Greedy

What is Greedy?

Follows the “looks good” strategy.

Recap the Graph Algorithm

- DFS (walking in a maze)
  - If we can explore, then explore.
  - If we can not explore, backtrack.
  - Do not re-visit a vertex.
  - Applications
    - Cycle
    - Topological
    - SCC

Recap the Graph Algorithm

- BFS (waterfront)
  - 1 step from r
  - 2 steps from r
  - …
  - Application
    - Shortest Path

Recap the Graph Algorithm

- Dijkstra (a generalized BFS)
  - Explore s.
  - Explore the closet vertex from s.
  - Explore the second closest vertex from s.
  - …
  - We can use Fibonacci heap to improve it.
- Bellman-Ford

Are they Greedy?

Do we have any other Greedy?

Examples

- Finding Shortest Path
  - Dijkstra.
- Finishing homework
  - Keep finishing the one with the **closest** deadline.

Is that optimal?

Formalize the problem

- **Input:** n homework, each homework j has a size s_j, and a deadline d_j.
- **Output:** output a time schedule of doing homework!

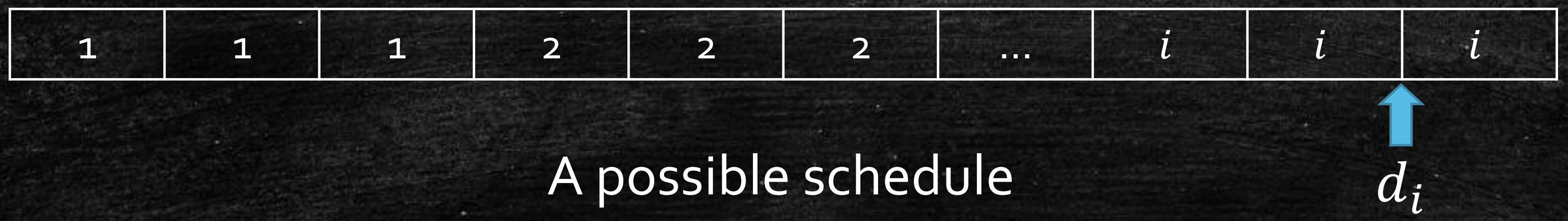
Algorithm

- Greedy
  - Keep finishing the homework with the **closest** deadline.
- Prove it is optimal.
- What is optimal?
- Claim: If we can not finish all the homework by the greedy order, then no one can finish all the homework on time.

Discussion

Proof

- Claim: If we can not finish all the homework by the greedy order, then no one can finish all the homework on time.
- Proof:
  - If there exist i, finished later than d_i, what do we have?



A possible schedule

Proof

- Claim: If we can not finish all the homework by the greedy order, then no one can finish all the homework on time.
- Proof:
  - If there exist i, finished later than d_i, what do we have?

$$\sum_{j \le i} s_i \ge d_i$$

No one can not finish all the homework (1~i) before d_i.

1	1	1	2	2	2	…	i	i	i

Greedy:$\forall j < i, d_j < d_i$

A possible schedule

d_i

Whatever order, the last homework (1~i) can not be on time

Minimum Spanning Tree

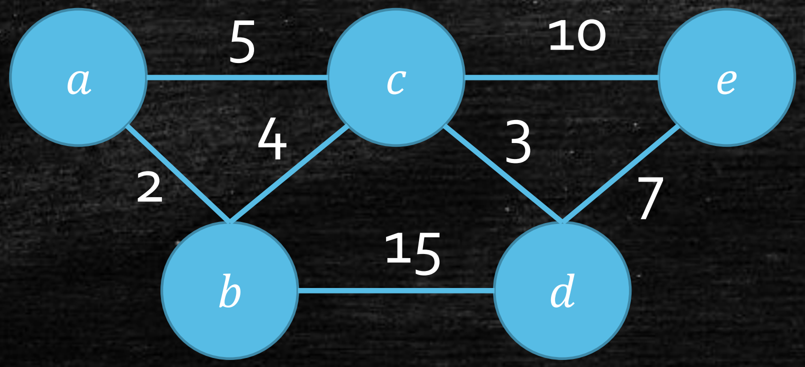
Prime & Kruskal

Spanning Tree

- **Input:** Given a connected undirected graph $G = (V, E)$
- **Output:** A spanning tree of G is, i.e., a subset of edges that forms a tree and contains all the vertices in G.
- Applications
  - Building a network, connecting all hubs via minimum number of cables.
- Solutions
  - BFS, DFS.

