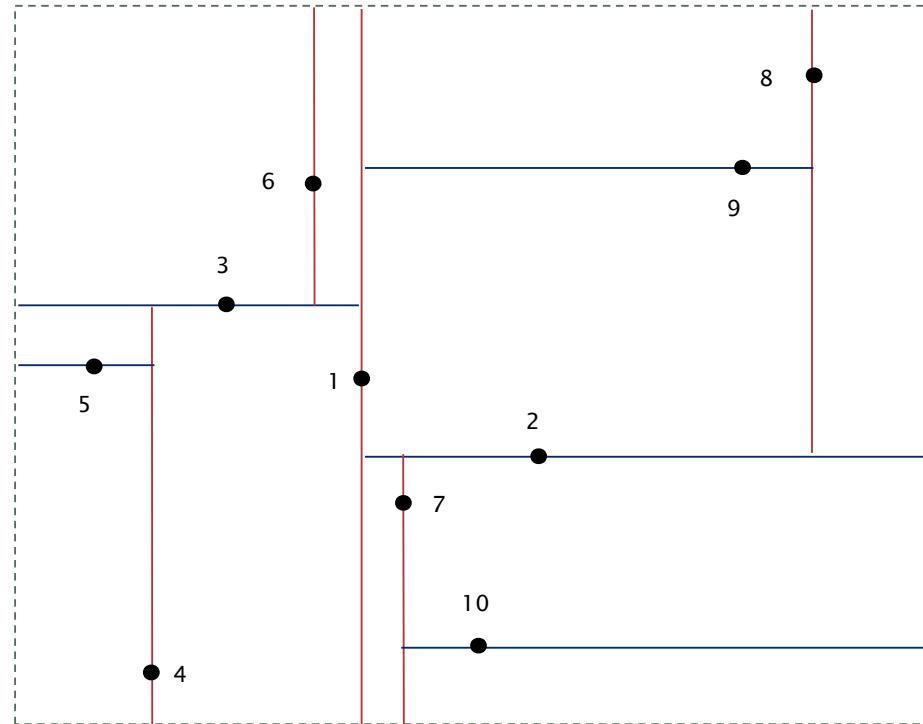
<http://www.youtube.com/watch?v=XH-groCeKbE>

Simulation of Flocking birds



2d-tree

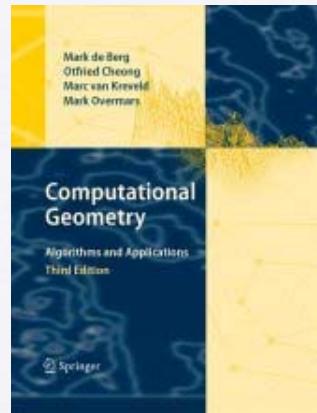
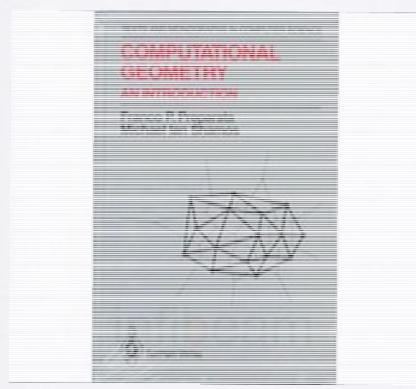
Recursively partition plane into two halfplanes.



Conclusion

- Computational geometry is devoted to the study of algorithms which can be stated in terms of geometry.
- Computational complexity: the difference between $O(n^2)$ and $O(n \log n)$ may be the difference between days and seconds of computation.
- Applications include: CAD/CAM, GIS, Integrated Circuit design, Computer vision,

References



- [1] [Franco P. Preparata](#) and Michael Ian Shamos.
Computational Geomtry - An Introduction.
Springer-Verlag.
- [2] Mark de Berg, [Otfried Cheong](#), [Marc van Kreveld](#),
[Mark Overmars](#).
Computational Geomtry: Algorithms and Applications
Third Edition (March 2008) Springer-Verlag.

Lecture 16 Strings

- String Sorts
- Tries
- Substring Search: KMP, BM, RK

String Sorts

- key-indexed counting
- LSD radix sort
- MSD radix sort

ACKNOWLEDGEMENTS: <http://algs4.cs.princeton.edu>

String Sorts

- key-indexed counting
 - LSD radix sort
 - MSD radix sort

Review: sorting algorithms

algorithm	guarantee	random	extra space	stable?	operations on keys
insertion sort	$\frac{1}{2} N^2$	$\frac{1}{4} N^2$	1	✓	<code>compareTo()</code>
mergesort	$N \lg N$	$N \lg N$	N	✓	<code>compareTo()</code>
quicksort	$1.39 N \lg N$ *	$1.39 N \lg N$	$c \lg N$		<code>compareTo()</code>
heapsort	$2 N \lg N$	$2 N \lg N$	1		<code>compareTo()</code>

* probabilistic

Lower bound. $\sim N \lg N$ compares required by any compare-based algorithm.

Key-indexed counting assumptions

Assumption. Keys are integers between 0 and $R - 1$.

Implication. Can use key as an array index.

Applications.

- Sort string by first letter.
- Sort class roster by section.
- Sort phone numbers by area code.
- Subroutine in a sorting algorithm.

input name	section	sorted result (by section)
Anderson	2	Harris
Brown	3	Martin
Davis	3	Moore
Garcia	4	Anderson
Harris	1	Martinez
Jackson	3	Miller
Johnson	4	Robinson
Jones	3	White
Martin	1	Brown
Martinez	2	Davis
Miller	2	Jackson
Moore	1	Jones
Robinson	2	Taylor
Smith	4	Williams
Taylor	3	Garcia
Thomas	4	Johnson
Thompson	4	Smith
White	2	Thomas
Williams	3	Thompson
Wilson	4	Wilson

*↑
keys are
small integers*

Key-indexed counting demo

Goal. Sort an array $a[]$ of N integers between 0 and $R - 1$.

- Count frequencies of each letter using key as index.
- Compute frequency cumulates which specify destinations.
- Access cumulates using key as index to move items.
- Copy back into original array.

```
int N = a.length;
int[] count = new int[R+1];

for (int i = 0; i < N; i++)
    count[a[i]+1]++;

for (int r = 0; r < R; r++)
    count[r+1] += count[r];

for (int i = 0; i < N; i++)
    aux[count[a[i]]++] = a[i];

for (int i = 0; i < N; i++)
    a[i] = aux[i];
```

i	a[i]	
0	d	
1	a	use a for 0
2	c	b for 1
3	f	c for 2
4	f	d for 3
5	b	e for 4
6	d	f for 5
7	b	
8	f	
9	b	
10	e	
11	a	

Key-indexed counting demo

Goal. Sort an array $a[]$ of N integers between 0 and $R - 1$.

- Count frequencies of each letter using key as index.
- Compute frequency cumulates which specify destinations.
- Access cumulates using key as index to move items.
- Copy back into original array.

```
int N = a.length;
int[] count = new int[R+1];

count frequencies → for (int i = 0; i < N; i++)
    count[a[i]+1]++;

for (int r = 0; r < R; r++)
    count[r+1] += count[r];

for (int i = 0; i < N; i++)
    aux[count[a[i]]++] = a[i];

for (int i = 0; i < N; i++)
    a[i] = aux[i];
```

i	a[i]	offset by 1 [stay tuned]	r count[r]
0	d		
1	a		
2	c		
3	f	0	
4	f	2	
5	b	3	
6	d	1	
7	b	2	
8	f	1	
9	b	3	
10	e		
11	a		

Key-indexed counting demo

Goal. Sort an array $a[]$ of N integers between 0 and $R - 1$.

- Count frequencies of each letter using key as index.
- Compute frequency cumulates which specify destinations.
- Access cumulates using key as index to move items.
- Copy back into original array.

```
int N = a.length;
int[] count = new int[R+1];

for (int i = 0; i < N; i++)
    count[a[i]+1]++;

for (int r = 0; r < R; r++)
    count[r+1] += count[r];

for (int i = 0; i < N; i++)
    aux[count[a[i]]++] = a[i];

for (int i = 0; i < N; i++)
    a[i] = aux[i];
```

compute cumulates →

i	a[i]	r	count[r]
0	d		
1	a		
2	c		
3	f	a	0
4	f	b	2
5	b	c	5
6	d	d	6
7	b	e	8
8	f	f	9
9	b		
10	e		
11	a		

6 keys < d, 8 keys < e
so d's go in a[6] and a[7]

Key-indexed counting demo

Goal. Sort an array $a[]$ of N integers between 0 and $R - 1$.

- Count frequencies of each letter using key as index.
- Compute frequency cumulates which specify destinations.
- Access cumulates using key as index to move items.
- Copy back into original array.

```
int N = a.length;
int[] count = new int[R+1];

for (int i = 0; i < N; i++)
    count[a[i]+1]++;

for (int r = 0; r < R; r++)
    count[r+1] += count[r];

move items → for (int i = 0; i < N; i++)
    aux[count[a[i]]++] = a[i];

for (int i = 0; i < N; i++)
    a[i] = aux[i];
```

i	a[i]	r count[r]		i	aux[i]
0	d			0	a
1	a			1	a
2	c			2	b
3	f	a	2	3	b
4	f	b	5	4	b
5	b	c	6	5	c
6	d	d	8	6	d
7	b	e	9	7	d
8	f	f	12	8	e
9	b	-	12	9	f
10	e			10	f
11	a			11	f

Key-indexed counting demo

Goal. Sort an array $a[]$ of N integers between 0 and $R - 1$.

- Count frequencies of each letter using key as index.
- Compute frequency cumulates which specify destinations.
- Access cumulates using key as index to move items.
- Copy back into original array.

```
int N = a.length;
int[] count = new int[R+1];

for (int i = 0; i < N; i++)
    count[a[i]+1]++;

for (int r = 0; r < R; r++)
    count[r+1] += count[r];

for (int i = 0; i < N; i++)
    aux[count[a[i]]++] = a[i];

copy back → for (int i = 0; i < N; i++)
    a[i] = aux[i];
```

i	a[i]	r	count[r]	i	aux[i]
0	a			0	a
1	a			1	a
2	b			2	b
3	b	a	2	3	b
4	b	b	5	4	b
5	c	c	6	5	c
6	d	d	8	6	d
7	d	e	9	7	d
8	e	f	12	8	e
9	f	-	12	9	f
10	f			10	f
11	f			11	f

Key-indexed counting analysis

Proposition. Key-indexed counting takes time proportional to $N+R$.

Proposition. Key-indexed counting uses extra space proportional to $N+R$.

Stable? Yes.

a[0]	Anderson	2	Harris	1	aux[0]
a[1]	Brown	3	Martin	1	aux[1]
a[2]	Davis	3	Moore	1	aux[2]
a[3]	Garcia	4	Anderson	2	aux[3]
a[4]	Harris	1	Martinez	2	aux[4]
a[5]	Jackson	3	Miller	2	aux[5]
a[6]	Johnson	4	Robinson	2	aux[6]
a[7]	Jones	3	White	2	aux[7]
a[8]	Martin	1	Brown	3	aux[8]
a[9]	Martinez	2	Davis	3	aux[9]
a[10]	Miller	2	Jackson	3	aux[10]
a[11]	Moore	1	Jones	3	aux[11]
a[12]	Robinson	2	Taylor	3	aux[12]
a[13]	Smith	4	Williams	3	aux[13]
a[14]	Taylor	3	Garcia	4	aux[14]
a[15]	Thomas	4	Johnson	4	aux[15]
a[16]	Thompson	4	Smith	4	aux[16]
a[17]	White	2	Thomas	4	aux[17]
a[18]	Williams	3	Thompson	4	aux[18]
a[19]	Wilson	4	Wilson	4	aux[19]

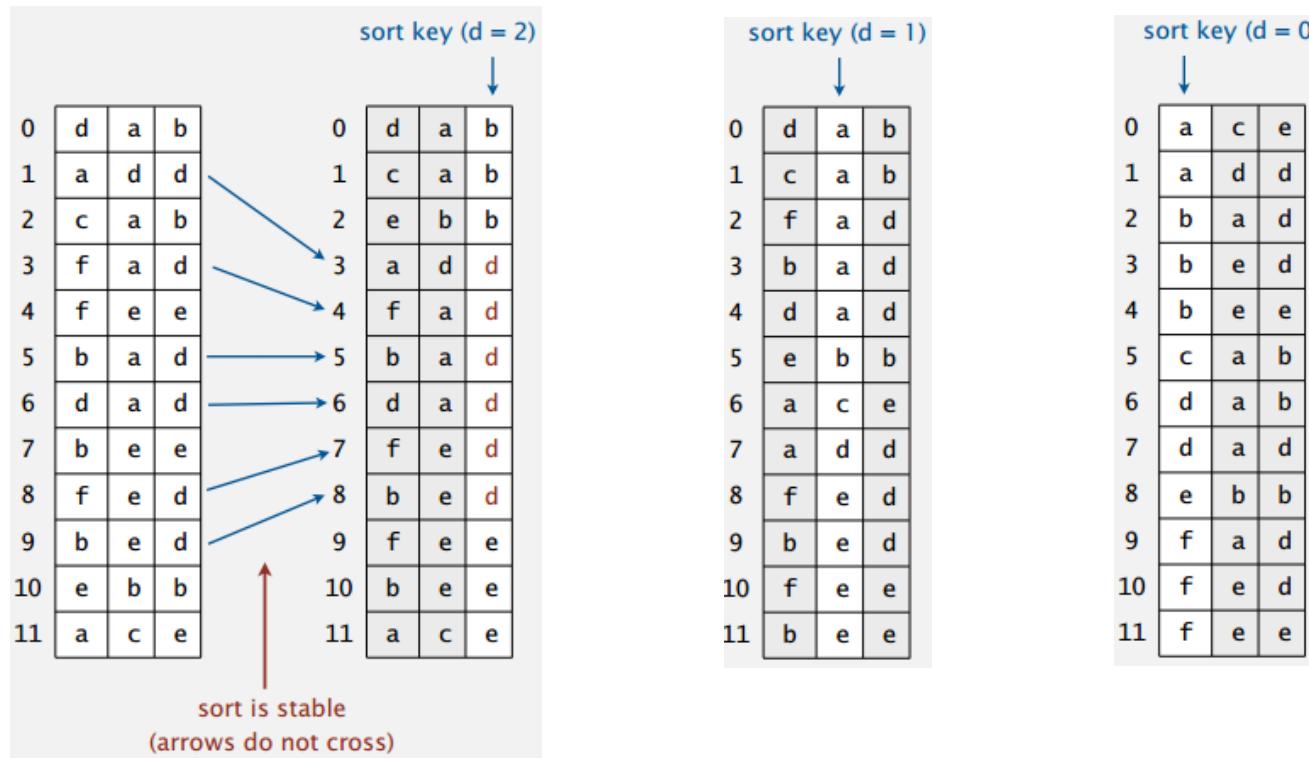
String Sorts

- key-indexed counting
- LSD radix sort
- MSD radix sort

Least-significant-digit-first string sort

LSD string (radix) sort.

- Consider characters from right to left.
- Stably sort using d_{th} character as the key (using key-indexed counting).



LSD string sort: correctness proof

Proposition. LSD sorts fixed-length strings in ascending order.

Pf. [by induction on i]

After pass i , strings are sorted by last i characters.

- If two strings differ on sort key, key-indexed sort puts them in proper relative order.
- If two strings agree on sort key, stability keeps them in proper relative order.

	sort key		
0	d	a	b
1	c	a	b
2	f	a	d
3	b	a	d
4	d	a	d
5	e	b	b
6	a	c	e
7	a	d	d
8	f	e	d
9	b	e	d
10	f	e	e
11	b	e	e

sorted from previous passes (by induction)

The diagram illustrates the state of an array after 11 passes of LSD sort. The array has 12 rows (0 to 11) and 4 columns (a, b, c, d). A blue arrow points down from row 2 to row 10, indicating the sort key for each row. A red arrow points up from row 11, indicating elements from previous passes.

Proposition. LSD sort is stable.

Pf. Key-indexed counting is stable.

Summary: sorting algorithms

algorithm	guarantee	random	extra space	stable?	operations on keys
insertion sort	$\frac{1}{2} N^2$	$\frac{1}{4} N^2$	1	✓	<code>compareTo()</code>
mergesort	$N \lg N$	$N \lg N$	N	✓	<code>compareTo()</code>
quicksort	$1.39 N \lg N$ *	$1.39 N \lg N$	$c \lg N$		<code>compareTo()</code>
heapsort	$2 N \lg N$	$2 N \lg N$	1		<code>compareTo()</code>
LSD sort †	$2 W(N + R)$	$2 W(N + R)$	$N + R$	✓	<code>charAt()</code>

* probabilistic
† fixed-length W keys

String Sorts

- key-indexed counting
- LSD radix sort
- MSD radix sort

Reverse LSD

- Consider characters from left to right.
- Stably sort using d_{th} character as the key (using key-indexed counting).

The diagram illustrates the Reverse LSD sorting process across three stages:

- sort key ($d = 0$)**: The first stage sorts by the least significant digit (the rightmost character). The input array (left) and sorted array (right) are:

0	d	a	b
1	a	d	d
2	c	a	b
3	f	a	d
4	f	e	e
5	b	a	d
6	d	a	d
7	b	e	e
8	f	e	d
9	b	e	d
10	e	b	b
11	a	c	e

0	a	d	d
1	a	c	e
2	b	a	d
3	b	e	e
4	b	e	d
5	c	a	b
6	d	a	b
7	d	a	d
8	e	b	b
9	f	a	d
10	f	e	e
11	f	e	d

- sort key ($d = 1$)**: The second stage sorts by the second least significant digit. The input array (left) and sorted array (right) are:

0	b	a	d
1	c	a	b
2	d	a	b
3	d	a	d
4	f	a	d
5	e	b	b
6	a	c	e
7	a	d	d
8	b	e	e
9	b	e	d
10	f	e	e
11	f	e	d

0	b	a	d
1	c	a	b
2	d	a	b
3	d	a	d
4	f	a	d
5	e	b	b
6	a	c	e
7	a	d	d
8	b	e	e
9	b	e	d
10	f	e	e
11	f	e	d

- sort key ($d = 2$)**: The third stage sorts by the most significant digit (the leftmost character). The input array (left) and sorted array (right) are:

0	c	a	b
1	d	a	b
2	e	b	b
3	b	a	d
4	d	a	d
5	f	a	d
6	a	d	d
7	b	e	d
8	f	e	d
9	a	c	e
10	b	e	e
11	f	e	e

0	c	a	b
1	d	a	b
2	e	b	b
3	b	a	d
4	d	a	d
5	f	a	d
6	a	d	d
7	b	e	d
8	f	e	d
9	a	c	e
10	b	e	e
11	f	e	e

not sorted!

Most-significant-digit-first string sort

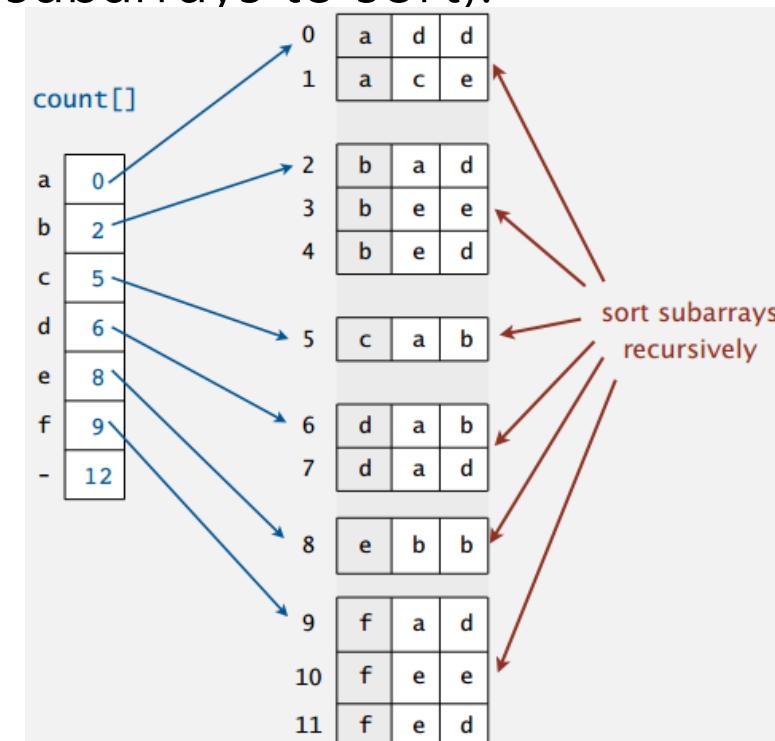
MSD string (radix) sort.

- Partition array into R pieces according to first character (use key-indexed counting).
- Recursively sort all strings that start with each character (key-indexed counts delineate subarrays to sort).

0	d	a	b
1	a	d	d
2	c	a	b
3	f	a	d
4	f	e	e
5	b	a	d
6	d	a	d
7	b	e	e
8	f	e	d
9	b	e	d
10	e	b	b
11	a	c	e

0	a	d	d
1	a	c	e
2	b	a	d
3	b	e	e
4	b	e	d
5	c	a	b
6	d	a	b
7	d	a	d
8	e	b	b
9	f	a	d
10	f	e	e
11	f	e	d

sort key



MSD string sort example

input		d								
she	are									
sells	by	to	by							
seashells	she	sells	seashells	sea	seashells	sea	seashells	seashells	seashells	seashells
by	sells	seashells	sea	seashells						
the	seashells	sea	seashells							
sea	sea	sells								
shore	shore	seashells	sells							
the	shells	she								
shells	she	shore								
she	sells	shells								
sells	surely	she								
are	seashells	surely								
surely	the	hi	the							
seashells	the									
need to examine every character in equal keys										
are	are	are	are	are	are	are	are	are	are	are
by	by	by	by	by	by	by	by	by	by	by
sea	sea	sea	sea	sea	sea	sea	sea	sea	sea	sea
seashells	seashells	seashells	seashells	seashells	seashells	seashells	seashells	seashells	seashells	seashells
seashells	seashells	seashells	seashells	seashells	seashells	seashells	seashells	seashells	seashells	seashells
sells	sells	sells	sells	sells	sells	sells	sells	sells	sells	sells
sells	sells	sells	sells	sells	sells	sells	sells	sells	sells	sells
she	she	she	she	she	she	she	she	she	she	she
shore	shore	shore	shore	shore	shore	shore	shore	shore	shore	shore
shells	shells	shells	shells	shells	shells	shells	shells	shells	shells	shells
she	she	she	she	she	she	she	she	she	she	she
surely	surely	surely	surely	surely	surely	surely	surely	surely	surely	surely
the	the	the	the	the	the	the	the	the	the	the
the	the	the	the	the	the	the	the	the	the	the
end of string goes before any char value										
output										
are										
by										
sea										
seashells										
seashells										
sells										
sells										
she										
she										
she										
she										
she										
shells										
shells										
shore										
shore										
shore										
surely										
surely										
the										
the										
the										

Trace of recursive calls for MSD string sort (no cutoff for small subarrays, subarrays of size 0 and 1 omitted)

Variable-length strings

Treat strings as if they had an extra char at end (smaller than any char).

