Chapter 4
Data Structures

Algorithm Theory
WS 2014/15

Fabian Kuhn

Examples

Dictionary:

- Operations: insert(*key,value*), delete(*key*), find(*key*)
- Implementations:
  - Linked list: all operations take $O(n)$ time (n: size of data structure)
  - Balanced binary tree: all operations take $O(\log n)$ time
  - Hash table: all operations take $O(1)$ times (with some assumptions)

Stack (LIFO Queue):

- Operations: push, pull
- Linked list: $O(1)$ for both operations

(FIFO) Queue:

- Operations: enqueue, dequeue
- Linked list: $O(1)$ time for both operations

Here: **Priority Queues (heaps)**, **Union-Find data structure**

Dijkstra's Algorithm

Single-Source Shortest Path Problem:

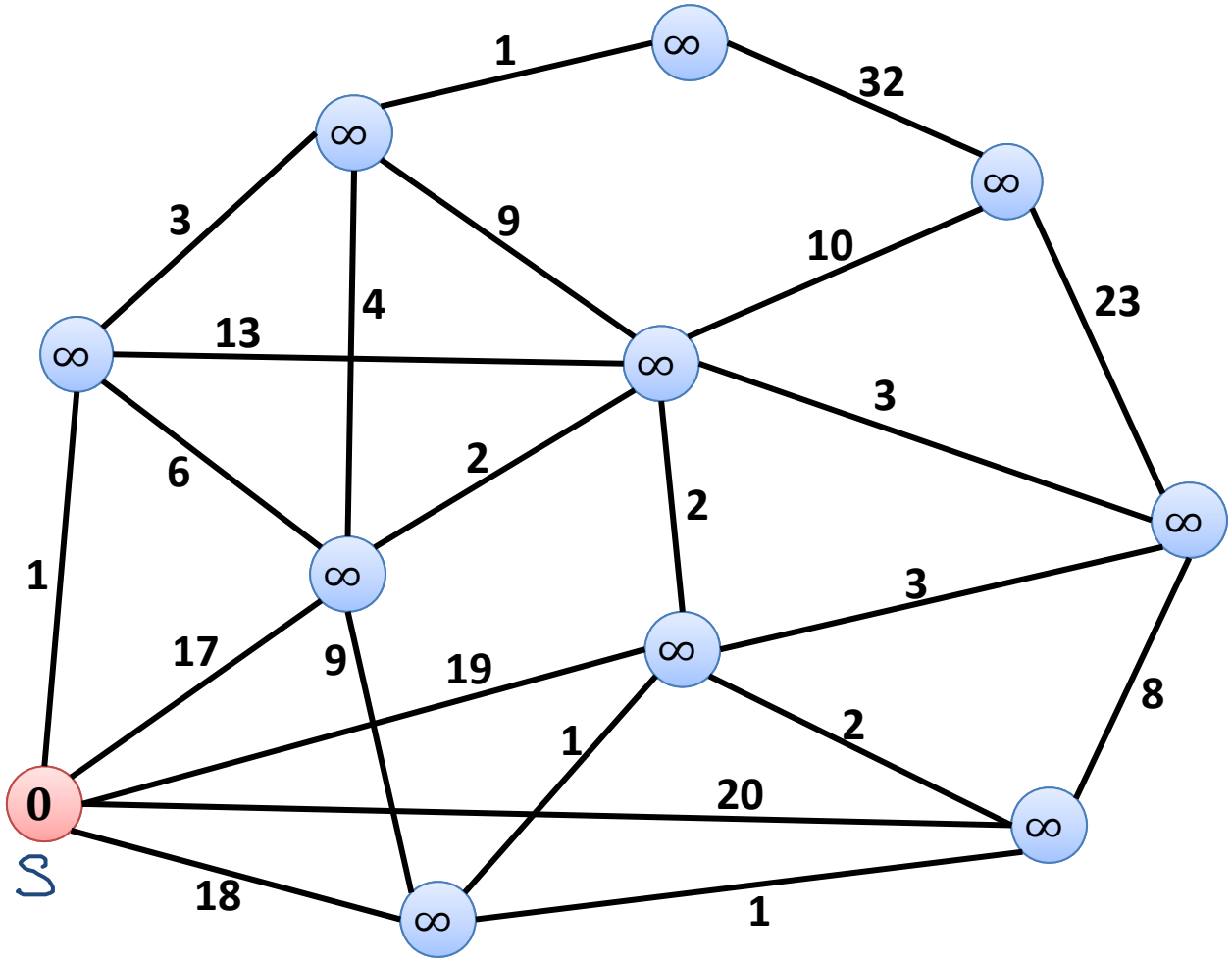
- **Given:** graph $G = (V, E)$ with edge weights $w(e) \geq 0$ for $e \in E$
  source node $s \in V$
- **Goal:** compute shortest paths from s to all $v \in V$