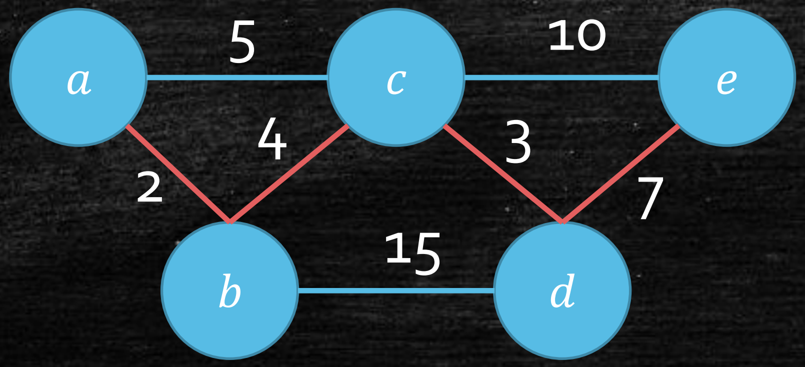
Minimum Spanning Tree

- **Input:** Given a connected undirected graph $G = (V, E)$, and a weight function $w(e)$ for each $e \in E$.
- **Output:** A spanning tree of G is, i.e., a subset of edges, with minimized total weight.
- Applications
  - Building a network, connecting all hubs via minimum number of cables.



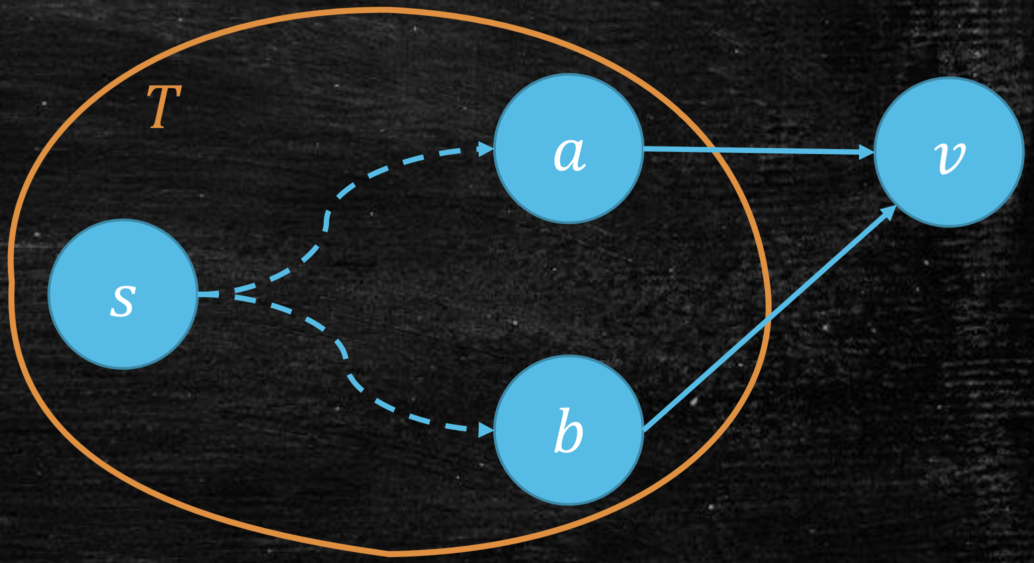
Minimum Spanning Tree

- **Input:** Given a connected undirected graph $G = (V, E)$, and a weight function $w(e)$ for each $e \in E$.
- **Output:** A spanning tree of G is, i.e., a subset of edges, with minimized total weight.
- Applications
  - Building a network, connecting all hubs via minimum number of cables.