0	s	e	a	-1							
1	s	e	a	s	h	e	l	l	s	-1	
2	s	e	l	l	s	-1					
3	s	h	e	-1							
4	s	h	e	-1							
5	s	h	e	l	l	s	-1				
6	s	h	o	r	e	-1					
7	s	u	r	e	l	y	-1				

she before shells

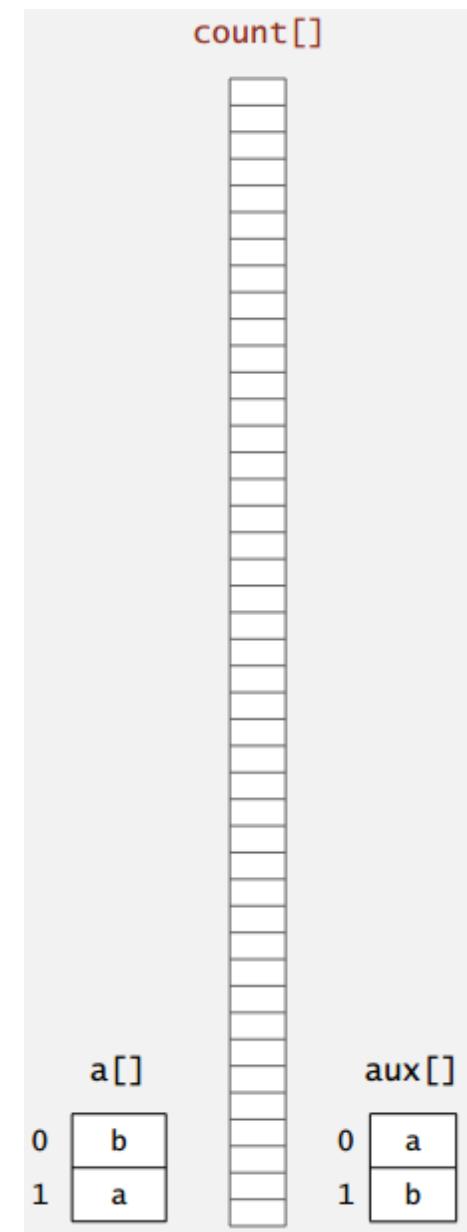
C strings. Have extra char '\0' at end => no extra work needed.

MSD string sort problem

Observation 1. Much too slow for small subarrays.

- Each function call needs its own count[] array.
- ASCII (256 counts): 100x slower than copy pass for $N=2$.
- Unicode (65536 counts): 32000x slower for $N=2$.

Observation 2. Huge number of small subarrays because of recursion.



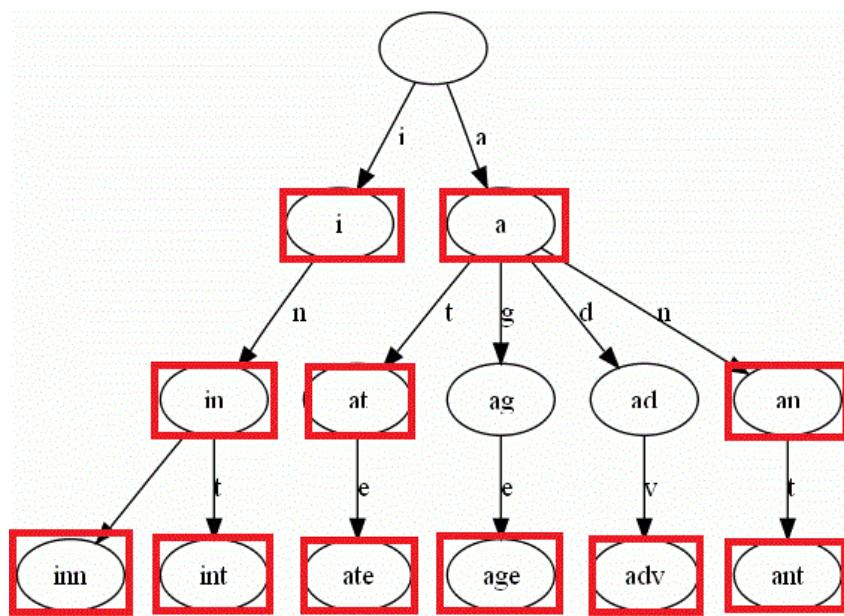
Summary: sorting algorithms

algorithm	guarantee	random	extra space	stable?	operations on keys
insertion sort	$\frac{1}{2} N^2$	$\frac{1}{4} N^2$	1	✓	<code>compareTo()</code>
mergesort	$N \lg N$	$N \lg N$	N	✓	<code>compareTo()</code>
quicksort	$1.39 N \lg N$ *	$1.39 N \lg N$	$c \lg N$		<code>compareTo()</code>
heapsort	$2 N \lg N$	$2 N \lg N$	1		<code>compareTo()</code>
LSD sort †	$2 W(N + R)$	$2 W(N + R)$	$N + R$	✓	<code>charAt()</code>
MSD sort ‡	$2 W(N + R)$	$N \log_R N$	$N + D R$	✓	<code>charAt()</code>
 $D = \text{function-call stack depth}$ $(\text{length of longest prefix match})$				<small>* probabilistic <small>† fixed-length W keys <small>‡ average-length W keys</small></small></small>	

Tries

- Retrieve
- DFA simulation
- Trie
- Radix Tree
- Suffix Trie(Tree)

Alphabet



Word:{i,a,in,at,an,inn,int,ate,age,adv,ant}

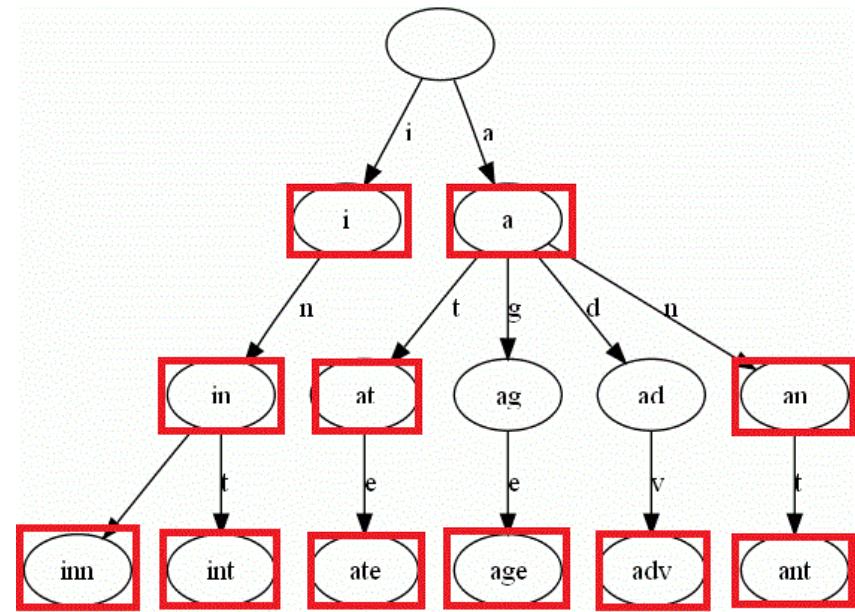
letters on the path \Leftrightarrow prefix of the word

Common Prefix \Leftrightarrow Common Ancestor

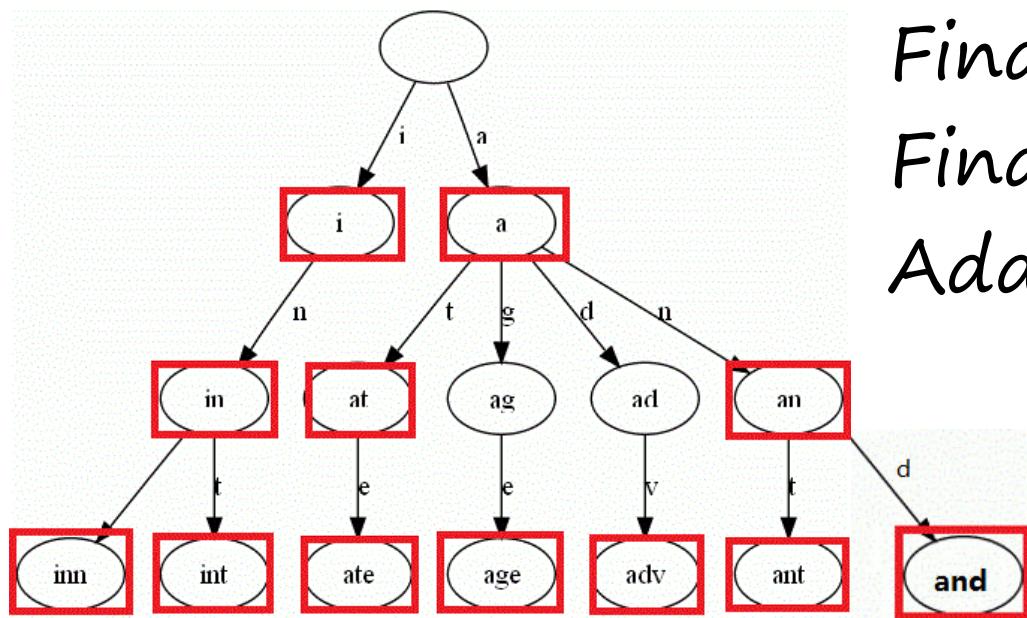
Leaf Node \Leftrightarrow longest prefix

Construct

- Node
 - Is Leaf? (Is End?)
 - Edge.
- Find word
 - Edge of next letter?
 - Exist: jump to next Node.
 - Not exist: return false.
 - Is end of the word?
- Add word
 - Edge of next letter?
 - Exist: jump to next Node.
 - Not exist: add new Node, jump to it.
 - Is end of the word?
 - Mark.



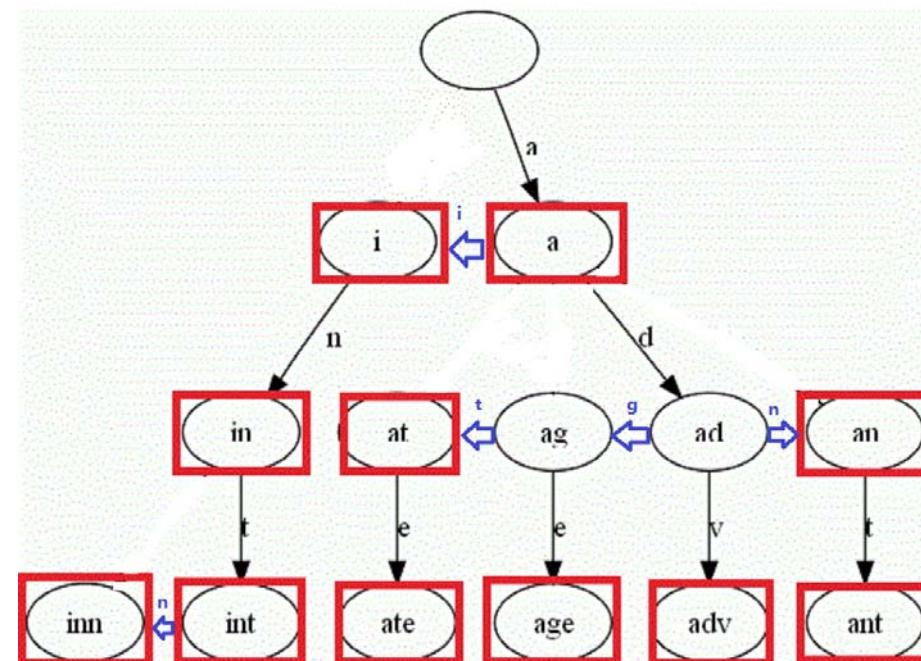
Example



Find "ant"
Find "and"
Add "and"

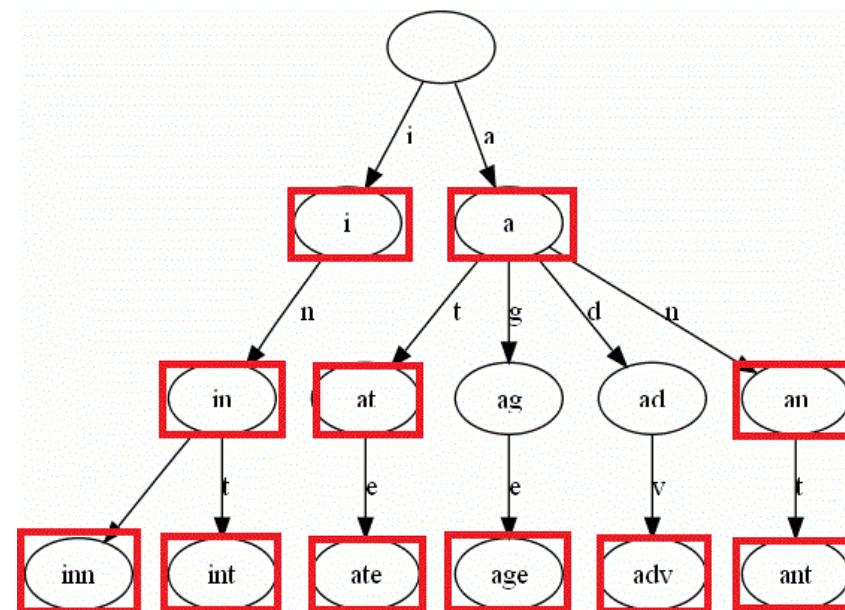
Other version

- Child-Brother Tree
 - Binary Tree
- Double Array Trie
- Ternary search tries
- How to save edge?
 - Array
 - List
 - BST



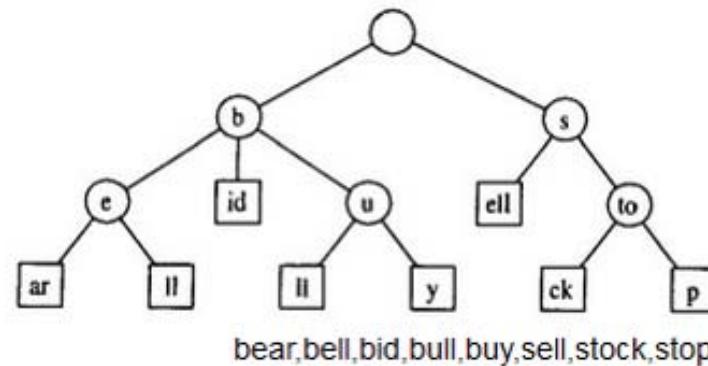
Analysis

- Time complexity
 - Add: length of string
 - Find: length of string
- Space complexity
 - Total length of string



Radix Tree

- Internal node has least two child
- Leaf Node <= number of words
- Space complexity
 - Number of words

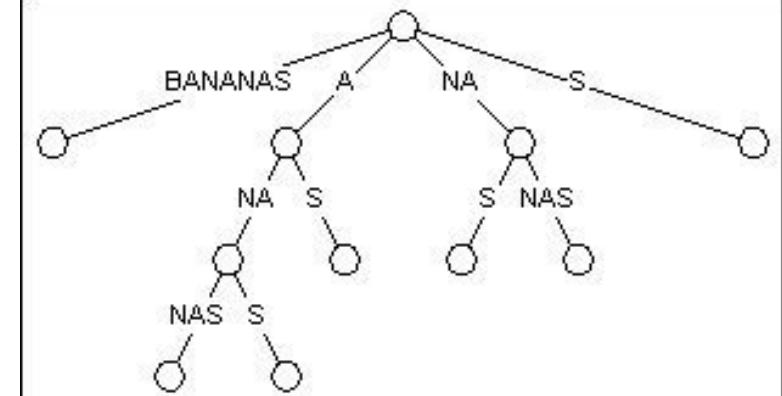
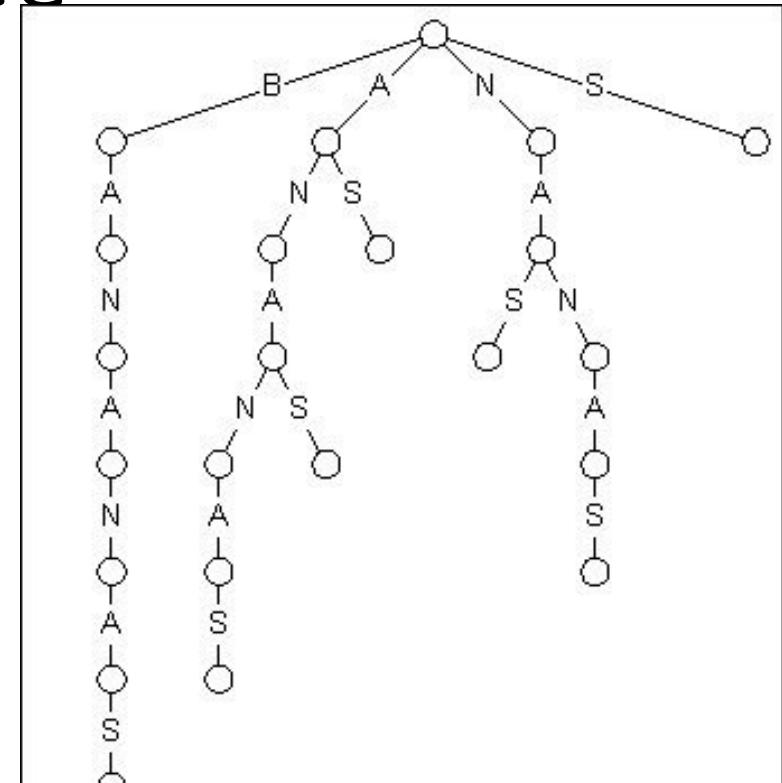


Application Scenarios

- Retrieving from dictionary
- Longest common prefix
- Sort
- Support
 - With KMP : Aho-Corasick automaton
 - Dictionary of all suffixes of word: Suffix Tree

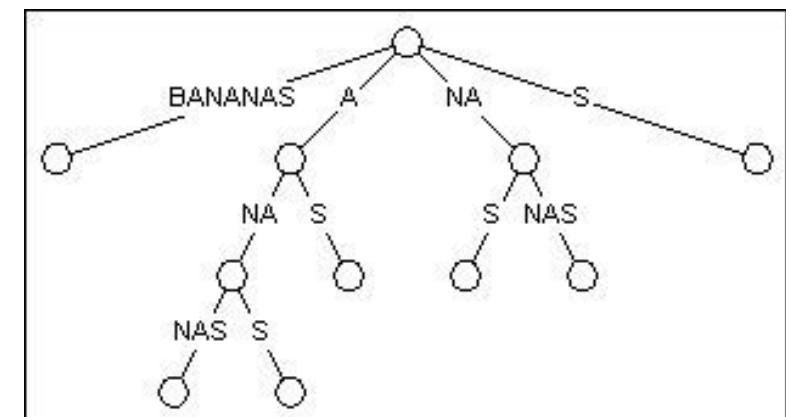
Suffix Tree

- Suffix Trie: Trie by all of suffix words
- Example:
 - “BANANAS”’s suffix Trie:
 - Dictionary:
 - BANANAS,ANANAS,NANAS,ANAS, NAS,AS,S
 - Path Compressed : Suffix Tree



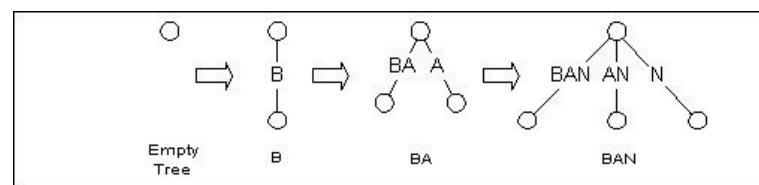
Suffix Tree: Application Scenarios

- Find substring
- Find k-th substring in MSD string sort
- Count the number of different substring
- Find the repeat time of substring
 - Find longest repeat substring
- Longest common substring
 - ≥ 2



Suffix Tree:construct

- Time complexity
 - Construction A Trie
 - $O(n^2)$
 - Esko Ukkonen Algorithm
 - Nodes $\leq 2 \cdot \text{letters}$
 - ~~step by step letter by letter prefix by prefix~~
 - deferred update
 - E.g.
 - BAN(for BANANAS)



Substring Search

- Knuth-Morris-Pratt Algorithm
- Boyer-Moore Algorithm
- Rabin-Karp Algorithm

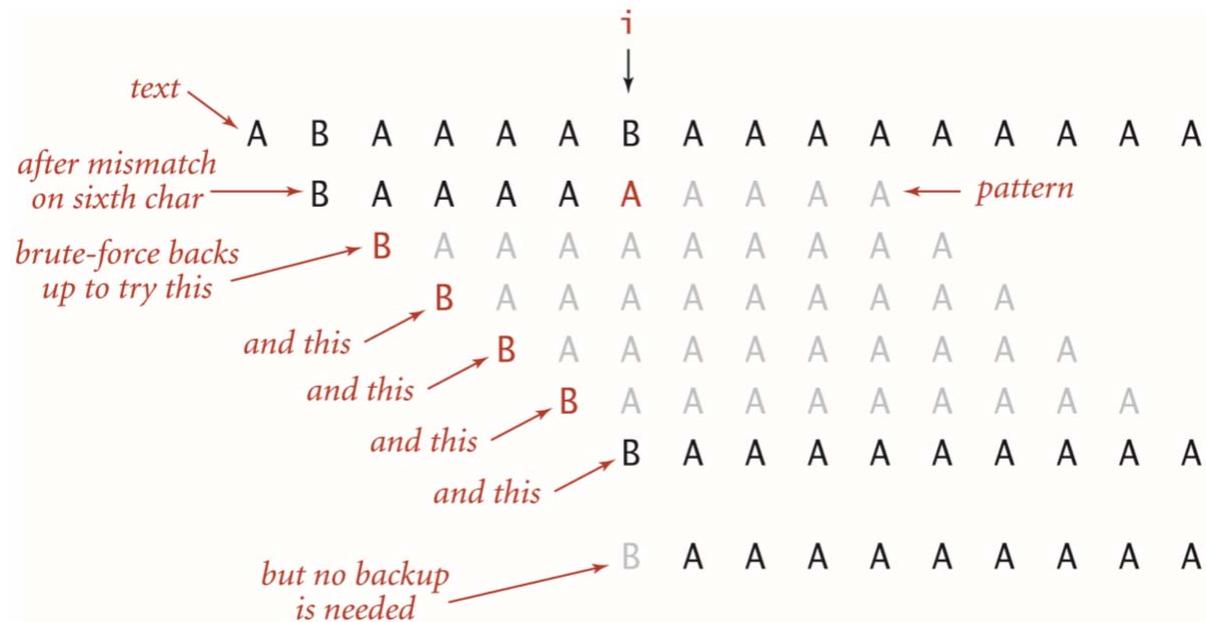
Substring Search

- Knuth-Morris-Pratt Algorithm
 - Boyer-Moore Algorithm
 - Rabin-Karp Algorithm

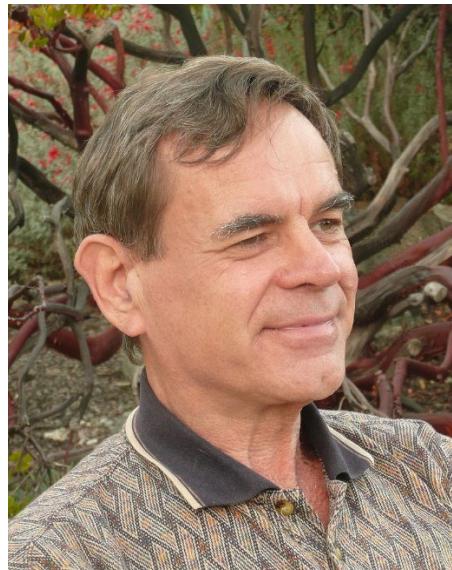
Knuth-Morris-Pratt Algorithm

- Solution with DFA simulation
- How to construct DFA
- Analysis of KMP

Brute-force substring search

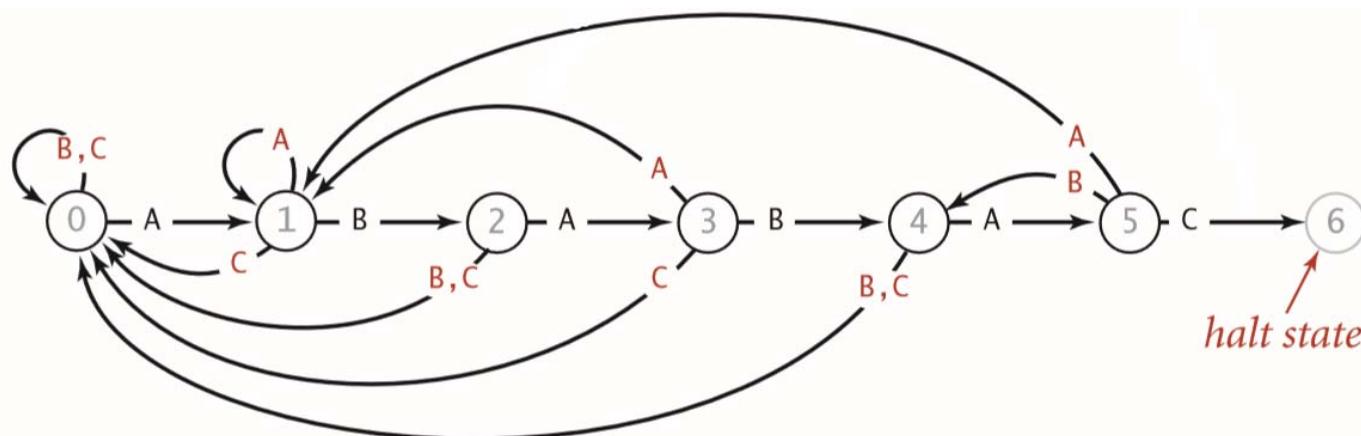


Knuth, Morris and Pratt (1974)



