CS3000: Algorithms & Data
Paul Hand

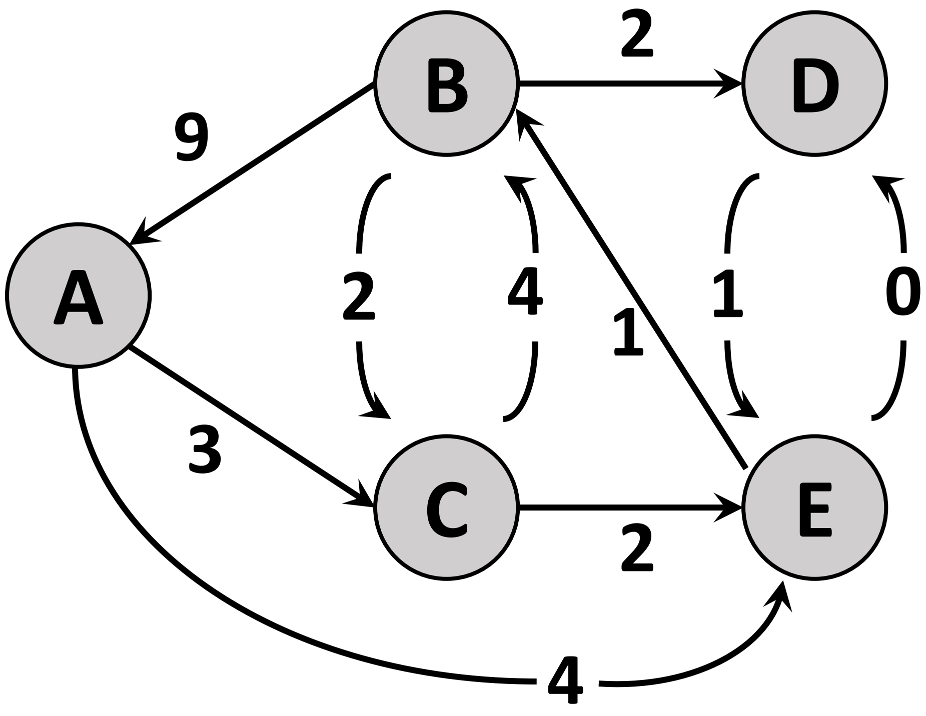
Lecture 17:

- Implementation of Dijkstra's Algorithm (Data Structures)

Mar 20, 2019

Execute Dijkstra's Algorithm: Activity
Find distances from A and the shortest path tree

	A	B	C	D	E
$d_0(u)$	**0**	∞	∞	∞	∞
$d_1(u)$	**0**				
$d_2(u)$	**0**				
$d_3(u)$	**0**				
$d_4(u)$	**0**				
$d_5(u)$	**0**				



Implementing Dijkstra

```
Dijkstra(G = (V,E,{ℓ(e)}, s):
  d[s] ← 0, d[u] ← ∞ for every u != s
  parent[u] ←⊥ for every u
  Q ← V                  // Q holds the unexplored nodes

  While (Q is not empty):
    u ← argmin_{w∈Q} d[w]     //Find closest unexplored
    Remove u from Q

    // Update the neighbors of u
    For ((u,v) in E):
      If (d[v] > d[u] + ℓ(u,v)):
        d[v] ← d[u] + ℓ(u,v)
        parent[v] ← u

  Return (d, parent)
```

current estimates

Priority Queues / Heaps

Priority Queues

- Need a data structure Q to hold key-value pairs

- Need to support the following operations
  - Insert(Q,k,v): add a new key-value pair
  - Lookup(Q,k): return the value of some key
  - ExtractMin(Q): identify the key with the smallest value
  - DecreaseKey(Q,k,v): reduce the value of some key

Priority Queues

- **Naïve approach:** Sorted List

Key	a	c	e	h	b	g	k	d	f
Value	1	3	5	8	10	20	42	45	50

- Activity: With n total items, how long would it take to perform
  - Insert(Q,k,v): add a new key-value pair?
  - Lookup(Q,k): return the value of some key?
  - ExtractMin(Q): identify the key with the smallest value?
  - DecreaseKey(Q,k,v): reduce the value of some key?

Priority Queues

- **Naïve approach:** linked lists

Key	a	c	e	h	b	g	k	d	f
Value	11	12	2	36	4	20	42	10	8

- Activity: With n total items, how long would it take to perform
  - Insert(Q,k,v): add a new key-value pair?
  - Lookup(Q,k): return the value of some key?
  - ExtractMin(Q): identify the key with the smallest value?
  - DecreaseKey(Q,k,v): reduce the value of some key?