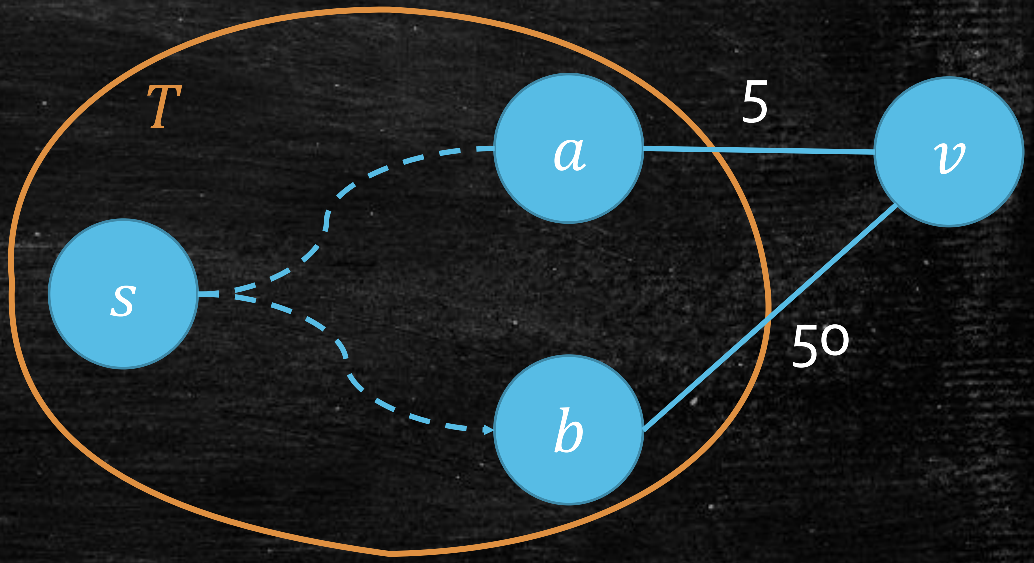
Dijkstra's growing idea

- Given a small SPT,
- choose a proper vertex v to find a larger SPT.
- New Plan for MST:
- Given a small MST,
- choose a proper vertex v to find a larger MST.



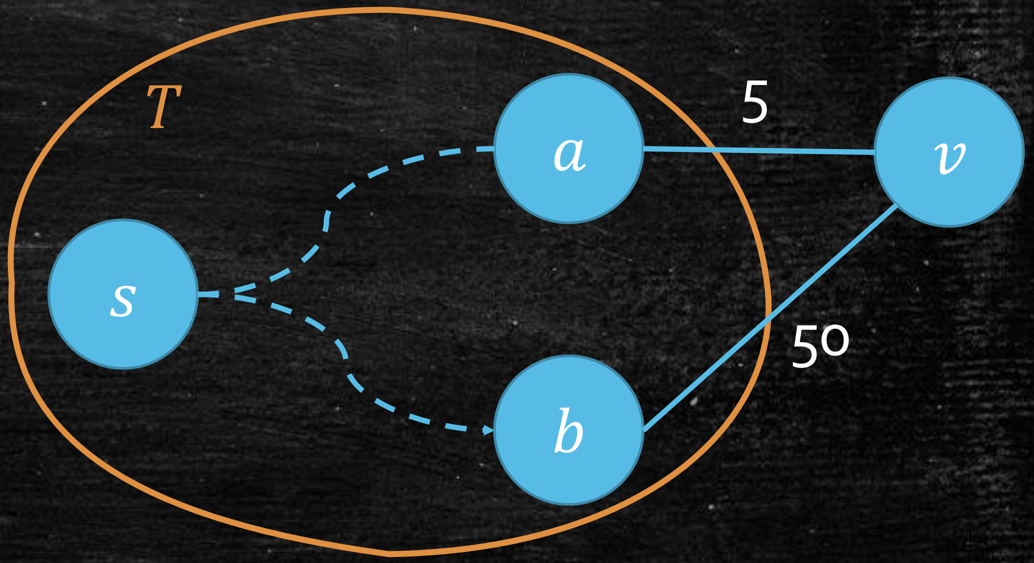
Prim's growing idea

- Given a small MST,
- choose a proper vertex v to find a larger MST.
- Which v is good?
- Dijkstra: v with smallest T-distance to s.
- Now: v with smallest cost!