Dijkstra's Algorithm:

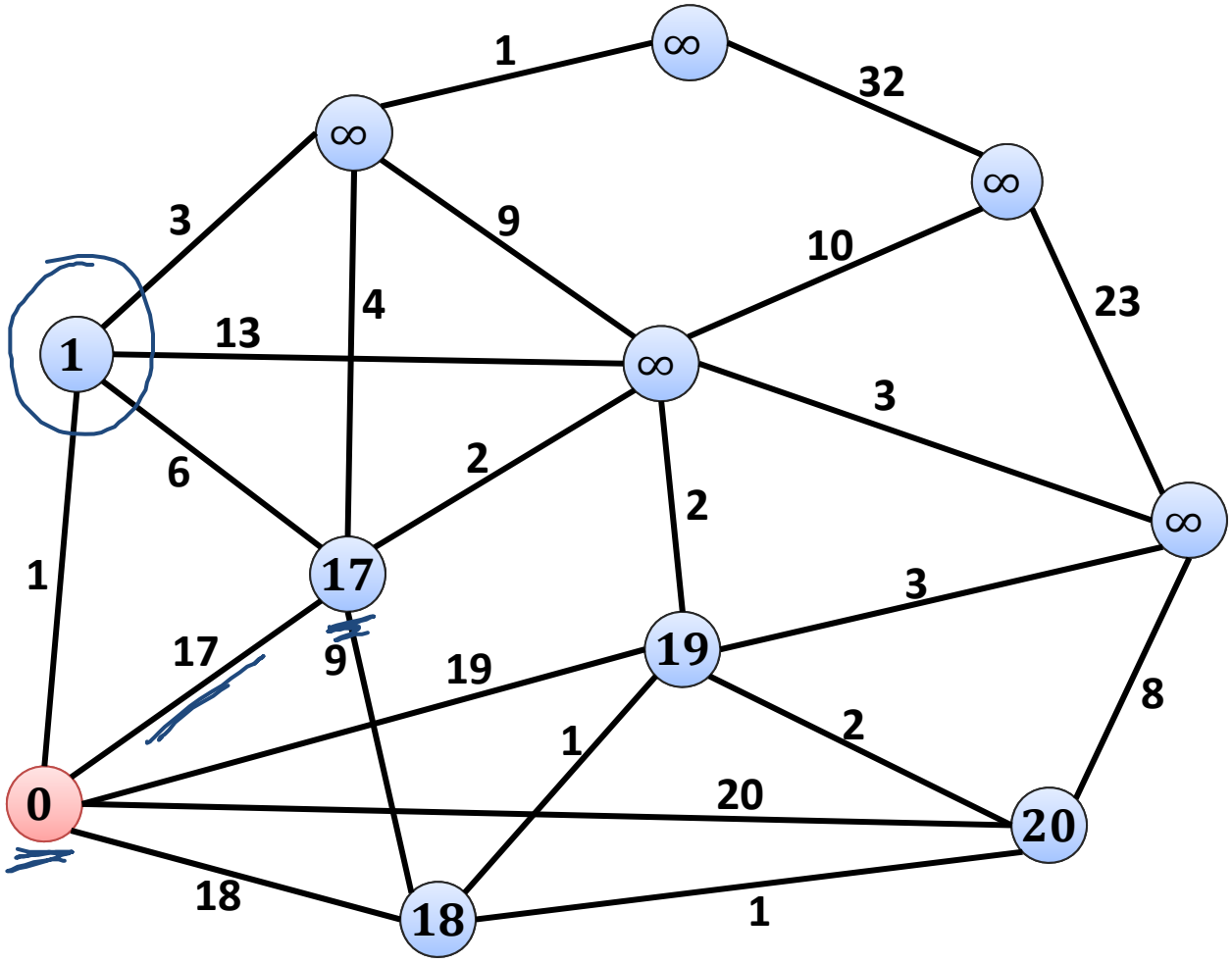
1. Initialize $d(s,s) = 0$ and $d(s,v) = \infty$ for all $v \neq s$
2. All nodes are unmarked
3. Get unmarked node u which minimizes $d(s,u)$:
4. For all $e = \{u,v\} \in E$, $d(s,v) = \min\{d(s,v), d(s,u) + w(e)\}$
5. mark node u
6. Until all nodes are marked

