ECS32B

Introduction to Data Structures

Graphs

Final Review

Lecture 28

Announcements

- Makeup SLAC next Monday at 6:30PM in Hutchison 73 will cover the Makeup assignment.
- TAs will hold their usual office hours in Hutchison 78B through next Tuesday. You may bring any questions.
- An optional review session for the final exam is being scheduled for next week.

Shortest Path Pseudocode

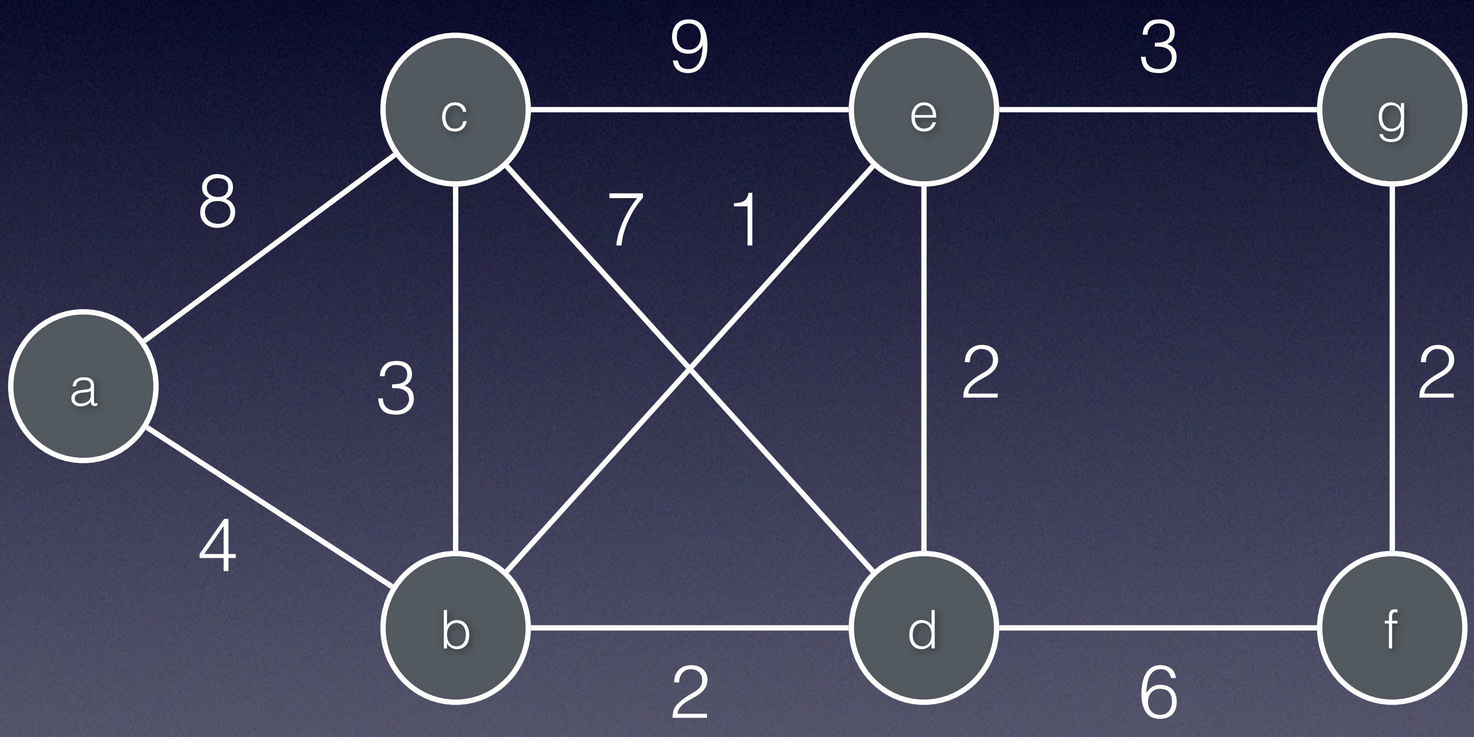
For each vertex v in the graph we keep
d(s,v): the current shortest path length from v to start
predecessor: the previous vertex on the shortest path

To initialize, set **d(s,v)** = ∞ and **predecessor** = ? for all vertices
For a start vertex **s** set **d(s,s)** = 0
Enqueue all vertices onto a priority queue with **d(s,v)** as priority

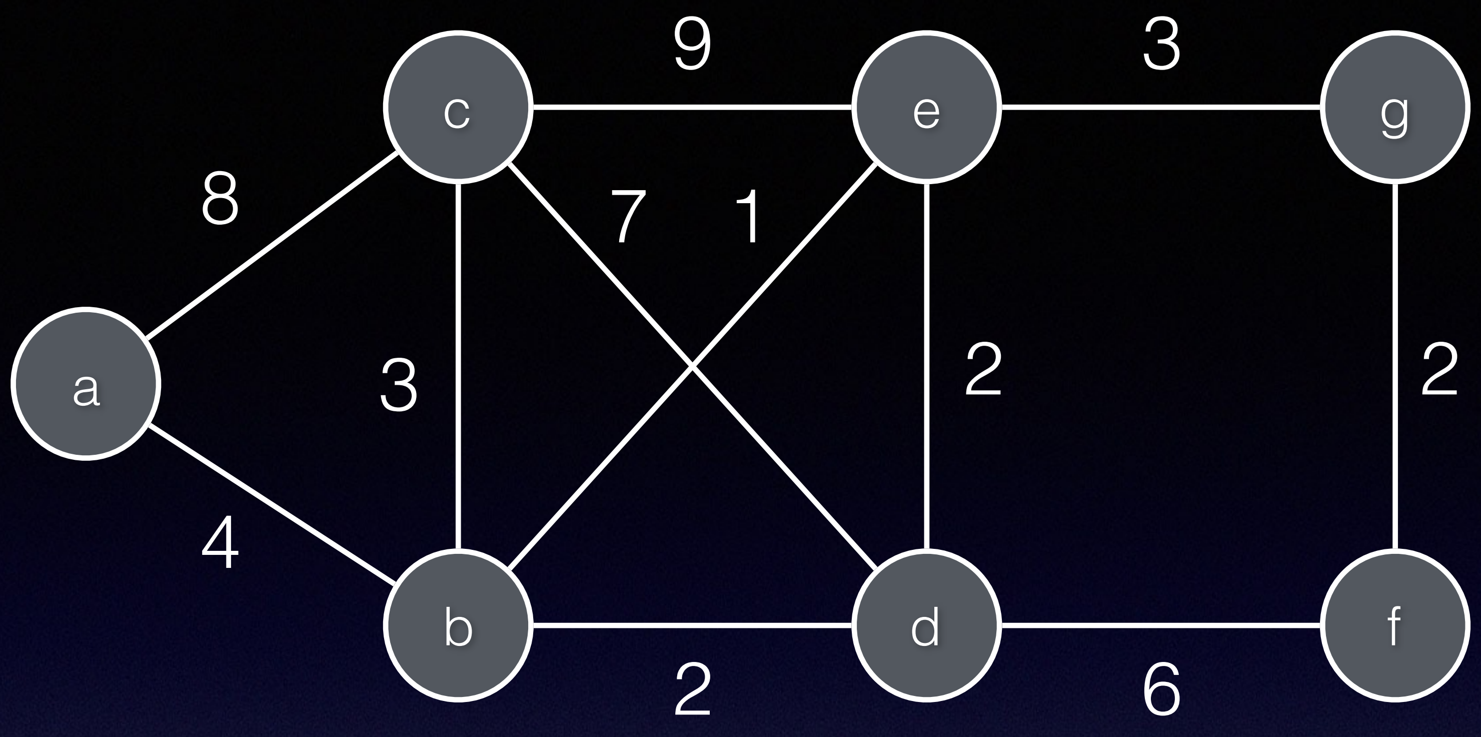
```
While there are queued vertices:
  Dequeue vertex u with smallest d(s,u)
  For each queued vertex v adjacent to u:
      If d(s,u) + weight(u,v) is smaller than d(s,v):
          update d(s,v) = d(s,u) + weight(u,v) and
          set predecessor of v to u
```

This is where we left off last time: We colored **grey** all vertices we dequeued and visited. And it looks like the priority queue is empty, so we're done!

But how do we know what the shortest path from **a** to anywhere is?



vertex:	a	b	c	d	e	f	g
d(a,v):	0	4	7	6	5	10	8
predecessor:	?	a	b	b	b	g	e

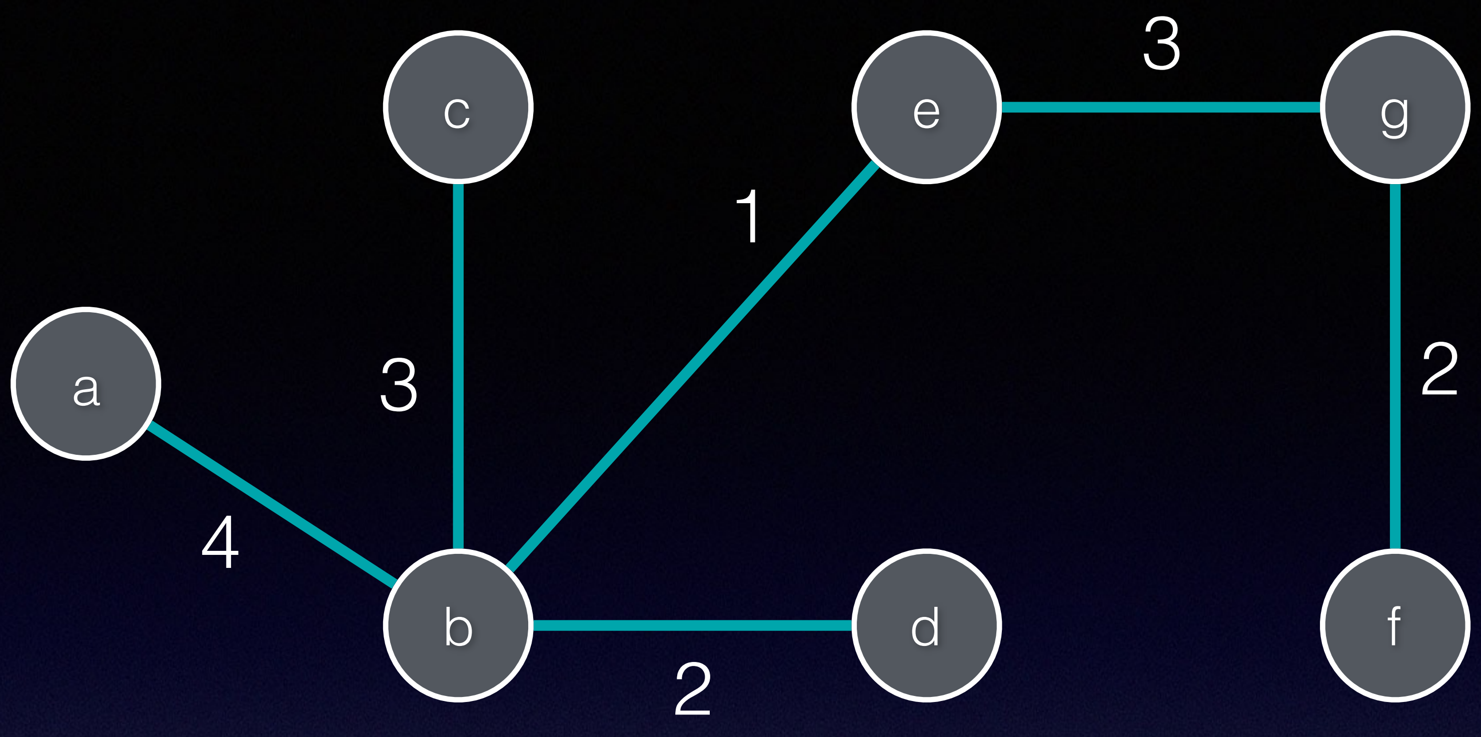


vertex:	a	b	c	d	e	f	g
d(a,v):	0	4	7	6	5	10	8
predecessor:	?	a	b	b	b	g	e

Choose the vertex that you want to go to, then follow the predecessor pointers back to **a**.

Let’s choose **g**, for example.