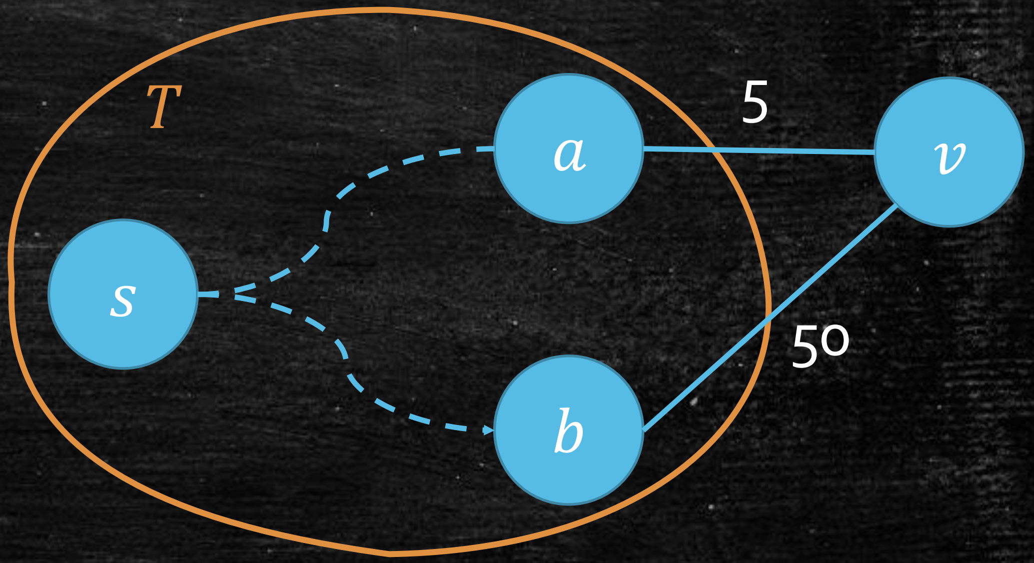
Prim’s growing idea

- Given a small MST,
- choose a proper vertex v to find a larger MST.
- Grow v with smallest cost!
- Is it correct?
- Challenge:
  - How to define small MST



How to define small MST?

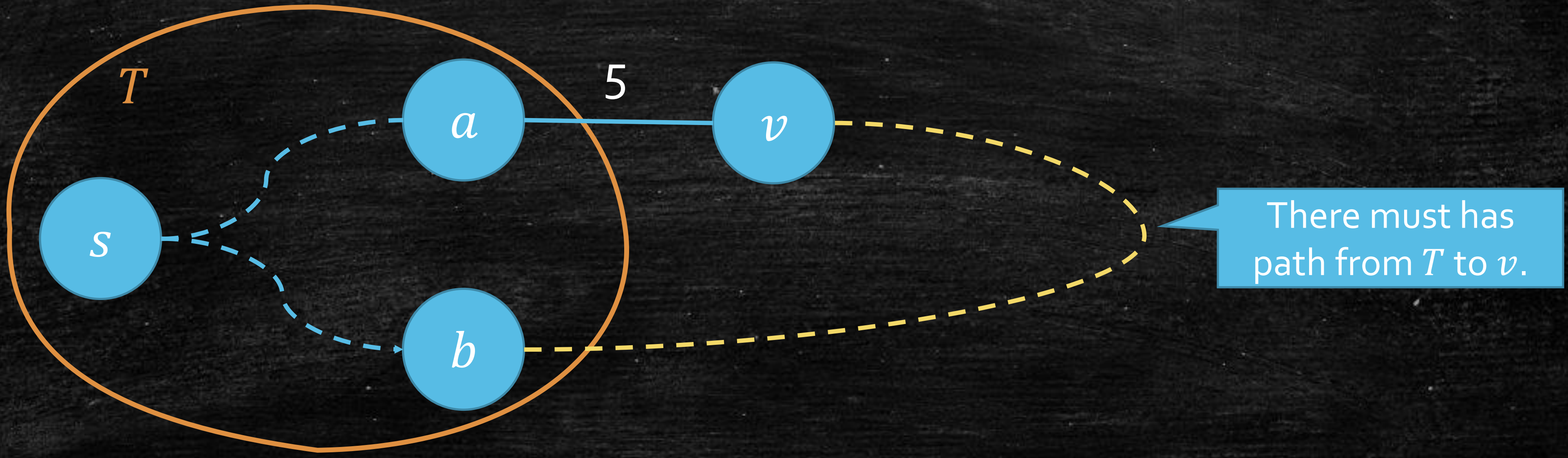
- $T = (V', E')$ is a small MST if it is an MST for V'.
- Problem
  - are those edges in T still ok?
- A better choice:
- T is a P-MST (Partial MST) if it is a part of a complete MST for G.



Correctness of Prim's Growing idea

- Let's say T^* is the complete MST that contains T, and suppose $(\mathrm{a}, \mathrm{v}) \notin T^*$.

