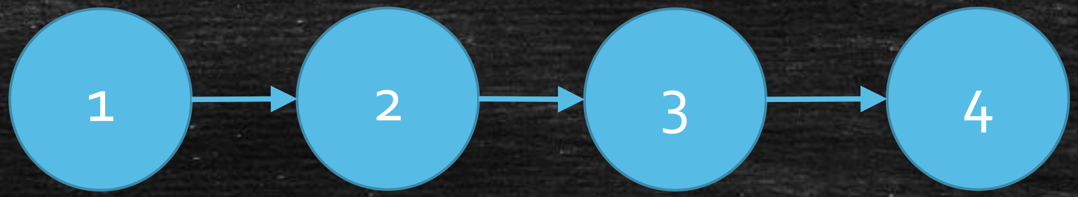
Shortest Path

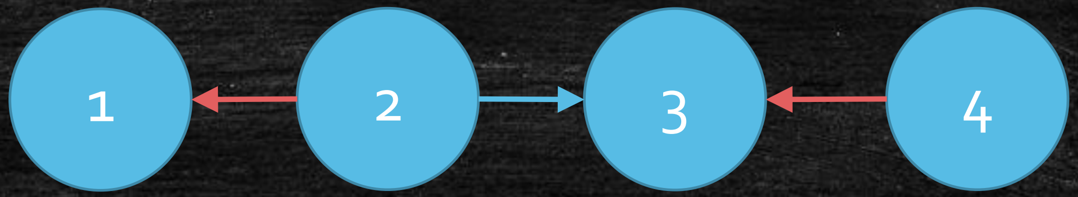
BFS and Dijkstra

What is path?

- Today we discuss directed graphs!
- 1 to 4 Path



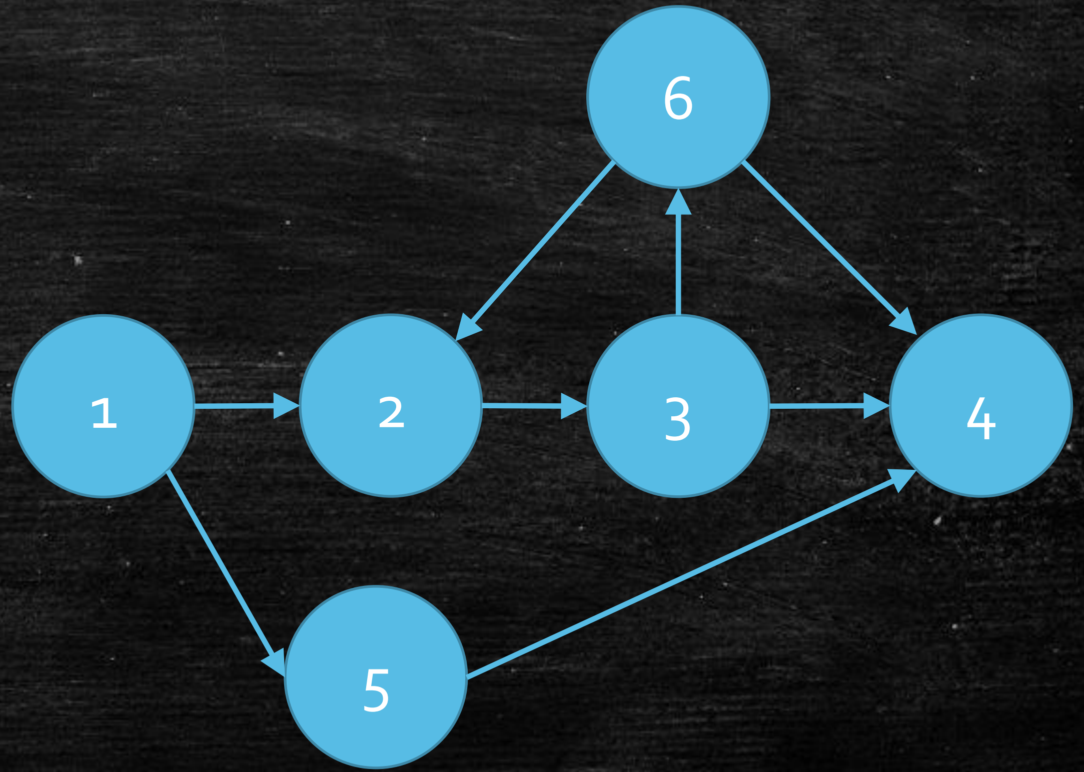
- Not a 1 to 4 path



- Length: the number of arcs in the path.

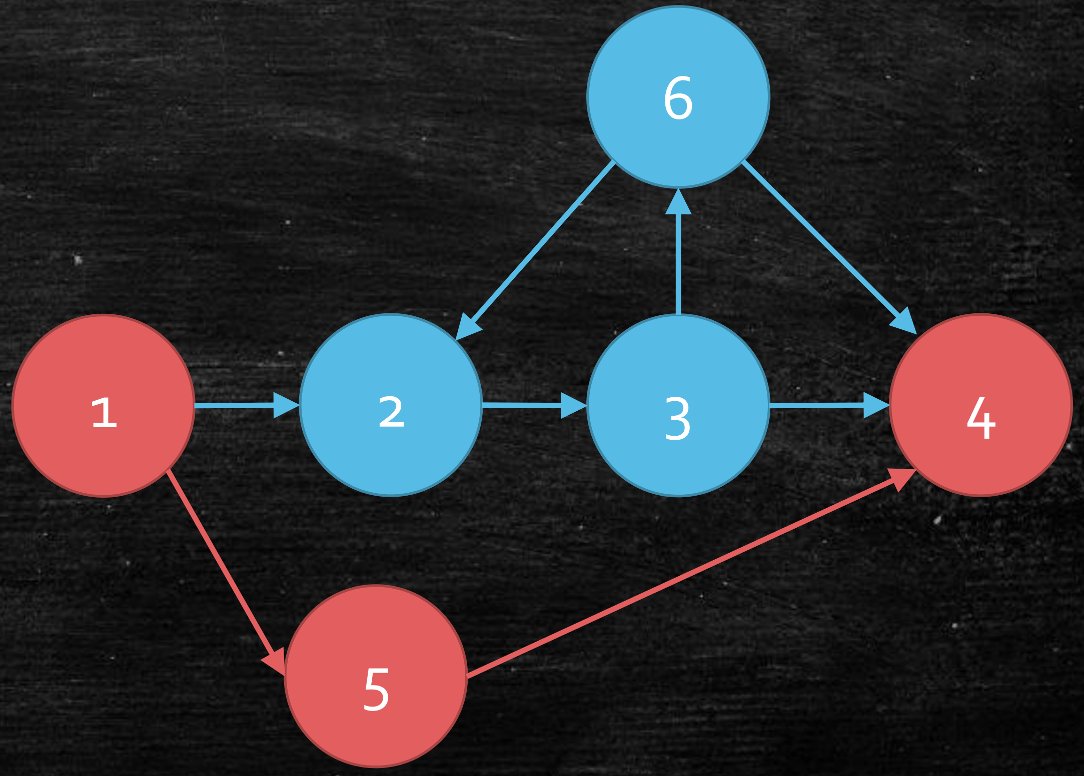
Vertices Distance

- How to define distance?
- $d(u, v)$: the length of shortest path from u to v.



Vertices Distance

- How to define distance?
- $d(u, v)$: the length of shortest path from u to v.
- $d(1,4) = 2$

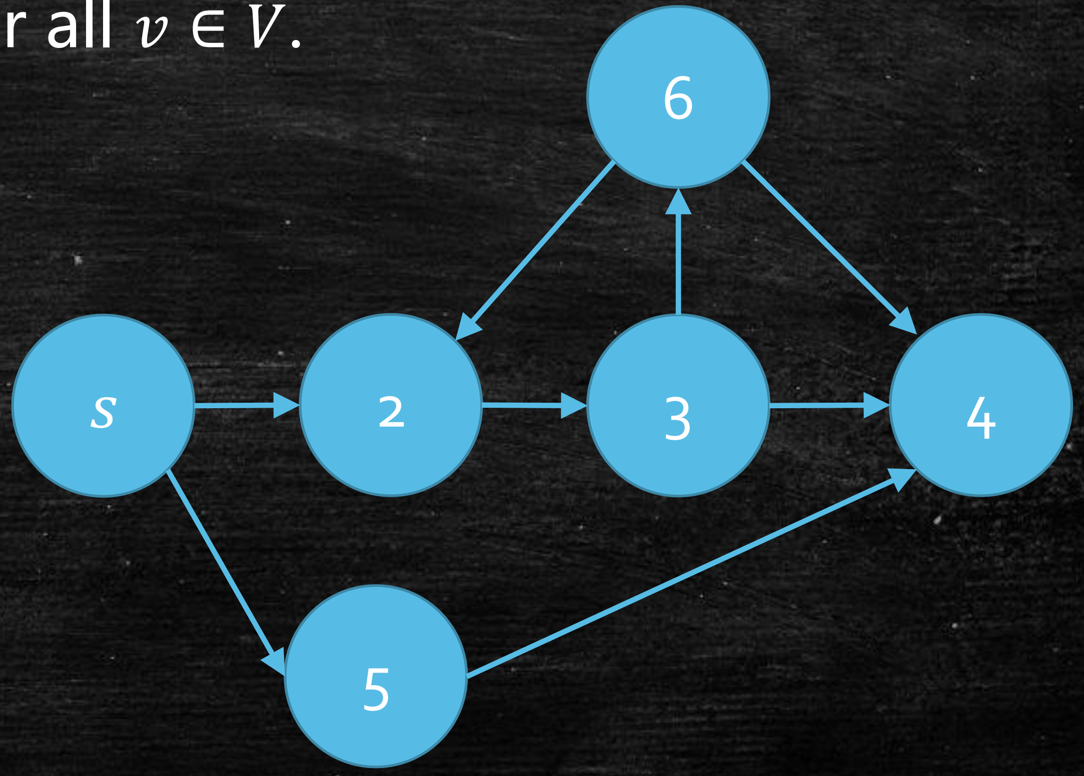


Single-Source Shortest Path Problems

- **Input:** A directed graph $G(V,E)$, represented by an Adjacent Matrix, and a source vertex s.
- **Output:** Distance $\mathrm{d}(s, v)$, for all $v \in V$.

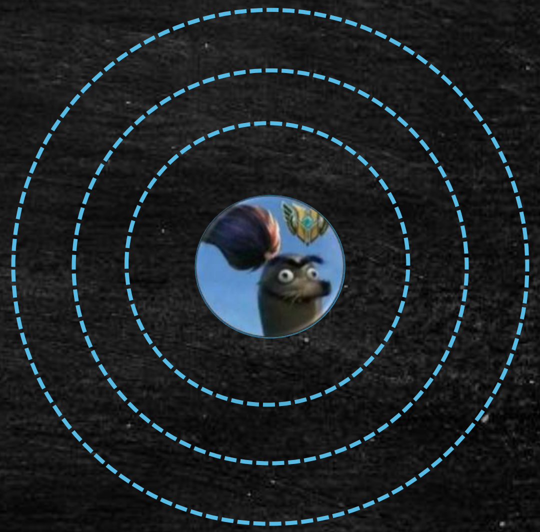
Single-Source Shortest Path Problems

- **Input:** A directed graph $G(V,E)$, represented by an Adjacent Matrix, and a source vertex s.
- **Output:** Distance $\mathrm{d}(s,v)$, for all $v \in V$.



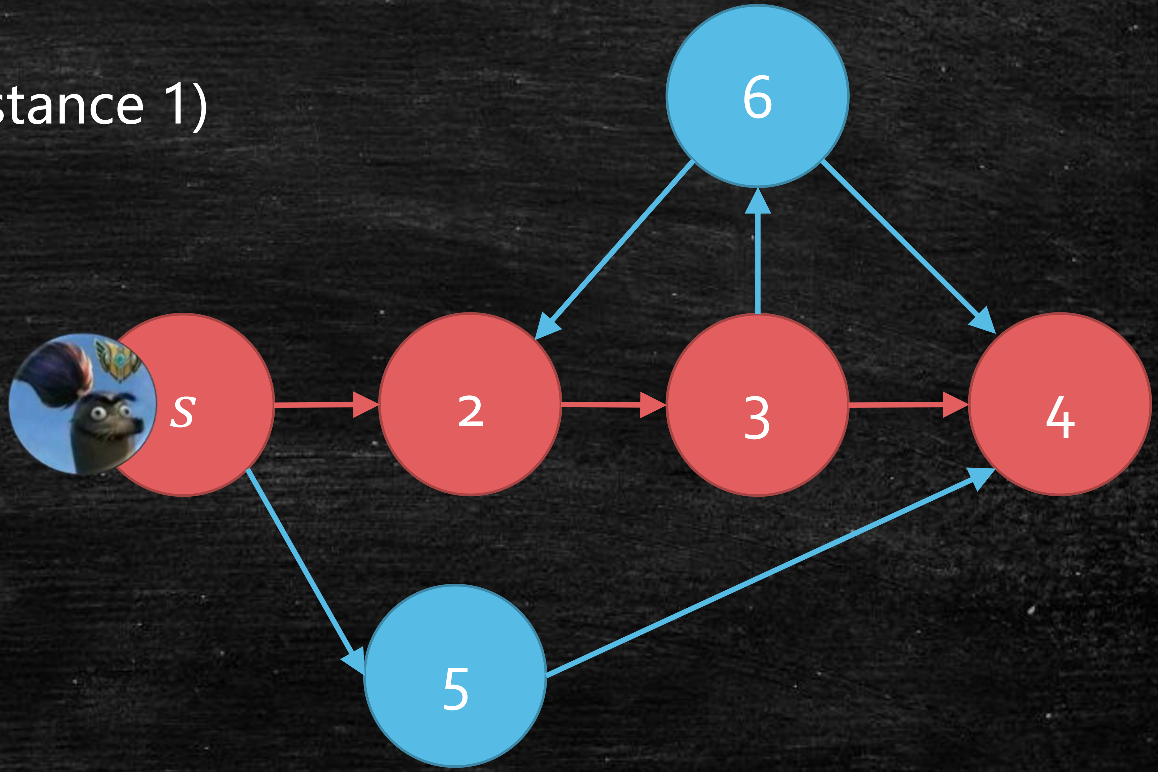
Key Idea

- **Input:** A directed graph $G(V,E)$, represented by an Adjacent Matrix, and a source vertex s.
- **Output:** Distance $\mathrm{d}(s, v)$, for all $v \in V$.
- Idea
  - Walk from s
  - Keep walking
  - Walk 1 step: Arrive distance 1 vertices
  - Walk 2 steps: Arrive distance 2 vertices
  - Walk 3 steps; Arrive distance 3 vertices
  - ……



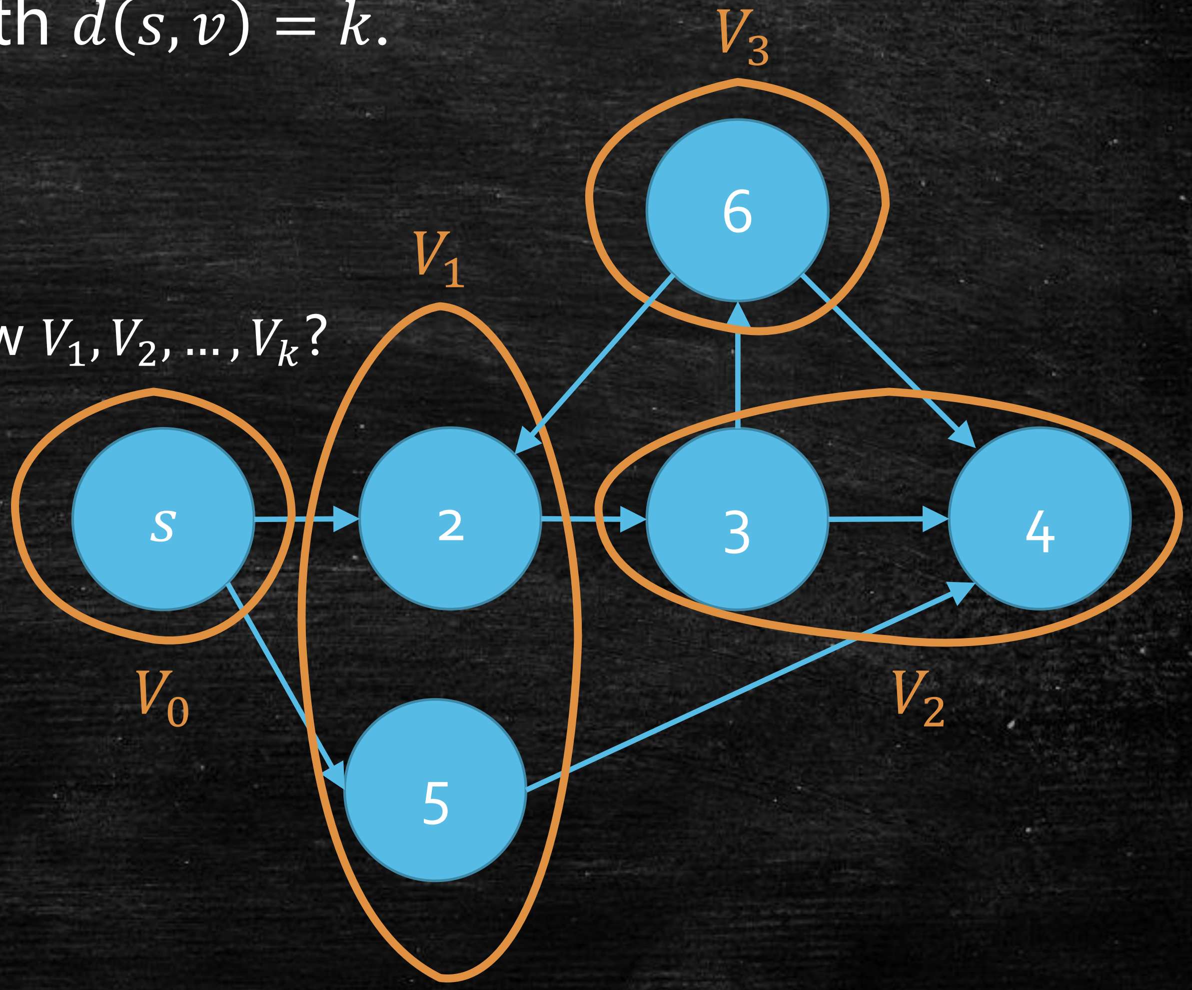
Can DFS help us?

- DFS after 4 explorations.
- Problems:
  - Vertex 5 not visited (only distance 1)
  - Arrive vertex 4 with length 3



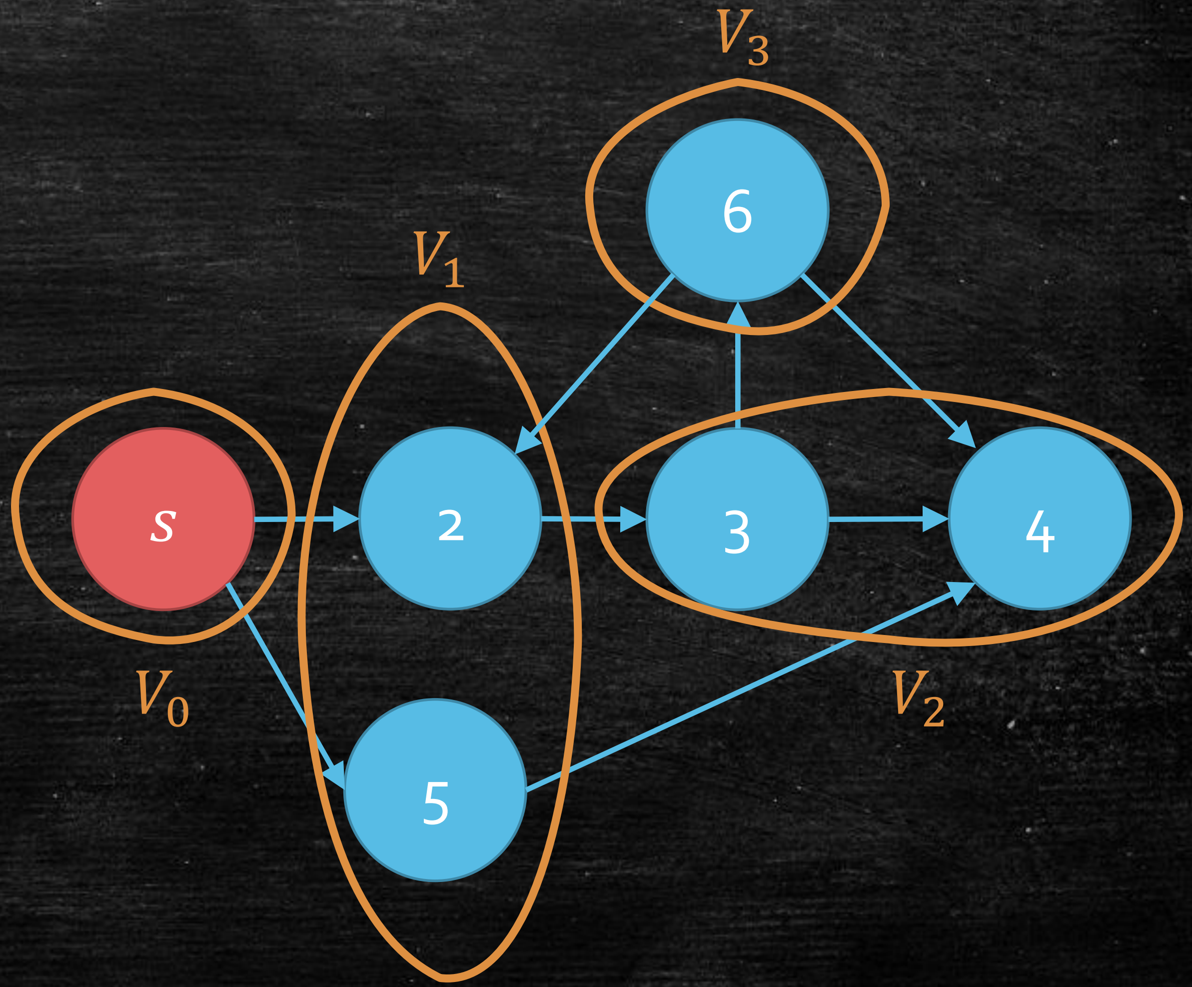
How to Implement the Idea?

- V_k: the set of vertices v with $d(s, v) = k$.
- $V_0 = \{s\}$
- Key question
  - Can we know V_{k+1}, if we know $V_1, V_2, \ldots, V_k$?
  - Yes!
  - $v \in V_{k+1}$ if and only if
    - $u \in V_k$ and (u, v) exists
    - $v \notin V_l,\ \forall l \leq k$.



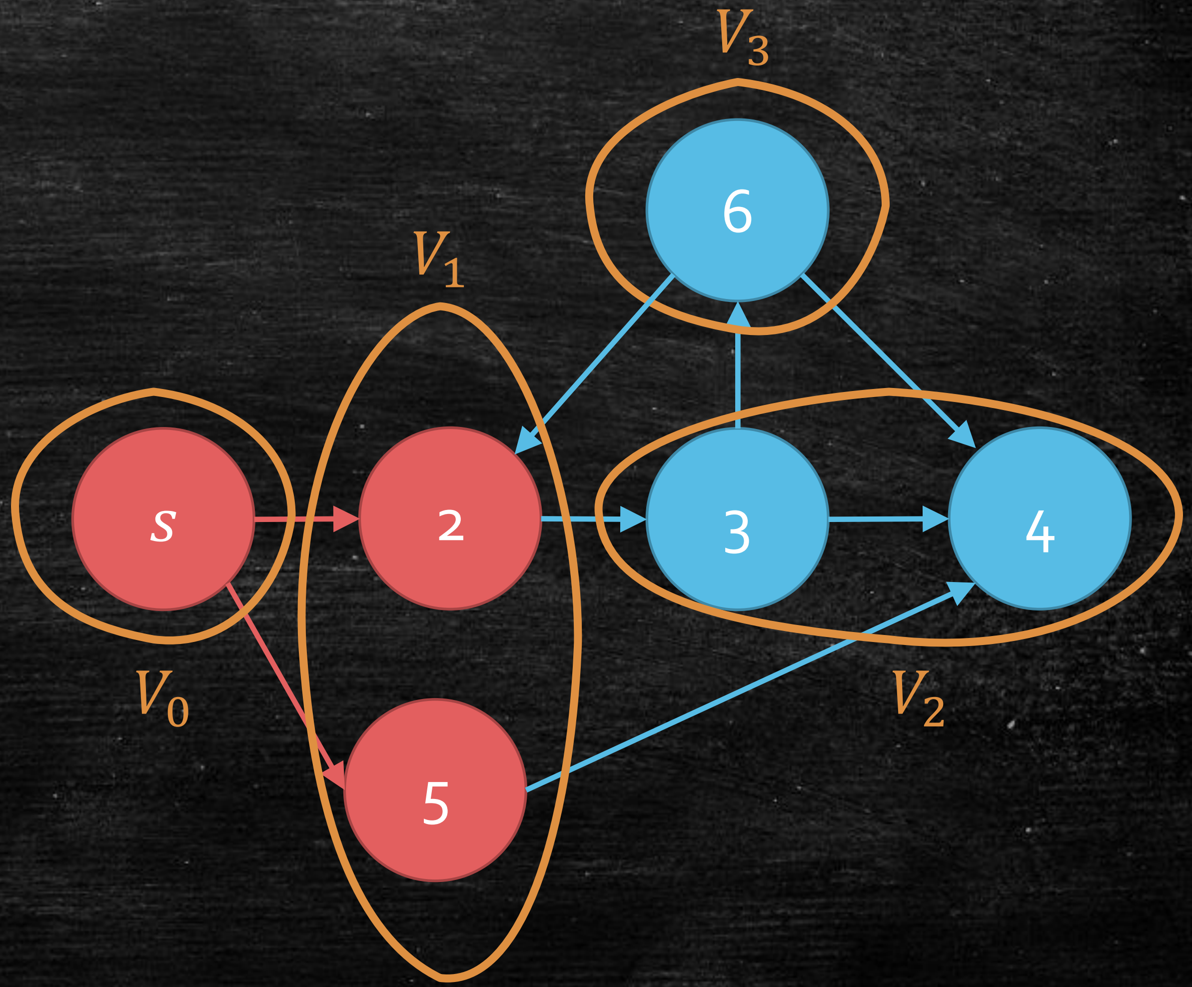
Breadth-First Search (BFS)

- A water frontier.
  - Explore s



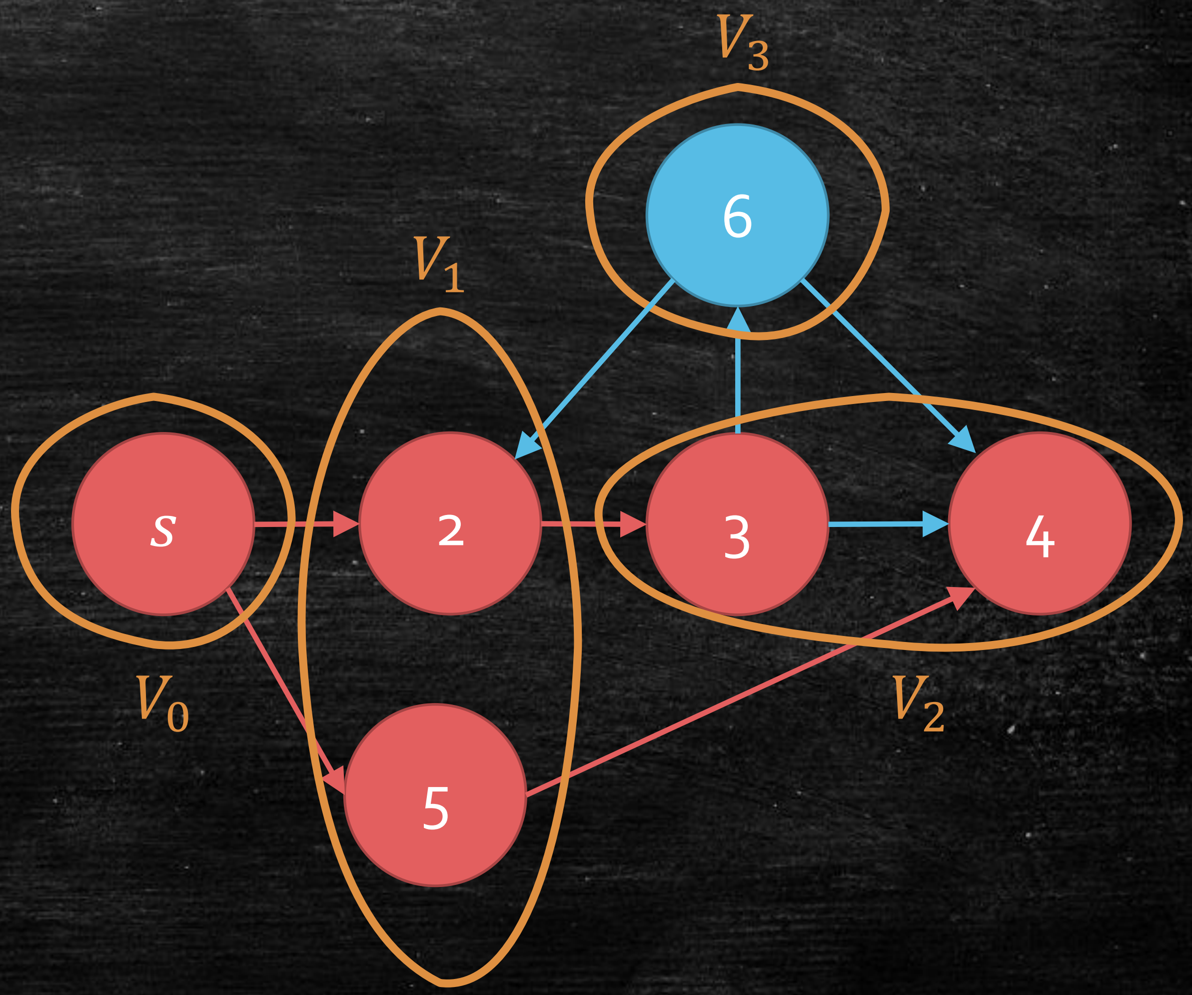
Breadth-First Search (BFS)