Overview

Consider the string B C B A A B A C A A B A B A C A A
the pattern A B A B A C



Steps

- Construct the DFA
- DFA simulation

How to construct the DFA

- Match situations:



	0	1	2	3	4	5
	A	B	A	B	A	C
A	1		3		5	
B		2		4		
C						6

How to construct the DFA

- Mismatch situations:

Maintain the state of the substring start from $s[1]$

copy $dfa[][]X$ to $dfa[][]j$, $X = dfa[pat.charAt(j)][X]$

	X					
	0	1	2	3	4	5
A	A	B	A	B	A	C
B	1	1	3	1	5	1
C	0	2	0	4	0	4
	0	0	0	0	0	6

Code

```
public KMP(String pat) { // Build DFA from pattern.  
    this.pat = pat;  
    int M = pat.length();  
    int R = 256;  
    dfa = new int[R][M];  
    dfa[pat.charAt(0)][0] = 1;  
    for (int X = 0, j = 1; j < M; j++) { // Compute dfa[][][j].  
        for (int c = 0; c < R; c++)  
            dfa[c][j] = dfa[c][X]; // Copy mismatch cases.  
        dfa[pat.charAt(j)][j] = j + 1; // Set match case.  
        X = dfa[pat.charAt(j)][X]; // Update restart state.  
    }  
}
```

Code

```
public int search(String txt) {  
    // simulate operation of DFA on text  
    int M = pat.length();  
    int N = txt.length();  
    int i, j;  
    for (i = 0, j = 0; i < N && j < M; i++) {  
        j = dfa[txt.charAt(i)][j];  
    }  
    if (j == M) return i - M; // found  
    return N; // not found  
}
```

Meaning of DFA

j	0	1	2	3	4	5
pat.charAt(j)	A	B	A	B	A	C
dfa[][][j]	1	1	3	1	5	1
	B	0	2	0	4	0
	C	0	0	0	0	6

A B B
Prefix: A AB
Suffix: B BB

A B A B A B
Prefix: A AB ABA ABAB ABABA
Suffix: B AB BAB ABAB BABAB

Analysis

- Time consumption: $O(n+m)$
- Advantage:
 - Low time consumption
 - Just traverse the string once, can be used in stream
- Disadvantage:
 - Space consumption depends on the character set

Substring Search

- Knuth-Morris-Pratt Algorithm
- Boyer-Moore Algorithm
- Rabin-Karp Algorithm

Brute Suffix Match

Algorithm: Brute-Suffix-Match(T, P)

```
1      j = 0;
2      while (j <= strlen(T) - strlen(P)) {
3          for (i = strlen(P) - 1; i >= 0 && P[i] == T[i + j]; --i)
4              if (i < 0)
5                  match;
6              else
7                  ++j;
8      }
```

Brute Suffix Match

Algorithm: Brute-Suffix-Match(T, P)

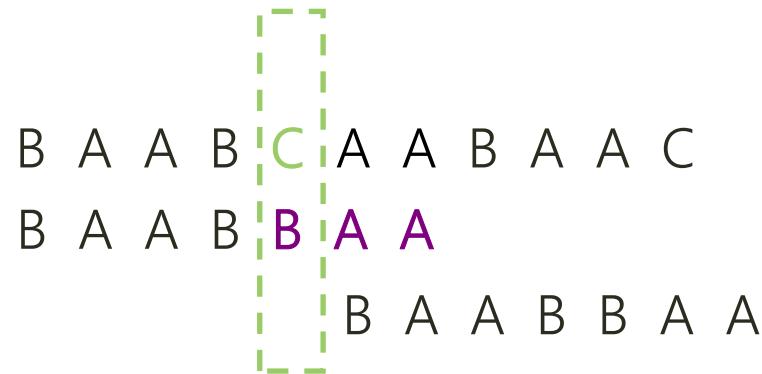
```
1      j = 0;  
2      while (j <= strlen(T) - strlen(P)) {  
3          for (i = strlen(P) - 1; i >= 0 && P[i] == T[i + j]; --i)  
4              if (i < 0)  
5                  match;  
6              else  
7                  ++j;  
8      }
```



The Bad Symbol Heuristic: Easy Case

Observation-1

If c is known not to occur in P , then we know we need not consider the possibility of an occurrence of P starting at T positions 1, 2, ... or $\text{length}(P)$



The Bad Symbol Heuristic: More Interesting Case

Observation-2

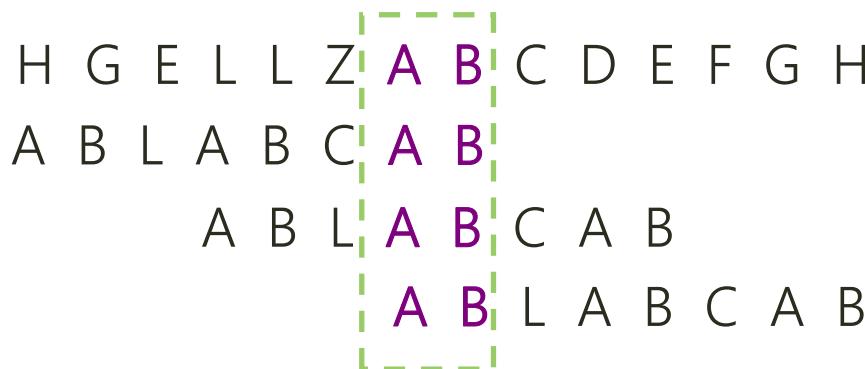
If the right-most occurrence of c in P is d from the right end of P , then we know we can slide P down d positions without checking for matches.



The Good Suffix Heuristic: Matched Suffix Occurs Elsewhere

Observation-3

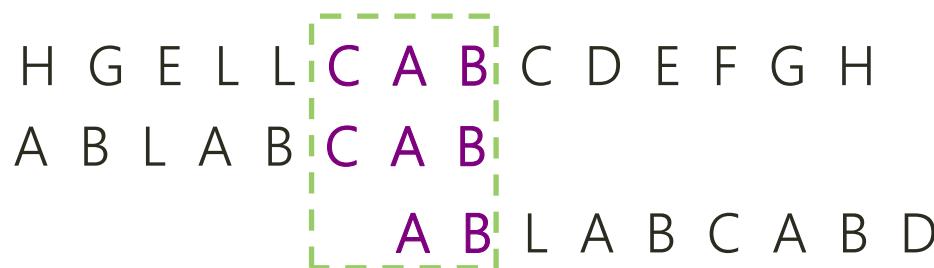
We know that the next m characters of T match the final m characters of P_{subp} . We can slide P down by some amount so that the discovered occurrence of subp in T is aligned with the rightmost occurrence of subp in P .



The Good Suffix Heuristic: Matched Suffix Not Occurs

Observation-4

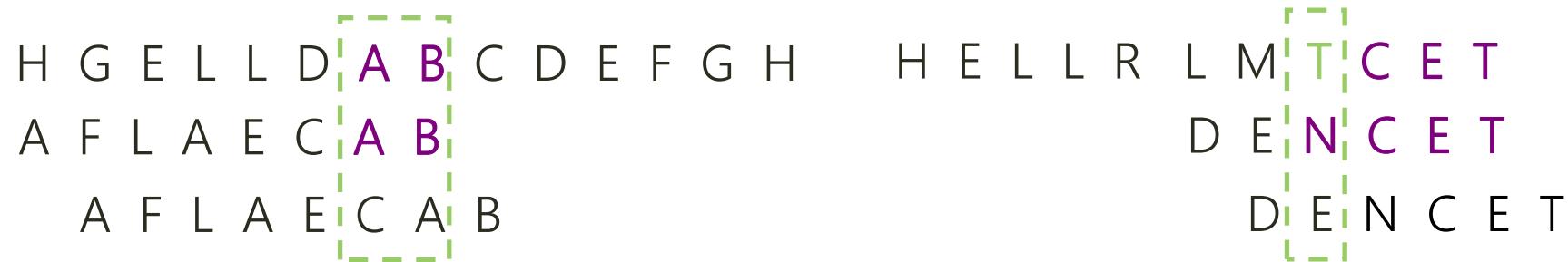
We know that the next m characters of T match the final m characters of P_{subp} . We can slide P down by some amount so that the discovered occurrence of subp in T is aligned with the occurrence of a max prefix in P .



The Good Suffix Heuristic: Matched Suffix Not Occurs

Observation-5

If the rightmost occurrence of c in P is to the right of the mismatched character or no matched suffix occurs in P, we would have to move P backwards to align the two unknown characters.



Boyer-Moore Algorithm Demo

WHICH-FINALLY-HALTS.-AT-THAT-POINT

AT- AT- AT- AT-
THAT THAT THAT THAT

Boyer-Moore Algorithm

```
void preBmBc(char *x, int m, int bmBc[]) {  
    int i;  
    for (i = 0; i < ASIZE; ++i)  
        bmBc[i] = m;  
    for (i = 0; i < m - 1; ++i)  
        bmBc[x[i]] = m - i - 1;  
}
```

```
suffix[m-1]=m;  
for (i=m-2; i>=0; --i){  
    q=i;  
    while(q>=0&&P[q]==P[m-1-i+q])  
        --q;  
    suffix[i]=i-q;  
}
```

```
void preBmGs(char *x, int m, int bmGs[]) {  
    int i, j, suff[XSIZE];  
    suffixes(x, m, suff);  
    for (i = 0; i < m; ++i)  
        bmGs[i] = m;  
    j = 0;  
    for (i = m - 1; i >= 0; --i)  
        if (suff[i] == i + 1)  
            for (; j < m - 1 - i; ++j)  
                if (bmGs[j] == m)  
                    bmGs[j] = m - 1 - i;  
    for (i = 0; i <= m - 2; ++i)  
        bmGs[m - 1 - suff[i]] = m - 1 - i;  
}
```

Boyer-Moore Algorithm

```
j = 0;  
while (j <= strlen(T) - strlen(P)) {  
    for (i = strlen(P) - 1; i >= 0 && P[i] == T[i + j]; --i)  
        if (i < 0)  
            match;  
        else  
            j += max(bmGs[i], bmBc[T[i]] - (m-1-i));  
}
```

Best:

 $O(N/M)$

Worst:

 $O(NM)$

Substring Search

- Knuth-Morris-Pratt Algorithm
- Boyer-Moore Algorithm
- Rabin-Karp Algorithm

Rabin Karp Algorithm

----for string
search

谢立伟

5130379098

2015.12

Basic Idea

Input:

$P[1..m]$: Target String

$T[1..n]$: Source Text

$(n \geq m)$

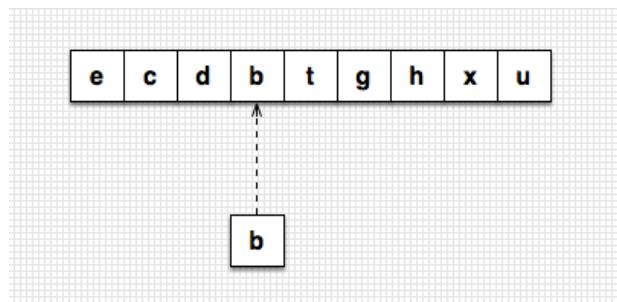
Output:

$s \rightarrow T[s+1, s+2, \dots s+m]$

$(s < n-m)$

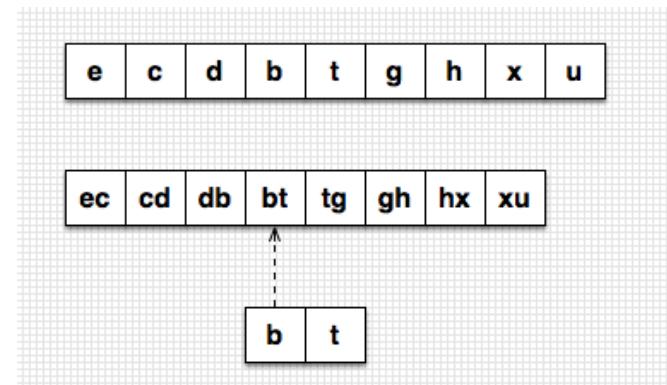
Basic Idea

when $m = 1$:



Reduce to a simple linear search problem.

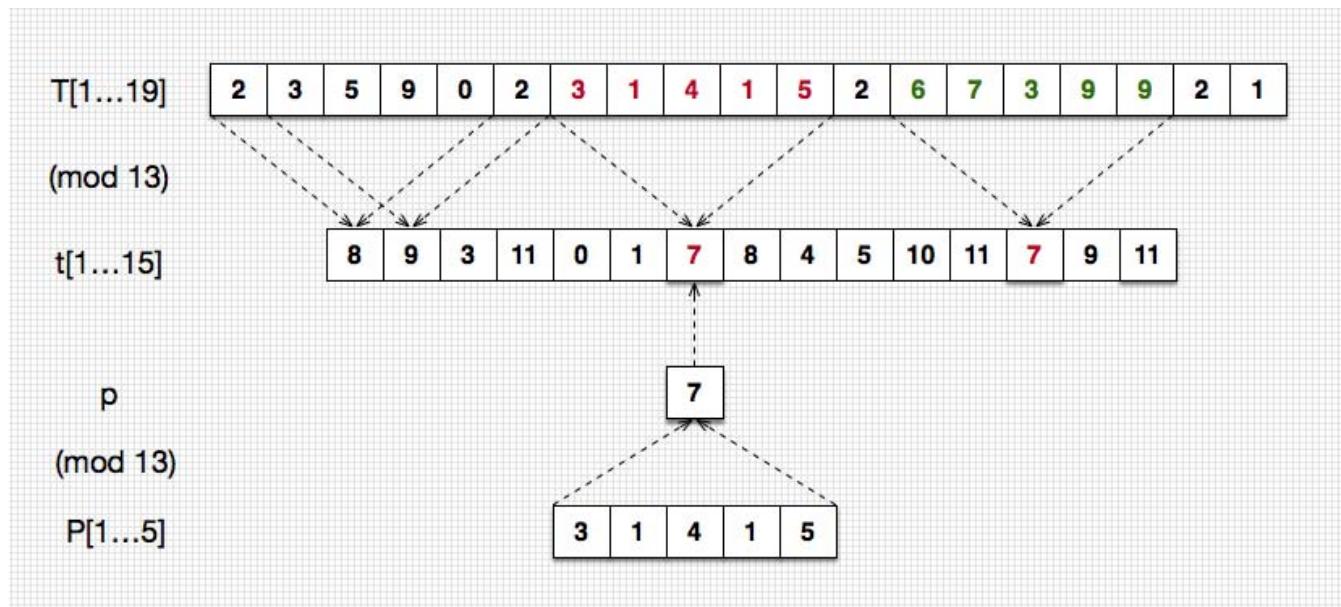
when $m > 1$:



Hash will help.

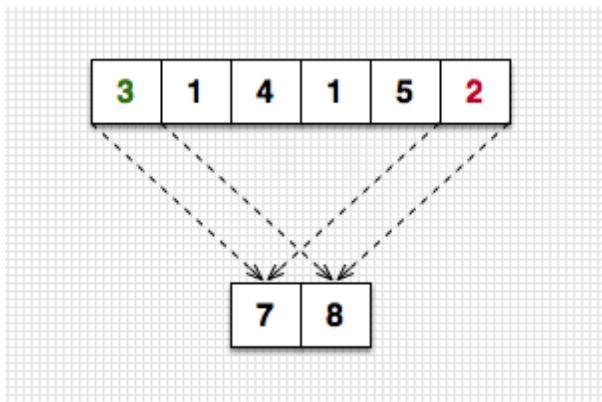
Detailed Procedure

- (For convenience, numbers only.)



Detailed Procedure

Optimization:



- 14152
- $10 * (31415 - 3 * 10000) + 2 \pmod{13}$
 - $10 * (7 - 3 * 3) + 2 \pmod{13}$
 - $8 \pmod{13}$

(Dynamic Programming)

Pseudo

RABIN-KARP-MATCHER(T, P, d, q)

```
n = T.length  
m = P.length  
h =  $d^{(m-1)} \bmod q$   
p = 0  
t[0] = 0  
for i = 1 to m  
    p = ( $d * p + P[i]$ ) mod q  
    t[0] = ( $d * t[0] + T[i]$ ) mod q  
for s = 0 to n-m  
    if p == t[s]  
        if P[1..m] == T[s+1..s+m]  
            print ("Find P with shift: " + s)  
if s < n-m  
    t[s+1] = ( $d * (t[s] - T[s+1]) * h + T[s+m+1]$ ) mod q
```

Space Complexity:

$O(n-m)$

Time Complexity:

$O(m * (n-m))$

Analysis

When q increases

pros

Reduce the chance of conflicts

Decrease the time of confirmation

cons

(May)Increase the space requirement

(May)Increase the time of Mod operation

数据压缩



成员：李子男、殷国航、穆迪、翟煜、阎姝含



1 压缩基础

2 数据压缩

3 图像压缩

4 视频压缩



1 压缩基础

2 数据压缩

3 图像压缩

4 视频压缩

数据压缩技术简单分类

表 1.2 数据压缩技术的简单分类

数 据 压 缩	冗余度压缩 (熵编码)	统计编码	霍夫曼编码、游程编码、二进制信源编码等		
			算术编码		
			基于字典的编码: LZW 编码等		
			其他编码 完全可逆的小波分解+统计编码等		
	熵压缩	特征抽取	分析/综合编码		子带、小波、分形、模型基等
					其他
		量化	无记忆量化		均匀量化、Max 量化、压扩量化等
			有 记 忆 量 化	序列量化	增量调制、线性预测、非线性预测、自适应预测、运动补偿预测等
				其他方法	序贯量化等
		分组量化	直接映射	矢量量化、神经网络、方块截尾等	
				正交变换: KLT、DCT、DFT、WHT 等	
			变换编码	非正交变换 其他函数变换等	

压缩原理

- 用更短的符号替代重复出现的字符串
- ABABABABABABAB -> 7AB
- 中华人民共和国 -> 中国 李子男->肉腿
- 保证对应关系，唯一可译
- 内容毫无重复，就很难压缩（均匀分布的随机字符串、 π 、任意排列的阿拉伯数字）
- 压缩是一个消除冗余的过程，好的压缩算法可以将冗余降到最低，以至于无法进一步压缩。

信息量

- 扔硬币（两种情况 10）、球赛（三种情况 两位）
- 假定一个字符（或字符串）在文件中出现的概率是 p ，那么在这个位置上最多可能出现 $1/p$ 种情况。
- 信息量：随机变量 X 取值为 x 时所携带信息的度量。

$$I(x) = \log_2 [1/p(x)] = -\log_2 p(x)$$

信息熵

$$I(x) = \log_2 [1/p(x)] = -\log_2 p(x)$$

- 若信源符号有n种取值: $U_1 \dots U_i \dots U_n$, 对应概率为: $P_1 \dots P_i \dots P_n$, 且各种符号的出现彼此独立
- 信息熵: 信息量的概率平均值、 $I(x)$ 的数学期望值

$$H(U) = E[-\log p_i] = -\sum_{i=1}^n p_i \log p_i$$

信息熵

$$H(U) = E[-\log p_i] = - \sum_{i=1}^n p_i \log p_i$$

- 两个文件都包含1024个符号，都是1KB。
- 甲文件的内容50%是a，30%b，20%是c。
 - $-0.5 * \log_2(0.5) - 0.3 * \log_2(0.3) - 0.2 * \log_2(0.2)$
 - $= 1.49$
- 乙文件的内容33%是a，33%是b，33%是c
 - $-0.33 * \log_2(0.33) * 3$
 - $= 1.58$
- 文件内容越是分散（随机），所需要的二进制位就越长。
- 信息熵用来衡量文件内容的随机性（又称不确定性）。