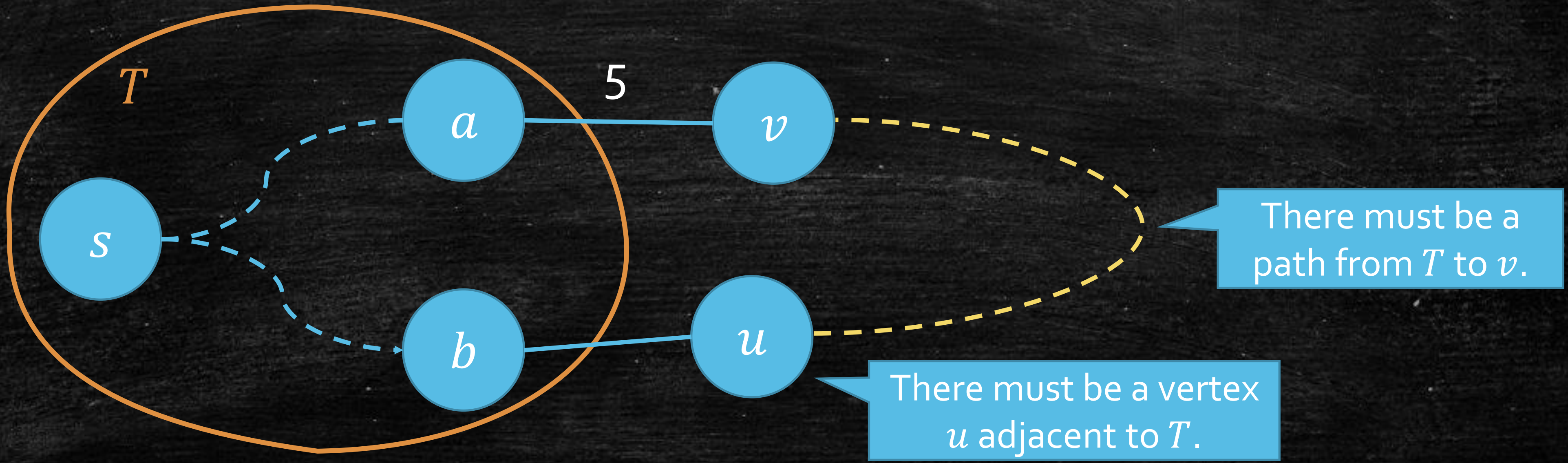
- **Given:** a small P-MST T.
- **Want:** a larger P-MST.
- Can we explore v (smallest cost) into T?



Correctness of Prim's Growing idea

- Let's say T^* is the complete MST that contains T, and suppose $(\mathrm{a}, \mathrm{v}) \notin T^*$.

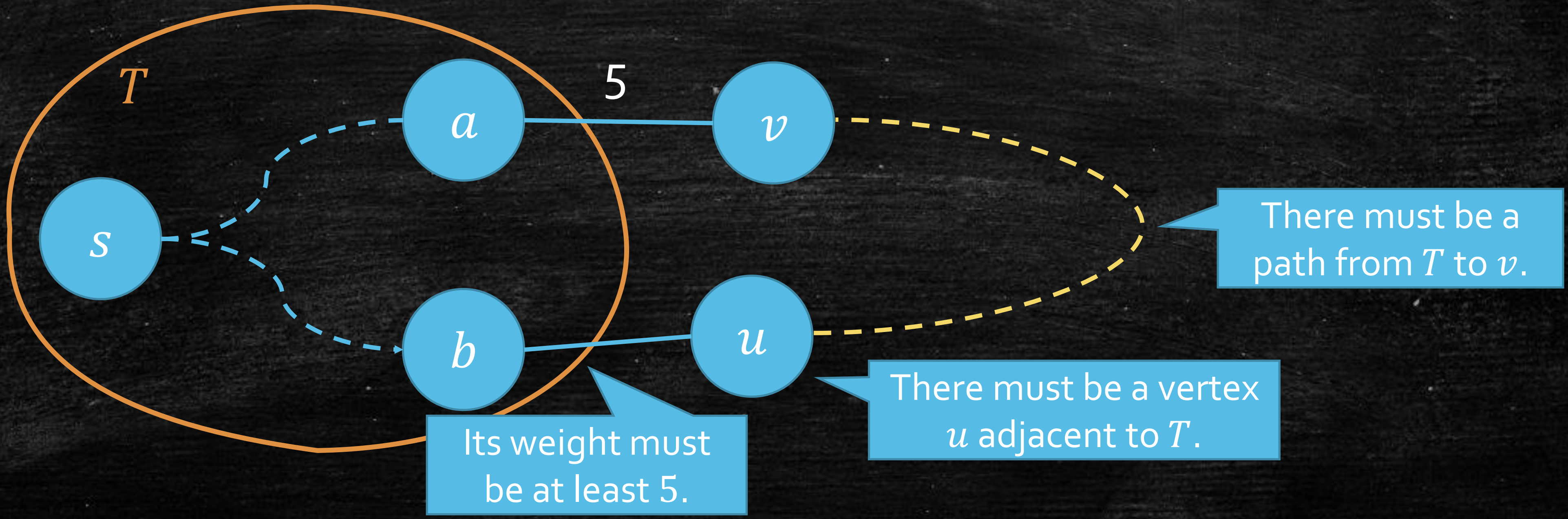
- **Given:** a small P-MST T.
- **Want:** a larger P-MST.
- Can we explore v (smallest cost) into T?



