

Data Structures and Algorithms

Prepared by Prof. Shereen Taie

IT251
Information Technology Department

Lecture 10



Lecture 10

Information Technology Department

Prepared by
Prof. Shereen Taie

www.midocean.ae

Learning Objectives:

- Single-Source Shortest Path
- Dijkstra's algorithm Applications
- Problem Dijkstra's algorithm
 - Idea
 - Dijkstra by example
 - Pseudocode
 - Dijkstra's algorithm - steps
- Dijkstra Animated Example
- Dijkstra's algorithm running time

Dijkstra's Algorithm

Chapter

04

Dijkstra's algorithm Applications:

1. Digital Mapping Services in Google Maps: Many times we have tried to find the distance in G-Maps, from one city to another, or from your location to the nearest desired location.

- There encounters the Shortest Path Algorithm, as there are various routes/paths connecting them but it has to show the minimum distance, so **Dijkstra's Algorithm is used to find the minimum distance between two locations along the path.**

2. Social Networking Applications: In many applications you might have seen the app suggests the list of friends that a particular user may know. How do you think many social media companies implement this feature efficiently, especially when the system has over a billion users.

- The standard **Dijkstra algorithm can be applied using the shortest path between users measured through handshakes or connections among them.**

Dijkstra's algorithm Applications:

3. Telephone Network: As we know, in a telephone network, each line has a bandwidth, 'b'. Bandwidth represents the amount of information that can be transmitted by the line. If we imagine a city to be a graph, the vertices represent the switching stations, and the edges represent the transmission lines and the weight of the edges represents 'b'. It **fall into the category of shortest distance problem, for which the Dijkstra is can be used.**

4. IP routing to find Open shortest Path First: Open Shortest Path First (OSPF) is a link-state routing protocol that is used to find the best path between the source and the destination router using its own Shortest Path First.

- Dijkstra's algorithm provides the shortest cost path from the source router to other routers in the network.

Dijkstra's algorithm Applications:

5. Flying Agenda: The agent has access to a database with all airports and flights. Specifically, the agent wants to determine the earliest arrival time for the destination given an origin airport and start time.

- Dijkstra's algorithm comes into use.

6. Designate file server: To designate a file server in a LAN(local area network), Dijkstra's algorithm can be used.

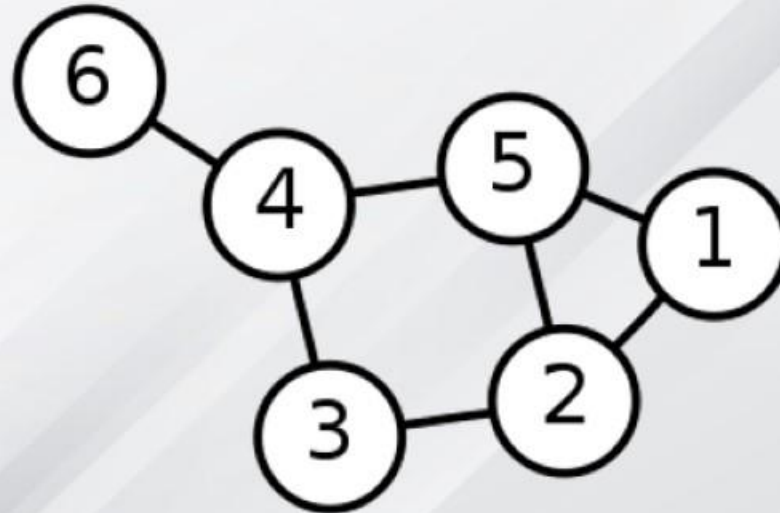
- To minimize the number of "hops" from the file server to every other computer on the network the idea is to use Dijkstra's algorithm to minimize the shortest path between the networks resulting in the minimum number of hops.

DIJKSTRA'S ALGORITHM

Single-source shortest paths for weighted graphs
(non-negative edges)

Single-Source Shortest Path Problem

Single-Source Shortest Path Problem - The problem of finding shortest paths from a source vertex v to all other vertices in the graph.



Exercise 1:

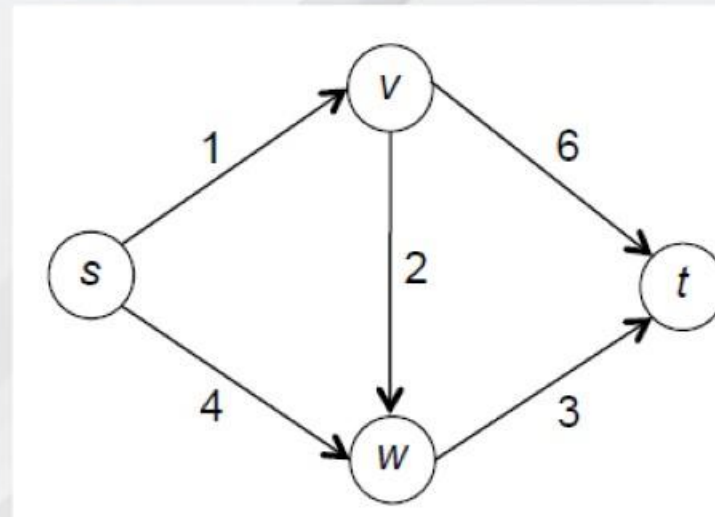
- Consider the following input to the single-source shortest path problem, with starting vertex s and with each edge labeled with its length:
- What are the shortest-path distances to s , v , w , and t , respectively?

a) 0, 1, 2, 3

b) 0, 1, 3, 6

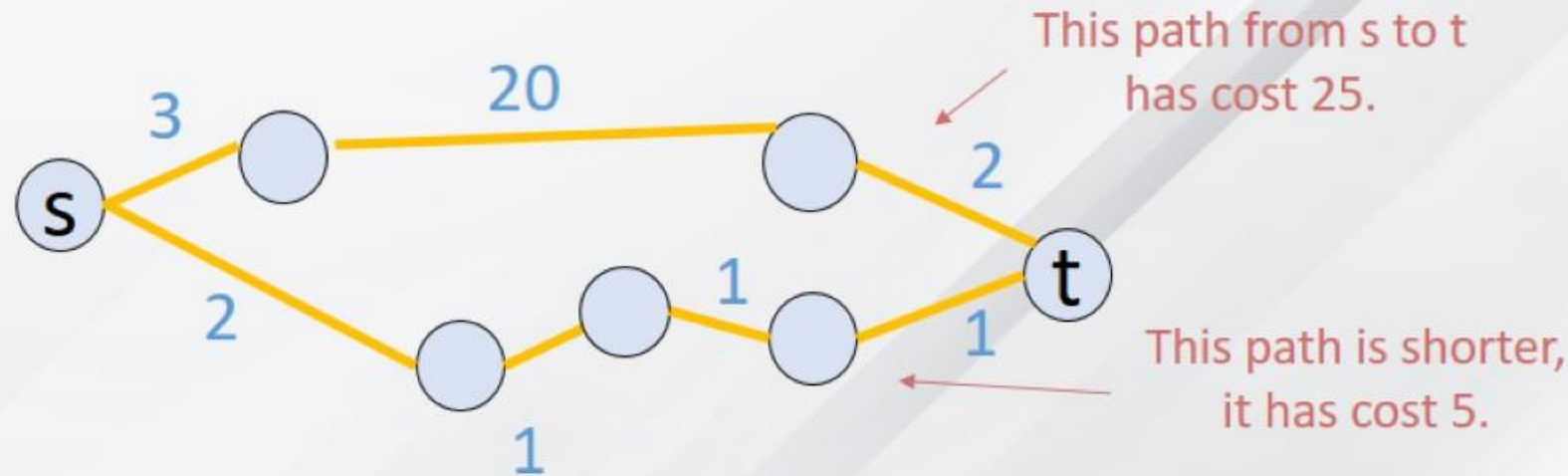
c) 0, 1, 4, 6

d) 0, 1, 4, 7

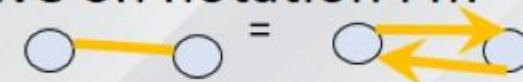


Single-Source Shortest path problem

- What is the **shortest path** between u and v in a weighted graph?
 - the **cost** of a path is the **sum of the weights along that path**
 - The **shortest path** is the one with the minimum cost.

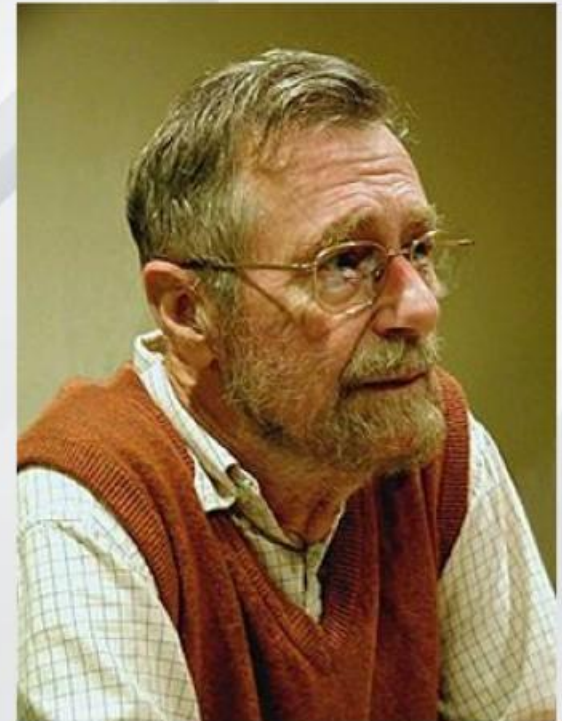


- The **distance** $d(u,v)$ between two vertices u and v is the cost of the shortest path between u and v .
- For this lecture **all graphs are directed**, but to save on notation I'm just going to draw undirected edges.



Edsger Wybe Dijkstra

- May 11, 1930 – August 6, 2002
- Received the 1972 A. M. Turing Award, widely considered the most prestigious **award in computer science**.
- The Schlumberger Centennial Chair of Computer Sciences at **The University of Texas at Austin from 1984 until 2000**
- Made a strong case against use of the GOTO statement in programming languages and helped lead to its deprecation.
- Known for his many essays on programming.



Dijkstra's algorithm

Dijkstra's algorithm - is a solution to the single-source shortest path problem in graph theory.

- Works on both **directed** and **undirected** graphs. However,
- All edges must have **nonnegative** weights.

Approach: Greedy

Input: Weighted graph $G=\{E,V\}$ and source vertex $v \in V$, such that all edge weights are nonnegative

04

Output: Lengths of shortest paths (or the shortest paths themselves) from a given source vertex $v \in V$ to all other vertices

Dijkstra's algorithm - Idea

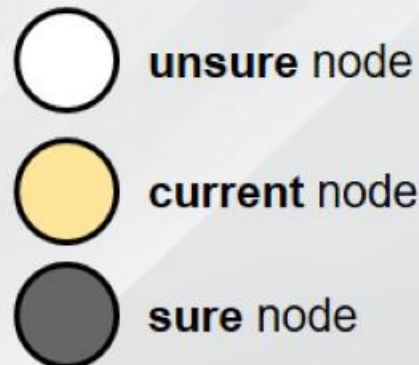
Source node is **s**.

For each node **v**, store an **over-estimate** of $d(s, v)$.

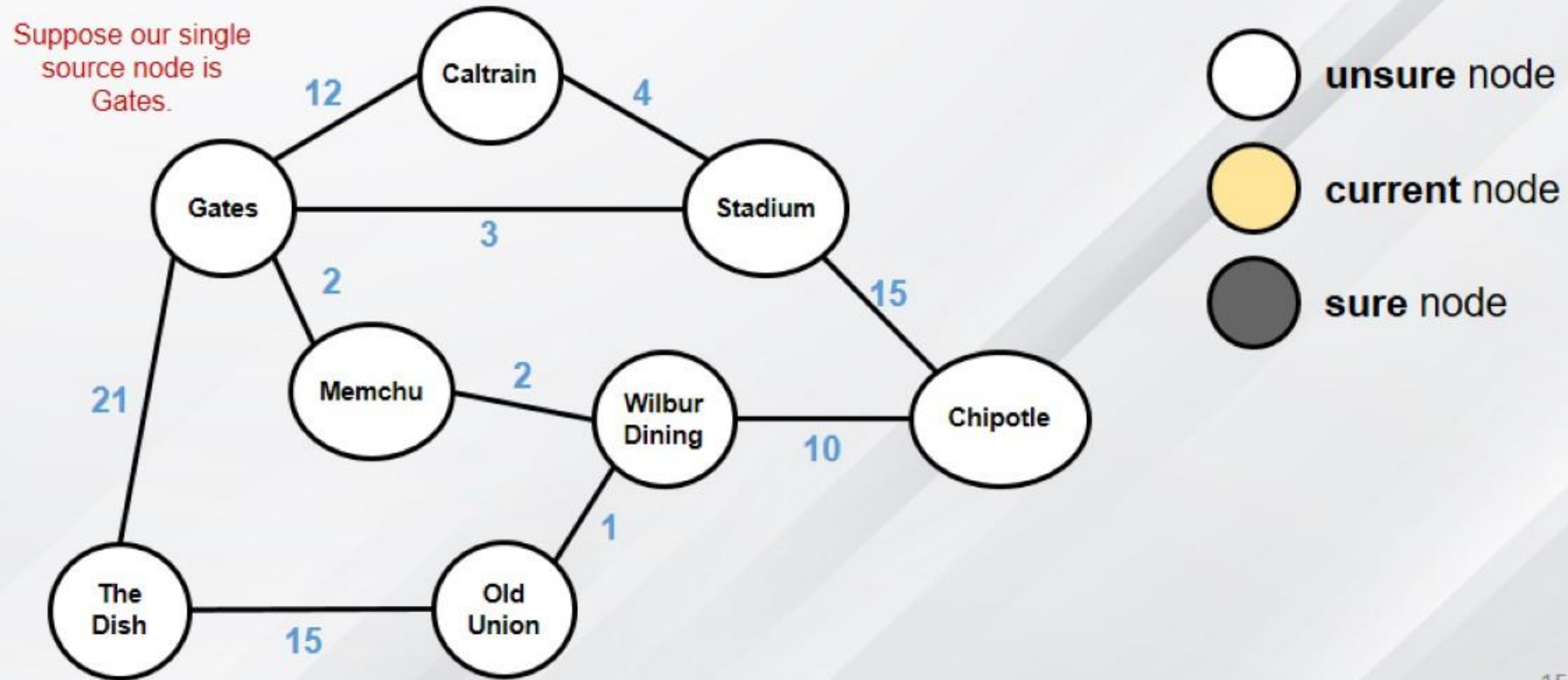
These over-estimates start off as ∞ , and we'll iteratively work to make these over-estimates tighter and tighter, until they reach the true distance.

- Each iteration of its main loop processes one new vertex.
- The algorithm's sophistication lies in its clever rule for selecting which vertex to process next: the **not-yet-processed vertex that appears to be closest to the starting vertex**.

For demonstration purposes, we'll use three colors again to keep track of the state of each node:



DIJKSTRA BY EXAMPLE



15

DIJKSTRA BY EXAMPLE



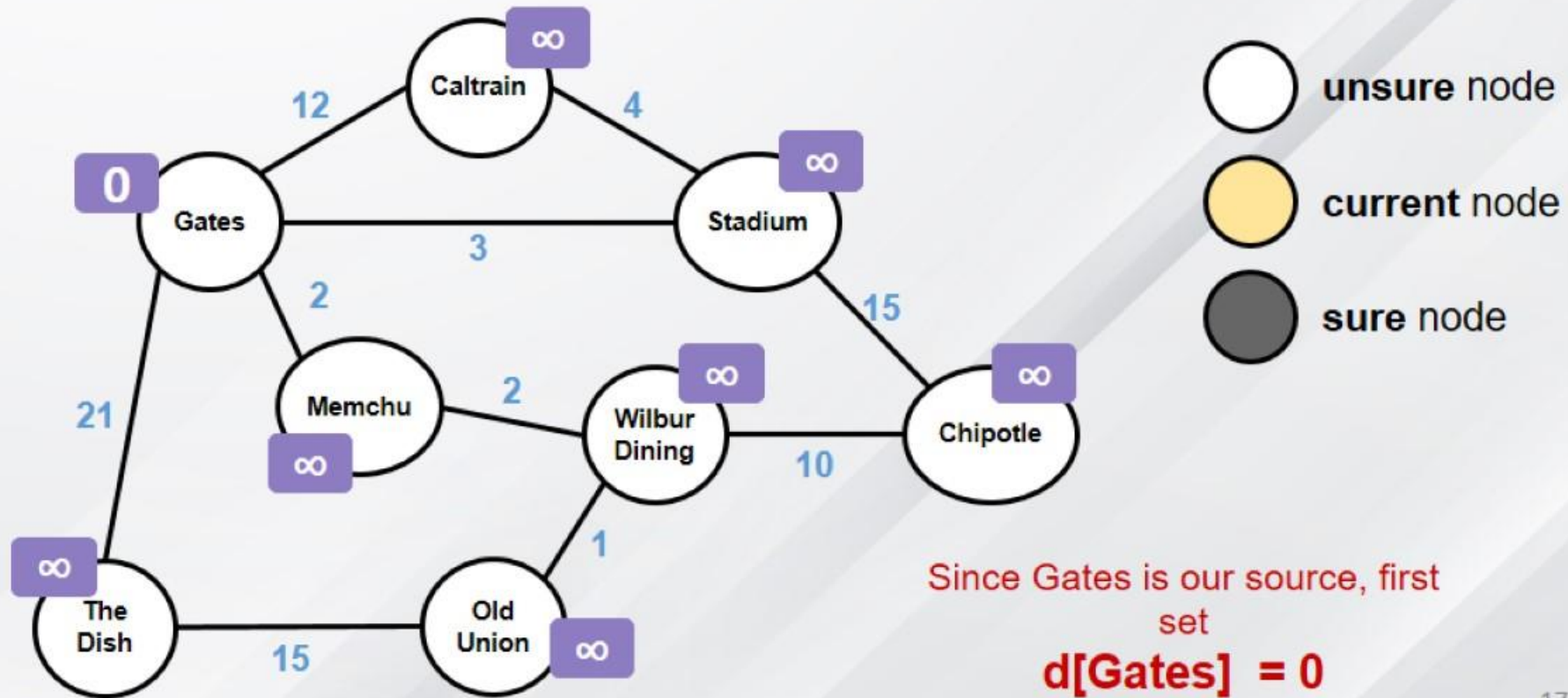
We initialize all our overestimates:

$$d[v] = \infty$$

for all vertices v .