Shortest path from **g** to **a** **g, e, b, a**



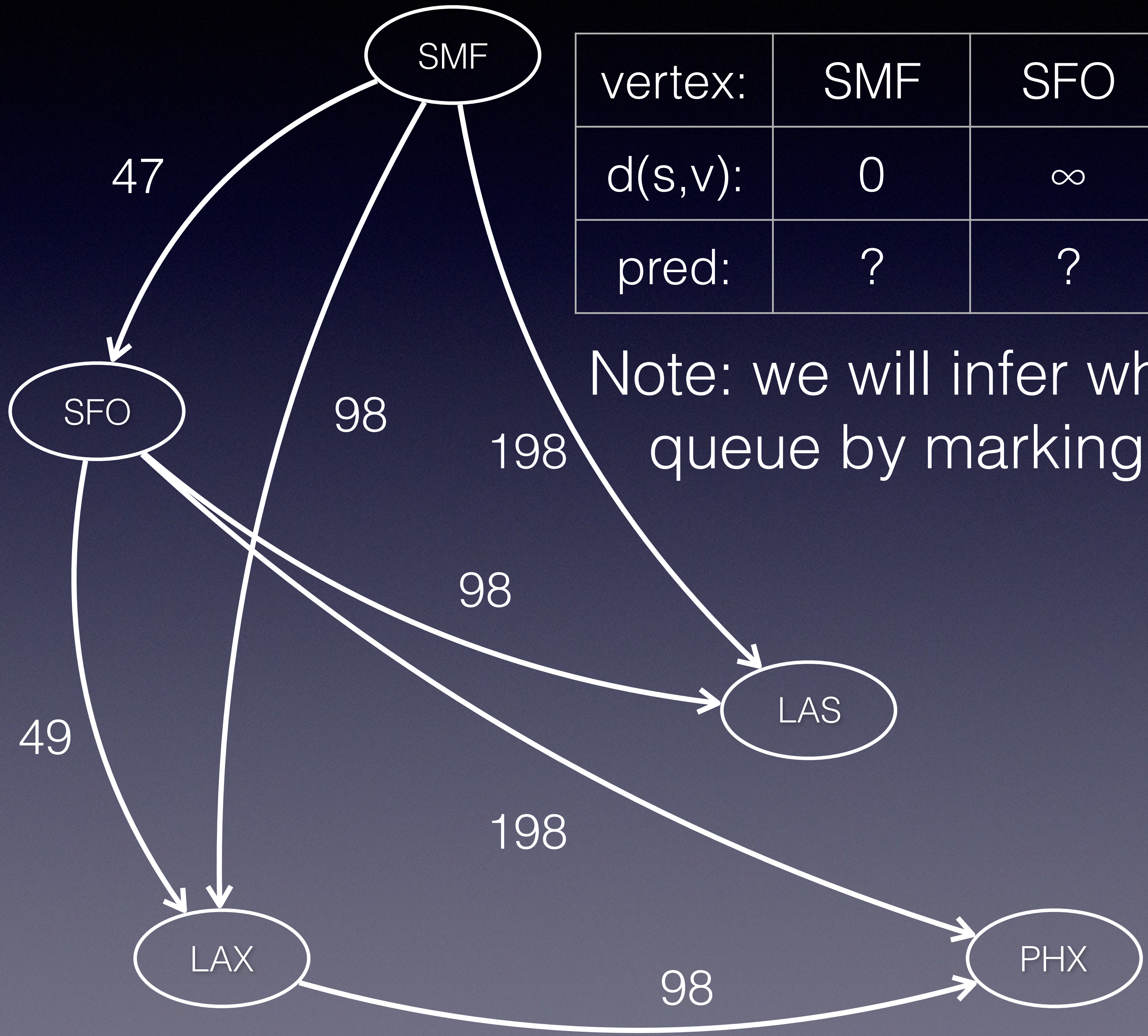
vertex:	a	b	c	d	e	f	g
d(a,v):	0	4	7	6	5	10	8
predecessor:	?	a	b	b	b	g	e

Predecessor always leads you closer to the start.

Notice that the links define a tree with the start as the root.

Similar to the BFS tree but with weights!

Dijkstra's Algorithm Review

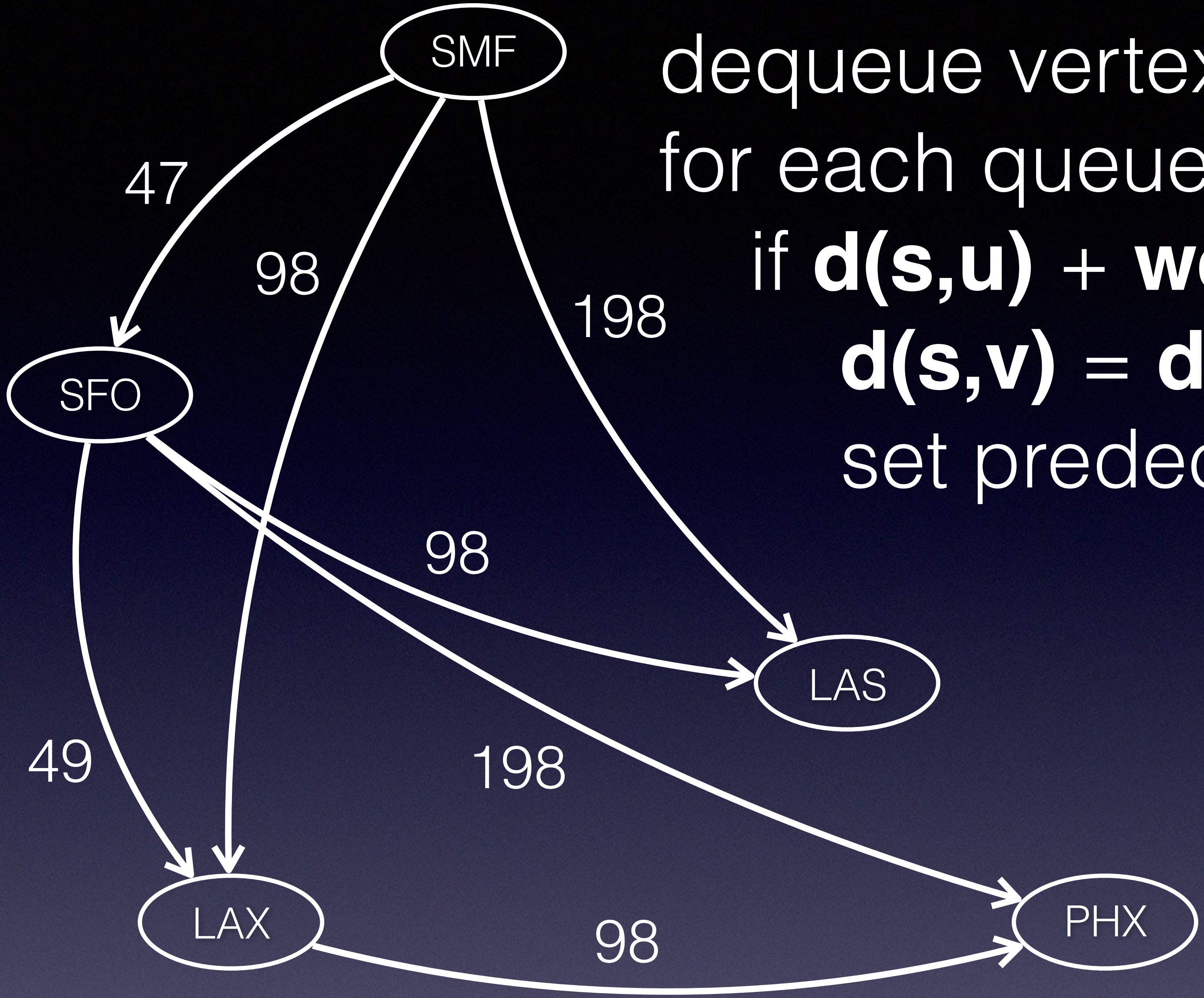


initialized table

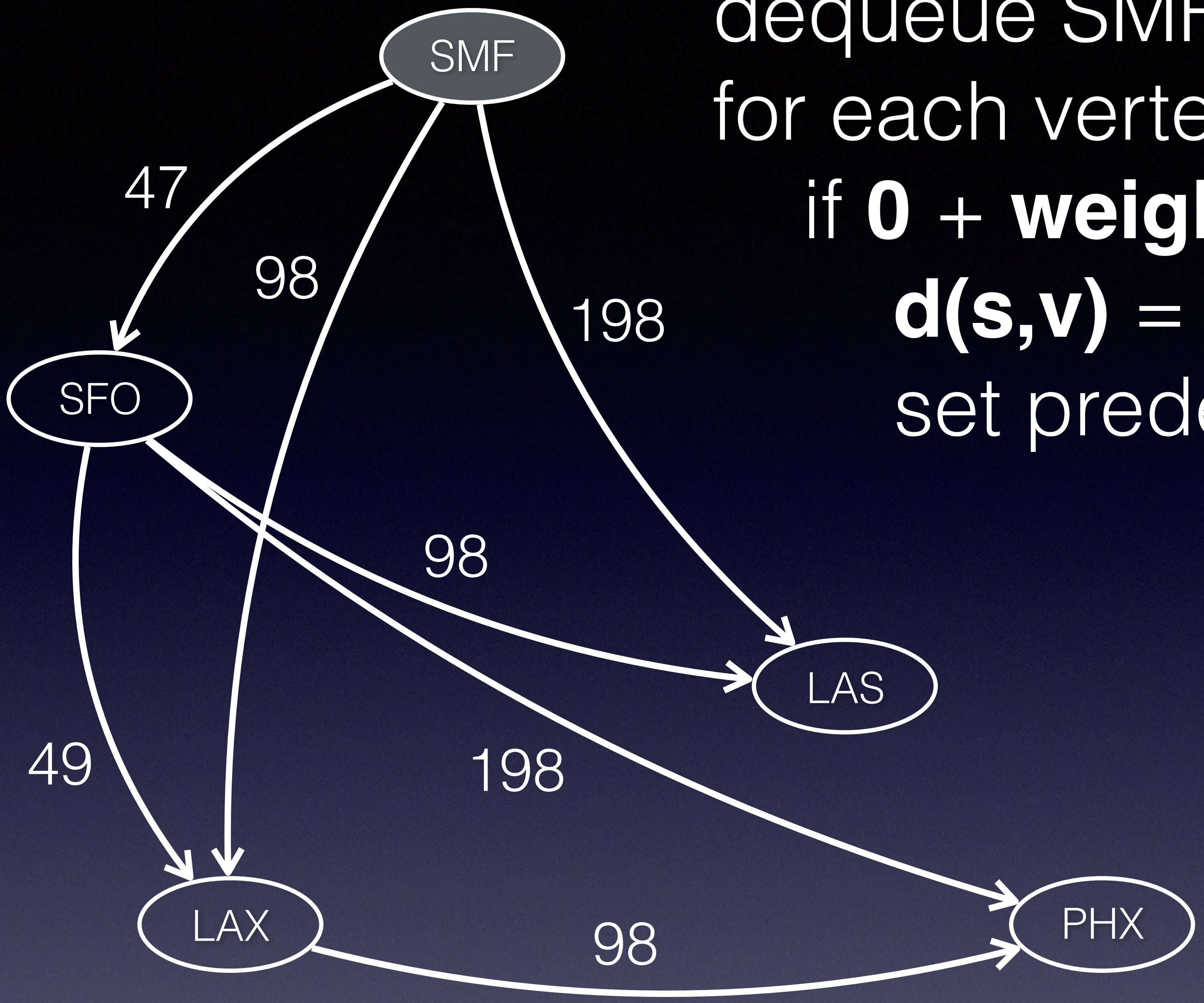
vertex:	SMF	SFO	LAS	LAX	PHX
d(s,v):	0	∞	∞	∞	∞
pred:	?	?	?	?	?