Correctness of Prim's Growing idea

- Let's say T^* is the complete MST that contains T, and suppose $(\mathrm{a}, \mathrm{v}) \notin T^*$.

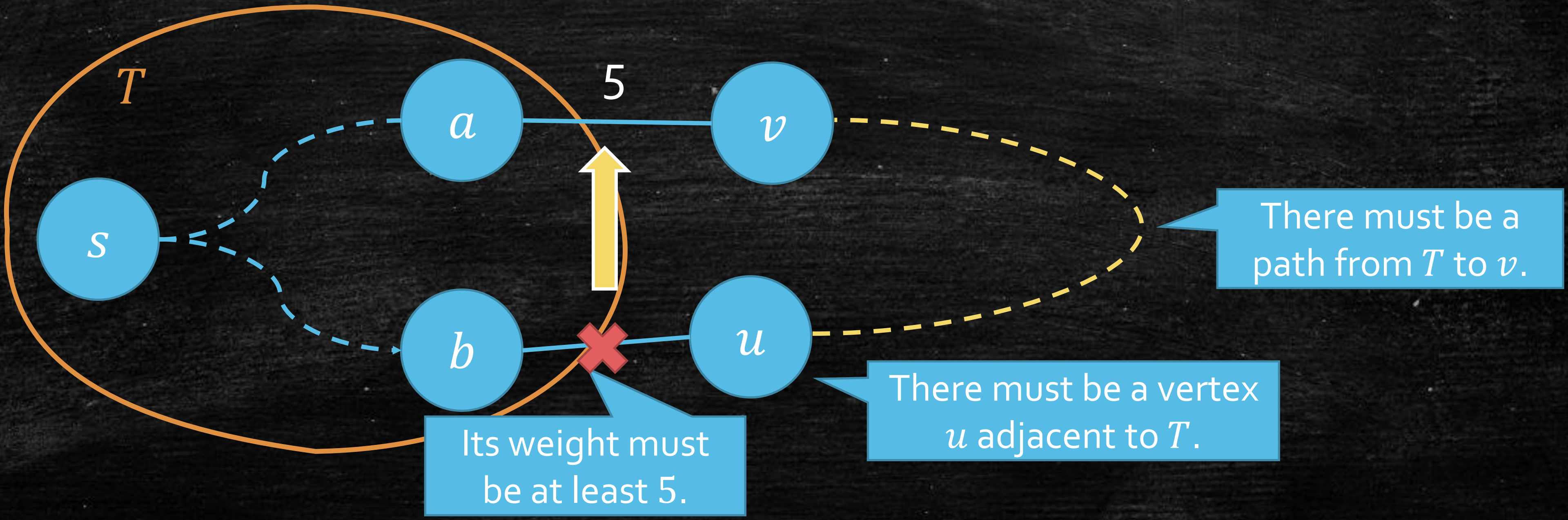
- **Given:** a small P-MST T.
- **Want:** a larger P-MST.
- Can we explore v (smallest cost) into T?



Correctness of Prim's Growing idea

- Let's say T^* is the complete MST that contains T, and suppose $(\mathrm{a}, \mathrm{v}) \notin T^*$.