- A water frontier.
  - Explore s
  - Explore V_1



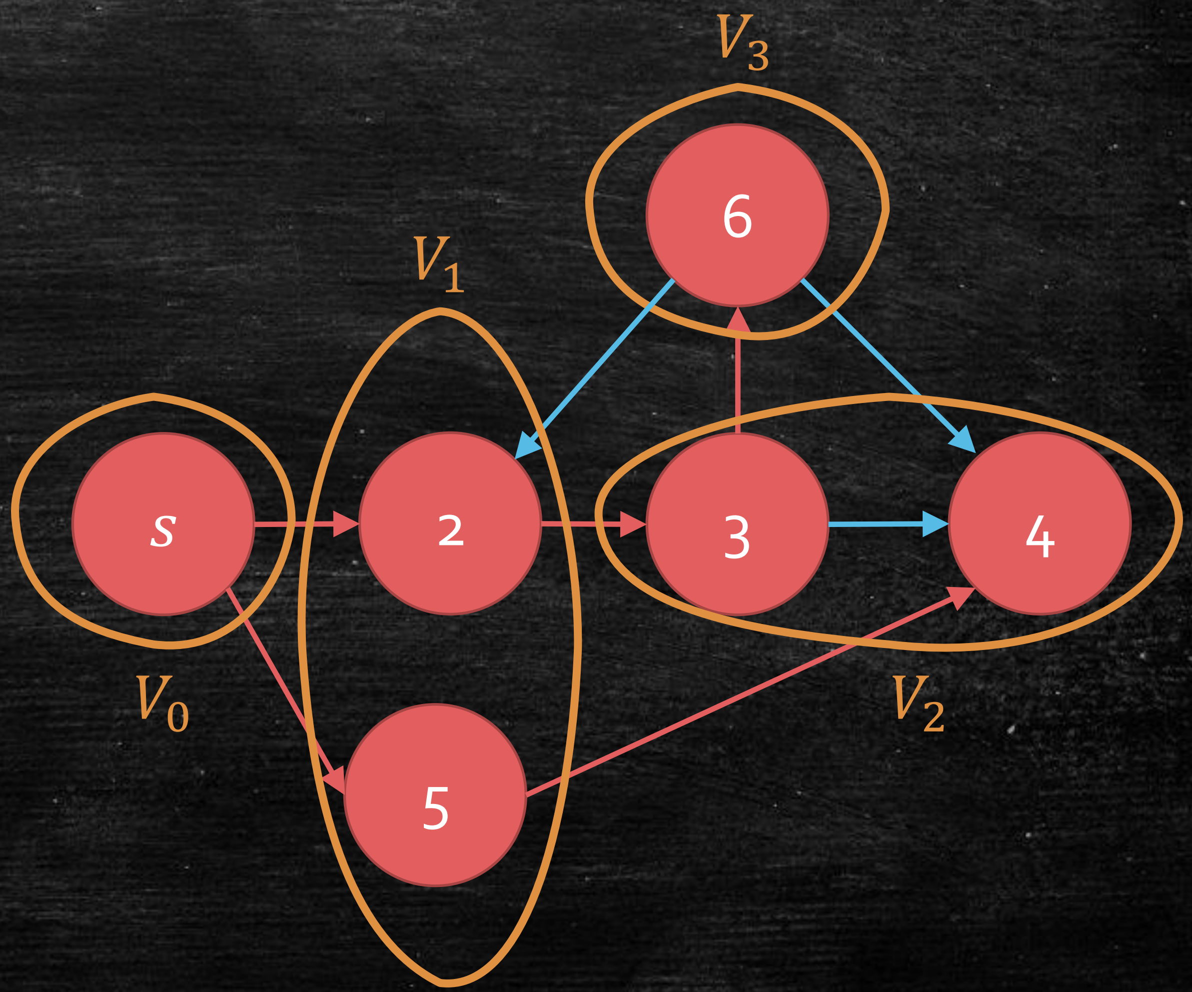
Breadth-First Search (BFS)

- A water frontier.
  - Explore s
  - Explore V_1
  - Explore V_2



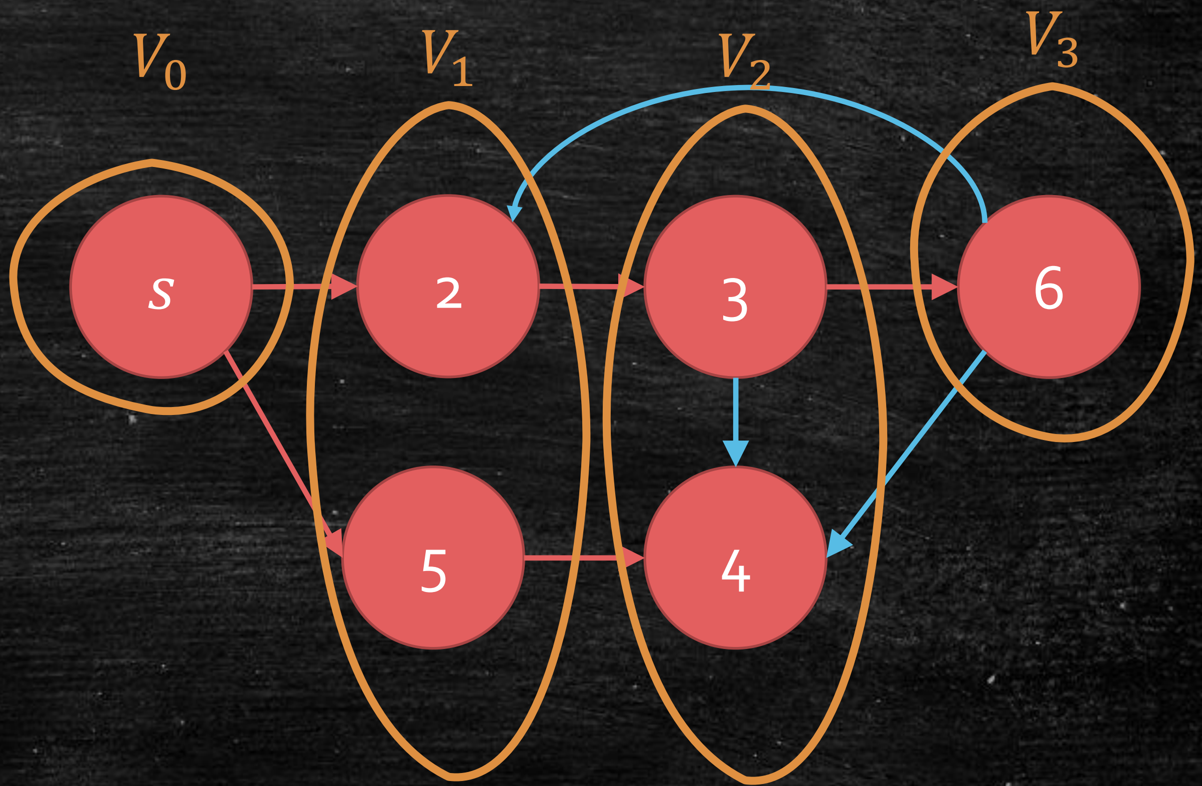
Breadth-First Search (BFS)

- A water frontier.
  - Explore s
  - Explore V_1
  - Explore V_2
  - …



BFS Tree

- A water frontier.
  - Explore s
  - Explore V_1
  - Explore V_2
  - …
- The layer of the vertex
- = The distance from s



How to program?

Breadth First Search

Function bfs(G, s)

  for each v $\in V$ $marked[v] \leftarrow [0]$

  $i \leftarrow 0$ (layer counter)

  $V_0 \leftarrow \{s\}$

  while V_i is not empty

    for each $u \in V_i$

      for each $(u, v) \in E$

        if $marked[v] = false$

          $marked[v] \leftarrow true$

          Add v into V_{i+1}

    $i \leftarrow i + 1$

Running Time? $\boldsymbol{O(V+E)}$

Each edge can be only checked once

Each vertex can be only marked once

Output Path?

- What if we want to output the shortest path?
- Solution
  - Maintain an array $pre[v]$ means who v is explored by.

Breadth First Search

Function bfs(G, s)
    for each $\mathrm{v} \in V$ $marked[v] \leftarrow [0]$
    $i \leftarrow 0$ (layer counter)
    $V_0 \leftarrow \{s\}$
    while V_i is not empty
        for each $u \in V_i$
            for each $(u, v) \in E$
                if $marked[v] = false$
                    $marked[v] \leftarrow true$
                    Add v into V_{i+1}
                    $pre[v] \leftarrow u$
        $i \leftarrow i + 1$

DFS vs BFS

	DFS	BFS
Detecting Cycles	YES	NO
Topological Ordering	YES	NO
Finding CCs	YES	YES
Finding SCCs	YES	NO
Shortest Path	NO	YES

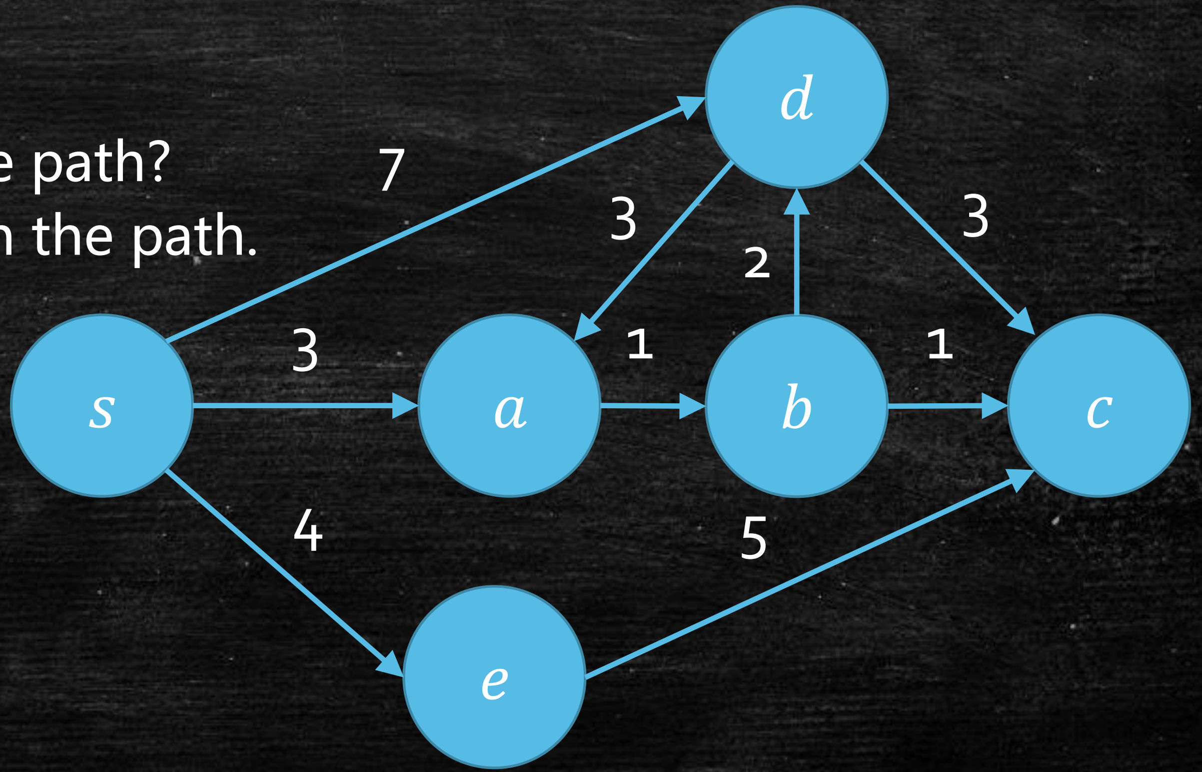
- Hard to separate cross edge and back edges in BFS
- Finish time is meaningful in BFS

What if edges have length?

Dijkstra Algorithm

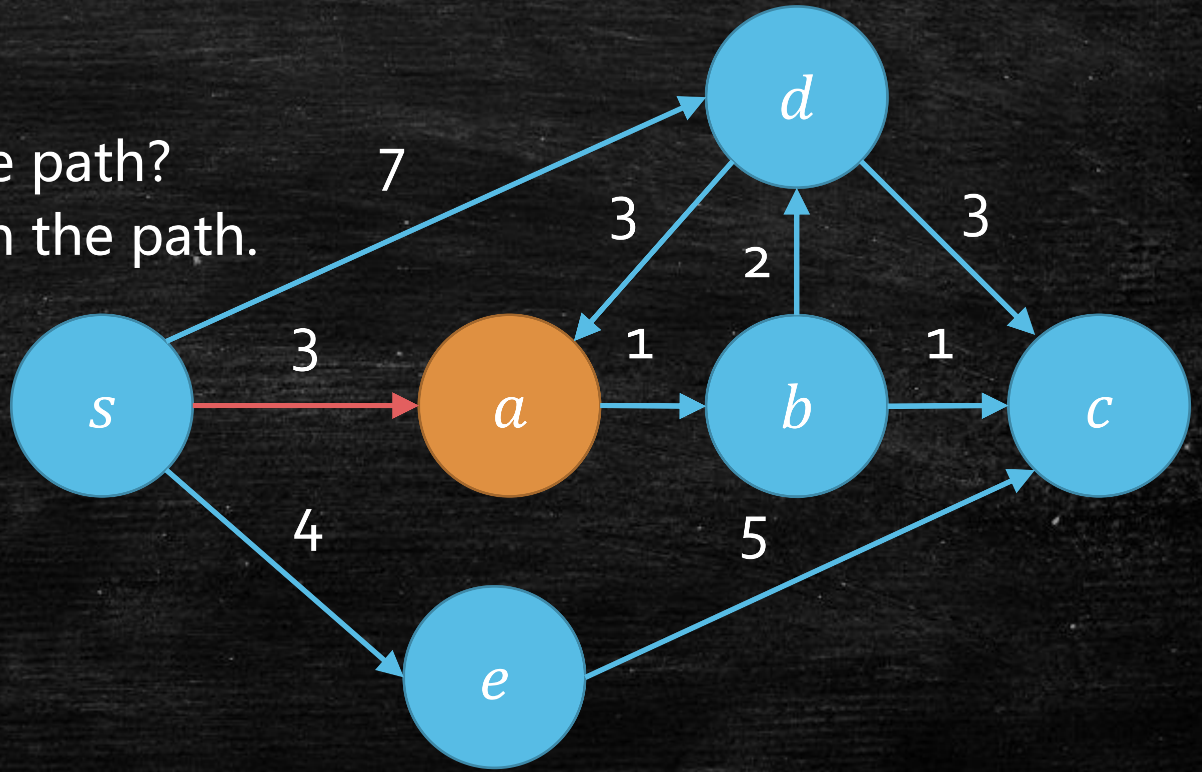
Single-Source Shortest Path for Weighted graphs

- New Input!
  - $w(u, v)$ for each edge (u, v)
  - Means the weight or length.
- **New Length of Path**
  - The number of edges in the path?
  - The sum of edges' length in the path.
  - Length $s \to e \to c = 9$
  - Length $s \to a \to b \to c = 5$



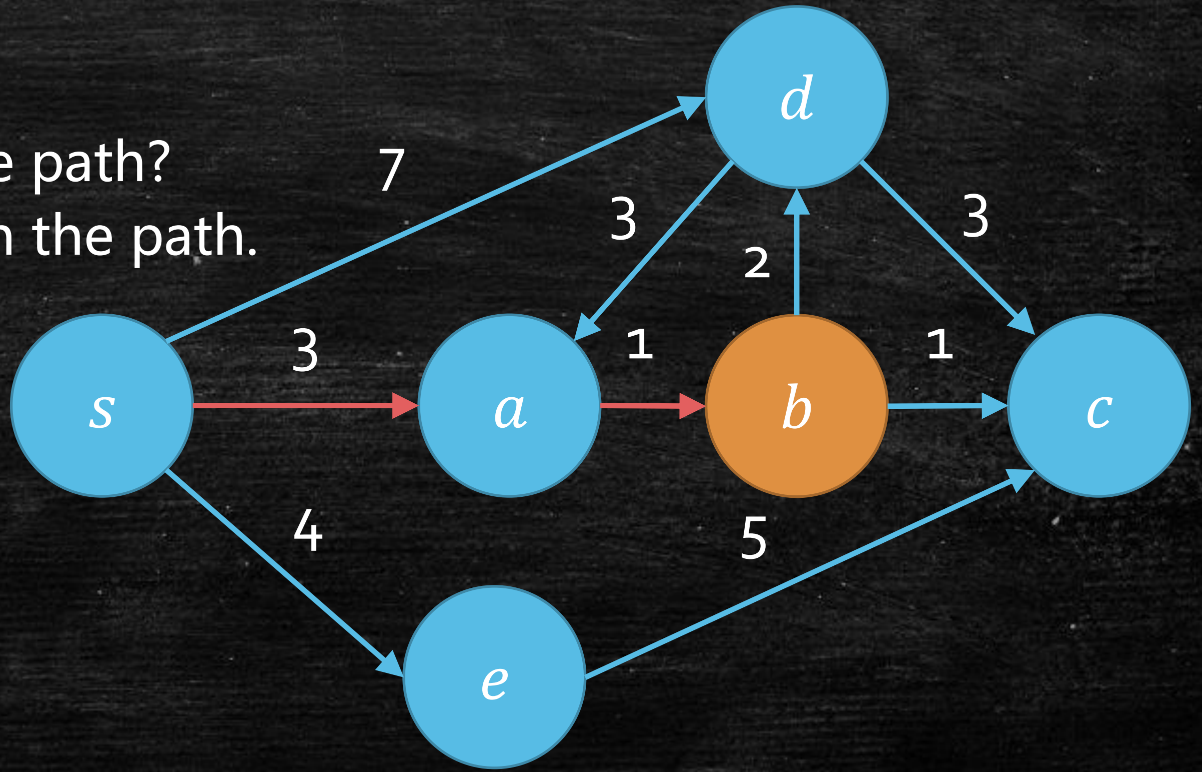
Single-Source Shortest Path for Weighted graphs

- New Input!
  - $w(u, v)$ for each edge (u, v)
  - Means the weight or length.
- **New Length of Path**
  - The number of edges in the path?
  - The sum of edges' length in the path.
  - Length $s \to e \to c = 9$
  - Length $s \to a \to b \to c = 5$



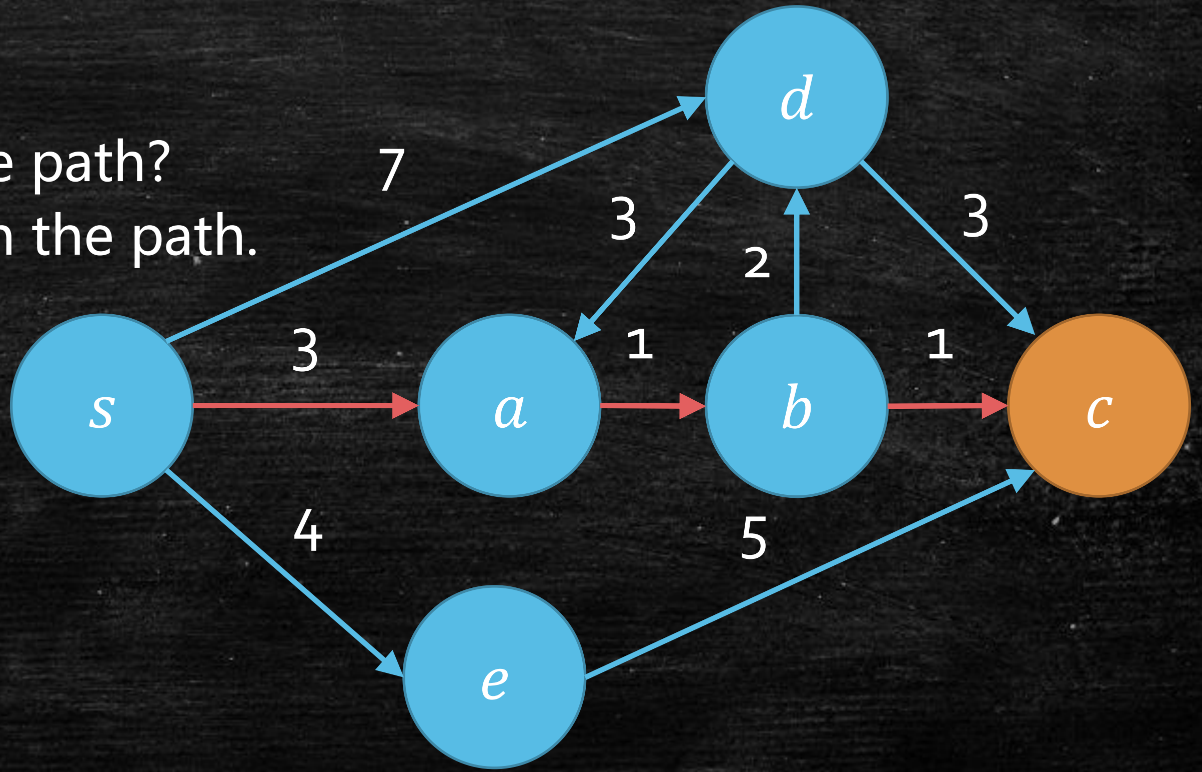
Single-Source Shortest Path for Weighted graphs

- New Input!
  - $w(u, v)$ for each edge (u, v)
  - Means the weight or length.
- **New Length of Path**
  - The number of edges in the path?
  - The sum of edges' length in the path.
  - Length $s \to e \to c = 9$
  - Length $s \to a \to b \to c = 5$



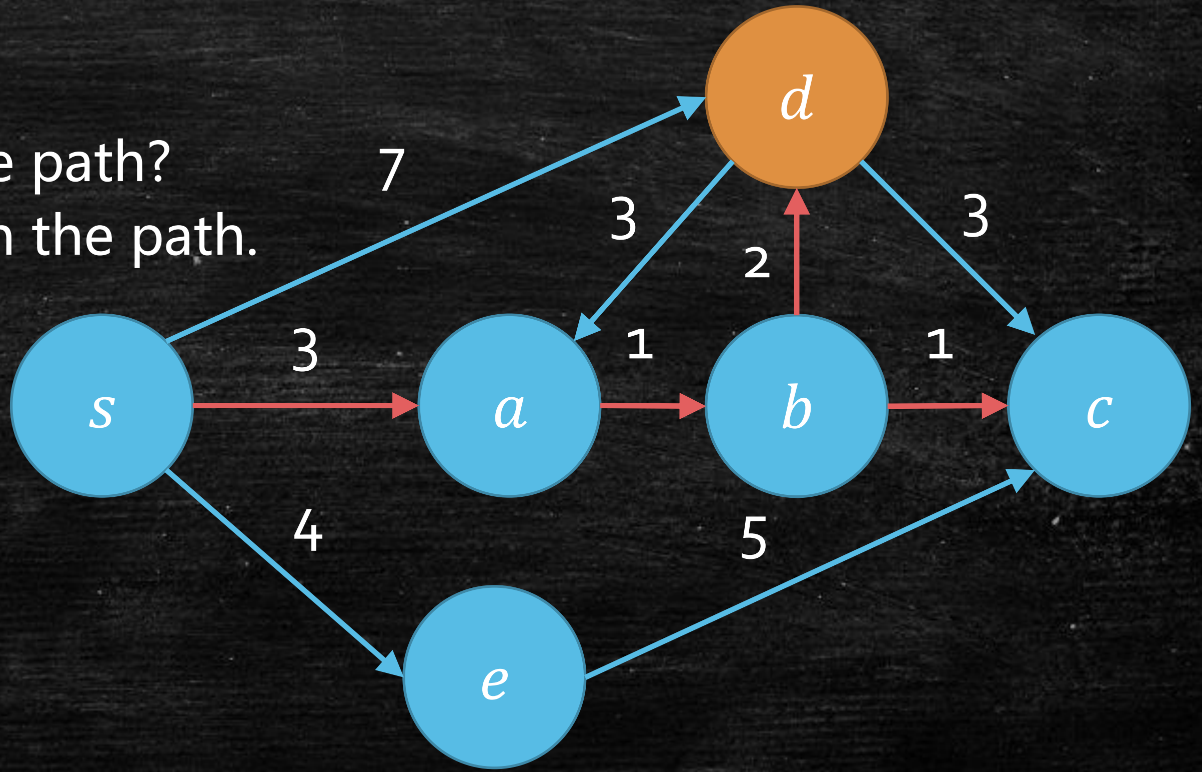
Single-Source Shortest Path for Weighted graphs

- New Input!
  - $w(u, v)$ for each edge (u, v)
  - Means the weight or length.
- **New Length of Path**
  - The number of edges in the path?
  - The sum of edges' length in the path.
  - Length $s \to e \to c = 9$
  - Length $s \to a \to b \to c = 5$



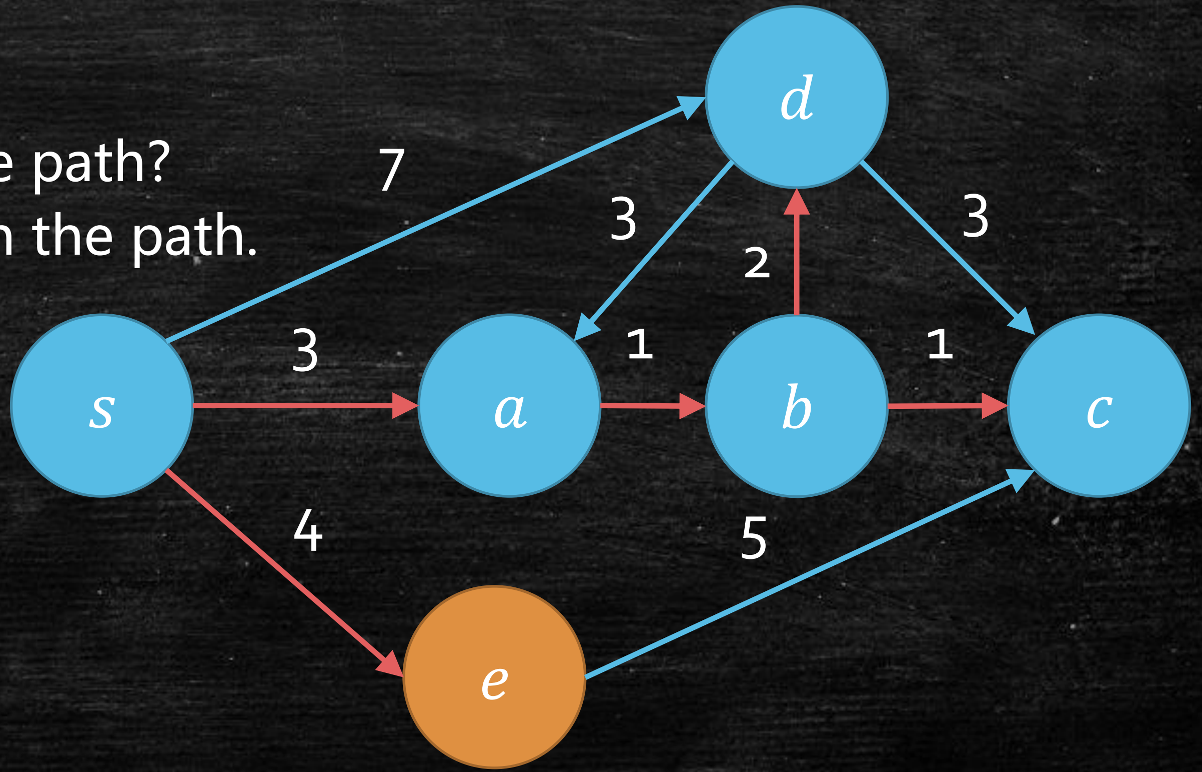
Single-Source Shortest Path for Weighted graphs

- New Input!
  - $w(u, v)$ for each edge (u, v)
  - Means the weight or length.
- **New Length of Path**
  - The number of edges in the path?
  - The sum of edges' length in the path.
  - Length $s \to e \to c = 9$
  - Length $s \to a \to b \to c = 5$



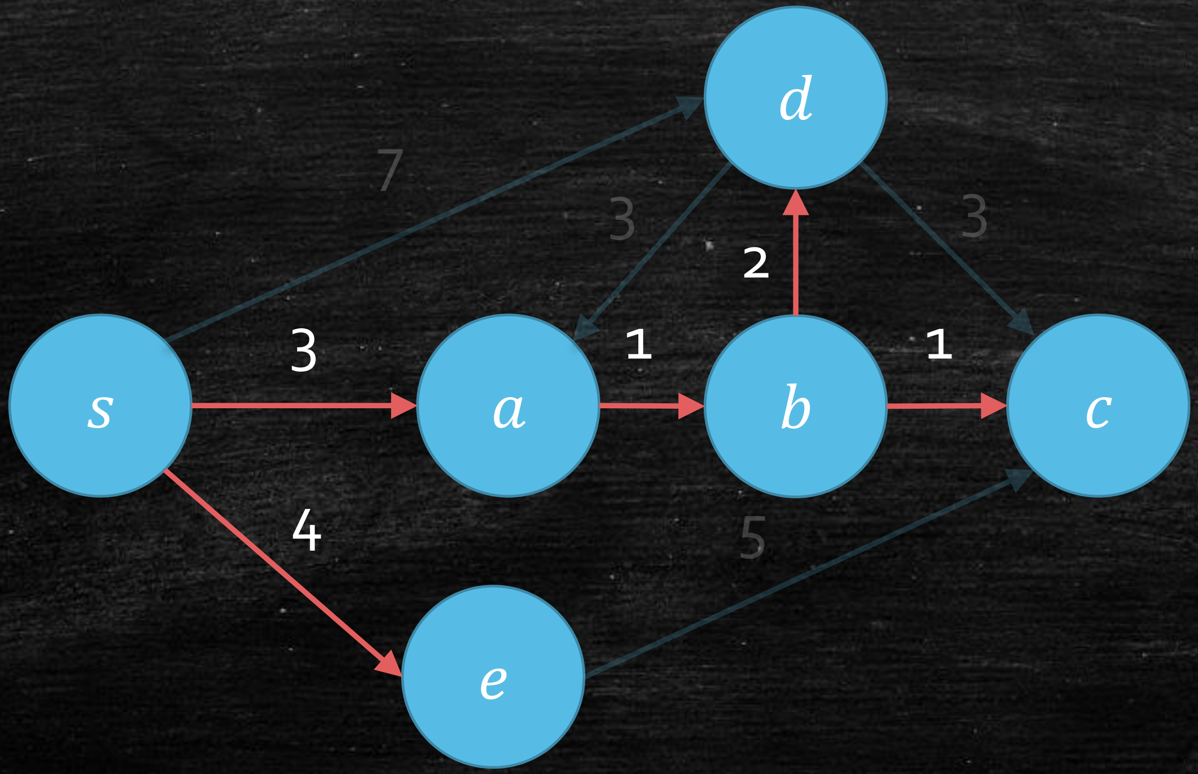
Single-Source Shortest Path for Weighted graphs

- New Input!
  - $w(u, v)$ for each edge (u, v)
  - Means the weight or length.
- **New Length of Path**
  - The number of edges in the path?
  - The sum of edges' length in the path.
  - Length $s \to e \to c = 9$
  - Length $s \to a \to b \to c = 5$



Rough Observation

- Can we use the BFS idea?
- Do all shortest paths form a tree?