Note: we will infer what's still in the priority queue by marking vertices as visited.

```
During each iteration we will:
dequeue vertex **u** with smallest **d(s,u)**
for each queued vertex **v** adjacent to u:
    if **d(s,u)** + **weight(u,v)** < **d(s,v)** :
        **d(s,v)** = **d(s,u)** + **weight(u,v)**
        set predecessor of **v** to **u**
```

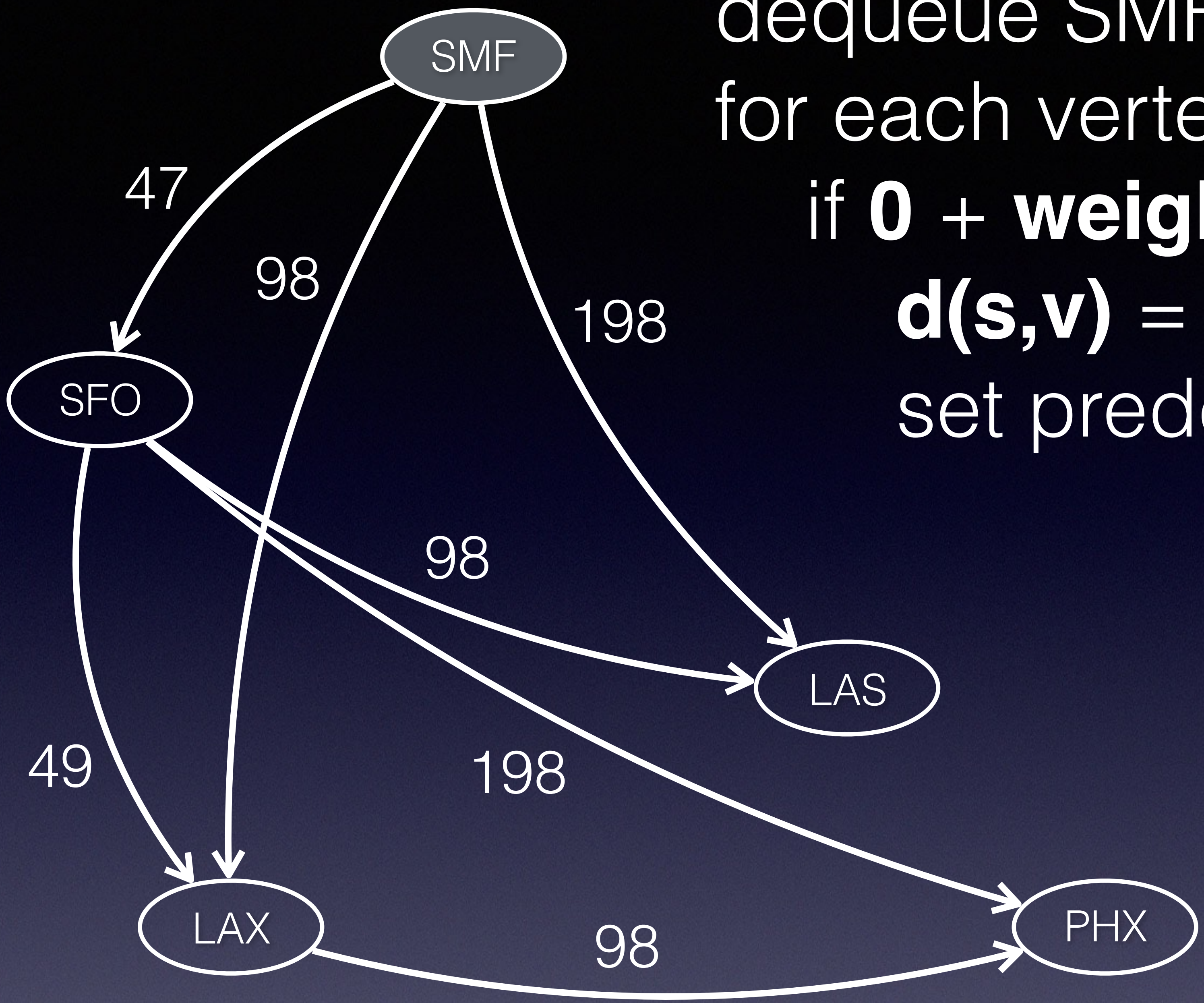


vertex:	SMF	SFO	LAS	LAX	PHX
d(s,v):	0	∞	∞	∞	∞
pred:	?	?	?	?	?



```
dequeue SMF
for each vertex v adjacent to SMF:
    if 0 + weight(SMF,v) < d(s,v):
        d(s,v) = 0 + weight(SMF,v)
        set predecessor of v to SMF
```

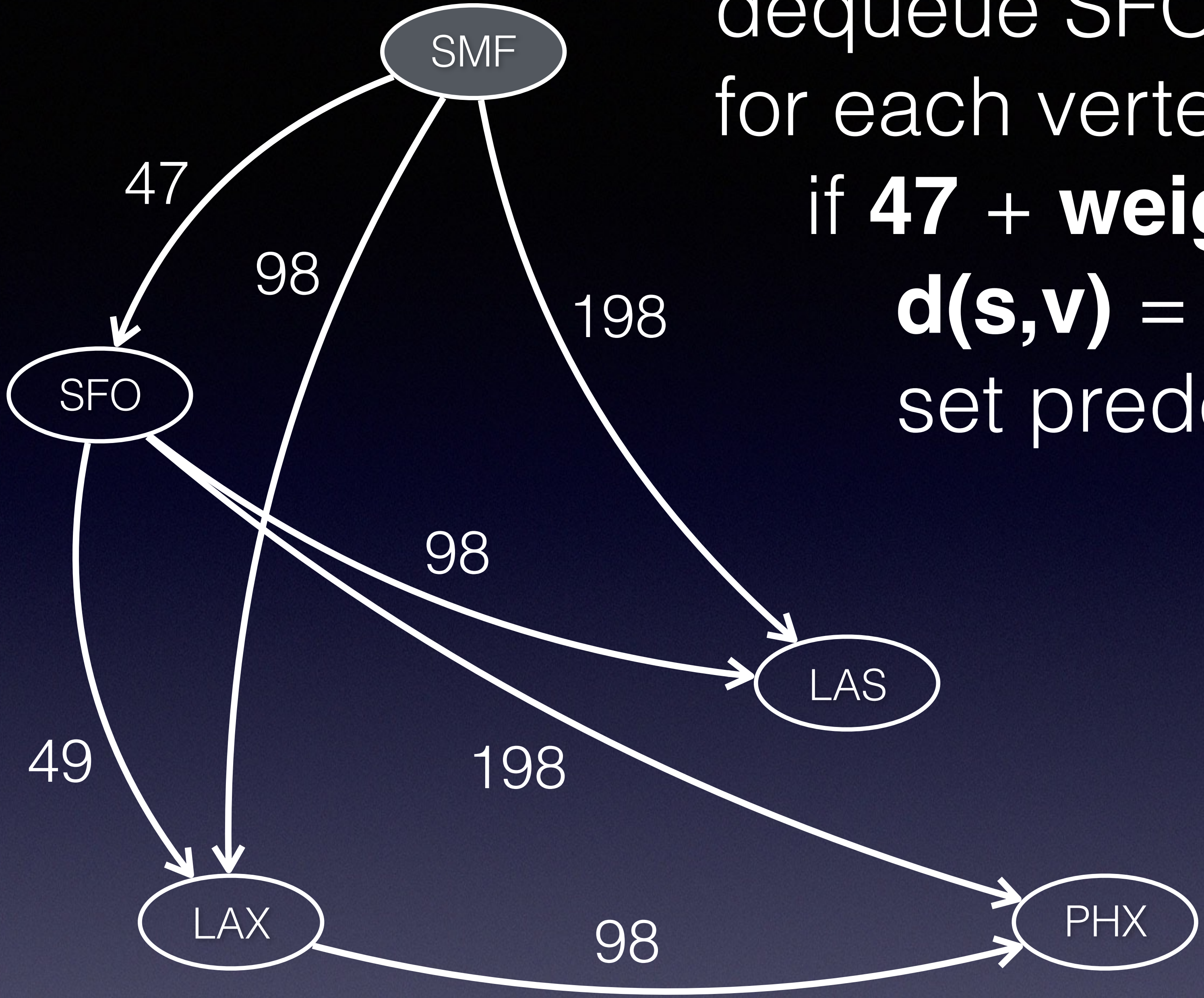
vertex:	SMF	SFO	LAS	LAX	PHX
d(s,v):	0	∞	∞	∞	∞
pred:	?	?	?	?	?



```
dequeue SMF
for each vertex **v** adjacent to SMF:
    if **0** + **weight(**SMF**,v)** < **d(s,v)**:
        **d(s,v)** = **0** + **weight(**SMF**,v)**
        set predecessor of **v** to SMF
```

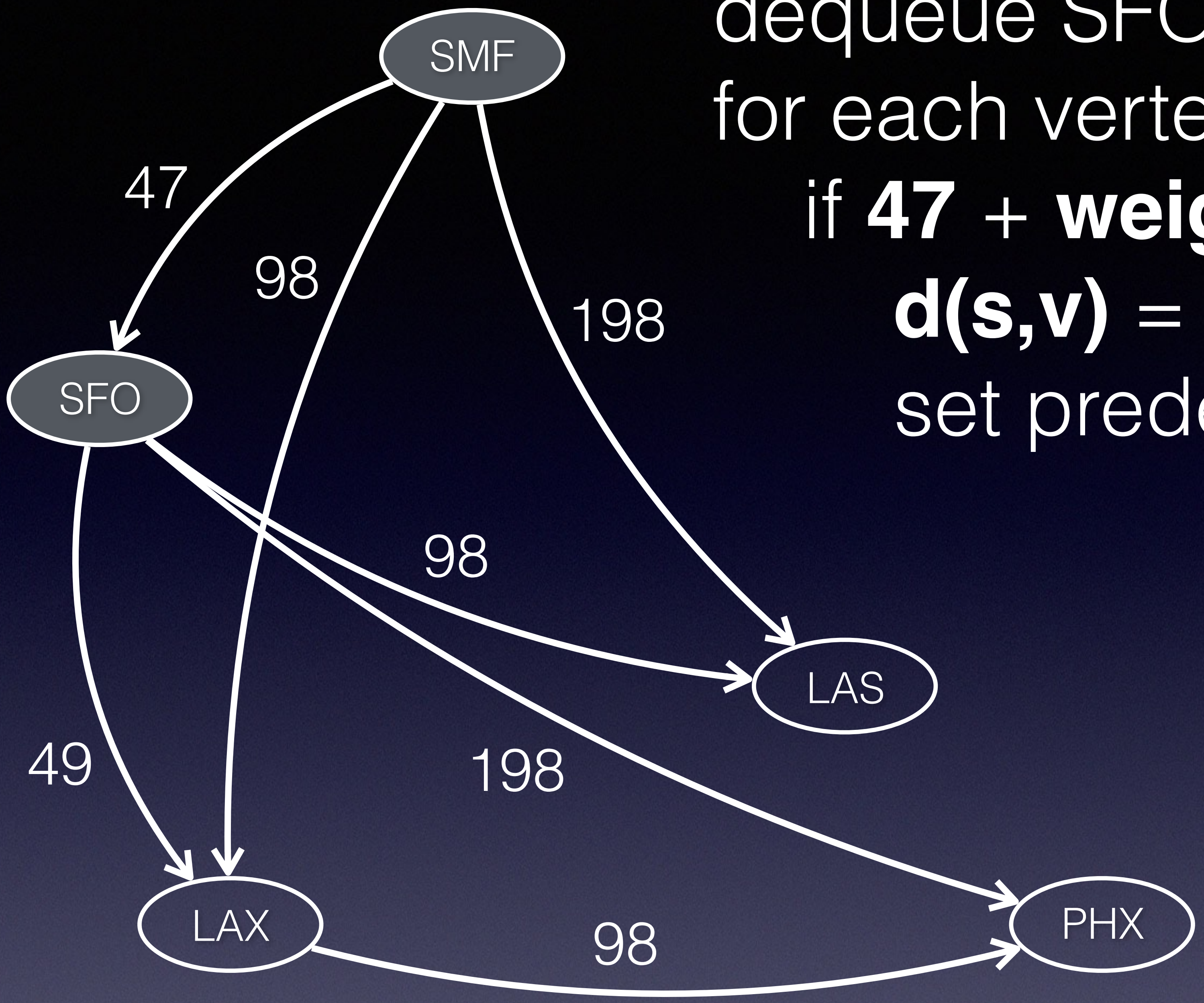
Updated table entries are highlighted.

vertex:	SMF	SFO	LAS	LAX	PHX
d(s,v):	0	47	198	98	∞
pred:	?	SMF	SMF	SMF	?



```
dequeue SFO
for each vertex **v** adjacent to SFO:
    if **47** + **weight(**SFO**,v)** < **d(s,v)**:
        **d(s,v)** = **47** + **weight(**SFO**,v)**
        set predecessor of **v** to SFO
```

