Chapter 4
Data Structures

Algorithm Theory
WS 2012/13

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

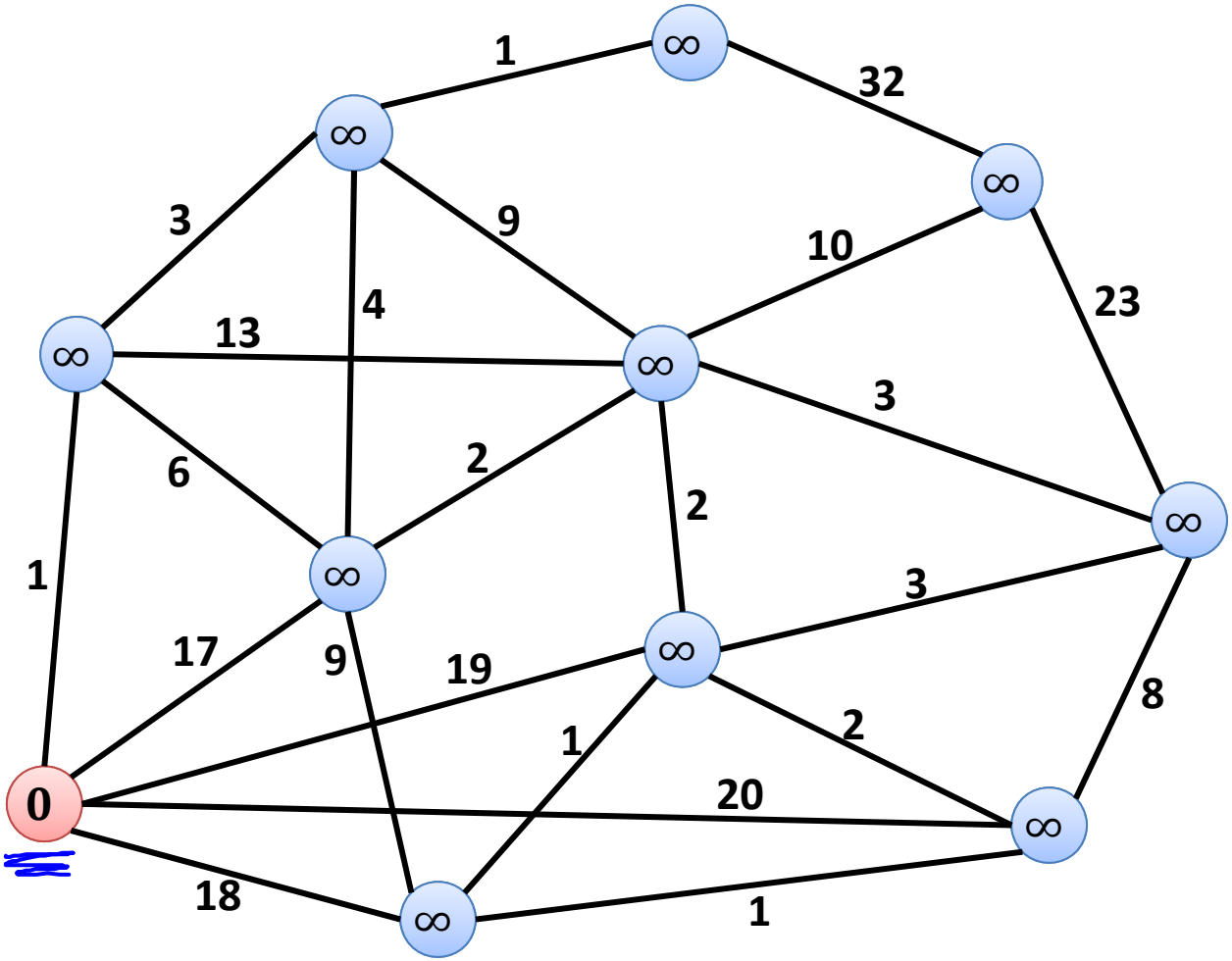
Single-Source Shortest Path Problem:

- **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 $v \in V$

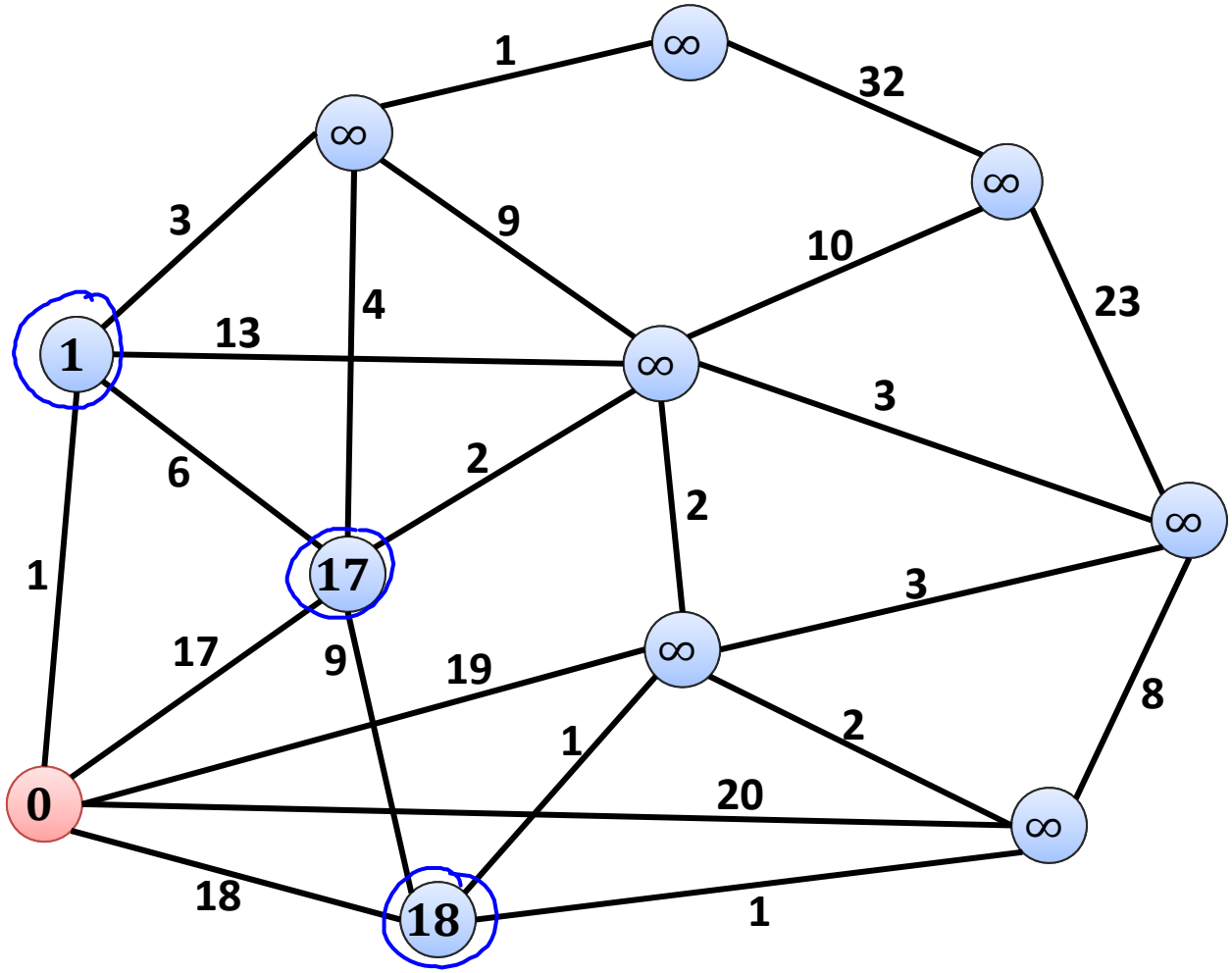
Dijkstra's Algorithm:

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

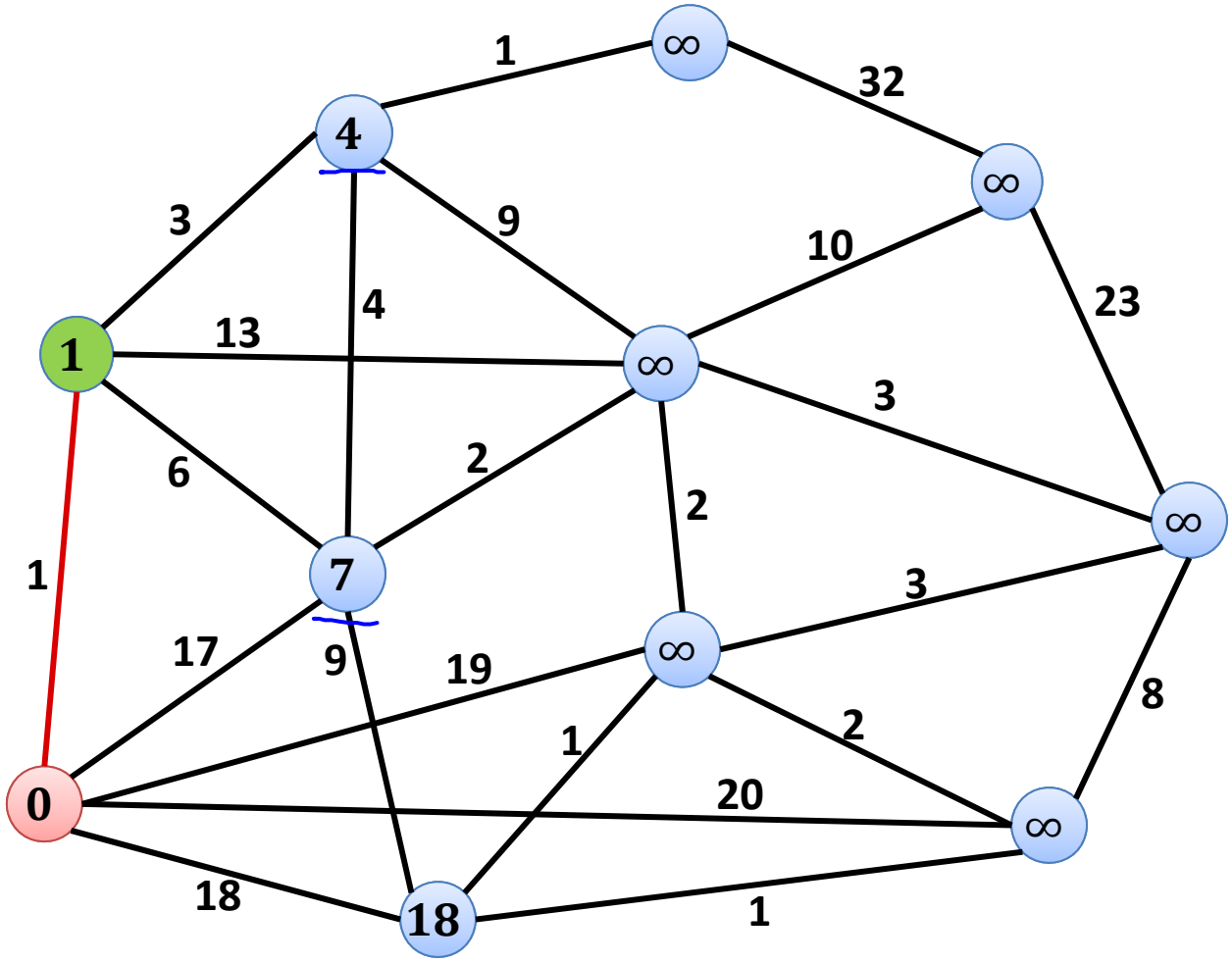
Example



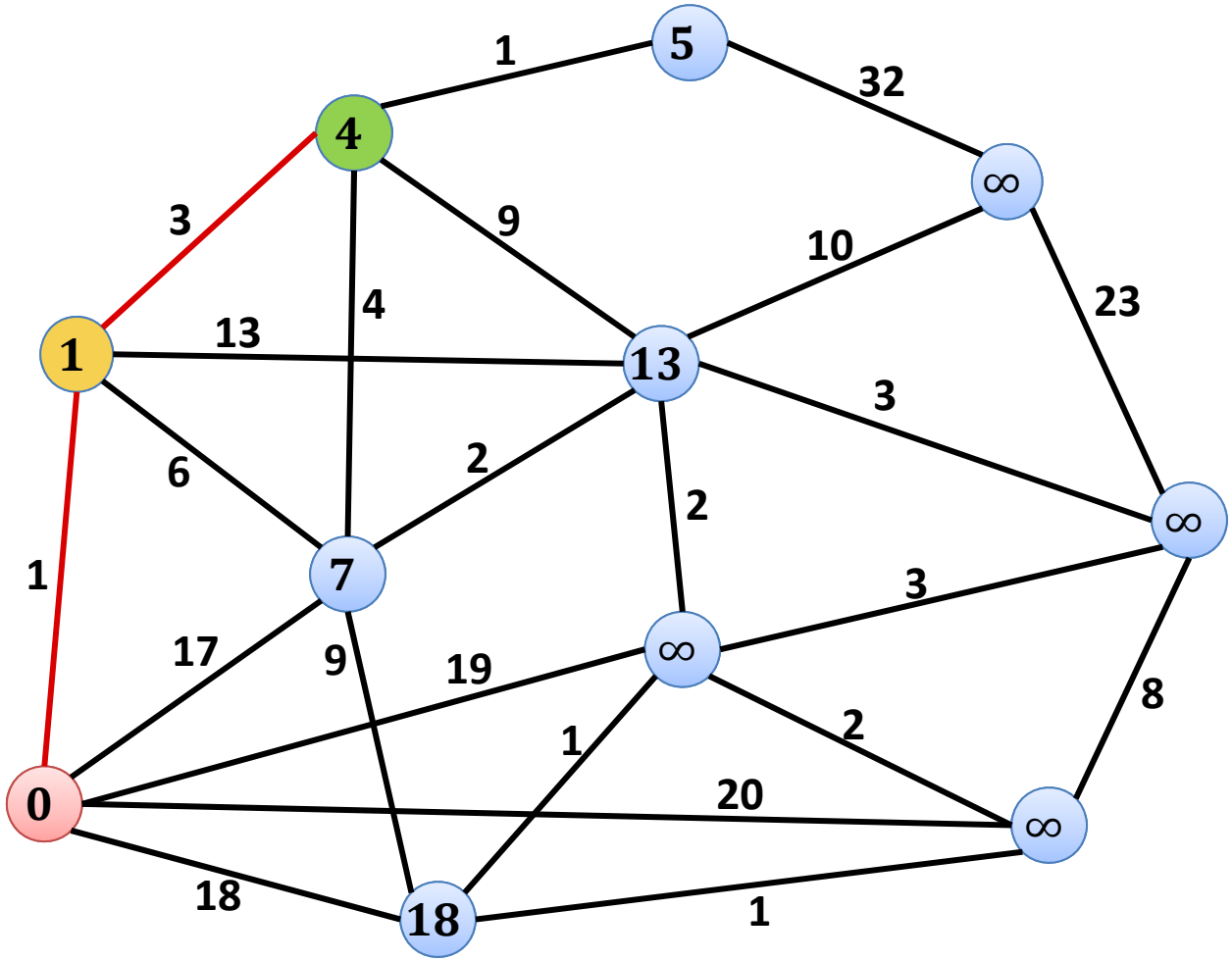
Example



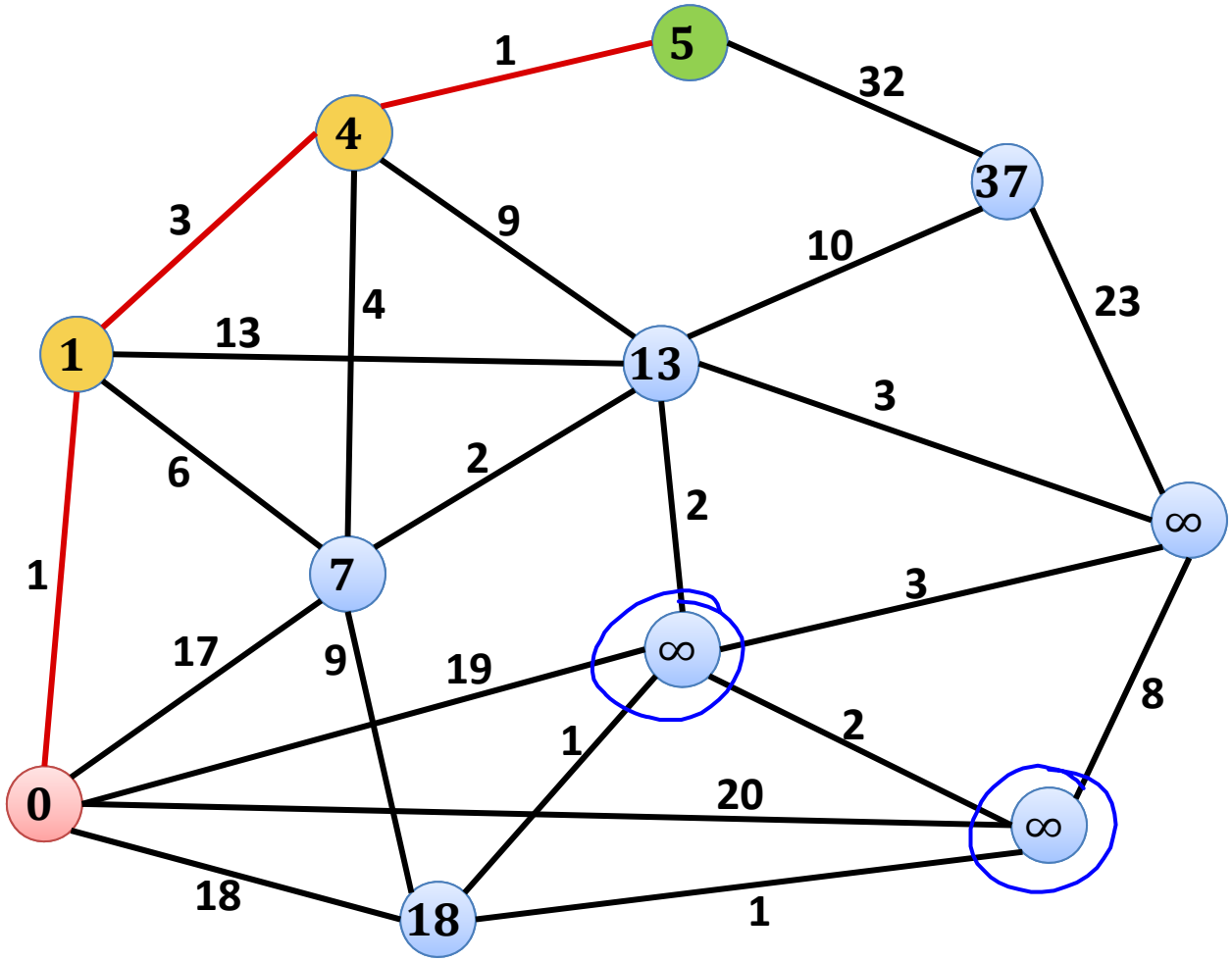
Example



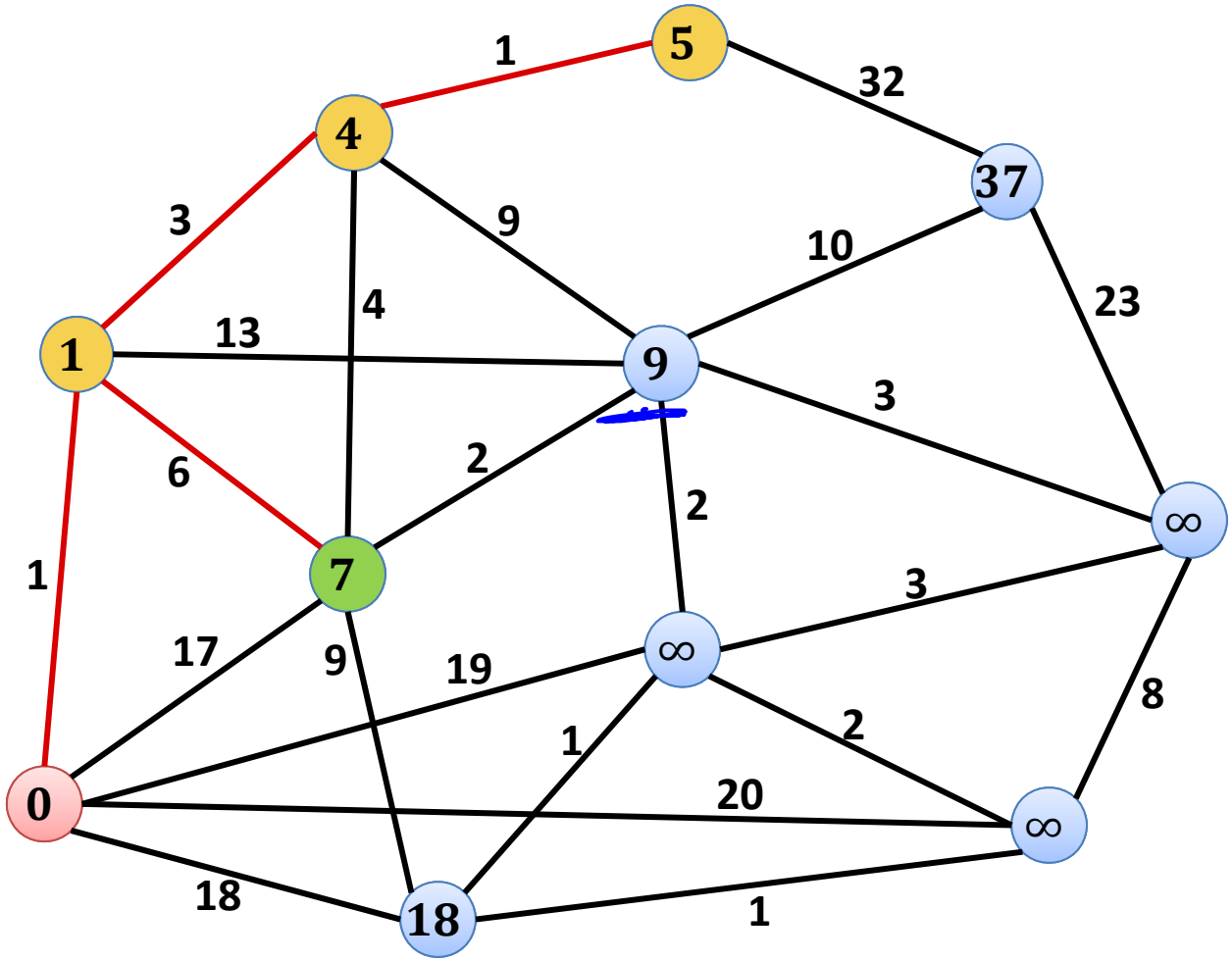
Example



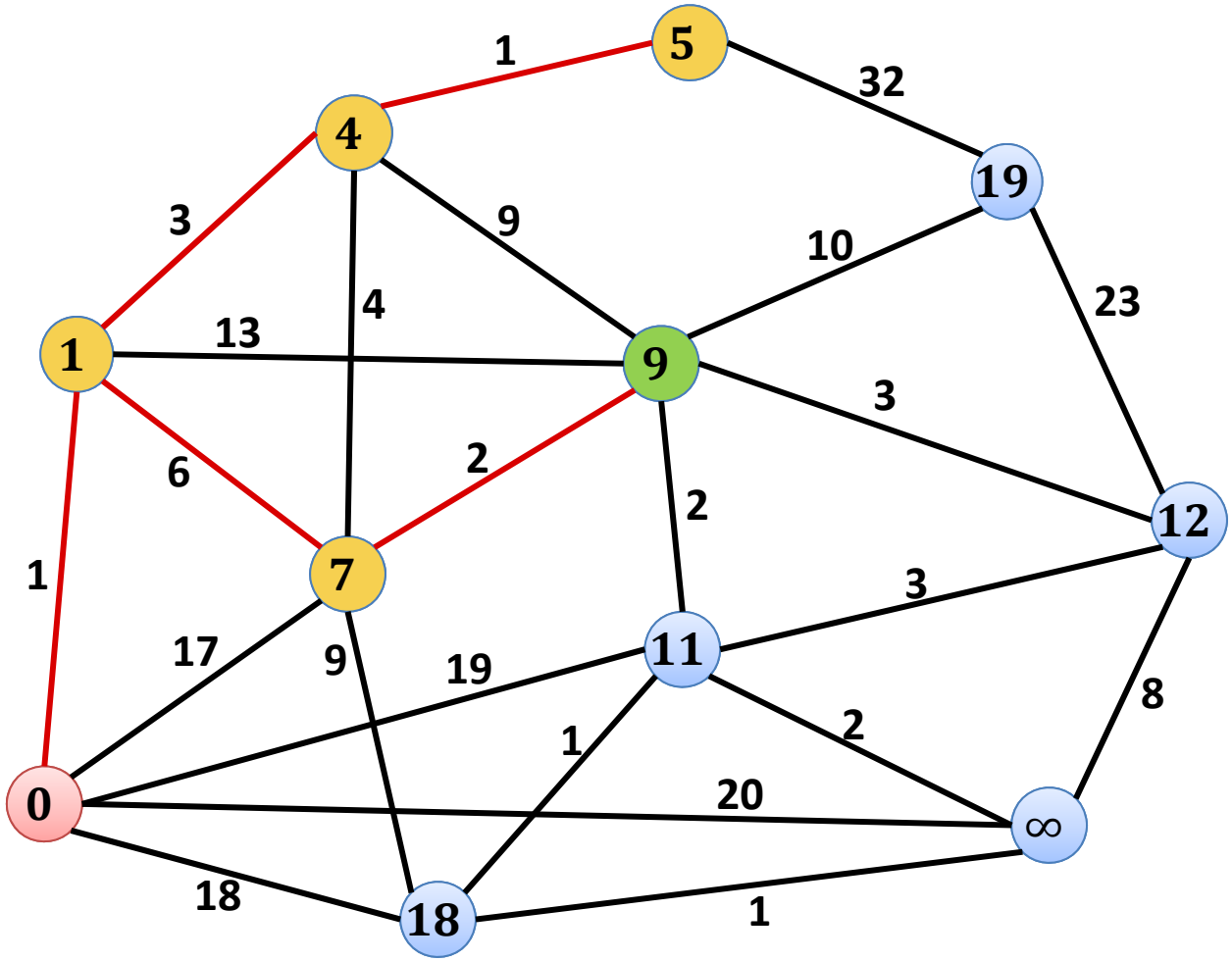
Example



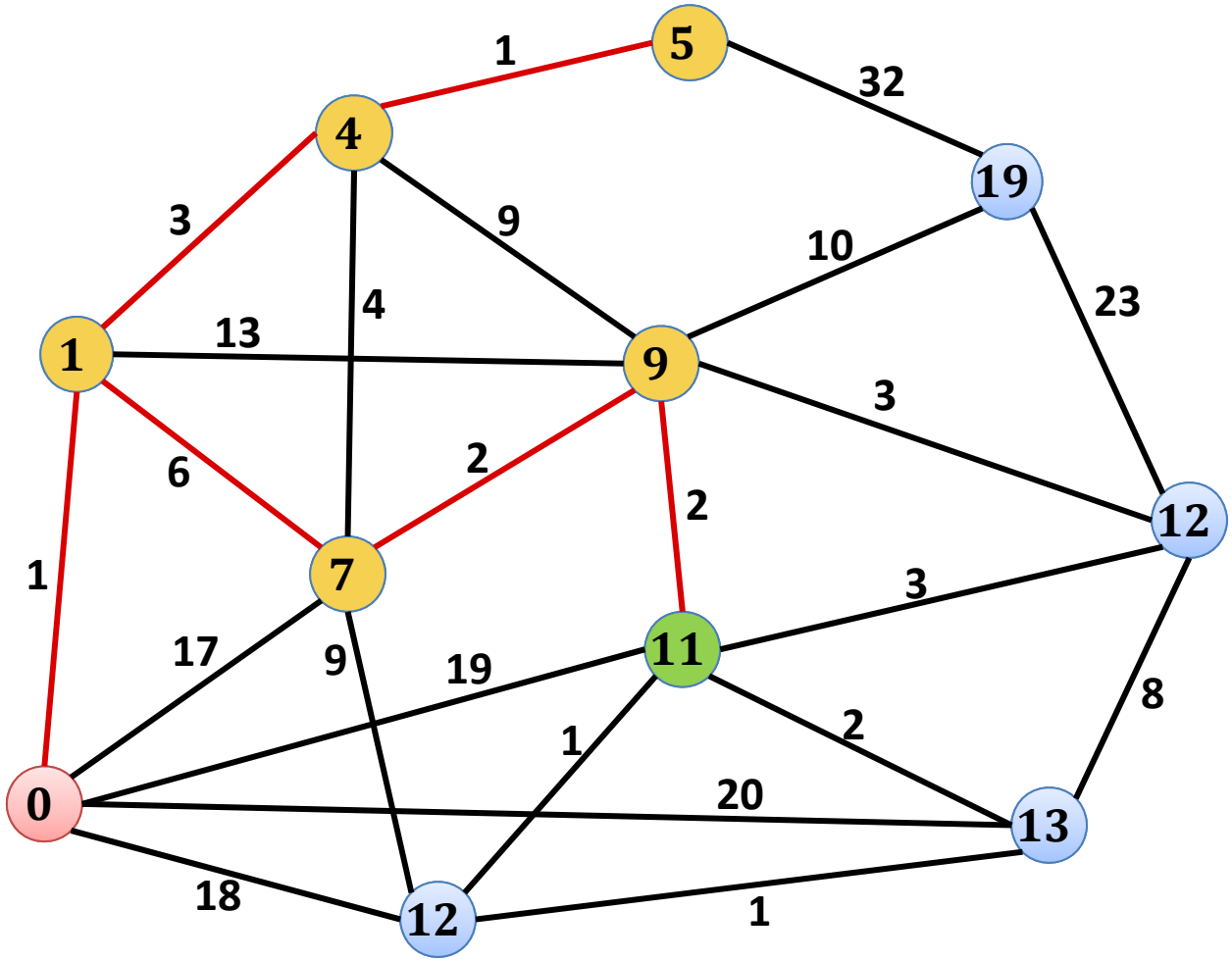
Example



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 are unmarked