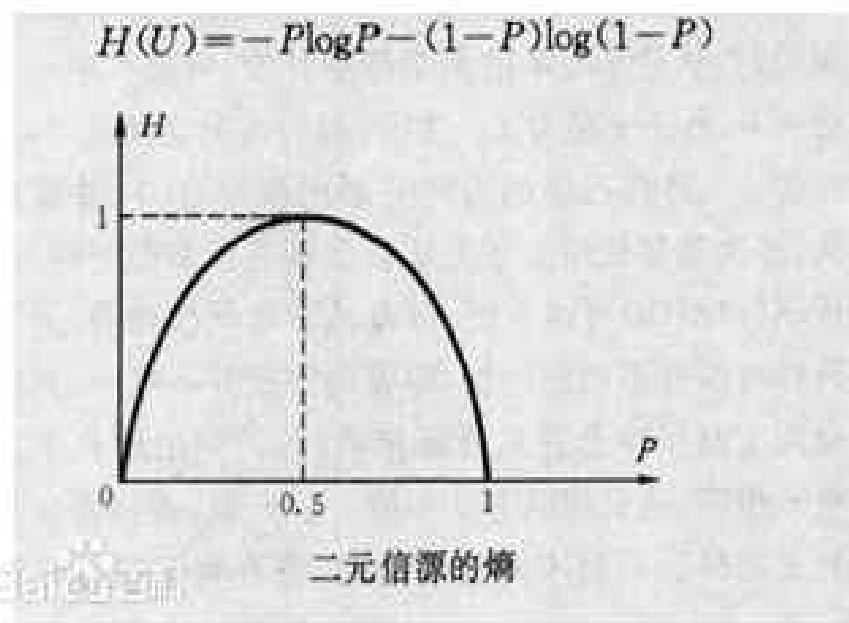
最大离散熵

- m表示信息元个数
- m= 2 时 p=0.5时， $H(p) = 1$
- 兀余度：

$$r = H_{\max}(X) - H(X) = \log m - H(X)$$

- 只要信源不是等概率分布， 就存在着数据压缩的可能性。

$$H(U) = E[-\log p_i] = -\sum_{i=1}^n p_i \log p_i$$



压缩比与编码效率

- 压缩比:

$$CR = \frac{\log m}{l}$$

$\log m$: 压缩前每个信源符号的编码位数

l : 压缩后平均每符号编码位数

- 编码效率:

$$\eta = \frac{H(X)}{l} (\%)$$

$H(X)$: 信息熵

$$H(X) = E[-\log p_i] = -\sum_{i=1}^n p_i \log p_i$$

压缩比与编码效率

$$X = \begin{Bmatrix} a_1 & a_2 & a_3 & a_4 \\ \frac{1}{2} & \frac{1}{4} & \frac{1}{8} & \frac{1}{8} \end{Bmatrix} \quad \log m = 2$$

$$H(X) = -\frac{1}{2}\log\frac{1}{2} - \frac{1}{4}\log\frac{1}{4} - 2 \times \frac{1}{8}\log\frac{1}{8} = 1.75 \text{ bit/字符}$$

$$\eta = \frac{H(X)}{l} (\%)$$

$$CR = \frac{\log m}{l}$$

$$H(U) = E[-\log p_i] = -\sum_{i=1}^n p_i \log p_i$$

$$\textcircled{1} \quad \begin{Bmatrix} a_1 & a_2 & a_3 & a_4 \\ 00 & 01 & 10 & 11 \end{Bmatrix}$$

$$\text{平均码长为 } l = 2 \times \sum_{j=1}^4 p_j = 2 \text{ bit/字符}$$

$$CR = 2/2 = 1, \quad \eta = 1.75/2 = 87.5\%;$$

$$\textcircled{2} \quad \begin{Bmatrix} a_1 & a_2 & a_3 & a_4 \\ 0 & 10 & 110 & 111 \end{Bmatrix}$$

$$\text{平均码长 } l = \sum p_j \cdot L_j = \frac{1}{2} \times 1 + \frac{1}{4} \times 2 + \frac{1}{8} \times 3 + \frac{1}{8} \times 3 = 1.75 \text{ bit/字符}$$

$$CR = CR_{\max} = 2/1.75$$

$$\eta = 1.75/1.75 \text{ 达到了 } 100\%$$

统计编码-莫尔斯电码

表 4.1 莫尔斯码

字母	莫尔斯码	铅字数	字母	莫尔斯码	铅字数
E	.	12000	M	..	3000
T	-	9000	F	2500
A	. -	8000	W	. --	2000
I	--	8000	Y	. . --	2000
N	--.	8000	G	.. -	1700
O	---	8000	P	* --	1700
S	***	8000	B	. ***	1600
H	****	6400	V	*** .	1200
R	* * *	6200	K	.. * -	800
D	- * *	4400	Q	.. * -	500
L	* . **	4000	J	* ---	400
U	* * -	3400	X	. ***	400
C	- * . *	3000	Z	... --	200

费诺编码

- 1、按照频率排序
- 2、划分两部分，使得左右两部分概率和尽可能接近
- 3、左边二进制为0，右边二进制为1
- 4、递归应用2、3步骤

费诺编码

消息符号	各个消息概率 $p(x_i)$	第一次分组	第二次分组	第三次分组	第四次分组	第五次分组	二元码字	码长 K_i
x_1	0.25	0	0				00	2
x_2	0.2		1				01	2
x_3	0.2	1	0	0			100	3
x_4	0.1		0	1			101	3
x_5	0.1	1	0				110	3
x_6	0.08		1		0		1110	4
x_7	0.05	1	1		1	0	11110	5
x_8	0.02		1		1	1	11111	5

费诺编码实现

```
int Group(CodeType FanoNode[],int low,int high){  
    float MinSum=FanoNode[low].data,MaxSum=FanoNode[high].data;  
    FanoNode[low].bit[FanoNode[low].length++]=1;  
    FanoNode[high].bit[FanoNode[high].length++]=0;  
    while(low+1<high){  
        if(MinSum>MaxSum){  
            MaxSum+=FanoNode[--high].data;  
            FanoNode[high].bit[FanoNode[high].length++]=0;  
        }  
        else{  
            MinSum+=FanoNode[++low].data;  
            FanoNode[low].bit[FanoNode[low].length++]=1;  
        }  
    }  
    return low;  
}  
  
void FanoEncoding(CodeType FanoNode[],int s,int t){  
    if(s<t){  
        int pivotloc=Group(FanoNode,s,t);  
        if(s<t-1){  
            FanoEncoding(FanoNode,s,pivotloc);  
            FanoEncoding(FanoNode,pivotloc+1,t);  
        }  
    }  
}
```

费诺编码

消息符号	各个消息概率 $p(x_i)$	第一次分组	第二次分组	第三次分组	第四次分组	第五次分组	二元码字	码长 K_i
x_1	0.25	0	0				00	2
x_2	0.2		1				01	2
x_3	0.2		0	0			100	3
x_4	0.1		1	1			101	3
x_5	0.1		0	0			110	3
x_6	0.08		1	0			1110	4
x_7	0.05		1	1	0		11110	5
x_8	0.02				1	0	11111	5

平均码长:

$$\begin{aligned}\bar{K} &= \sum_{i=1}^8 p(x_i)K_i \\ &= 0.25 \times 2 + 0.2 \times 2 + 0.2 \times 3 + 0.1 \times 3 + 0.1 \times 3 + 0.08 \times 4 + 0.05 \times 5 + 0.02 \times 5 \\ &= 2.77 \text{ 码元/符号}\end{aligned}$$

信源熵:

$$\begin{aligned}H(X) &= -\sum_{i=1}^8 p_i \log p_i \\ &= 0.050 + 0.464 + 0.464 + 0.332 + 0.332 + 0.292 + 0.216 + 0.113 \\ &= 2.71 \text{ bit/符号}\end{aligned}$$

编码效率:

$$\eta = \frac{H(X)}{\bar{K}} = \frac{2.71}{2.77} = 0.98$$

Huffman编码

- 概率排序
- 构建哈夫曼树，合并两个最小概率的事件，直到概率达到1为止
- 一个支路为0，一个支路为1
- 顺序记录路径上的1和0，即为哈夫曼码

Huffman编码

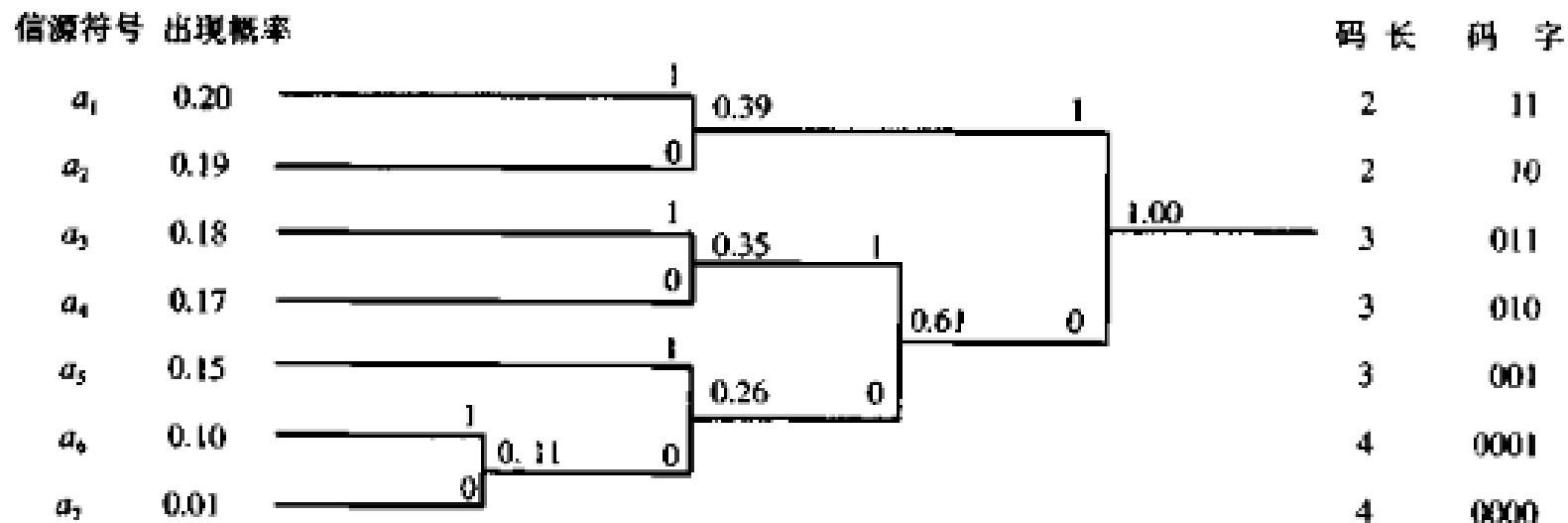


图 4.6 霍夫曼编码示例

Huffman编码

1、根据权值初始化哈夫曼树

```
for(int i=0;i<n;i++)
{
    HTree[i].weight=a[i];
    HTree[i].LChild=-1;
    HTree[i].RChild=-1;
    HTree[i].parent=-1;
}
```

2、开始建立哈夫曼树，

```
int x,y;
for (int i=n;i<2*n-1;;i++) {
    selectMin(x,y,o,i);//从1到i选择两个权值最小的
    HTree[x].parent=HTree[y].parent=i;
    HTree[i].weight=HTree[x].weight+HTree[y].weight;
    HTree[i].LChild=x;
    HTree[i].RChild=y;
    HTree[i].parent=-1;
}
```

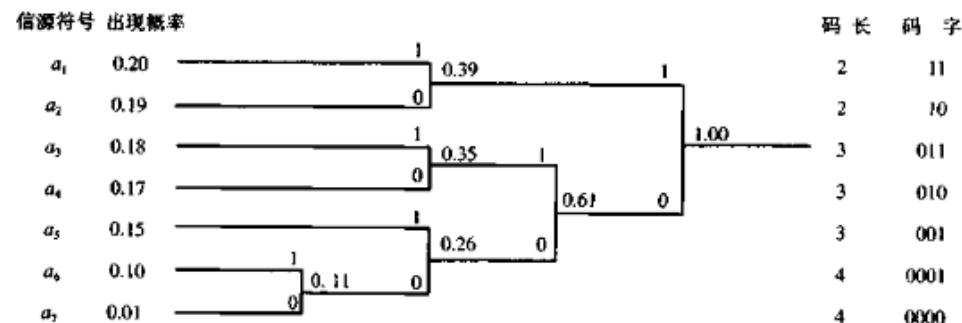


图 4.6 霍夫曼编码示例

Huffman编码

二、建立哈夫曼编码表

```
void Huffman::CreateCodeTable(char b[],int n)
{
    HCodeTable=new HCode [n];
    for (int i=0;i<n;i++)
    {
        HCodeTable[i].data=b[i];//生成编码表
        int child=i;
        int parent=HTree[i].parent;
        int k=0;
        while (parent!=-1)
        {
            if(child==HTree[parent].LChild)
                HCodeTable[i].code[k]='0';//左孩子标 '0' ;
            else
                HCodeTable[i].code[k]='1';//右孩子标 '1' ;
            k++;
            child=parent;
            parent=HTree[child].parent;
        }
    }
}
```

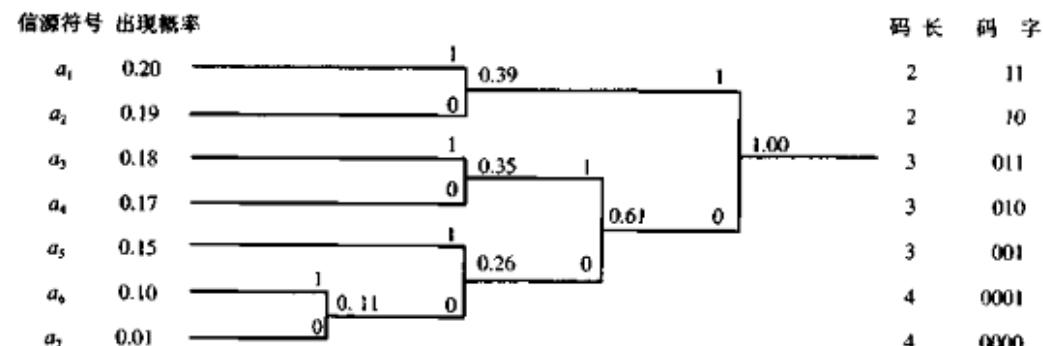


图 4.6 霍夫曼编码示例

Huffman 编码

四、解码：利用哈夫曼树进行解码，编码串左到右依次逐位判断，从根结点开始根据每一位是 0 还是 1，确定是左分支还是右分支，直到到叶子节点为止，从编码表中找到对应字符并输出

```
void Huffman::Decode(char*s,char*d,int n)//s 为编码串，  
{  
    while(*s!='\0')  
    {  
        int parent=2*n-1;//根节点在 HTree 中的下标  
        while(HTree[parent].LChild!=-1)  
        {  
            if(*s=='0')  
                parent=HTree[parent].LChild;  
            else  
                parent=HTree[parent].RChild;  
            s++;  
        }  
        *d=HCodeTable[parent].data;  
        cout<<*d;  
        d++;  
    }  
}
```

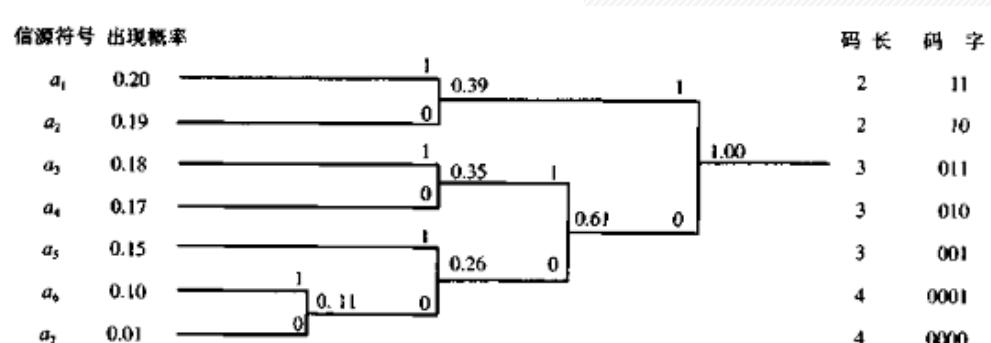


图 4.6 霍夫曼编码示例

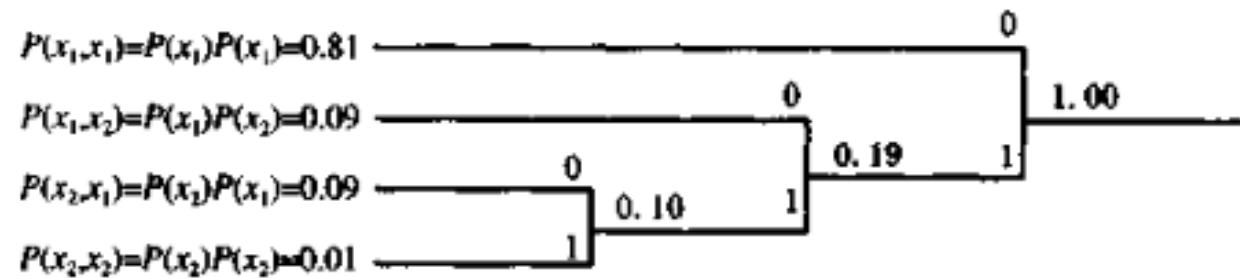
Huffman编码特点

- 霍夫曼编码对不同的信源的编码效率是不同的。
当信源概率是 2 的负幂时，霍夫曼码的编码效率达到 100%；当信源概率相等时，其编码效率最低。
只有在概率分布很不均匀时，霍夫曼编码才会收到显著的效果。
- 解码时，必须参照这一霍夫曼编码表才能正确译码。
在信源的存储与传输过程中必须首先存储或传输这一霍夫曼编码表。在实际计算压缩效果时，必须考虑霍夫曼编码表占有的比特数。

提高编码效率

- $X = \{x_1, x_2\} = \{\text{黑}, \text{白}\}$
- $P(x_1) = 0.9 \quad P(x_2) = 0.1$
- $H(X) = -0.9 * \log 0.9 - 0.1 * \log 0.1 = 0.469$
- 码长至少要1位，则编码效率只有 $H(X)/1 = 46.9\%$

提高编码效率

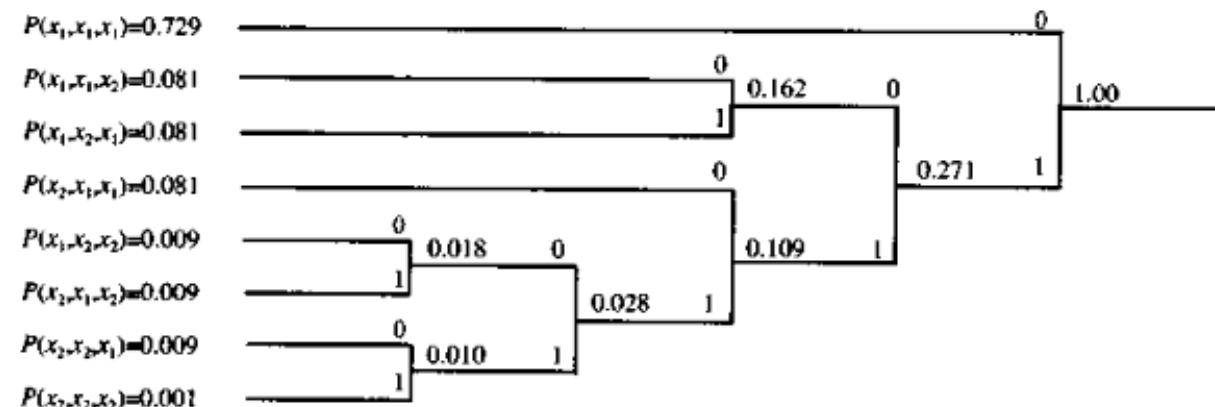


{0,10,110,111}

平均编码长度: $l_2 = 0.81 + 0.09 \times 2 + 0.09 \times 3 + 0.01 \times 3 = 1.29 \text{ bit/pel}$ $l_2/2 = 0.645 \text{ bit/pel}$

编码效率: $\eta_2 = \frac{H(X)}{l_2/2} = \frac{0.469}{0.645} = 72.7\%$

提高编码效率



$$W^3 = \{0,100,101,110,11100,11101,11110,11111\}$$

平均编码长度: $I_3 = 0.729 + 3 \times 3 \times 0.081 + 5 \times (3 \times 0.009 + 0.001) = 1.598 \text{ bit/pel}$

$$I_3/3 = 0.5327 \text{ bit/pel} .$$

编码效率: $\eta_3 = \frac{H(X)}{I_3/3} = \frac{0.469}{0.5327} = 88.0\%$

游程编码

- 用一个符号值或串长代替具有相同值的连续符号
- 55555777773332222111111
- (5, 6) (7, 5) (3, 3) (2, 4) (1, 7)

游程编码

- aaaaaaaaaaabbbaxxxxxyyzyx
- a10b3a1x4y3z1y1x1 70%
- a10b3ax4y3zyx 54%
- A0b10a13x14y18z21y22x23
- 适用于大量重复数据

游程编码应用

- 二值图像

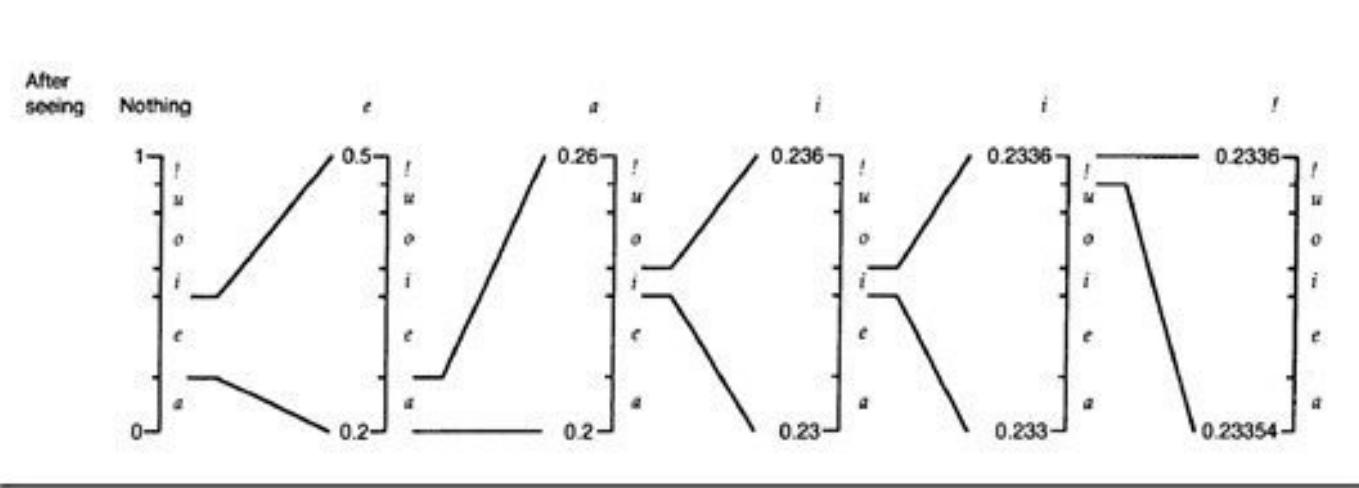
RL 长度	白游程码字	黑游程码字	RL 长度	白游程码字	黑游程码字
0	00110101	0000110111	32	00011011	000001101010
1	000111	010	33	00010010	000001101011
2	0111	11	34	00010011	000011010010
3	1000	10	35	00010100	000011010011
4	1011	011	36	00010101	000011010100
5	1100	0011	37	00010110	000011010101
6	1110	0010	38	00010111	000011010110
7	1111	00011	39	00101000	000011010111
8	10011	000101	40	00101001	000001101100
9	10100	000100	41	00101010	000001101101
10	00111	0000100	42	00101011	0000011011010
11	01000	0000101	43	00101100	0000011011011
12	001000	0000111	44	00101101	0000001010100
13	000011	00000100	45	00000100	0000001010101
14	110100	00000111	46	00000101	0000001010110
15	110101	000011000	47	00001010	0000001010111
16	101010	0000010111	48	00001011	0000001100100
17	101011	0000011000	49	01010010	0000001100101