Priority Queues

- **Naïve approach:** linked lists

Key	a	c	e	h	b	g	k	d	f
Value	11	12	2	36	4	20	42	10	8

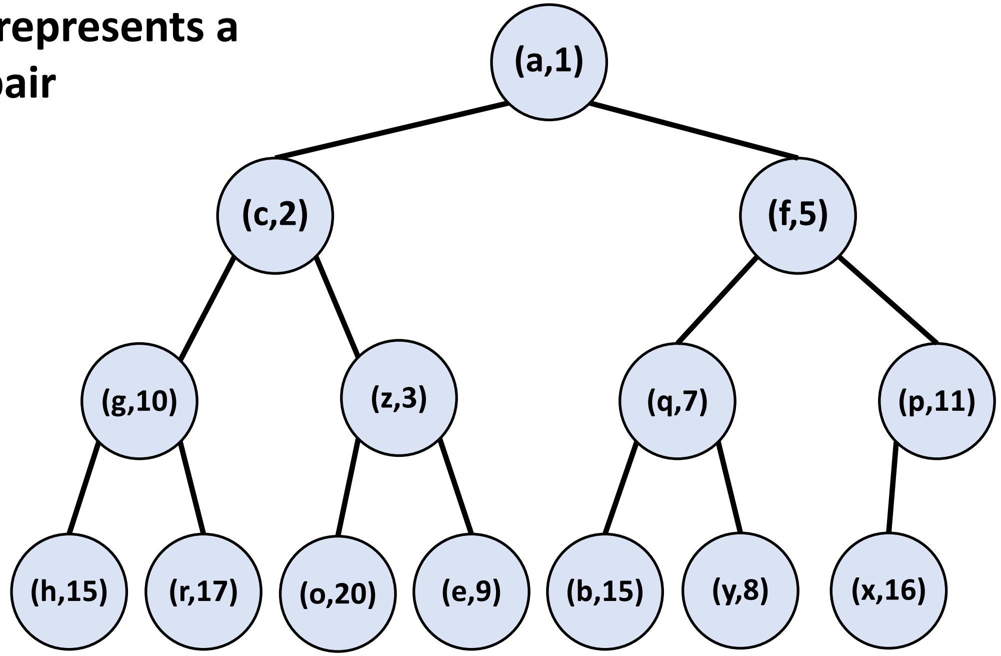
  - Insert takes O(1) time
  - ExtractMin, DecreaseKey take O(n) time

- **Binary Heaps:** implement all operations in O(log n) time where n is the number of keys

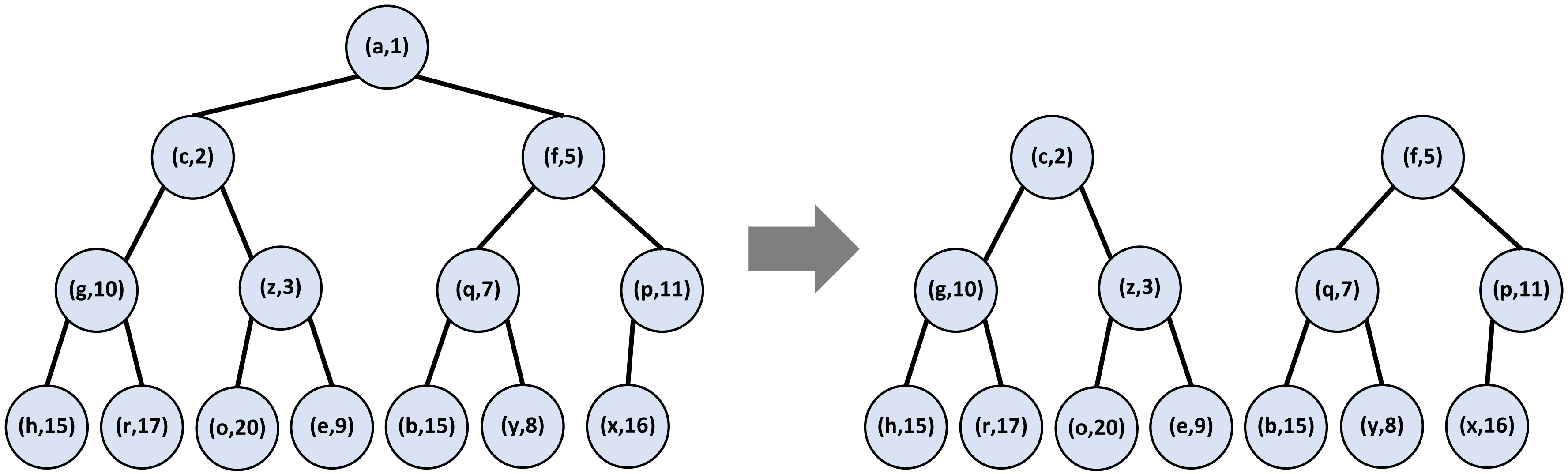
Heaps

- **Organize key-value pairs as a binary tree**
  - Later we'll see how to store pairs in an array
- **Heap Order:** If a is the parent of b, then v(a) ≤ v(b)