can be seen as a dynamic programming algorithm

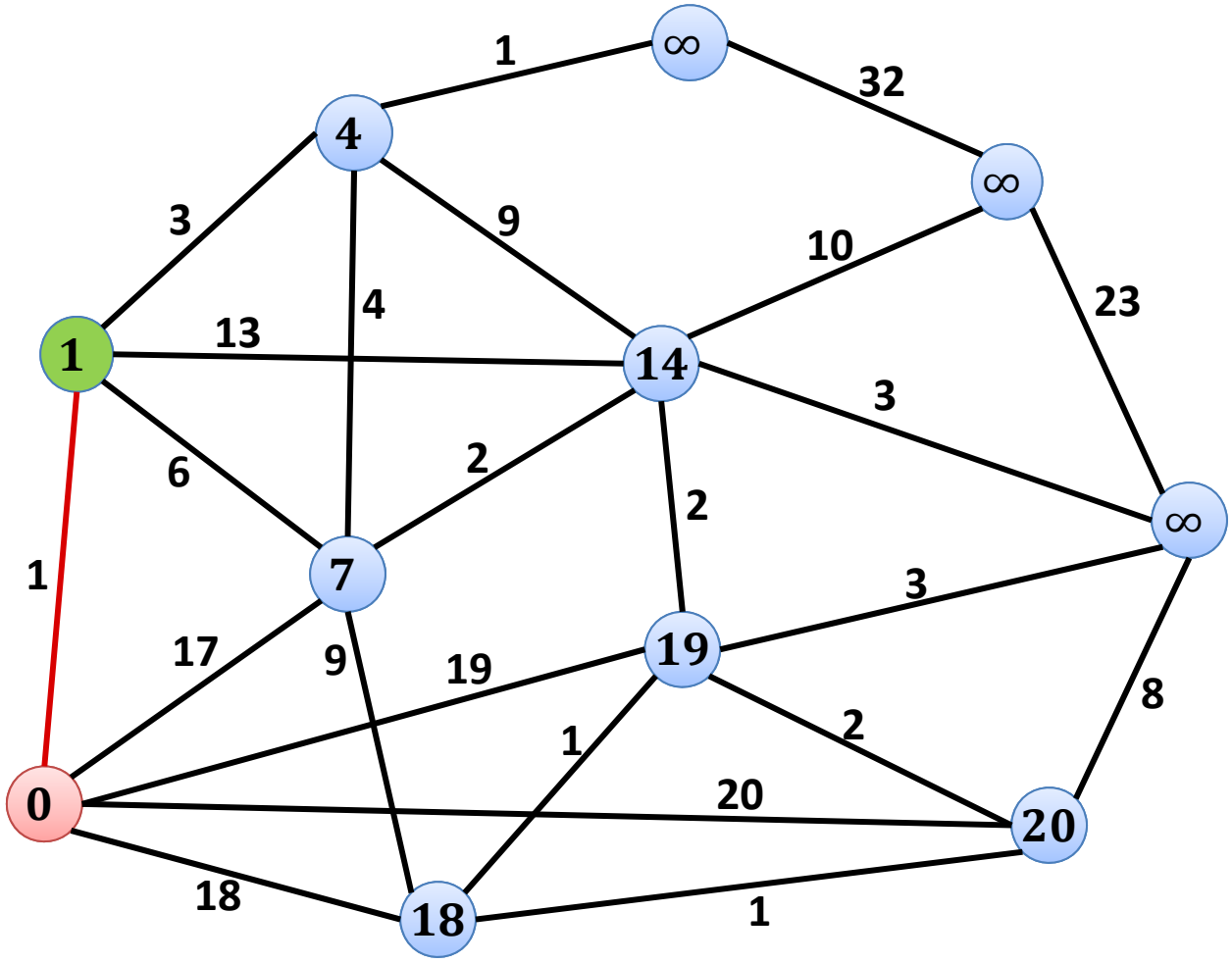