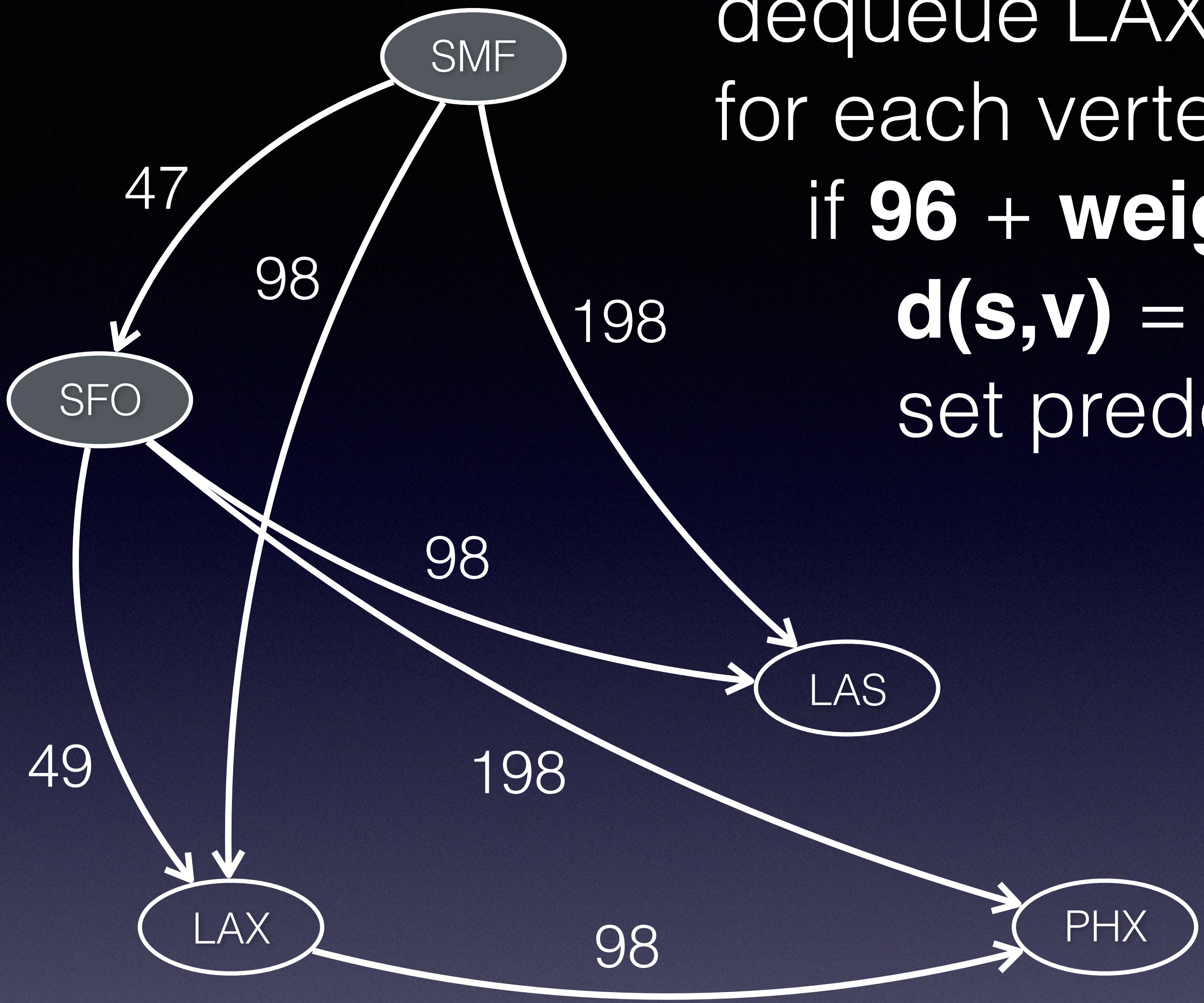
vertex:	SMF	SFO	LAS	LAX	PHX
d(s,v):	0	47	198	98	∞
pred:	?	SMF	SMF	SMF	?



```
dequeue SFO
for each vertex **v** adjacent to SFO:
    if **47** + **weight(**SFO**,v)** < **d(s,v)**:
        **d(s,v)** = **47** + **weight(**SFO**,v)**
        set predecessor of **v** to SFO
```

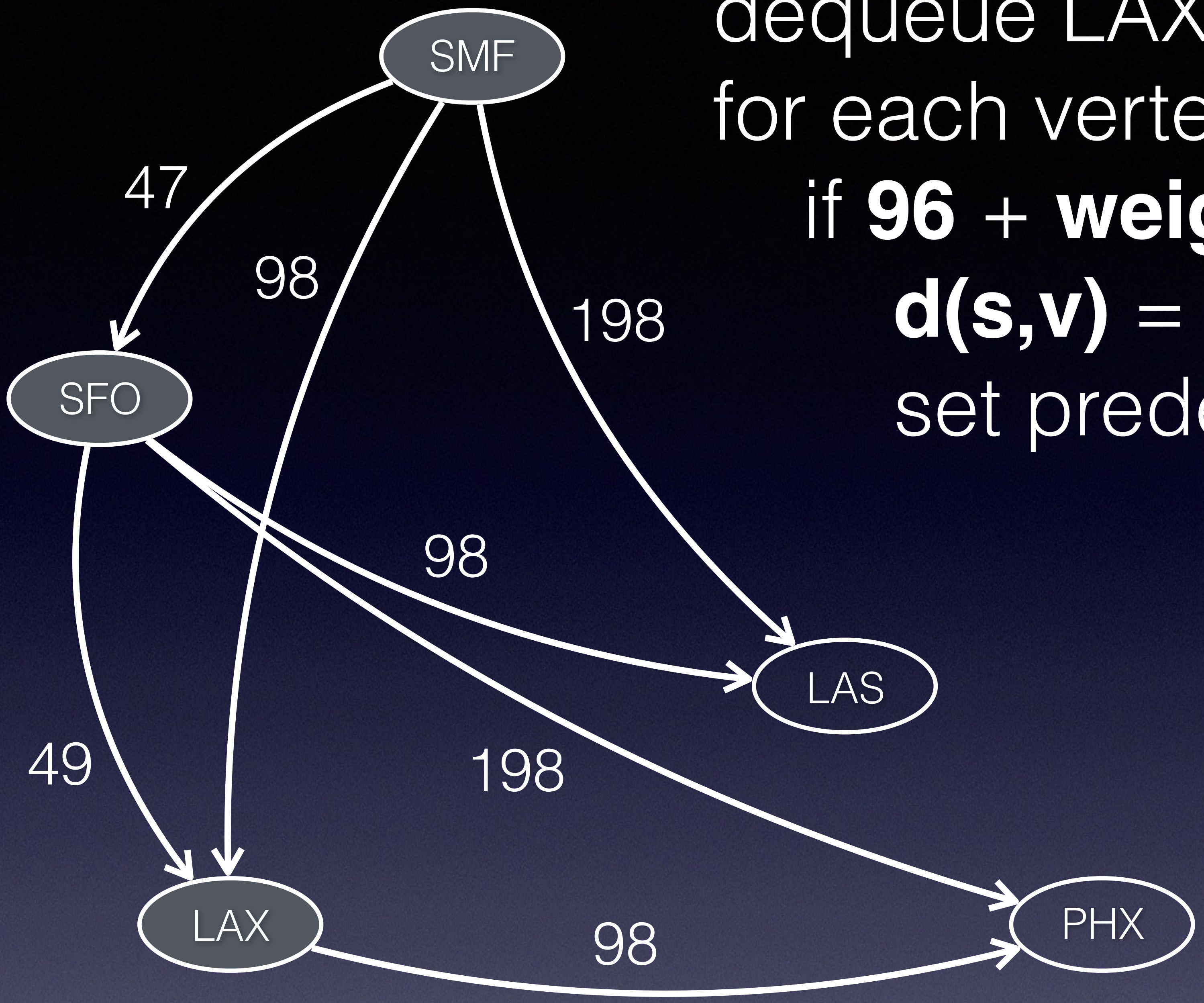
Updated table entries are highlighted.

vertex:	SMF	SFO	LAS	LAX	PHX
d(s,v):	0	47	145	96	245
pred:	?	SMF	SFO	SFO	SFO



```
dequeue LAX
for each vertex **v** adjacent to LAX:
    if **96** + **weight(**LAX,**v)** < **d(s,v)**:
        **d(s,v)** = **96** + **weight(**LAX,**v)**
        set predecessor of **v** to LAX
```

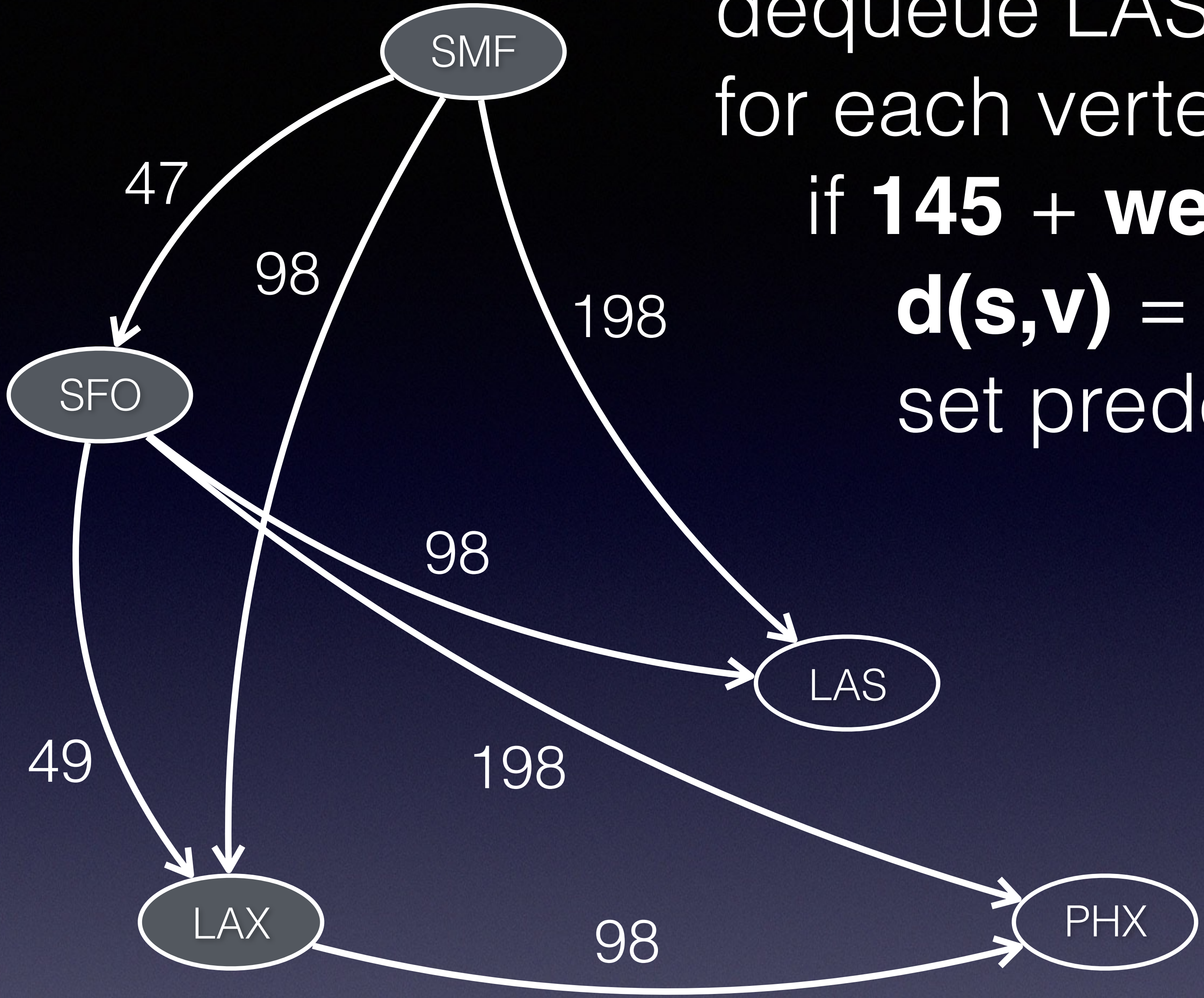
vertex:	SMF	SFO	LAS	LAX	PHX
d(s,v):	0	47	145	96	245
pred:	?	SMF	SFO	SFO	SFO



```
dequeue LAX
for each vertex v adjacent to LAX:
    if 96 + weight(LAX,v) < d(s,v):
        d(s,v) = 96 + weight(LAX,v)
        set predecessor of v to LAX
```