算术编码

假设某个字符的出现概率为 80%，该字符事实上只需要 $-\log_2(0.8)$
 $= 0.322$ 个二进制位进行编码

难道真的能只输出 0.322 个 0 或 0.322 个 1 吗？

算术编码



算术编码的输出是：一个小数

算术编码对整条信息（无论信息有多么长），其输出仅仅是一个数，而且是一个介于0和1之间的二进制小数。

例如算术编码对某条信息的输出为1010001111，那么它表示小数0.1010001111，也即十进制数0.64

算术编码

例：考虑某条信息中可能出现的字符仅有 a b c 三种，我们要压缩保存的原始信息为
bccb

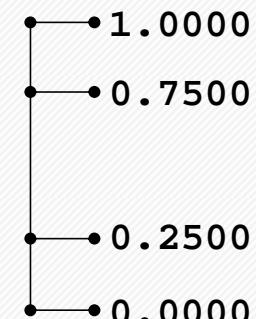
第一步：首先获取字符出现的概率表，我们将0-1区间按照概率的比例分配给三个字符，即a从0.0000到0.2500，b从0.2500到0.7500，c从0.7500到1.0000。用图形表示就是：

符号	概率
a	0.25
b	0.5
c	0.25

$$P_c = 1/4$$

$$P_b = 1/2$$

$$P_a = 1/4$$

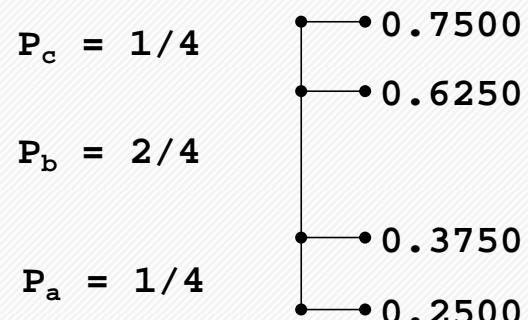


算术编码

例：考虑某条信息中可能出现的字符仅有 a b c 三种，我们要压缩保存的原始信息为
bccb

第二步：现在我们拿到第一个字符b，让我们把目光投向b对应的区间0.2500-0.7500。让我们再按照概率分布比例划分0.2500-0.7500这一区间，划分的结果可以用图形表示为：

符号	概率
a	0.25
b	0.5
c	0.25



算术编码

例：考虑某条信息中可能出现的字符仅有 a b c 三种，我们要压缩保存的原始信息为
bccb

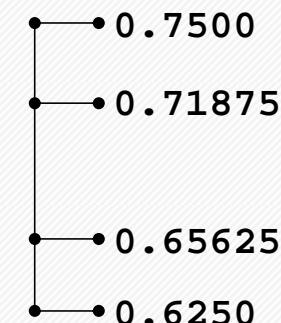
第三步：接着我们拿到字符c，我们现在要关注上一步中得到的c的区间 0.6250-0.7500。我们用概率分布划分区间0.6250-0.7500：

符号	概率
a	0.25
b	0.5
c	0.25

$$P_c = 1/4$$

$$P_b = 1/2$$

$$P_a = 1/4$$



算术编码

例：考虑某条信息中可能出现的字符仅有 a b c 三种，我们要压缩保存的原始信息为
bccb

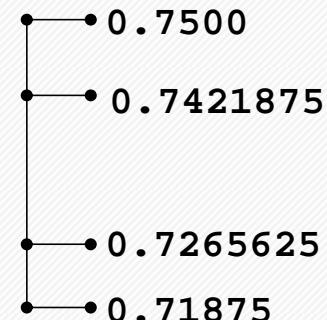
第四步：现在输入下一个字符c，关注上一步中得到的c的区间 0.71875-0.7500，
我们来划分c的区间0.71875 -0.7500：

符号	概率
a	0.25
b	0.5
c	0.25

$$P_c = 1/4$$

$$P_b = 1/2$$

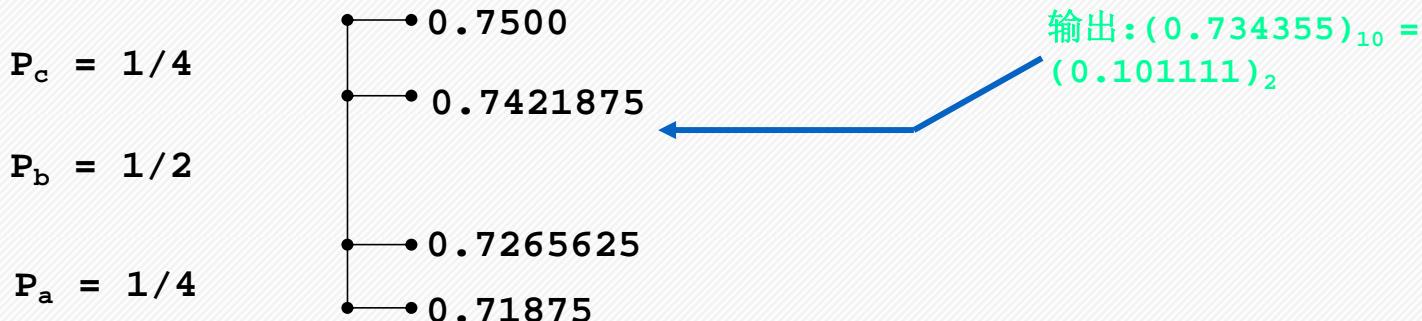
$$P_a = 1/4$$



算术编码

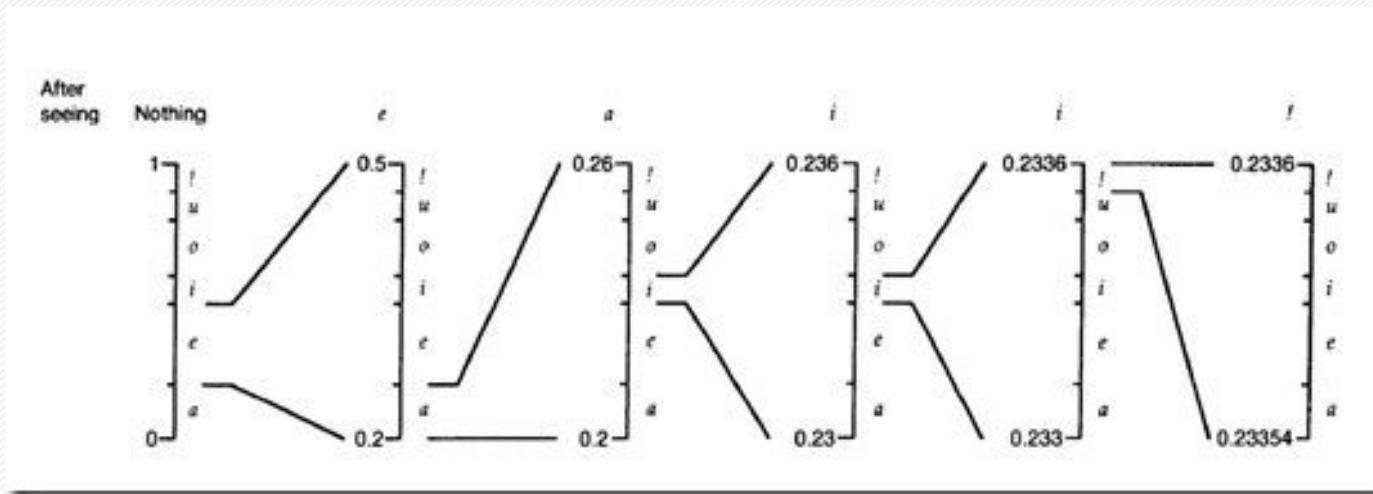
例：考虑某条信息中可能出现的字符仅有 a b c 三种，我们要压缩保存的原始信息为
bccb

第五步：输入最后一个字符b，因为是最后一个字符，不用再做进一步的划分了，
上一步中得到的b的区间为0.7265625-0.7421875，转化为二进制为
(0.1011101~，0.1011111)，选择区间内的一个数0.101111，去掉前面没有
意义的0和小数点，我们可以输出101111，这就是信息被压缩后的结果，我们完
成了一次最简单的算术压缩过程



算术编码

1. 获取信源概率表
2. 根据概率表不断压缩区间
3. 获得数值



算术编码实现问题

信源概率表的获取：

1. 经验数值
2. 提前扫描一遍数据

最终编码数值选取：

1. 区间内任意值
2. 区间特殊位置，如下边界
3. 最大相同位

算术解码

例：考虑某条信息中可能出现的字符仅有 a b c 三种，我们要压缩保存的编码信息为 101111

第一步：在101111前添加0.，变为0.101111，再将二进制码转为十进制码：
0.734355

算术解码

例：考虑某条信息中可能出现的字符仅有 a b c 三种，我们要压缩保存的编码信息为 101111

第二步：首先获取字符出现的概率表，我们将0-1区间按照概率的比例分配给三个字符，即a从0.0000到0.2500，b从0.2500到0.7500，c从0.7500到1.0000。用图形表示就是：

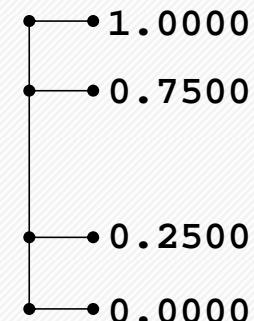
由于编码为：0.734355，在[0.25,0.75)区间，故我们得知原始信息第一位为b。

符号	概率
a	0.25
b	0.5
c	0.25

$$P_c = 1/4$$

$$P_b = 1/2$$

$$P_a = 1/4$$



算术解码

例：考虑某条信息中可能出现的字符仅有 a b c 三种，我们要压缩保存的编码信息为 101111

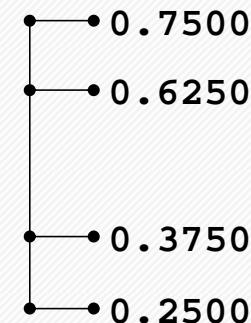
第三步：现在我们拿到第一个字符b，让我们把目光投向b对应的区间0.2500-0.7500。让我们再按照概率分布比例划分0.2500-0.7500这一区间。由于编码为0.734355，在c区间，故我们知道原始信息第二位为c

符号	概率
a	0.25
b	0.5
c	0.25

$$P_c = 1/4$$

$$P_b = 2/4$$

$$P_a = 1/4$$



算术解码

例：考虑某条信息中可能出现的字符仅有 a b c 三种，我们要压缩保存的编码信息为 101111

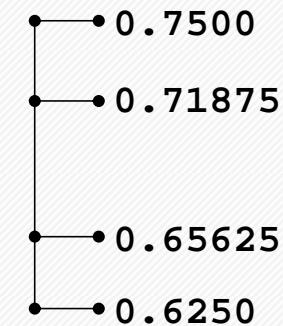
第四步：拿到字符c后，我们现在要关注上一步中得到的c的区间 0.6250-0.7500。
我们用概率分布划分区间0.6250-0.7500。
编码信息为：0.734355，获得第三位是c

符号	概率
a	0.25
b	0.5
c	0.25

$$P_c = 1/4$$

$$P_b = 1/2$$

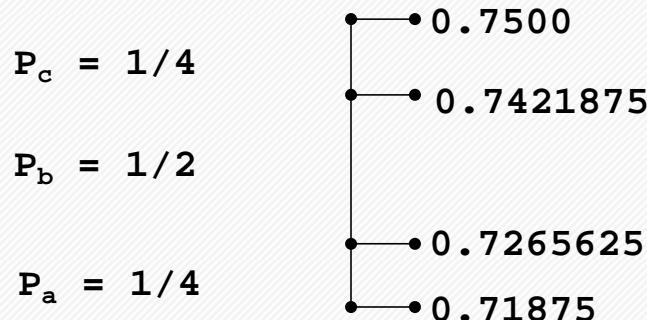
$$P_a = 1/4$$



算术解码

例：考虑某条信息中可能出现的字符仅有 a b c 三种，我们要压缩保存的编码信息为 101111

第五步：关注上一步中得到的c的区间 0.71875-0.7500，我们来划分c的区间。
编码为：0.734355，第四位为b。



算术解码的结束

1. 使用特殊位置的编码，如边界中值
2. 加入特殊字符，如！、EOF
3. 记录字符串长度

算术编码的自适应模型

在现实生活中，实现知道**精确信源概率**很难且不切实际的。静态模型需要在压缩前对信息内字符的分布进行统计，这一统计过程将消耗大量的时间，使得本来就比较慢的算术编码压缩更加缓慢

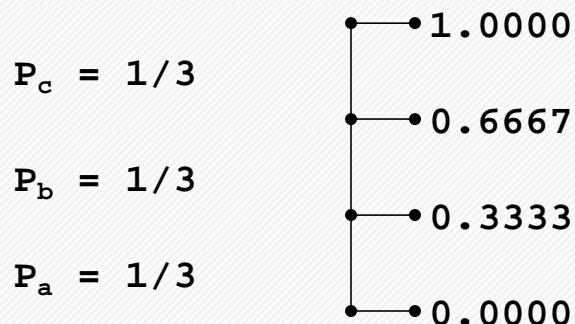
必须再**消耗一定的空间**保存静态模型统计出的概率分布，保存模型所用的空间将使我们重新远离熵值。

因此最有效的方法是**在编码过程中估算概率**。

算术编码

例：考虑某条信息中可能出现的字符仅有 a b c 三种，我们要压缩保存的原始信息为
bccb

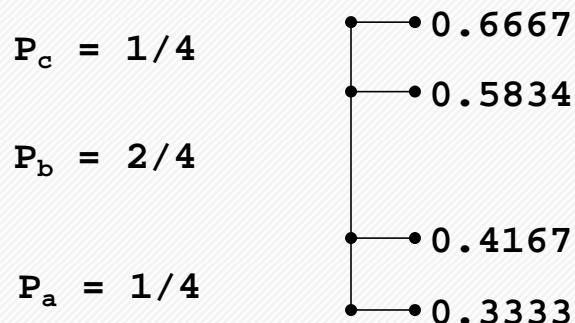
第一步：在没有开始压缩进程之前，我们对 a b c 三者在信息中的出现概率一无所知，即认为三者的出现概率相等，都为 $1/3$ ，我们将0-1区间按照概率的比例分配给三个字符，即a从0.0000到0.3333，b从0.3333到0.6667，c从0.6667到1.0000。用图形表示就是：



算术编码

例：考虑某条信息中可能出现的字符仅有 a b c 三种，我们要压缩保存的原始信息为
bccb

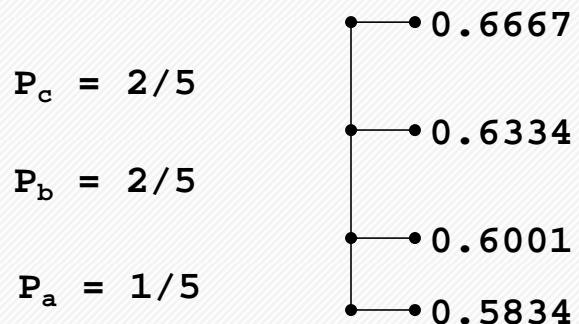
第二步：现在我们拿到第一个字符b，让我们把目光投向b对应的区间0.3333-0.6667。这时由于多了字符b，三个字符的概率分布变成： $P_a = 1/4$ ， $P_b = 2/4$ ， $P_c = 1/4$ 。好，让我们按照新的概率分布比例划分0.3333-0.6667这一区间，划分的结果可以用图形表示为：



算术编码

例：考虑某条信息中可能出现的字符仅有 a b c 三种，我们要压缩保存的原始信息为
bccb

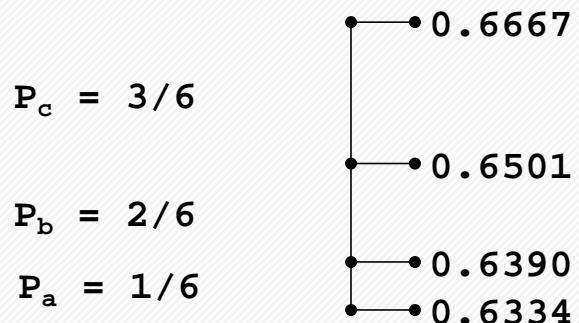
第三步：接着我们拿到字符c，我们现在要关注上一步中得到的c的区间 0.5834-0.6667。新添了 c以后，三个字符的概率分布变成 $P_a=1/5$ ， $P_b=2/5$ ， $P_c=2/5$ 。我们用这个概率分布划分区间 0.5834-0.6667：



算术解码

例：考虑某条信息中可能出现的字符仅有 a b c 三种，我们要压缩保存的原始信息为
bccb

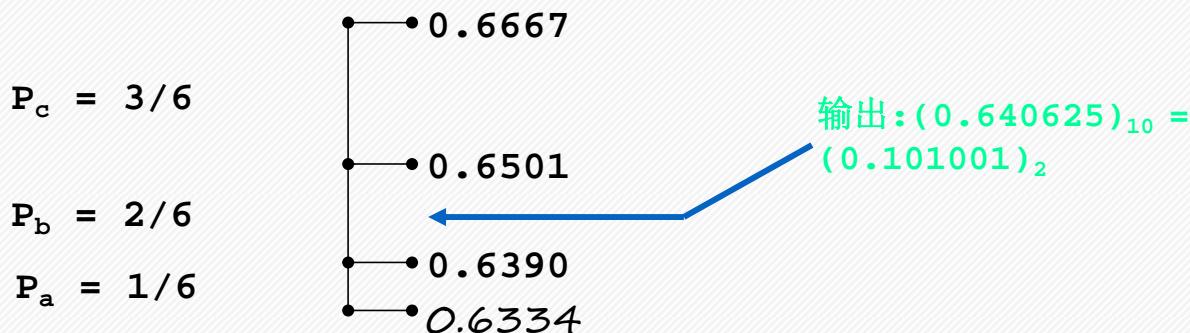
第四步：现在输入下一个字符c，三个字符的概率分布为： $P_a=1/6$ ， $P_b=2/6$ ， $P_c=3/6$ 。我们来划分c的区间0.6334-0.6667：



算术编码

例：考虑某条信息中可能出现的字符仅有 a b c 三种，我们要压缩保存的原始信息为
bccb

第五步：输入最后一个字符b，因为是最后一个字符，不用再做进一步的划分了，上一步中得到的b的区间为0.6390-0.6501，好，让我们在这个区间内随便选择一个容易变成二进制的数，例如0.640625，将它变成二进制0.101001，去掉前面没有太多意义的0和小数点，我们可以输出101001，这就是信息被压缩后的结果，我们完成了一次最简单的算术压缩过程



算术解码

例：考虑某条信息中可能出现的字符仅有 a b c 三种，我们要压缩保存的编码信息为
101001

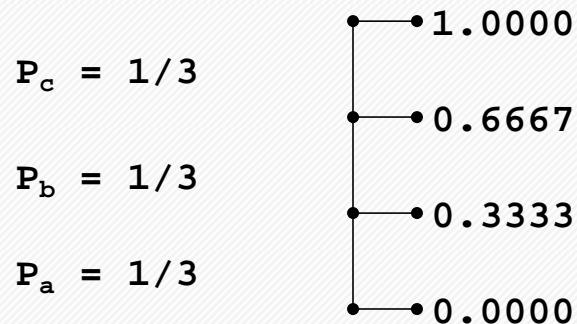
第一步：在101001前添加0.，变为0.101001，再将二进制码转为十进制码：
0.640625

算术解码

例：考虑某条信息中可能出现的字符仅有 a b c 三种，我们要压缩保存的编码信息为 101001

第二步：假设我们对 a b c 三者在信息中的出现概率一无所知，即认为三者的出现概率相等，也就是都为 $1/3$ ，我们将0-1区间按照概率的比例分配给三个字符。用图形表示就是：

由于编码为0.640625，我们获取到第一个字符为b。



算术解码

例：考虑某条信息中可能出现的字符仅有 a b c 三种，我们要压缩保存的编码信息为 101001

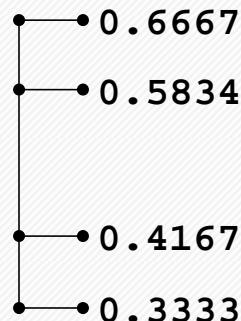
第三步：现在我们拿到第一个字符b，让我们把目光投向b对应的区间0.3333-0.6667。这时由于多了字符b，三个字符的概率分布变成： $P_a=1/4$ ， $P_b=2/4$ ， $P_c=1/4$ 。

由于编码为0.640625，我们获取到第二个字符为c

$$P_c = 1/4$$

$$P_b = 2/4$$

$$P_a = 1/4$$



算术解码

例：考虑某条信息中可能出现的字符仅有 a b c 三种，我们要压缩保存的编码信息为
101001

译码同编码进行的过程一致，不断更新概率表，根据概率表划分区间，判断数值在哪个区间，找到对应的字符。

算术编码扩展

0阶 普通模型

1阶

如果我们将模型变成统计符号在某个特定符号后的出现概率，那么，
模型就成为了 1 阶上下文自适应模型。

2阶、3阶.....