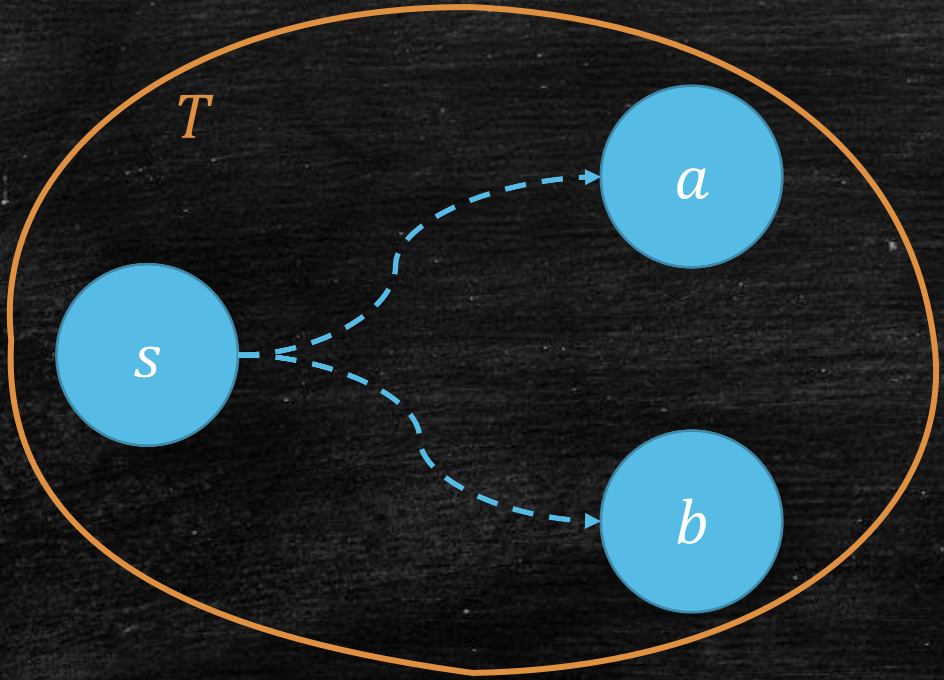


Try to prove!

- Question: do we always have a **Shortest Path Tree** for a general graph?
- **Shortest Path Tree (SPT)**
  - $v \in T$, $s \rightarrow v$ path in T is the shortest path in G.
- Start point
  - $\{s\}$ is a SPT.
- Next
  - Can we always explore current SPT until all vertices are included?

Key Task

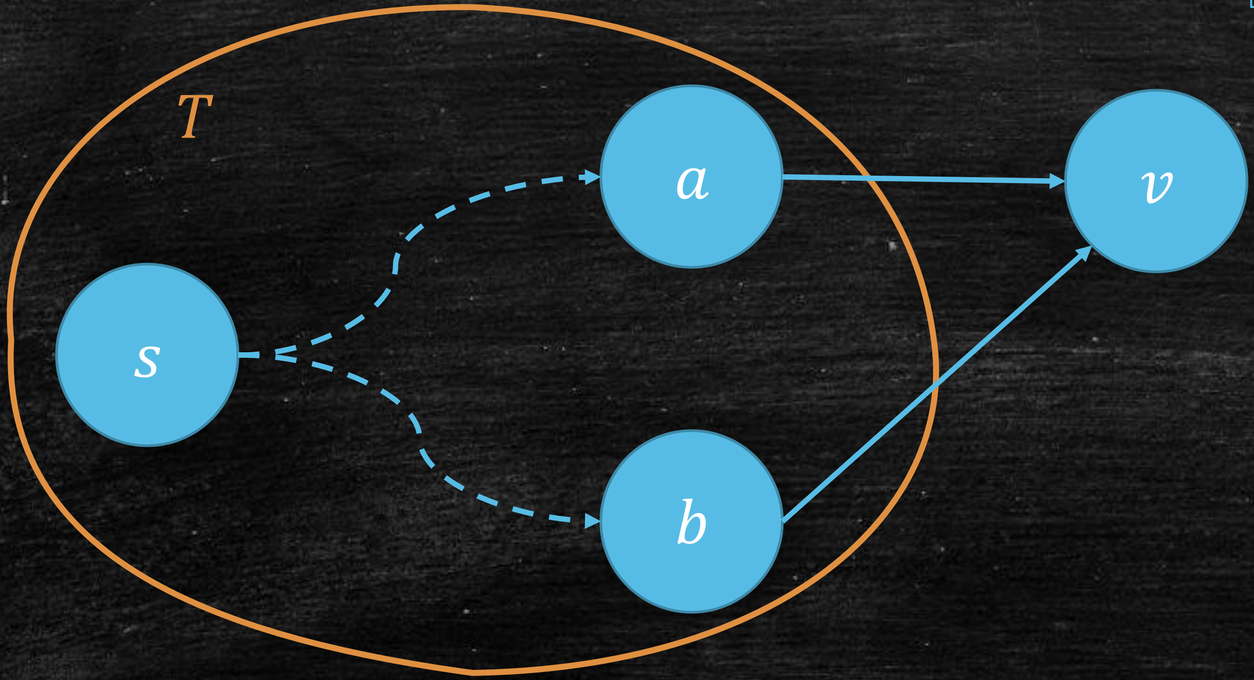
- **Given:** a small SPT (not contains all the vertices)
- **Want:** a larger SPT



Key Task

- Can we explore v into T?

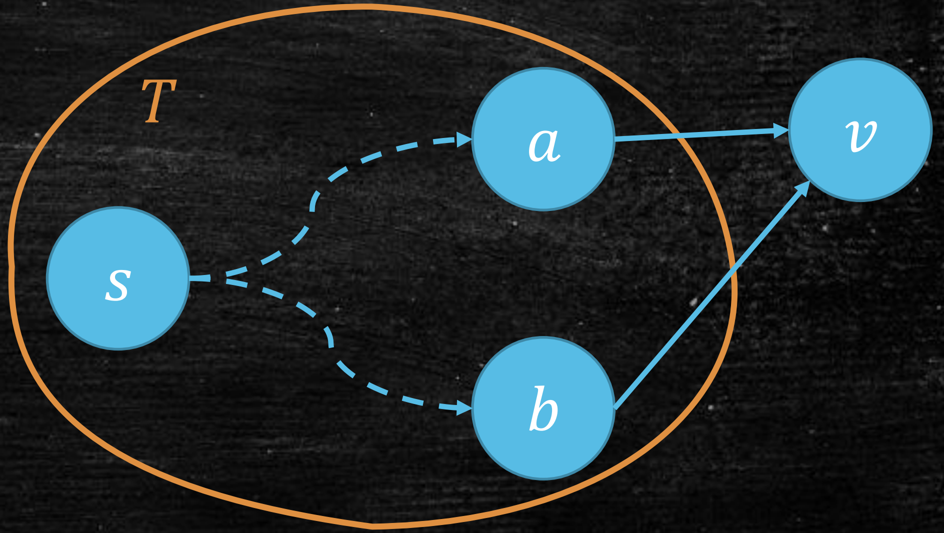
- **Given:** a small SPT (not contains all the vertices)
- **Want:** a larger SPT



Key Task

- Property of **SPT**
  - True distance: $dist(u) = d(s,u)$
  - Local distance: $dist_T(u)$ only allows to go through T.
  - Basic property $dist_T(u)=dist(u)$ if $u \in T$
- $dist_T(v)$: shortest T-path $s \to T \to v$
- $dist_T(v) = \min_{u \in T} dist_T(v) + d(u,v)$

- **Given:** a small SPT (not contains all the vertices)
- **Want:** a larger SPT
- Can we explore v into T?

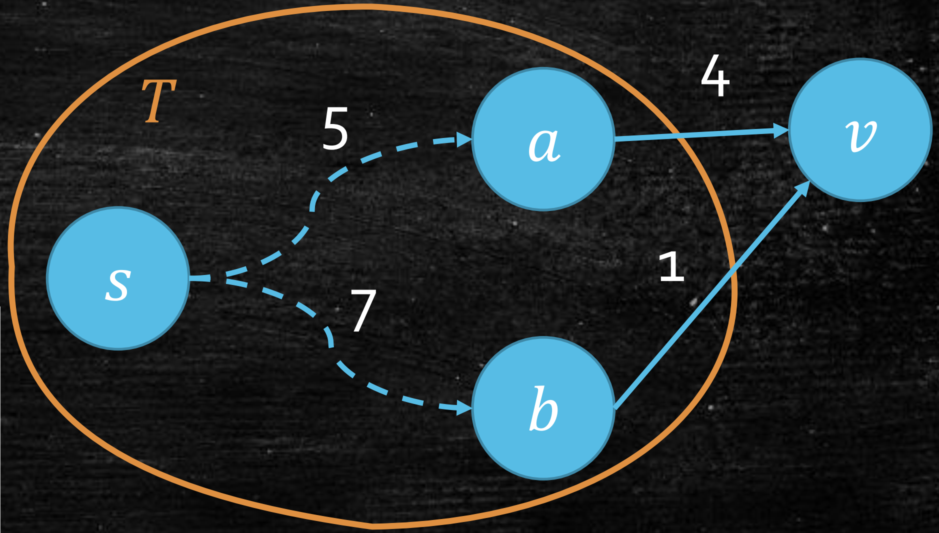


Key Task

- Facts for T
  - True distance: $dist(v) = d(s, v)$
  - Local distance: $dist_T(v)$ only allows to go through vertices in T.
  - Basic property $dist_T(v) = dist(v)$ if $v \in T$
- $dist_T(v)$: shortest T-path $s \to T \to v$
- $dist_T(v) = \min_{u \in T} dist_T(u) + d(u, v)$

- **Given:** a small SPT (not contains all the vertices)
- **Want:** a larger SPT
- Can we explore v into T?

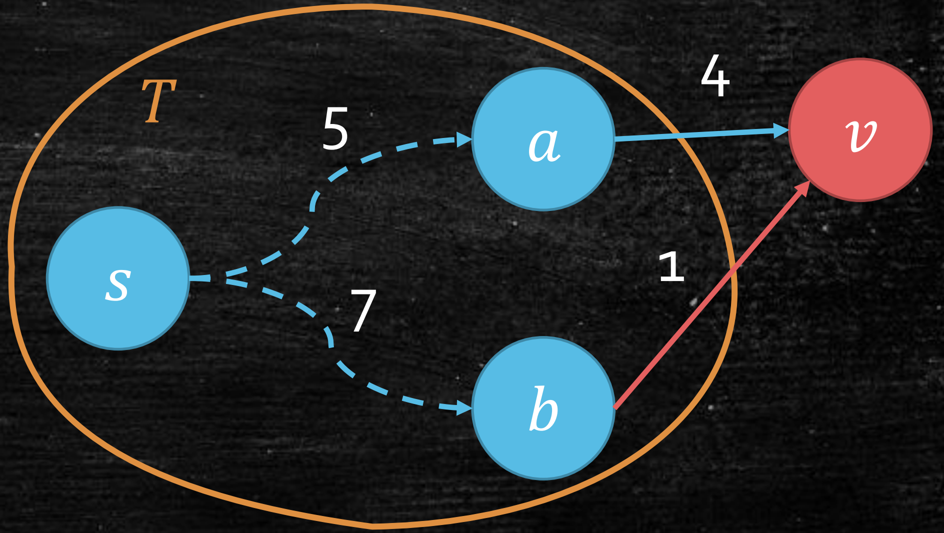
- $s \to a \to v = 9$
- $s \to b \to v = 8$
- $dist_T(v) = 8$



Key Task

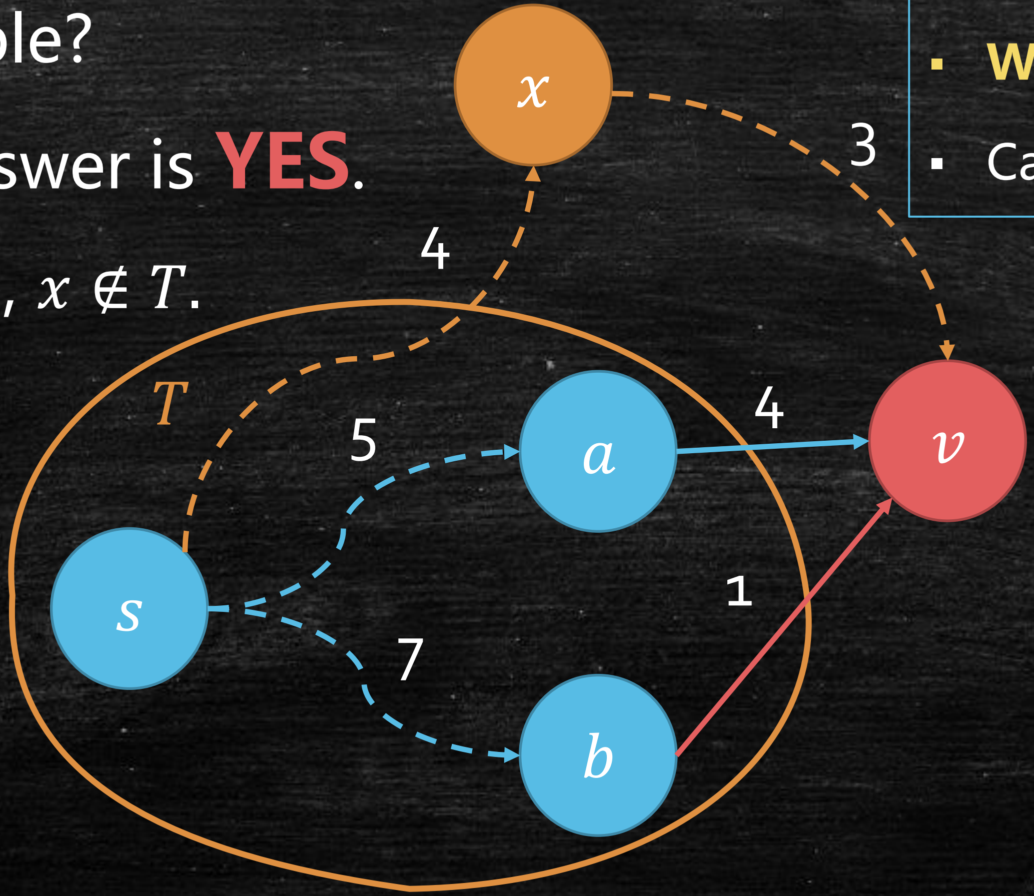
- Try to explore v into T
- Naturally, we should connect it to $\underset{u \in T}{\operatorname{argmin}}\, dist_T(u) + d(u,v)$
- Is that still an SPT?
  - Need to keep: Shortest T-path is the shortest path in G.
  - All the other vertices except v is ok
  - Shortest T-path: $dist_T(v)$
  - Key challenge: $dist_T(v) \le dist(v)$?

- **Given:** a small SPT (not contains all the vertices)
- **Want:** a larger SPT
- Can we explore v into T?



Prove $dist_T(v) \le dist(v)$

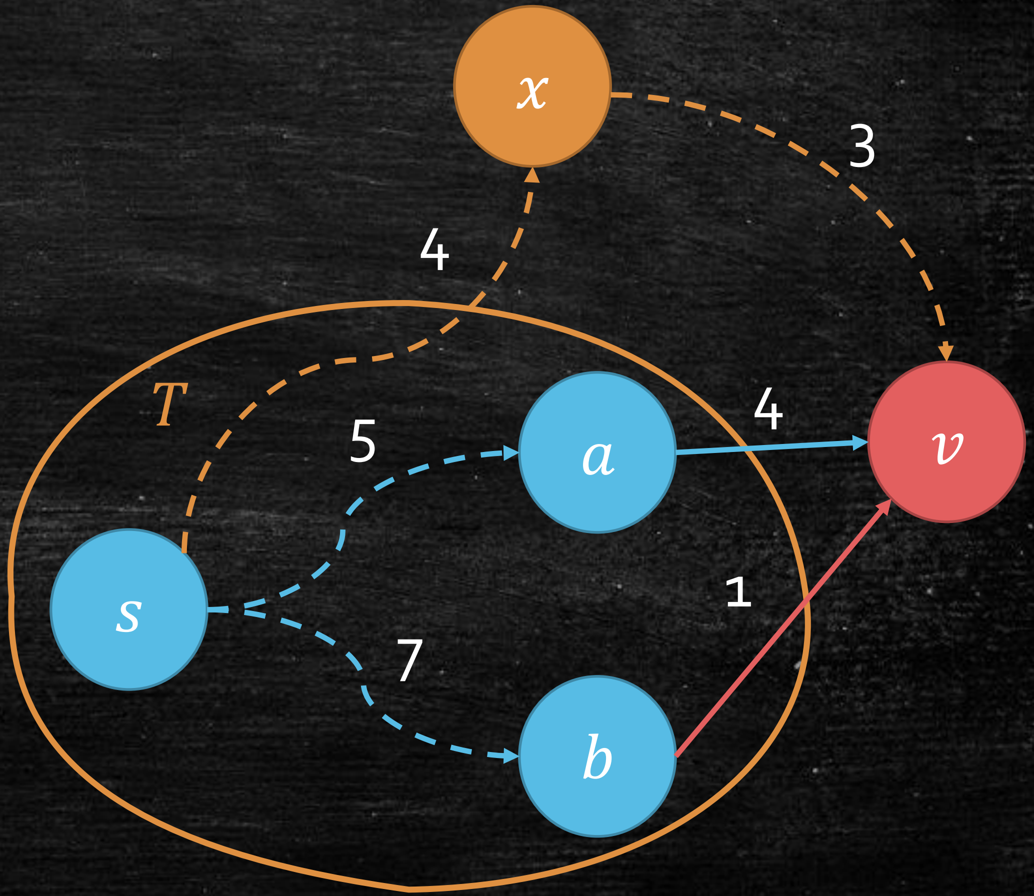
- Assume $dist_T(v) > dist(v)$
- Is that possible?
- Sorry, the answer is **YES**.
- $s \to x \to v = 7,\ x \notin T.$



- **Given:** a small SPT (not contains all the vertices)
- **Want:** a larger SPT
- Can we explore v into T?

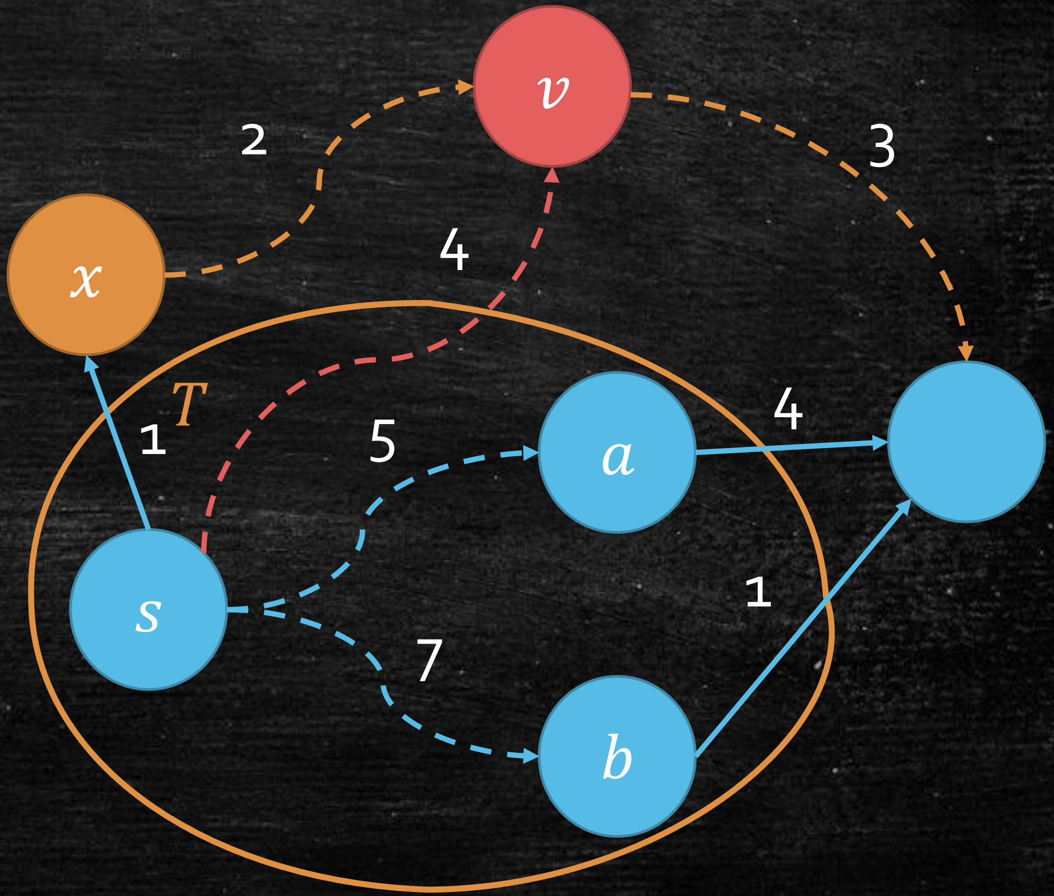
How to handle it?

- Recall BFS idea
- Each time, we explore a closest vertex.
- What happens now?
- $\boldsymbol{x}$ is a closer vertex than $\boldsymbol{v}$.
- Why not explore $\boldsymbol{x}$?
- Formalize: Choose the vertex v with **smallest** $dist_T(v)$!



Prove $dist_T(v) \le dist(v)$ **AGAIN**!

- Try to explore v into T
- Naturally, we should connect it to $\underset{u \in T}{\operatorname{argmin}} dist_T(u)$
- Assume $dist_T(v) > dist(v)$
- $x \notin T,\ s \to x \to v < dist_T(v)$
- $dist_T(x)$ is a part of $s \to x \to v$
- $dist_T(x) < dist_T(v)$
- **Contradiction!**



Yah! Success

- **Given:** a small SPT (not contains all the vertices)
- **Want:** a larger SPT
- Can we explore v into T?
- Yes!
- We can find $v = \underset{u \in T}{\operatorname{argmin}}\, dist_T(u)$ to explore! (Closest)
- Finally, we can get SPT that contains all vertices!
  - Assume s can arrive all vertices

Dijkstra Algorithm

Dijkstra$(G = (V, E), s)$

1. Initialize

- $T = \{s\}$,
- $tdist[s] = 0,\ tdist[v] \leftarrow \infty$ for all v other than s.
- $tdist[v] \leftarrow w(s, v)$ for all $(s, v) \in E$.