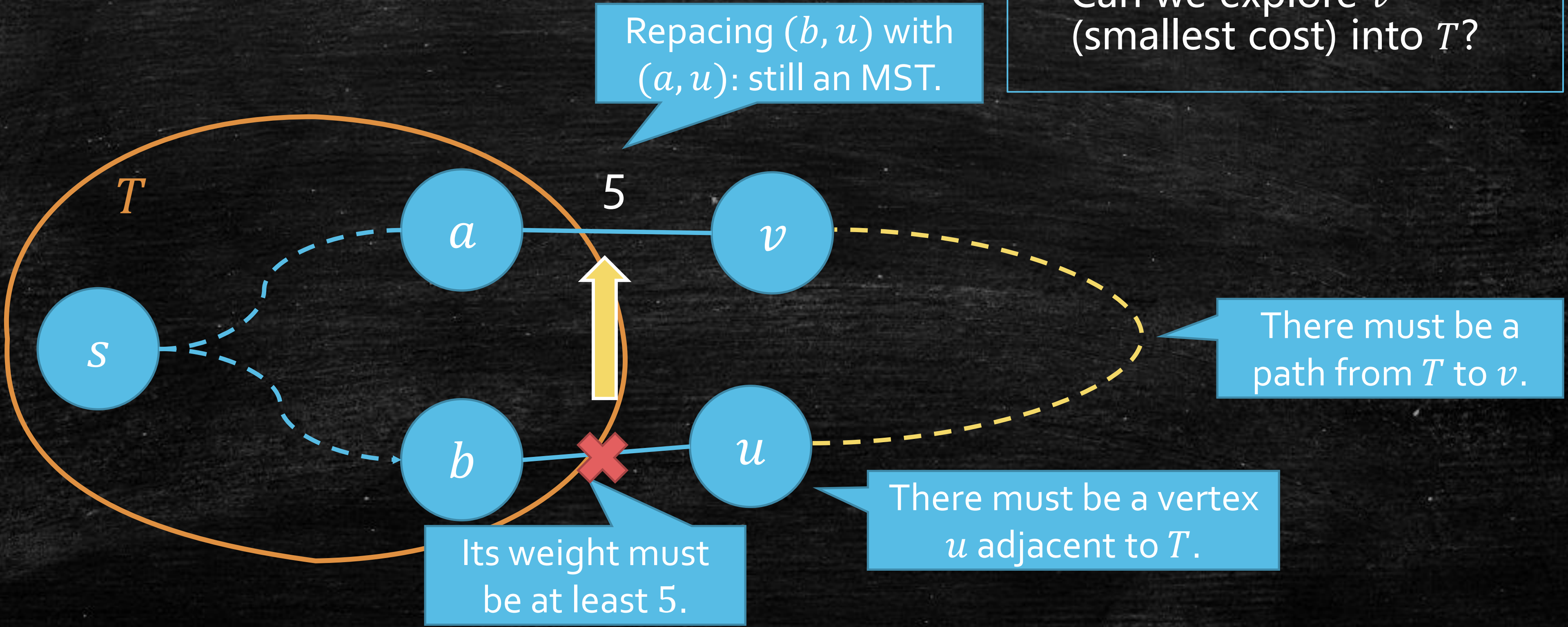
- **Given:** a small P-MST T.
- **Want:** a larger P-MST.
- Can we explore v (smallest cost) into T?



Correctness of Prim's Growing idea

- Let's say T^* is the complete MST that contains T, and suppose $(\mathrm{a}, \mathrm{v}) \notin T^*$.

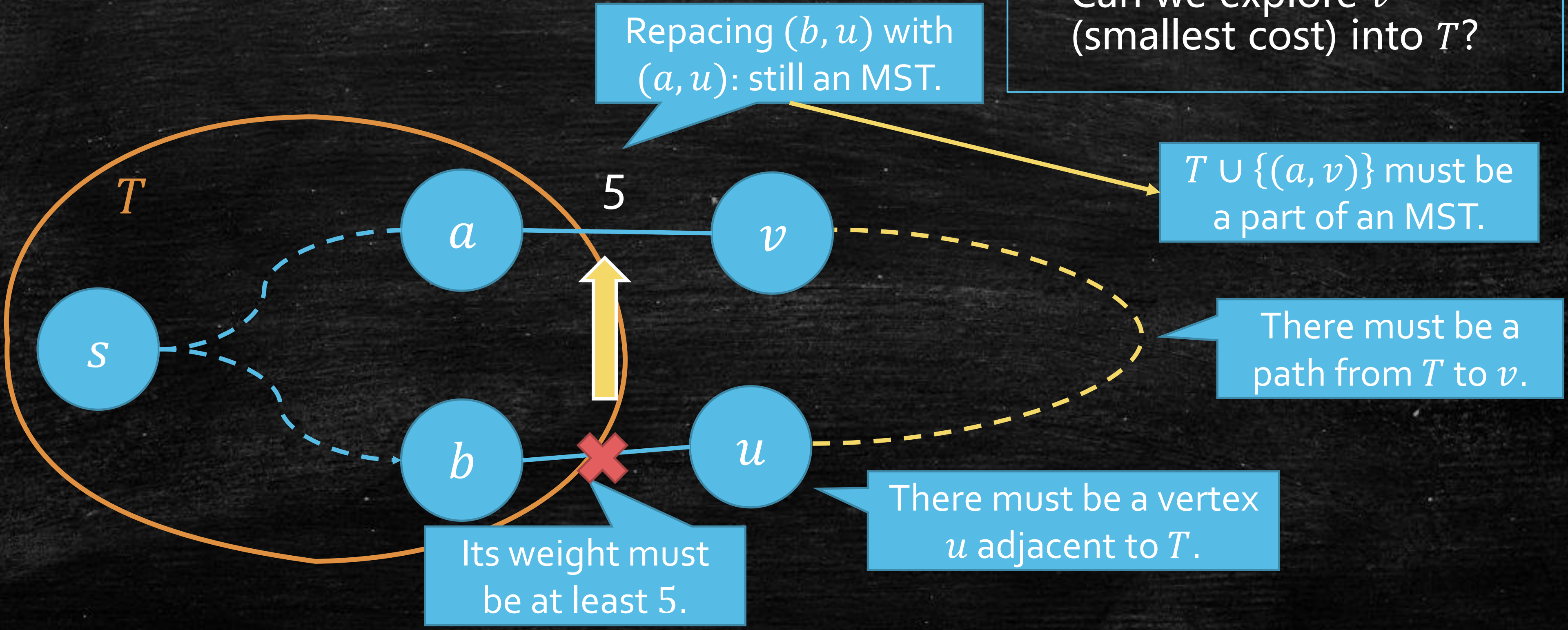
- **Given:** a small P-MST T.
- **Want:** a larger P-MST.
- Can we explore v (smallest cost) into T?