   seems easy

3. Get unmarked node u which minimizes $d(s,u)$:

   need to get min efficiently

4. For all $e = \{u,v\} \in E$, $d(s,v) = \min\{d(s,v), d(s,u) + w(e)\}$

   update values

5. mark node u

   remove min

6. Until all nodes are marked

Priority Queue / Heap

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

- **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 are unmarked

   create heap, insert all nodes with key $d(s,v)$

3. Get unmarked node u which minimizes $d(s,u)$:

   get-min

4. mark node u

   delete-min

5. For all $e = \{u,v\} \in E$, $d(s,v) = \min\{d(s,v), d(s,u) + w(e)\}$

   decrease-key for all neighbors v of u

6. Until all nodes are marked is-empty?

Analysis

Number of priority queue operations for Dijkstra:

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

$G=(V,E)$
$|V|=n$, $|E|=m$

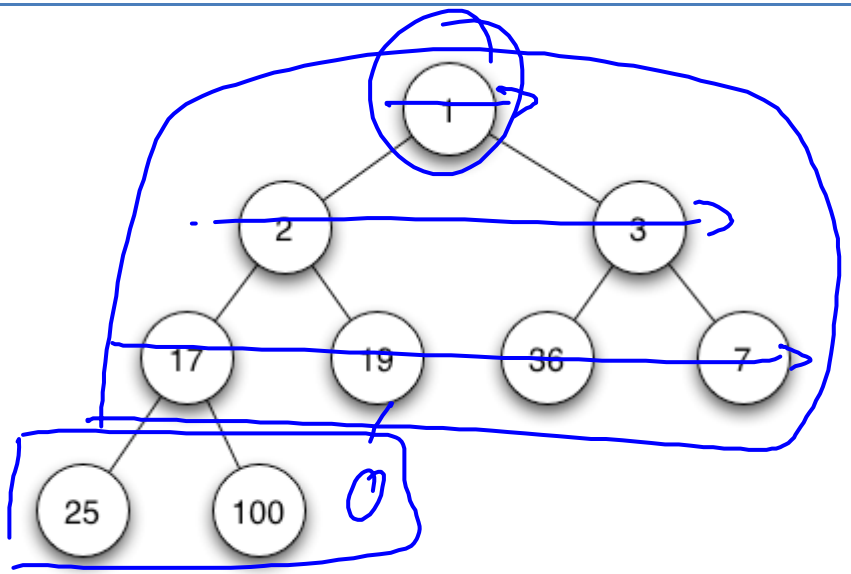
check whether we're done

initialization

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, $m \leq n$): $O(m \log n)$

cost of Dijkstra:

$O(m \log n)$

Better Implementation

- Can we do better?
- Cost of Dijkstra with complete binary min-heap implementation:

$$O(|E|\log|V|)$$

- Can be improved if we can make decrease-key cheaper…
- Cost of merging two heaps is expensive
- We will get there in two steps:

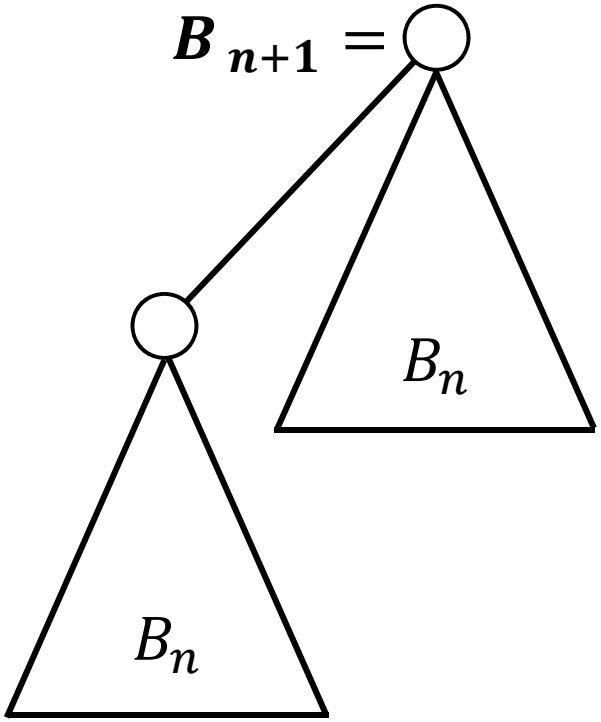
  Binomial heap → Fibonacci heap

Definition: Binomial Tree

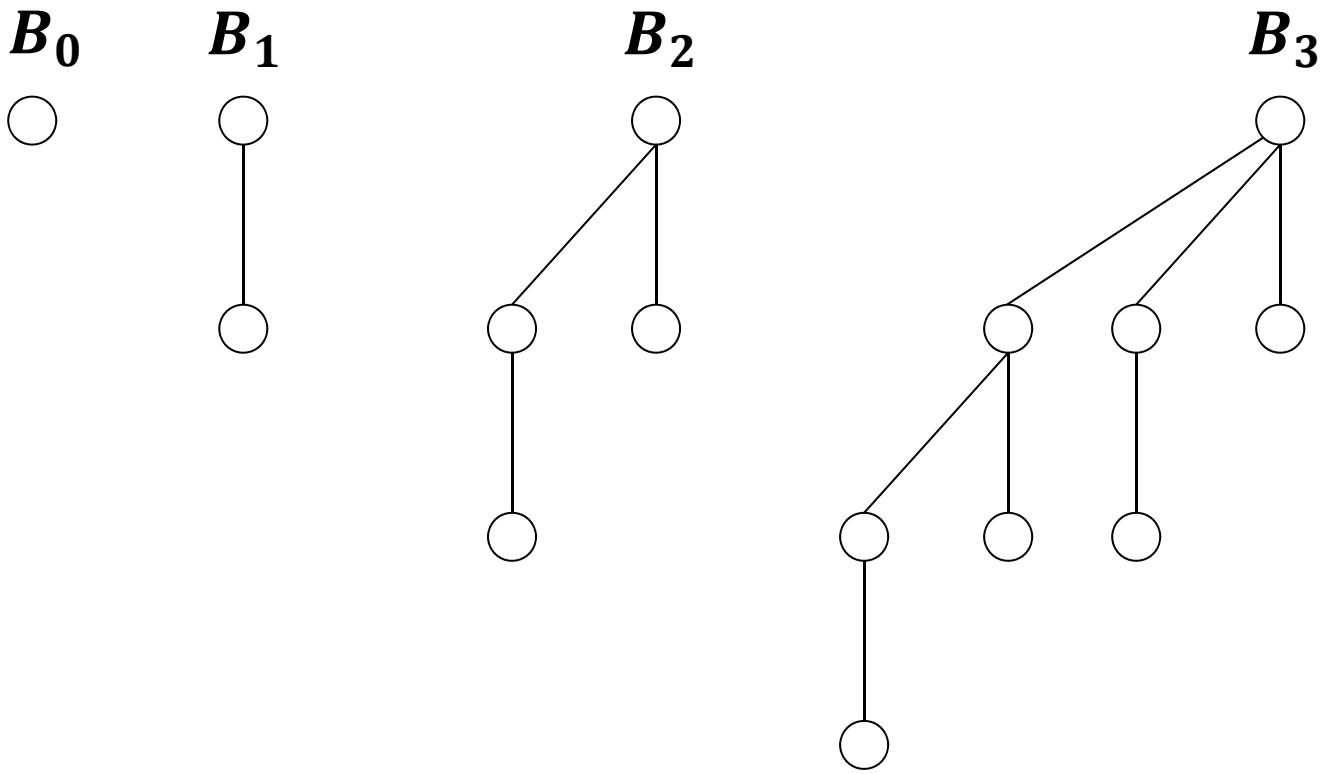
Binomial tree B_n of order n $(n \geq 0)$:

$\boldsymbol{B_0} =$ ○

$\boldsymbol{B_{n+1}} =$



Binomial Trees



Properties

1. Tree B_n has 2^n nodes

2. Height of tree B_n is n

3. Root degree of B_n is n

4. In B_n, there are exactly $\binom{n}{i}$ nodes at depth i

Binomial Coefficients

- Binomial coefficient:

$$\binom{n}{k} : \#\text{ of } k-\text{element}-\text{subsets of a set of size } n$$

- Property: $\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}$

Pascal triangle:

$$\begin{array}{ccccccccccc}
 & & & & & 1 & & & & & \\
 & & & & 1 & & 1 & & & & \\
 & & & 1 & & 2 & & 1 & & & \\
 & & 1 & & 3 & & 3 & & 1 & & \\
 & 1 & & 4 & & 6 & & 4 & & 1 & \\
1 & & 5 & & 10 & & 10 & & 5 & & 1
\end{array}$$

Number of Nodes at Depth i in B_n

Claim: In B_n, there are exactly $\binom{n}{i}$ nodes at depth i

Binomial Heap

- Keys are stored in nodes of binomial trees of different order

  n nodes: there is a binomial tree B_i or order i iff
  bit i of base-2 representation of n is 1.