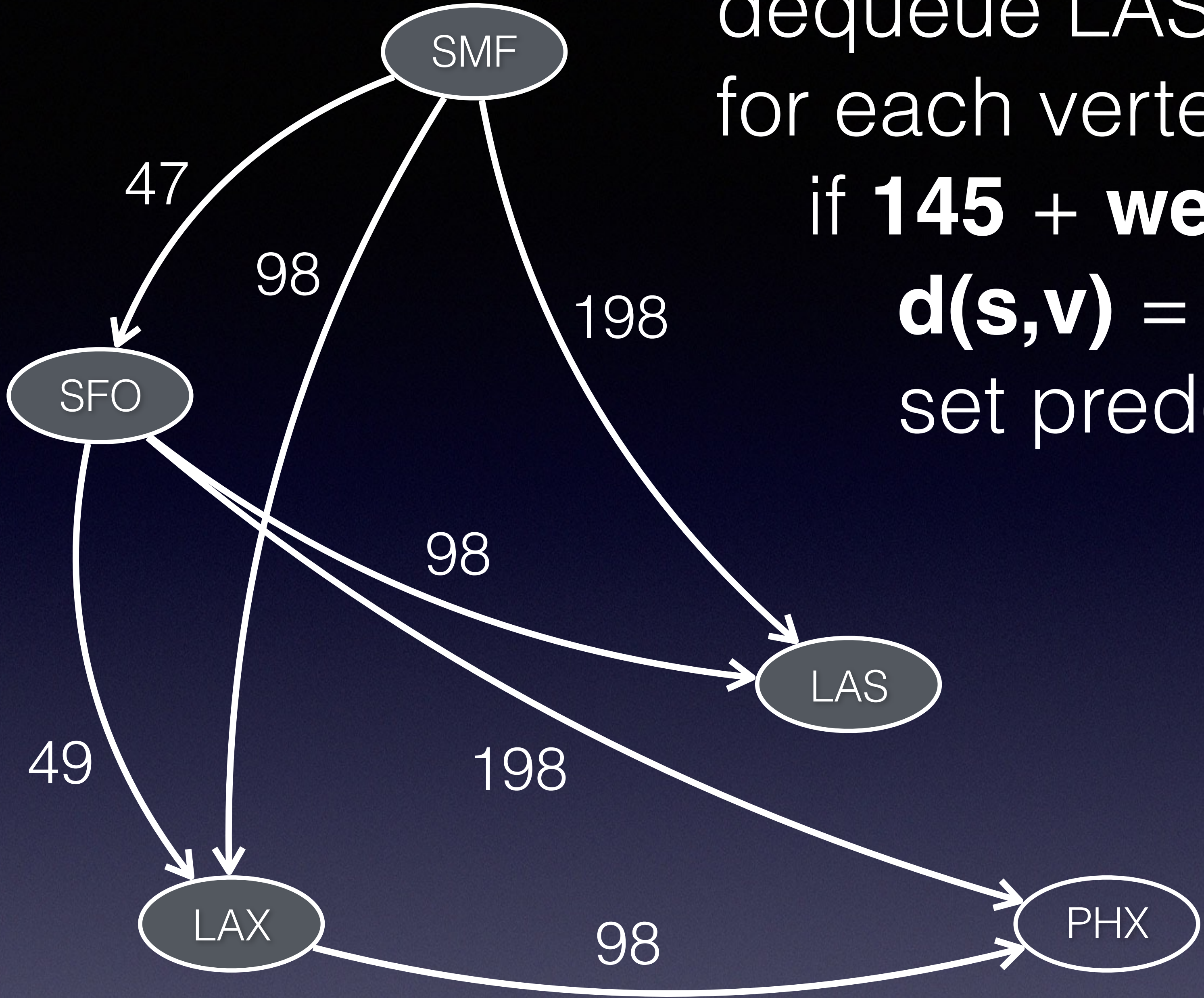
Updated table entries are highlighted.

vertex:	SMF	SFO	LAS	LAX	PHX
d(s,v):	0	47	145	96	194
pred:	?	SMF	SFO	SFO	LAX



```
dequeue LAS
for each vertex **v** adjacent to LAS:
    if **145** + **weight(**LAS**,v)** < **d(s,v)**:
        **d(s,v)** = **145** + **weight(**LAS**,v)**
        set predecessor of **v** to LAS
```

vertex:	SMF	SFO	LAS	LAX	PHX
d(s,v):	0	47	145	96	194
pred:	?	SMF	SFO	SFO	LAX

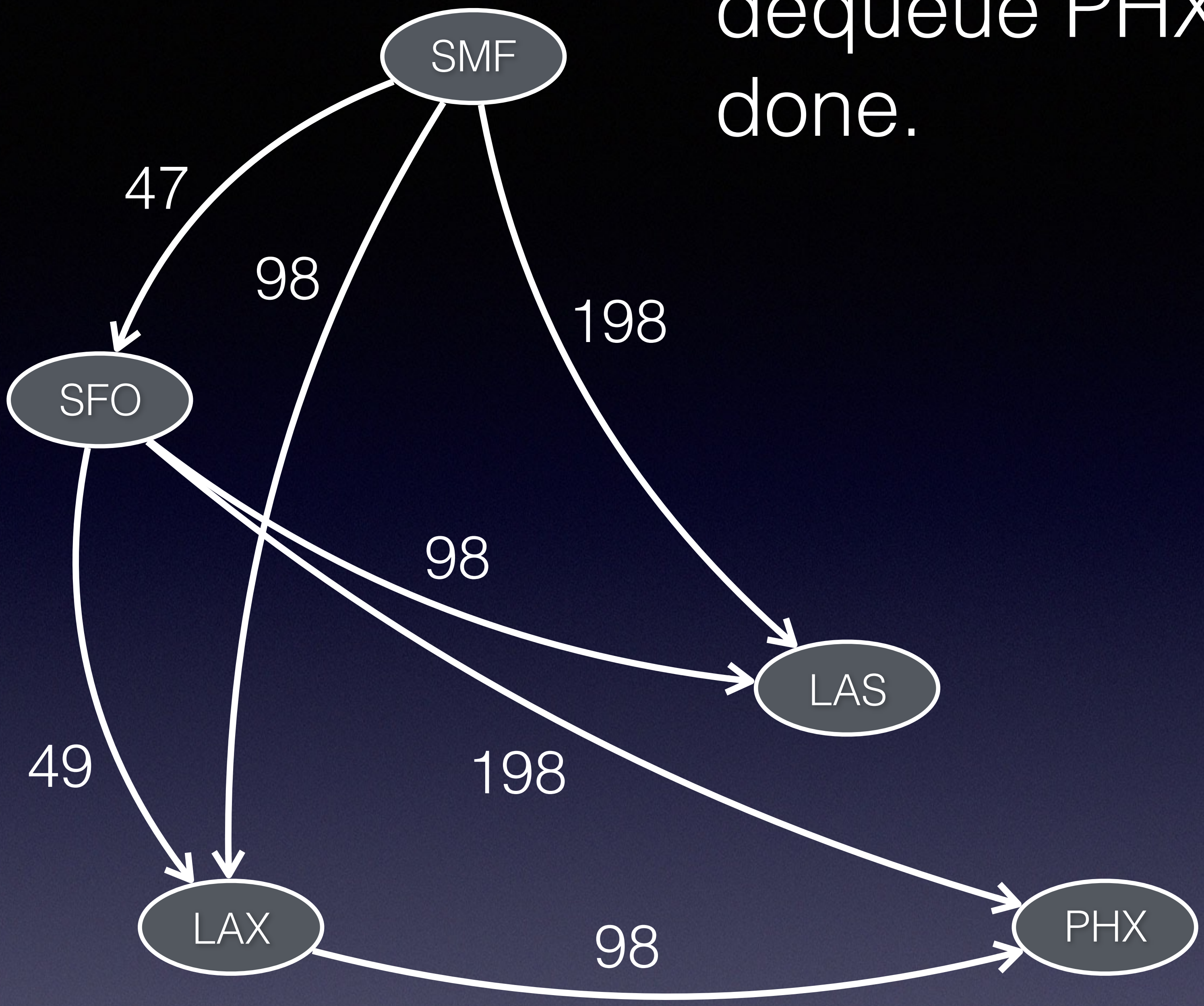


```
dequeue LAS
for each vertex v adjacent to LAS:
    if 145 + weight(LAS,v) < d(s,v):
        d(s,v) = 145 + weight(LAS,v)
        set predecessor of v to LAS
```

No updated table entries.

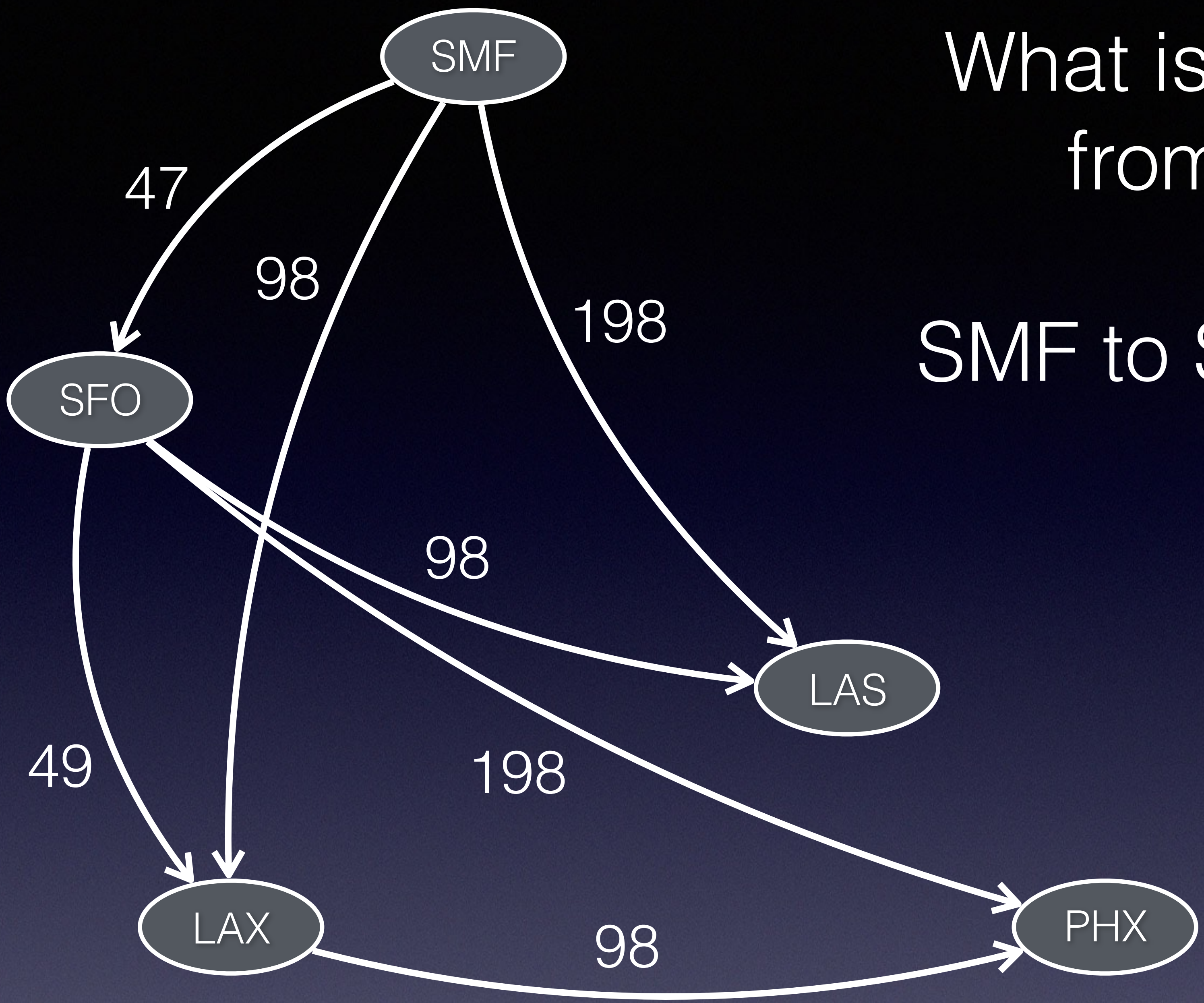
vertex:	SMF	SFO	LAS	LAX	PHX
d(s,v):	0	47	145	96	194
pred:	?	SMF	SFO	SFO	LAX

dequeue PHX
done.



No updated table entries.