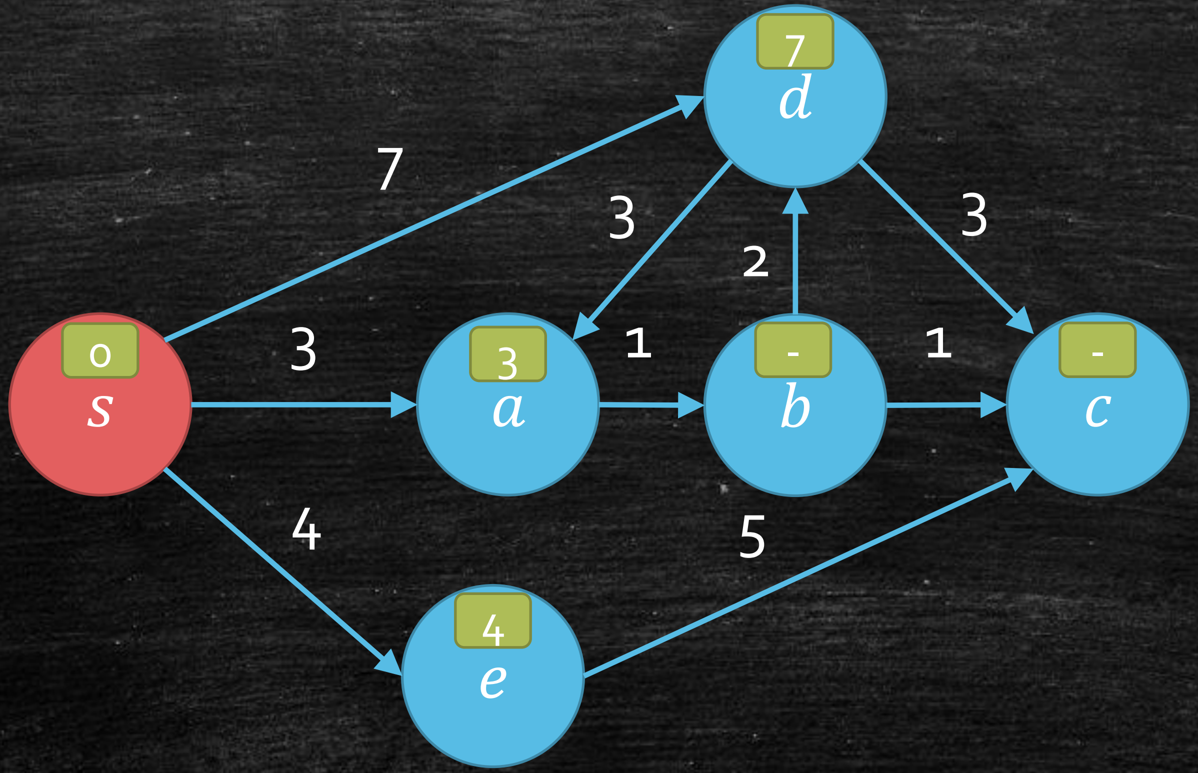
2. Explore

- Find $v \notin T$ with smallest $tdist[v]$.
- $T \leftarrow T + \{v\}$

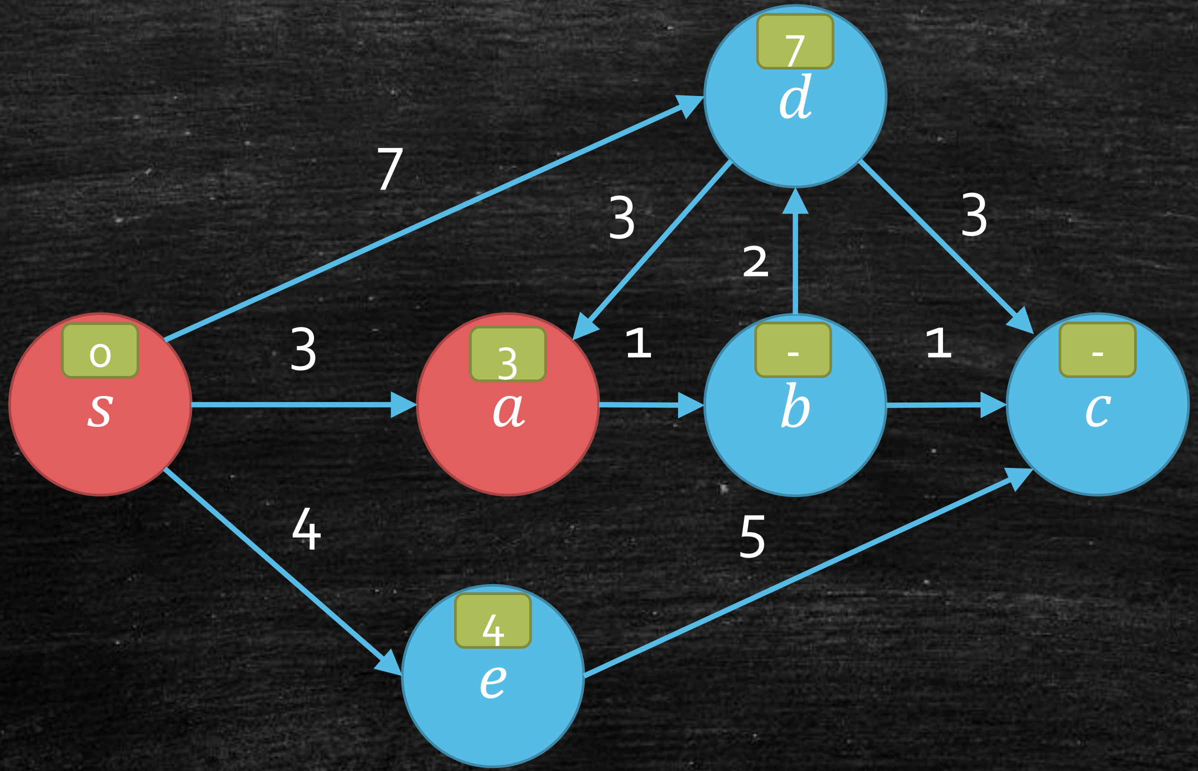
3. Update $tdist[u]$

- $tdist[u] = \min\{tdist[u], tdist[v] + w(v, u)\}$ for all $(v, u) \in E$

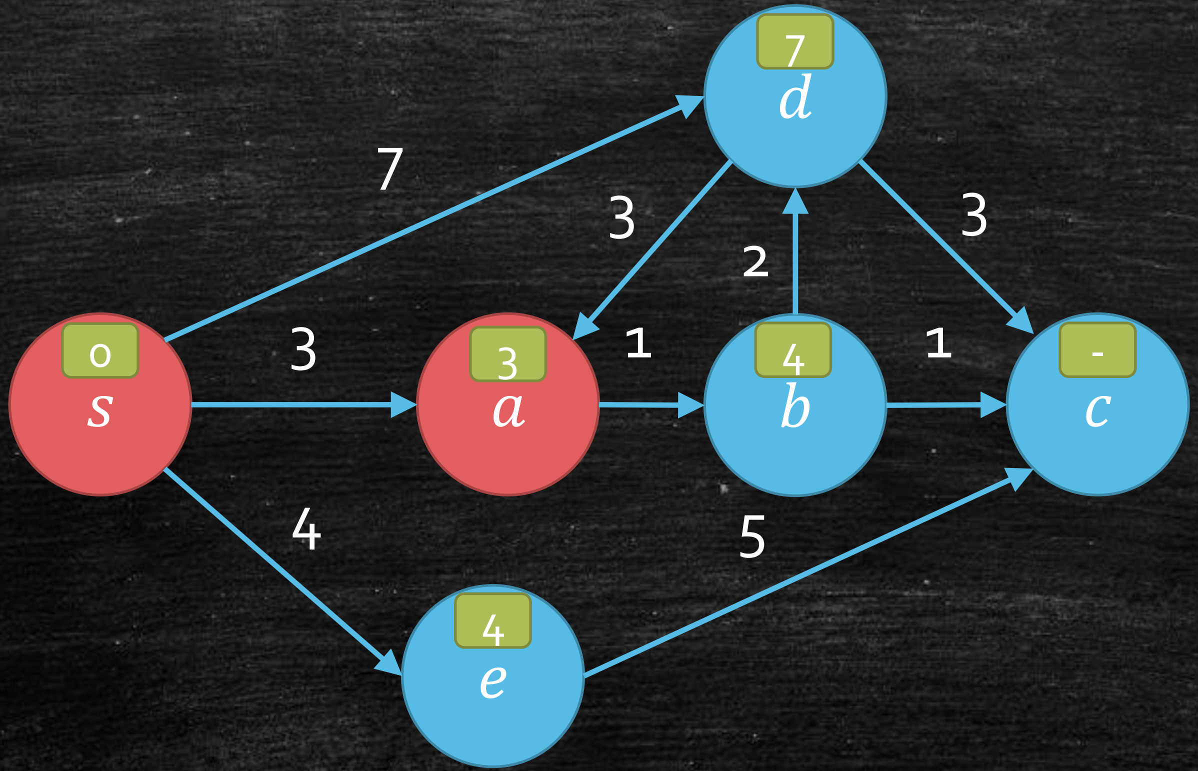
Sample Run



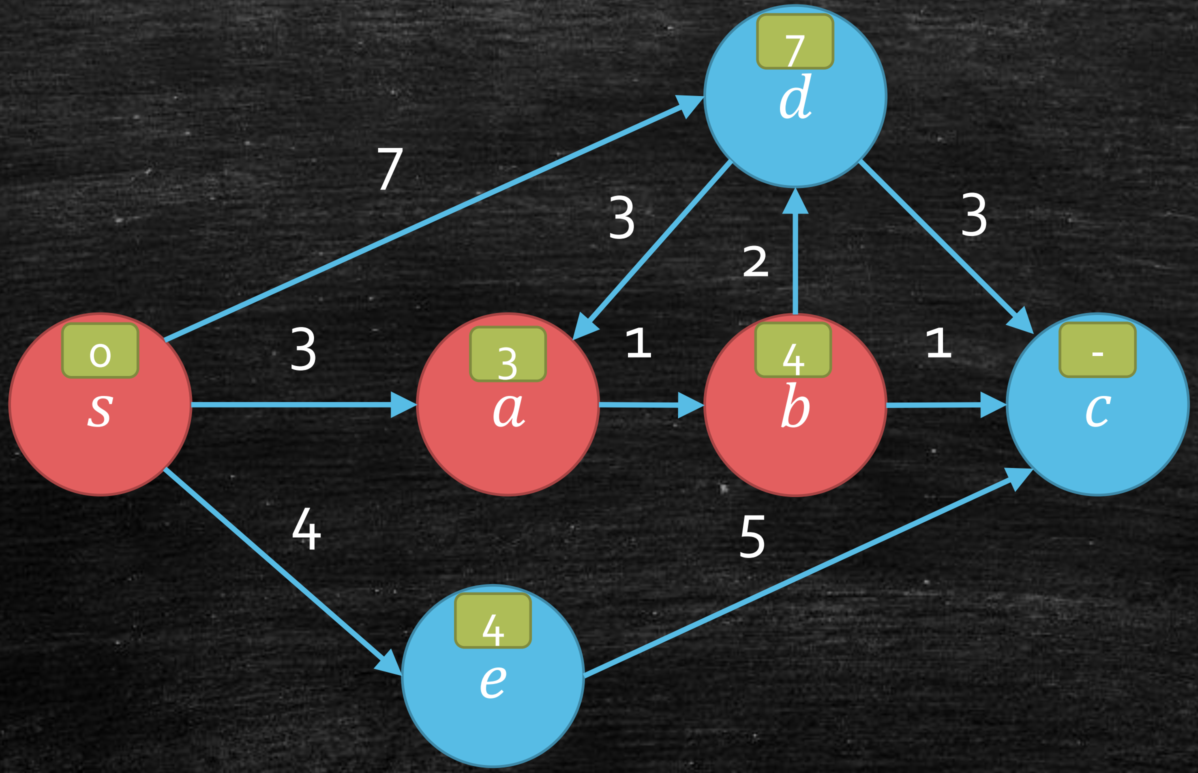
Sample Run



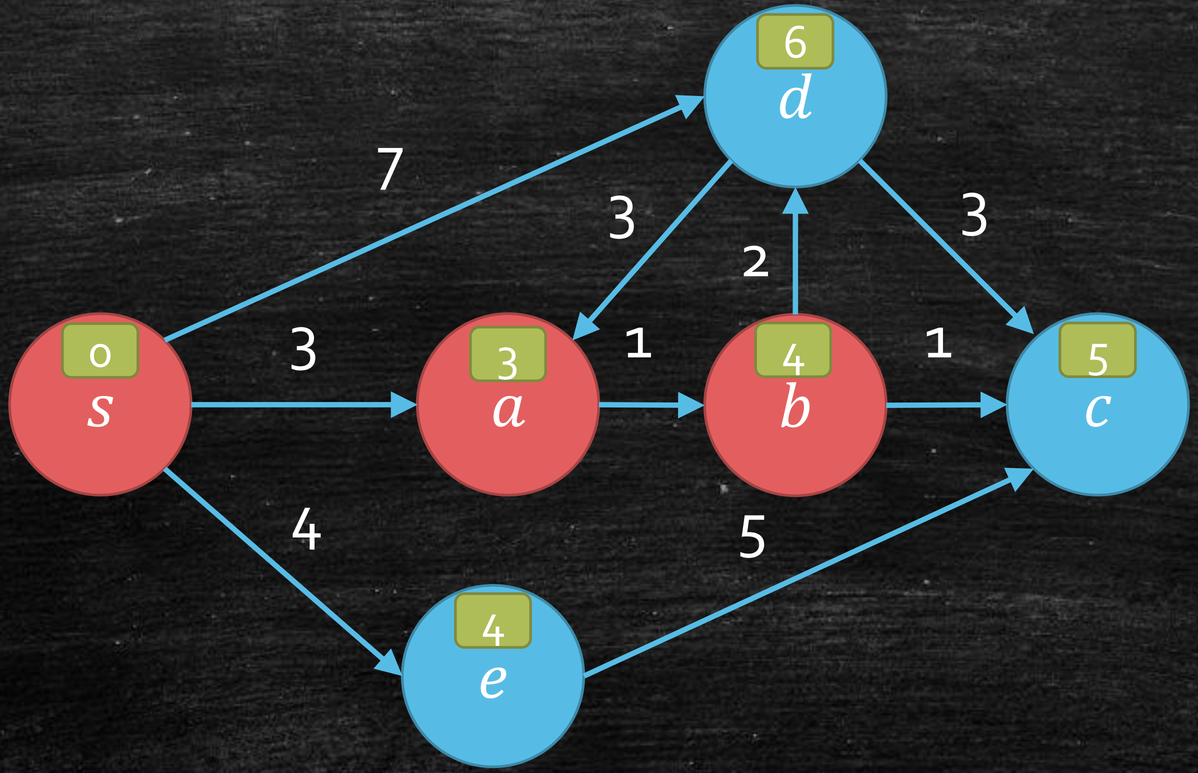
Sample Run



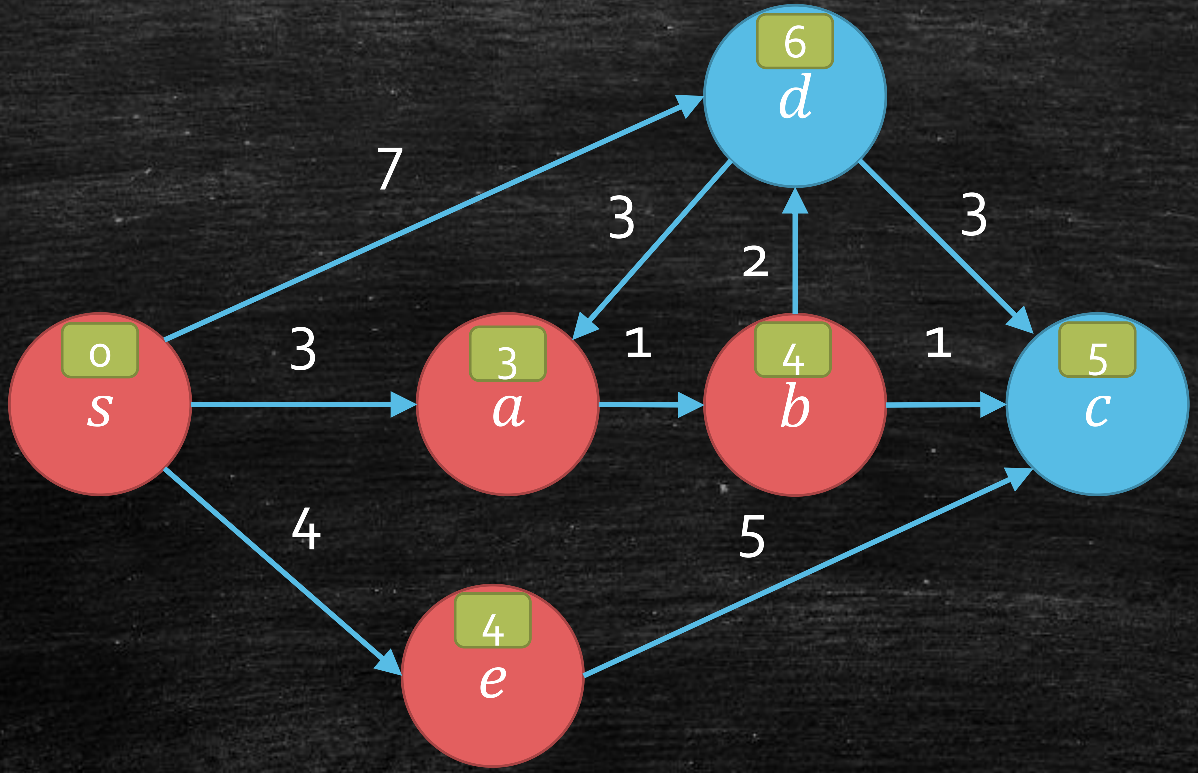
Sample Run



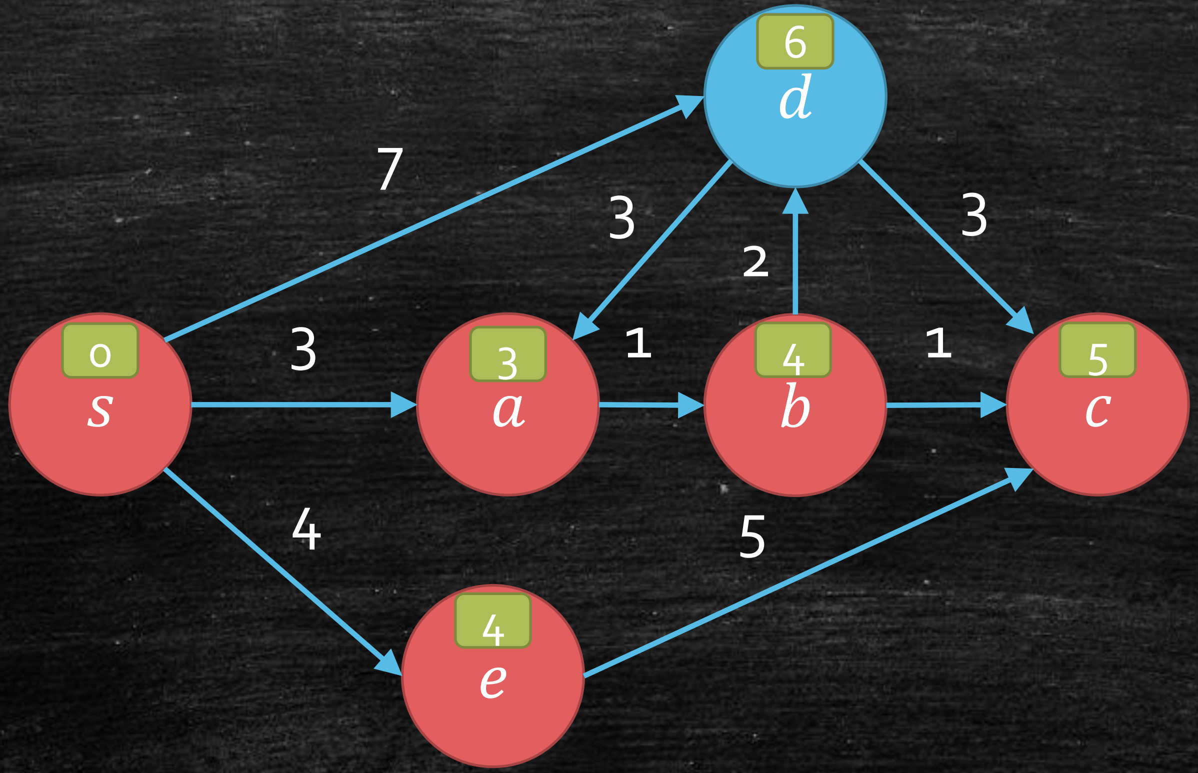
Sample Run



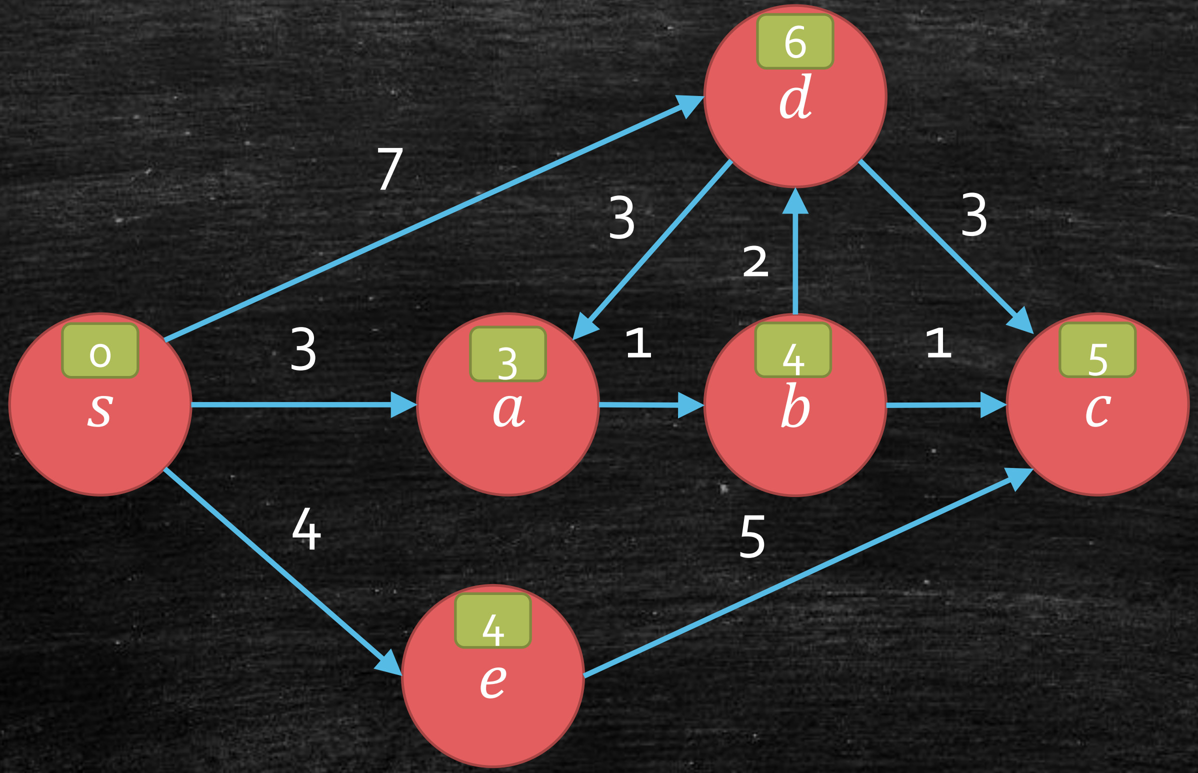
Sample Run



Sample Run



Sample Run



Output a path?

Dijkstra$(G = (V, E), s)$

1. **Initialize**
   - $T \leftarrow \{s\}$
   - $tdist[v] \leftarrow w(s, v)$, $pre[v] \leftarrow s$ for all $(s, v) \in E$.
2. **Explore**
   - Find $v \notin T$ with smallest $tdist[v]$.
   - $T \leftarrow T + \{v\}$
3. **Update $tdist[u]$**
   - $tdist[u] = \min\{tdist[u], tdist[v] + w(v, u)\}$ for all $(v, u) \in E$.
   - If $tdist[u]$ is updated, then $pre[u] \leftarrow v$.

Time Complexity

Dijkstra$(G = (V, E), s)$

1. **Initialize**
   - $T \leftarrow \{s\}$
   - $tdist[v] \leftarrow w(s,v),\ pre[v] \leftarrow s$ for all $(s,v) \in E$.
2. **Explore**
   - Find $v \notin T$ with smallest $tdist[v]$.
   - $T \leftarrow T + \{v\}$
3. **Update $tdist[u]$**
   - $tdist[u] = \min\{tdist[u], tdist[v] + w(v,u)\}$ for all $(v,u) \in E$.
   - If $tdist[u]$ is updated, then $pre[u] \leftarrow v$.

$|V|$ rounds

$|E|$ rounds

$|E|$ rounds

Time Complexity: Conclusion

- Find Min
  - $|V|$ rounds
- Update
  - |E| rounds
- If we use simple array, then
  - First round find min: $|V| - 1$
  - Second round find min: $|V| - 2$
  - …
  - Find min totally: $O(|V|^2)$
  - Each update: $O(1)$
  - Update totally: $O(|E|)$
  - Algorithm totally: $\boldsymbol{O(|V|^2 + |E|)}$

Improve Dijkstra by Heap!

- Find Min
  - $|V|$ rounds
- Update
  - |E| rounds
- What about heap?

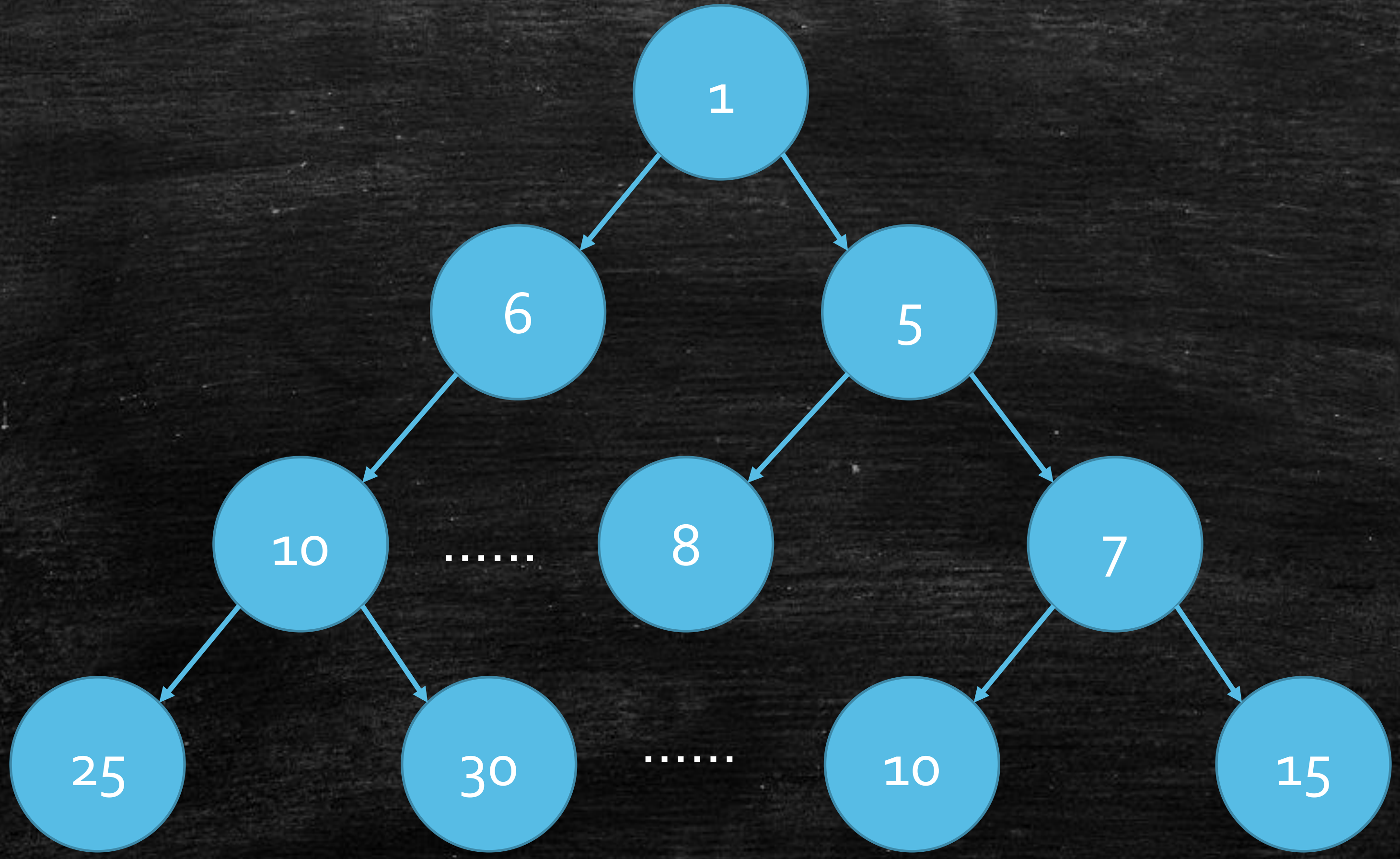
	Pop Max	Insert	Update Key	Merge
Binary Heap	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(n)$
d-nary Heap	$O(d\log_d n)$	$O(\log_d n)$	$O(\log_d n)$	$O(n)$
Binomial Heap	$O(\log n)$	$O(1)$	$O(\log n)$	$O(\log n)$
Fibonacci	$O(\log n)$	$O(1)$	$O(1)$	$O(1)$

Improve Dijkstra by Heap!

- Binary Heap
  - Find Min: $O(|V|\log|V|)$
  - Update: $O(|E|\log|E|)$
  - Totally: $\boldsymbol{O((|V|+|E|)\log|V|)}$
- d-nary Heap
  - Find Min: $O(|V|d\log_d|V|)$
  - Update: $O(|E|\log_d|E|)$
  - Set $d=|E|/|V|$
  - Totally: $\boldsymbol{O(|E|\log_{|E|/|V|}|V|)}$
- Fibonacci Heap
  - Find Min: $O(|V|\log|V|)$
  - Update: $O(|E|)$
  - Totally: $\boldsymbol{O(|E|+|V|\log|V|)}$
  - Better than $\boldsymbol{O(|V|^2+|E|)}$

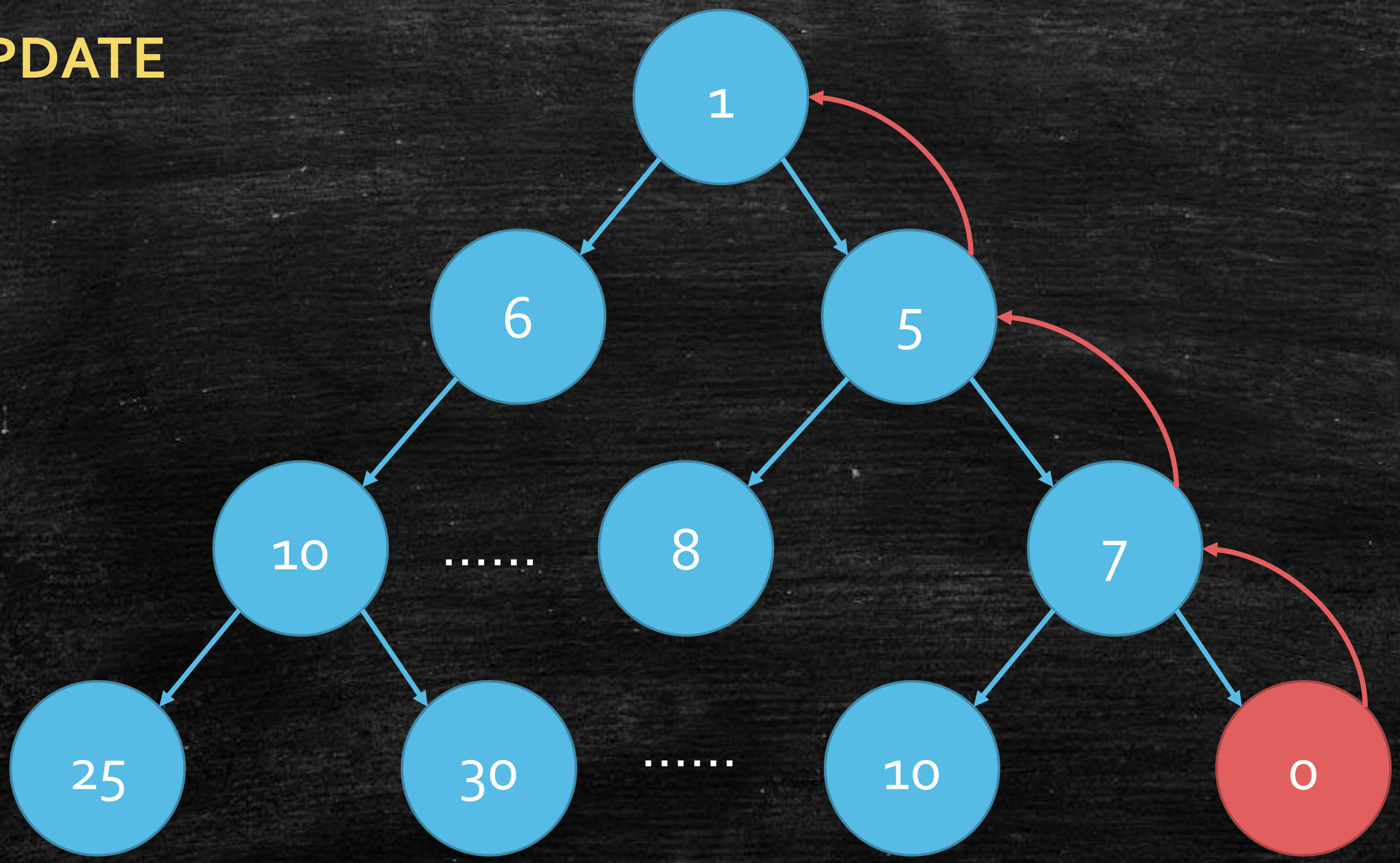
	Pop Min	Insert	Update Key	Merge
Binary Heap	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(n)$
d-nary Heap	$O(d\log_d n)$	$O(\log_d n)$	$O(\log_d n)$	$O(n)$
Binomial Heap	$O(\log n)$	$O(1)$	$O(\log n)$	$O(\log n)$
Fibonacci	$O(\log n)$	$O(1)$	$O(1)$	$O(1)$