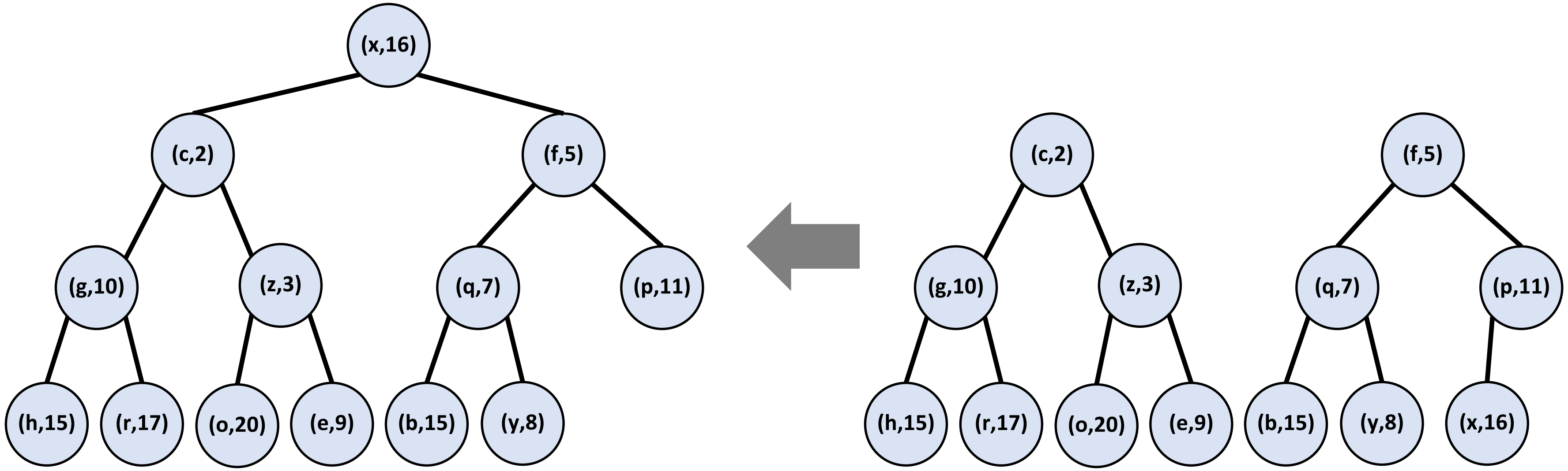
Each node represents a key-value pair



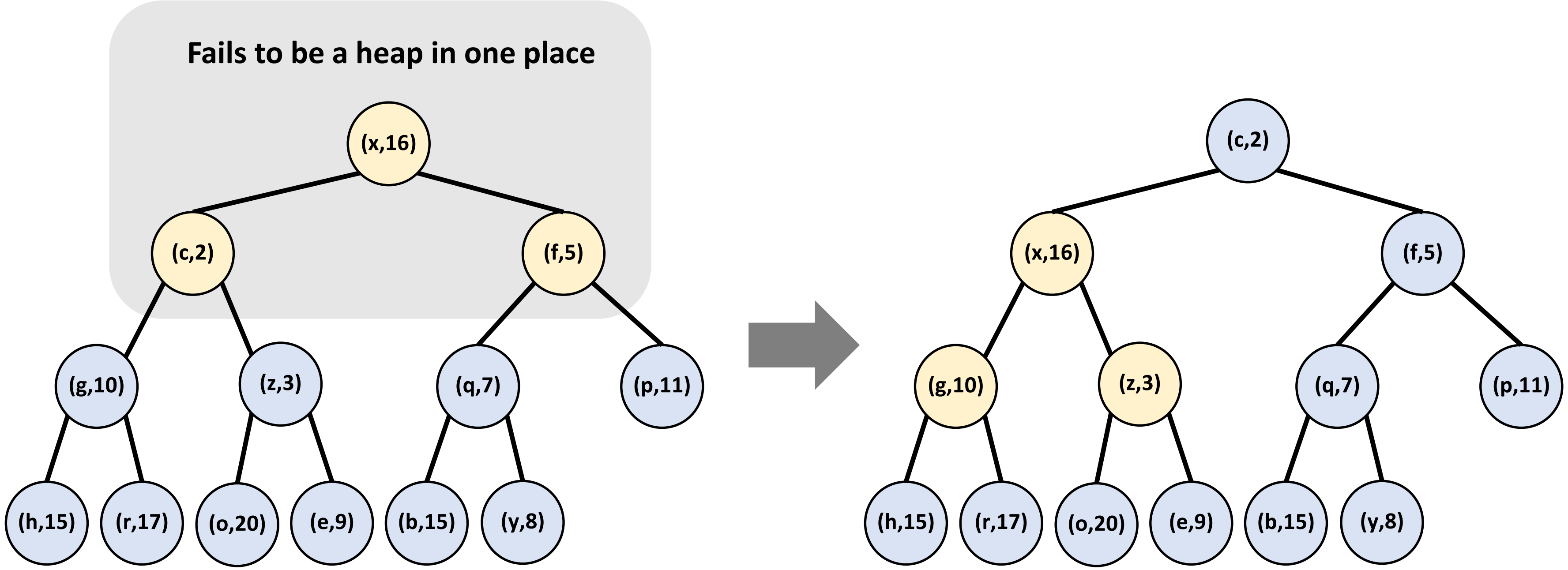
Implementing ExtractMin