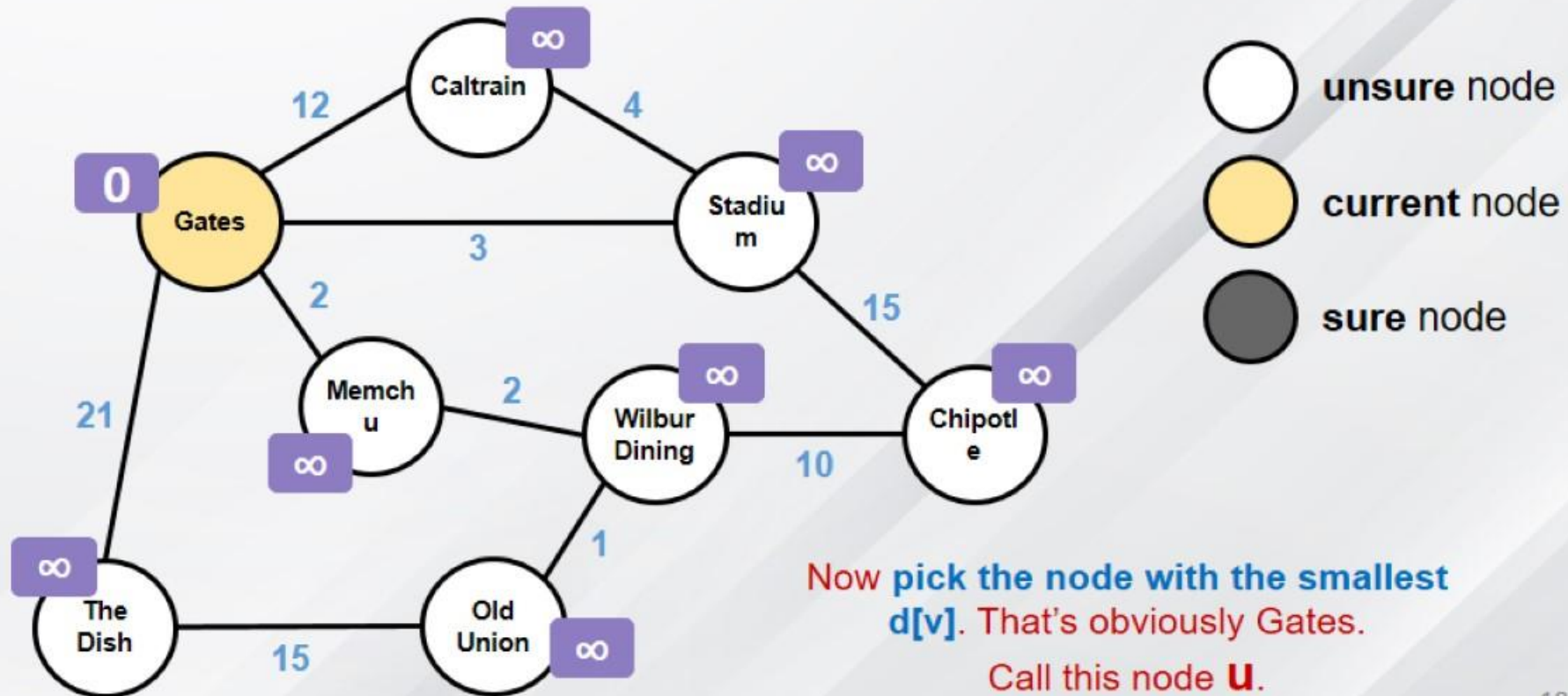
16

DIJKSTRA BY EXAMPLE



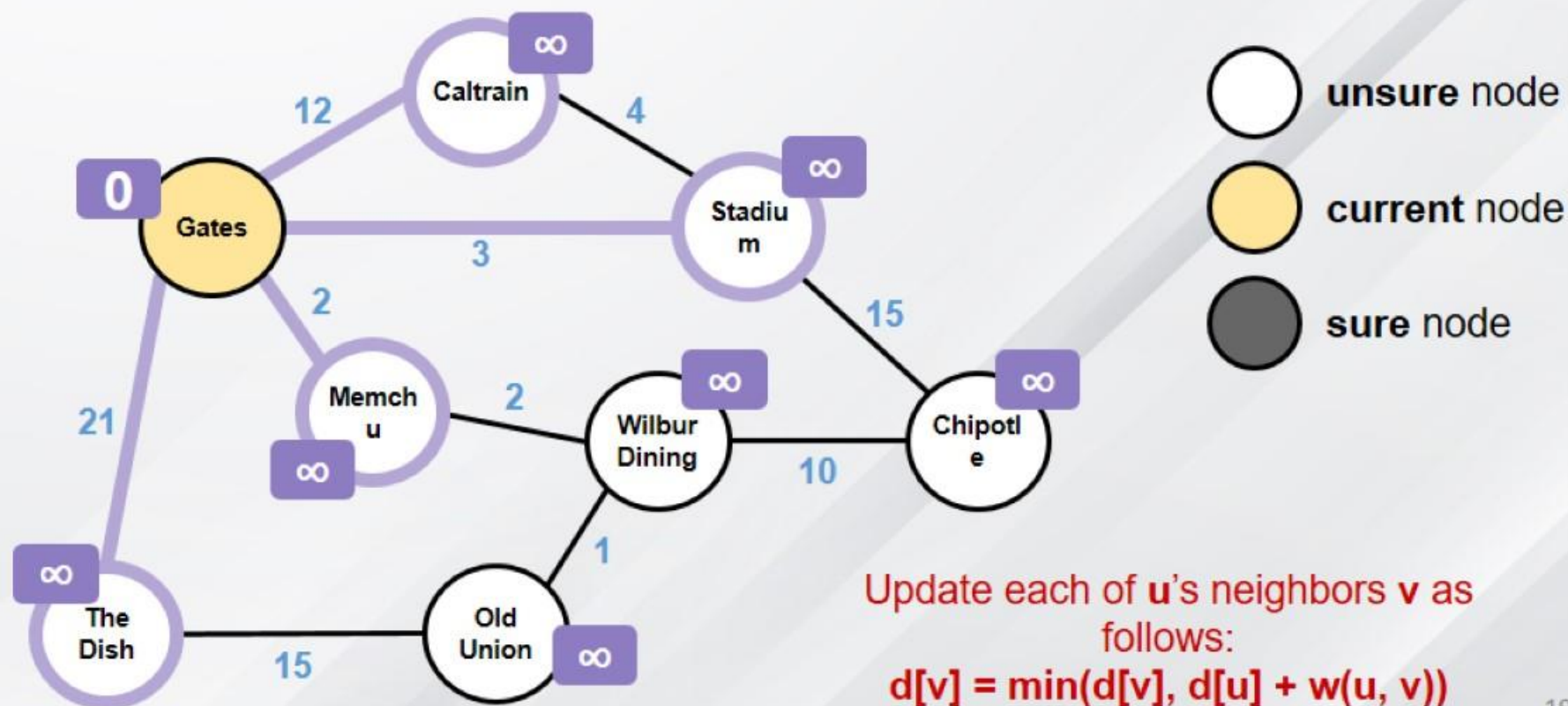
17

DIJKSTRA BY EXAMPLE



18

DIJKSTRA BY EXAMPLE



Update each of **u**'s neighbors **v** as follows:

$$d[v] = \min(d[v], d[u] + w(u, v))$$

$$= \min(\infty, 0 + 12) = \min(\infty, 12) = 12$$

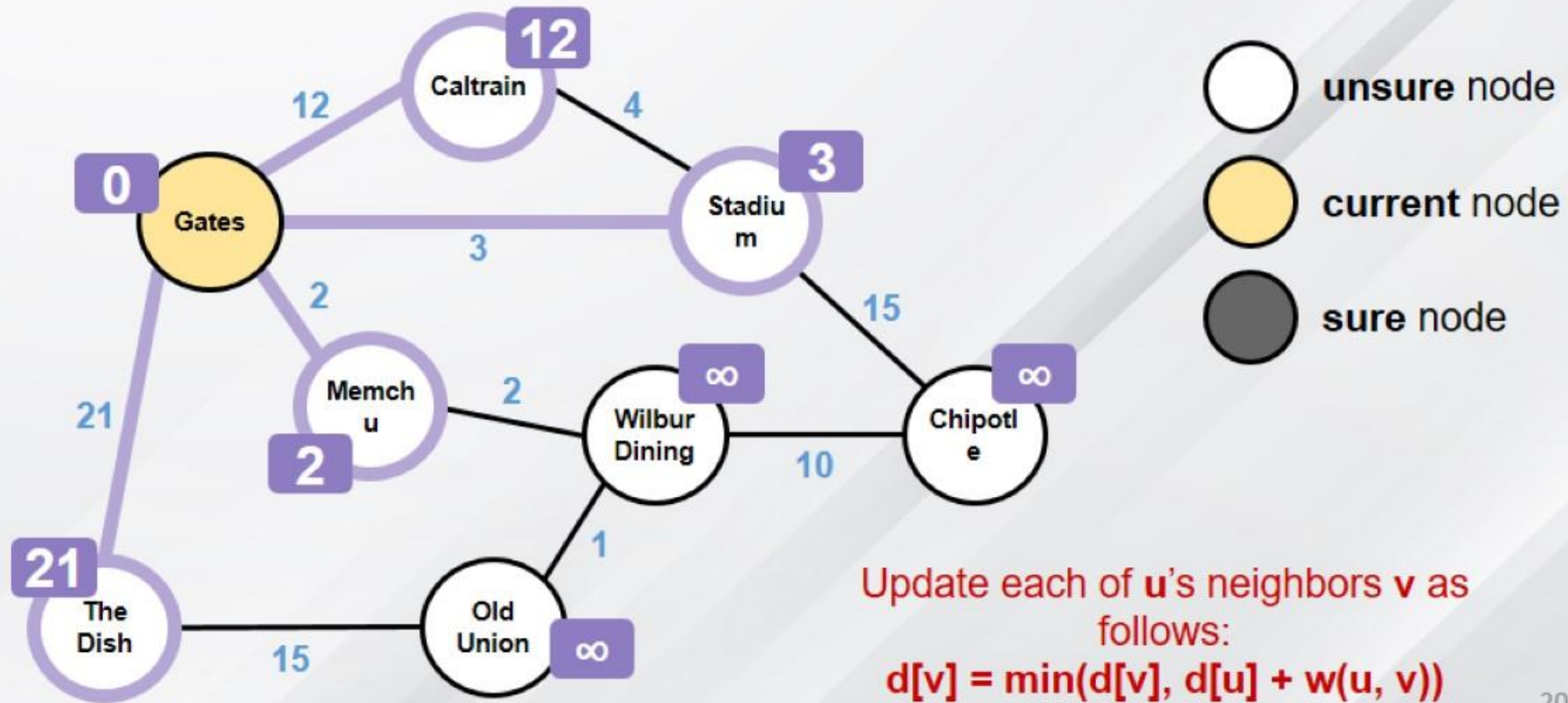
$$\min(\infty, 0 + 3) = 3$$

$$= \min(\infty, 0 + 2) = 2$$

$$= \min(\infty, 0 + 21) = 21$$

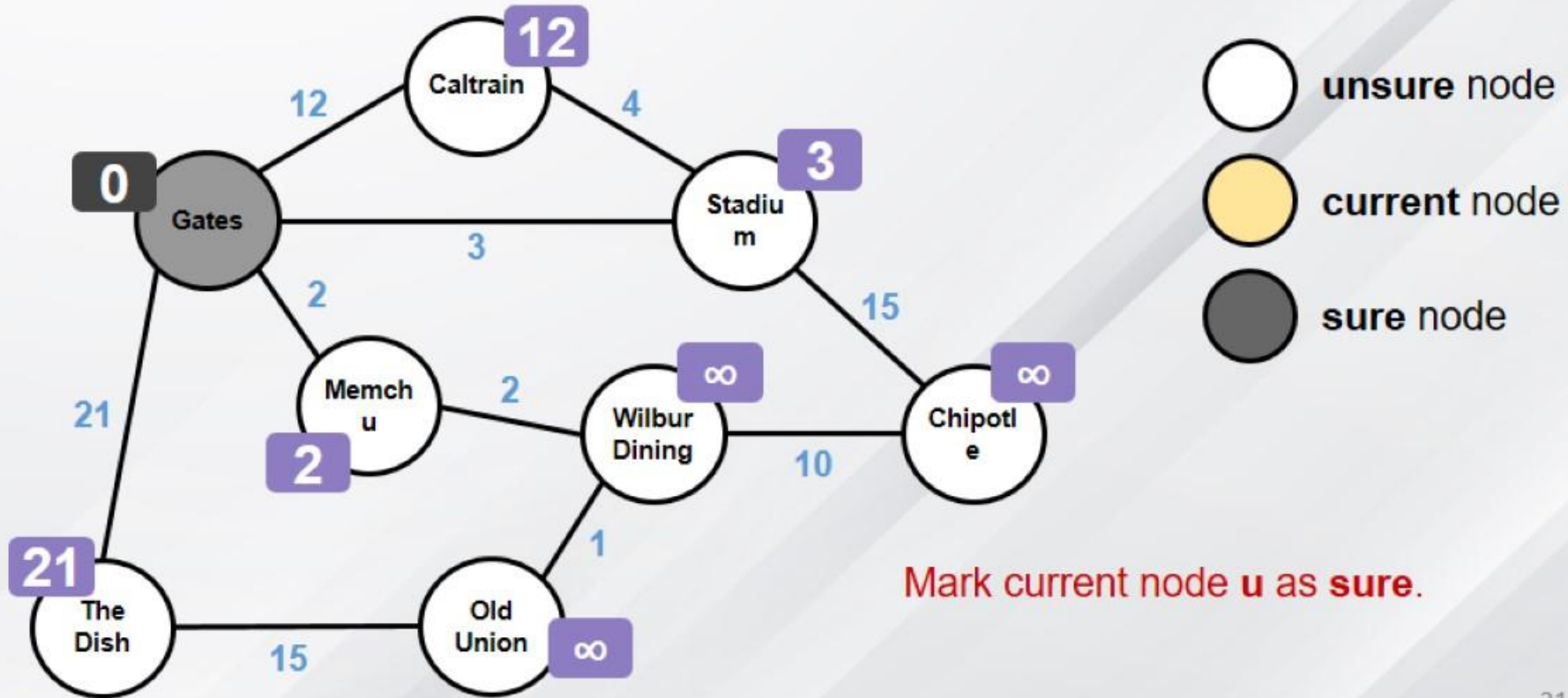
19

DIJKSTRA BY EXAMPLE



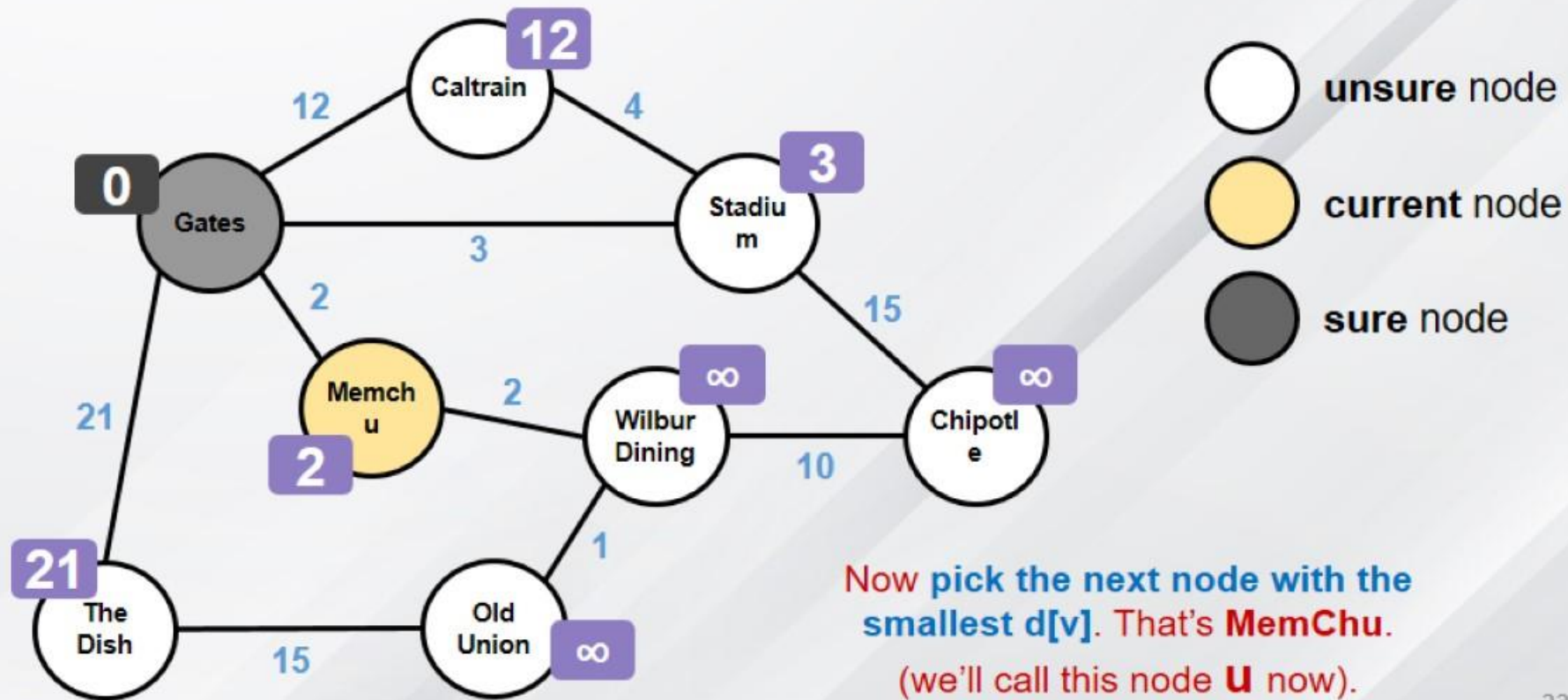
20

DIJKSTRA BY EXAMPLE

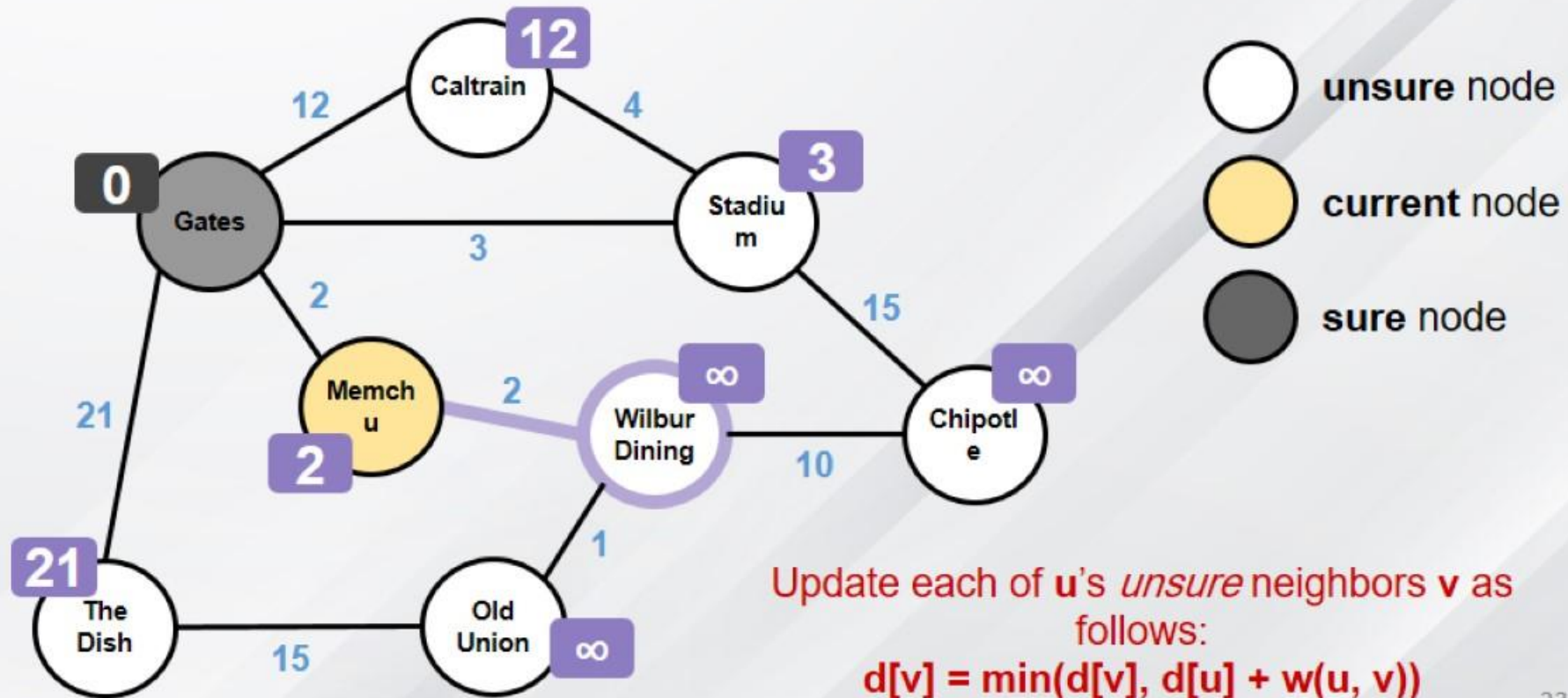


Mark current node **u** as **sure**.