Quick Review (or Preview?): Binary Heap



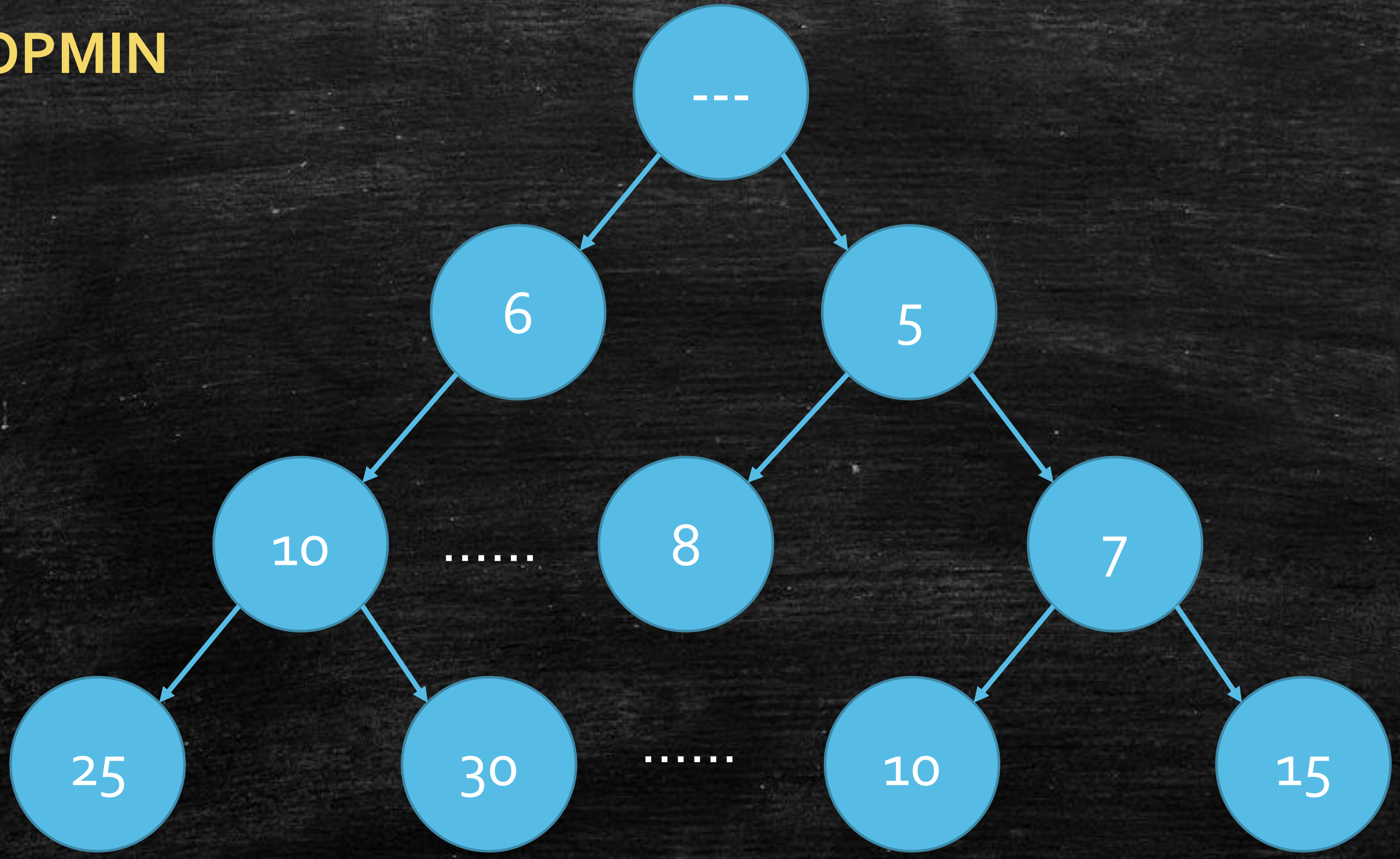
Quick Review (or Preview?): Binary Heap

UPDATE



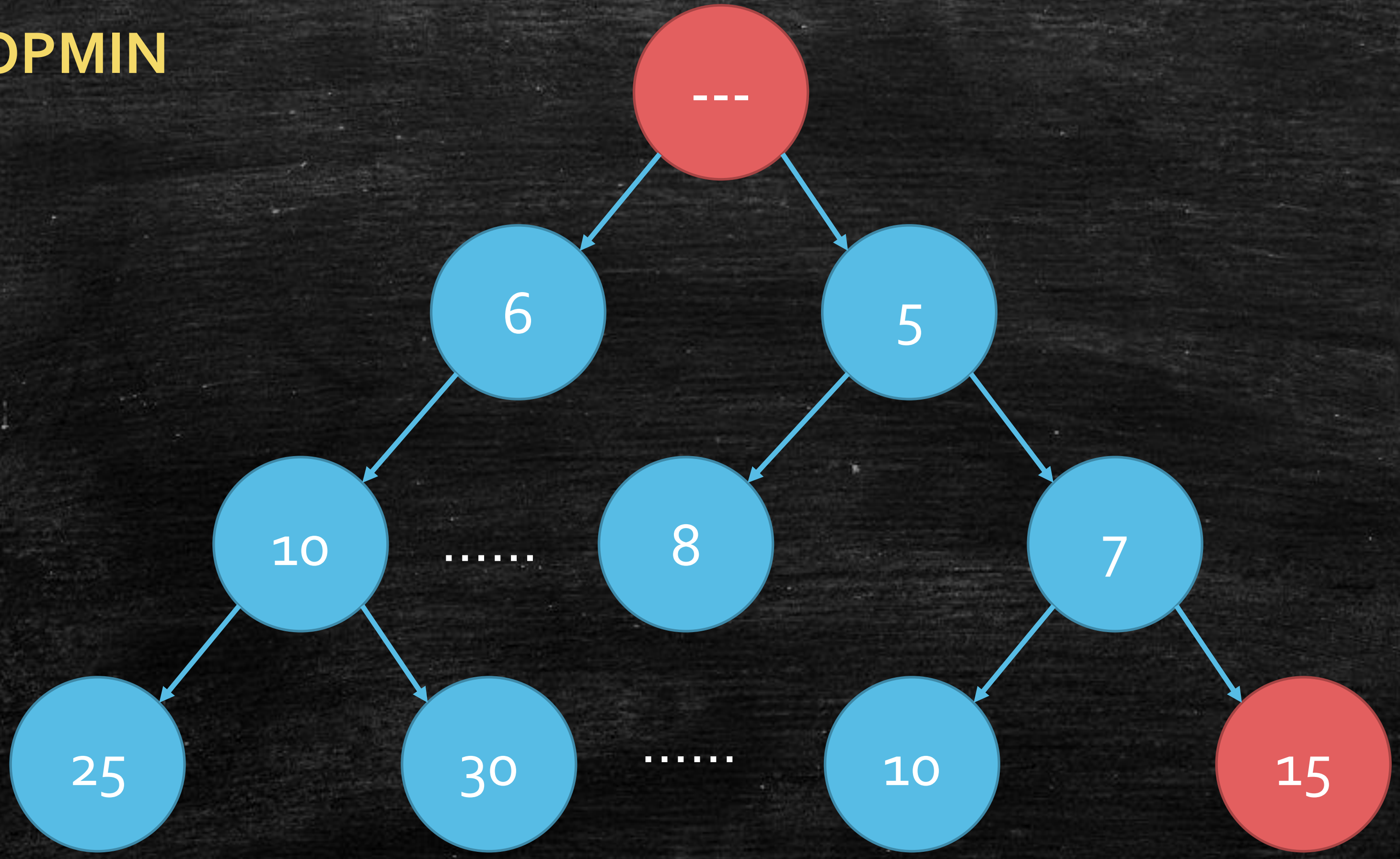
Quick Review (or Preview?): Binary Heap

POPMIN



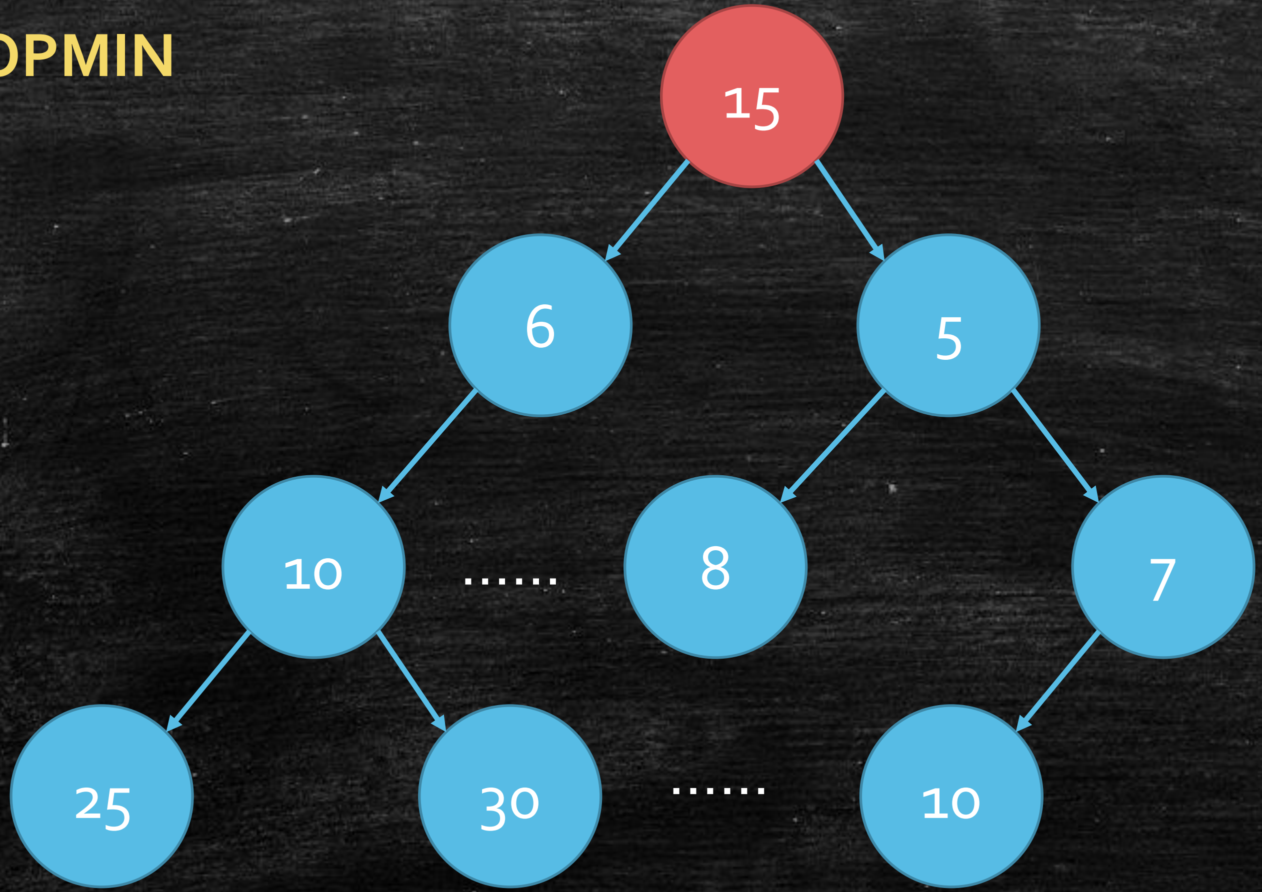
Quick Review (or Preview?): Binary Heap

POPMIN



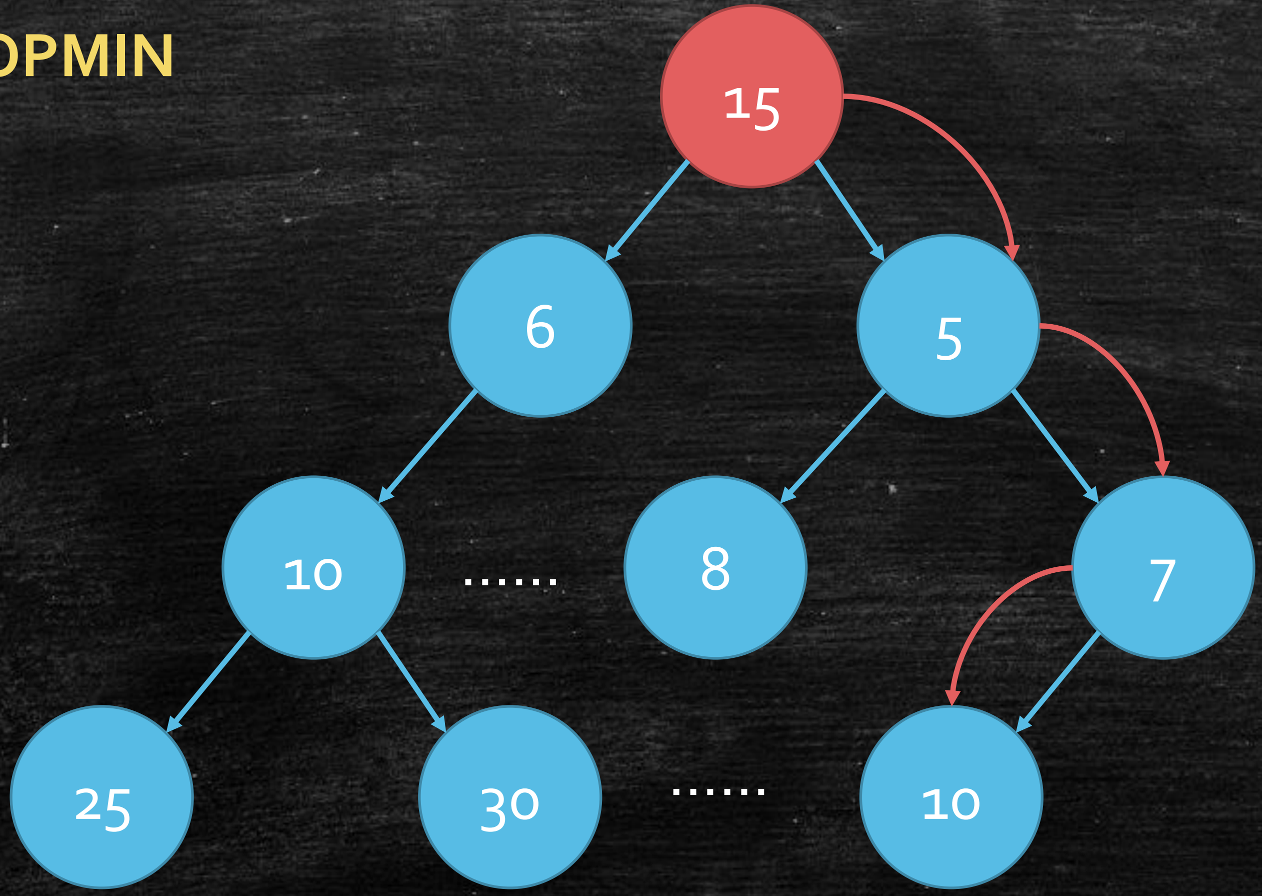
Quick Review (or Preview?): Binary Heap

POPMIN



Quick Review (or Preview?): Binary Heap

POPMIN

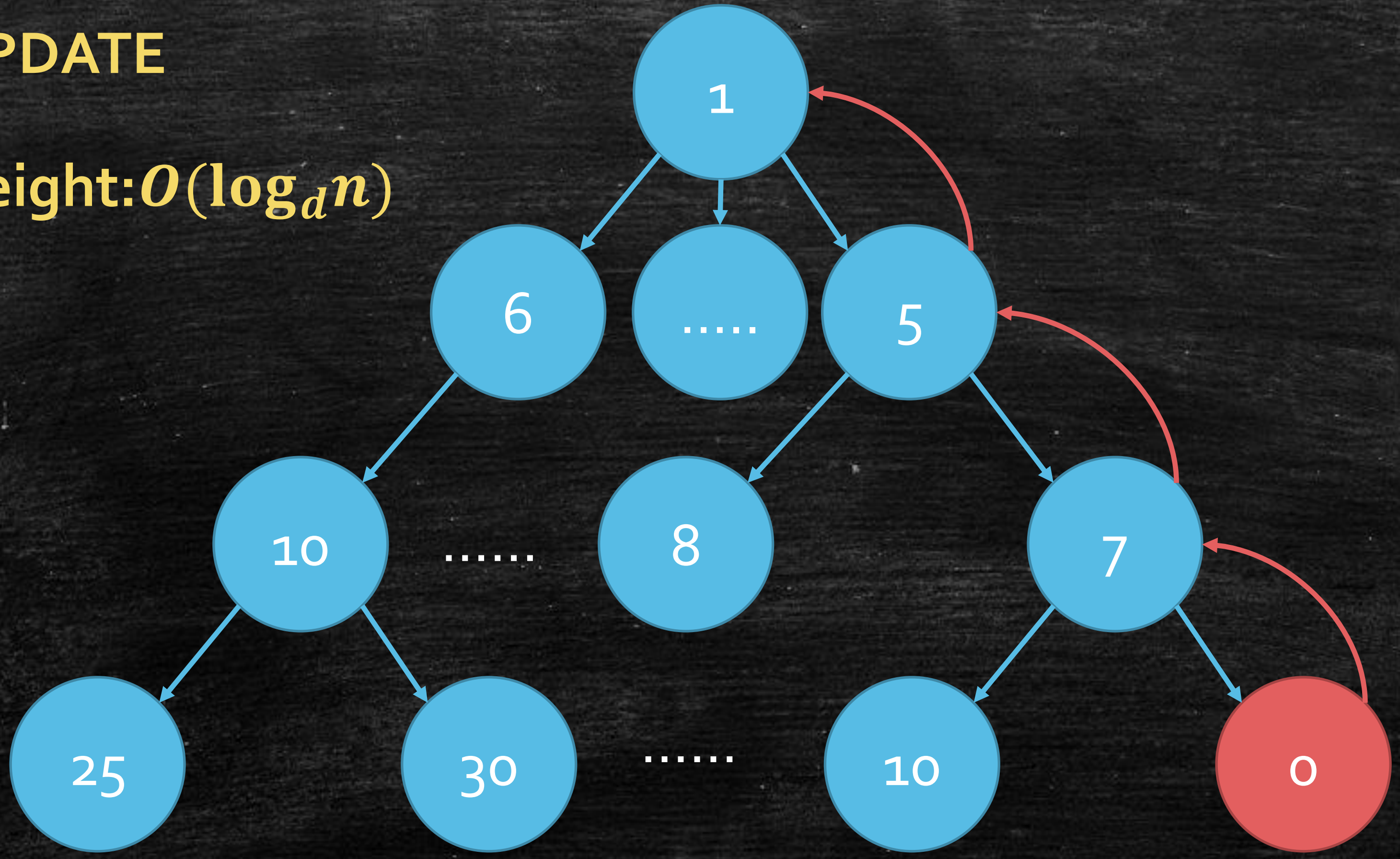


It is still balance!

Quick Review (or Preview?): d-nary Heap

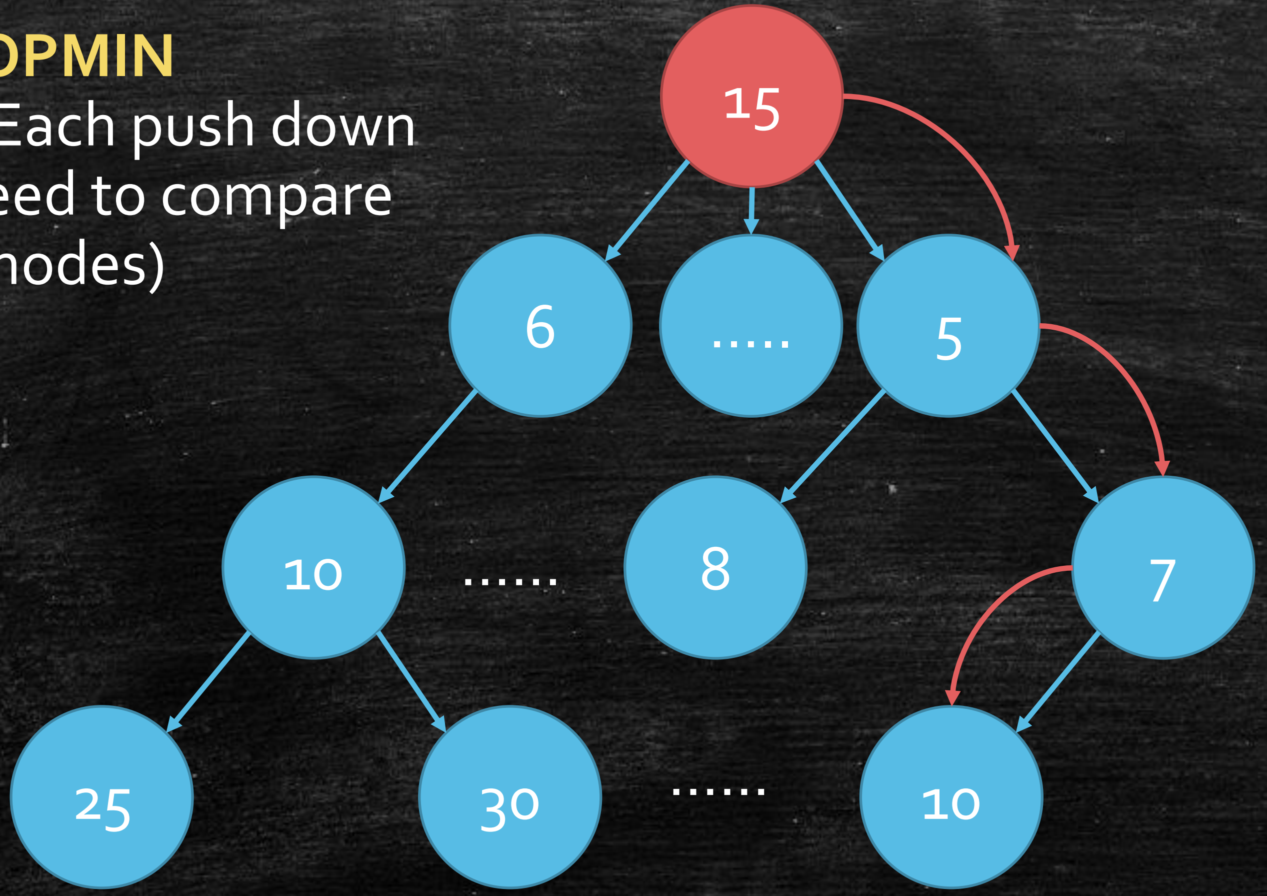
UPDATE

Height: $O(\log_d n)$



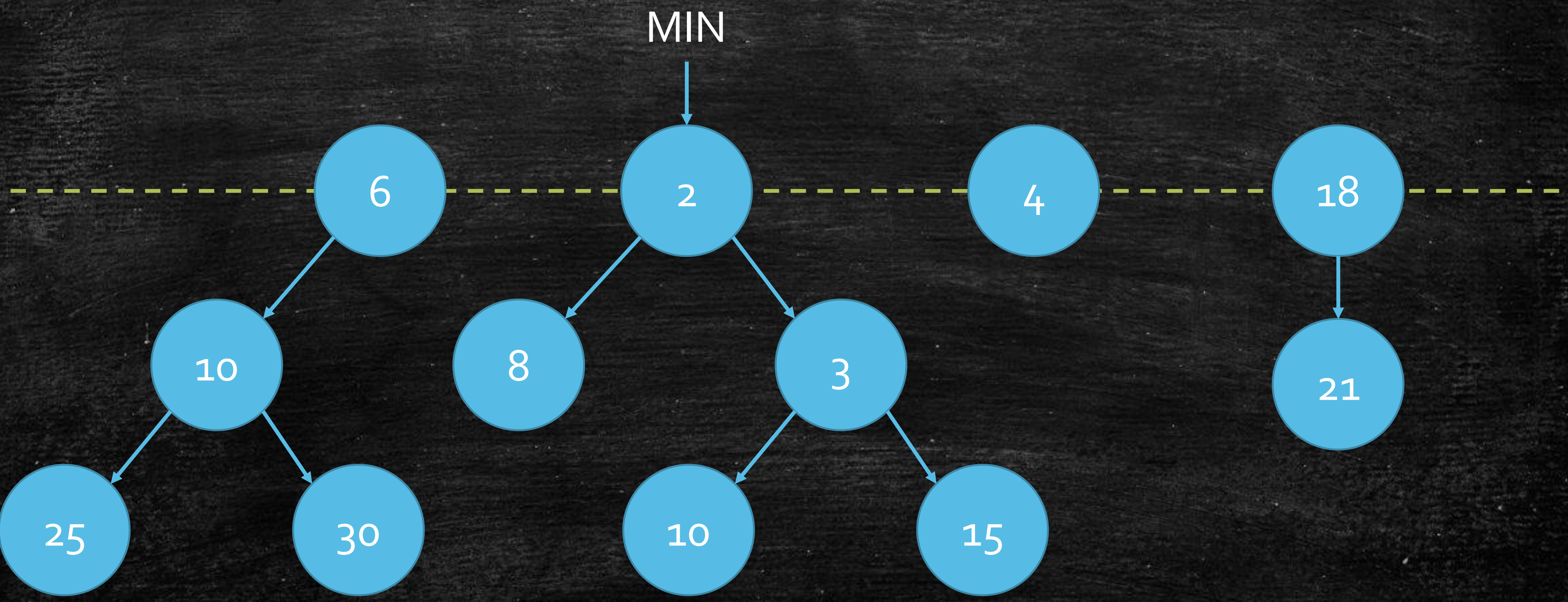
Quick Review (or Preview?): d-nary Heap

POPMIN
(*Each push down
Need to compare
d nodes)