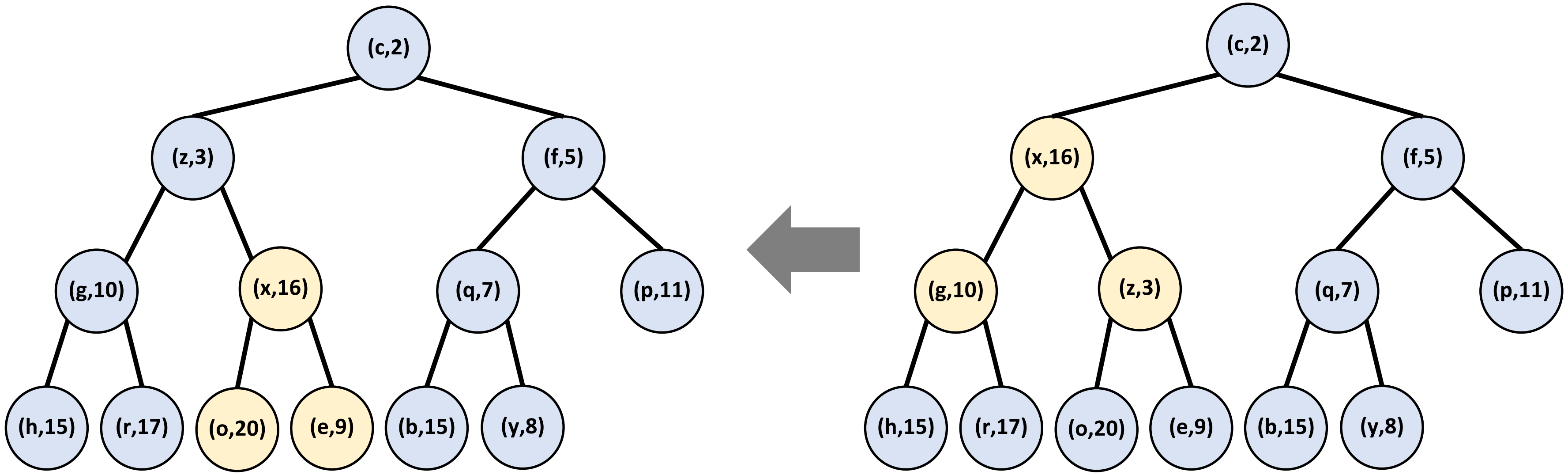
Implementing ExtractMin



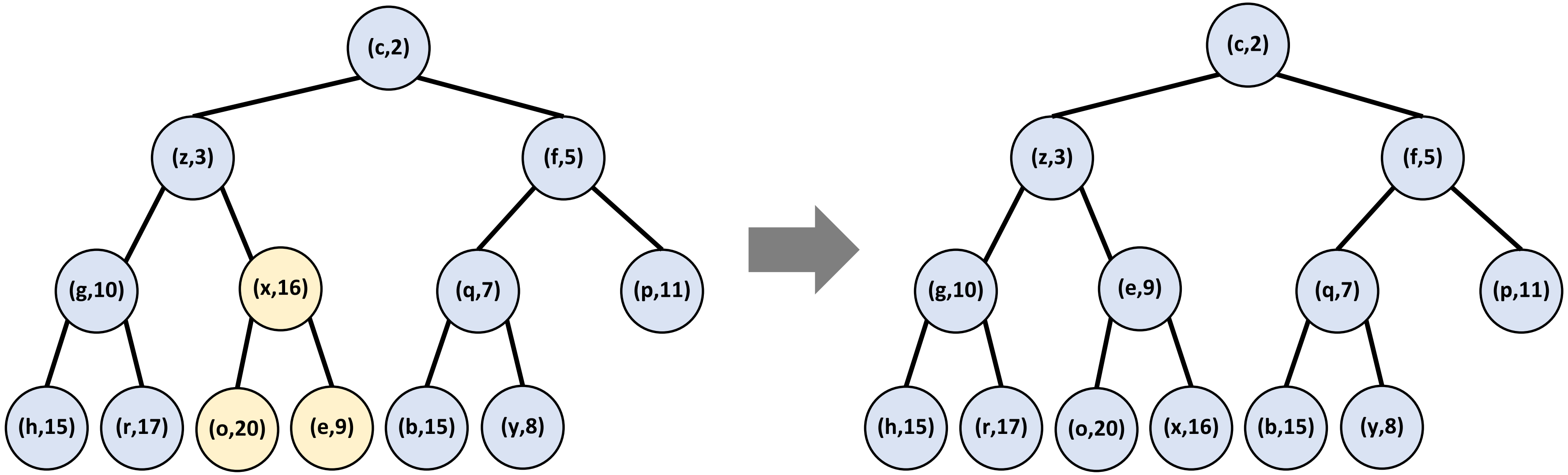
Implementing ExtractMin



Implementing ExtractMin