Example



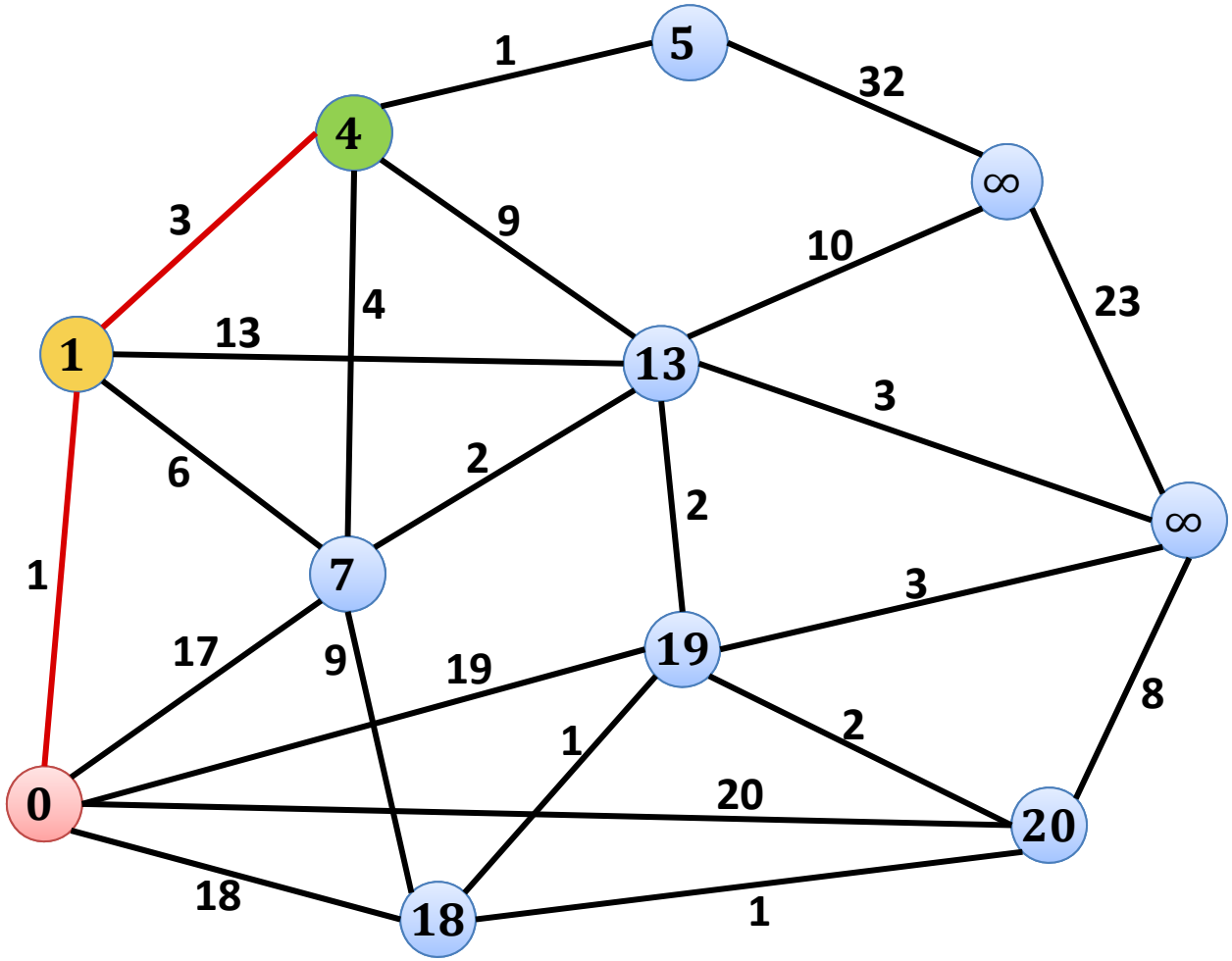
Example



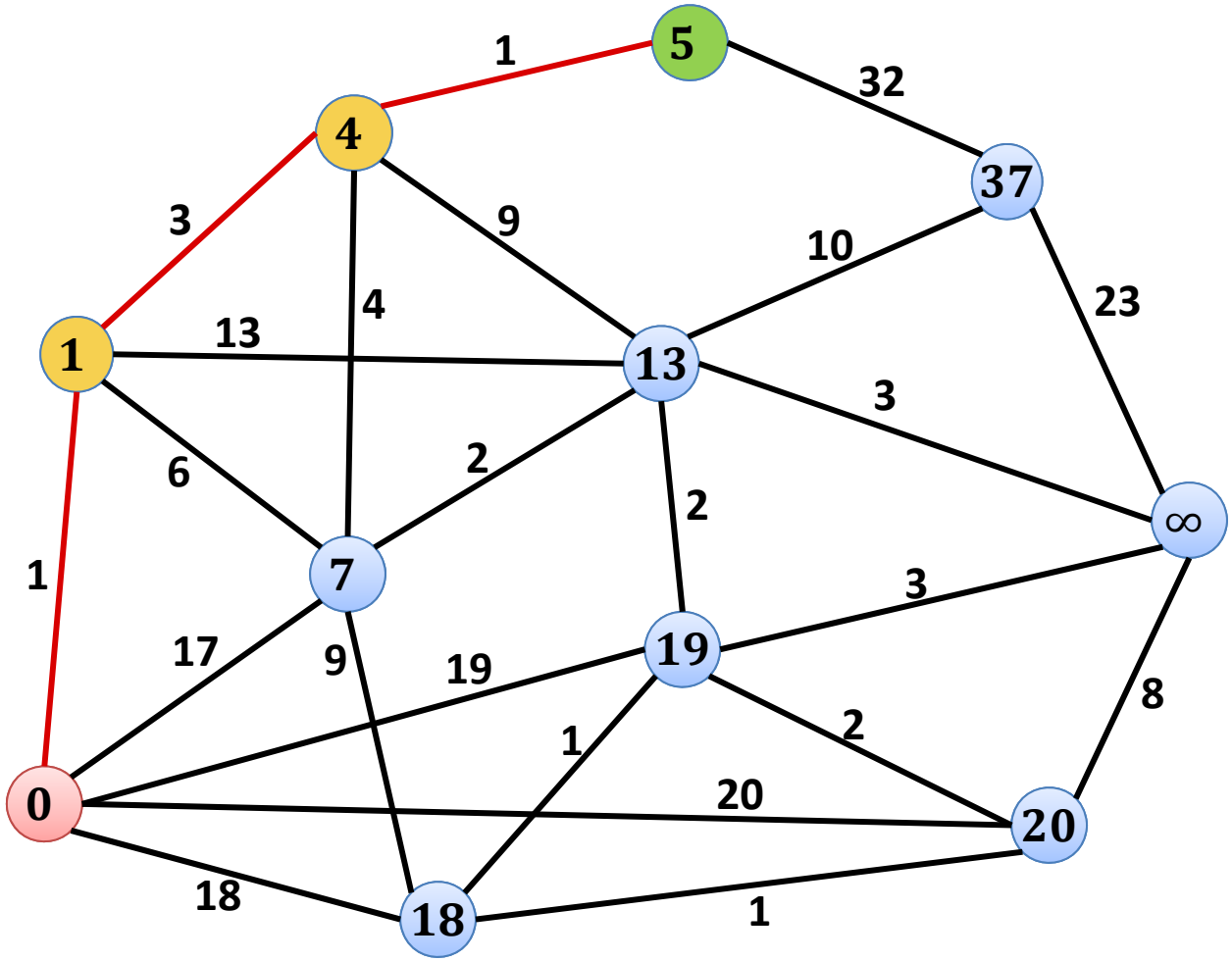