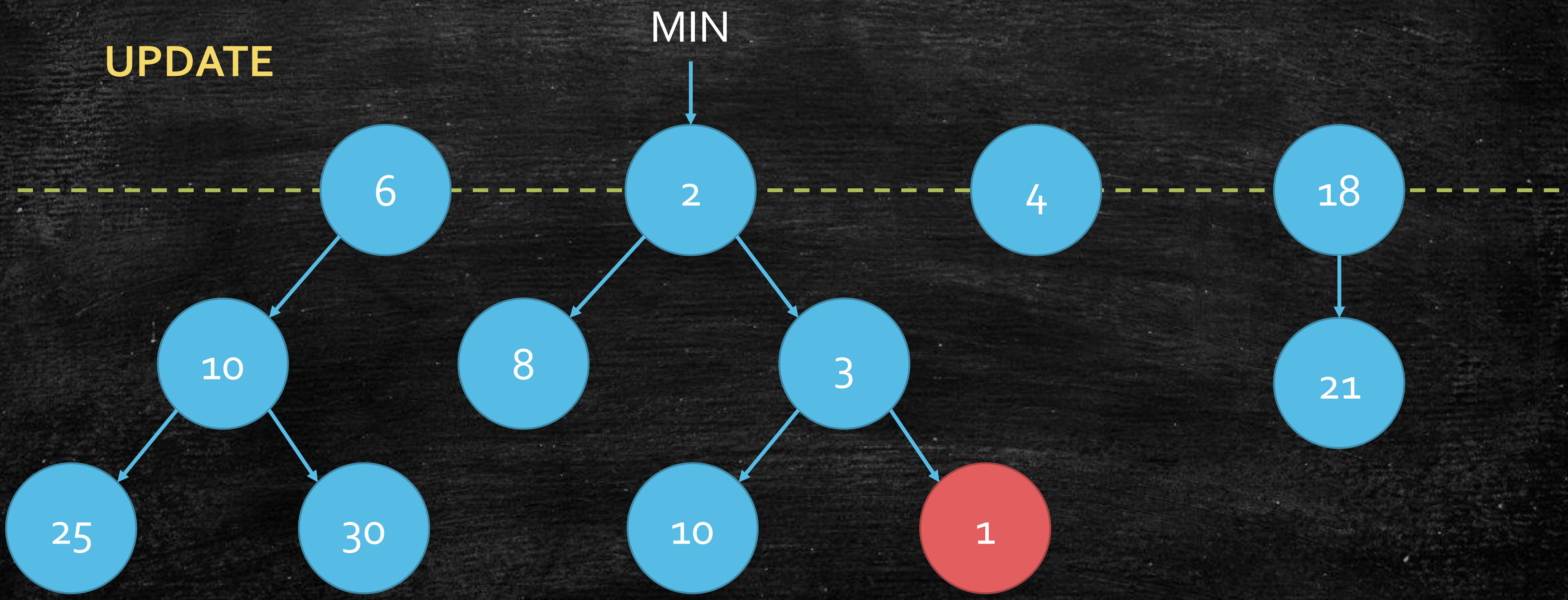


Quick Review (or Preview?): Fibonacci Heap

Quick Review (or Preview?): Fibonacci Heap

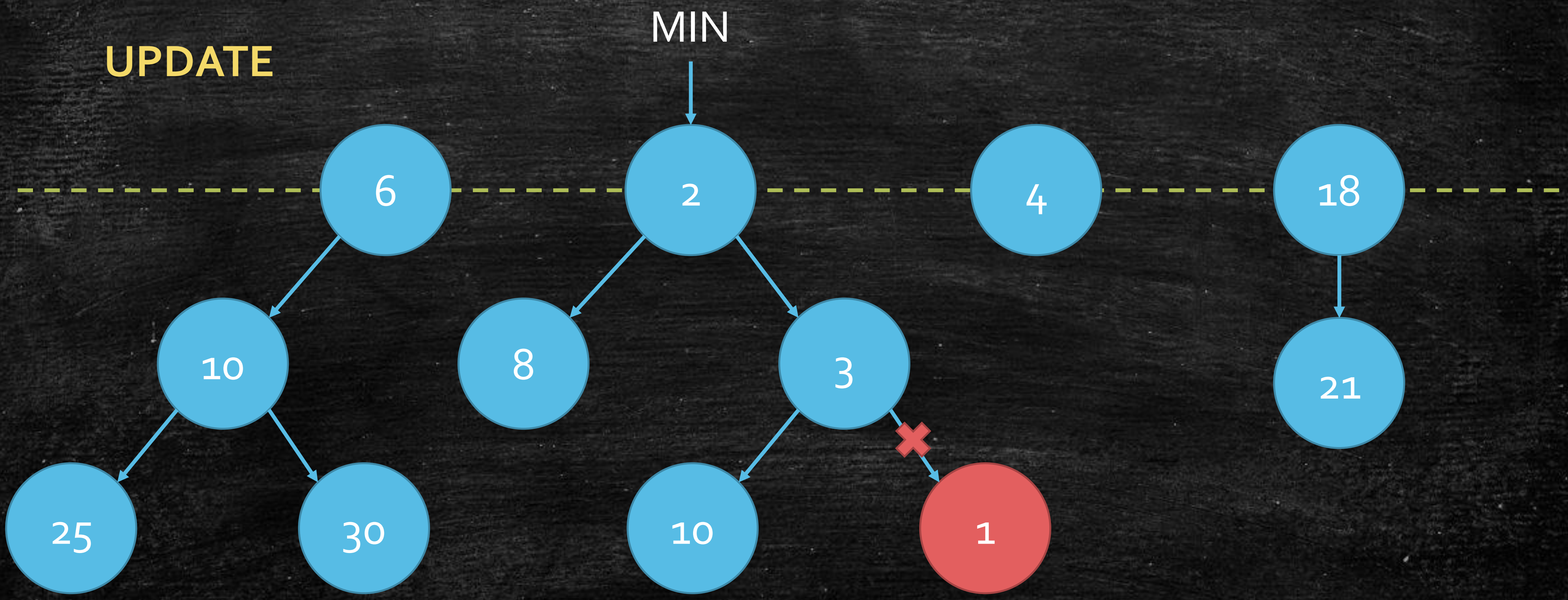


Quick Review (or Preview?): Fibonacci Heap

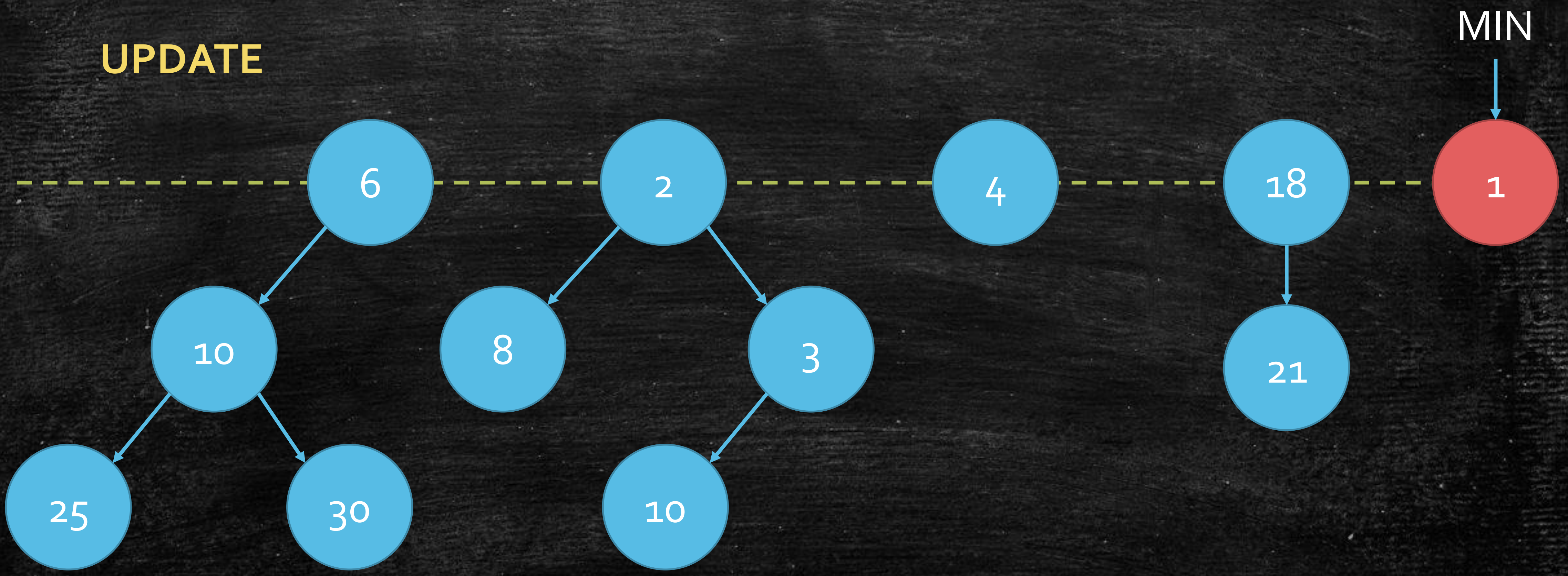


Quick Review (or Preview?): Fibonacci Heap

UPDATE

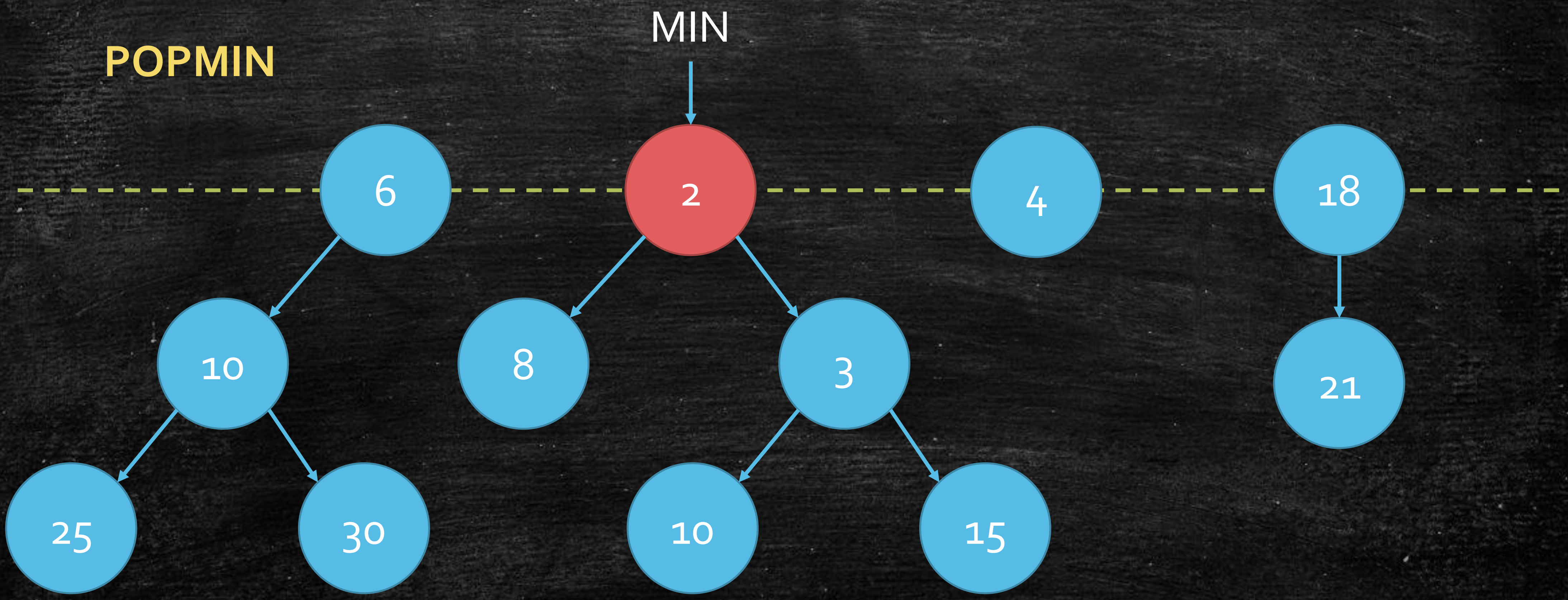


Quick Review (or Preview?): Fibonacci Heap



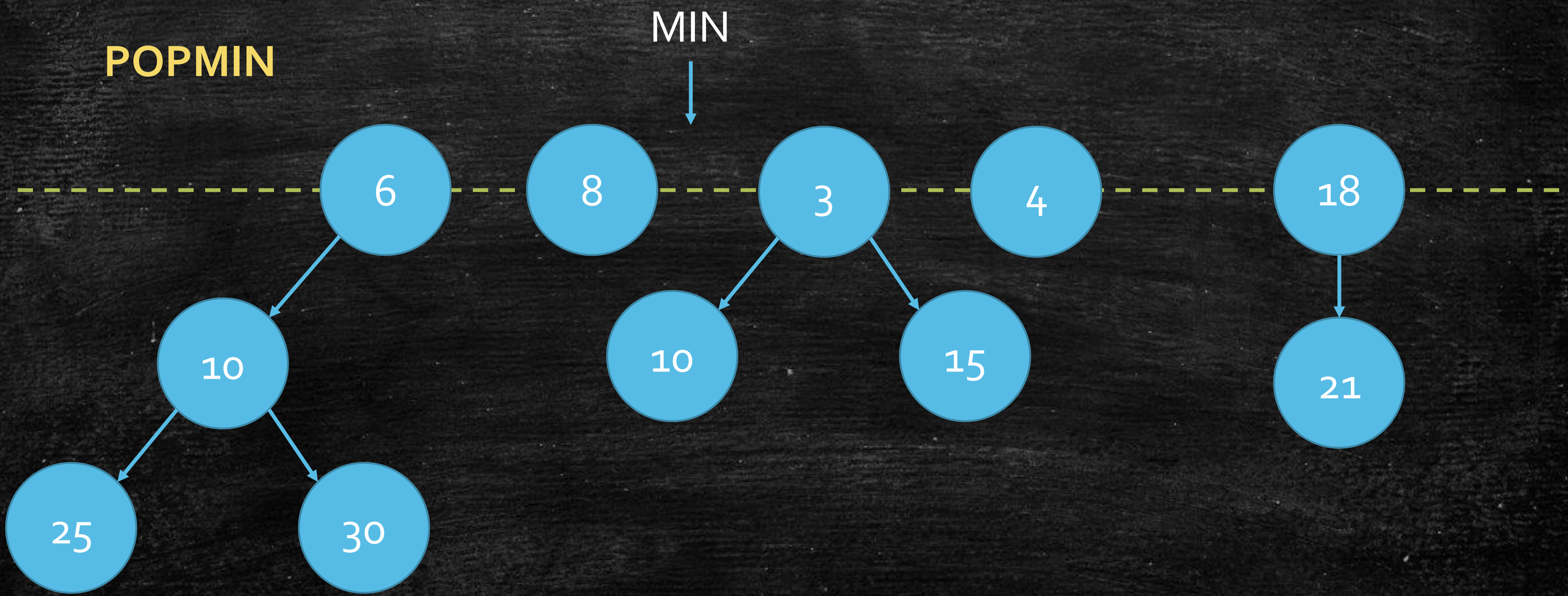
Quick Review (or Preview?): Fibonacci Heap

POPMIN

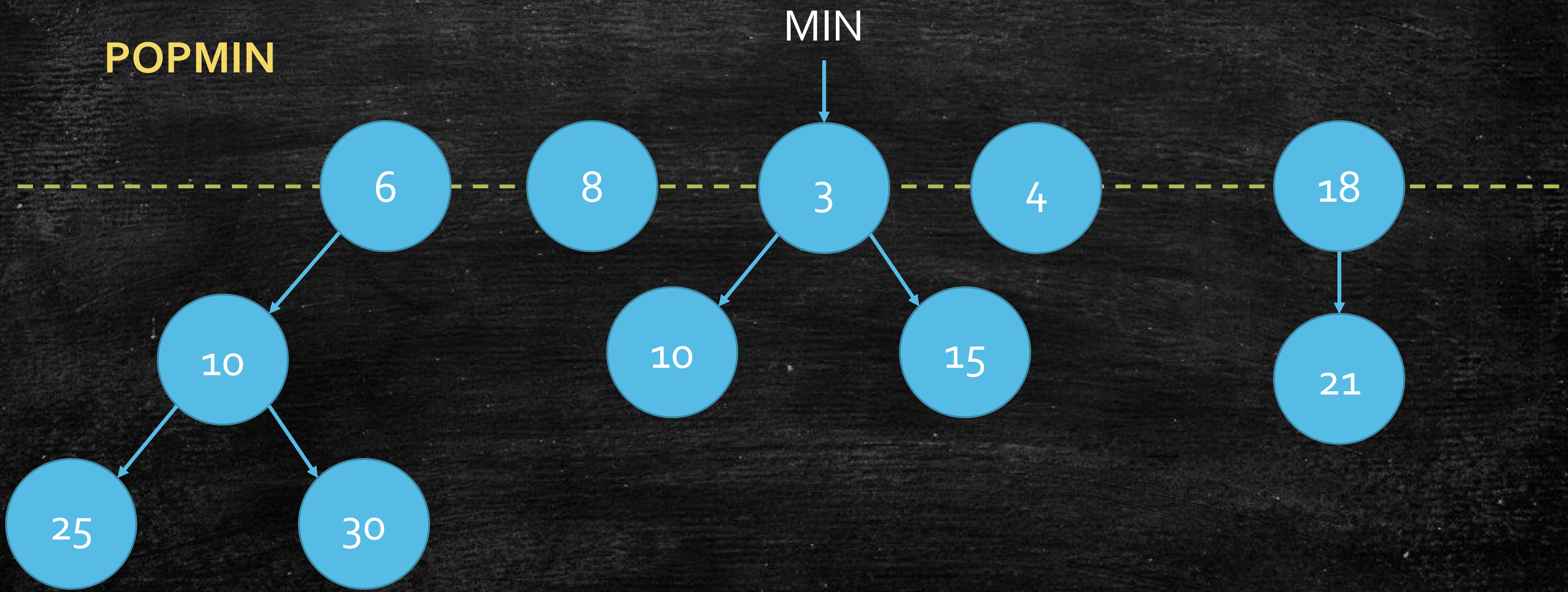


Quick Review (or Preview?): Fibonacci Heap

POPMIN



Quick Review (or Preview?): Fibonacci Heap



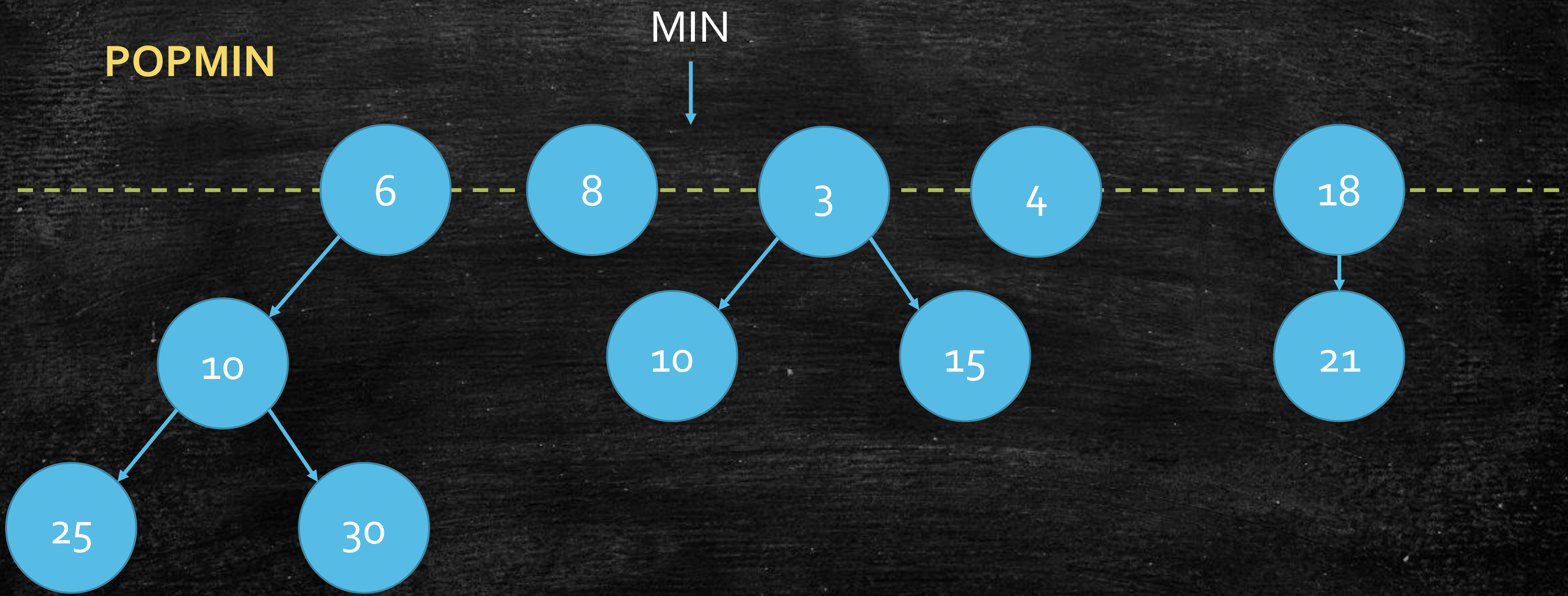
Still many problems!

- Update seems good: $\boldsymbol{O(1)}$
- Pop Min need to compare all the roots?
- It can be very bad: $\boldsymbol{O(n)}$!

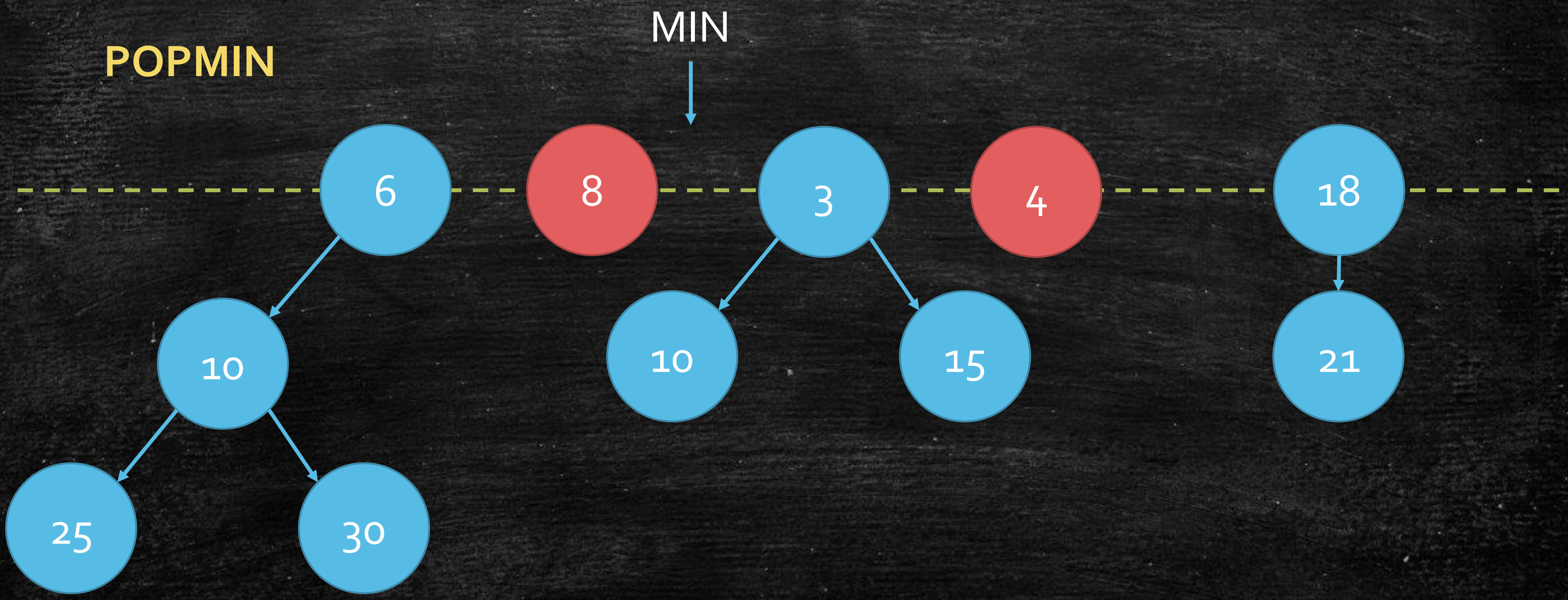
Fix the problem!

- Update seems good: $\boldsymbol{O}(\mathbf{1})$
- Pop Min need to compare all the roots?
- It can be very bad: $\boldsymbol{O}(\boldsymbol{n})$!
- Solution
  - Each degree at most has one root!
    - 1 root with degree 1, 1 root with degree 2……
  - Bound Largest degree → Bound the number of roots!