算术编码扩展

优点：

进一步压缩数据

缺点：

空间、查找时间

最理想的情况是采用 3 阶自适应模型。

此时，如果结合算术编码，对信息的压缩效果将达到惊人的程度。

采用更高阶的模型需要消耗的系统空间和时间至少在目前还无法让人接受。

算术编码自适应模型

初始概率

- 初始值均设为1
- 转义码

“转义码”是混在压缩数据流中的特殊的记号，用于通知解压缩程序下一个上下文在此之前从未出现过，需要使用低阶的上下文进行编码。

算术编码自适应模型

存储空间问题

采用**数组结构**存储所有可能出现的上下文。

采用**树结构**存储所有出现过的上下文。

- 将 0 阶上下文表存储在数组中，每个数组元素包含了指向相
应的 1 阶上下文表的指针，1 阶上下文表中又包含了指向 2
阶上下文表的指针.....
- 只有出现过的上下文才拥有已分配的节点，没有出现过的上
下文不必占用内存空间。

算术编码评价

1. 概率大的符号对应的区间大，描述所需的空间小。
2. 随着输入序列长度增加，平均编码所用空间趋于信源熵。

算术编码的局限性

1. 算术编码过程中的移位和输出不均匀，需要缓冲存储器。因此只适用于分段信息。
2. 算术编码每次递推都要做乘法，并且必须在一个信源符号的处理周期内完成，有时难以实时。
3. 算术编码是一种**对错误敏感**的编码方法，如有一位出错，就会导致整个消息译错。要求高质量的信道，或采取检错反馈方式。
3. 相比huffman实现更复杂，特别是硬件实现。

算术编码的压缩效率

我们知道，表示一个 $[0, 1)$ 的小数 l_N 至少需要 $-\log_2 l_N$ 二进制位。

假设 l_N 为我们最终得到的子区间长度，那么我们所需的那个数就可以用 $-\log_2 l_N$ 个二进制位表示，为什么呢。

假设最终的区间为 $[0.101100 \sim 0.101101 \sim]$ ，那么区间长度为 0.00000^\sim

我们至少需要 $-\log_2 l_N$ 个二进制位表示，而我们最终需要的 0.10110 位数比 0.00000^\sim 少，故而可以用 $-\log_2 l_N$ 个二进制位表示。

$l_N = \prod_{k=1}^N p(s_k)$, 其中 $p(s_k)$ 表示我们编码的第 k 个字符的频率。

故而我们每个字符平均需要的位数 $B_s = \frac{-\log_2 l_N + \sigma}{N}$ ，其中 σ 是个常数

因为你编码中必须得存一下字符频率等等相关信息，这些信息也是占用空间的。

算术编码的压缩效率

and thus

$$B_S \leq \frac{\sigma - \sum_{k=1}^N \log_2 p(s_k)}{N} \text{ bits/symbol.} \quad (1.23)$$

Defining $E\{\cdot\}$ as the expected value operator, the expected number of bits per symbol is

$$\begin{aligned} \bar{B} = E\{B_S\} &\leq \frac{\sigma - \sum_{k=1}^N E\{\log_2 p(s_k)\}}{N} = \frac{\sigma - \sum_{k=1}^N \sum_{m=0}^{M-1} p(m) \log_2 p(m)}{N} \\ &\leq H(\Omega) + \frac{\sigma}{N} \end{aligned} \quad (1.24)$$

Since the average number of bits per symbol cannot be smaller than the entropy, we have

$$H(\Omega) \leq \bar{B} \leq H(\Omega) + \frac{\sigma}{N}, \quad (1.25)$$

and it follows that

$$\lim_{N \rightarrow \infty} \{\bar{B}\} = H(\Omega), \quad (1.26)$$

算术编码评价

s 表示待编码的符号串， b 表示编码 s 所需要的位数。当 s 变长，其概率 $P(s)$ 就变小， b 就变长。因此 b 的增长速度将与 $\log_2 P(s)$ 的缩小速度相同。因此两者的积应为常数或接近一个常数。信息论已经证明了下面的不等式

$$2 \leq 2bP(s) < 4$$

同时去对数得

$$1 - \log_2 P(s) \leq b < 2 - \log_2 P(s)$$

随着 s 变长，其概率 $P(s)$ 就缩小， $-\log_2 P(s)$ 也就变成了一个大数，所以 b 的极限就是 $-\log_2 P(s)$ 。

而 $-\log_2 P(s)$ 是整个字符串的熵，所以算术编码理论上讲能把符号串压缩到其理论极限。

算术编码 扩位

由于实际的计算机的精度不可能无限长，运算中出现溢出是一个明显的问题。

解决：

1. 多数机器都有16位、32位或者64位的精度，因此这个问题可使用比例缩放方法解决。
2. 我们注意到，要是区间的上下界中前面几个字符是一样的，那么以后编码的时候它们还是一样不变的。举个例子，要是编码区间为 $[0.1101, 0.1111]$ ，那么后来再怎么编码，得到的区间还是 $[0.11\sim, 0.11\sim)$ 前面几个字符是一样的。那么我们是不是可以进行输出了呢，这样就可以避免溢出啦！

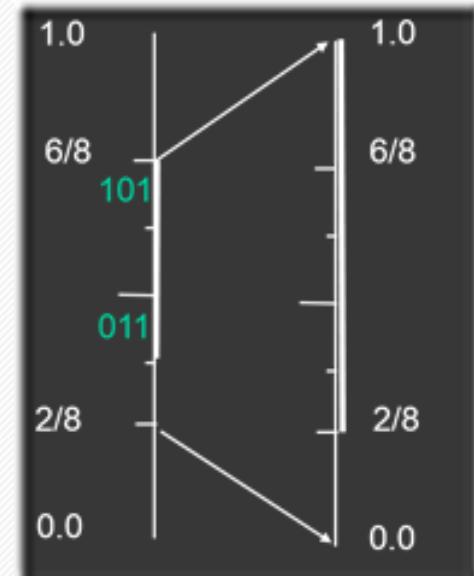
算术编码 扩位

区间为 $[0.10\sim, 0.01\sim)$ 始终这样下去的话，上述方法是无用的。

0	1	1	0	0	1	1	1	1	1	1	1	1	0	0
													1	0
0	1	1	0	1	0	0	0	0	1	0	0	1	1	

我们先看这个例子，假设区间是 $[0.011, 0.101)$ ，那么画图来看的话区间就是处于 $[3/8, 6/8)$ 之间，我们将原先区间的 $[2/8, 6/8)$ 放大一倍，那么此时原先的子区间就变成了 $[2/8, 1)$ ，可以参见右图。

我们注意到放大后，如果编码下一个字符的时候，子区间存在于上半部分，也就是上图右边 $[4/8, 1)$ 之间，那么也就是上图左边 $[4/8, 6/8)$ 的位置，这个部分的编码为10，所以输出10。



算术编码 扩位

首先记录一下从 $[2/8, 6/8]$ 放大到区间 $[0, 1]$ 的次数

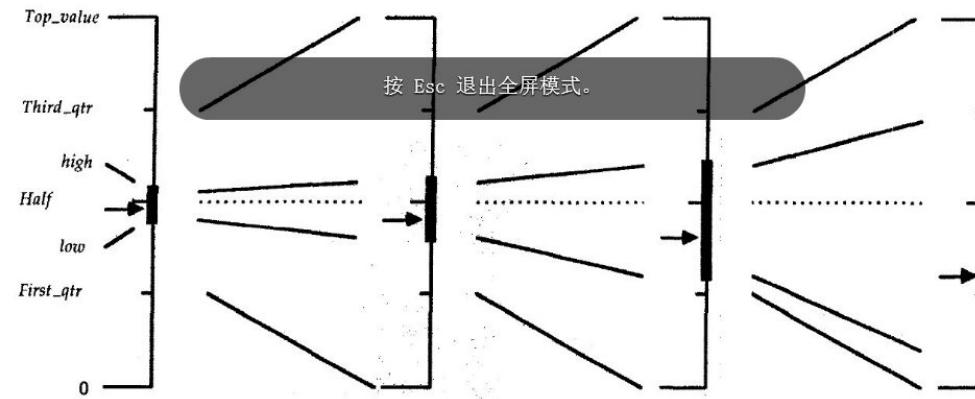
`bits_to_follow`，直到区间长度大于0.5为止。

然后开始编码下一个字符，如果区间存在于上半部，则输出10000，其中0的个数为`bits_to_follow`个。

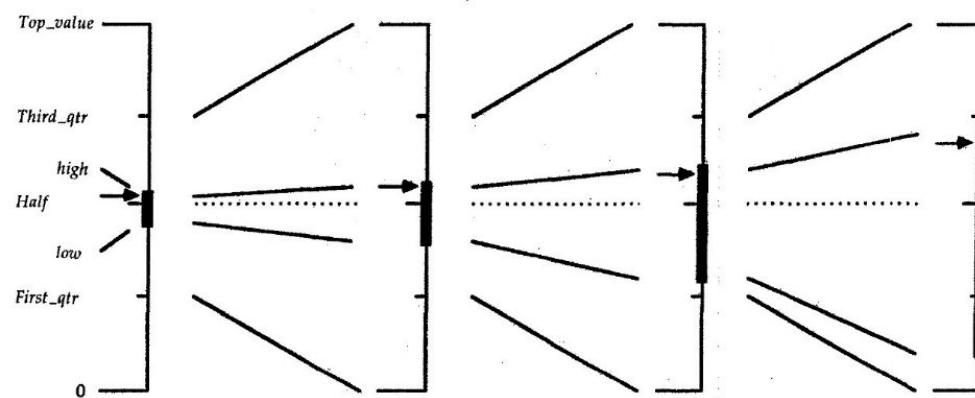
如果区间存在于下半部，则输出01111，其中1的个数为

`bits_to_follow`个。如果区间位于 $[2/8, 6/8]$ 则继续放大，`bits_to_follow`也随之增加。

(a)



(b)



过
渡
页



1

压缩基础

2

数据压缩

3

图片压缩

4

视频压缩

LZ系列算法

- 《A Universal Algorithm for Sequential Data Compression》

By Jacob Ziv & Abraham Lempel 1977

他们提出的算法称为LZ77，之后又提出很多变种算法，基本都以LZ开头，包括gif使用的LZW。LZ及以后的改进算法，将变长的符号串映射为定长的编码。

LZ77

- LZ77算法使用“滑动窗口压缩”
- 将部分字符串用类似指针的结构所替代

LZ77

- 1、从当前压缩位置开始，考察未编码的数据，并试图在滑动窗口中找出最长的匹配字符串，如果找到，则进行步骤 2，否则进行步骤 3.
- 2、输出三元符号组（ $\text{off}, \text{len}, \text{c}$ ）。其中 off 为窗口中匹配字符串相对窗口边界的偏移， len 为可匹配的长度， c 为下一个字符，即不匹配的第一个字符。然后将窗口向后滑动 $\text{len}+1$ 个字符，继续步骤 1.
- 3、输出三元符号组（0,0,c）。其中 c 为下一个字符。然后将窗口向后滑动一个字符，继续步骤 1.

LZ77

窗口大小10

abcdbbccaa aba
eaaaabaee



<off=0,len=2,c=a>

LZ77

dbbccaaaba eaaabae



<off=0,len=0,c=e>

LZ77

bbccaaabae aaabaee



<off=4,len =6,c=e>

LZW

- 从原始文件中提取出一些字符，并用这些字符创建一个编码表。
- 编码表一般每项为12个bit，这样编码表一共有4096项。0~255一般是固定的，为ASCII码值。256通常表示新的编码表的开始，257表示压缩结束，写在压缩文件尾。从258开始，每个值都代表一个字符串（长度大于等于2）

LZW

编码表每一项的结构

```
struct
```

```
{
```

```
    bool used ; //该项是否被编码。
```

```
    int prev; //前缀编码索引(0~4095)。
```

```
    char c; //本项对应字符。
```

```
}
```

在一开始初始化编码表0~255项，设置prev为null，c为对应的字符

LZW

步骤1：开始时的词典包含所有可能的根(Root)，设置他们的前缀P为空；

步骤2：读取第一个字符C，并当做下一个字符的前缀；

步骤3：判断前缀+字符P+C是否在词典中

(1) 如果“是”： $P := P + C$ // (将两者合并，称为下一个字符的前缀)；

(2) 如果“否”

① 把代表当前前缀P的码字输出到码字流；

② 把P+C添加到词典；

③ 令 $P = C$ ；

LZW

• ababcabcd

输出：97

编码表

编码	前缀	当前字符
258	97'a'	b

LZW

• **abbcabcd**

输出： 97 98

编码表

编码	前缀	当前字符
258	97'a'	b
259	98'b'	a

LZW

• **ababcabcd**

输出：97 98 258

编码表

编码	前缀	当前字符
258	97'a'	b
259	98'b'	a
260	258'ab'	c

LZW

•ababccabcd

输出：97 98 258 99

编码表

编码	前缀	当前字符
258	97'a'	b
259	98'b'	a
260	258'ab'	c
261	99'c'	c

LZW

•ababcabcd

输出: 97 98 258 99 260 100

编码表

编码	前缀	当前字符
258	97'a'	b
259	98'b'	a
260	258'ab'	c
261	99'c'	c
262	260'abc'	c



1

压缩基础

2

数据压缩

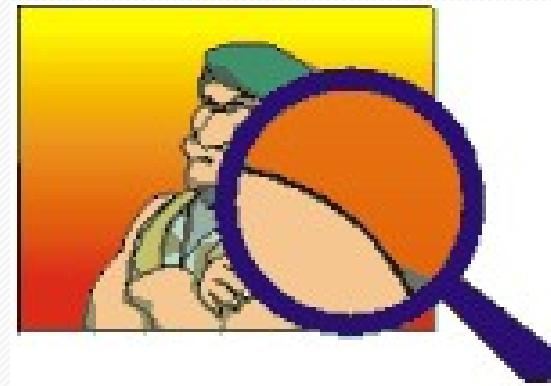
3

图像压缩

4

视频压缩

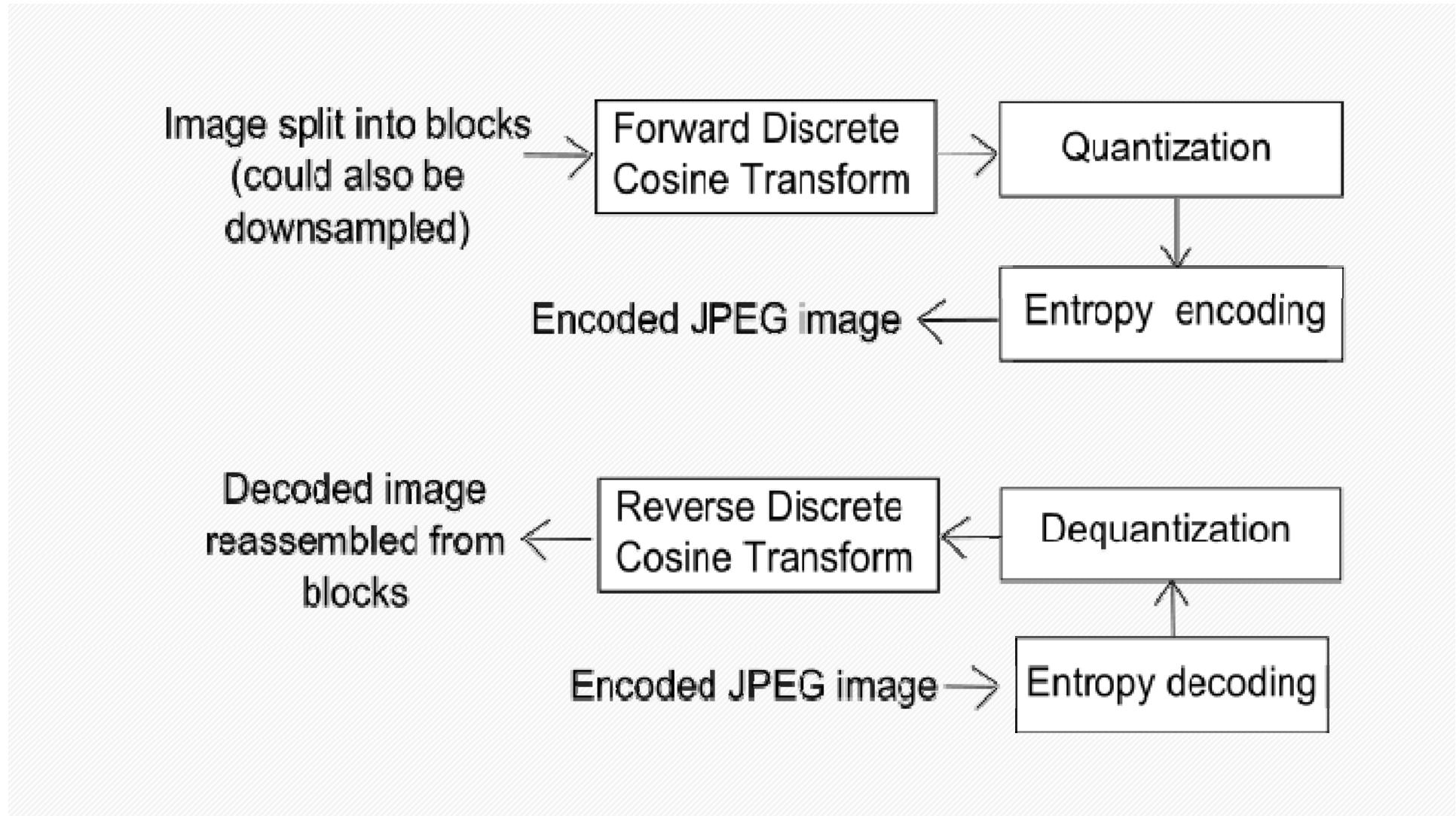
- 位图：像素点阵
- 矢量图：参数



- 有损压缩——JPEG
- 无损压缩——PNG, GIF

JPEG

- **Joint Photographic Expert Group**
- 有损压缩格式
- 允许用不同的压缩比例对文件进行压缩，支持多种压缩级别，压缩比率通常在10: 1到40: 1之间



颜色模式转换 (Color Space Transformation)

- RGB -> YCbCr色彩系统 (Y 亮度brightness, Cb, Cr代表色度chrominance(分别对于蓝色和红色))

$$Y' = 0.299 \cdot R' + 0.587 \cdot G' + 0.114 \cdot B'$$

$$P_B = -0.168736 \cdot R' - 0.331264 \cdot G' + 0.5 \cdot B'$$

$$P_R = 0.5 \cdot R' - 0.418688 \cdot G' - 0.081312 \cdot B'$$

采样 (Downsampling)

- 人眼对亮度变换的敏感度要比对色彩变换的敏感度高出很多
- Y分量比Cb, Cr分量重要
- **RGB888 -> YUV411和YUV422**
- 接下来的操作Y, Cb, Cr被分开处理

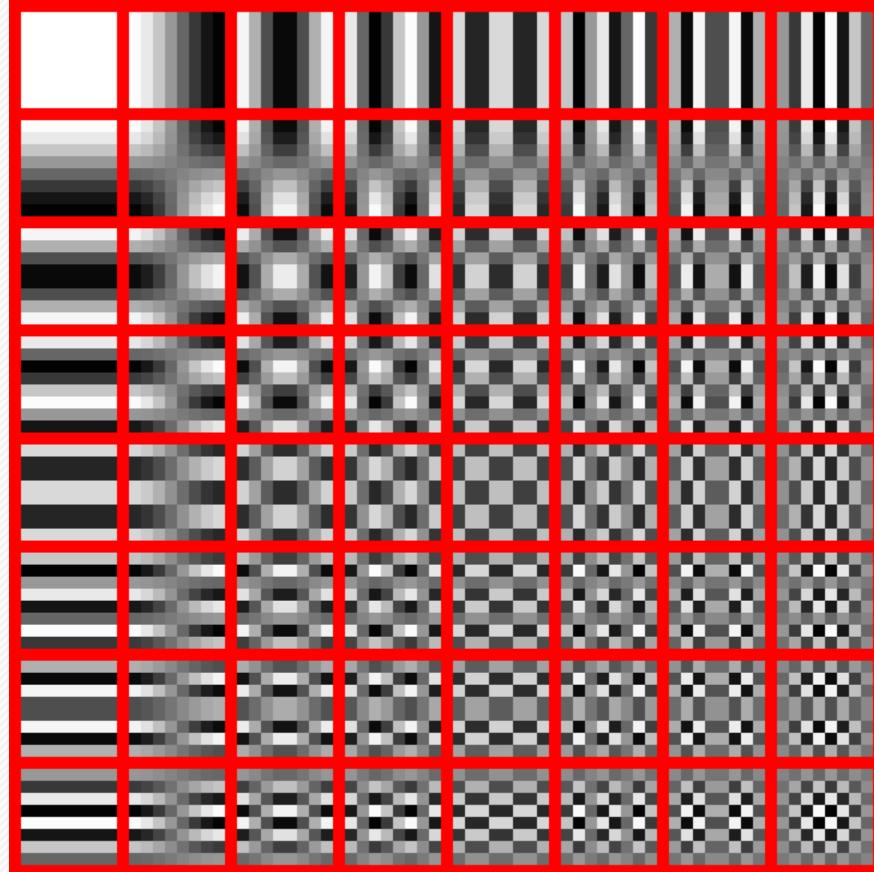
分块 (Block Splitting)

- 将程序分成**8*8**的小块，长或宽不足**8**的倍数需要补足(比如用黑色块)

离散余弦变换（Discrete Cosine Transform）

- 类似离散傅立叶变换，但是只使用实数
- 能量集中特性，大多数的自然信号(包括声音和图像)的能量都集中在离散余弦变换后的低频部分，人眼对高频部分不敏感（只涉及细节部分），因此将这些高频部分抛弃
- 图像数据转换为**DCT**频率系数

DCT把8*8的输入
变成一个这64种模
式的组合，这64种
模式就是二维的
DCT的基础函数，
输出的值就是转换
的系数



$$\begin{bmatrix} 52 & 55 & 61 & 66 & 70 & 61 & 64 & 73 \\ 63 & 59 & 55 & 90 & 109 & 85 & 69 & 72 \\ 62 & 59 & 68 & 113 & 144 & 104 & 66 & 73 \\ 63 & 58 & 71 & 122 & 154 & 106 & 70 & 69 \\ 67 & 61 & 68 & 104 & 126 & 88 & 68 & 70 \\ 79 & 65 & 60 & 70 & 77 & 68 & 58 & 75 \\ 85 & 71 & 64 & 59 & 55 & 61 & 65 & 83 \\ 87 & 79 & 69 & 68 & 65 & 76 & 78 & 94 \end{bmatrix}$$

$$g = \begin{array}{c} \xrightarrow{x} \\ g = \begin{bmatrix} -76 & -73 & -67 & -62 & -58 & -67 & -64 & -55 \\ -65 & -69 & -73 & -38 & -19 & -43 & -59 & -56 \\ -66 & -69 & -60 & -15 & 16 & -24 & -62 & -55 \\ -65 & -70 & -57 & -6 & 26 & -22 & -58 & -59 \\ -61 & -67 & -60 & -24 & -2 & -40 & -60 & -58 \\ -49 & -63 & -68 & -58 & -51 & -60 & -70 & -53 \\ -43 & -57 & -64 & -69 & -73 & -67 & -63 & -45 \\ -41 & -49 & -59 & -60 & -63 & -52 & -50 & -34 \end{bmatrix} \end{array} \downarrow y.$$

$$G_{u,v} = \frac{1}{4}\alpha(u)\alpha(v) \sum_{x=0}^7 \sum_{y=0}^7 g_{x,y} \cos\left[\frac{(2x+1)u\pi}{16}\right] \cos\left[\frac{(2y+1)v\pi}{16}\right]$$

where

- u is the horizontal spatial frequency, for the integers $0 \leq u < 8$.
- v is the vertical spatial frequency, for the integers $0 \leq v < 8$
- $\alpha(u) = \begin{cases} \frac{1}{\sqrt{2}}, & \text{if } u = 0 \\ 1, & \text{otherwise} \end{cases}$ is a normalizing scale factor to make the transformation orthonormal
- $g_{x,y}$ is the pixel value at coordinates (x, y)
- $G_{u,v}$ is the DCT coefficient at coordinates (u, v) .