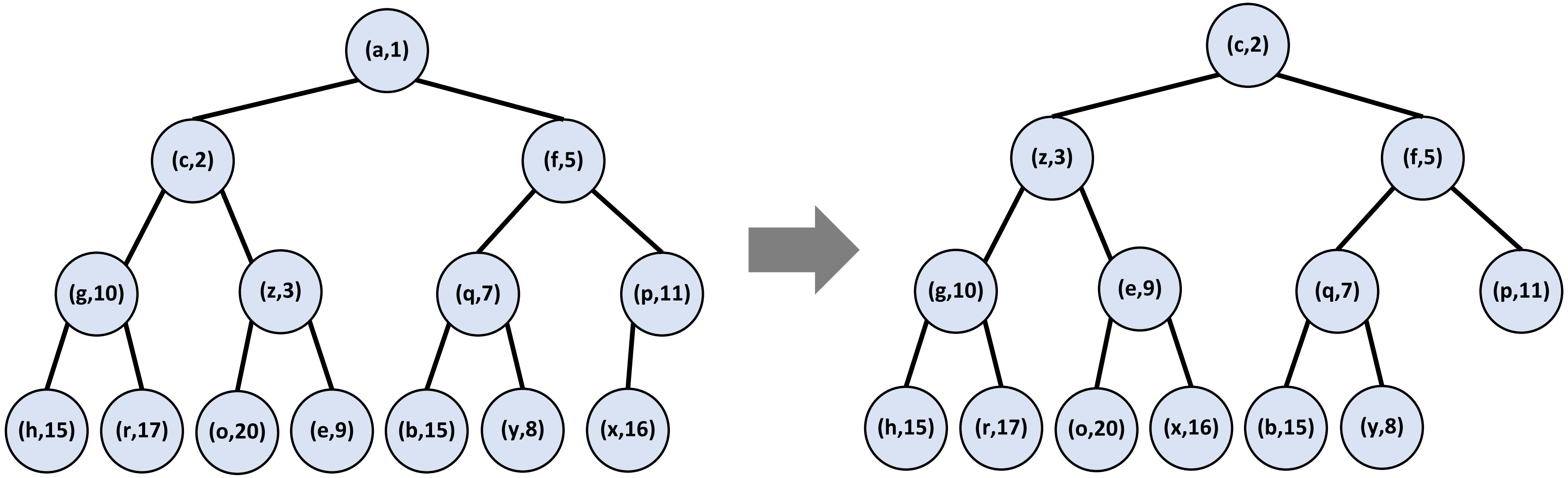


Implementing ExtractMin

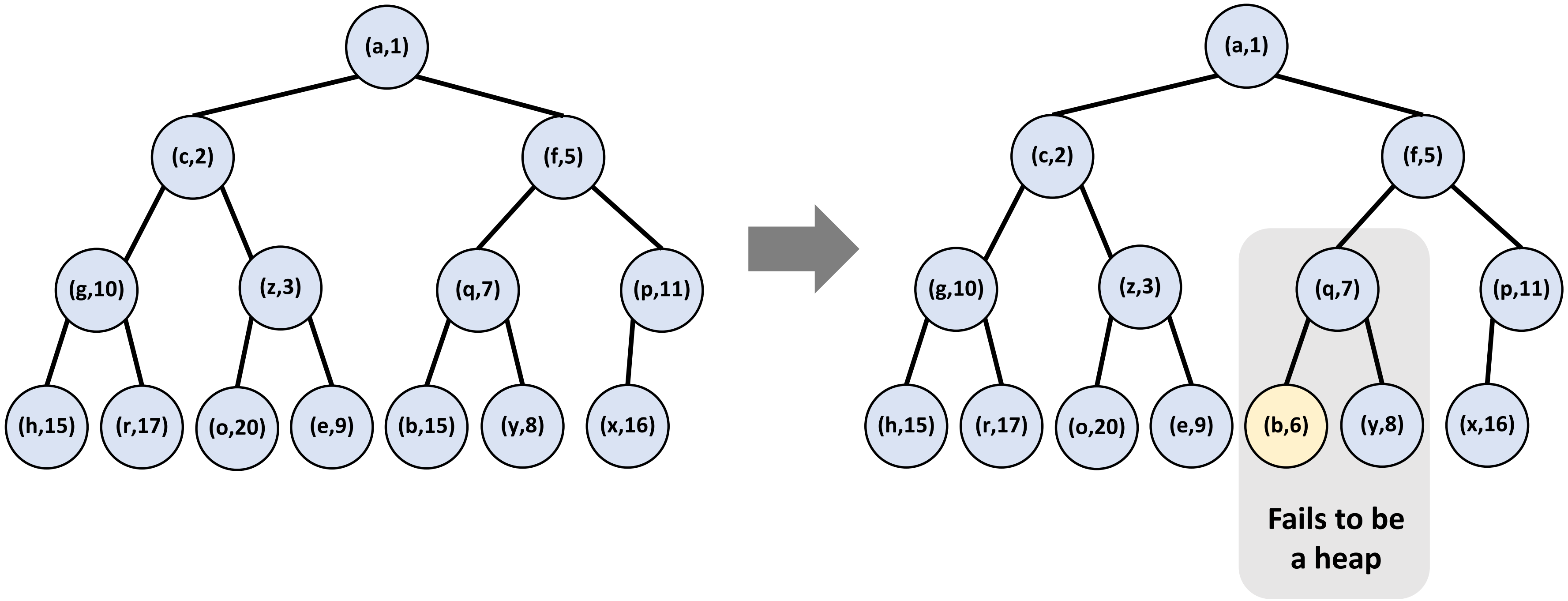


Implementing ExtractMin