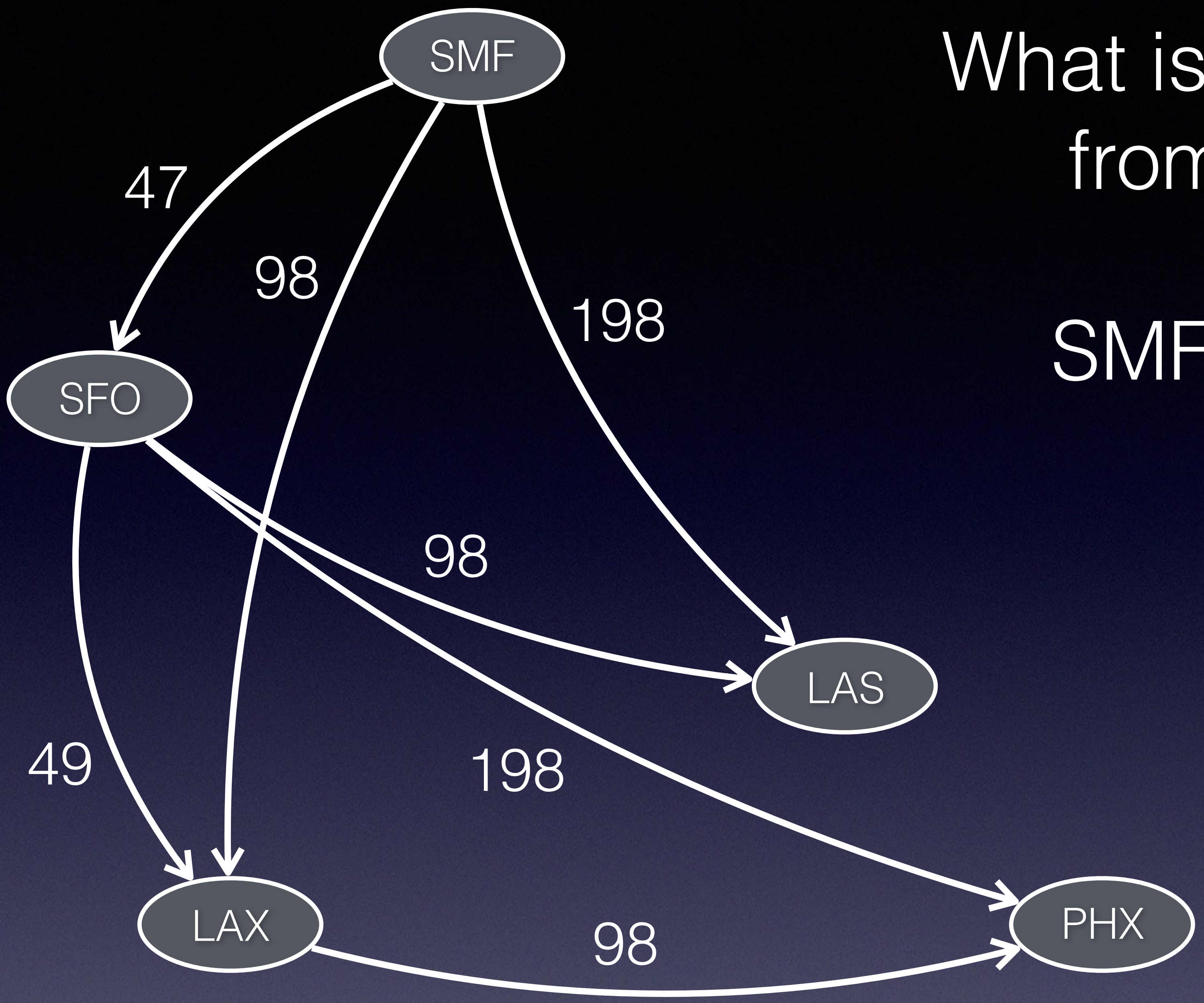
vertex:	SMF	SFO	LAS	LAX	PHX
d(s,v):	0	47	145	96	194
pred:	?	SMF	SFO	SFO	LAX



What is the shortest path from SMF to PHX?

SMF to SFO to LAX to PHX
cost = 194

vertex:	SMF	SFO	LAS	LAX	PHX
d(s,v):	0	47	145	96	194
pred:	?	SMF	SFO	SFO	LAX

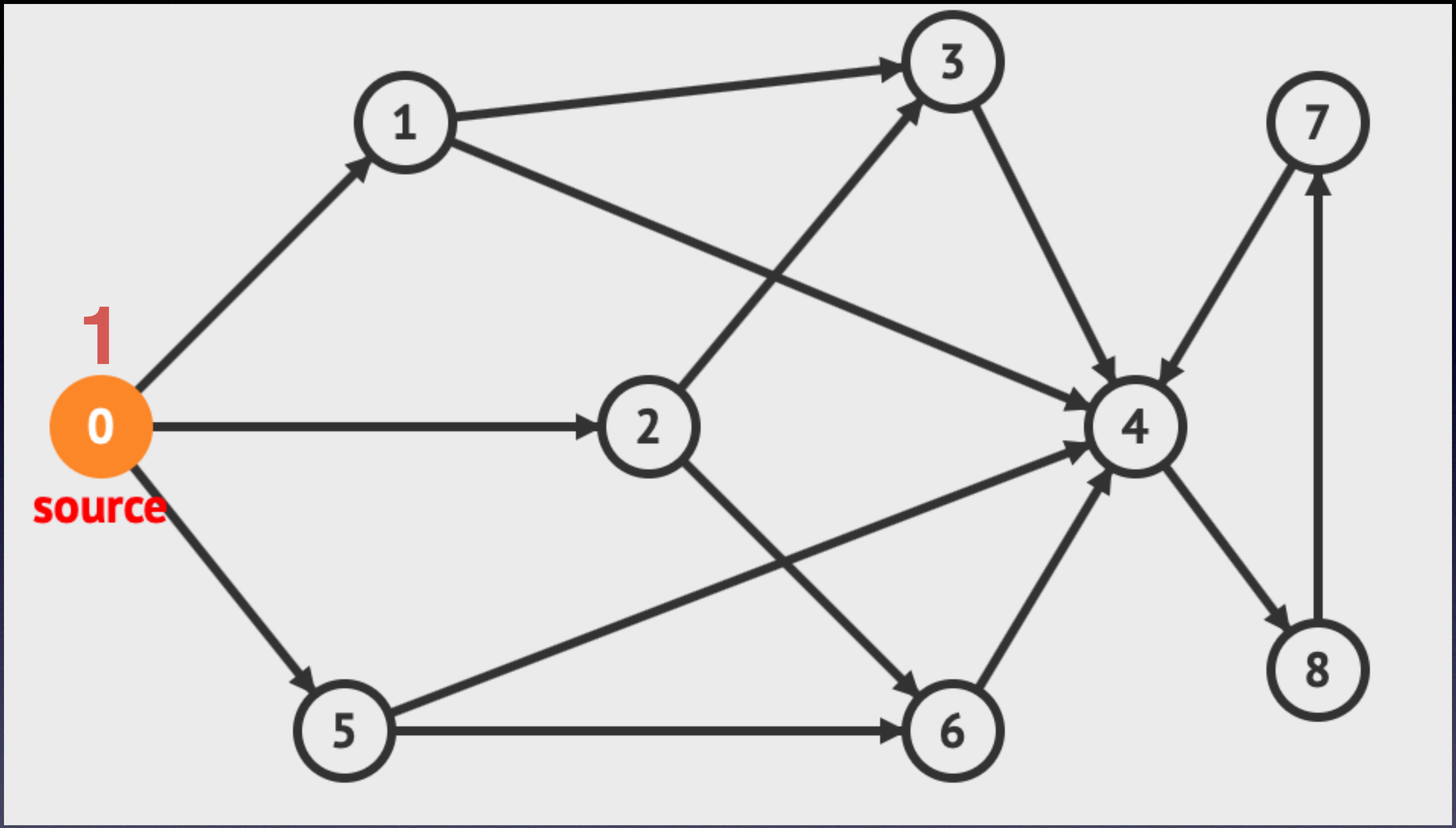


What is the shortest path from SMF to LAS?

SMF to SFO to LAS
cost = 145

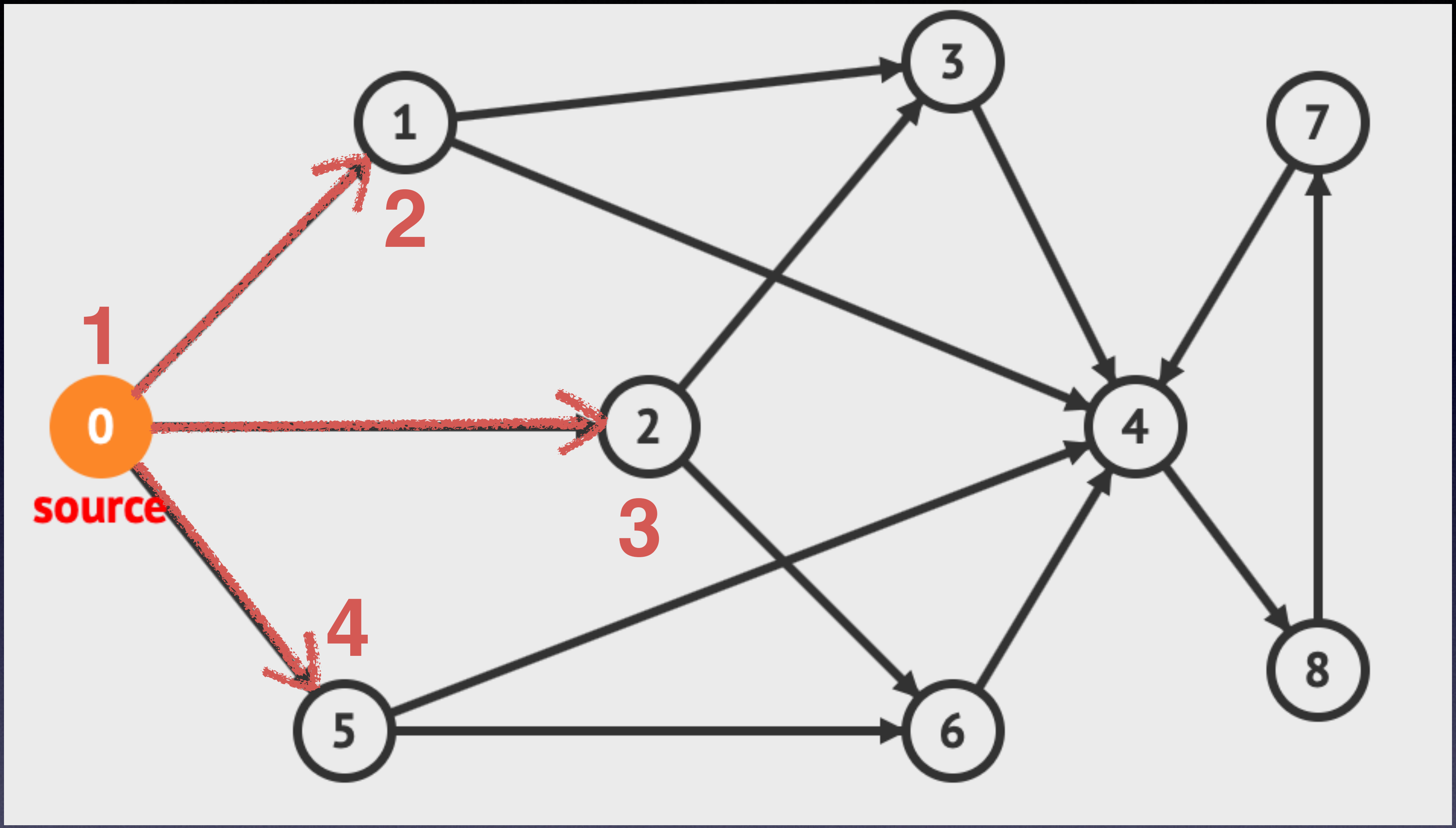
vertex:	SMF	SFO	LAS	LAX	PHX
d(s,v):	0	47	145	96	194
pred:	?	SMF	SFO	SFO	LAX