DC系数 (DC coefficient) : 决定了这个块的基础色调 u

$$G = \begin{bmatrix} -415.38 & -30.19 & -61.20 & 27.24 & 56.12 & -20.10 & -2.39 & 0.46 \\ 4.47 & -21.86 & -60.76 & 10.25 & 13.15 & -7.09 & -8.54 & 4.88 \\ -46.83 & 7.37 & 77.13 & -24.56 & -28.91 & 9.93 & 5.42 & -5.65 \\ -48.53 & 12.07 & 34.10 & -14.76 & -10.24 & 6.30 & 1.83 & 1.95 \\ 12.12 & -6.55 & -13.20 & -3.95 & -1.87 & 1.75 & -2.79 & 3.14 \\ -7.73 & 2.91 & 2.38 & -5.94 & -2.38 & 0.94 & 4.30 & 1.85 \\ -1.03 & 0.18 & 0.42 & -2.42 & -0.88 & -3.02 & 4.12 & -0.66 \\ -0.17 & 0.14 & -1.07 & -4.19 & -1.17 & -0.10 & 0.50 & 1.68 \end{bmatrix} \xrightarrow{v.}$$

AC系数

量化 (Quantization)

- 人的眼睛不擅于识别高频率的亮度变化，因此可以丢掉高频的信息
- 两个**8*8**的矩阵：处理亮度的频率系数；针对色度的频率系数
- 将频率系数除以量化矩阵的值之后取整
- 在**JPEG**算法中，由于对亮度和色度的精度要求不同，分别对亮度和色度采用不同的量化表。前者细量化，后者粗量化。
- 数字越大，压缩率越大
- 量化表是控制 **JPEG** 压缩比的关键
- 除掉了一些高频量，损失了很多细节信息。

$$B_{j,k} = \text{round} \left(\frac{G_{j,k}}{Q_{j,k}} \right) \text{ for } j = 0, 1, 2, \dots, 7; k = 0, 1, 2, \dots, 7$$

where G is the unquantized DCT coefficients; Q is the quantization matrix above; and B is the quantized DCT coefficients.

$$\begin{bmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{bmatrix}$$

$$B = \begin{bmatrix} -26 & -3 & -6 & 2 & 2 & -1 & 0 & 0 \\ 0 & -2 & -4 & 1 & 1 & 0 & 0 & 0 \\ -3 & 1 & 5 & -1 & -1 & 0 & 0 & 0 \\ -3 & 1 & 2 & -1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

熵编码（Entropy Coding）

- 分别处理**DC&AC**

DC系数的差分脉冲调制编码

- 8*8的图像块经过DCT变换之后得到的DC系数有两个特点：
 - 系数的数值比较大；
 - 相邻的8*8图像块的DC系数值变化不大；
- 采用差分脉冲调制编码DPCM（Difference Pulse Code Modulation）
 - 取同一个图像分量中每个DC值与前一个DC值的差值来进行编码。对差值进行编码所需要的位数会比对原值进行编码所需要的位数少了很多。

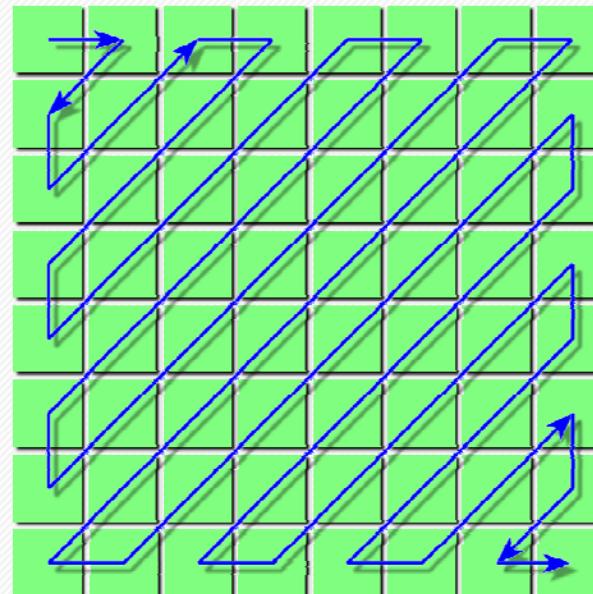
中间格式的计算 (DC&AC)

- JPEG中为了更进一步节约空间，并不直接保存数据的具体数值，而是将数据按照位数分为**16**组，保存在表里面。这也就是所谓的变长整数编码**VLI**。
 - 找到对应组
 - 从小到大的顺序

数值	组	实际保存值
0	0	-
-1, 1	1	0, 1
-3, -2, 2, 3	2	00, 01, 10, 11
-7, -6, -5, -4, 4, 5, 6, 7	3	000, 001, 010, 011, 100, 101, 110, 111
-15, .., -8, 8, .., 15	4	0000, .., 0111, 1000, .., 1111
-31, .., -16, 16, .., 31	5	00000, .., 01111, 10000, .., 11111
-63, .., -32, 32, .., 63	6	-
-127, .., -64, 64, .., 127	7	-
-255, .., -128, 128, .., 255	8	-
-511, .., -256, 256, .., 511	9	-
-1023, .., -512, 512, .., 1023	10	-
-2047, .., -1024, 1024, .., 2047	11	-
-4095, .., -2048, 2048, .., 4095	12	-
-8191, .., -4096, 4096, .., 8191	13	-
-16383, .., -8192, 8192, .., 16383	14.	-
-32767, .., -16384, 16384, .., 32767	15.	-

Zigzag 扫描排序 (“Zigzag” Ordering)

将频率相近的放在一起，以便于将来使用RLE算法



The figure shows a directed graph with 10 nodes arranged in two rows of five. Directed edges connect node i to node j if the matrix entry at row i and column j is non-zero. The edges are colored green.

- 顺序式编码（Sequential Encoding）：encodes coefficients of a single block at a time (in a zigzag manner).
- 递增式编码（Progressive Encoding）：encodes similar-positioned coefficients of all blocks in one go, followed by the next positioned coefficients of all blocks, and so on. It should be noted here that once all similar-positioned coefficients have been encoded, the next position to be encoded is the one occurring next in the zigzag traversal as indicated in the figure above.

行程编码算法 (RLE/RLC)

- Run-length encoding

- x is the non-zero, quantized AC coefficient.
- $RUNLENGTH$ is the number of zeroes that came before this non-zero AC coefficient.
- $SIZE$ is the number of bits required to represent x .
- $AMPLITUDE$ is the bit-representation of x .

Symbol 1 Symbol 2
(RUNLENGTH, SIZE) (AMPLITUDE)

RUNLENGTH和SIZE一共占用一个byte（各4个bit）所以有连续超过16个0的时候拆出一个(15, 0)

SIZE和AMPLITUDE是对原值进行VLI之后得到的组数（即长度）

EOB (0,0) 结束符

AC系数的行程长度编码(RLC)

- 63个系数中含有很多值为0的系数。因此，可以采用行程编码RLC（Run Length Coding）来更进一步降低数据的传输量。
- Eg: 5,0,0,6,0,7,8 -> (0, 5), (2, 6), (1, 7), (0, 8)

最终结果

$(0, 2)(-3); (1, 2)(-3); (0, 2)(-2); (0, 3)(-6); (0, 2)(2); (0, 3)(-4);$

$(0, 1)(1); (0, 2)(-3); (0, 1)(1); (0, 1)(1); (0, 3)(5); (0, 1)(1); (0, 2)(2);$

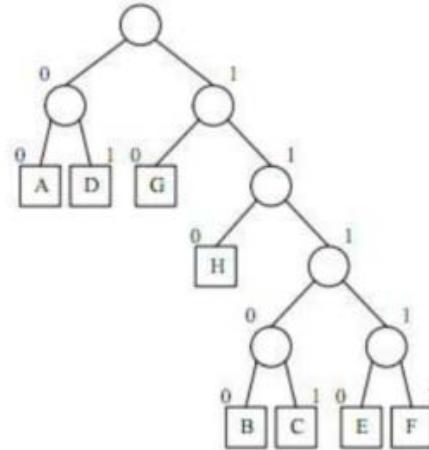
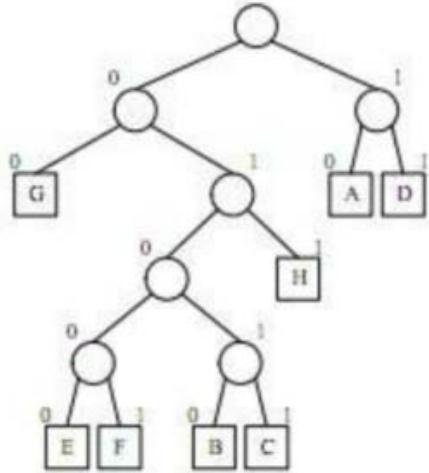
$(0, 1)(-1); (0, 1)(1); (0, 1)(-1); (0, 2)(2); (5, 1)(-1); (0, 1)(-1); (0, 0).$

熵编码

- JPEG标准具体规定了两种熵编码方式：**Huffman**编码和算术编码。
- 一般使用熵编码
- **Huffman**编码时**DC**系数与**AC**系数分别采用不同的**Huffman**编码表，对于亮度和色度也采用不同的**Huffman**编码表。因此一共有四颗哈夫曼树
- 将(**RUNLENGTH**, **SIZE**)进行哈夫曼编码

范式哈夫曼编码（Canonical Huffman Code）

- 特点：可根据编码位长算出编码
- 中心思想：使用某些强制约定，仅通过很少的数据便能重构出哈夫曼编码树的结构
- 让所有缺的点在左边



序号	同组序号	符号	位长	编码
0	0	A	2	00
1	1	D	2	01
2	2	G	2	10
3	0	H	3	110
4	0	B	5	11100
5	1	C	5	11101
6	2	E	5	11110
7	3	F	5	11111

Huffman表

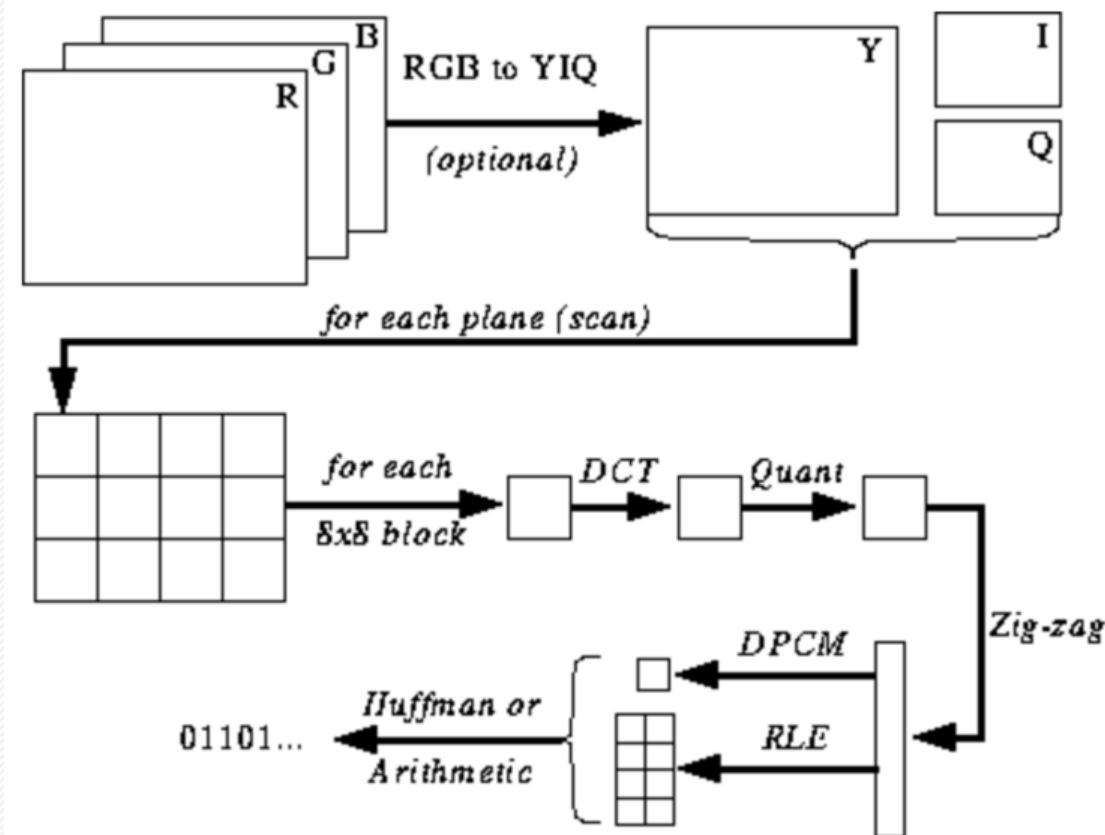
- DHT (Define Huffman Table)
- 标记代码: 2bytes 0xFFC4
- 数据长度: 2bytes
- 哈夫曼表: 数据长度 - 2bytes
 - 表ID&类型: 1bytes
 - 0x00表示DC直流0号表;
 - 0x01表示DC直流1号表;
 - 0x10表示AC交流0号表;
 - 0x11表示AC交流1号表。
 - 不同位数的码字数量 16bytes
 - 编码内容 码字数量之和bytes

哈夫曼表的解析

- 在读出哈夫曼表的数据后，建立哈夫曼树：
 - 第一个码字位数为n，则它的码字为n个0
 - 第k+1个码字位数如果和第k个码字位数相同，则它的码字为第k个的码字+1，如果位数不相同则在+1后补足位数

11 00 02 02 00 05 01 06 01 00 00 00 00 00 00 00
00 01 11 02 21 03 31 41 12 51 61 71 81 91 22 13 32

序号	码字长度	码字	权值
1	2	00	0x00
2	2	01	0x01
3	3	100	0x11
4	3	101	0x02
5	5	11000	0x21
6	5	11001	0x03
7	5	11010	0x31
8	5	11011	0x41
9	5	11100	0x12
10	6	111010	0x51
11	7	1110110	0x61
12	7	1110111	0x71
13	7	1111000	0x81
14	7	1111001	0x91
15	7	1111010	0x22
16	7	1111011	0x13
17	8	11111000	0x32



PNG

- Portable Network Graphic, PNG
- PNG's not GIF
- 无损压缩算法——采用LZ77算法的派生算法进行压缩
- 支持透明通道alpha

GIF

- **Graphics Interchange Format**
- 采用无损压缩算法——**LZW**压缩算法
- 采用隔行存放的方式，可以边解码边显示

参考资料

- <https://www.cs.cf.ac.uk/Dave/Multimedia/node234.html>
- <https://en.wikipedia.org/wiki/JPEG>
- <http://blog.csdn.net/carson2005/article/details/7753499>
- <http://wenku.baidu.com/view/b086ae4bcf84b9d528ea7a9b.html>



1

压缩基础

2

数据压缩

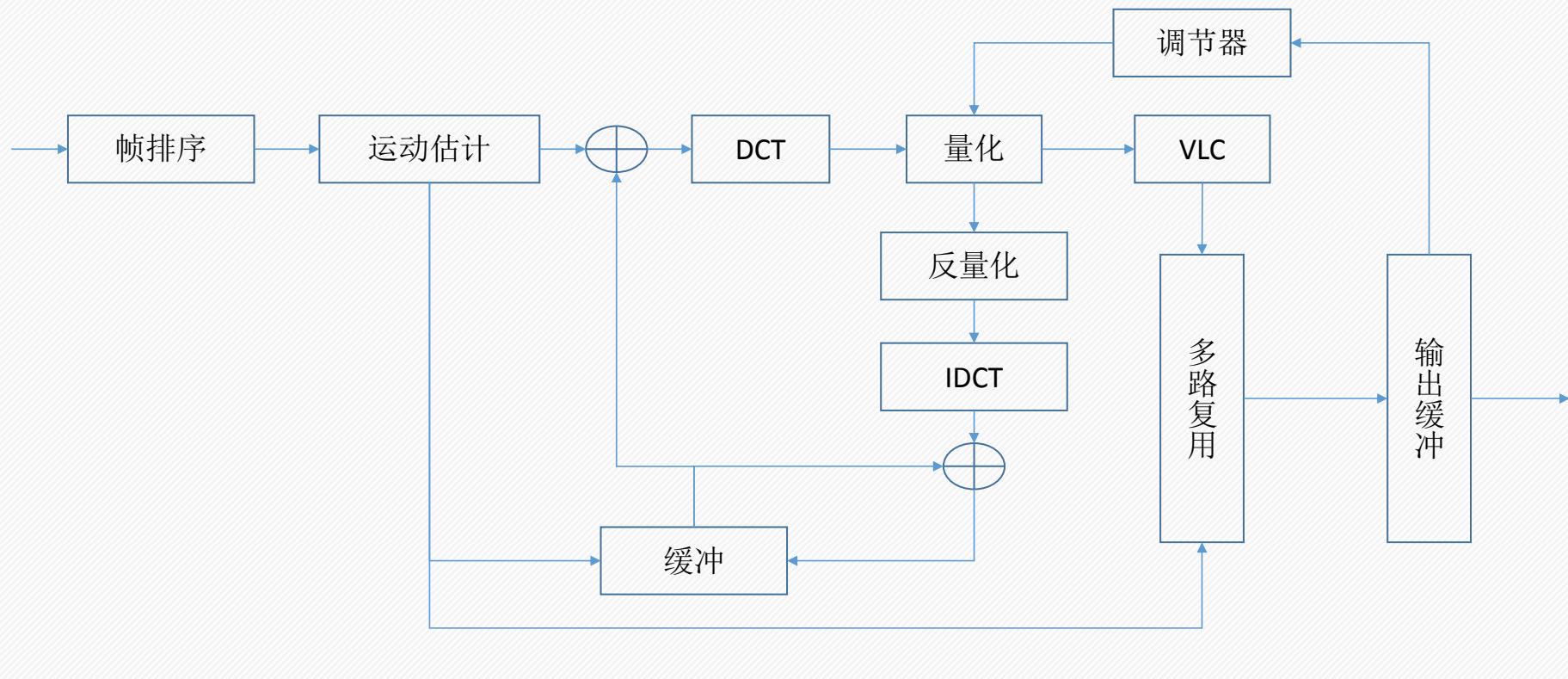
3

图片压缩

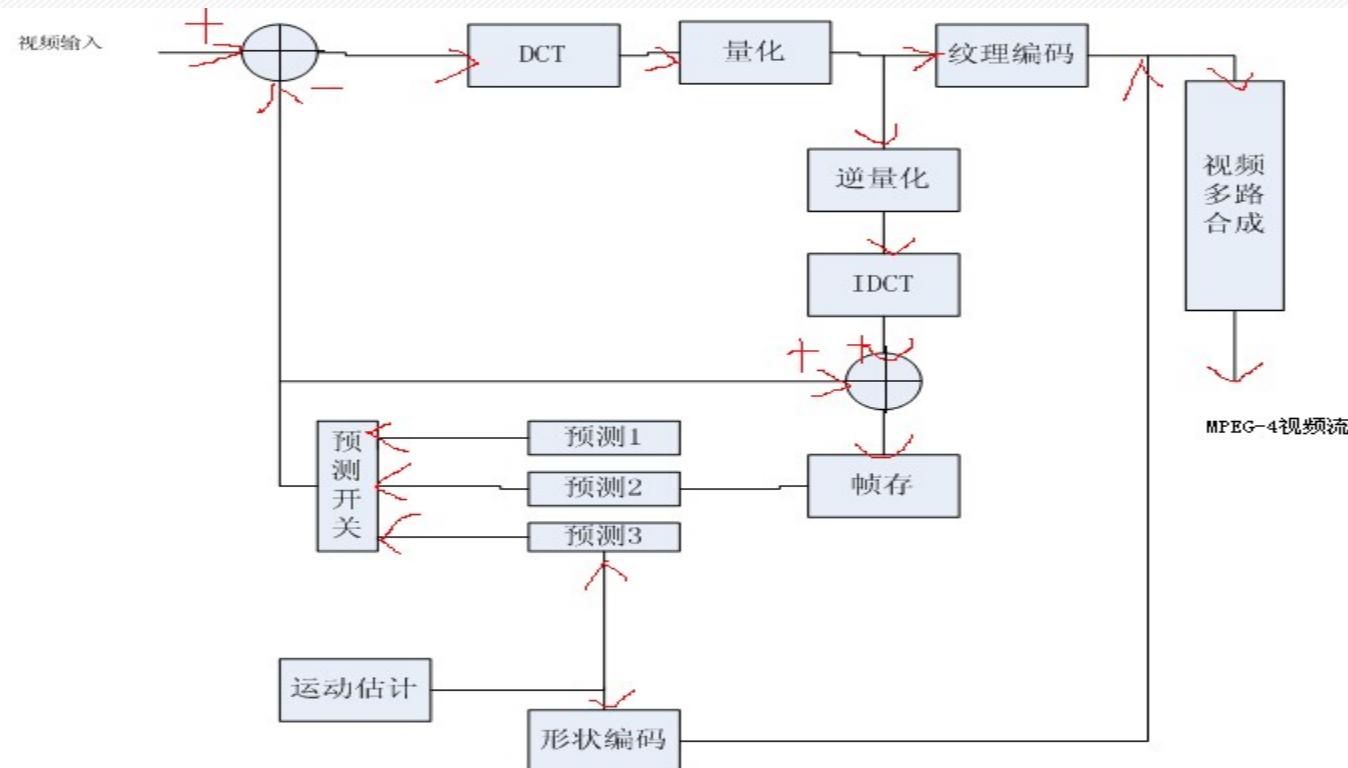
4

视频压缩

MPEG



MPEG4



H.264

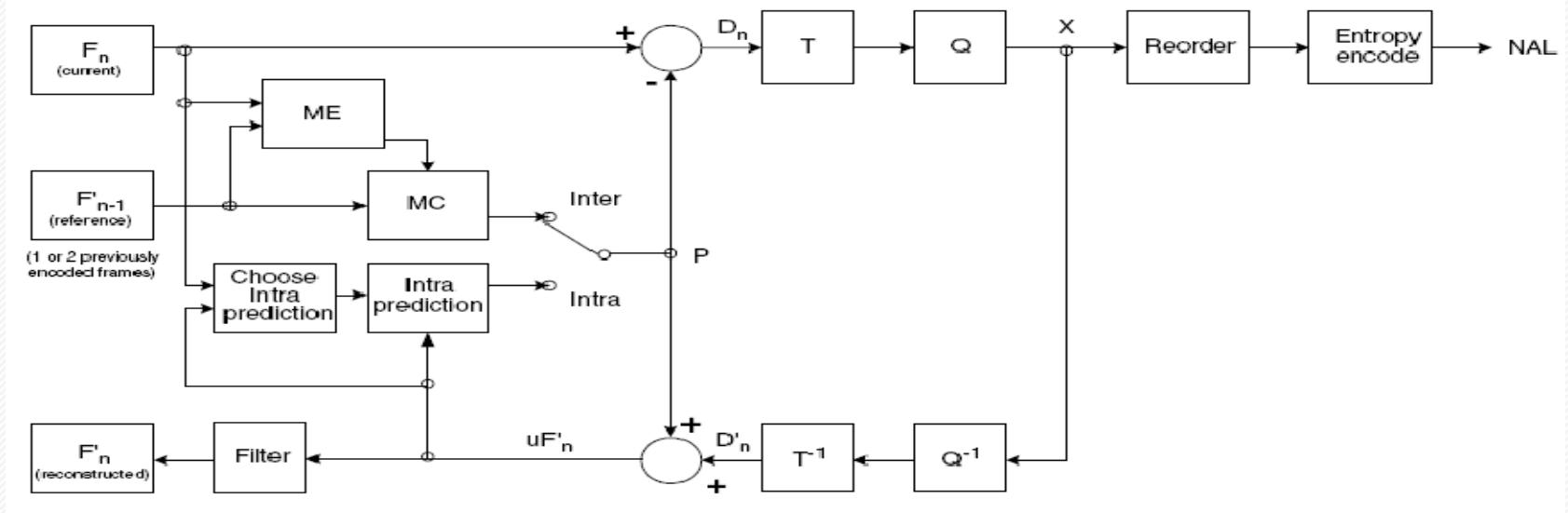


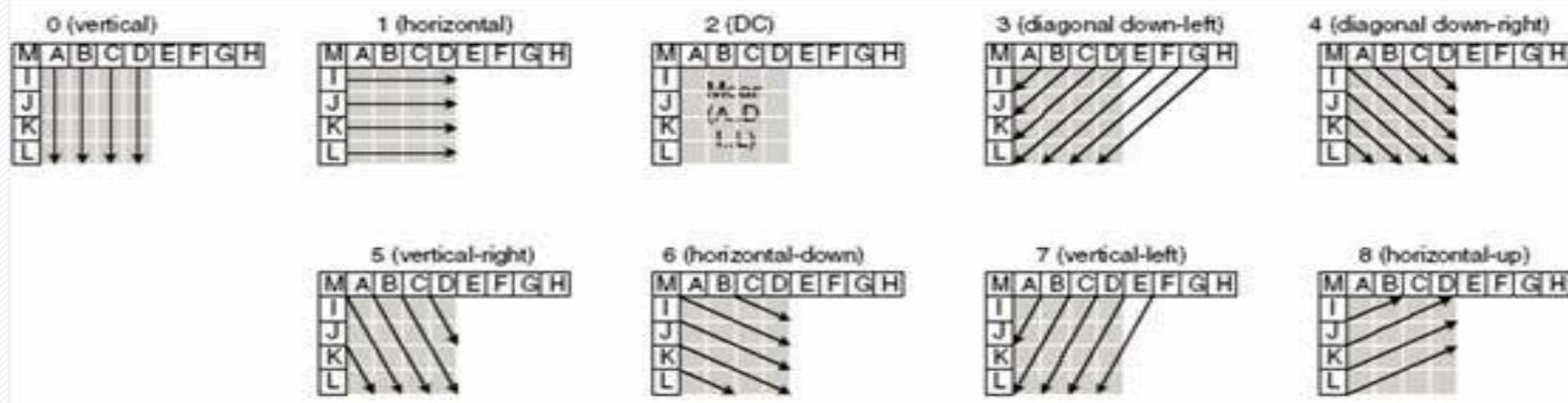
Figure 6.1 H.264 Encoder

帧内预测

4x4块：9种模板

16x16块：前4种模板

枚举所有模板，选择cost最小的一个



帧内预测 (Cont.)