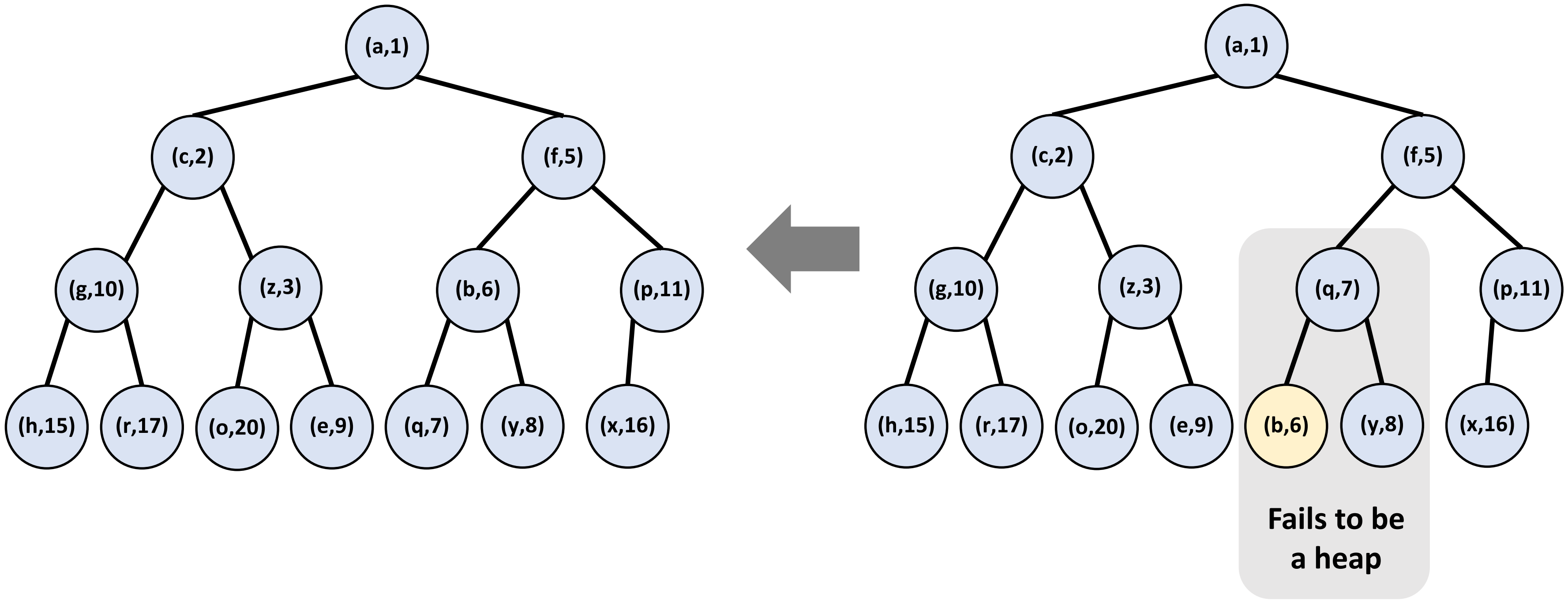
- Three steps:
  - Pull the minimum from the root
  - Move the last element to the root
  - Repair the heap-order (heapify down)