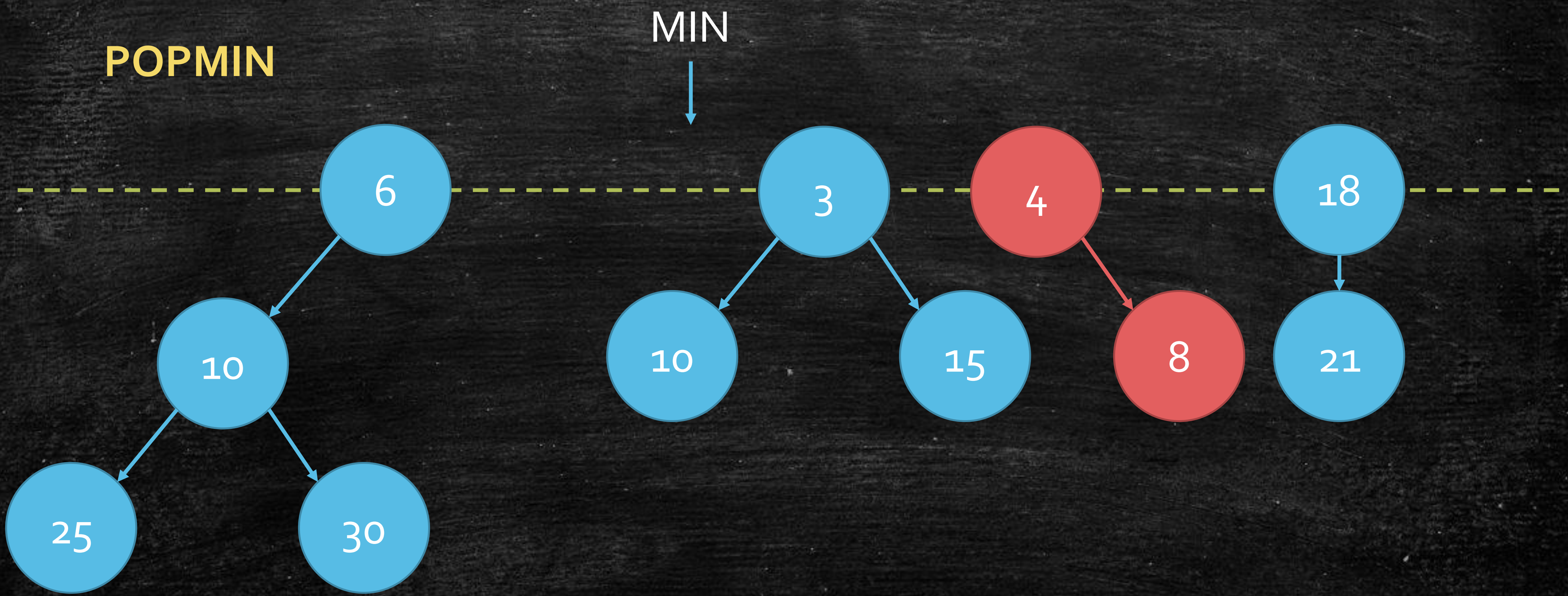
Before move the MIN pointer: Merge



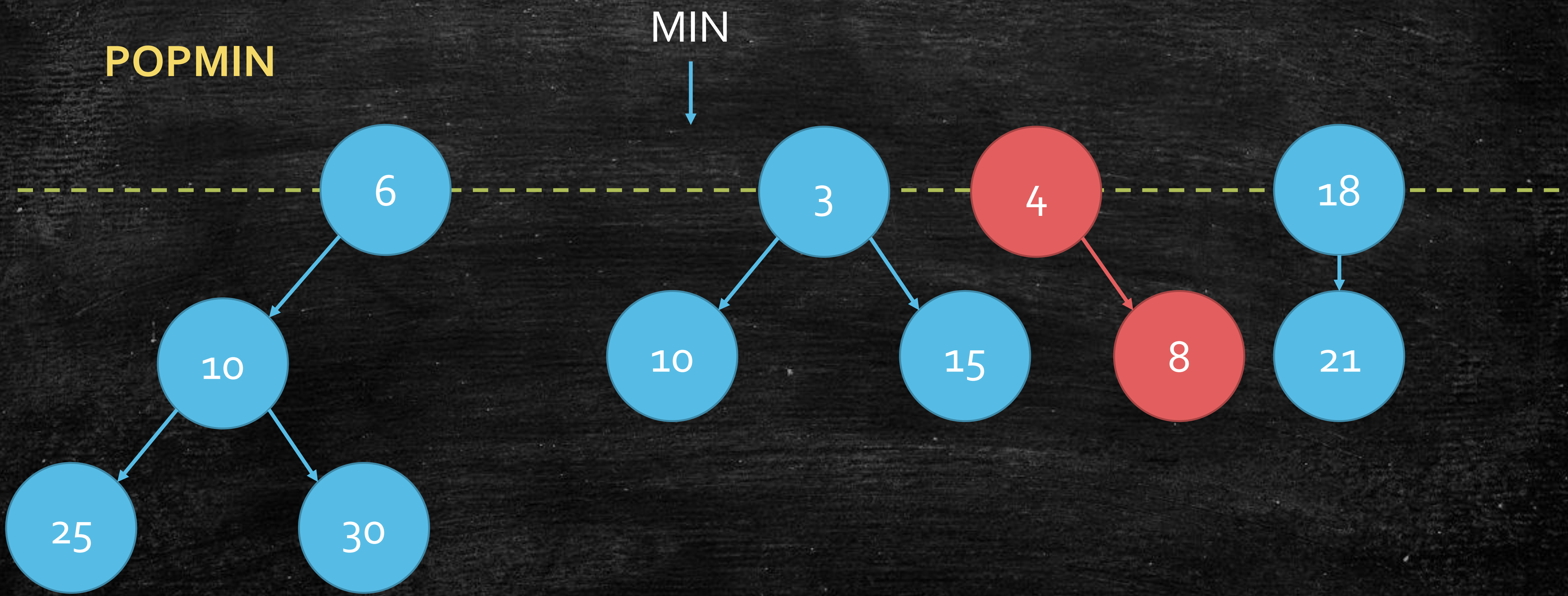
Before move the MIN pointer: Merge



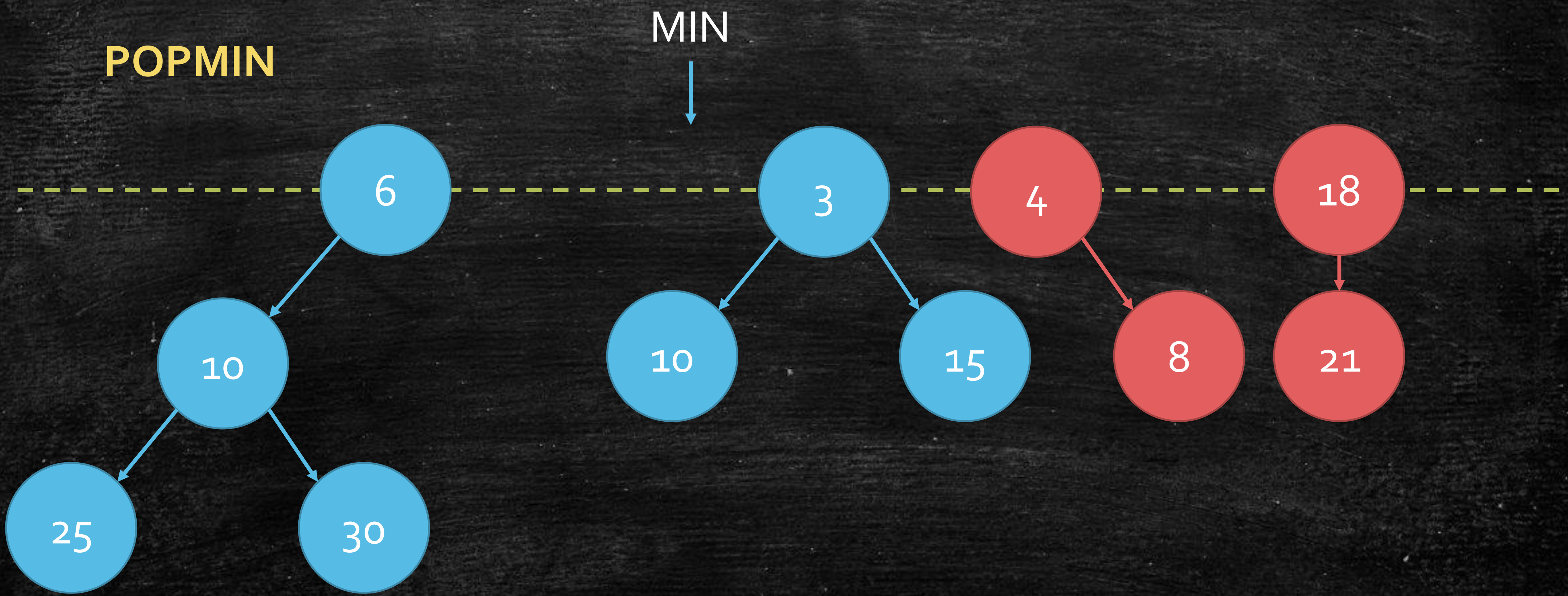
Before move the MIN pointer: Merge



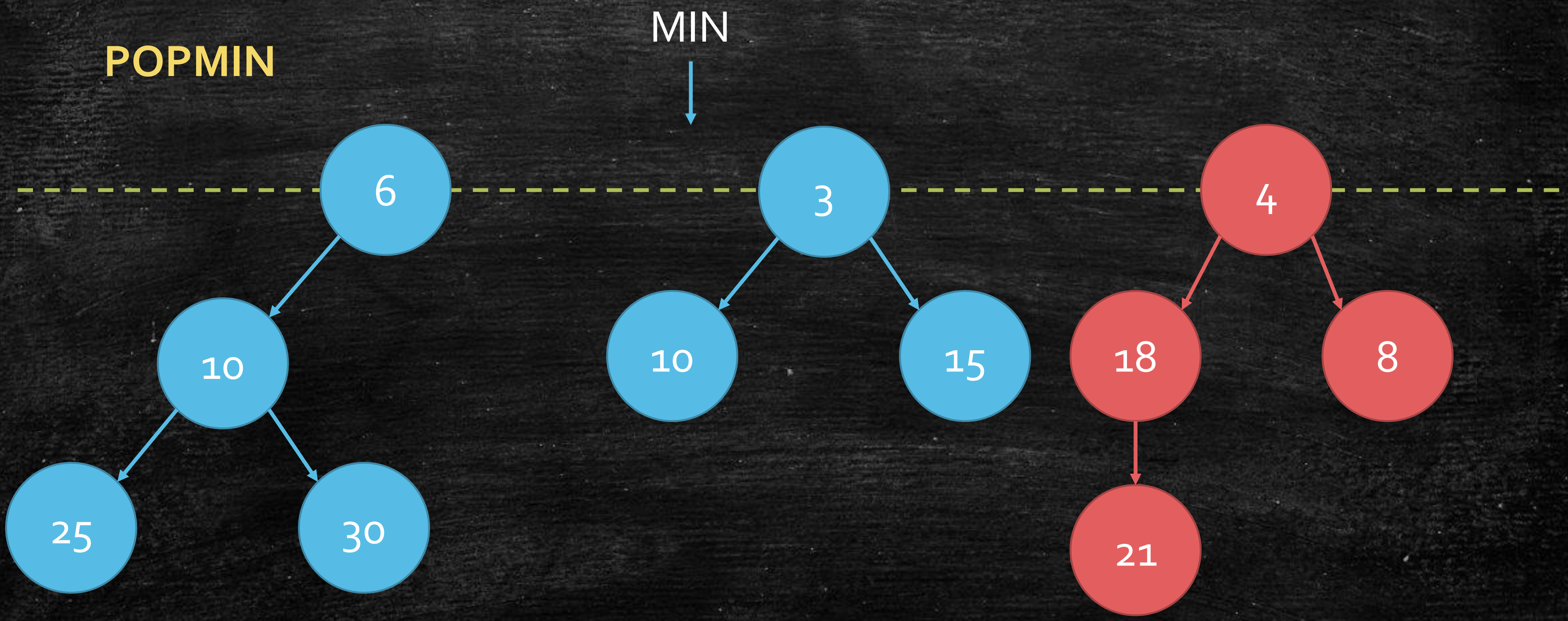
Before move the MIN pointer: Merge



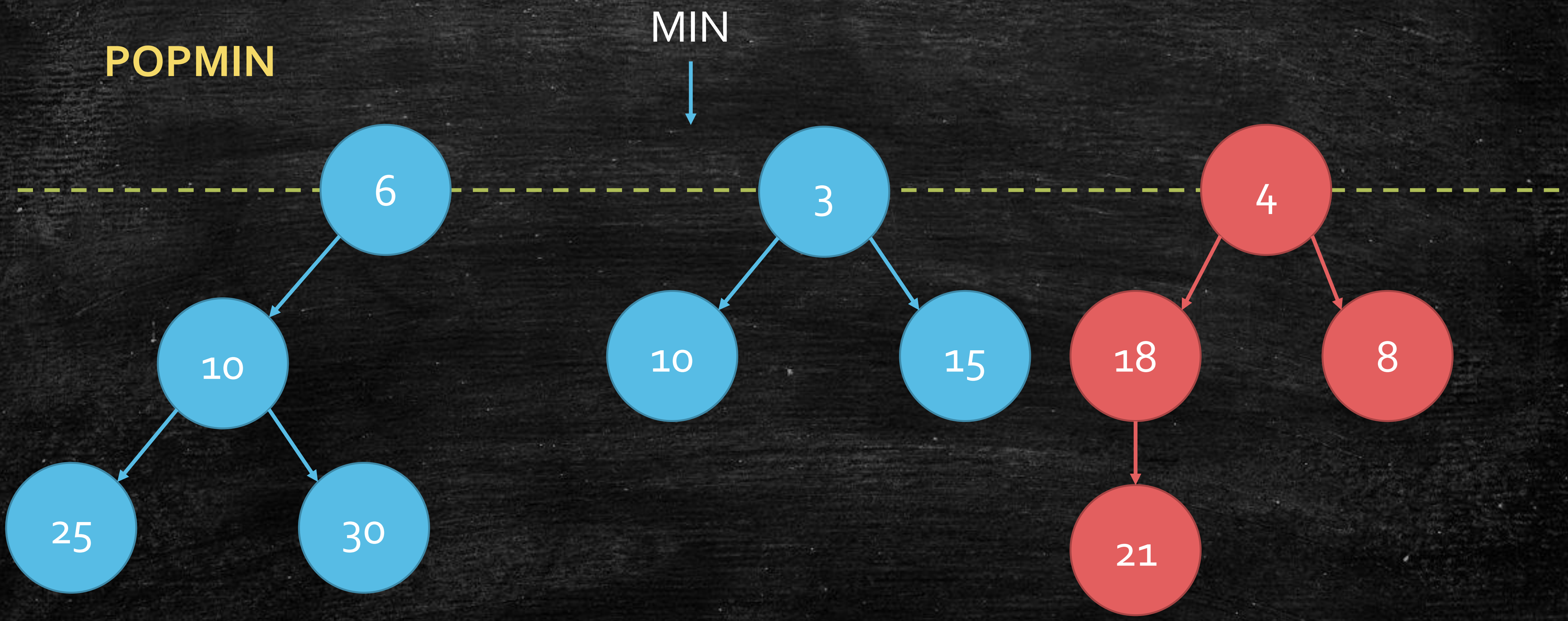
Before move the MIN pointer: Merge



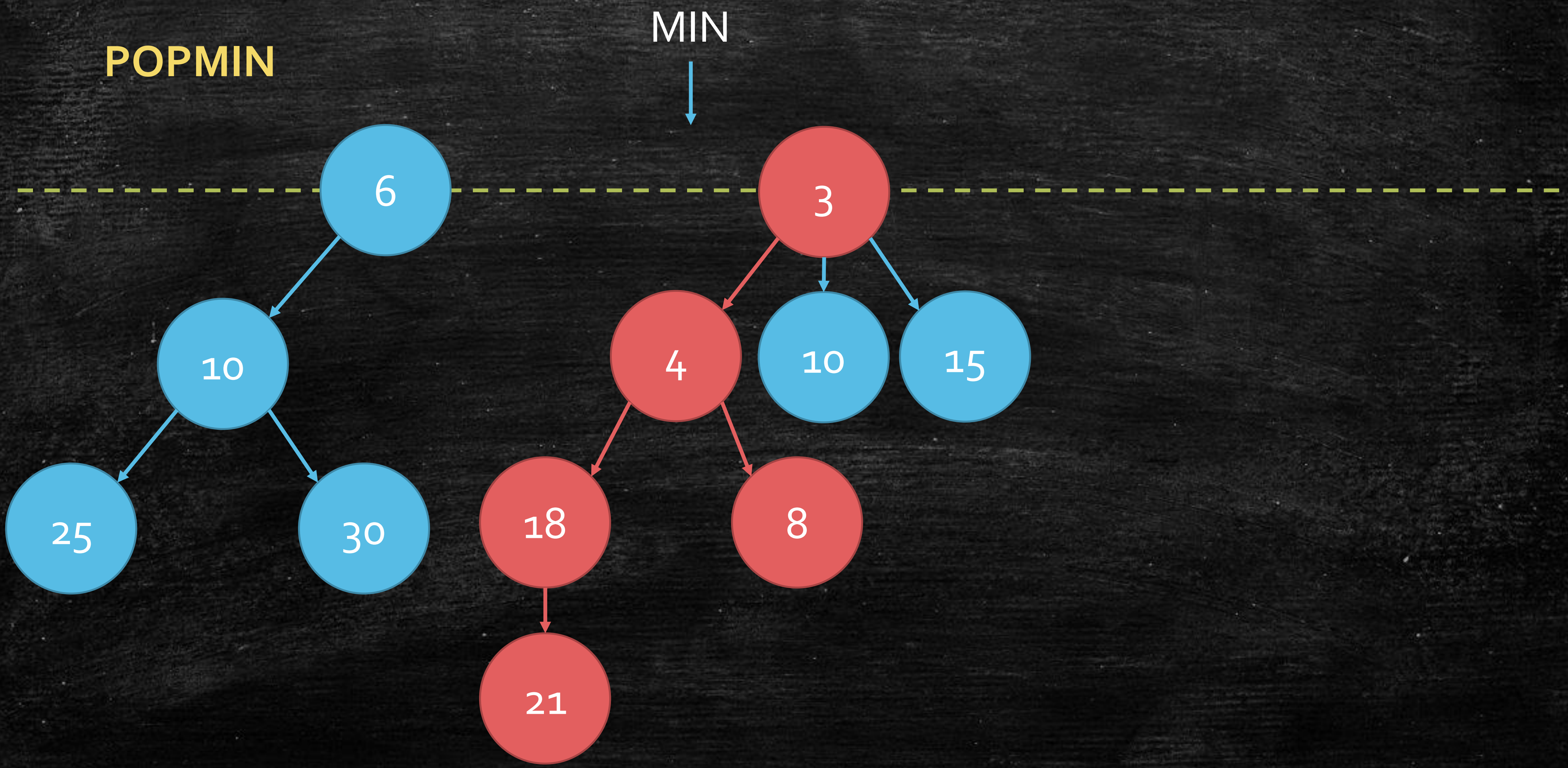
Before move the MIN pointer: Merge



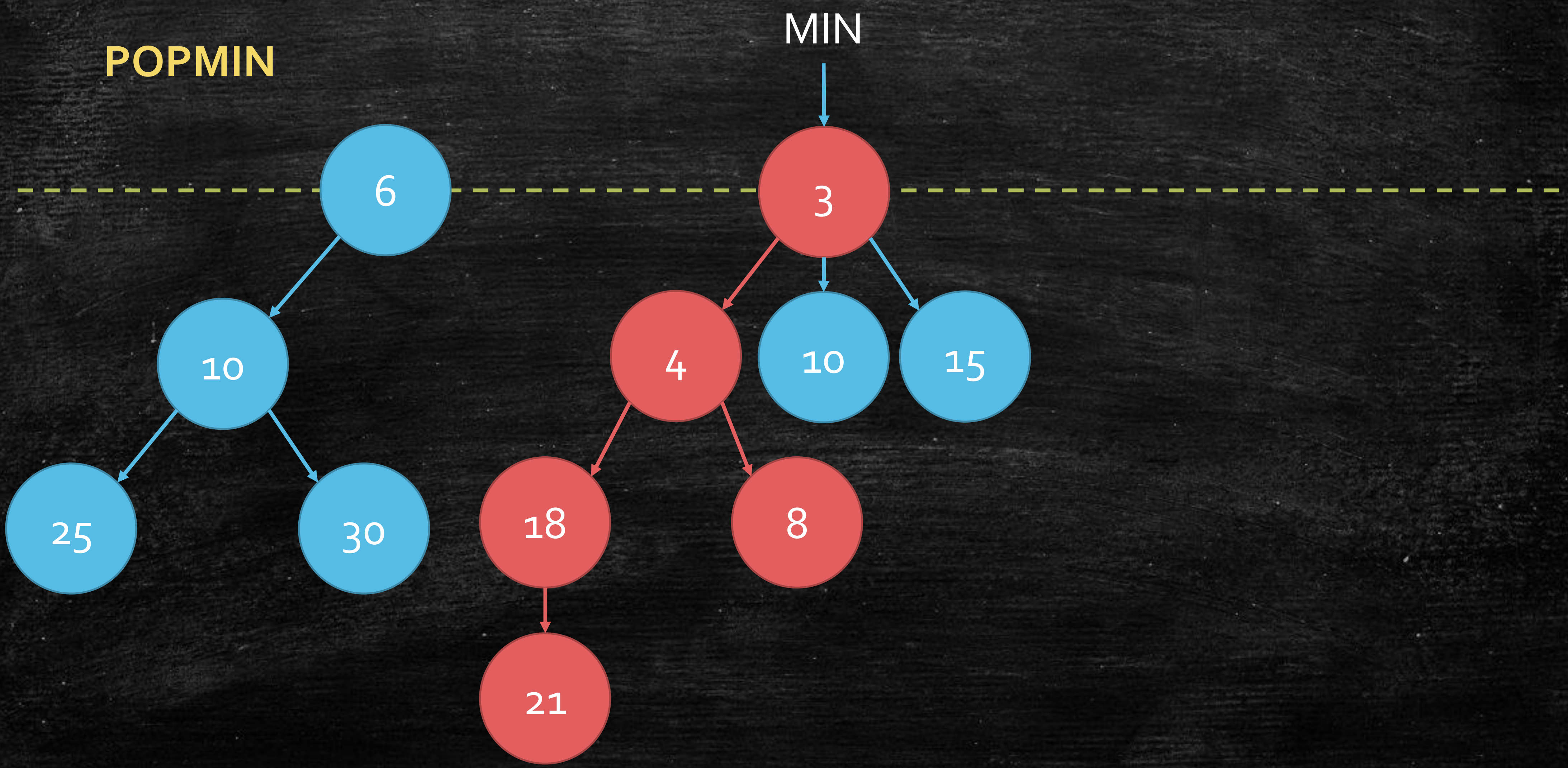
Before move the MIN pointer: Merge



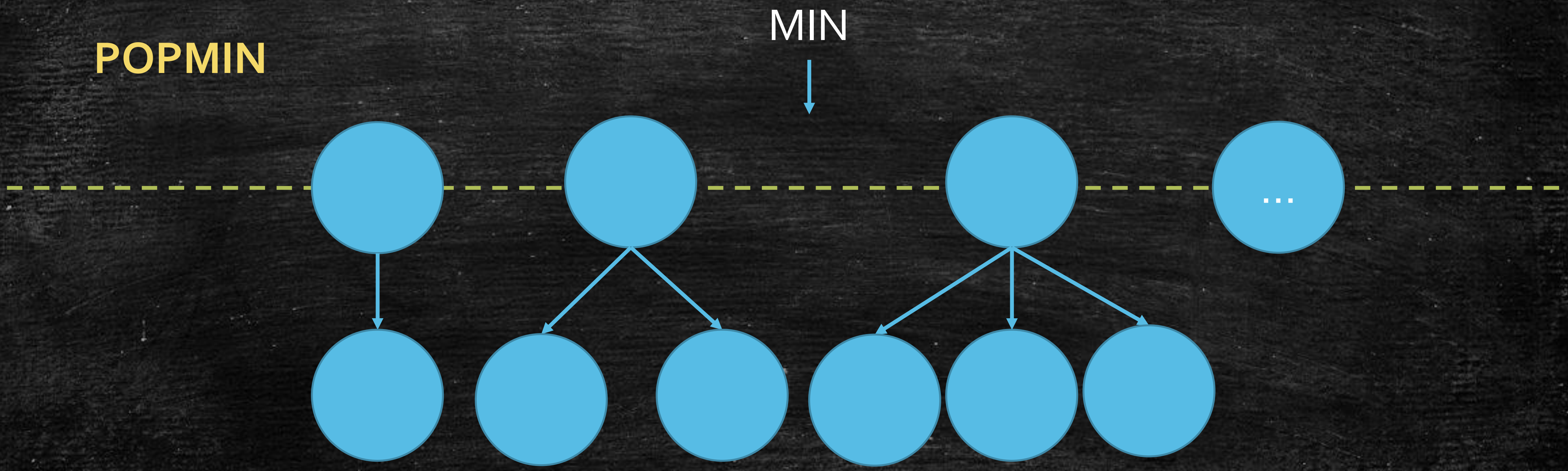
Before move the MIN pointer: Merge



Before move the MIN pointer: Merge



If we do not do anything?

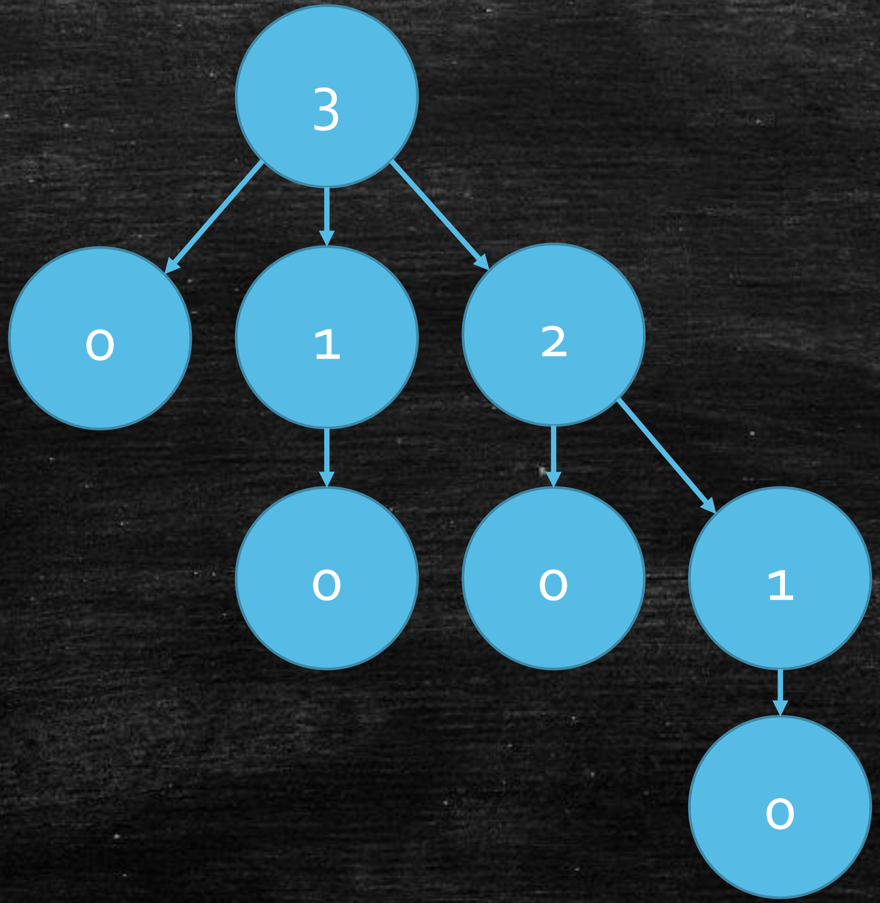


Degree k root is size $k + 1$, number of roots = largest degree = $\sqrt{n}$.

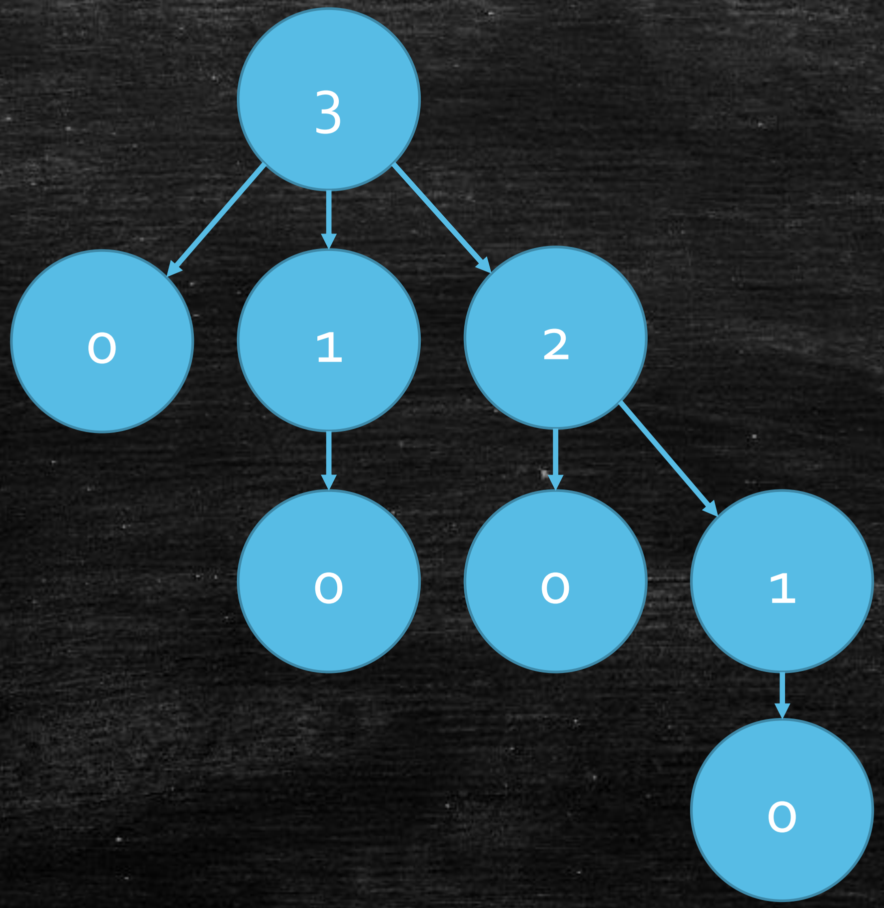
How to make a degree k tree large?

- Build a **good tree** at the beginning.
- We can not break the **good** property a lot!

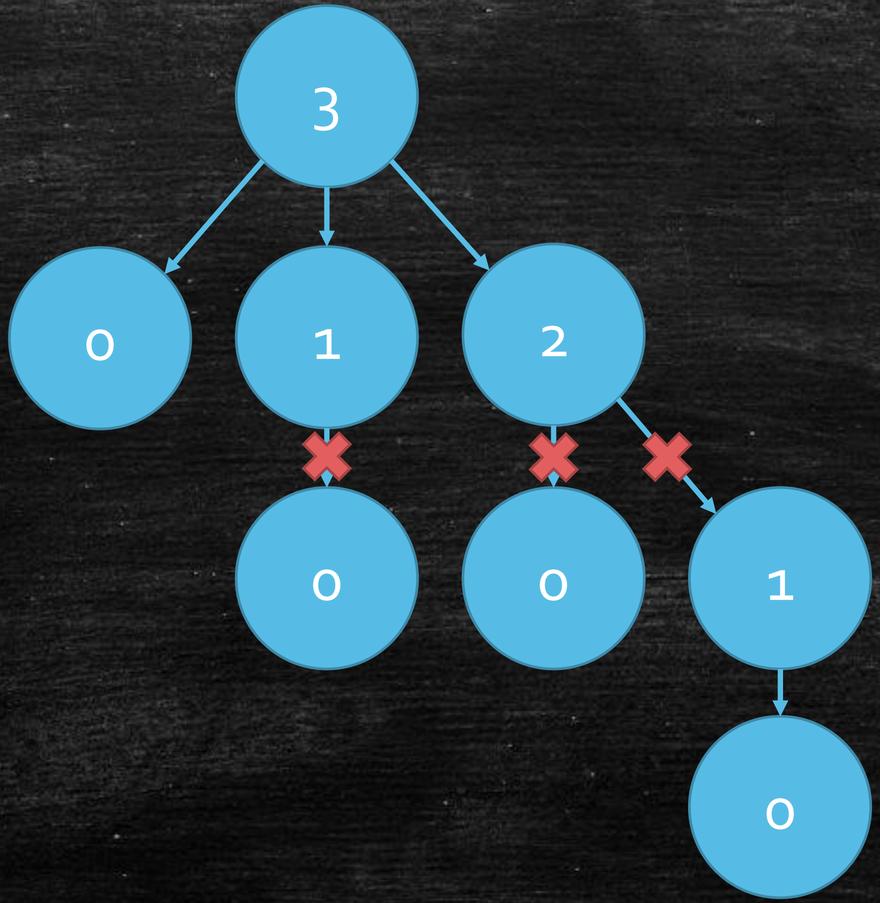
Build a Good Tree (Recall Binomial Heap)



Will it become bad?

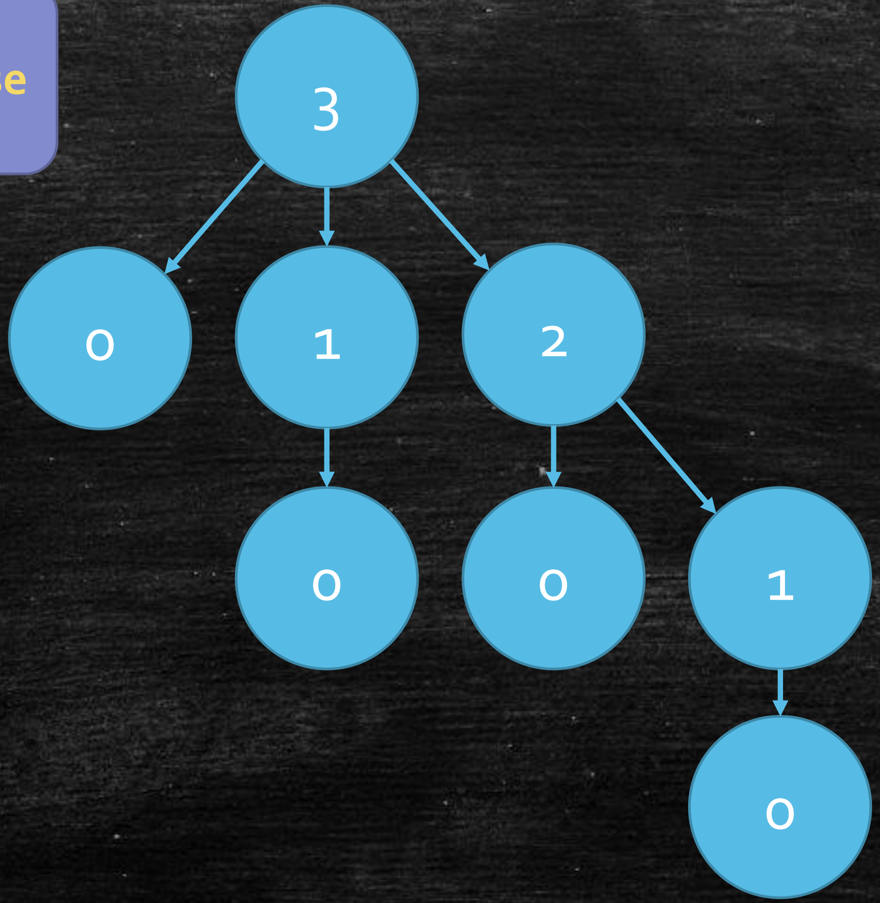


Will it become bad?



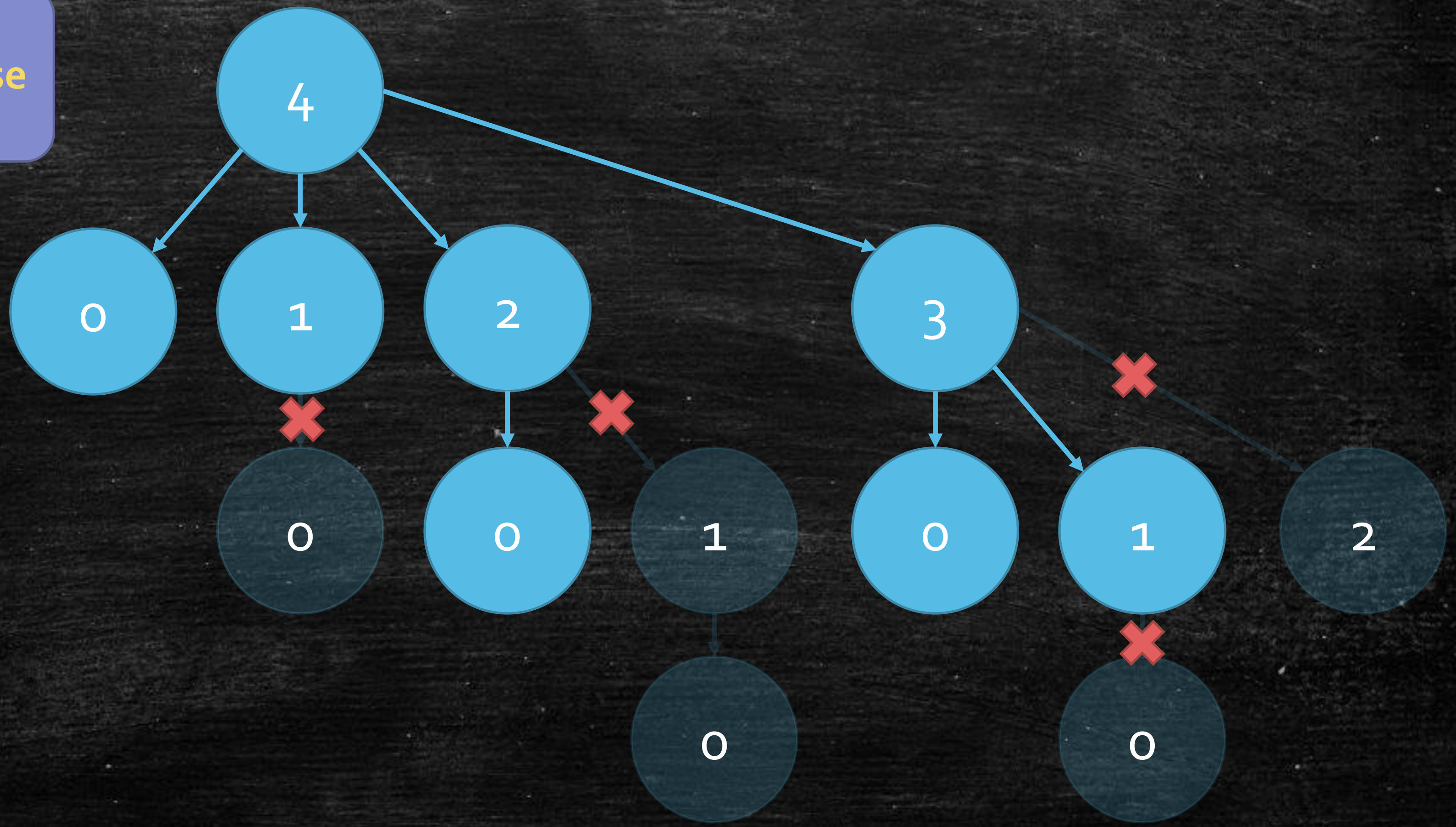
Build a Good Tree (Recall Binomial Heap)

We only allow each non-root node to **lose one child.**



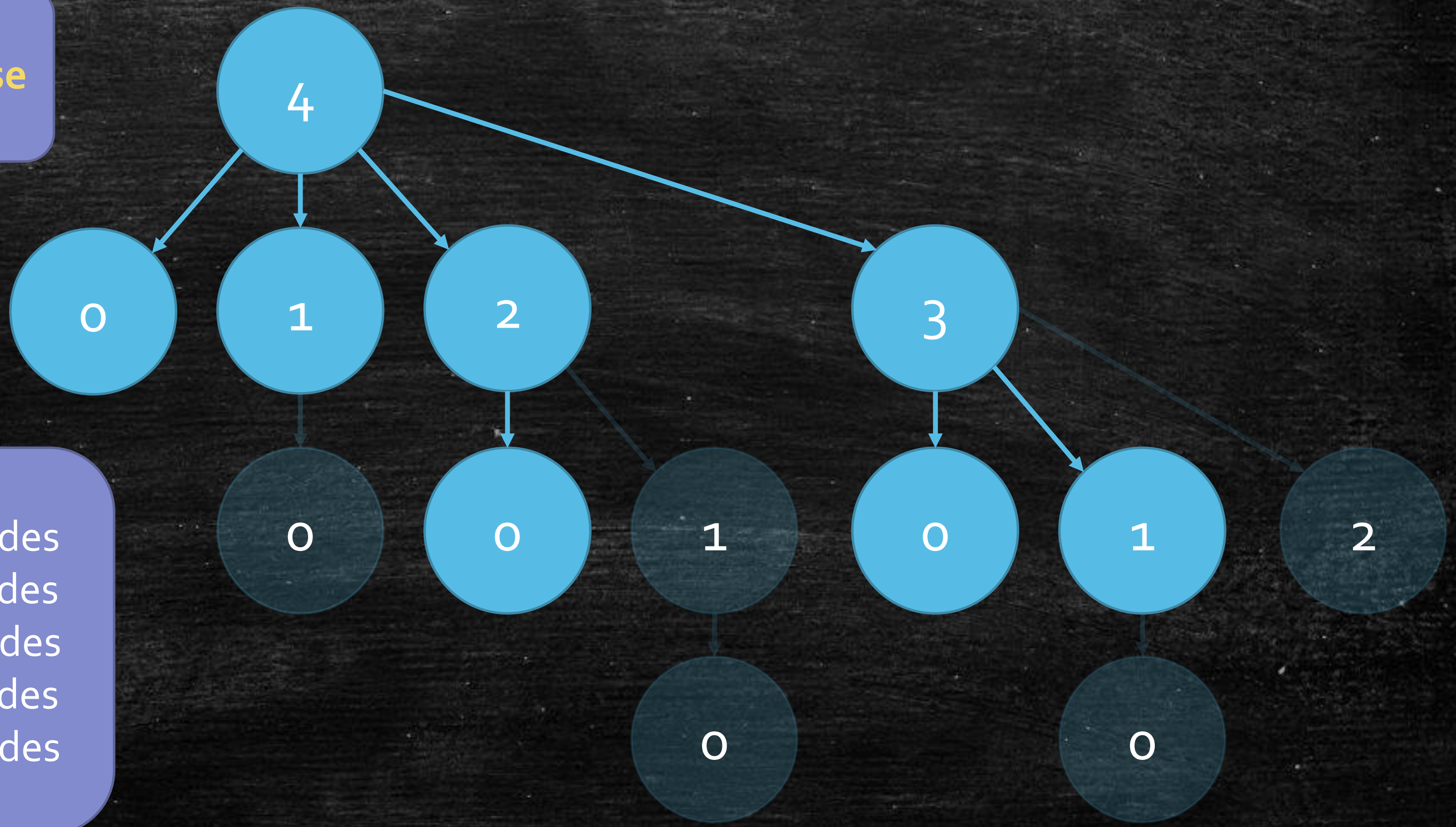
Maximum Broken tree

We only allow each non-root node to **lose one child.**



Maximum Broken tree

We only allow each non-root node to **lose one child.**



Degree 0 subtree: 1 nodes
Degree 1 subtree: 1 nodes
Degree 2 subtree: 2 nodes
Degree 3 subtree: 3 nodes
Degree 4 subtree: 5 nodes