Example



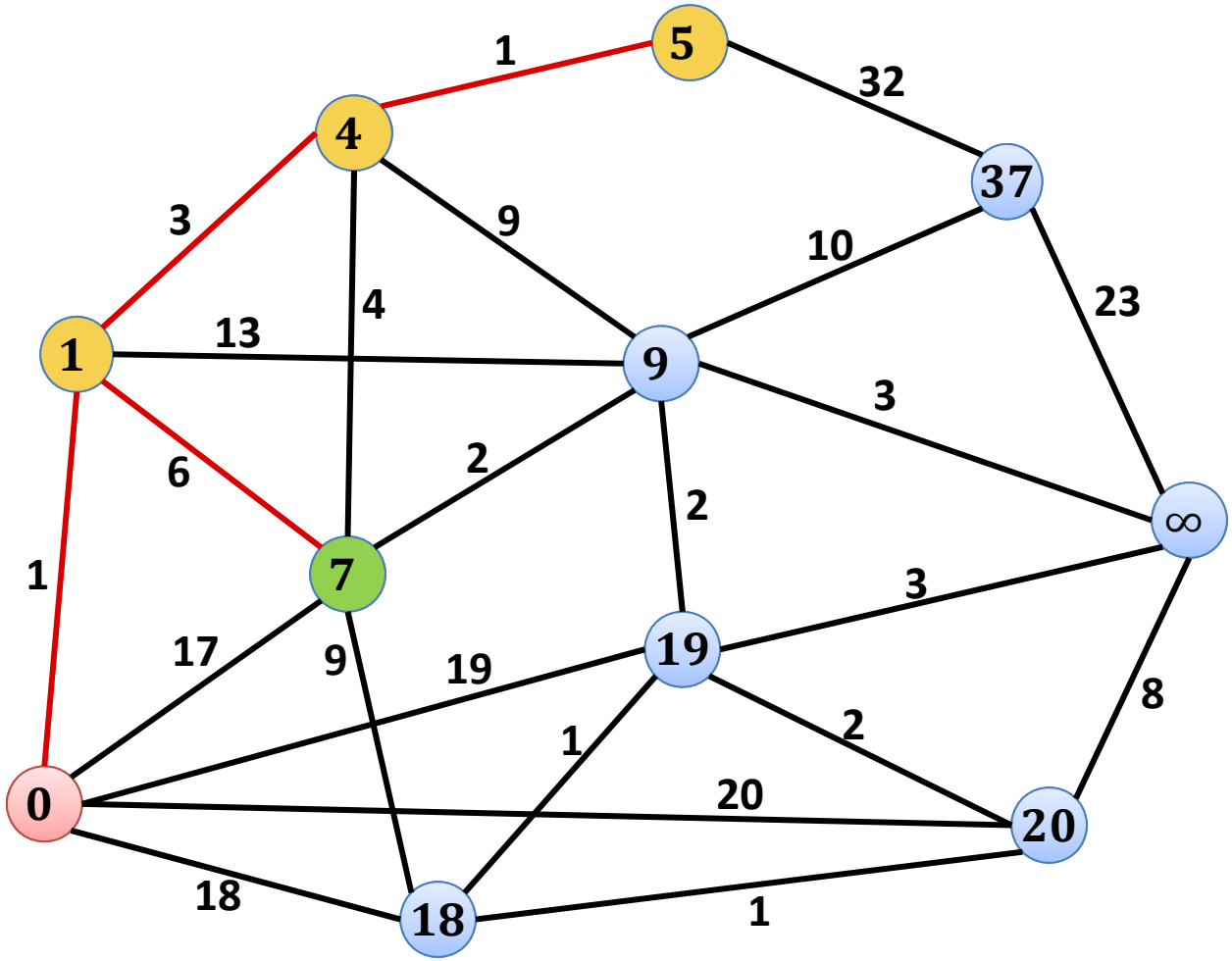
Example