DIJKSTRA BY EXAMPLE



DIJKSTRA BY EXAMPLE



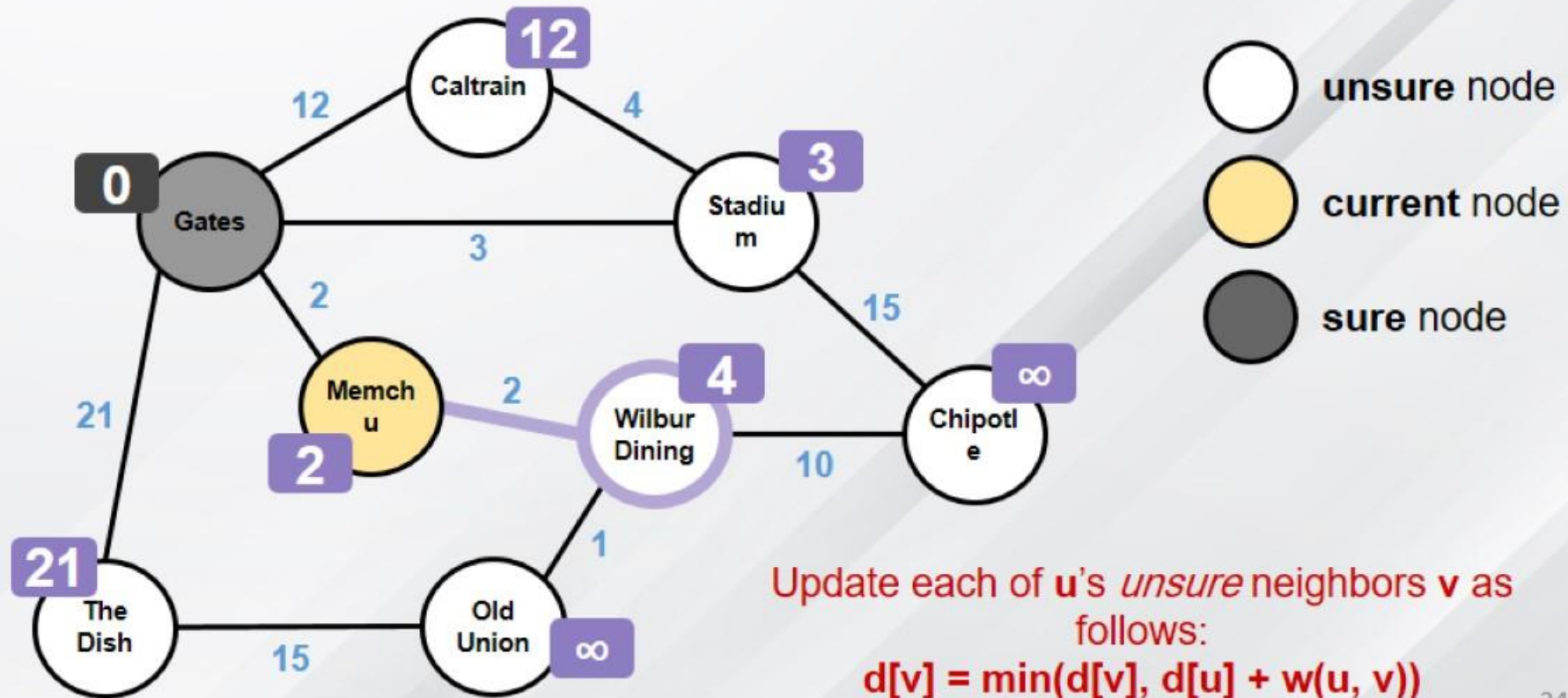
Update each of u 's *unsure* neighbors v as follows:

$$d[v] = \min(d[v], d[u] + w(u, v))$$

$$= \min(\infty, 2+2)=4$$

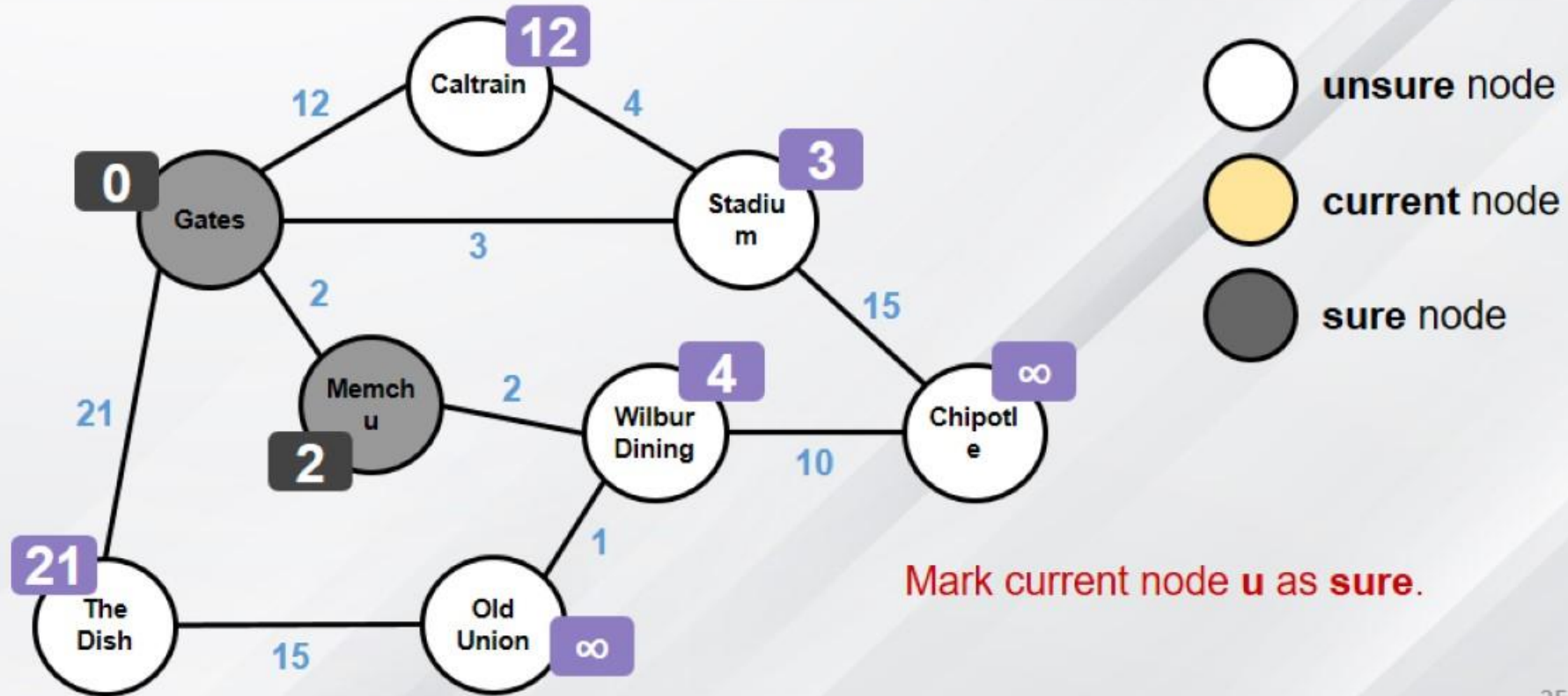
23

DIJKSTRA BY EXAMPLE



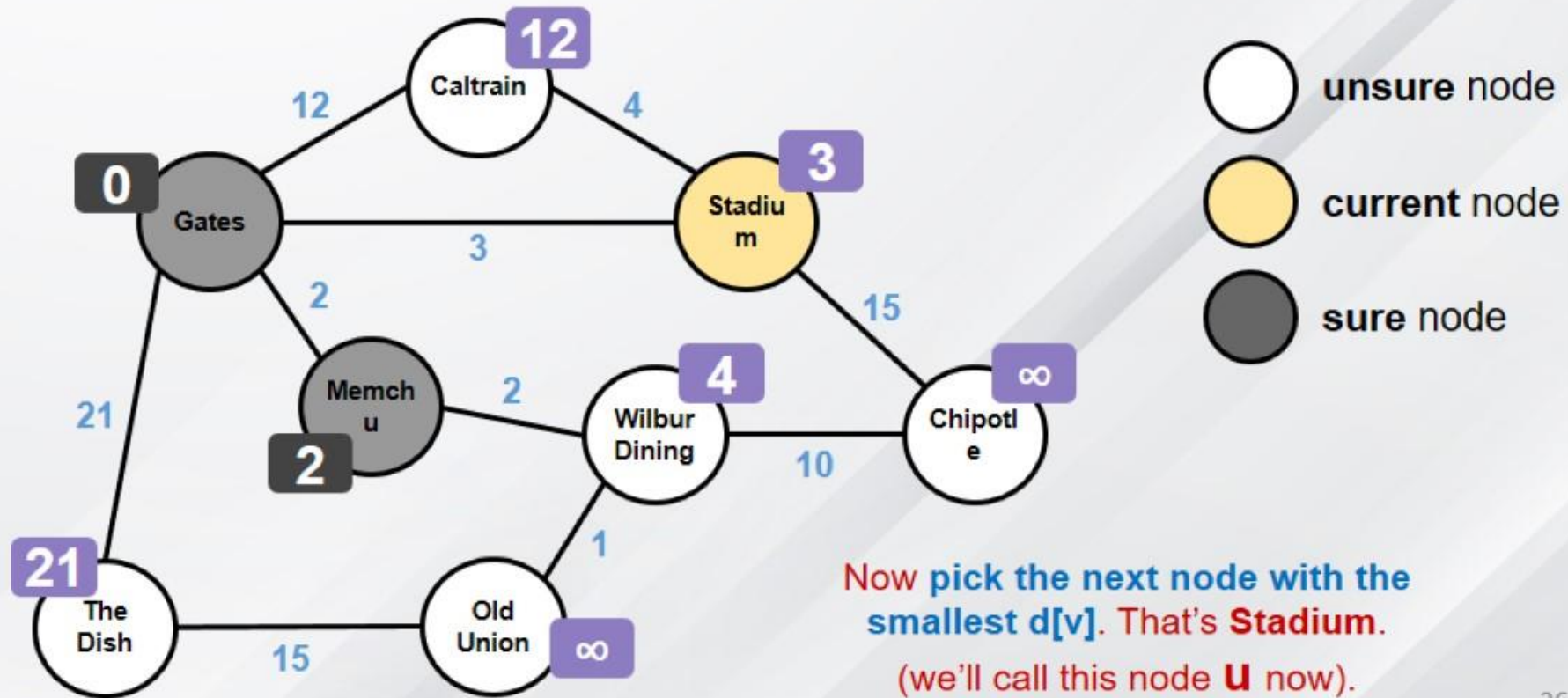
24

DIJKSTRA BY EXAMPLE

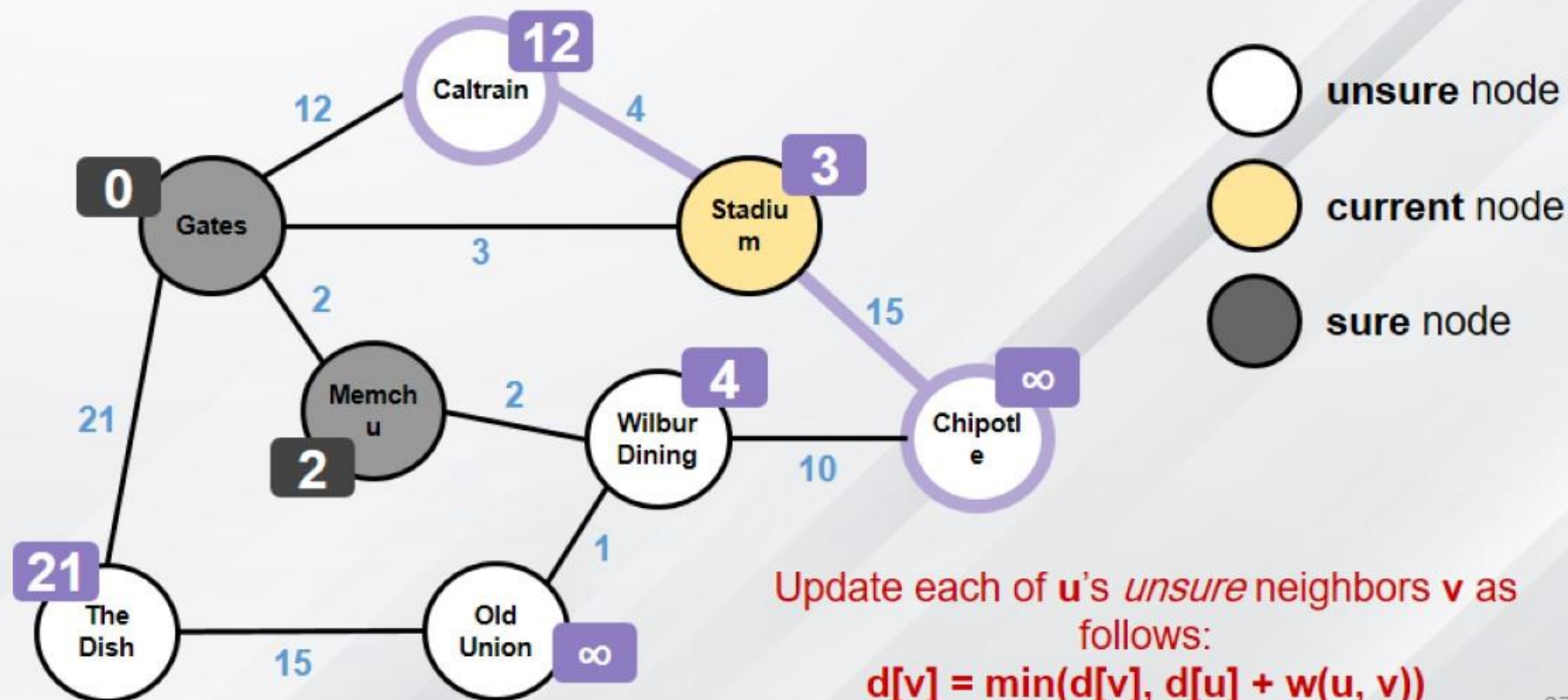


25

DIJKSTRA BY EXAMPLE



DIJKSTRA BY EXAMPLE



Update each of u 's *unsure* neighbors v as follows:

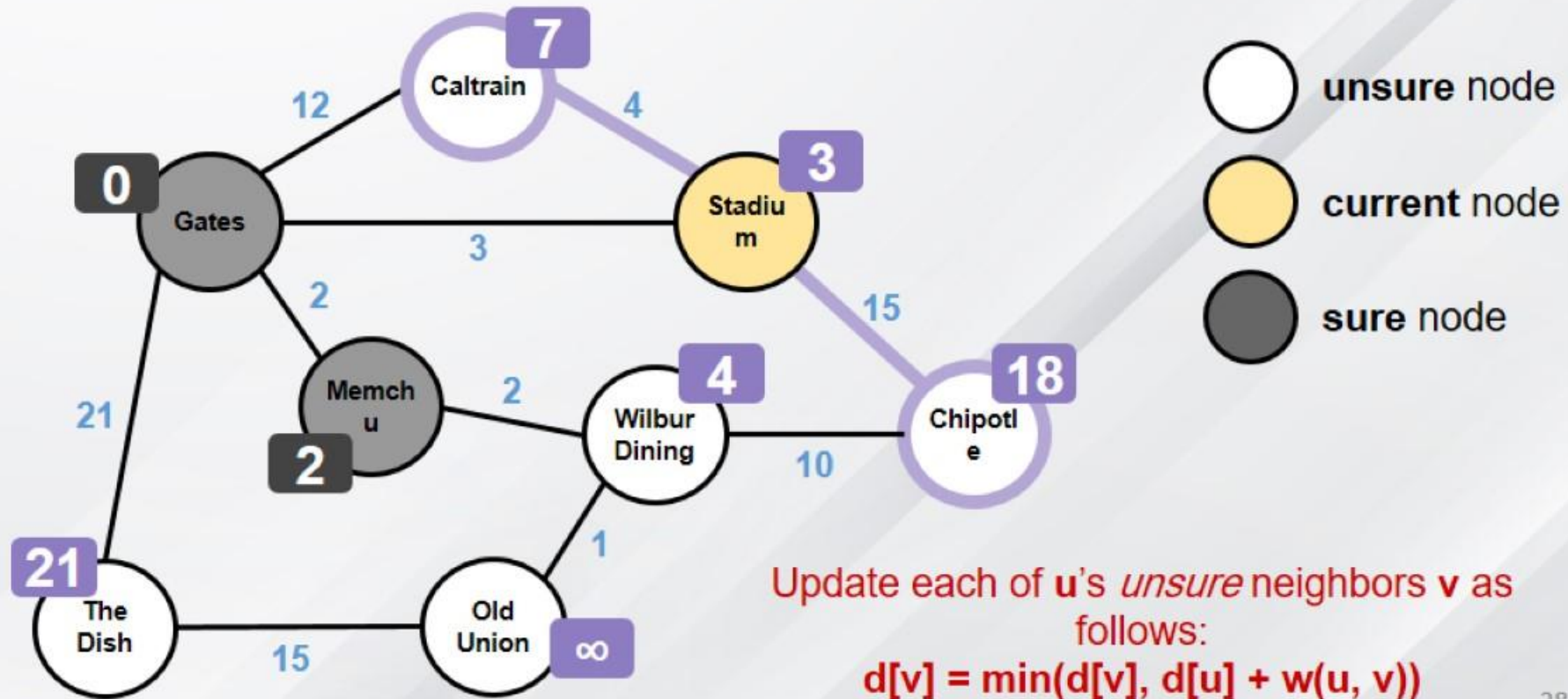
$$d[v] = \min(d[v], d[u] + w(u, v))$$

$$\text{Min}(12, 3+4) = \text{min}(12, 7) = 7$$

$$\text{Min}(\infty, 3+15) = \text{min}(\infty, 18) = 18$$

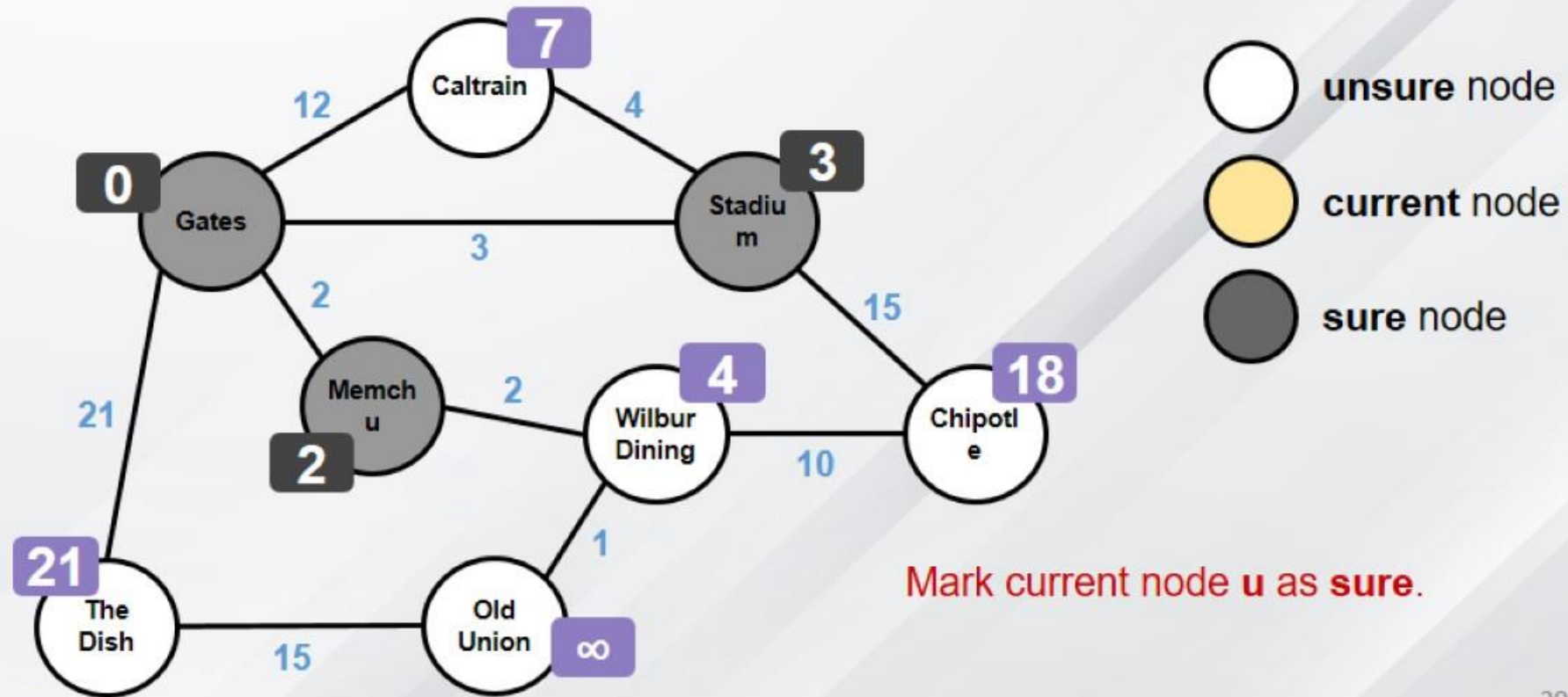
27

DIJKSTRA BY EXAMPLE

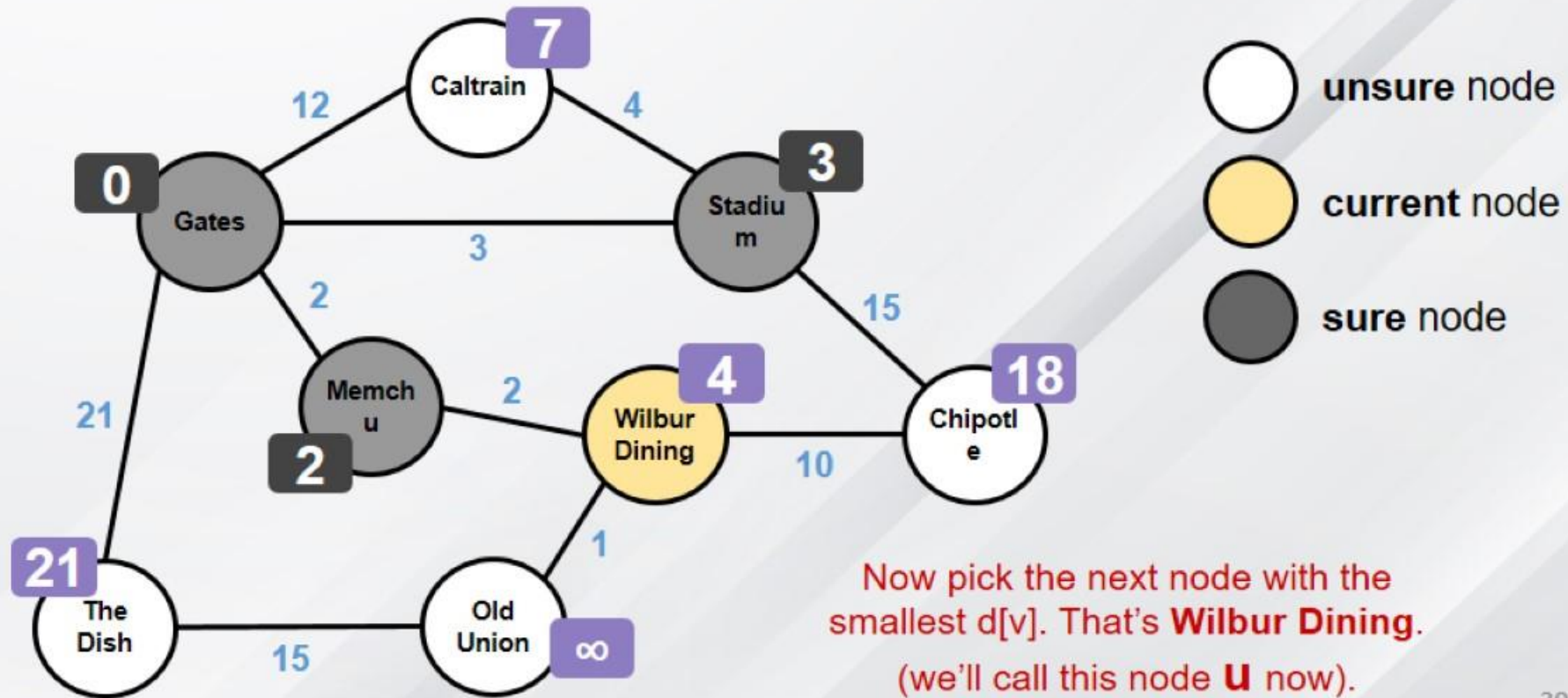


28

DIJKSTRA BY EXAMPLE

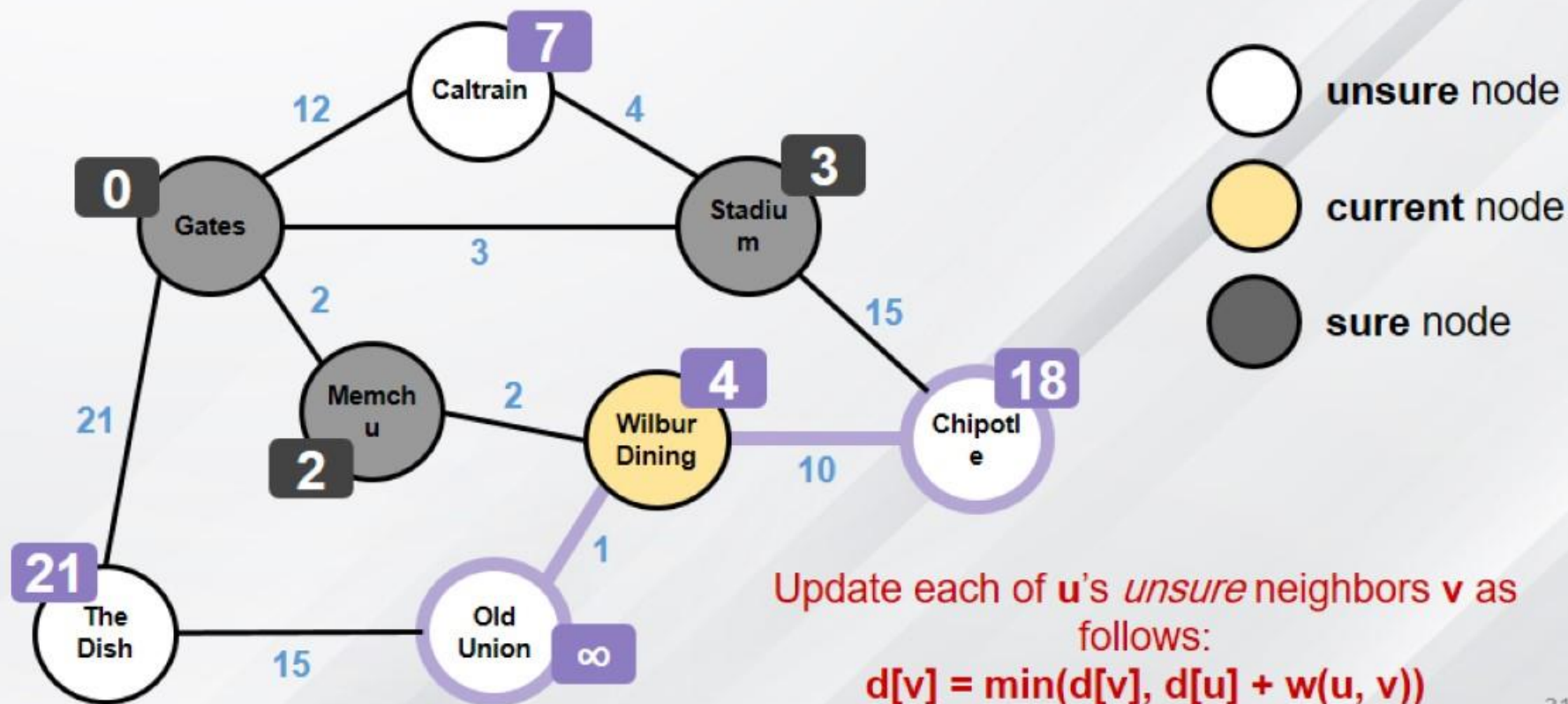


DIJKSTRA BY EXAMPLE



30

DIJKSTRA BY EXAMPLE



Update each of u 's *unsure* neighbors v as follows:

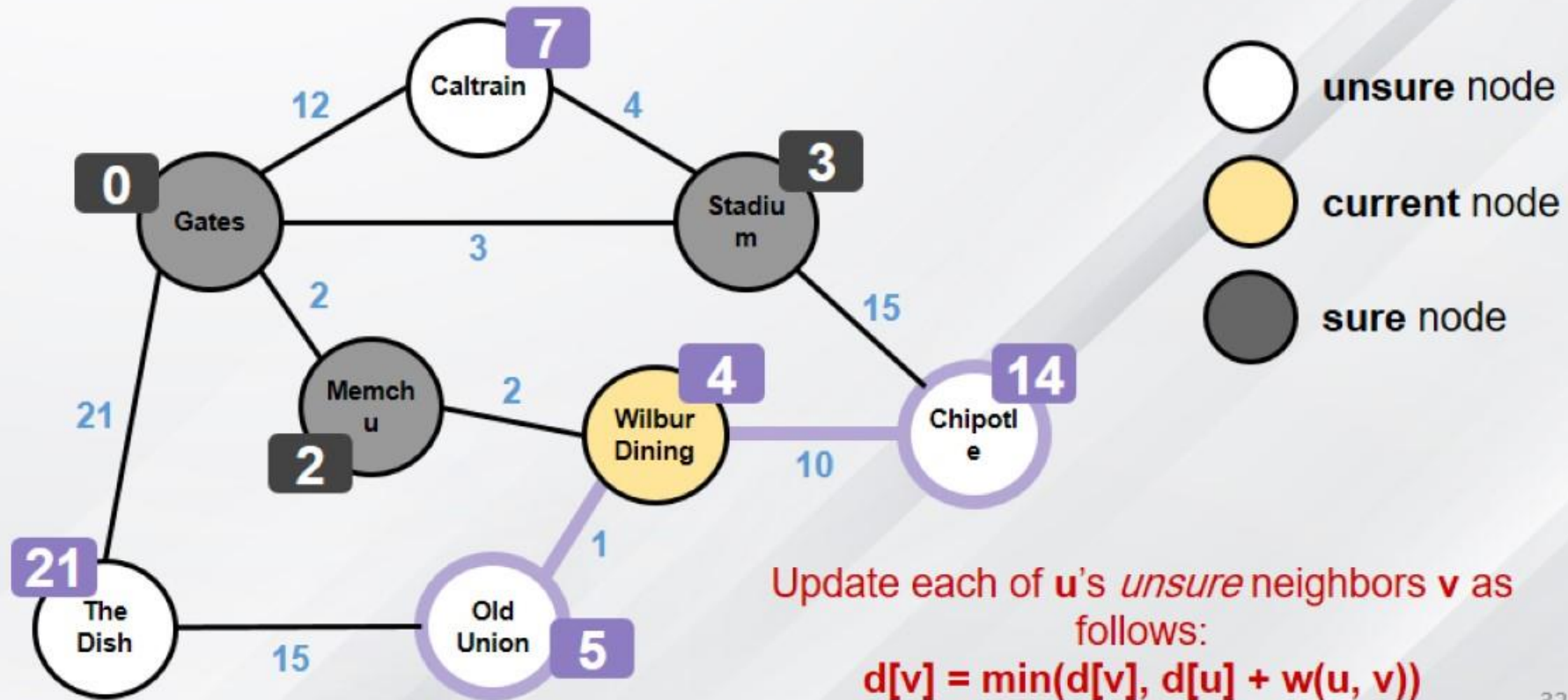
$$d[v] = \min(d[v], d[u] + w(u, v))$$

$$\text{Min}(18, 4+10)=14$$

$$\text{Min}(\infty, 4+1)=5$$

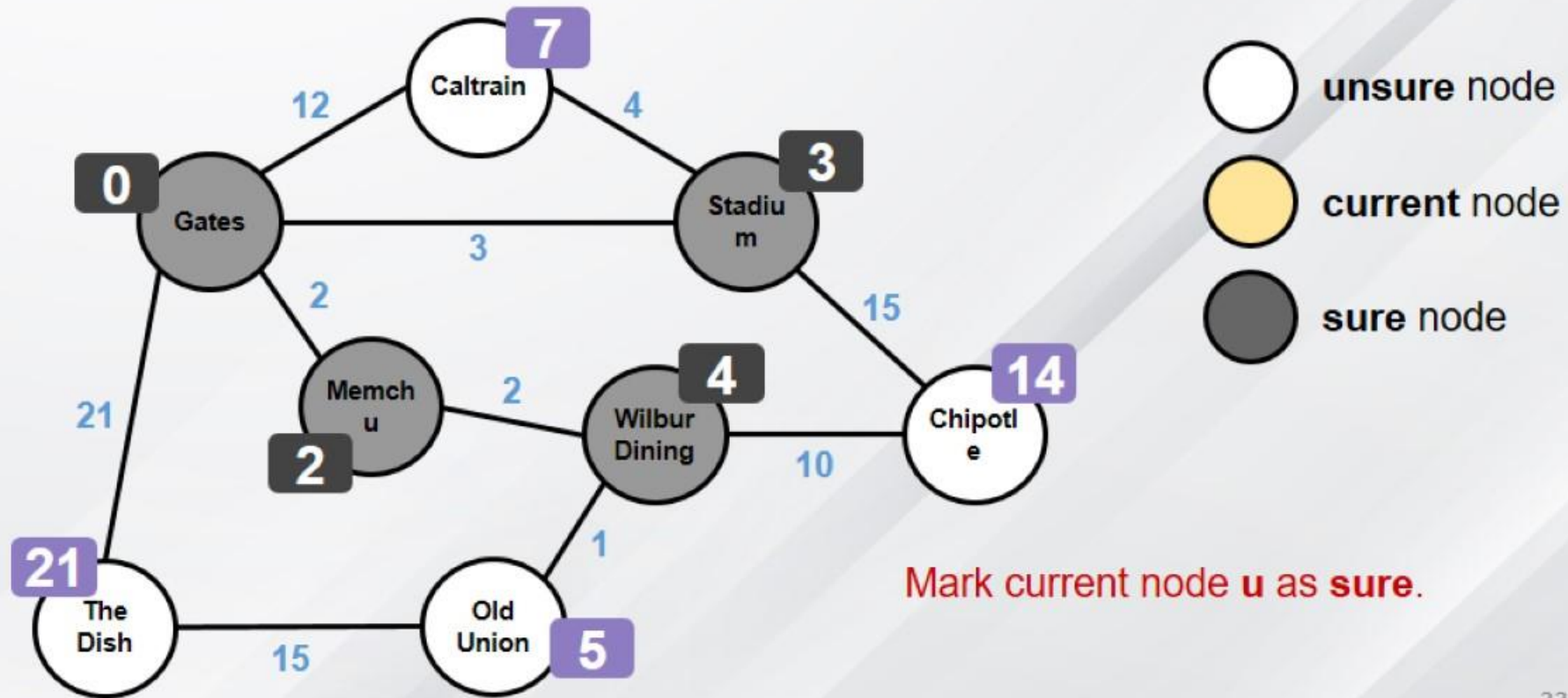
31

DIJKSTRA BY EXAMPLE

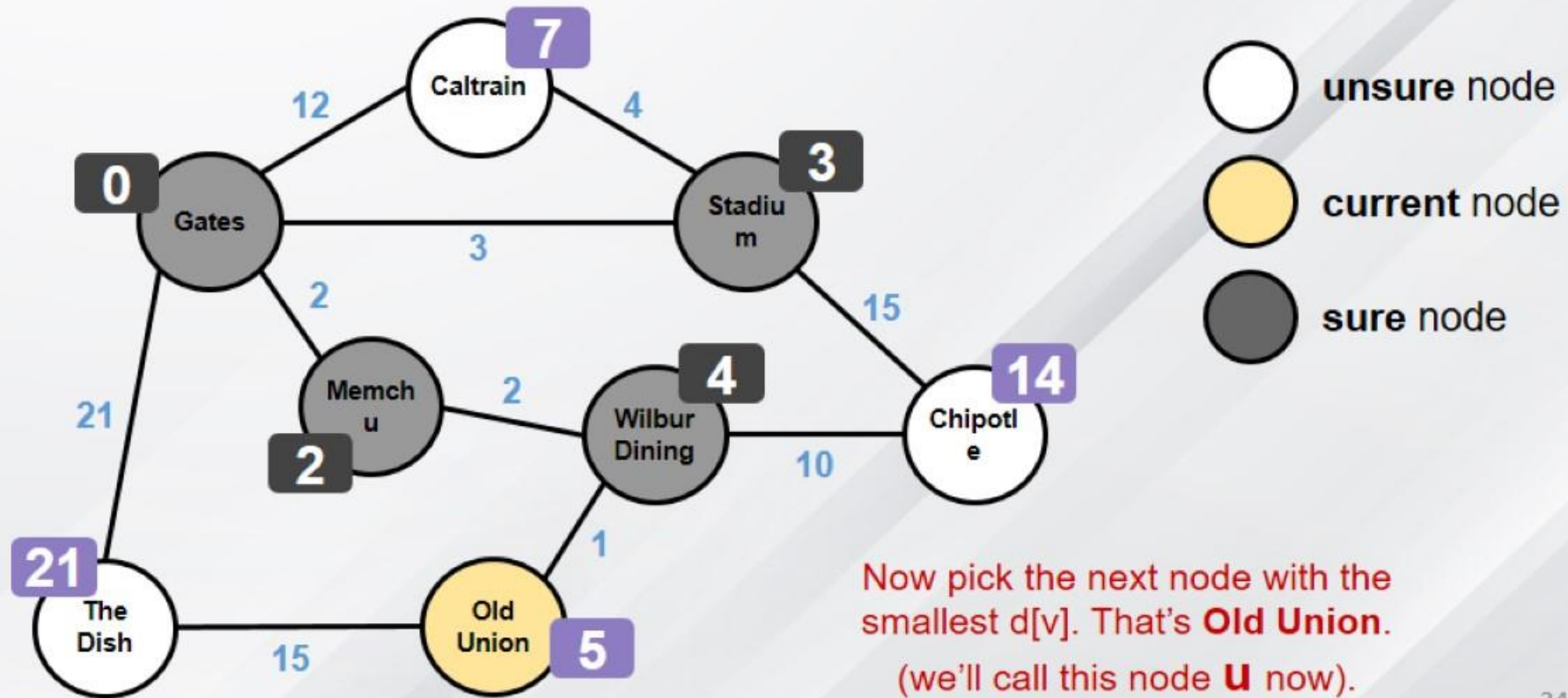


32

DIJKSTRA BY EXAMPLE

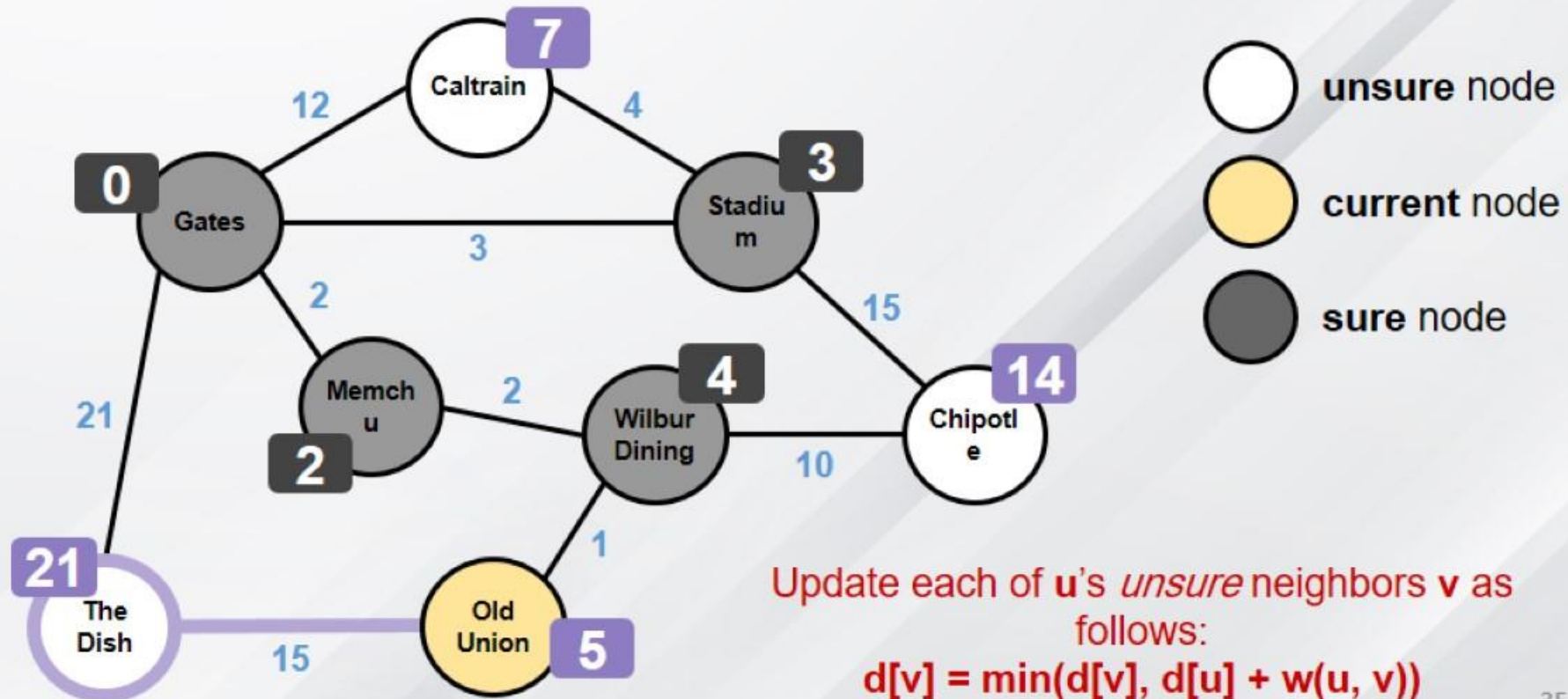


DIJKSTRA BY EXAMPLE



34

DIJKSTRA BY EXAMPLE



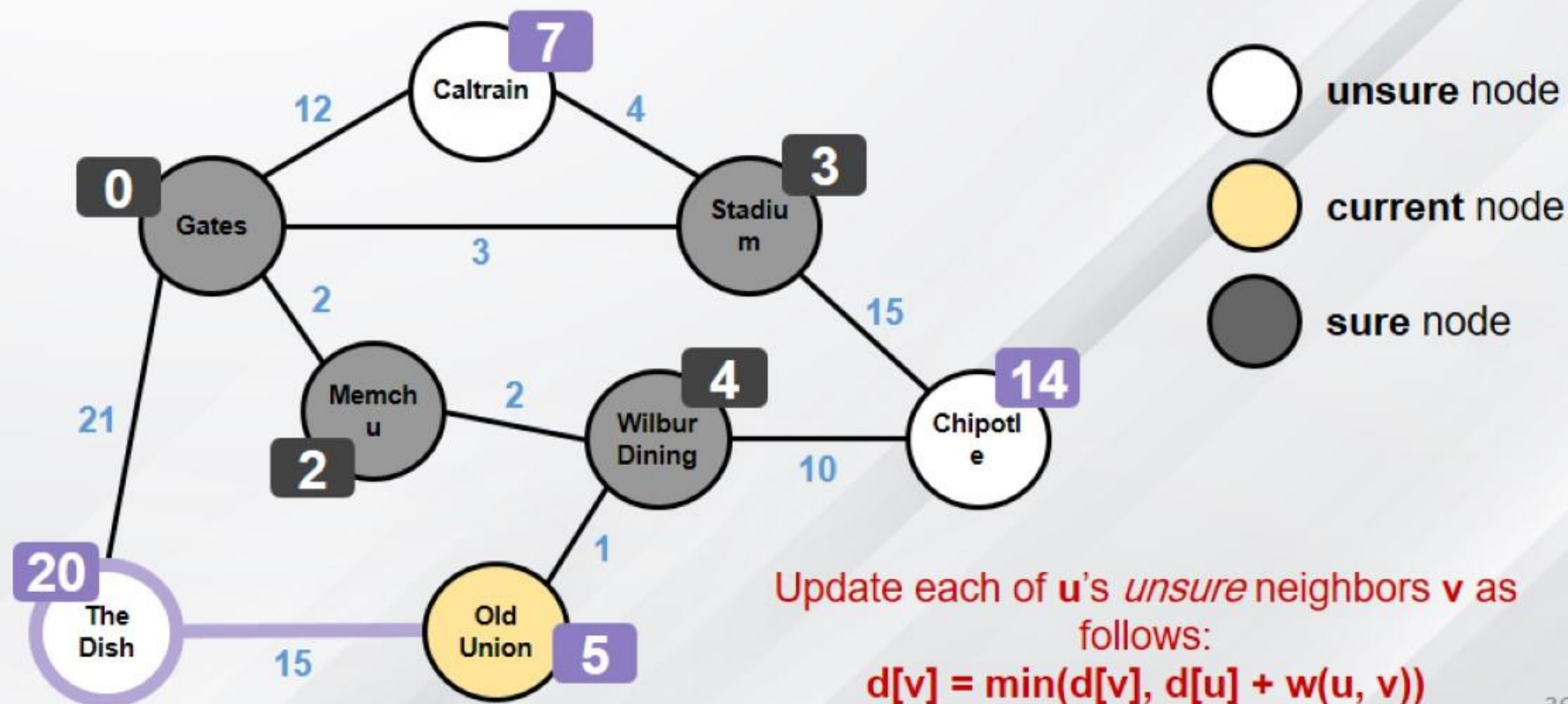
Update each of u 's *unsure* neighbors v as follows:

$$d[v] = \min(d[v], d[u] + w(u, v))$$

$$\text{Min}(21, 5+15)=20$$

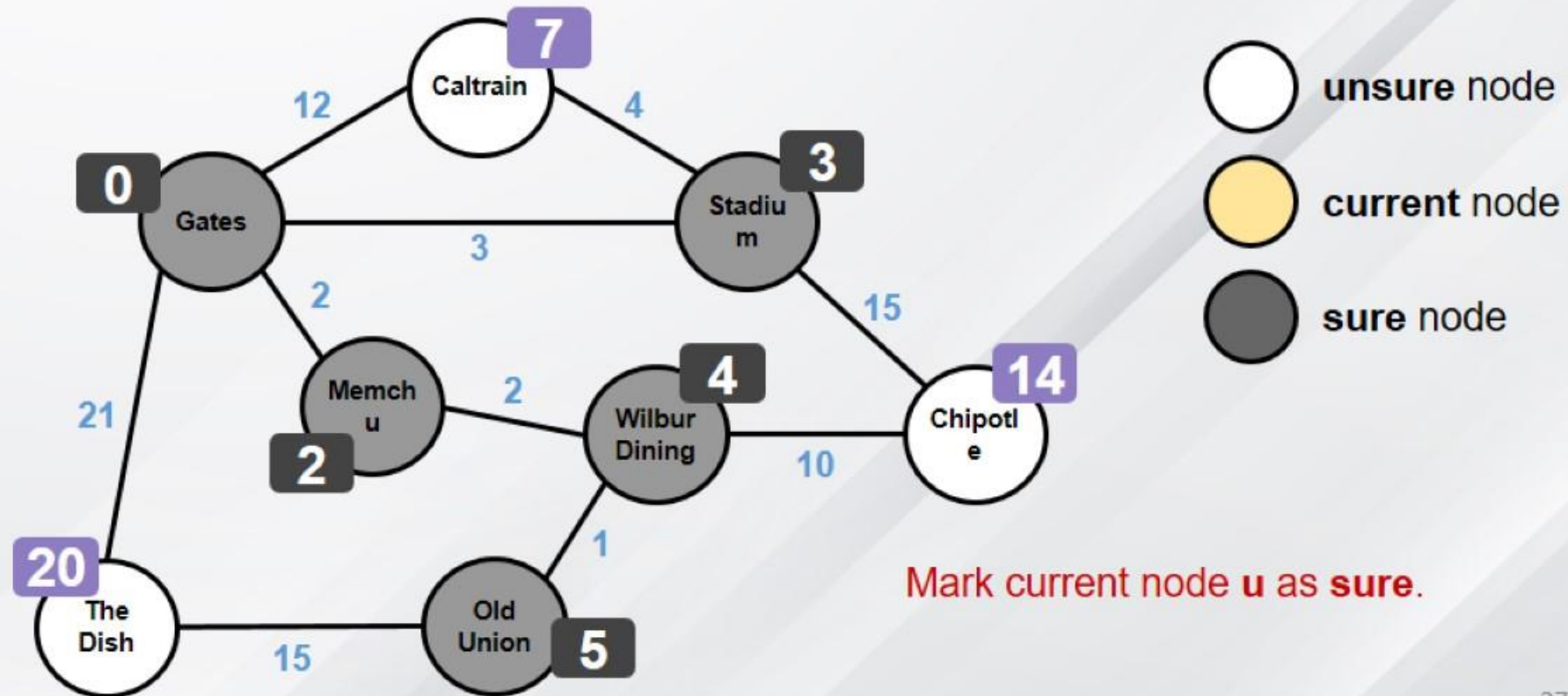
35

DIJKSTRA BY EXAMPLE

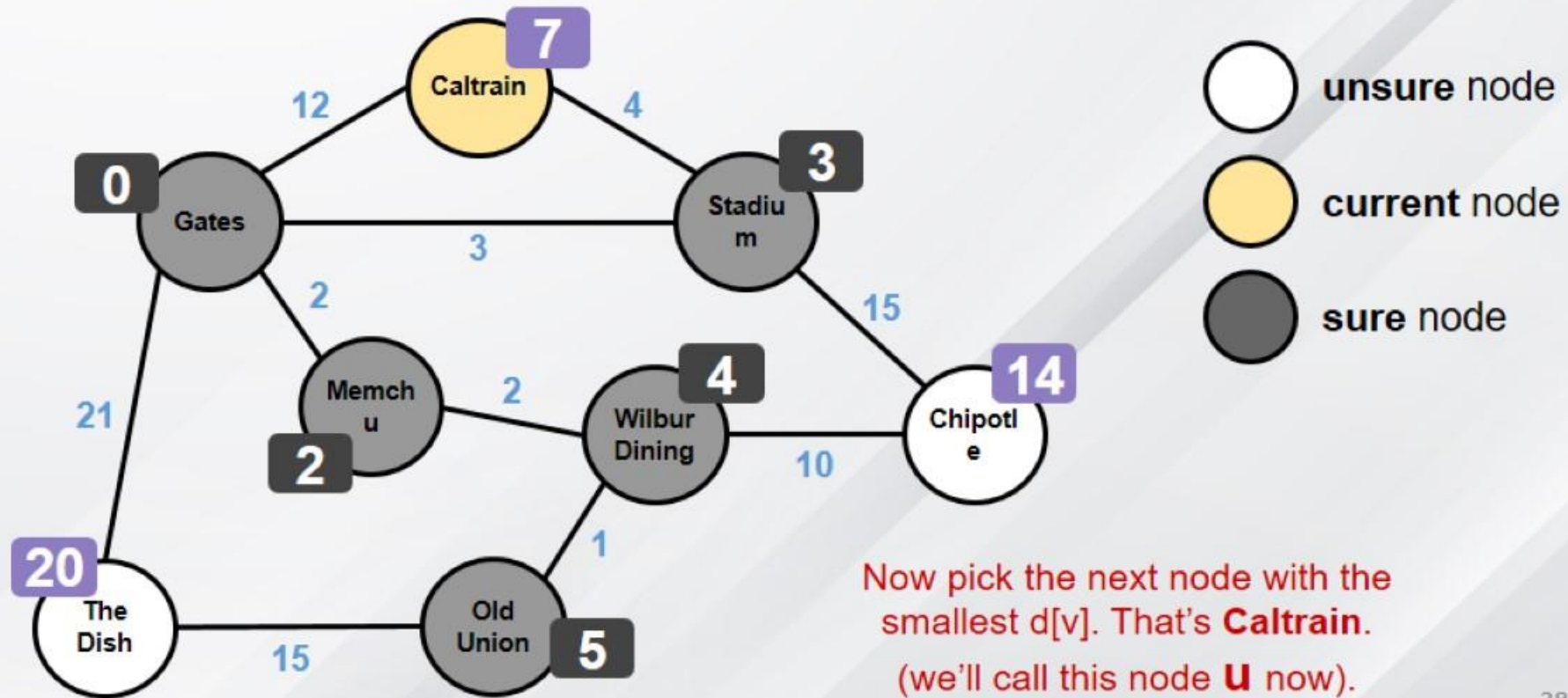


36

DIJKSTRA BY EXAMPLE

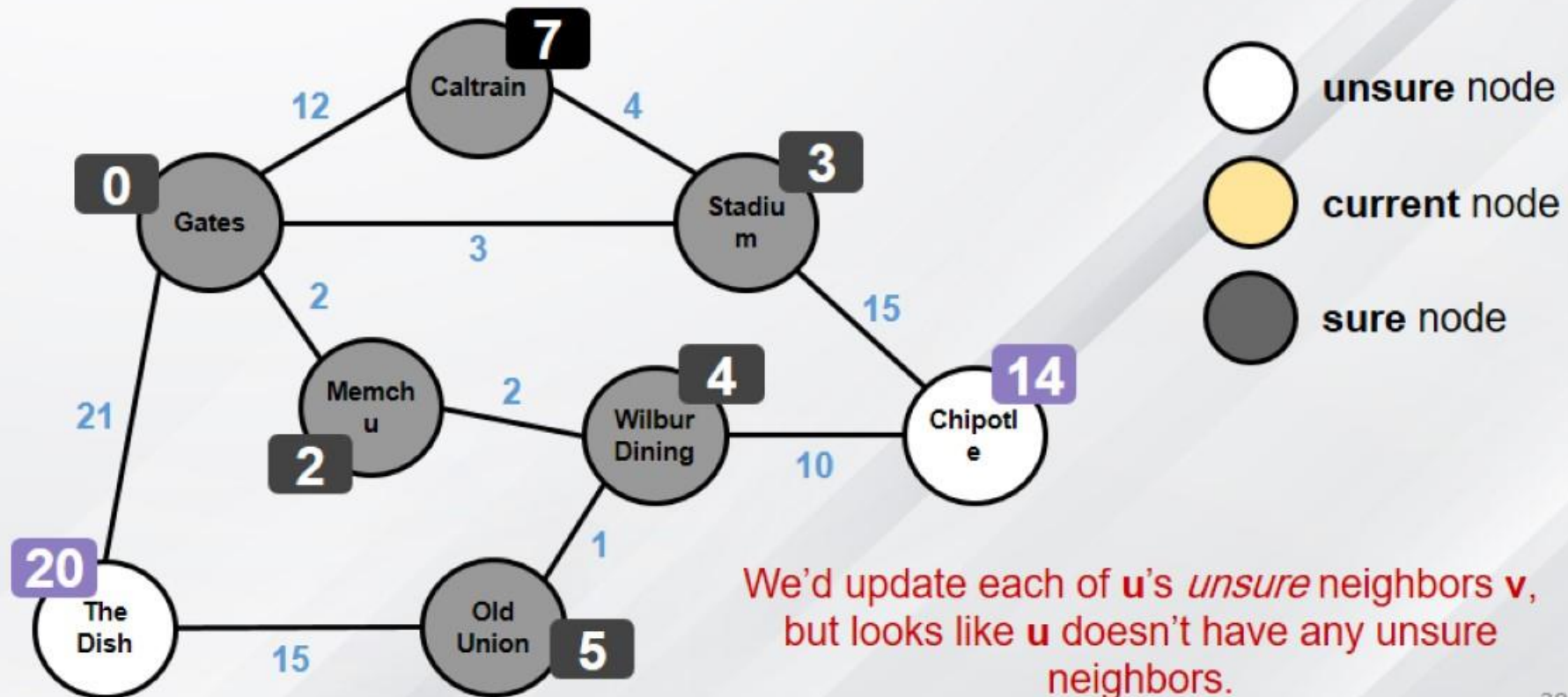


DIJKSTRA BY EXAMPLE



38

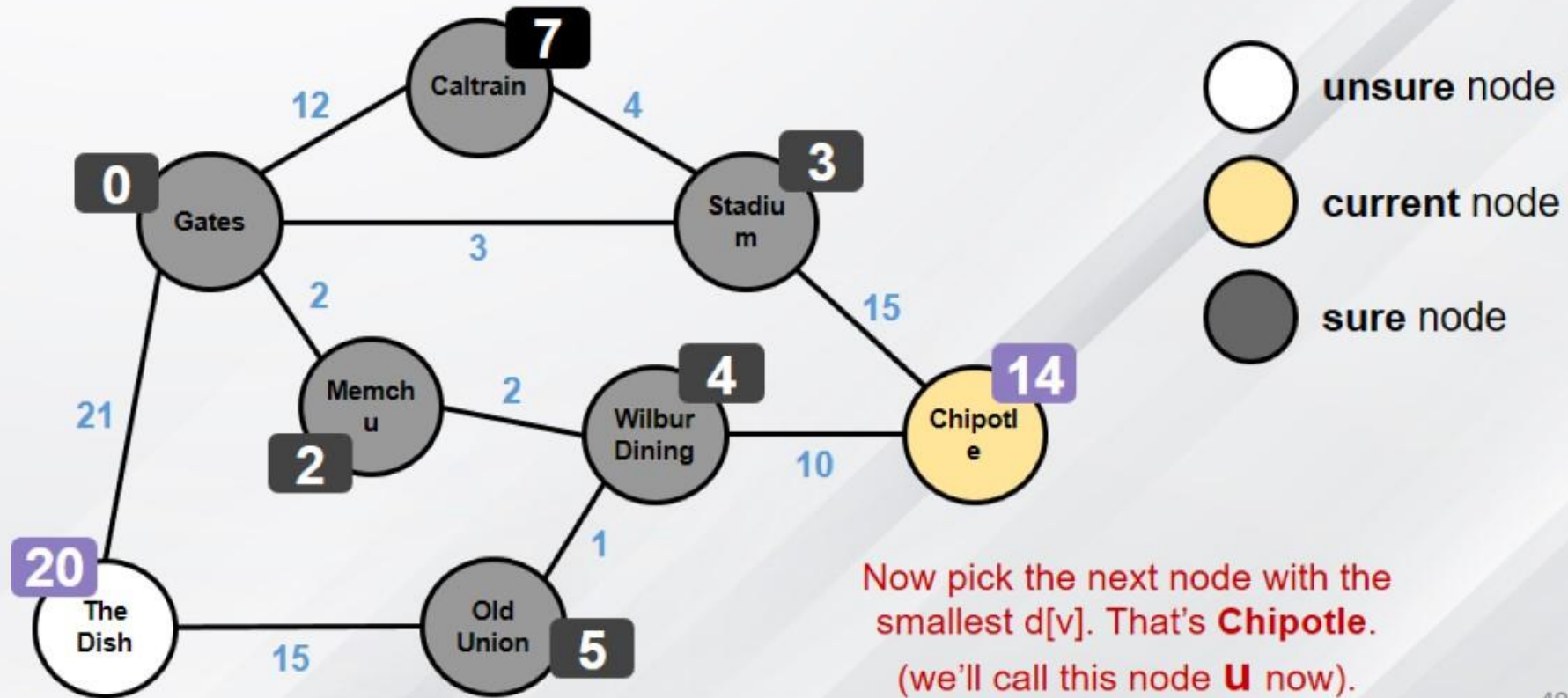
DIJKSTRA BY EXAMPLE



We'd update each of **u**'s *unsure* neighbors **v**,
but looks like **u** doesn't have any unsure
neighbors.
Just mark **u** as **sure**.

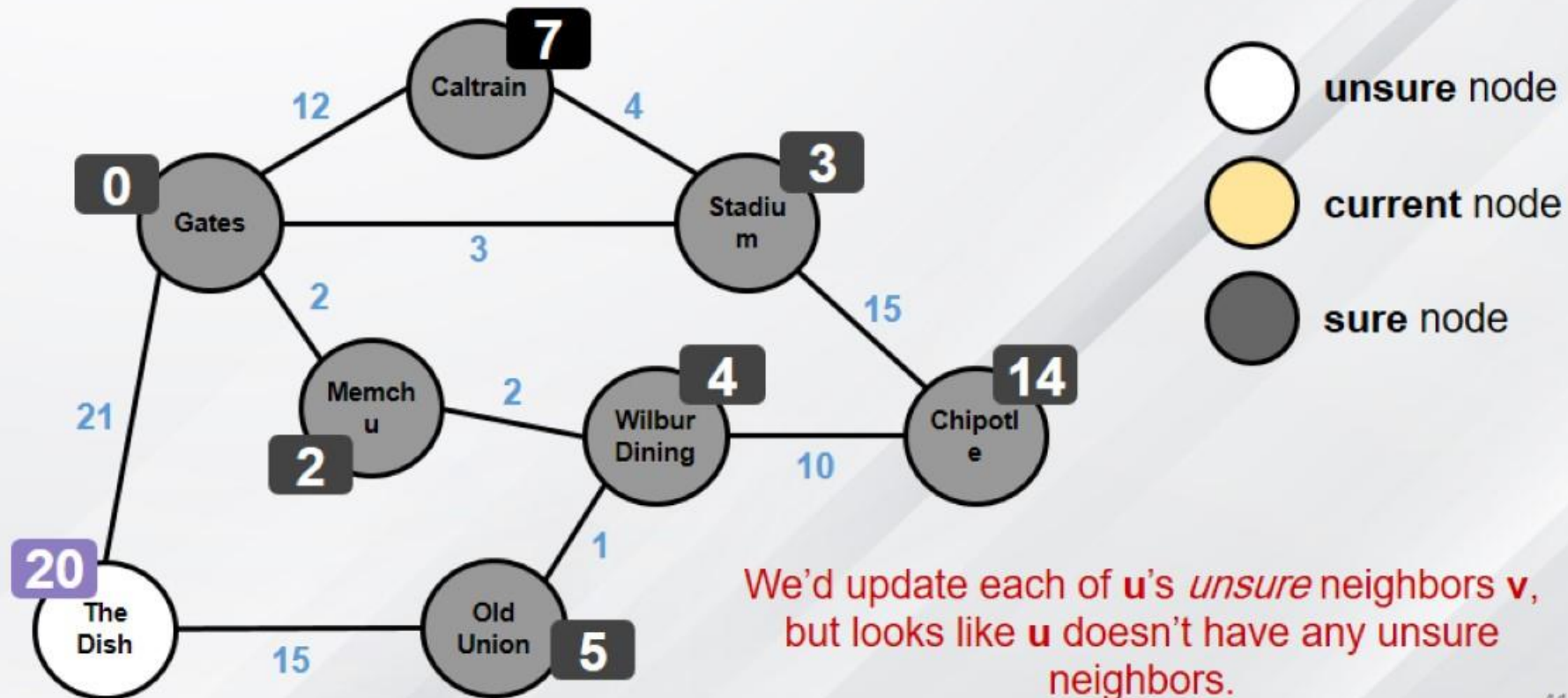
39

DIJKSTRA BY EXAMPLE



40

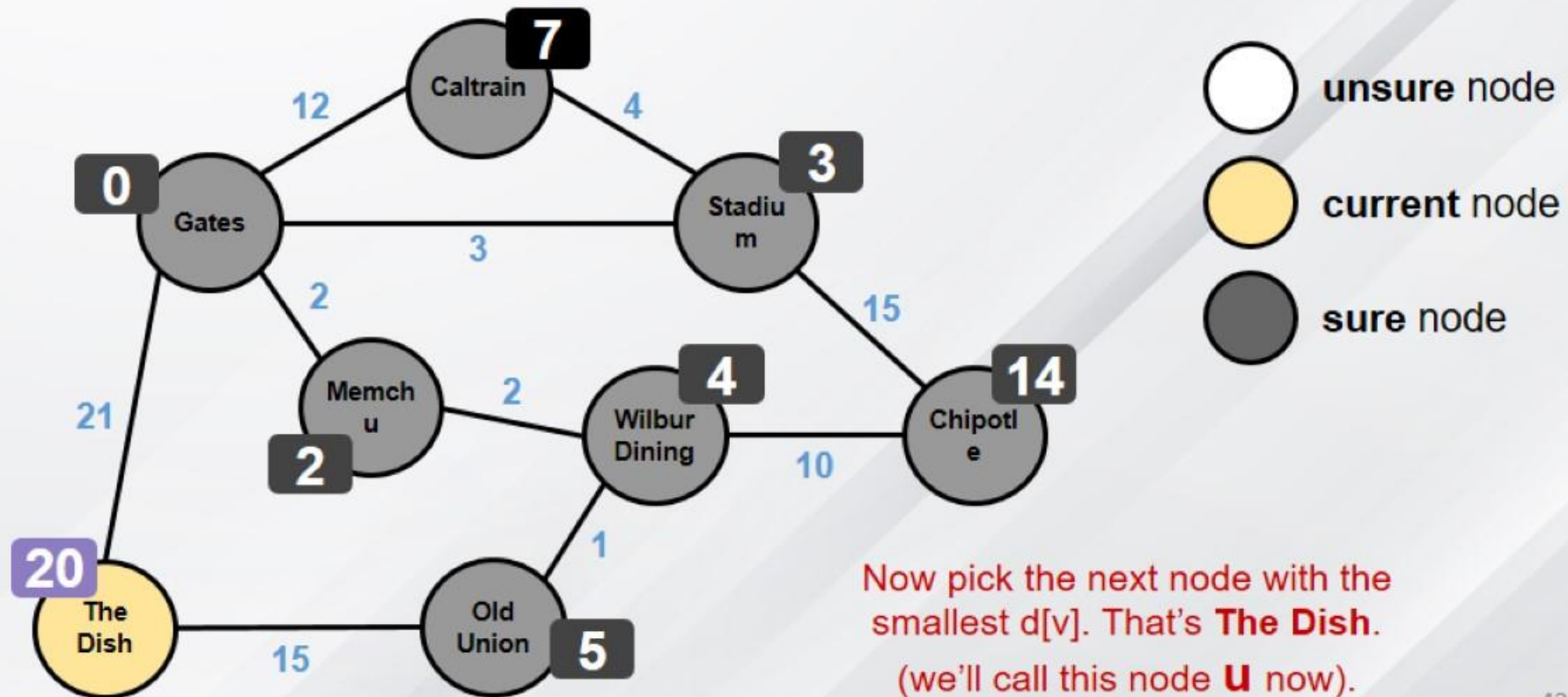
DIJKSTRA BY EXAMPLE



We'd update each of **u**'s *unsure* neighbors **v**,
but looks like **u** doesn't have any unsure
neighbors.
Just mark **u** as **sure**.