Implementing DecreaseKey



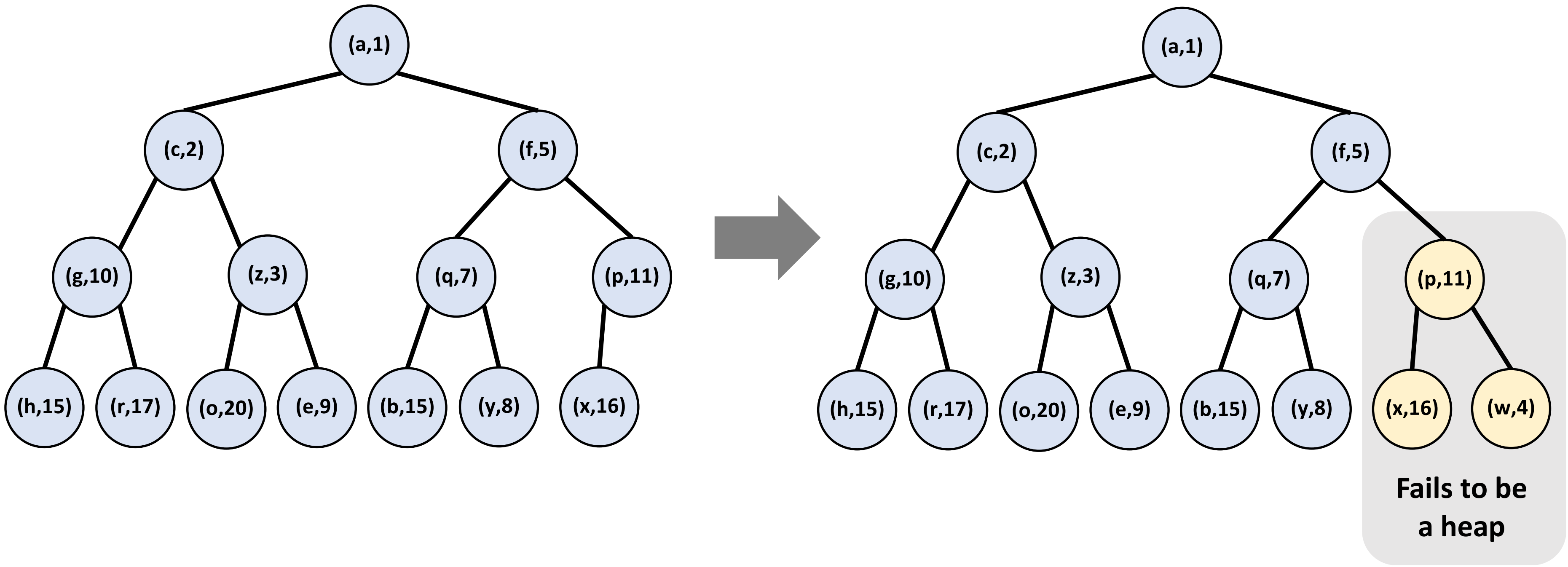
Implementing DecreaseKey



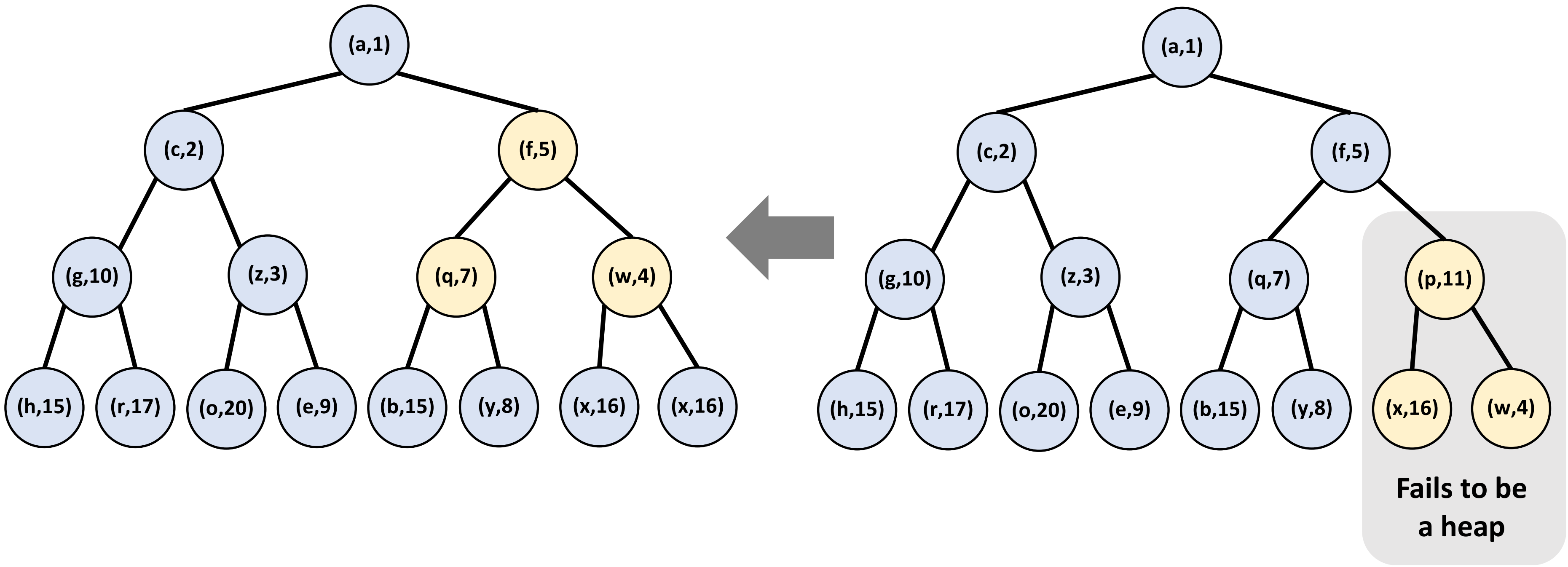
Implementing DecreaseKey

- Two steps:
  - Change the key
  - Repair the heap-order (heapify up)