Breadth First Search Review



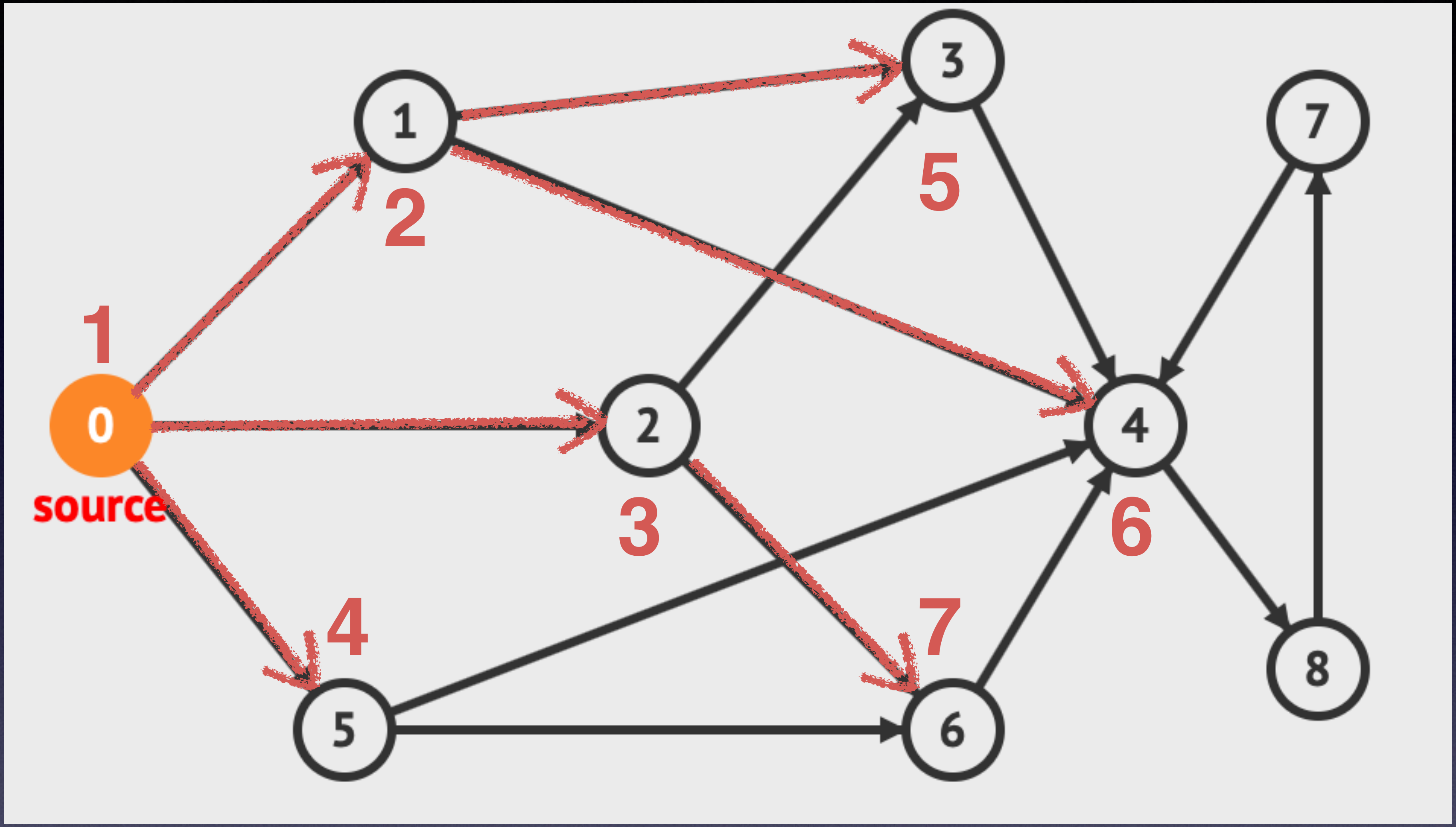
```
number start vertex 1
for each numbered vertex **u**:
    for each unnumbered vertex **v** adjacent to **u** in order:
        give **v** the next number in sequence
        add edge (u,v) to tree (optional)
```

Breadth First Search Review



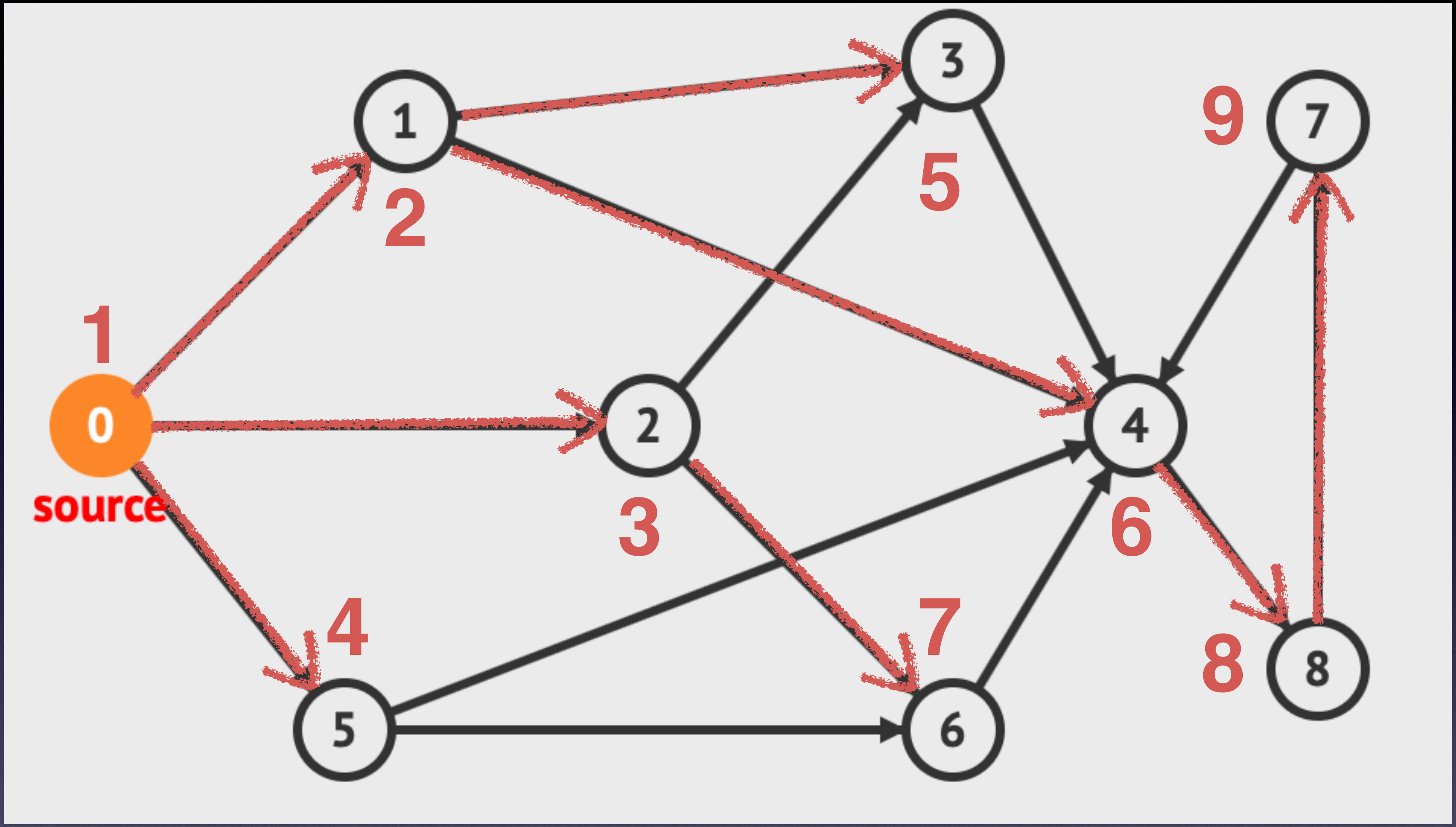
```
number start vertex 1
for each numbered vertex **u**:
    for each unnumbered vertex **v** adjacent to **u** in order:
        give **v** the next number in sequence
        add edge (u,v) to tree (optional)
```

Breadth First Search Review



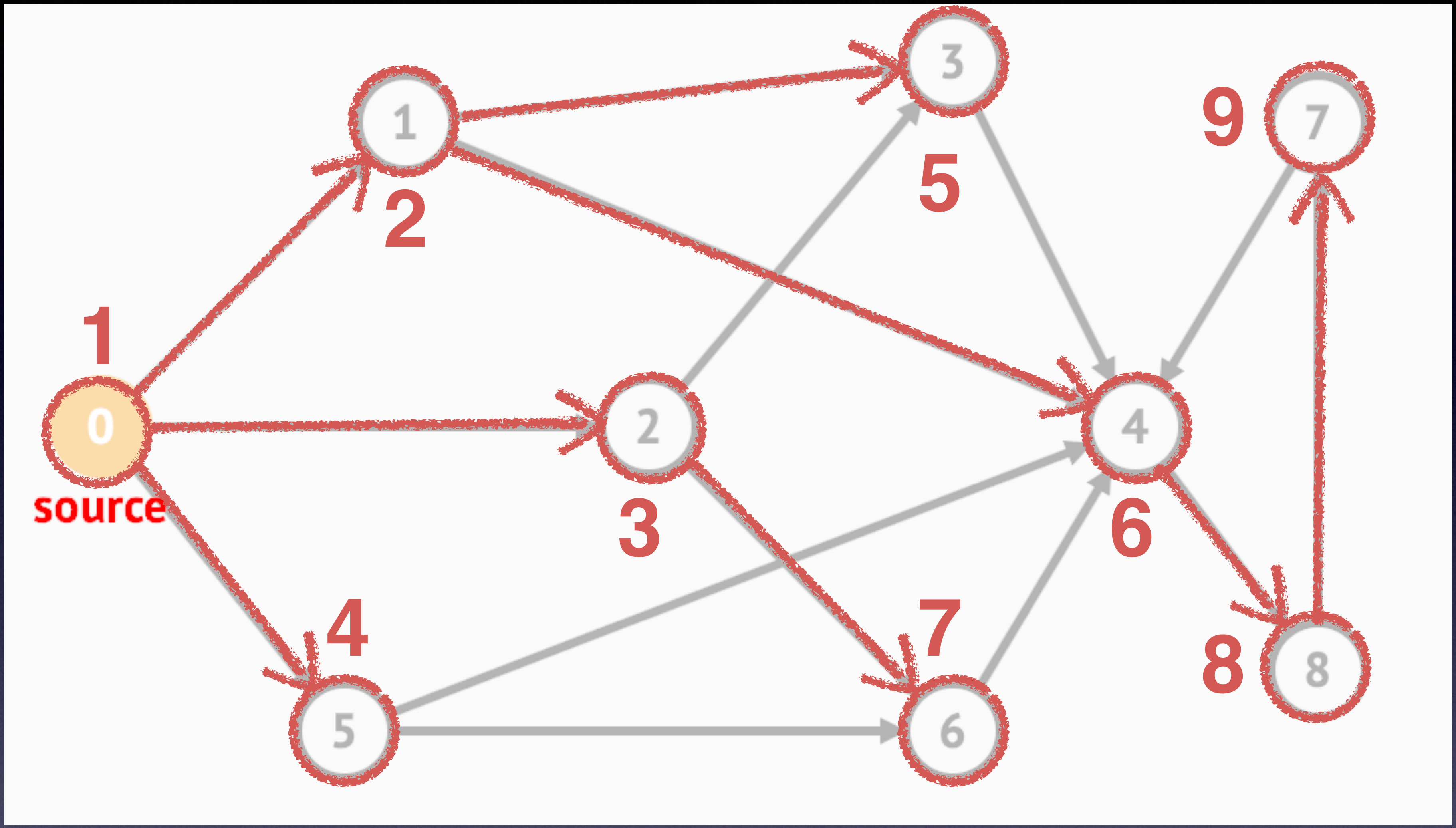
```
number start vertex 1
for each numbered vertex **u**:
    for each unnumbered vertex **v** adjacent to **u** in order:
        give **v** the next number in sequence
        add edge (u,v) to tree (optional)
```

Breadth First Search Review



Here is our vertex visit order: [0,1,2,5,3,4,6,8,7]