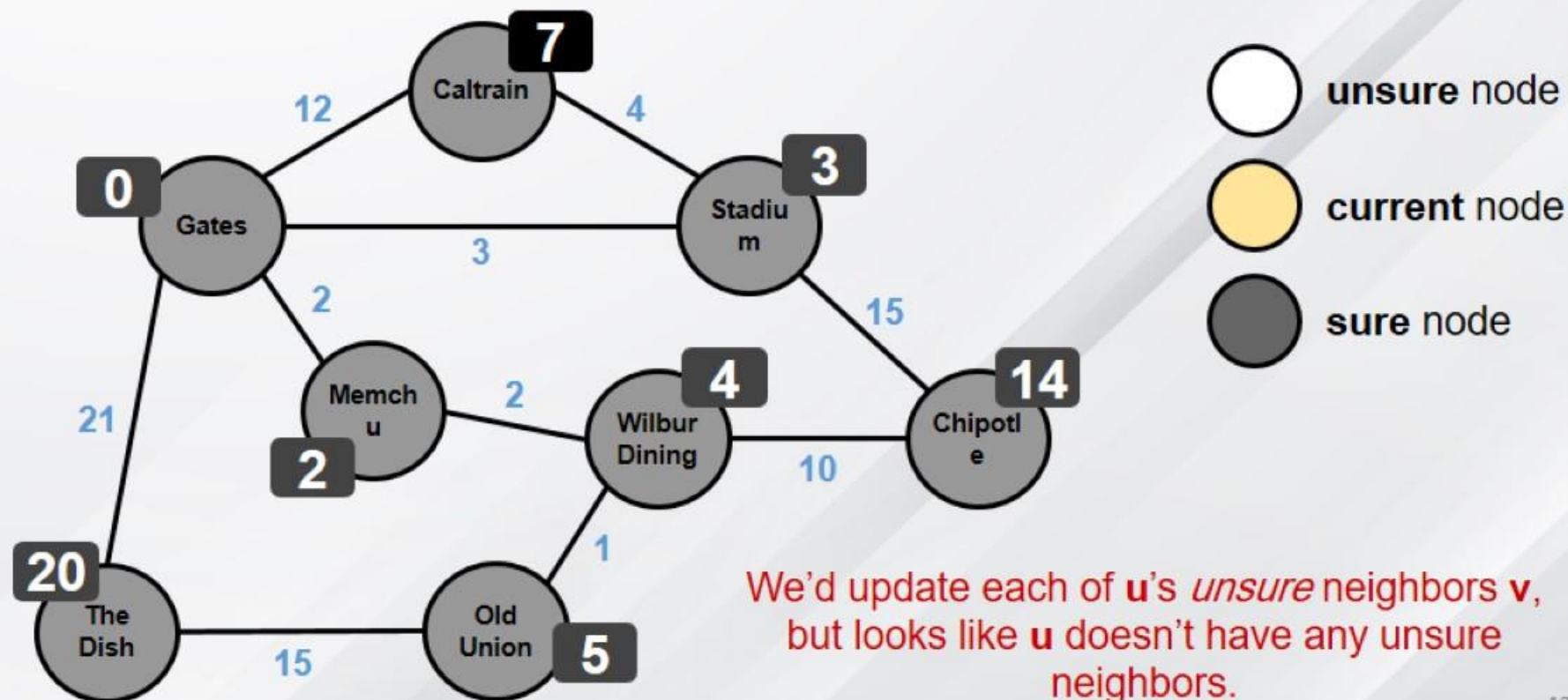
41

DIJKSTRA BY EXAMPLE



42

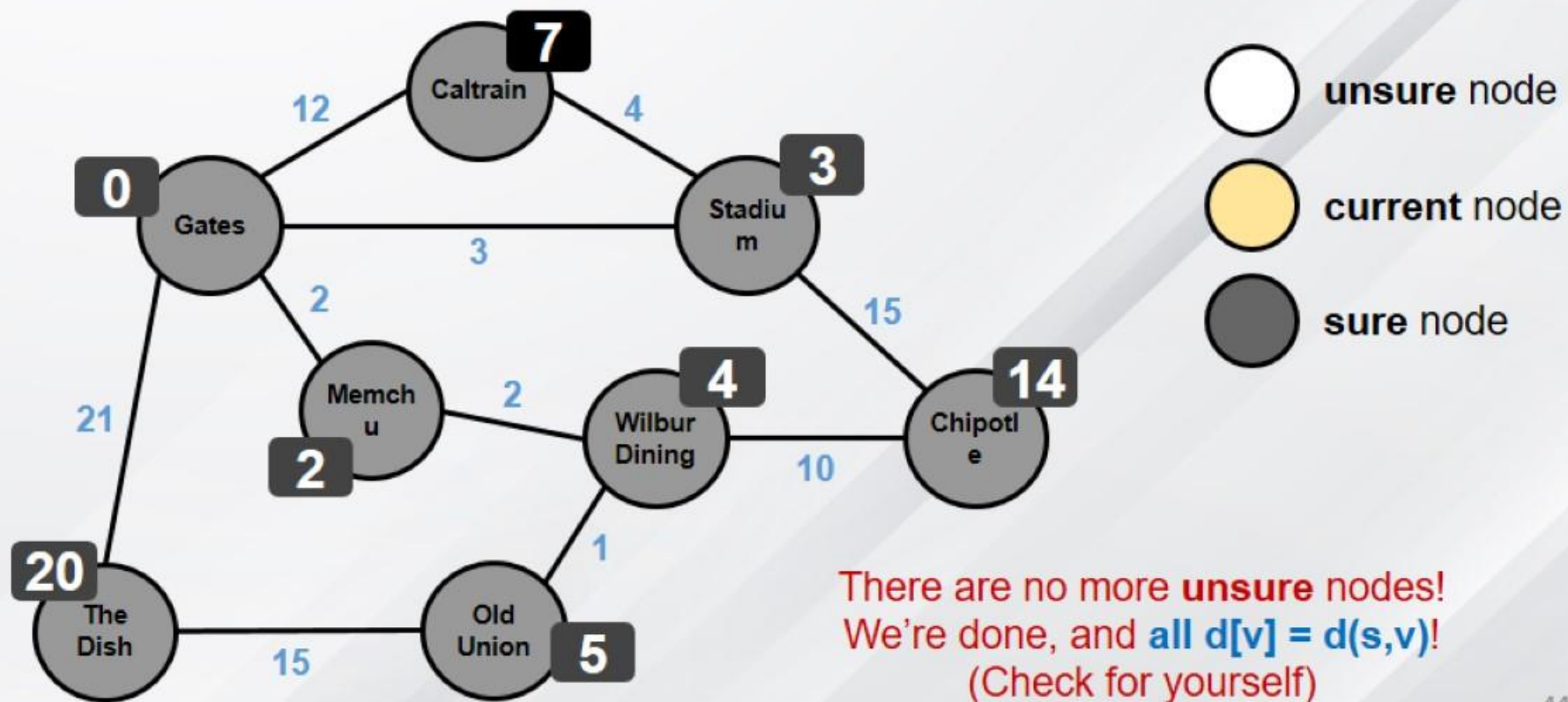
DIJKSTRA BY EXAMPLE



We'd update each of **u**'s *unsure* neighbors **v**,
but looks like **u** doesn't have any unsure
neighbors.
Just mark **u** as **sure**.

43

DIJKSTRA BY EXAMPLE



Chapter

04

DIJKSTRA PSUEDOCODE



DIJKSTRA PSUEDOCODE

DIJKSTRA(G,s):

Set all vertices to **not-sure**.

$d[v] = \infty$ for all v in V

$d[s] = 0$

while there are **not-sure** nodes:

 pick the **not-sure** node u with smallest estimate $d[u]$

 for v in u .neighbors:

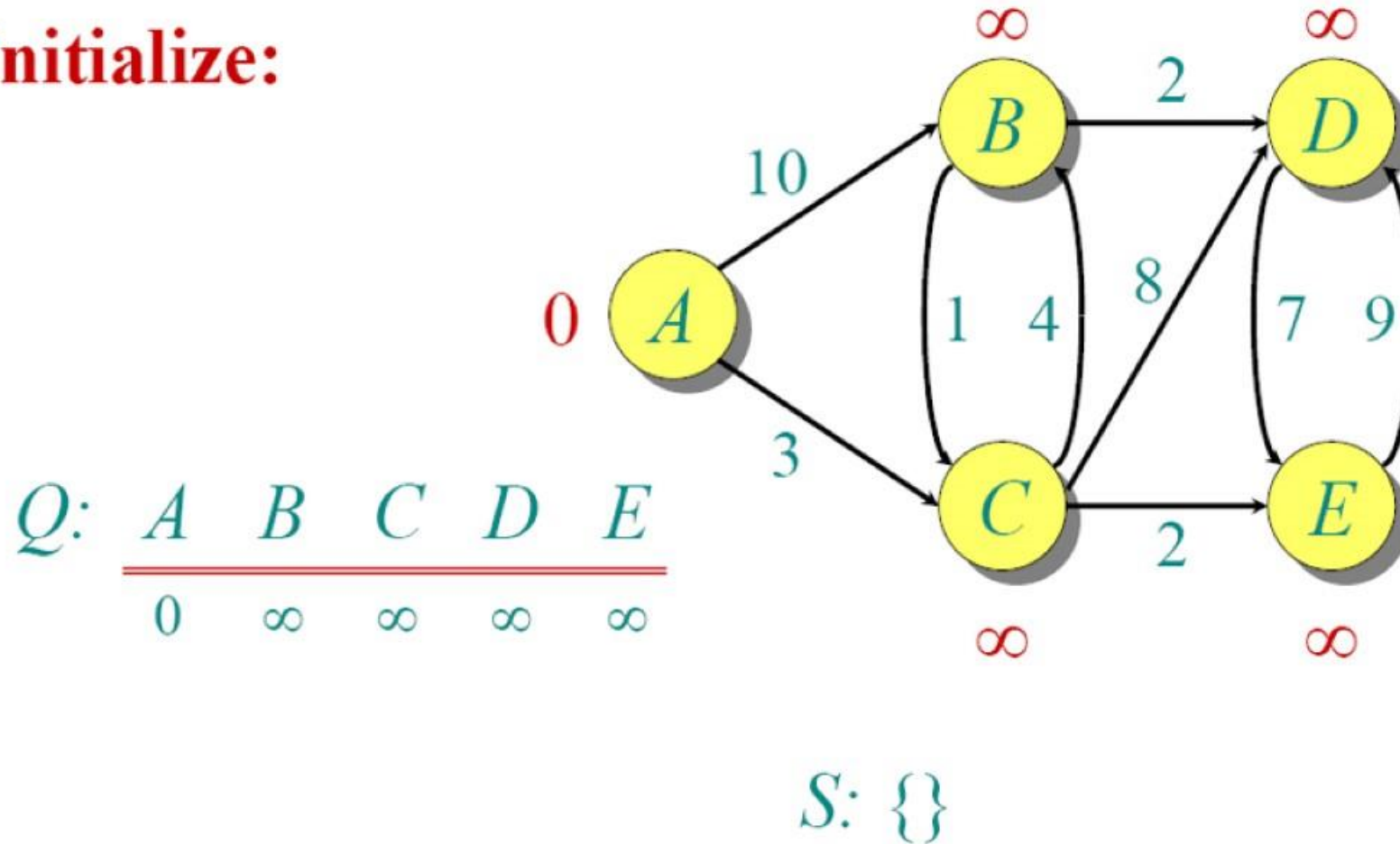
$d[v] \leftarrow \min(d[v], d[u] + w(u,v))$ // update step

 mark u as **sure**

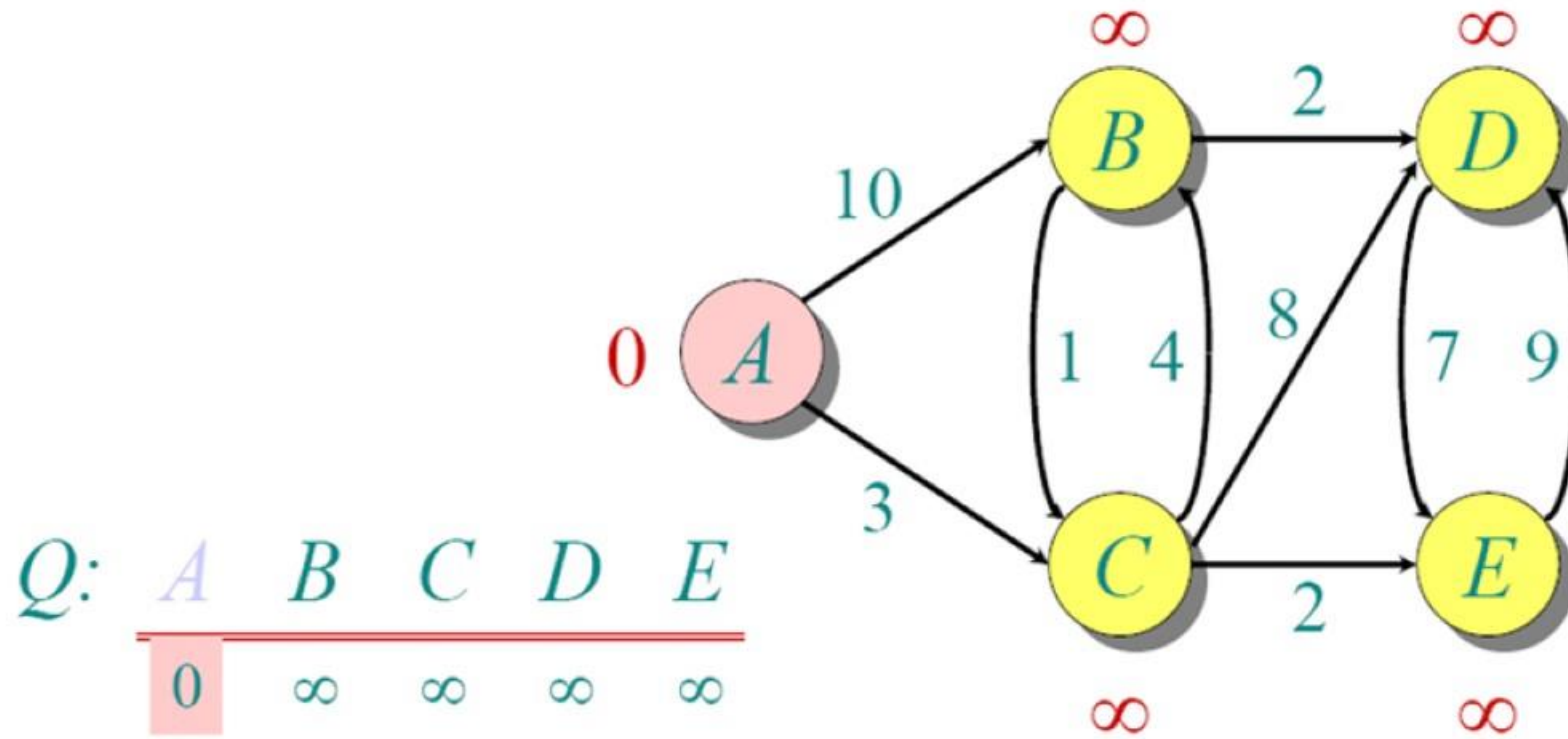
now all $d[v]$ indeed equals **the true distance** $d(s,v)$

Dijkstra Animated Example 1

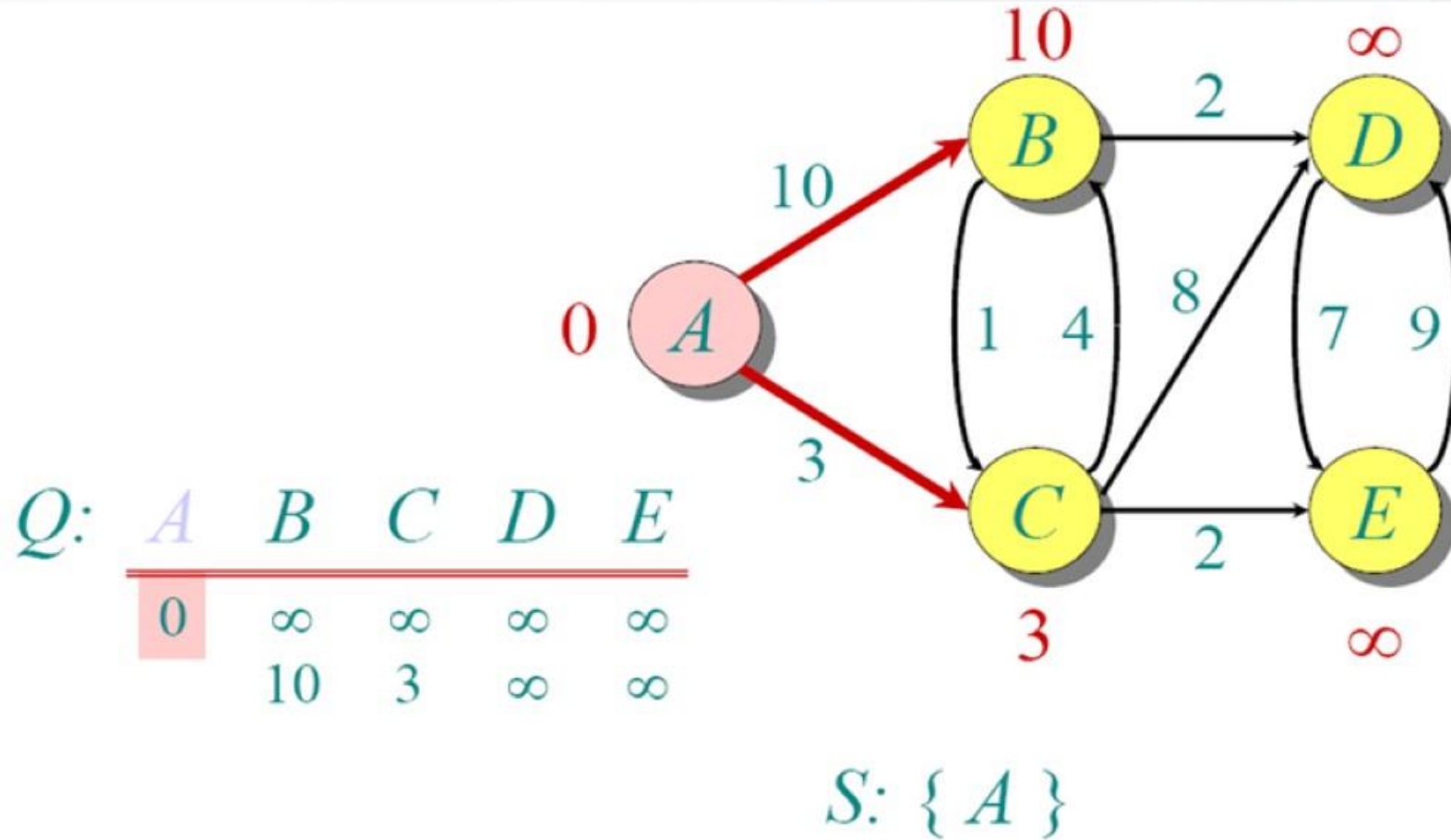
Initialize:



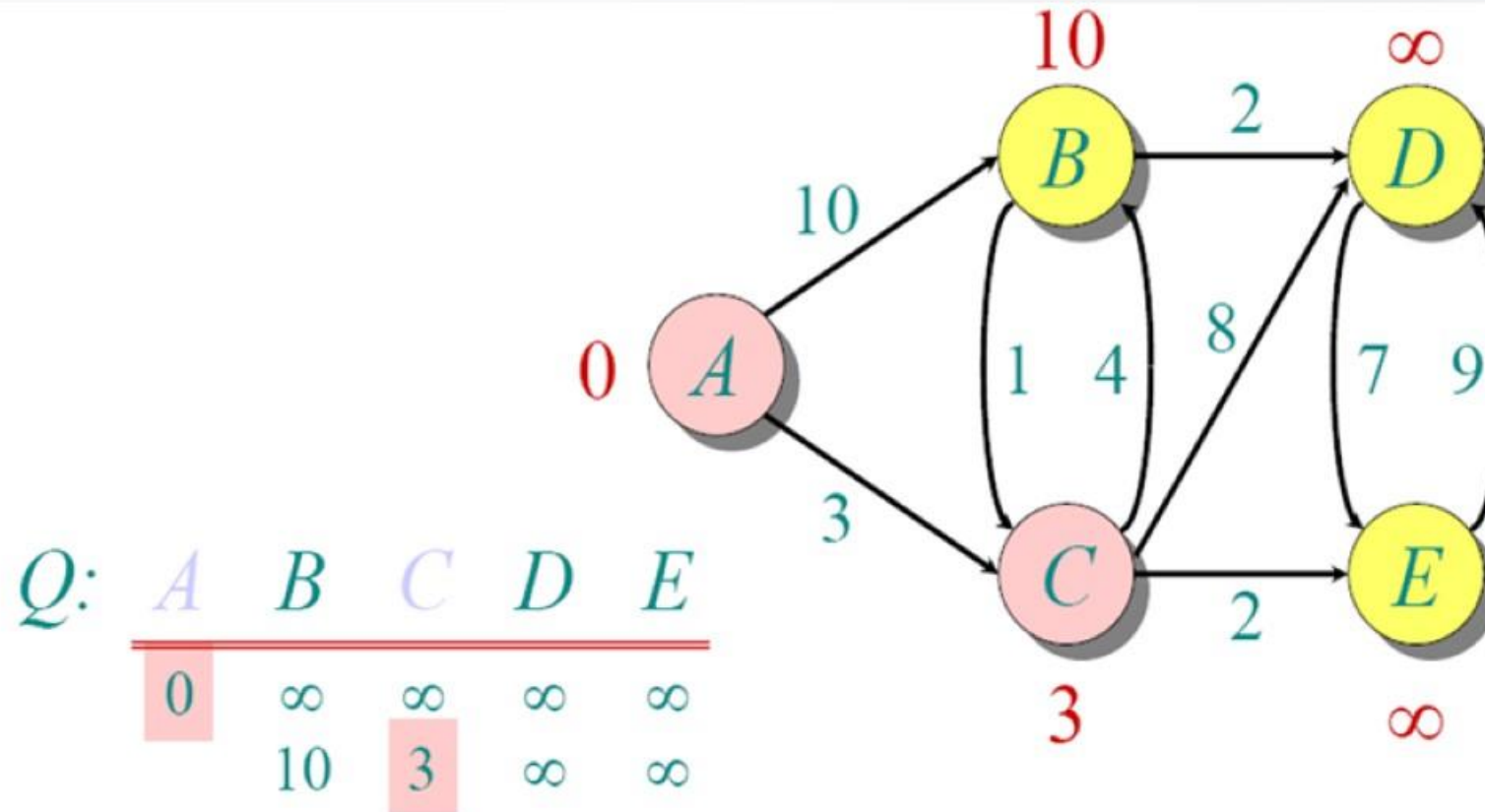
Dijkstra Animated Example 1



Dijkstra Animated Example 1

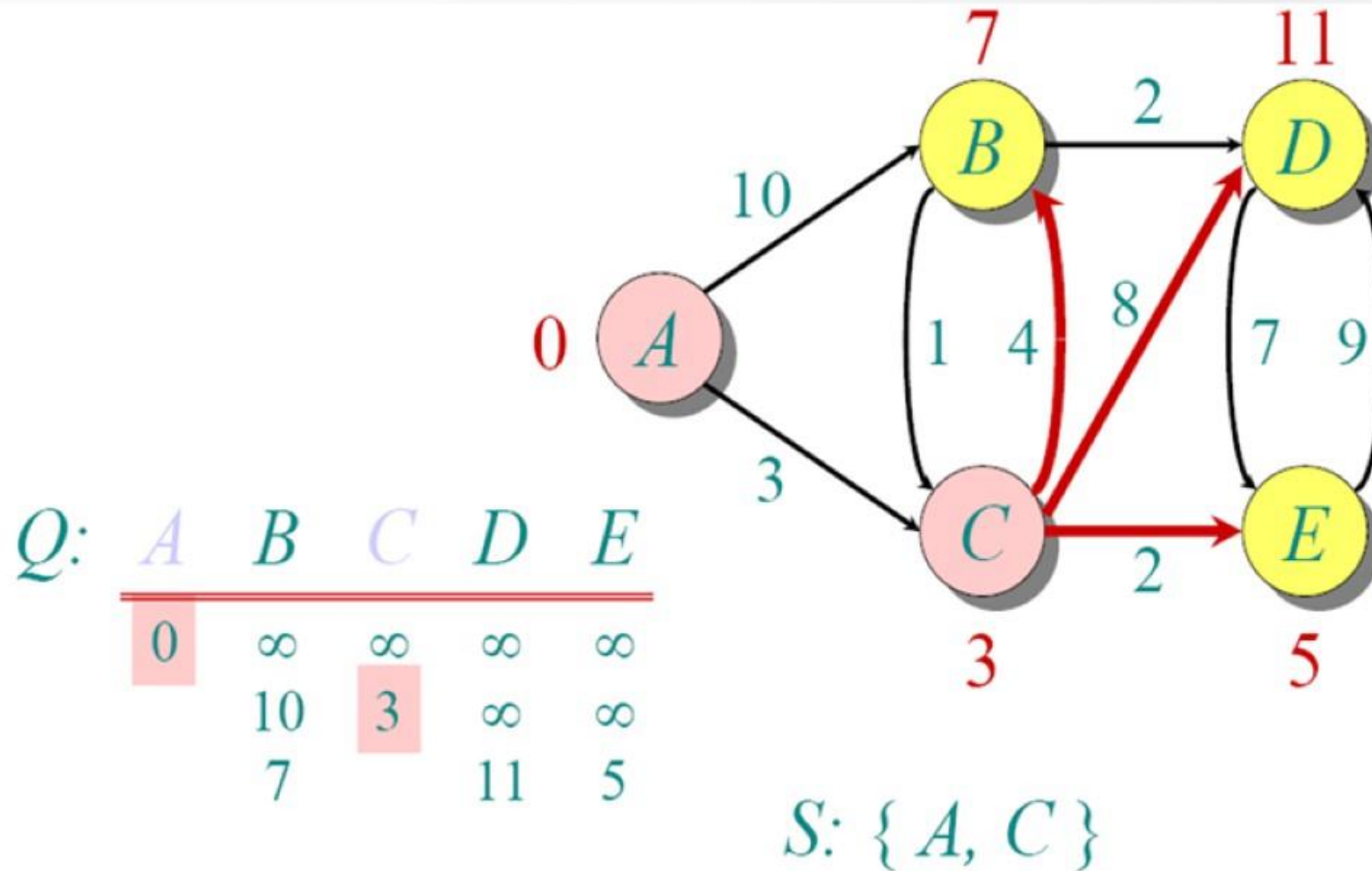


Dijkstra Animated Example 1

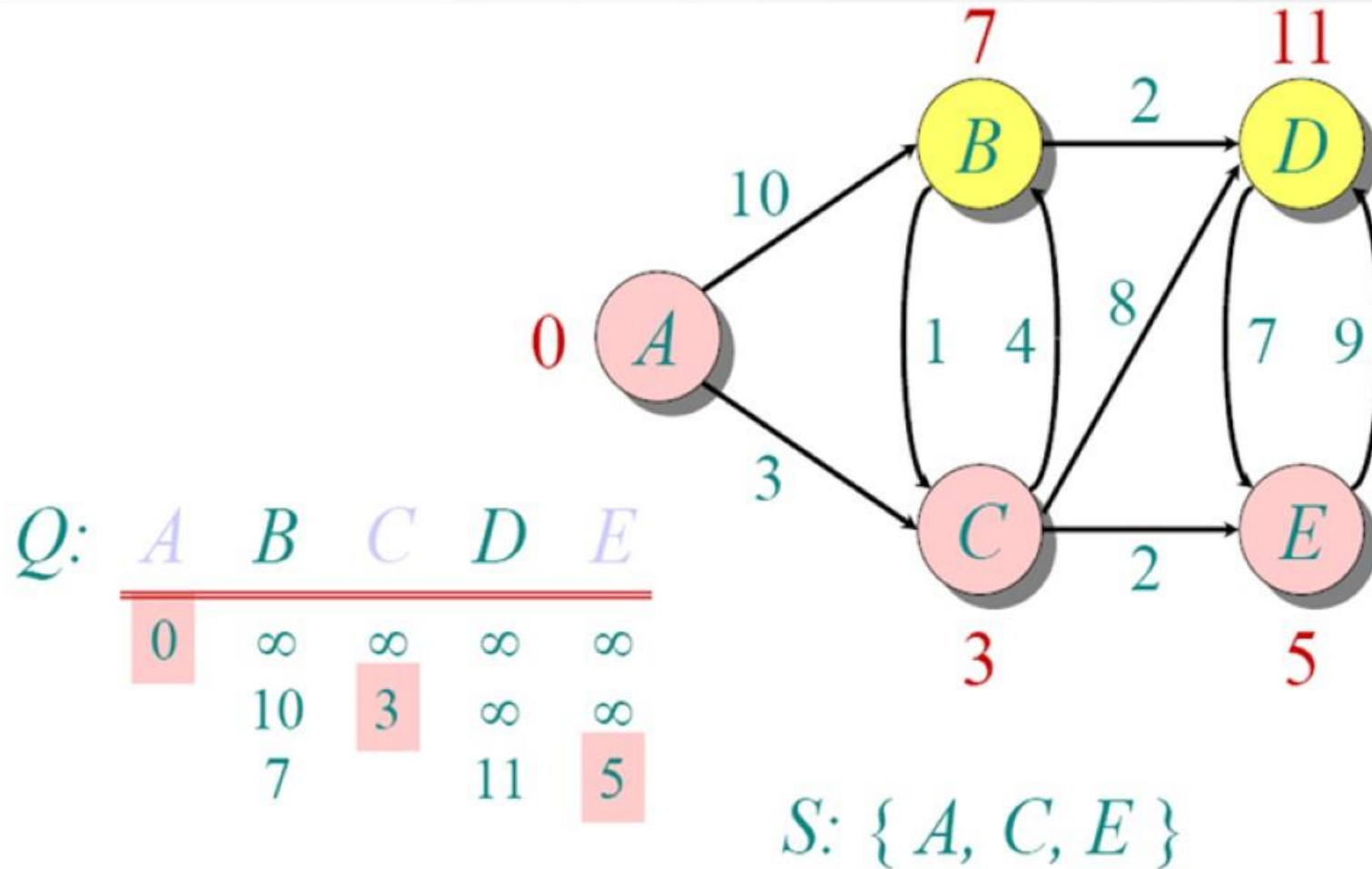


S: { A, C }

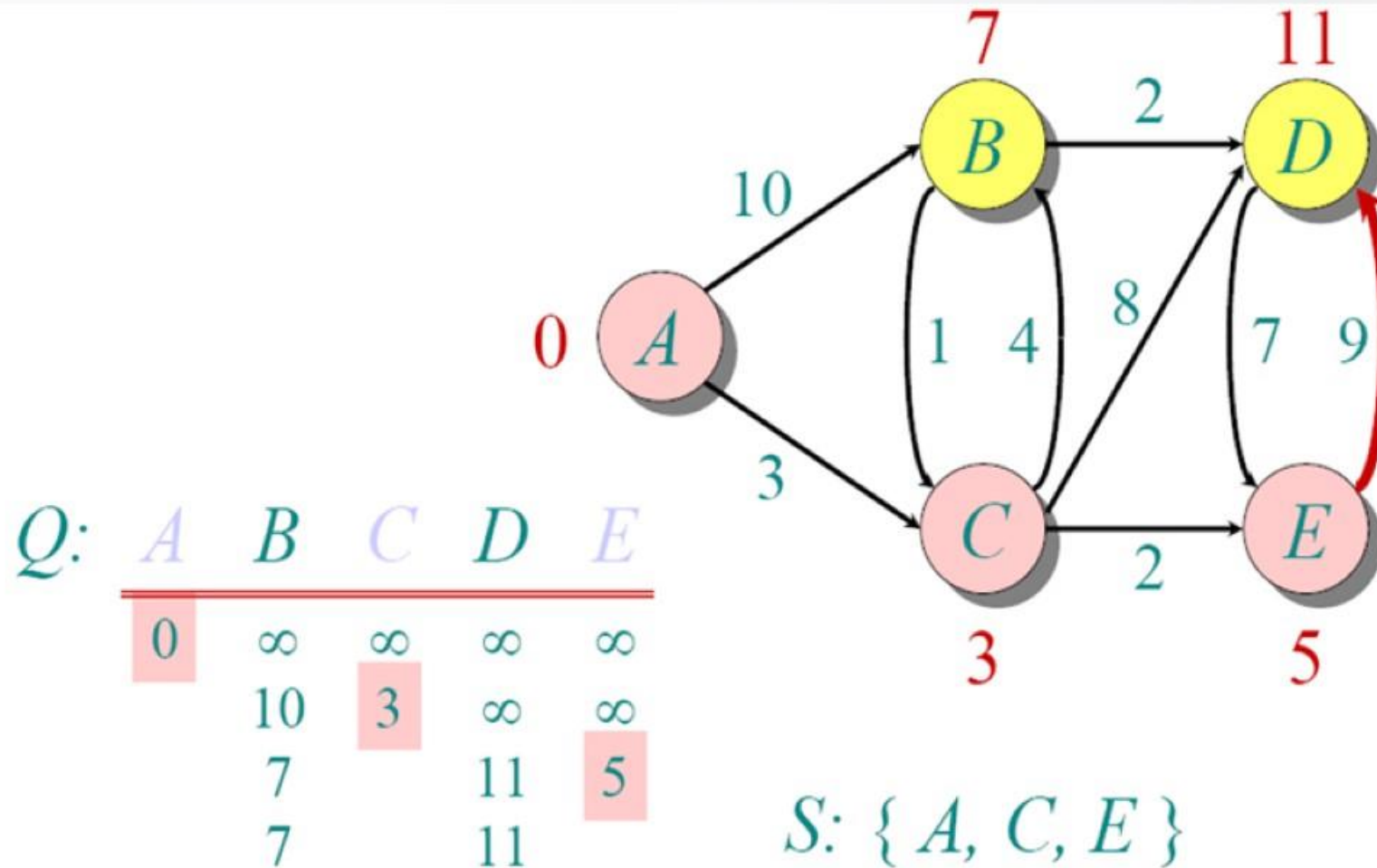
Dijkstra Animated Example 1



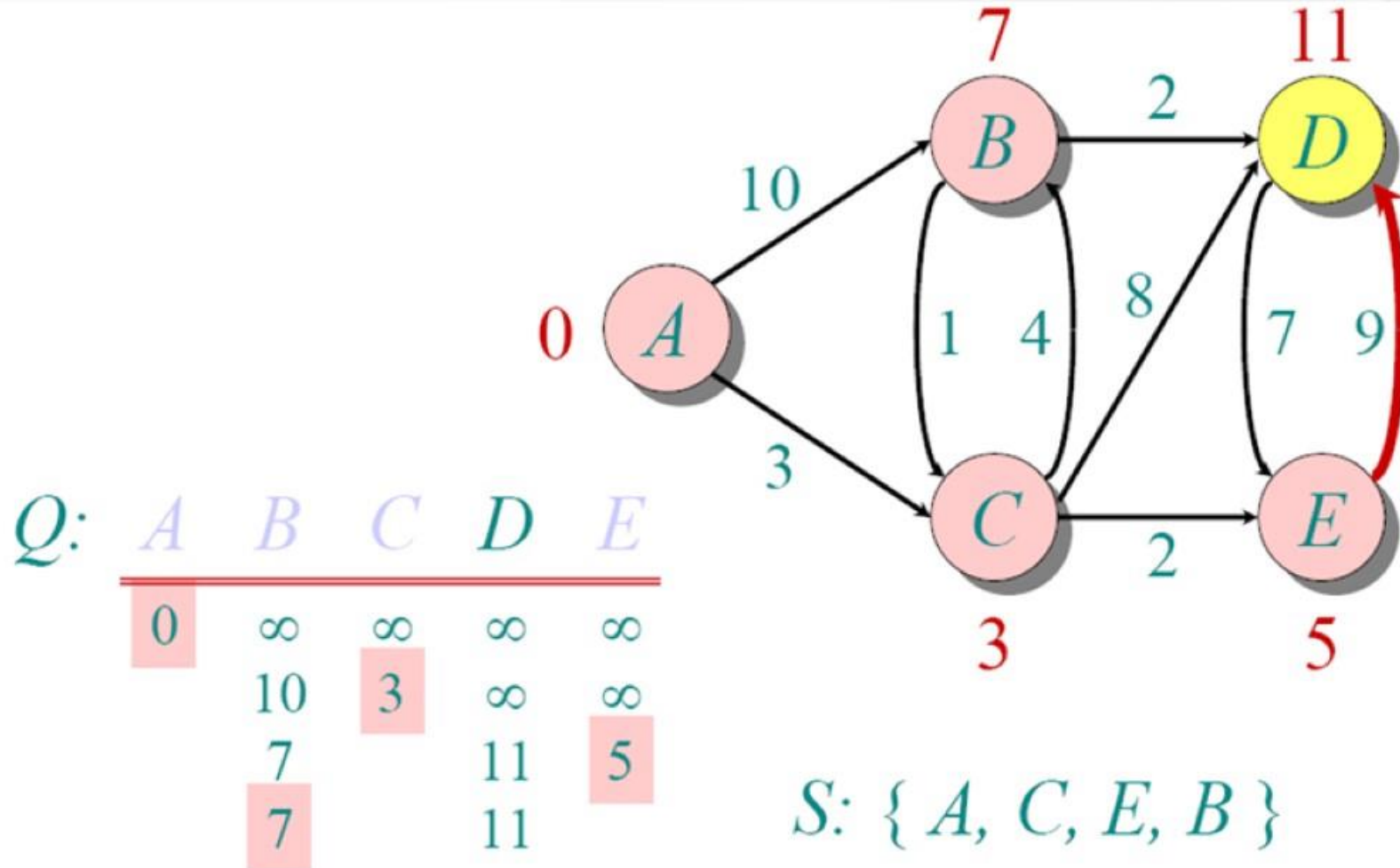
Dijkstra Animated Example 1



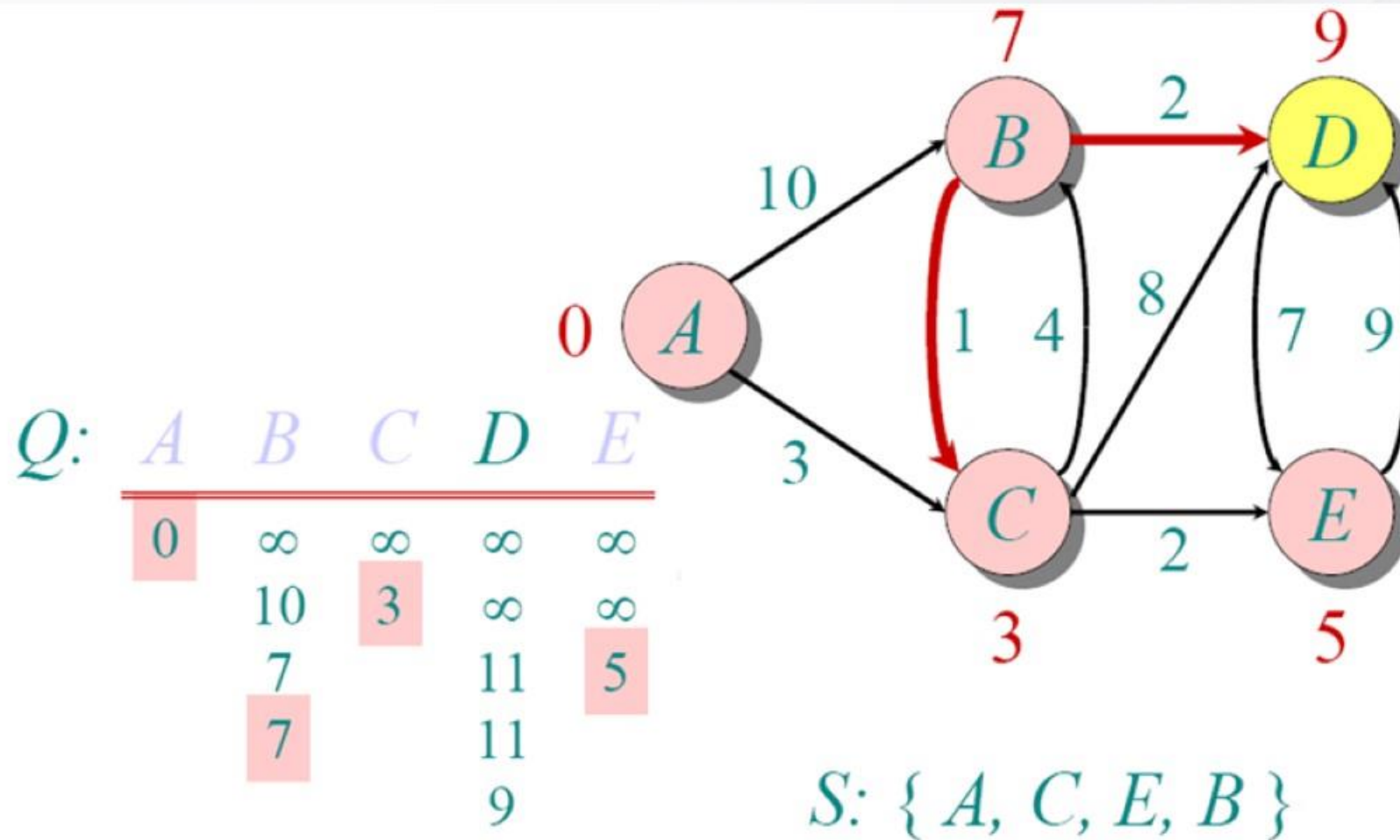
Dijkstra Animated Example 1



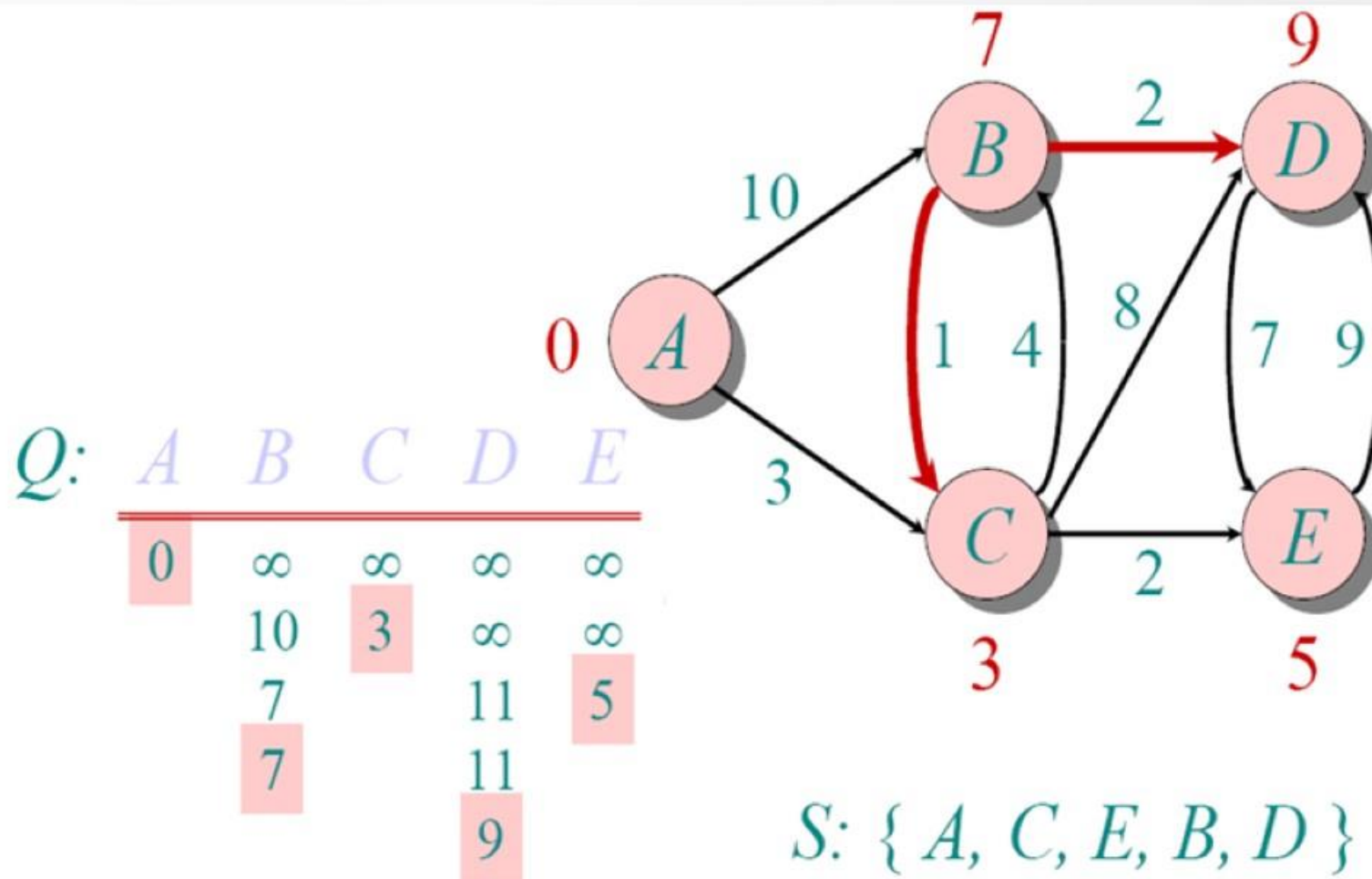
Dijkstra Animated Example 1



Dijkstra Animated Example 1



Dijkstra Animated Example 1

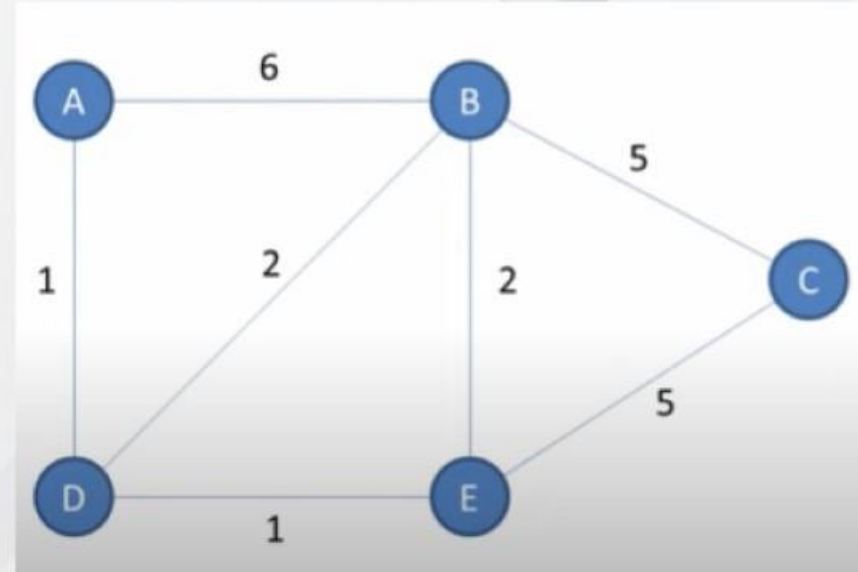


Vertex	Shortest distance from start	Previous vertex
A	0	
C	3	A
E	5	C
B	7	C
D	9	B

Dijkstra Example 2

- Find the shortest distance from A to each vertex in the graph
- Find the shortest path from A to each vertex in the graph

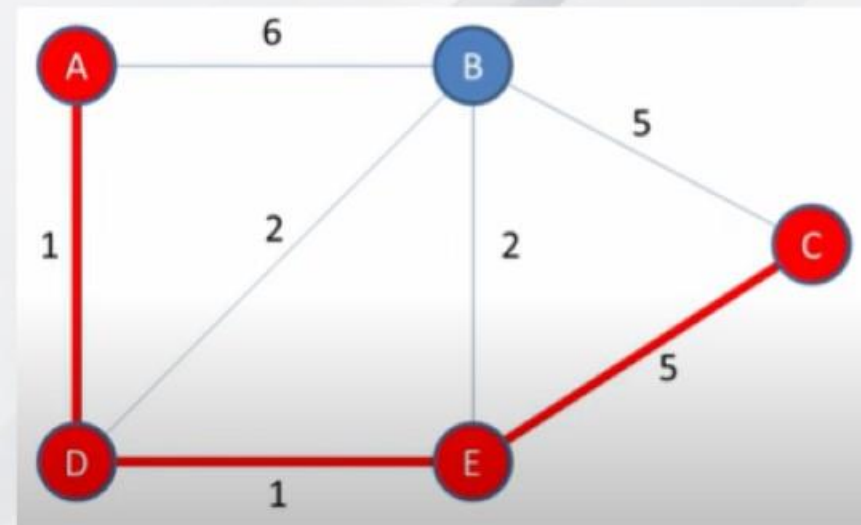
Vertex	Shortest distance from A	Previous vertex
A	0	
B	3	D
C	7	E
D	1	A
E	2	D



Dijkstra Example 2

- Ex: Find the shortest path from A to C

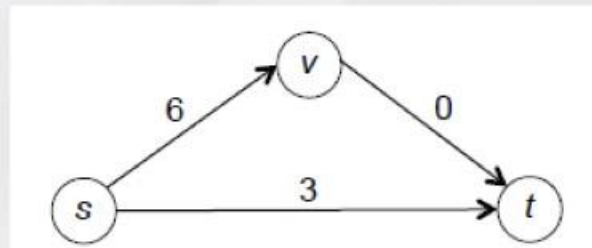
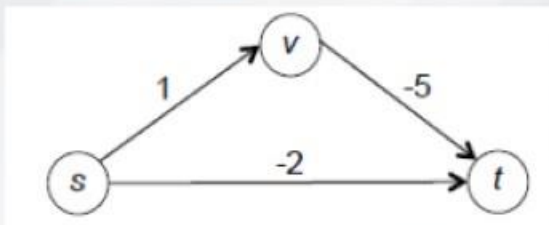
Vertex	Shortest distance from A	Previous vertex
A	0	
B	3	D
C	7	E
D	1	A
E	2	D



Path = A → D → E → C

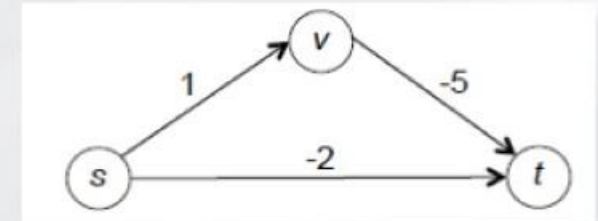
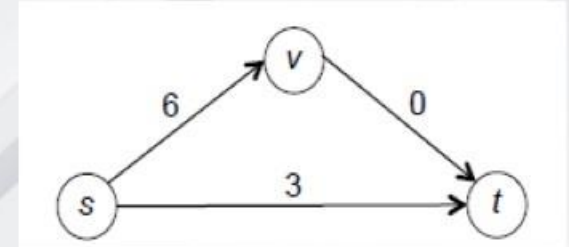
Dijkstra's algorithm

- You might be wondering why it matters whether or not edges have negative edge lengths. Can't we just force all the edge lengths to be nonnegative by adding a big number to every edge's length?
- The problem is that different paths from one vertex to another might not have the same number of edges.
- If we add some number to the length of each edge, then the lengths of different paths can increase by different amounts, and a shortest path in the new graph might be different than in the original graph.



Dijkstra's algorithm

- There are two paths from s to t : the direct path (which has length -2) and the two-hop path $s \rightarrow v \rightarrow t$ (which has length $1 + (-5) = -4$).
- The latter has the smaller (that is, more negative) length, and is the shortest s - t path.
- To force the graph to have nonnegative edge lengths, we could add 5 to every edge's length:
- The shortest path from s to t has switched, and is now the direct s - t edge (which has length 3, better than the alternative of 6).
- Running a shortest-path algorithm on the **transformed graph would not produce a correct answer for the original graph.**



Exercise 2:

- Which of the following running times best describes a straightforward implementation of Dijkstra's algorithm for graphs in adjacency-list representation? As usual, n and m denote the number of vertices and edges, respectively, of the input graph.

a) $O(m + n)$

b) $O(m \log n)$

c) $O(n^2)$

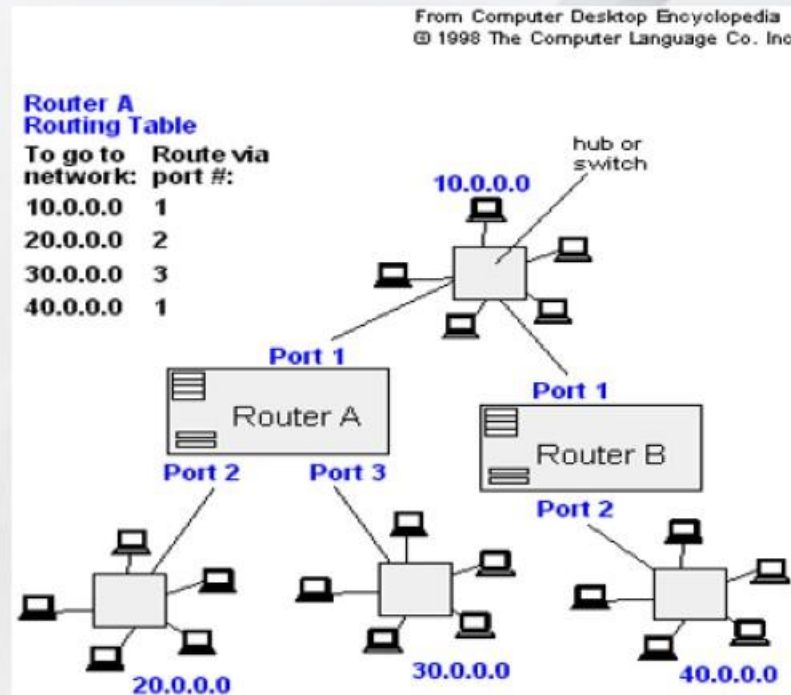
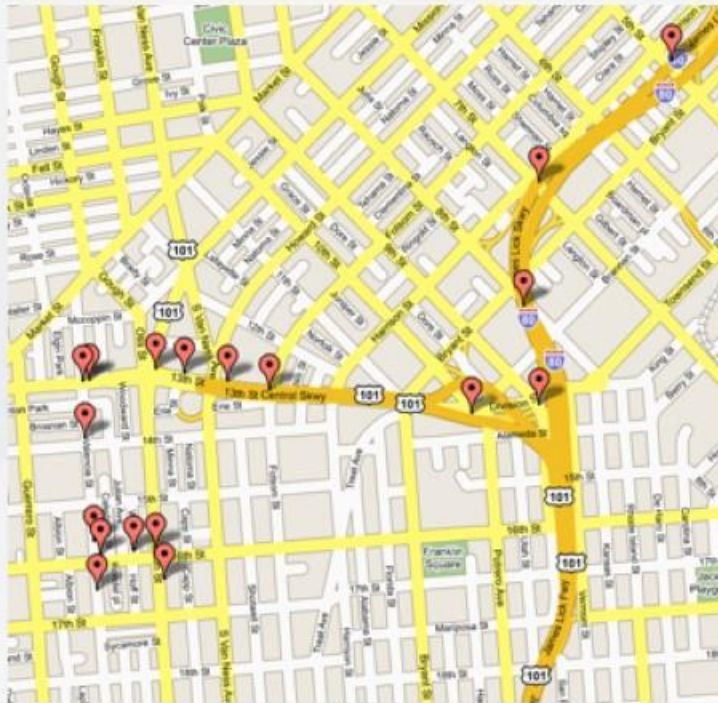
d) $O(mn)$

Dijkstra's algorithm running time

- The number of iterations is $O(n)$ and each takes time $O(m)$.
- The overall running time is $O(mn)$.

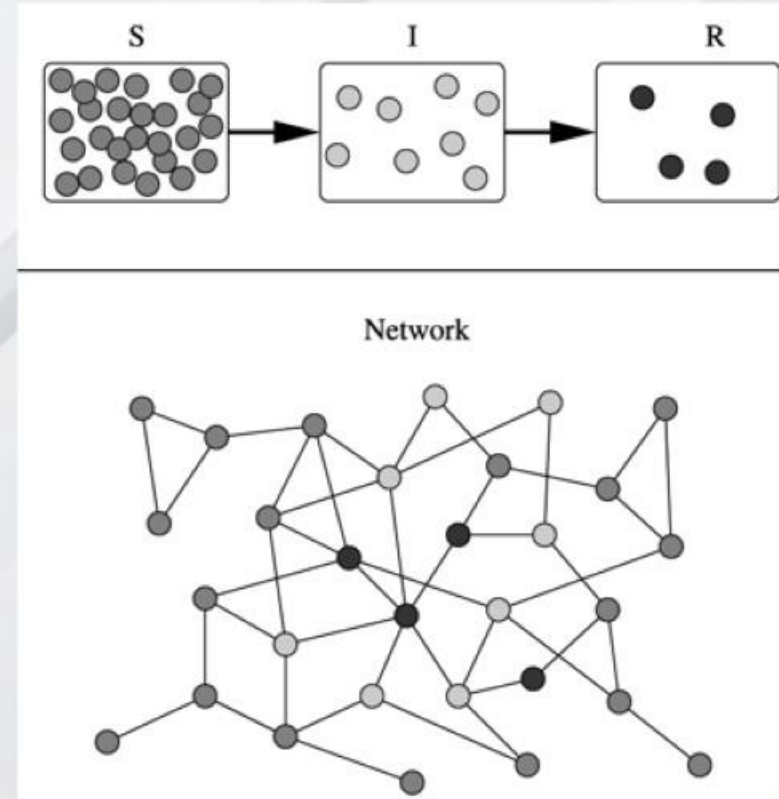
Applications of Dijkstra's Algorithm

- Traffic Information Systems are most prominent use
- Mapping (Map Quest, Google Maps)
- Routing Systems



Applications of Dijkstra's Algorithm

- One particularly relevant this : epidemiology
- Prof. Lauren Meyers (Biology Dept.) **uses networks to model the spread of infectious diseases** and **design prevention and response strategies**.
- **Vertices represent individuals**, and **edges their possible contacts**. It is **useful to calculate how a particular individual is connected to others**.
- **Knowing the shortest path lengths to other individuals** can be a relevant **indicator of the potential of a particular individual to infect others**.