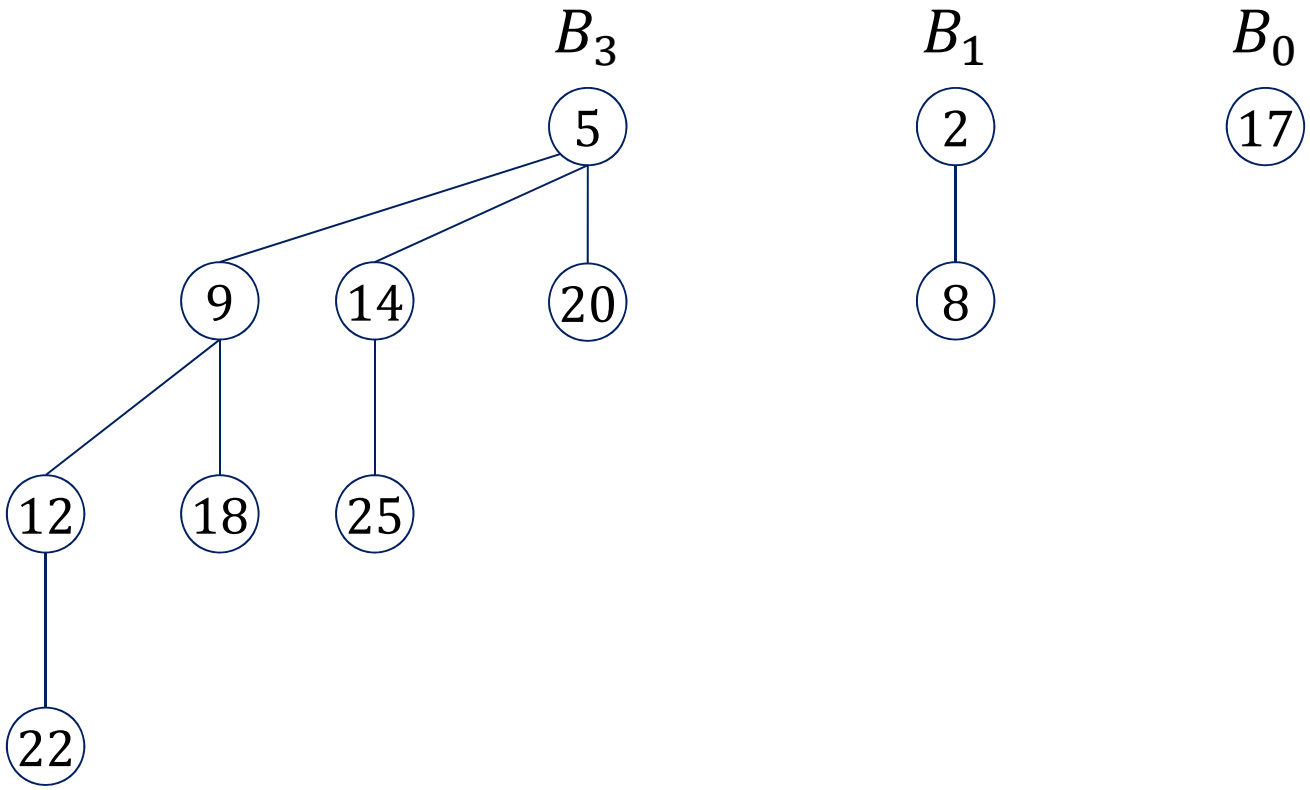
- **Min-Heap Property:**

  Key of node $v \leq$ keys of all nodes in sub-tree of v

Example

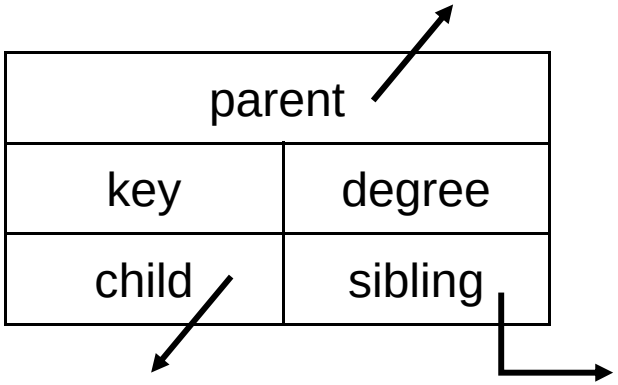
- 10 keys: $\{2, 5, 8, 9, 12, 14, 17, 18, 20, 22, 25\}$

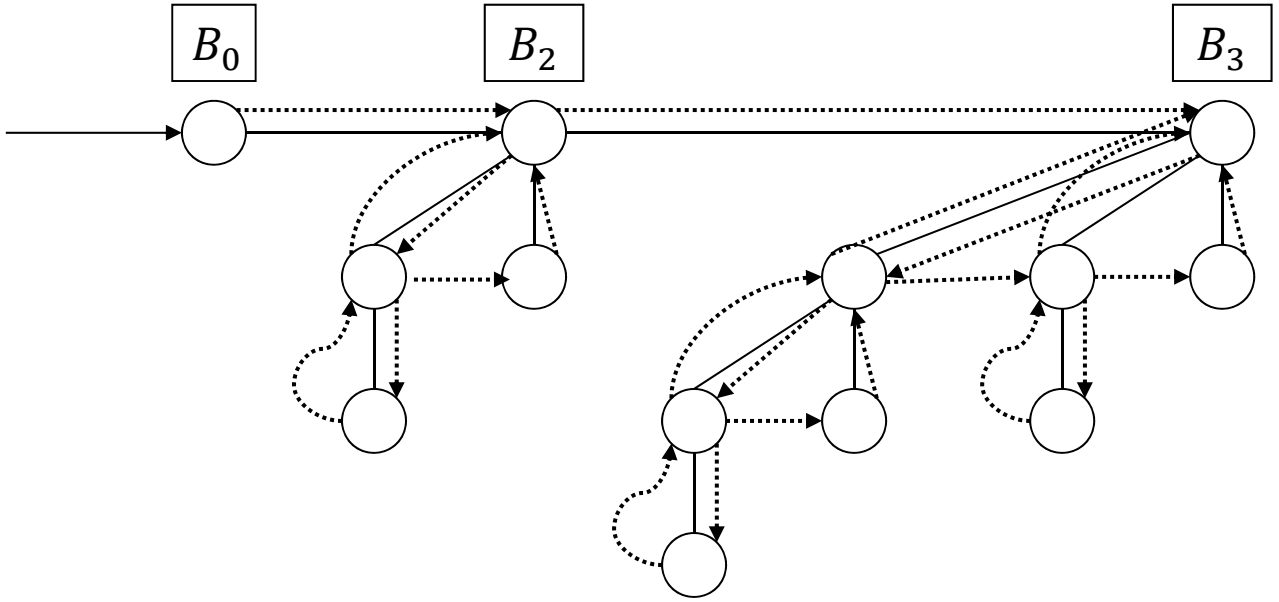
- Binary representation of n: $(11)_2 = 1011$
  → trees B_0, B_1, and B_3 present



Child-Sibling Representation

Structure of a node:



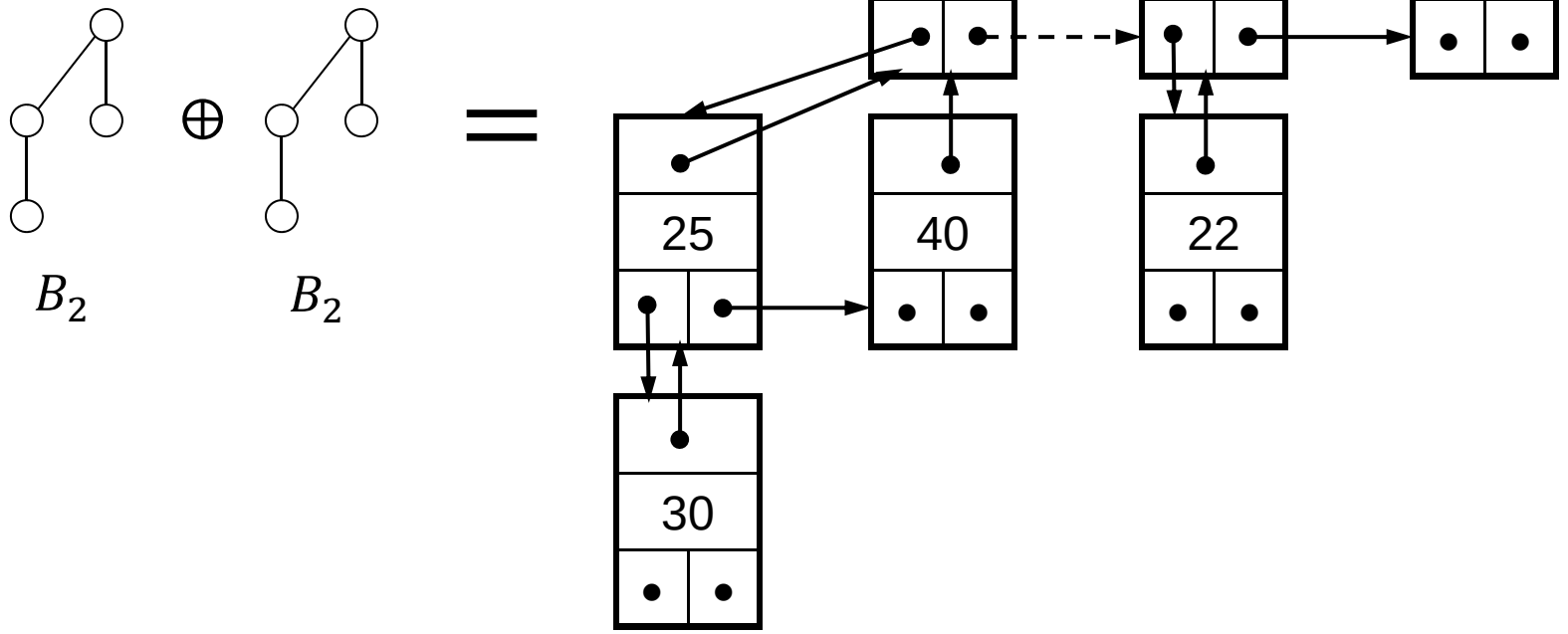


Link Operation

- Unite two binomial trees of the same order to one tree:

$$B_n \oplus B_n \Rightarrow B_{n+1}$$

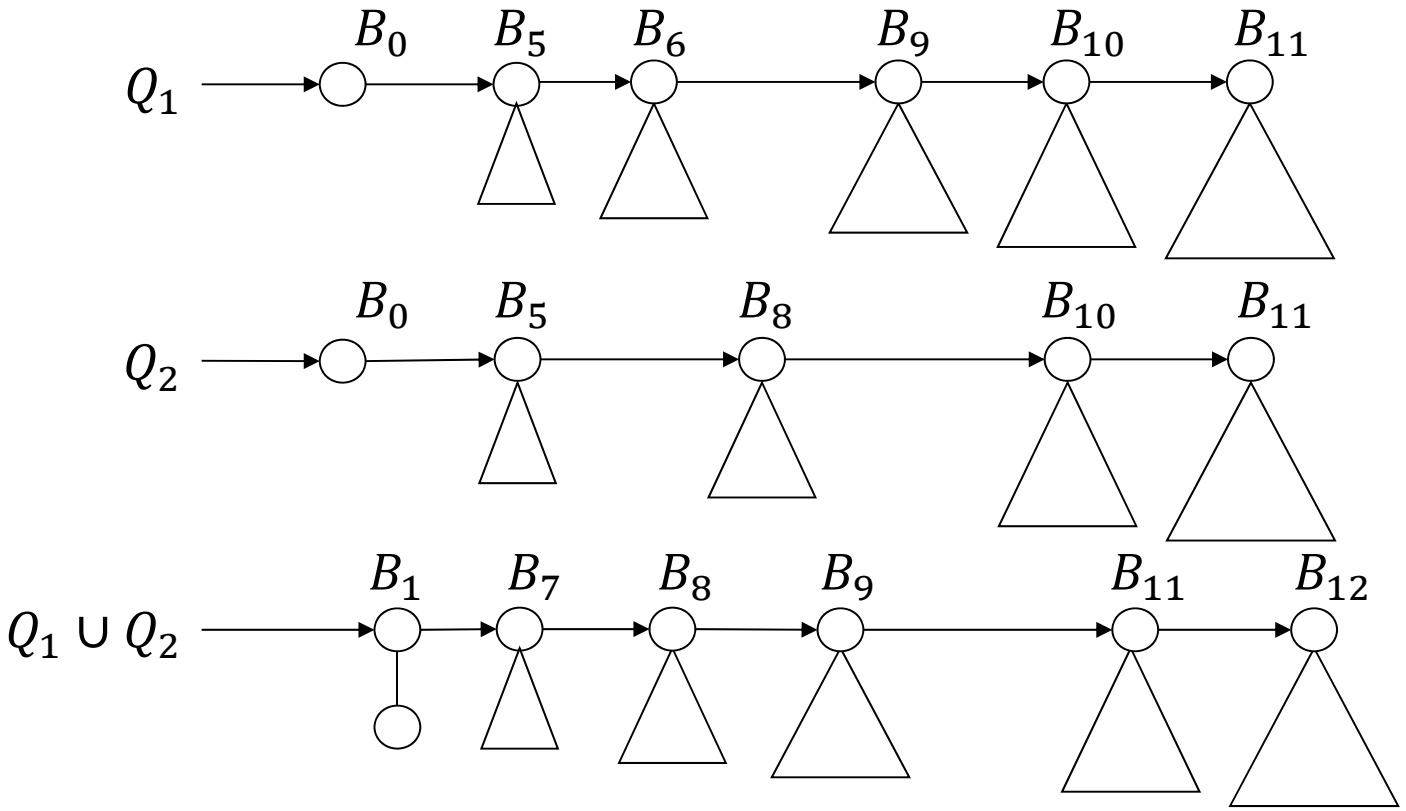
- Time: $O(1)$



Merge Operation

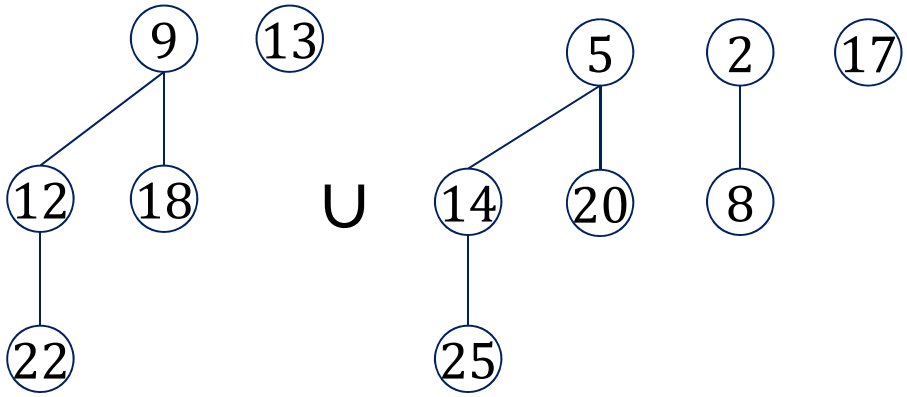
Merging two binomial heaps:

- **For $i = 0, 1, \dots, \log n$:**
  If there are 2 or 3 binomial trees B_i: apply link operation to merge 2 trees into one binomial tree B_{i+1}



Time:
$O(\log n)$

Example



Operations

Initialize: create empty list of trees

Get minimum of queue: time $O(1)$ (if we maintain a pointer)

Decrease-key at node v:

- Set *key* of node v to new key
- Swap with parent until min-heap property is restored
- Time: $O(\log n)$

Insert *key* x into queue Q:

1. Create queue Q' of size 1 containing only x
2. Merge Q and Q'

- Time for insert: $O(\log n)$

Operations

Delete-Min Operation:

1. Find tree B_i with minimum root r

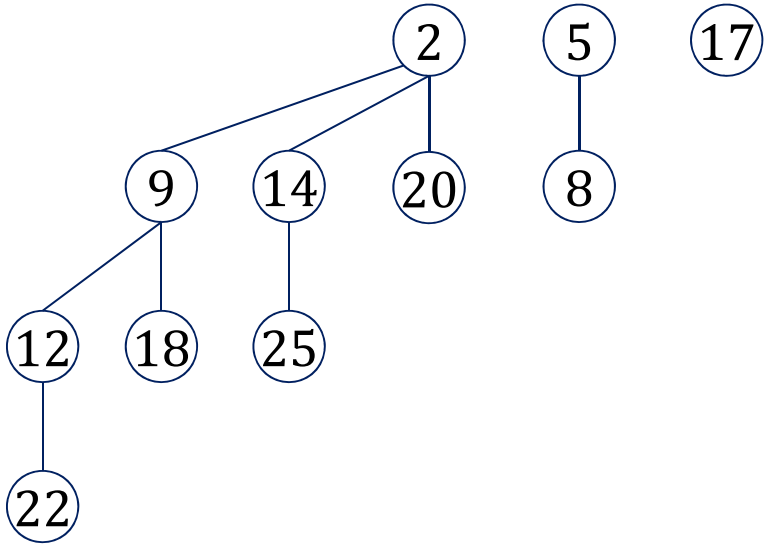
2. Remove B_i from queue Q → queue Q'

3. Children of r form new queue Q''

4. Merge queues Q' and Q''

- **Overall time: $O(\log n)$**

Delete-Min Example



Complexities Binomial Heap

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