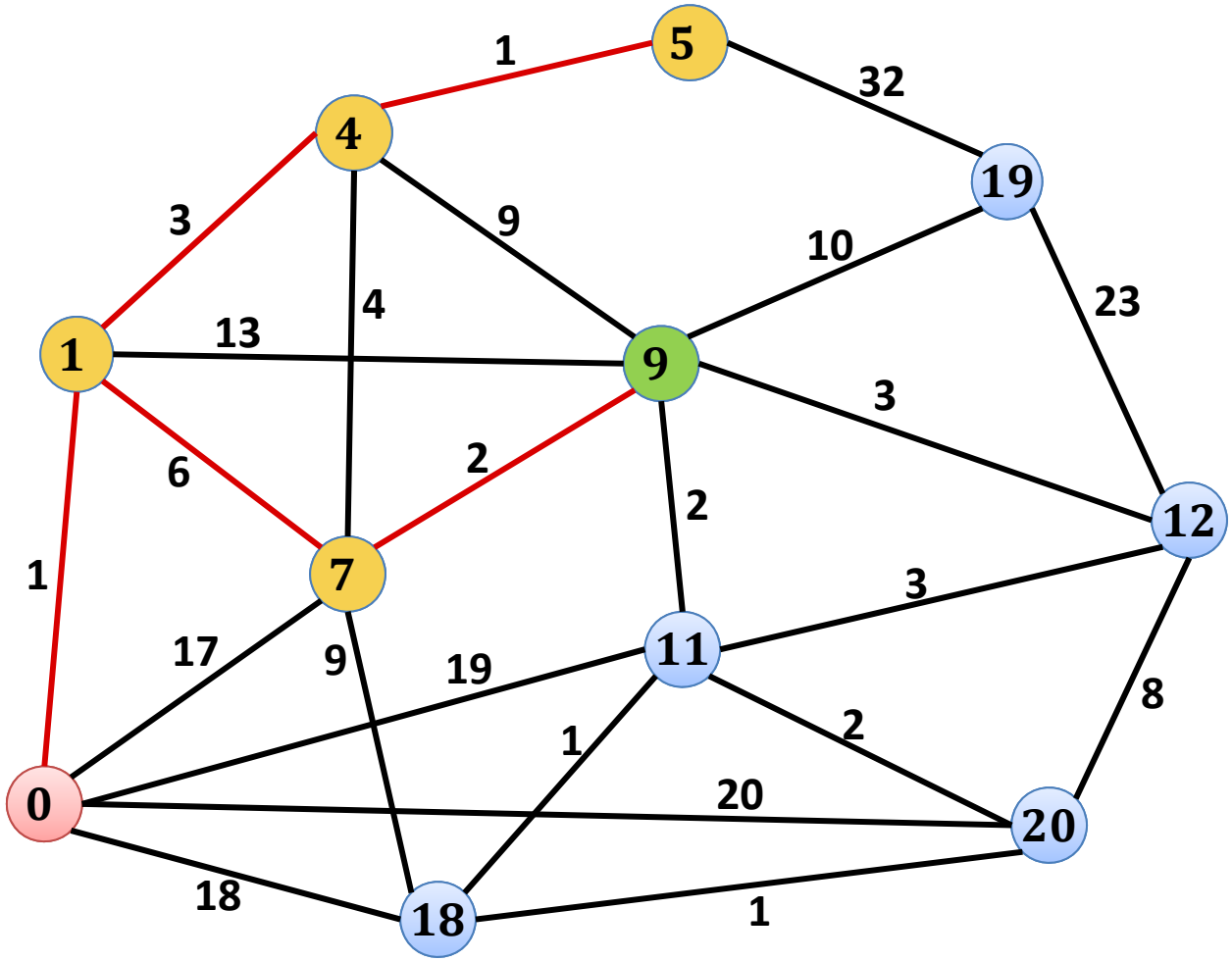
Example



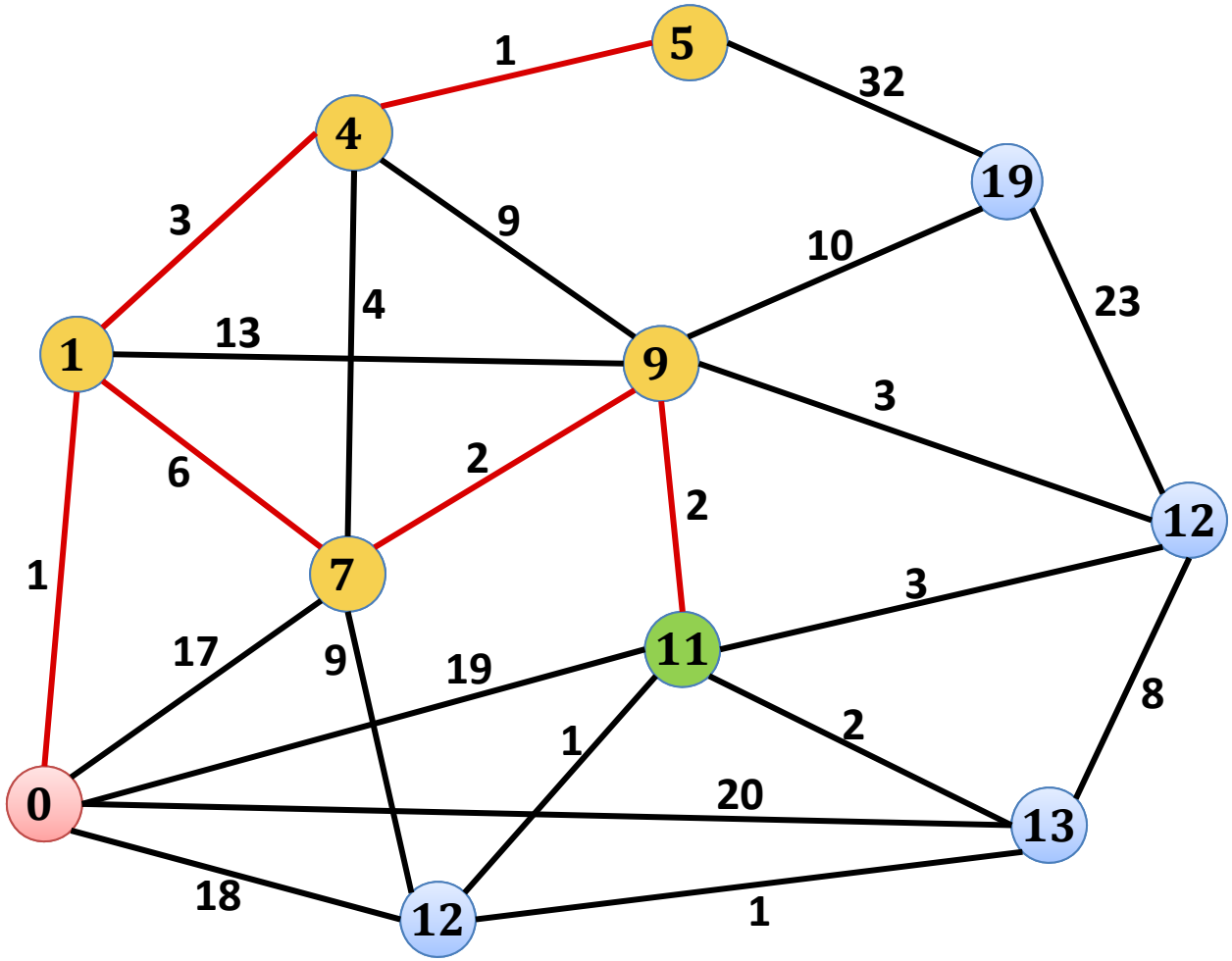
Example



Example



Example



Implementation of Dijkstra's Algorithm

Dijkstra's Algorithm:

1. Initialize $d(s,s) = 0$ and $d(s,v) = \infty$ for all $v \neq s$
2. All nodes $v \neq s$ are unmarked

   need to keep track of set of unmarked nodes (data struct. for unmarked nodes)
3. Get unmarked node u which minimizes $d(s,u)$:

   need to get unmarked node with smallest guess
4. For all $e = \{u,v\} \in E$, $d(s,v) = \min\{d(s,v), d(s,u) + w(e)\}$

   need to be able to update guesses

   decrease guesses
5. mark node u

   remove u from data struct. of unmarked nodes
6. Until all nodes are marked

Priority Queue / Heap

- Stores (*key,data*) pairs (like dictionary)
- But, different set of operations:

keys do not need to be unique

- **Initialize-Heap**: creates new empty heap
- **Is-Empty**: returns true if heap is empty
- **Insert**(*key,data*): inserts (*key,data*)-pair, returns pointer to entry
- **Get-Min**: returns (*key,data*)-pair with minimum *key*
- **Delete-Min**: deletes minimum (*key,data*)-pair
- **Decrease-Key**(*entry,newkey*): decreases *key* of *entry* to *newkey*
- **Merge**: merges two heaps into one

Implementation of Dijkstra's Algorithm

Store nodes in a priority queue, use $d(s,v)$ as keys:

1. Initialize $d(s,s) = 0$ and $d(s,v) = \infty$ for all $v \neq s$
2. All nodes $v \neq s$ are unmarked
3. Get unmarked node u which minimizes $d(s,u)$:
4. mark node u
5. For all $e = \{u,v\} \in E$, $d(s,v) = \min\{d(s,v), d(s,u) + w(e)\}$
6. Until all nodes are marked

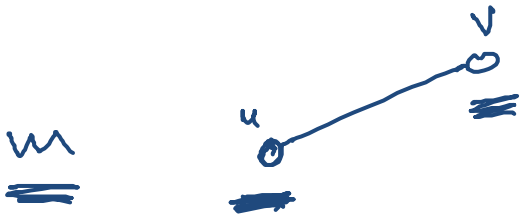
Analysis

G has n nodes, m edges. $G=(V,E)$, $n=|V|$, $m=|E|$

Number of priority queue operations for Dijkstra:

- **Initialize-Heap**: $\mathbf{1}$
- **Is-Empty**: n $|V|$
- **Insert**: n $|V|$
- **Get-Min**: n $|V|$
- **Delete-Min**: n $|V|$
- **Decrease-Key**: $|E|$
- **Merge**: $\mathbf{0}$

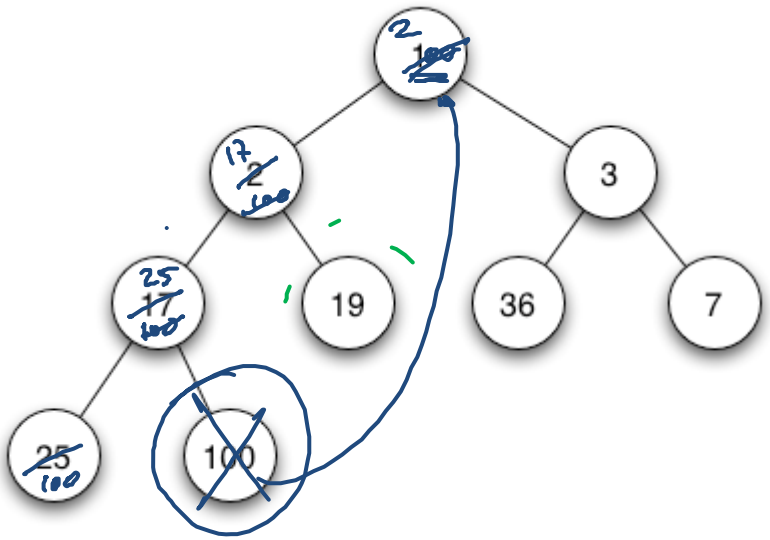
$m \geq n-1$



Priority Queue Implementation

Implementation as min-heap:

→ complete binary tree,
e.g., stored in an array



- **Initialize-Heap**: $O(1)$
- **Is-Empty**: $O(1)$
- **Insert**: $O(\log n)$
- **Get-Min**: $O(1)$
- **Delete-Min**: $O(\log n)$
- **Decrease-Key**: $O(\log n)$
- **Merge** (heaps of size m and n, $m \leq n$): $O(m \log n)$

Dijkstra!

$O(m \cdot \log n)$