Conclusion

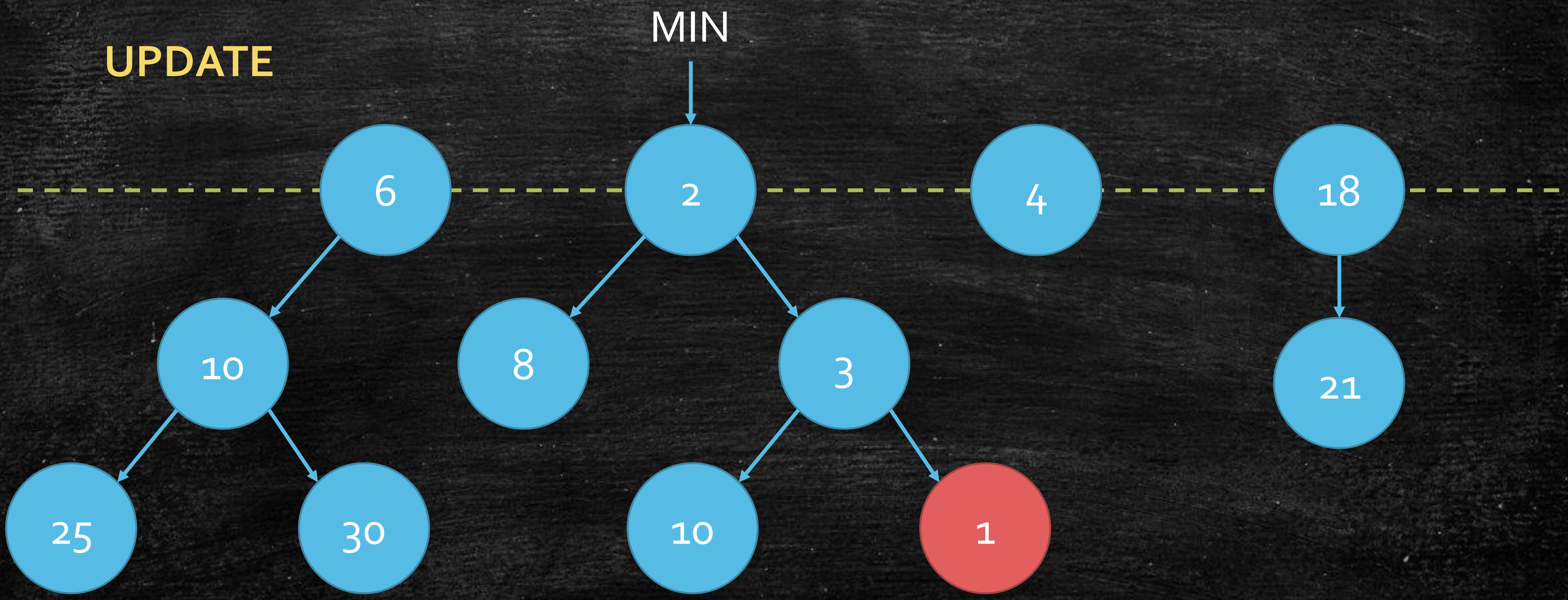
- Degree k root contains
- At least $F(k)$ nodes
- $F(k) = \sum_{i=1}^{k} fib(i) = O(C^k)$
- Max degree is around $D = O(\log n)$.

How to maintain this property?

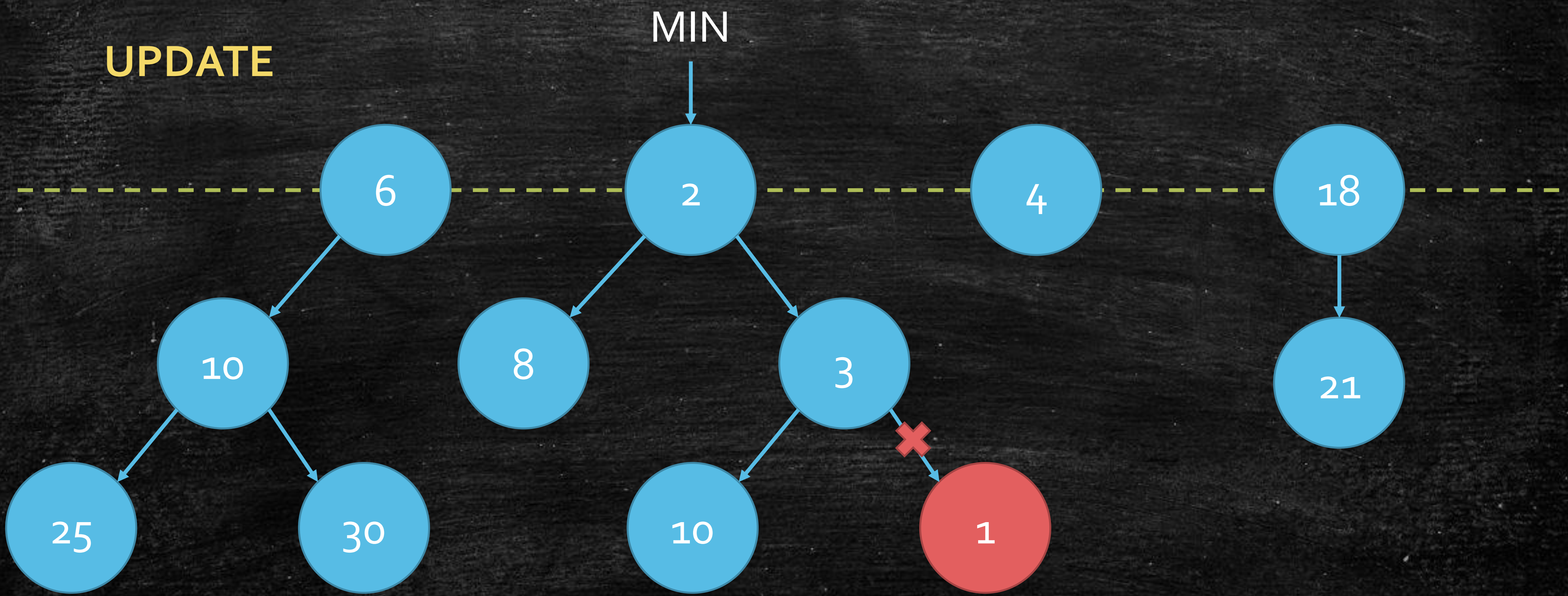
Cascading Cut

Fibonacci Heap: Cascading Cut

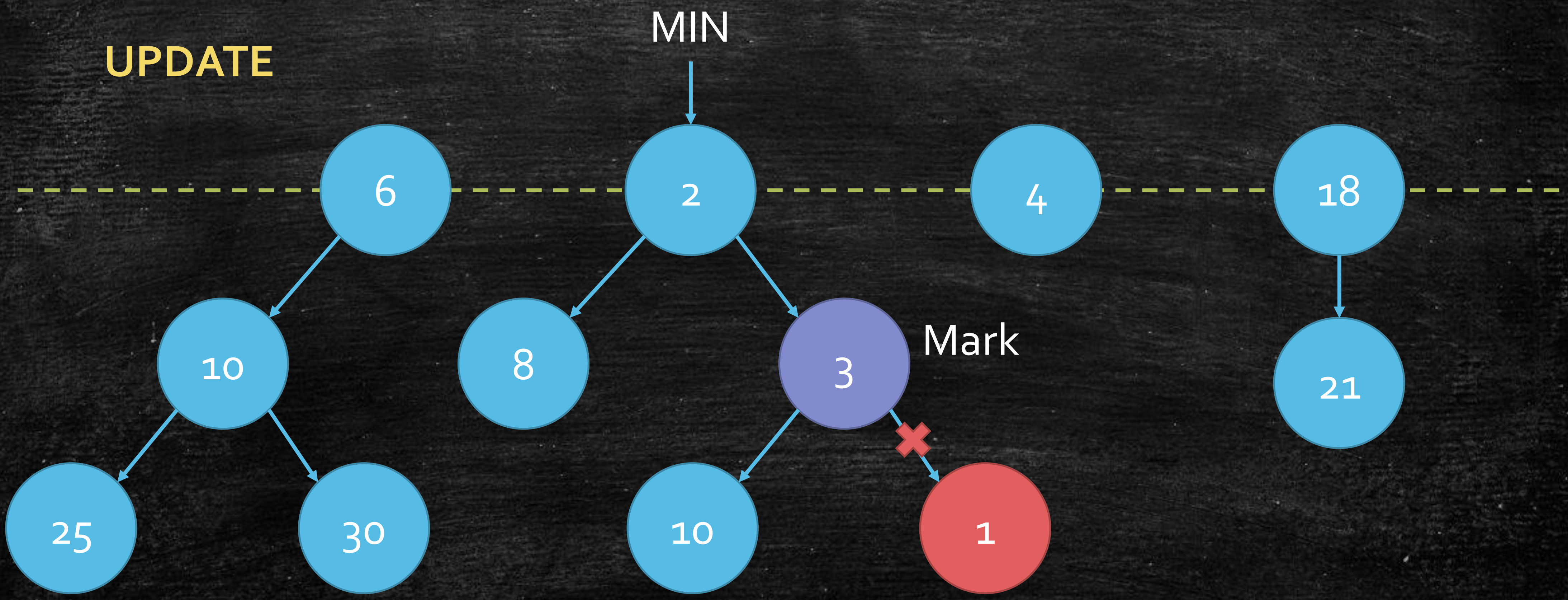
UPDATE



Fibonacci Heap: Cascading Cut

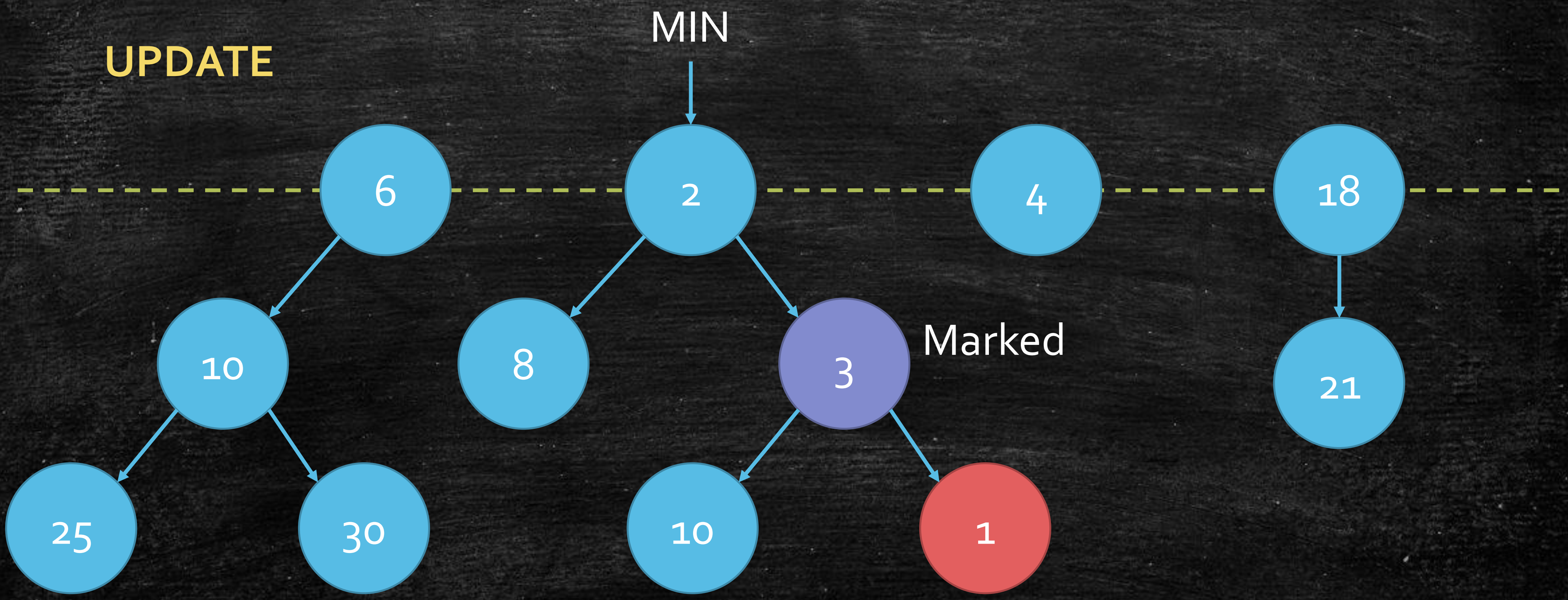


Fibonacci Heap: Cascading Cut



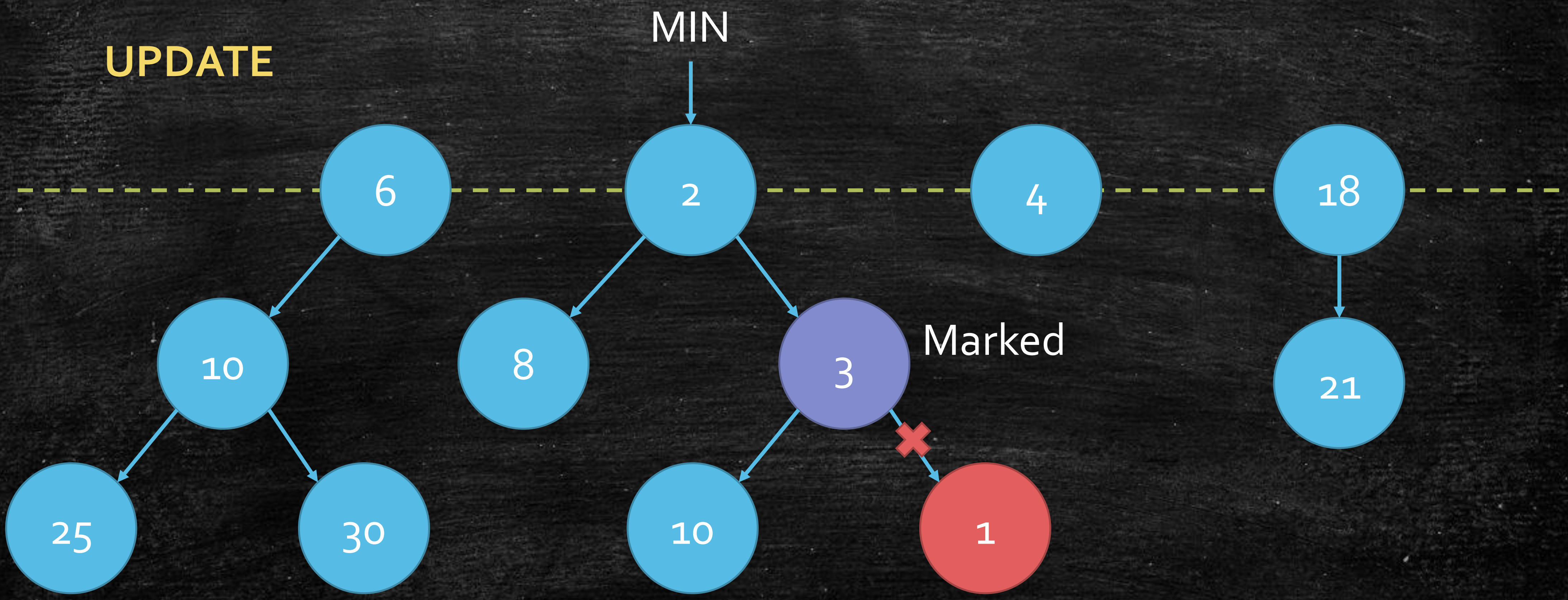
Fibonacci Heap: Cascading Cut

UPDATE

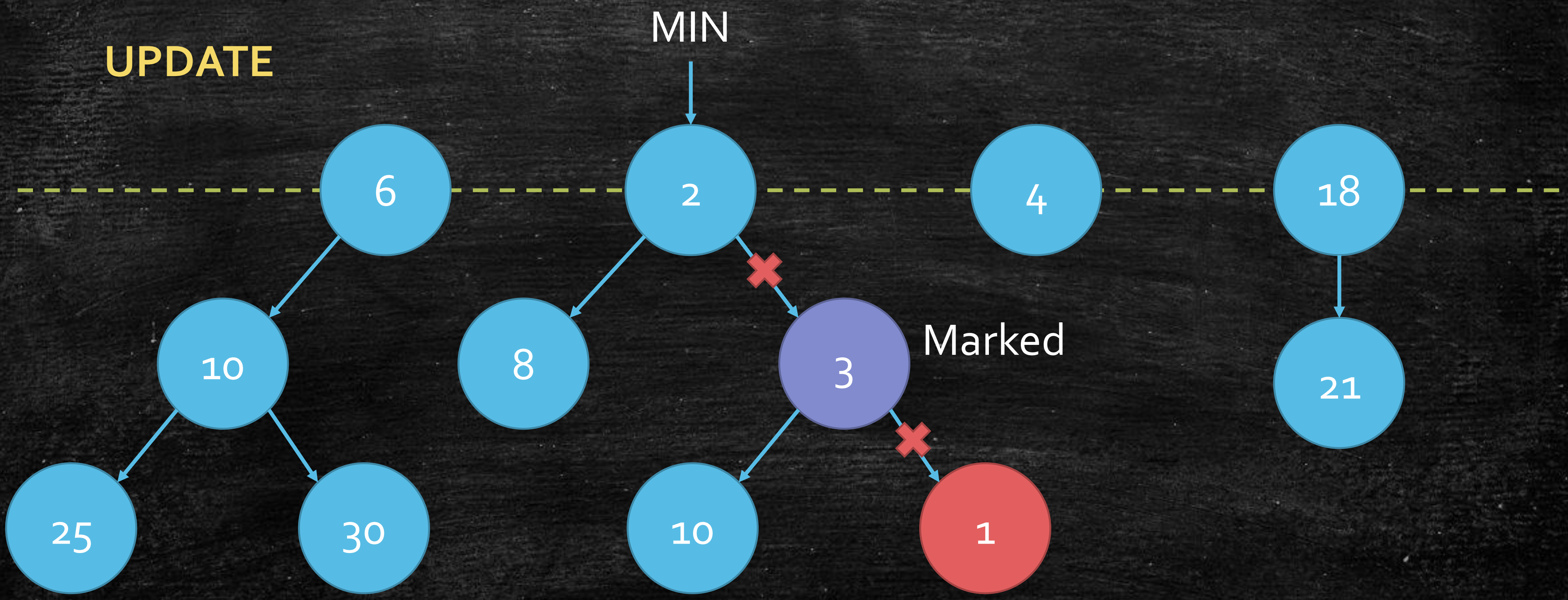


Fibonacci Heap: Cascading Cut

UPDATE



Fibonacci Heap: Cascading Cut



Still many problems...

- What we have:
- We can control $D = O(\log n)$ before moving Min Pointer.
- **But!**
- How long we pay for the **cascading cut**?
- How long we pay for the **root merging**?
- They may be very large at one time
- But we can use **amortized analysis**.

Time Complexity: Update

- Original cut: 1
- Cascading cut: $<$ #marked nodes. (called m)
- Time: $\boldsymbol{O(m)}$

Time Complexity: POPMIN

- Delete Min
  - Time = $O(D)$
- Merge
  - D is max degree
  - #roots(before merging) $\le$ #roots(before POPMIN) $+ D$
  - Time = $O(t^- + D - t^+)$
- Pointer move to new Min
  - Time = $O(t^+)$
- Totally: $\boldsymbol{O(t^- + 2D)}$

Amortized Analysis: Potential Function

- C: actual cost of an operation
- $\hat{C}$: Amortized cost of an operation
- Some operation may have **small** C make **later operation** bad.
- Let it pay for it by **itself**, so we let $\hat{C} = C + \delta \cdot \Delta\Phi$.
- Φ is a function to evaluate current state.

A chosen constant.

- $\sum \hat{C} = \sum C + \sum \delta \cdot \Delta\Phi = \sum C + \delta \cdot \Phi$

A chosen constant.

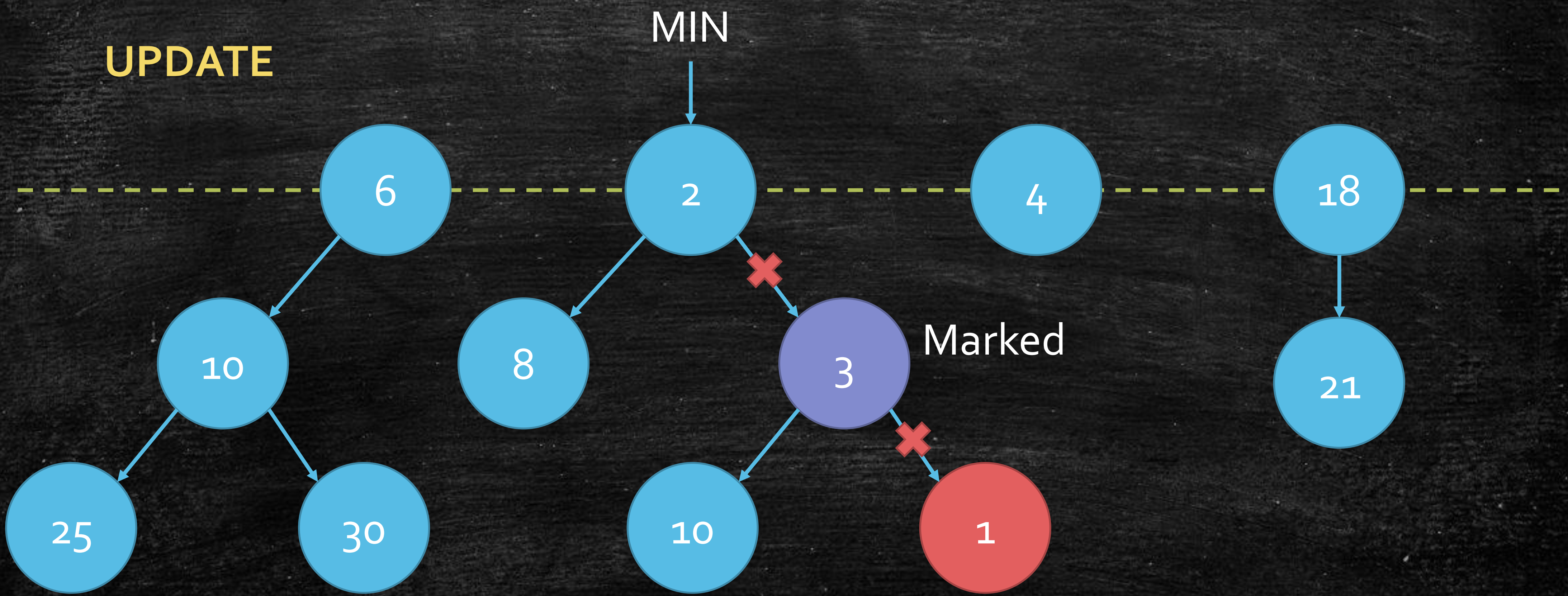
Amortized Analysis: Stack

- Operations
  - Pop all elements one by one.
  - Push one element.
- Potential Function
  - $\Phi = \#\text{elemnts}$
- **Push**
  - $C = \boldsymbol{O(1)}$
  - $\hat{C} = O(1) + \delta \cdot 1 = \boldsymbol{O(1)}$
- **Pop**
  - $C = \boldsymbol{O(k)}$
  - $\hat{C} = O(k) + \delta \cdot (-k) = \boldsymbol{O(1)}$

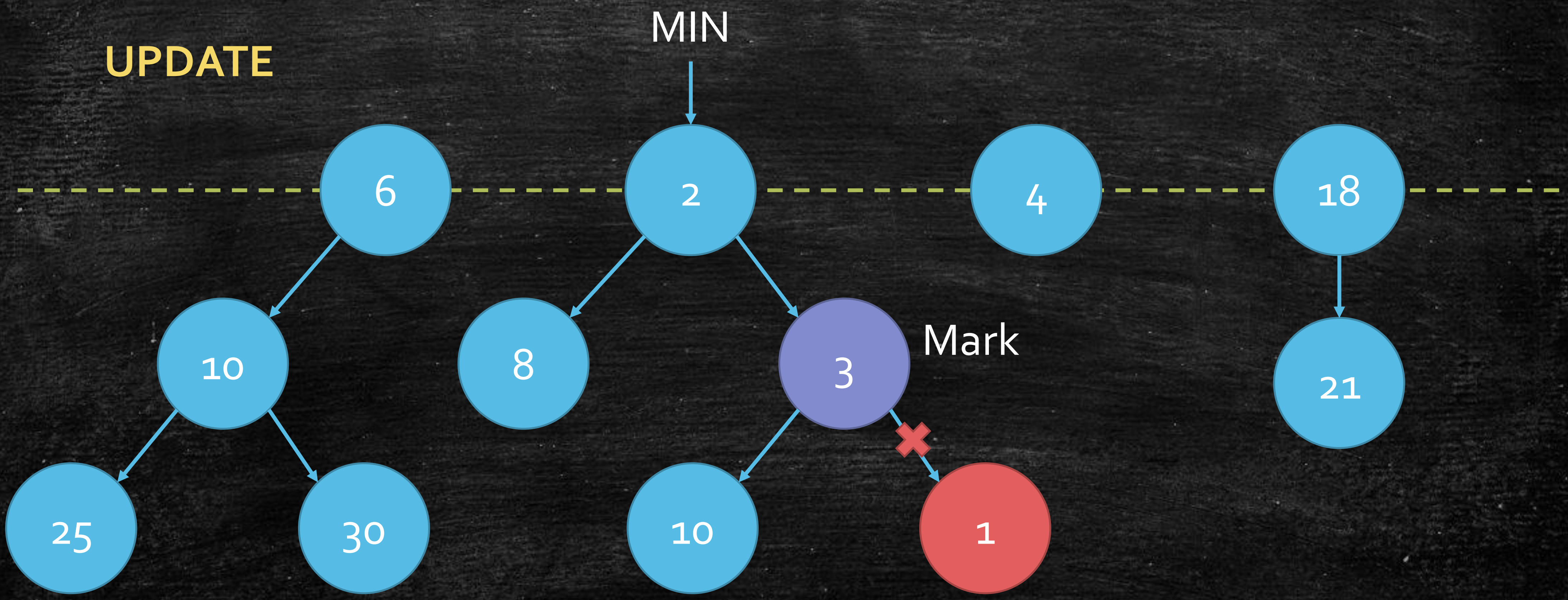
Amortized Analysis: Fibonacci Heap

- **Update:** $O(m)$
- **Pop Min:** $O(t^{-} + D)$
- What is bad?
  - #marked nodes
  - #roots
- Potential Function: $\boldsymbol{\Phi = t + 2m}$
- Why we need $2m$?
- m has two bad things
  - One more cut!
  - One potential root!

Fibonacci Heap: Cascading Cut



Fibonacci Heap: Cascading Cut



Amortized Analysis: Fibonacci Heap

- **Update:** $O(m)$
- **Pop Min:** $O(t^{-} + D)$
- Potential Function： $\boldsymbol{\Phi = t + 2m}$
- **Update**
  - $\hat{C} = O(\#\mathrm{CC} + 1) + \delta \cdot \Delta\Phi = \mathrm{O}(\#\mathrm{CC} + 1) + \delta \cdot (-\#\mathrm{CC} + 1) = \boldsymbol{O(1)}$
  - $\#CC$ cascading cuts, remove $\#CC$ mark
  - one basic cut, one more mark

We can choose it

Time Complexity: POPMIN

- Delete Min
  - Time = $O(D)$
- Merge
  - D is max degree
  - #roots(before merging) $\leq$ #roots(before POPMIN) $+ D$
  - Time = $O(t^- + D - t^+)$
- Pointer move to new Min
  - Time = $O(t^+)$
- Totally: $\boldsymbol{O(t^- + 2D)}$

Amortized Analysis: Fibonacci Heap

- **Update:** $O(m)$
- **Pop Min:** $O(t^- + D)$
- Potential Function： $\boldsymbol{\Phi = t + 2m}$
- **Update**
  - $\hat{C} = O(\#CC + 1) + \delta \cdot \Delta\Phi = \mathrm{O}(\#\mathrm{CC} + 1) + \delta \cdot (-\#\mathrm{CC} + 1) = \boldsymbol{O(1)}$
  - $\#CC$ cascading cuts, remove $\#CC$ mark
  - one basic cut, one more mark

We can choose it

- **Pop Min**
  - $\hat{C} = O(t^- + 2D) + \delta \cdot \Delta t \leq O(t^- + 2D) + \delta \cdot (D - t^-) = \boldsymbol{O(D)} = \boldsymbol{O(\log n)}$
  - $t^+ \leq D$

We can choose it

Conclusion

Dijkstra + Fibonacci Heap = $O(|E| + |V|\log|V|)$

Today's goal

- Learn **Dijkstra**
  - Why it is **correct?**
  - How to **design** if you are Dijkstra?
  - How to use **Heap** to improve Dijkstra?
  - How to use **Data Structures** to improve **Algorithms**?
- Learn **Amortized Analysis**
  - Roughly get the idea is ok.