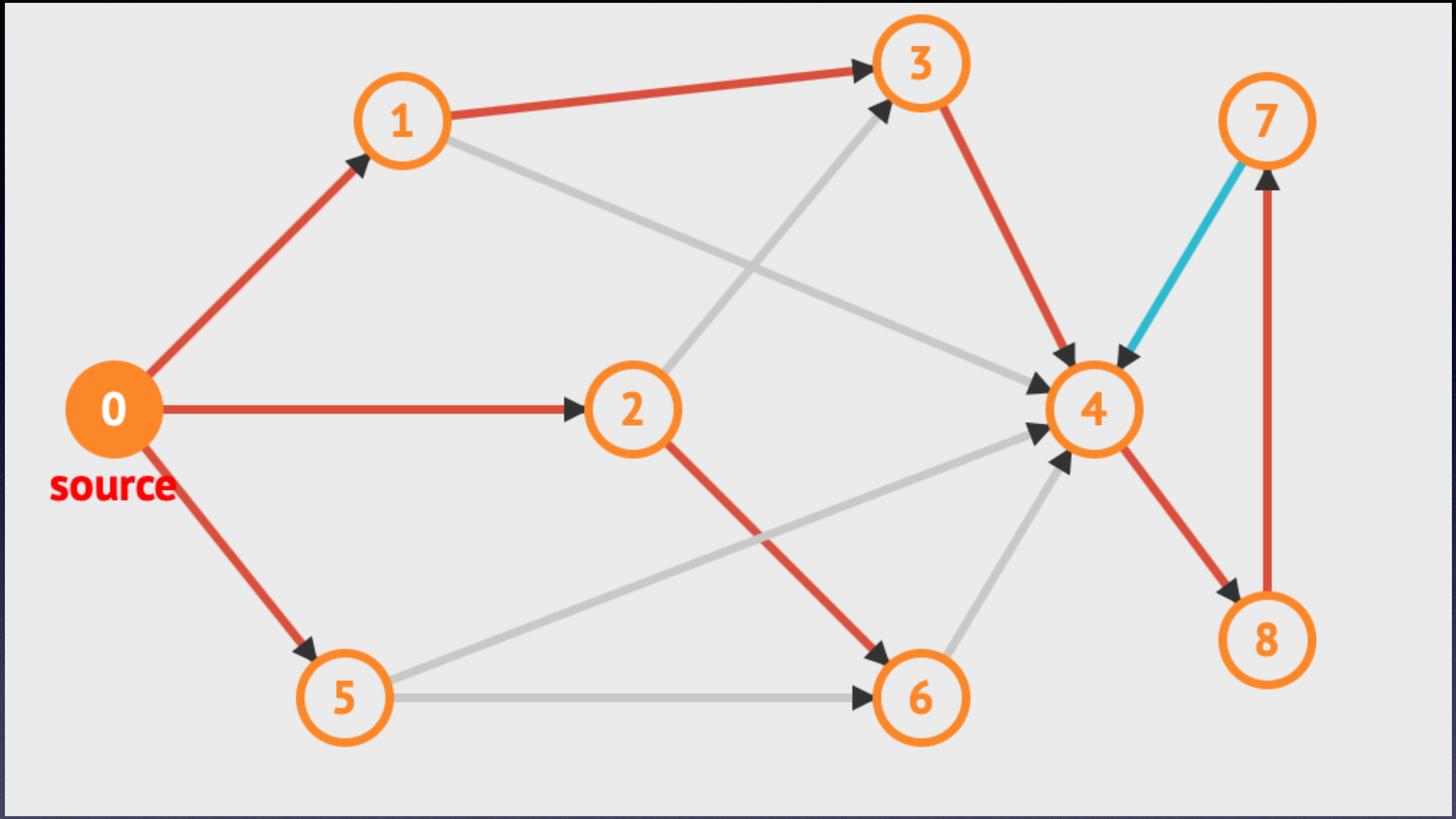
Breadth First Search Review



Here is our BFS tree.

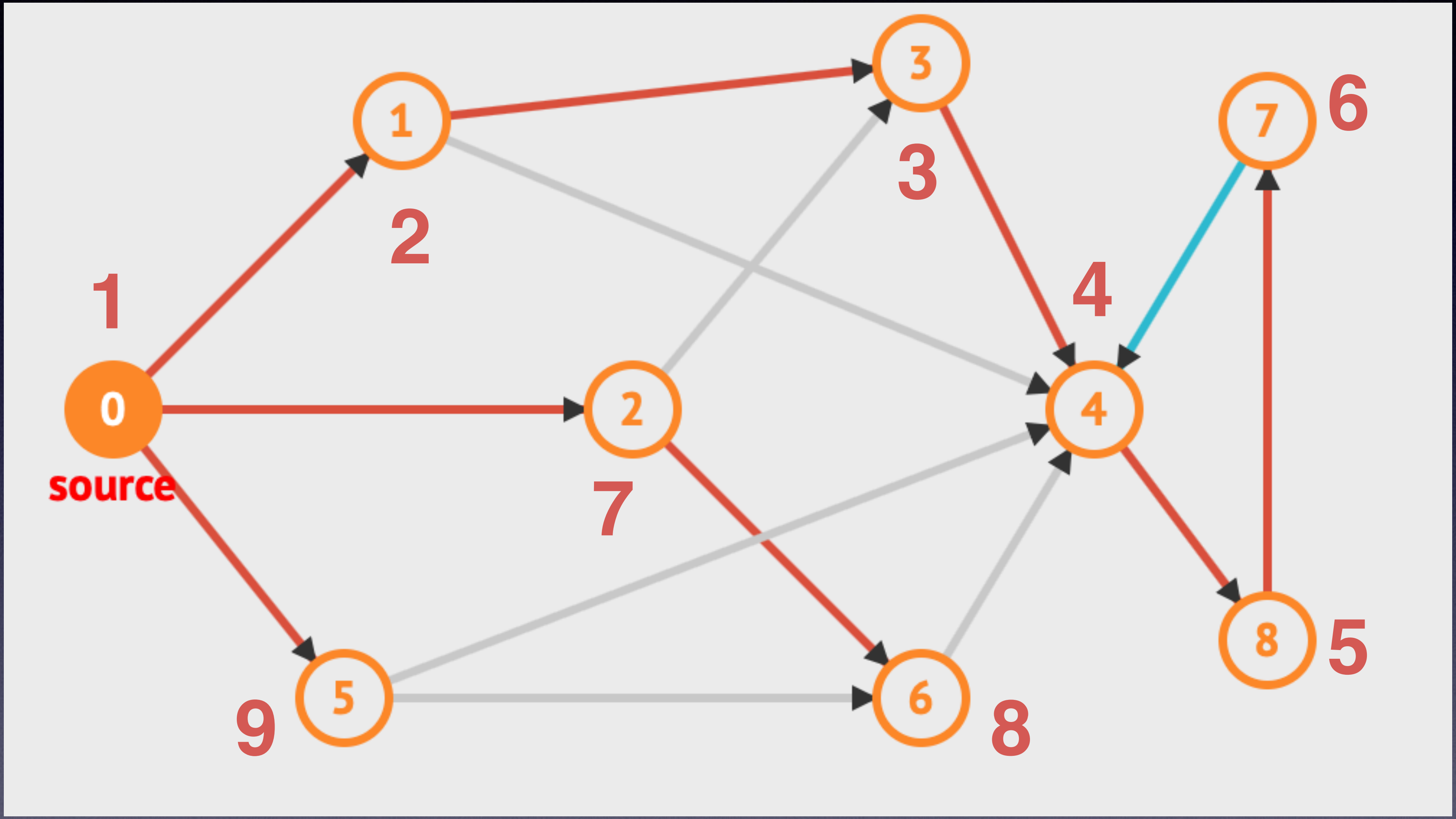
To get a shortest path from the **root** vertex to any other vertex **v**, start at **v** and follow edges to parent until you get to the **root**.

Depth First Search Review



Beginning with a start, keep walking the graph towards unvisited vertices (always preferring the smallest label in a tie) until you can't proceed further. If you can't proceed, backup along your current path (towards the start) until you can. If you backup all the way to the start and can't proceed, you're done unless you need more than one tree to traverse the graph.

Depth First Search Review

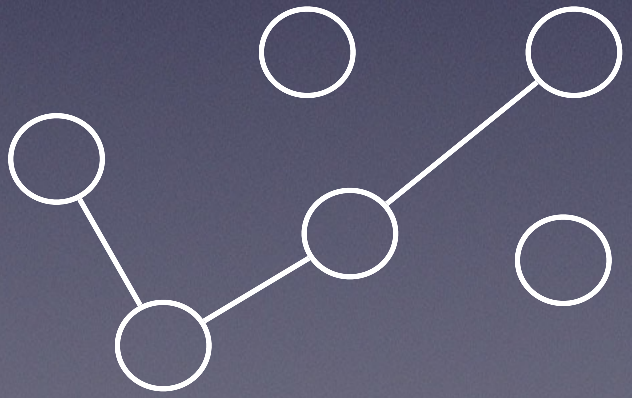


Here is our vertex visit order: [0,1,3,4,8,7,2,6,5]

Analysis of Search

Both depth first search and breadth first search have similar **O(V+E)** time complexities.

The **O(V)** term comes from the fact that both algorithms visit each vertex only once, regardless of how many edges.

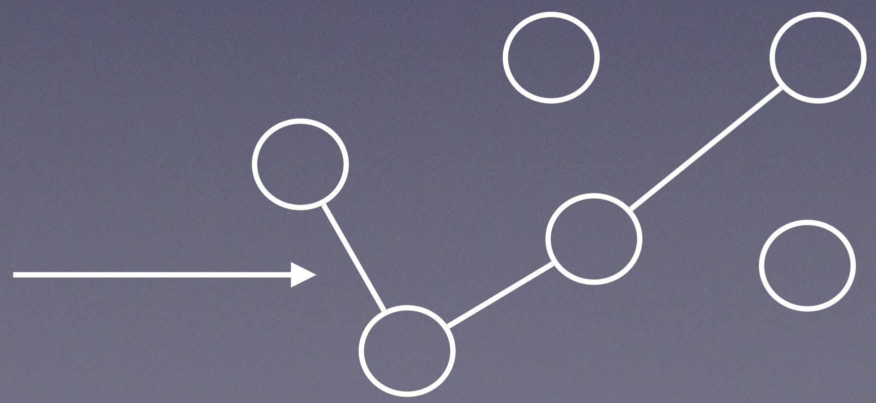


Analysis of Search

Where does **O(E)** come from?

When visiting each vertex, both algorithms have a loop that examines all adjacencies. There are O(V) adjacencies. So we could say that search is just O(V^2). However, when using an adjacency list, an edge is examined at most once from each end. Summing over all vertices, **O(E)** is a tighter bound. And that's also a good example of **amortization**.

An undirected edge (u,v) is examined at u and v



What’s Next?

You’re done with the formal part of the curriculum.

This self contained course had two objectives

- **Improving your language skills**. Become more adept at writing programs and Python language syntax, in particular how to use abstract data types and object oriented programming techniques. Knowing one language’s syntax well makes it much easier to pick up another.

- **Improving your problem solving skills**. Become a better problem solver by learning classic (textbook) techniques of computer science: data structures, algorithms, and algorithm analysis. Learn how to solve specific complex problems.

What’s Next?

Want to hone your software development skills:

ECS34 Software Development in UNIX and C/C++

UNIX Operating system tools (e.g., find, grep, sort, uniq, head) and programming environment. Software engineering: building quality programs using integrated development environments, debugging tools, version control, unit testing. Methods for debugging and verification. Principles of programming in C and advanced object-oriented programming in C++ including inheritance, polymorphism and operator overloading.

Extensive programming.

What’s Next?

Want more of the fundamental mathematics of computer science:

ECS20 Discrete Mathematics for Computer Science

Discrete mathematics of particular utility to computer science. Proofs by induction. Propositional and first-order logic. Sets, functions, and relations. Big-O and related notations. Recursion and solutions of recurrence relations. Combinatorics. Probability on finite probability spaces. Graph theory.

What's Next?

Want to know more about how your computer works:

ECS50 Computer Organization and Machine-Dependent Programming

Fundamental concepts of computer architecture (e.g. RISC vs CISC); Comparative study of different hardware architectures via programming in the machine languages; The use of compilers and operating system software in providing abstractions and machine independence to the programmer. Introduction to I/O devices and programming.

Final Exam

- In this classroom on Thursday 6/13 at 1:00PM
- Our final exam will cover the material presented in lectures 14 to 27 beginning with search.
- The corresponding chapters in Miller and Ranum are 5, 6, and 7.
- You will get <u>four</u> pages of notes (front and back)

Final Exam Study Guide

- Use the practice final exam as a study guide
- Note the topics covered in the lecture slides (since 14) and summarized on the following slides.
- Include summaries of data structures and procedures (i.e. important definitions, pseudocode, examples) in your notes.
- Make sure you would be able to do problems like the ones on the sample final exam with the aid of your notes.
- Finally, don't panic during the practice final or the actual final exam. These exams are hard, but curved.

Chapter 5 Search

- Sequential search
- Binary search
  - There is an iterative and recursive algorithm
  - They have different midpoint calculations
    - iterative: midpoint = len(aList) // 2
    - recursive: midpoint = (first + last) // 2

Chapter 5 Search

- Hashing
  - How to calculate the basic hash function %
  - How linear probing (with and without skip) works for collision resolution.
  - How to insert and search for keys.
  - What trick makes hashing an O(1) search?

Chapter 5 Sorting

- Know these sorting algorithms covered in lecture and homework:
    - Insertion sort
    - Selection sort
    - Bubble sort
    - Merge sort
    - Quick sort

Chapter 6 Trees

- Binary search trees
  - How insert and delete work.
- AVL trees
  - Calculating balance.
  - Basic left and right rotation.
  - What happens when a simple rotation doesn’t work?

Chapter 6 Trees

- Expression Trees
  - How construct an expression tree
- Tree Traversal
  - inorder traversal
  - preorder traversal.
  - postorder traversal

Chapter 6 Trees

- Heaps
  - The heap order property and how to restore it using percUp and percDown.
  - How insert and delete root work.
  - Converting from list to tree representation and back again.
  - Using a heap to sort a list.
  - Priority queues

Chapter 7 Graphs

- Graph definitions and representations
- Graph search (traversal)
    - Breadth first search
    - Depth first search
    - For both know the order vertices are visited and how trees are defined.
    - Know how recursive DFS is employed for topological sorting.
- Shortest path

Thanks!