Cost计算

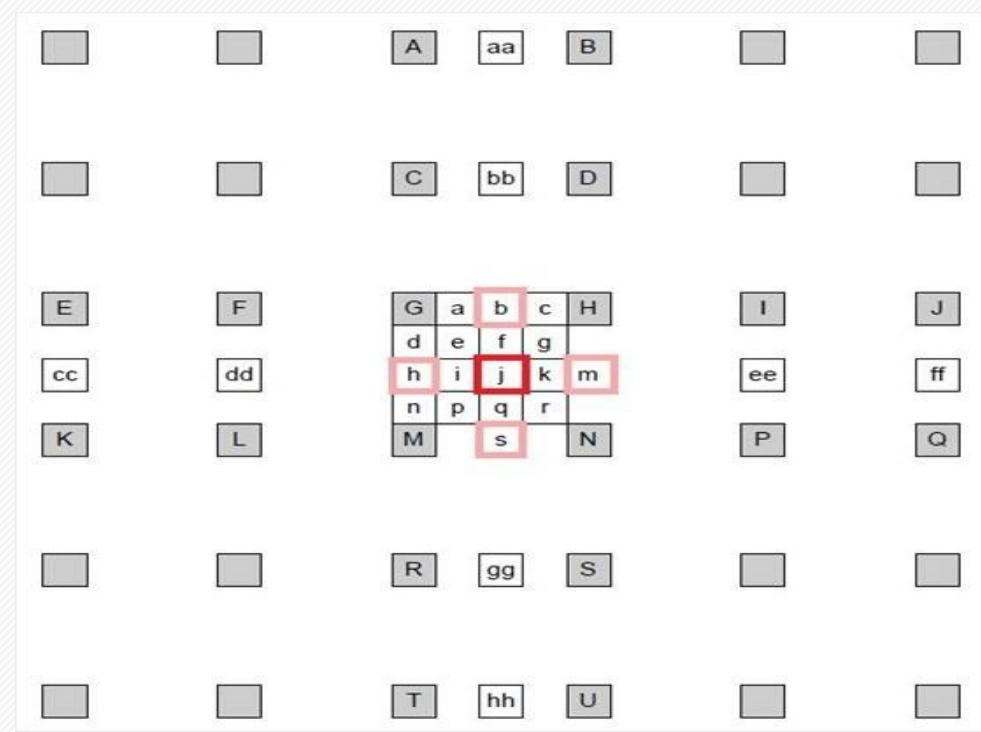
- SAD (Sum of Absolute Difference) 也可以称为SAE (Sum of Absolute Error)，即绝对误差和。它的计算方法就是求出两个像素块对应像素点的差值，将这些差值分别求绝对值之后再进行累加。
- SATD (Sum of Absolute Transformed Difference) 即Hadamard变换后再绝对值求和。它和SAD的区别在于多了一个“变换”。
- SSD (Sum of Squared Difference) 也可以称为SSE (Sum of Squared Error)，即差值的平方和。它和SAD的区别在于多了一个“平方”。

运动估计

1/4像素插值

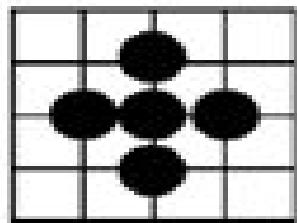
$$b = \text{round}((E - 5F + 20G + 20H - 5I + J) / 32)$$

$$a = \text{round}((G+b)/2)$$
$$e = \text{round}((b+h)/2)$$



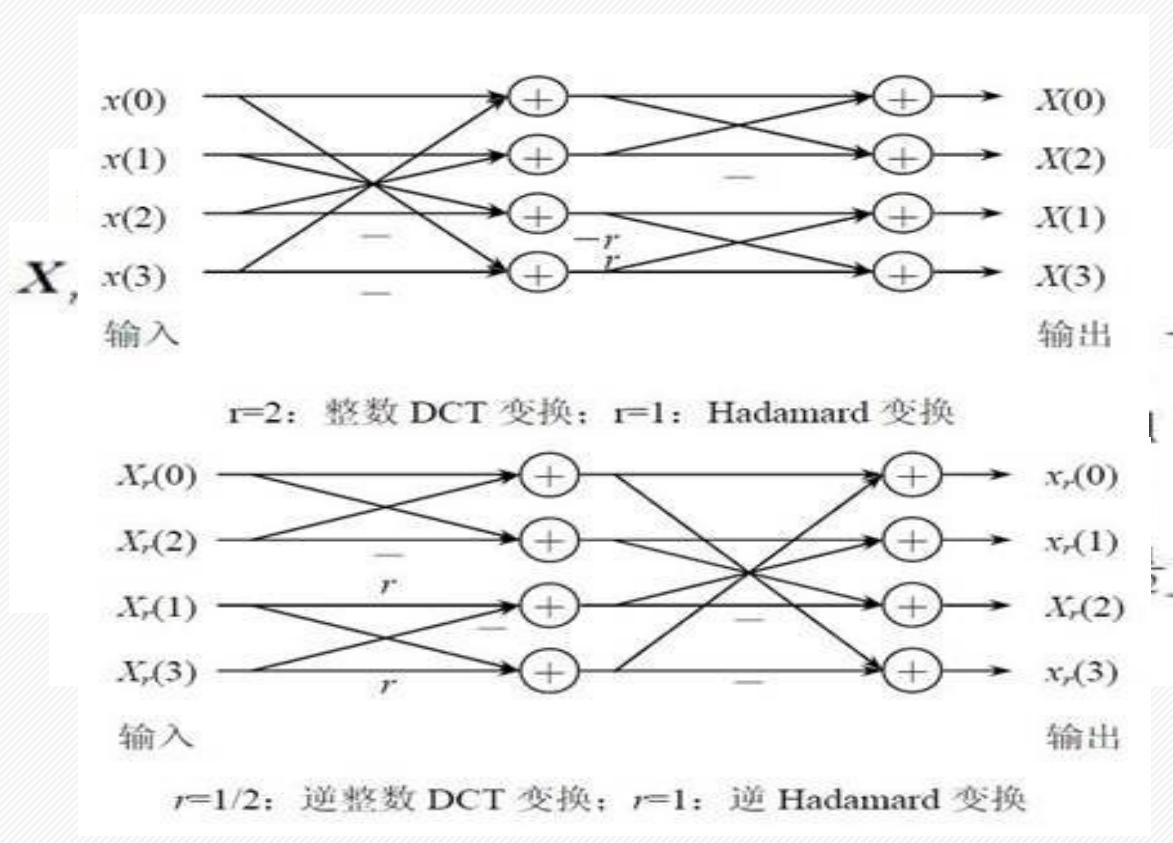
运动估计 (Cont.)

以搜索起点为中心，采用下图所示的小菱形模板（模板半径为1）搜索。计算各点的匹配误差，得到MBD（最小误差）点。如果MBD点在模板中心，则搜索结束，此时的MBD 点就是最优匹配点，对应的像素块就是最佳匹配块；如果MBD点不在模板中心位置，则以现在MBD 点为中心点，继续进行小菱形搜索，直至MBD点落在中心点为止。



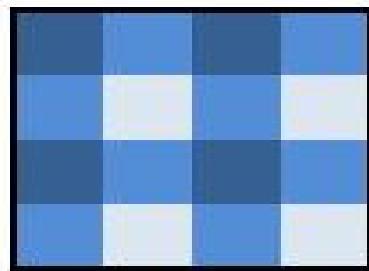
小菱形模板

DCT (离散余弦变换)



量化

$$|Z_{ij}| = (|W_{ij}| * MF + f) \gg q\text{bits}$$
$$\text{sign}(Z_{ij}) = \text{sign}(W_{ij})$$



value	PF	MF (QP=5)
small	$b*b/4$	2893
medium	$a*b/2$	4559
big	$a*a$	7282

熵编码

- CAVLC
- CABAC

- http://www.iqiyi.com/w_19rtmsdogd.html

THANKS!



Lecture 16 Review

What I've taught ≠ What you've learnt



Course goals

- To become proficient in the application of fundamental *algorithm design techniques*
- To gain familiarity with the main theoretical tools used in the *analysis of algorithms*
- To study and analyze different algorithms for many of “standard” *algorithmic problems*
- To introduce students to some of the *prominent subfields* of algorithmic study in which they may wish to pursue further study

Course contents

- Algorithm design techniques
 - Divide-and-conquer, Greedy, Dynamic Programming
- Analysis of algorithms
 - O , Ω , Θ , Worst-case and Average-case, Recurrences, Amortized analysis
- “Standard” algorithmic problems
 - Sorting, Graph theory, Network flow
- Prominent subfields
 - NP theory, Backtracking, Randomization, Approximation, Lower bound, Computational Geometry, Substring search, Data compression

Divide-and-conquer

- **What:** Divide-conquer-combine
- **When:** Non-overlapping subproblems
- **Examples:** Quicksort, mergesort, Select, Closest pair.....
- **Test:**

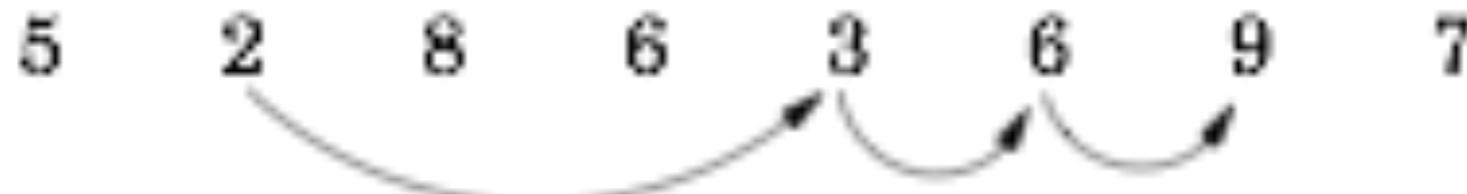
Given a sorted array of distinct integers $A[1, \dots, n]$, you want to find out whether there is an index i for which $A[i] = i$.

Greedy

- **What:** Step by step, always choose local optimal solution
- **When:** local optimal -> global optimal
- **Examples:** Kruskal, Prim, Huffman code, Fractional knapsack.....
- **Test:**
Show how to find the maximum spanning tree of a graph.

Dynamic programming

- **What:**
 1. write the paradigm: Problems -> subproblems
 2. solve it in a bottom-up way
- **When:** subproblems overlap
- **Examples:** Dijkstra, Knapsack, LCS, Matrix chain
- **Test:** *Longest increasing subsequence*



Course contents

- Algorithm design techniques
 - Divide-and-conquer, Greedy, Dynamic Programming
- Analysis of algorithms
 - O , Ω , Θ , Worst-case and Average-case, Recurrences, Amortized analysis
- “Standard” algorithmic problems
 - Sorting, Graph theory, Network flow
- Prominent subfields
 - NP theory, Backtracking, Randomization, Approximation, Lower bound, Computational Geometry, Substring search, Data compression

Complexity Analysis Tools

- Notation: O , Ω , Θ
- Recurrences and Master theorem

Master theorem

Simplified Master Theorem

Let $a \geq 1$, $b > 1$ and $c, d, w \geq 0$ be constants, and let $T(n)$ be defined on the nonnegative integers by the recurrence

$$f(n) = \begin{cases} w & \text{if } n = 1 \\ af(n/b) + cn^d & \text{if } n > 1 \end{cases}$$

Then

$$f(n) = \begin{cases} \Theta(n^d) & \text{if } a < b^d \\ \Theta(n^d \log n) & \text{if } a = b^d \\ \Theta(n^{\log_b a}) & \text{if } a > b^d \end{cases}$$

Complexity Analysis Tools

- **Notation:** O , Ω , Θ
- Recurrences and Master theorem
- Average-case, worst-case analysis
- Amortized analysis
- **Test:** $n!$ and 2^n ,
 $T(n) = 3T(n/2) + n$
 $T(n) = T(n-1) + n$

Course contents

- Algorithm design techniques
 - Divide-and-conquer, Greedy, Dynamic Programming
- Analysis of algorithms
 - O , Ω , Θ , Worst-case and Average-case, Recurrences, Amortized analysis
- “Standard” algorithmic problems
 - [Sorting](#), [Graph theory](#), [Network flow](#)
- Prominent subfields
 - NP theory, Backtracking, Randomization, Approximation, Lower bound, Computational Geometry, Substring search, Data compression

Sorting

	Average	Worst
Insertion	$O(n^2)$	$O(n^2)$
Selection	$O(n^2)$	$O(n^2)$
Bubble	$O(n^2)$	$O(n^2)$
Quick	$O(n \log n)$	$O(n^2)$
Bottom-up merge	$O(n \log n)$	$O(n \log n)$
Merge	$O(n \log n)$	$O(n \log n)$
Heap	$O(n \log n)$	$O(n \log n)$
Radix	$O(kn)$	$O(kn)$
Bucket	$O(n)$	$O(n^2)$

Test: Give an optimal algorithm for sorting

Graph Traversal

	Adj. Matrix	Adj. List	Application
BFS	$O(n^2)$	$O(n+m)$	Shortest path
DFS	$O(n^2)$	$O(n+m)$	Acyclicity Topological sort SCC

Test: Give a linear-time algorithm to find an odd-length cycle in a undirected graph.

Minimum Spanning Tree

	Without	With	DS
Kruskal	$O(n^2)$	$O(m \log m)$	Disjoint set
Prim	$O(n^2)$	$O(m \log n)$	Heap

Test: Show Prim or Kruskal is valid for graphs with *negative* weights.

Shortest path

Single-source	Graph types	Complexity
Dijkstra	Positive weight	$O(m \log n)$
Bellman-Ford	Any	$O(mn)$
Topological sort	DAG	$O(m+n)$
All-pairs	Floyd-Warshall	$O(n^3)$

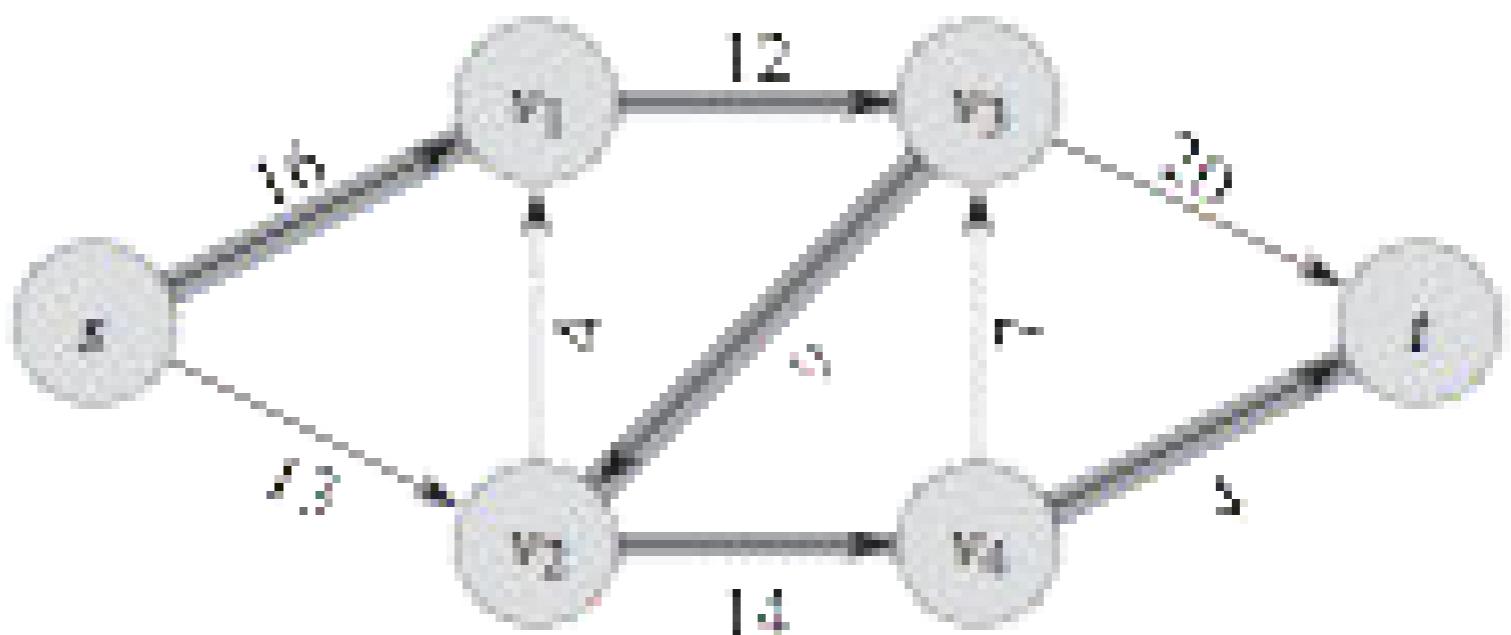
Test: Give an algorithm to find the shortest cycle containing some edge e .

Maximum flow

Ford–Fulkerson	$O(E f^*)$
Edmonds–Karp	$O(V^2E \log c^*), O(VE^2)$

Applications: Bipartite Matching, Edge/Vertex Disjoint path

Test: The value of max flow.



Course contents

- Algorithm design techniques
 - Divide-and-conquer, Greedy, Dynamic Programming
- Analysis of algorithms
 - O , Ω , Θ , Worst-case and Average-case, Recurrences, Amortized analysis
- “Standard” algorithmic problems
 - Sorting, Graph theory, Network flow
- Prominent subfields
 - NP theory, Backtracking, Randomization, Approximation, Lower bound, Computational Geometry, Substring search, Data compression

Prominent Subfields

NP-theory

Approximation algorithm

Lower bounds

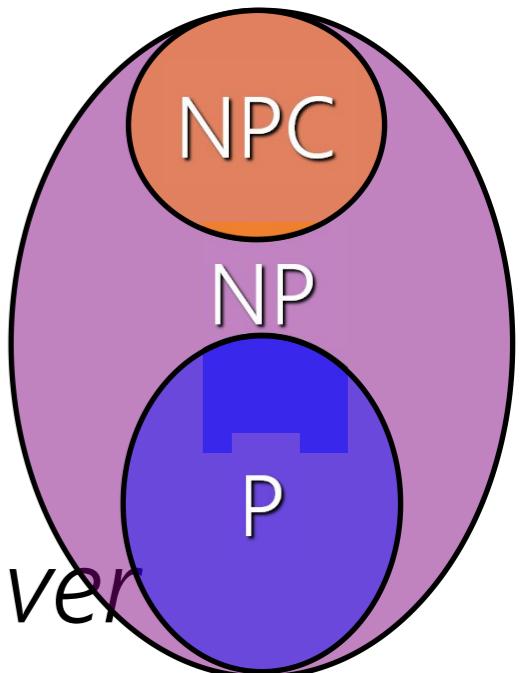
Computational geometry

Substring search

Data compression

NP-theory

- **P**: deterministic TM, Polynomial time
- **NP**: Nondeterministic TM, Polynomial time
- **A \leq_p B**: Use B to solve A, A is less harder than B.
- **NPC**: the hardest problems in NP.
3-SAT, 3-coloring, Hamiltonian, Vertex Cover, TSP.....
- **Test**: Show clique is NPC given vertex cover is NPC.



Backtracking

- **What:** A kind of intelligent exhaustive search
- **When:** partial solution can be quickly checked
- **Examples:** 3-coloring, 8-queen.....

Lower bounds

- **What:** A problem has lower bounds $\Omega(f(n))$ if all algorithms for it are $\Omega(f(n))$.
- Optimal algorithms
- In decision tree model, *Sorting*, *Element uniqueness*, *Closest pair*, *Convex hull*, and *Euclidean MST* have lower bound $\Omega(n \log n)$.

Randomized algorithm

- **What:** algorithms that play dice.
- **Types:** Monte Carlo and Las Vegas
- **Examples:** Randomized Quicksort, String equivalence check, Primality check.....
- **Test:** *Tell the difference between Monte Carlo and Las Vegas algorithm.*

Approximation algorithm

- **What:** a trade-off between precision and time.(polynomial time algorithm)
- **When:** optimization problems
- Approximation ratio, hardness result
- **Examples:** Vertex cover, Knapsack, TSP.....
- **Test:** *Show there is no constant ratio approximation algorithms for TSP unless P=NP.*

Computational geometry

- **What:** algorithms which can be stated in terms of geometry
- Geometric sweeping
- **Examples:** Maximal points, Convex hull, Diameter.....

Substring search

- **What:** Match a pattern in texts.

T (pattern): U N D E R

S (text): D O U U N D E R S T A N D M E ?

- Brute-Force, KMP, AC-automation, Rabin-Karp, suffix tree, ...

Data Compression

- Compression reduces the size of a file:
 - To save space when storing it.
 - To save time when transmitting it.
 - Most files have lots of redundancy.
- Run length, Huffman, LZW, JPEG...

Course contents

- Algorithm design techniques
 - Divide-and-conquer, Greedy, Dynamic Programming
- Analysis of algorithms
 - O , Ω , Θ , Worst-case and Average-case, Recurrences, Amortized analysis
- “Standard” algorithmic problems
 - Sorting, Graph theory, Network flow
- Prominent subfields
 - NP theory, Backtracking, Randomization, Approximation, Lower bound, Computational Geometry, Substring search, Data compression

Course Goals

- To become proficient in the application of fundamental *algorithm design techniques*
- To gain familiarity with the main theoretical tools used in the *analysis of algorithms*
- To study and analyze different algorithms for many of “standard” *algorithmic problems*
- To introduce students to some of the *prominent subfields* of algorithmic study in which they may wish to pursue further study

The End