Implementing Insert



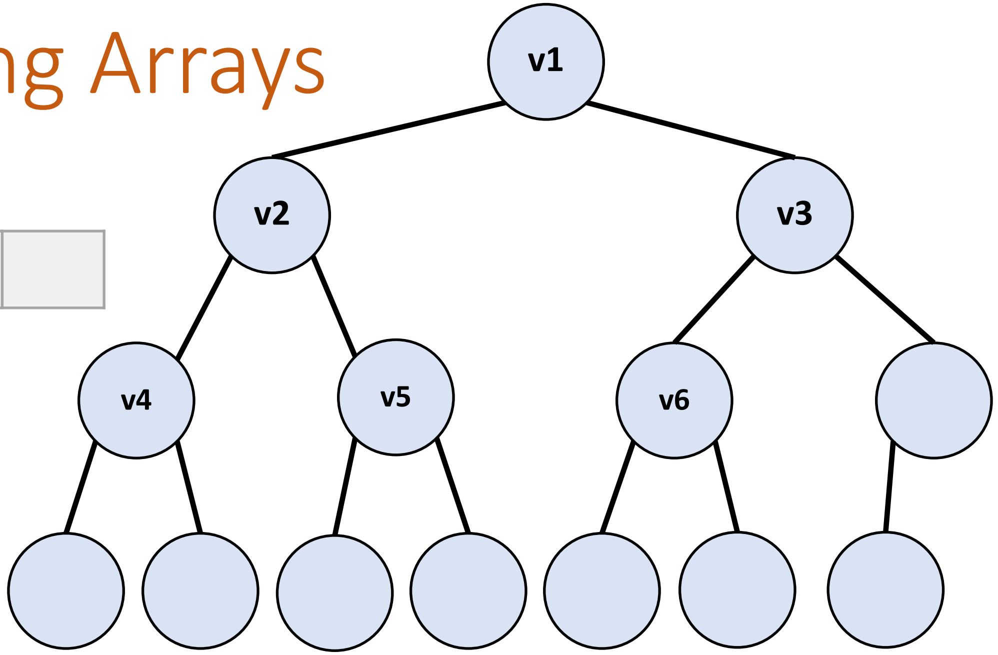
Implementing Insert



Implementing Insert

- Two steps:
  - Put the new key in the last location
  - Repair the heap-order (heapify up)

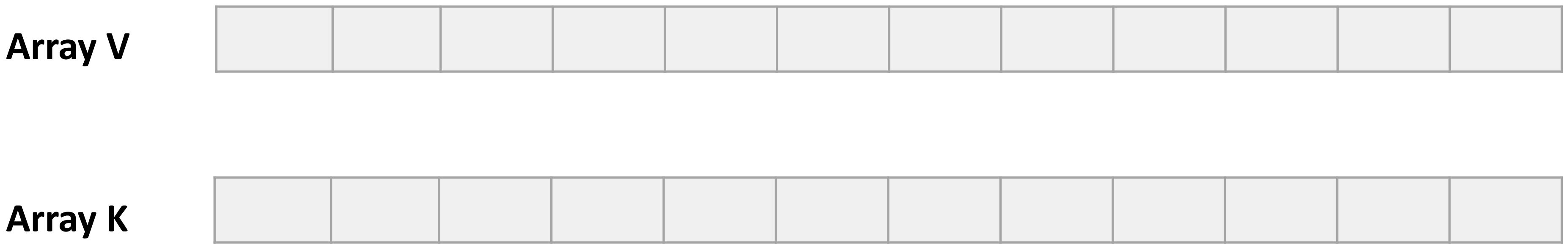
Implementation of Binary Tree Using Arrays



Array V	**v1**	**v2**	**v3**	**v4**	**v5**	**v6**						

- Maintain an array V holding the values (in order of top to bottom, followed by left to right)
- For any node i in the binary tree, what is the index of
  - LeftChild(i) =
  - RightChild(i) =
  - Parent(i) =
- Draw the tree on the array above.

Implementation of Priority Queue Using Arrays



- Maintain an array V holding the (key,value) at each node the binary tree
- Maintain an array K mapping keys index
  - Can find the value for a given key in $O(1)$ time

Binary Heaps

- **Heapify:**
  - O(1) time to fix a single triple
  - With n keys, might have to fix O(log n) triples
  - Total time to heapify is O(log n)

- **Lookup** takes O(1) time
- **ExtractMin** takes O(log n) time
- **DecreaseKey** takes O(log n) time
- **Insert** takes O(log n) time

Implementing Dijkstra with Heaps

```
Dijkstra(G = (V,E,{ℓ(e)}, s):
  Let Q be a new heap
  Let parent[u]←⊥ for every u
  Insert(Q,s,0), Insert(Q,u,∞) for every u != s

  While (Q is not empty):
    (u,d[u]) ← ExtractMin(Q)

    For ((u,v) in E):
      d[v] ← Lookup(Q,v)
      If (d[v] > d[u] + ℓ(u,v)):
        DecreaseKey(Q,v,d[u] + ℓ(u,v))
        parent[v]←u

  Return (d, parent)
```

Lookup takes O(1) time
ExtractMin takes O(log n) time
DecreaseKey takes O(log n) time
Insert takes O(log n) time

How much time does Dijkstra take?

Dijkstra Summary:

- **Dijkstra's Algorithm** solves **single-source shortest paths** in non-negatively weighted graphs
  - Algorithm can fail if edge weights are negative!

- **Implementation:**
  - A **priority queue** supports all necessary operations
  - Implement priority queues using **binary heaps**
  - Overall running time of Dijkstra: $O(m \log n)$

- **Compare to BFS**