Latest Updates, Innovations, and Real-World Applications of Data Structures and Algorithms



1. Introduction

- Focuses on the **latest updates, innovations, and real-world applications** of Data Structures and Algorithms.
- As systems grow more complex, **Data Structures and Algorithms (DSA)** becomes **the hidden engine that powers scalability, speed, and intelligent decision-making in modern technology**—from AI to cloud computing.

2. Recent Developments in Data Structures

A. Advanced Data Structures for Big Data

1. Bloom Filters & Variants

- o Used in databases, web caching, and distributed systems (e.g., Google Bigtable).
- o Provide extremely fast membership checks with minimal memory usage.

2. LSM Trees (Log-Structured Merge Trees)

- o Used in modern storage systems such as Cassandra, LevelDB, and RocksDB.
- o Optimized for write-heavy workloads.

3. Trie-based Structures

- o Power autocomplete, spell-checking, and IP routing.

2. Recent Developments in Data Structures

B. Graph-Based Data Structures in Modern Systems

- **Social networks, recommendation engines, and navigation apps** rely heavily on graph databases (Neo4j, JanusGraph).
- **Graph structures** allow modeling complex relationships used in:
 - o Fraud detection
 - o Route optimization
 - o Knowledge graphs (Google Search).

3. Recent Developments in Algorithms

A. Algorithms in Artificial Intelligence

1. Deep Learning Optimization Algorithms

- o Improved versions of SGD (AdamW, RMSprop) accelerate AI model training.
- o Fast matrix multiplication algorithms speed up neural networks.

2. Search Algorithms in AI

A* variants, Monte Carlo Tree Search (MCTS), and heuristic search power:

- robotics
- autonomous vehicles
- game AI (e.g., AlphaGo)

3. Recent Developments in Algorithms

B. Algorithms for Large-Scale Data Processing

- **MapReduce algorithms** streamline distributed computing.
- **Streaming algorithms** (e.g., Count-Min Sketch) allow real-time processing of huge data streams such as social media or network traffic.

C. Quantum-Inspired Algorithms

- **Research on quantum search** and approximate optimization algorithms is growing.
- Although not fully practical yet, **they inspire faster algorithmic solutions.**

4. Modern Applications of DSA in Real Life

1. Cloud Computing & Distributed Systems

- Load balancing, resource allocation, container scheduling (Kubernetes) all depend on optimized algorithms.
- Hashing, trees, and graphs play a major role in making cloud services scalable.

2. Cybersecurity

- Cryptographic algorithms and hash functions secure data transmission.
- Intrusion detection systems use pattern-matching and graph algorithms.

3. Search Engines & Recommendation Systems

Inverted indexes, tries, PageRank, and collaborative filtering algorithms enable fast, relevant search and recommendations on:

- o Google
- o YouTube
- o Netflix
- o Amazon

4. Autonomous Vehicles

- Path-finding algorithms (A^* , Dijkstra) and sensor fusion data structures allow cars to navigate safely.

5. Healthcare & Bioinformatics

- Genome sequencing relies on suffix trees, dynamic programming, and hashing to analyze genetic data quickly.

Skills Needed for the Future



Skills Needed for the Future:

- Algorithmic thinking
- Understanding trade-offs: time vs. space
- Ability to work with large-scale data
- Familiarity with AI-related algorithmic tools
- Continuous learning of new data structures

Conclusion:

- Data Structures and Algorithms remain at **the heart of every modern technology.**
- From AI to cloud computing and cybersecurity, **today's innovations depend on efficient structures and optimized algorithms.**
- Mastering DSA empowers you not only to understand these systems but also to **create the next generation of technological solutions.**



Thank You