Correctness of Prim's Growing idea

- Let's say T^* is the complete MST that contains T, and suppose $(\mathrm{a}, \mathrm{v}) \notin T^*$.

- **Given:** a small P-MST T.
- **Want:** a larger P-MST.
- Can we explore v (smallest cost) into T?



Prim Algorithm [Jarník '30, Prim '57, Dijkstra '59]

Prim$(G = (V, E))$

1. Initialize

- $T \leftarrow \{\,\}, S \leftarrow \{s\}$; #s is an arbitrary vertex.
- $cost[s] = 0$, $cost[v] \leftarrow \infty$ for all v other than s.
- $cost[v] \leftarrow w(s, v), pre[v] = s$ for all $(s, v) \in E$.

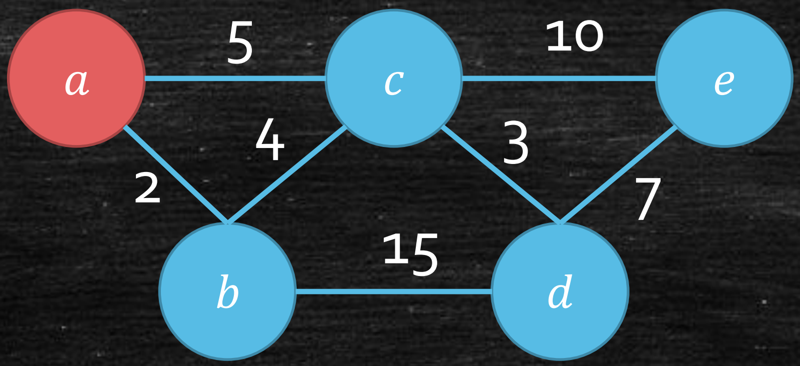
2. Explore

- Find $v \notin S$ with smallest $cost[v]$.
- $S \leftarrow S + \{v\}$; $T \leftarrow T + \{(pre[v], v)\}$

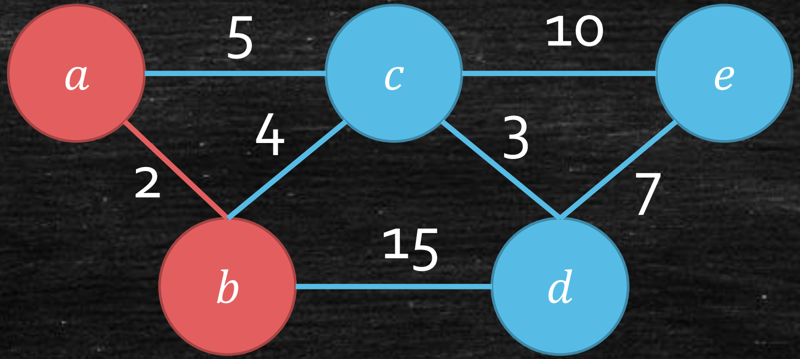
3. Update $cost[u]$

- $cost[u] = \min\{cost[u], w(v, u)\}$ for all $(v, u) \in E$
- If $cost[u]$ is updated, then $pre[u] = v$.

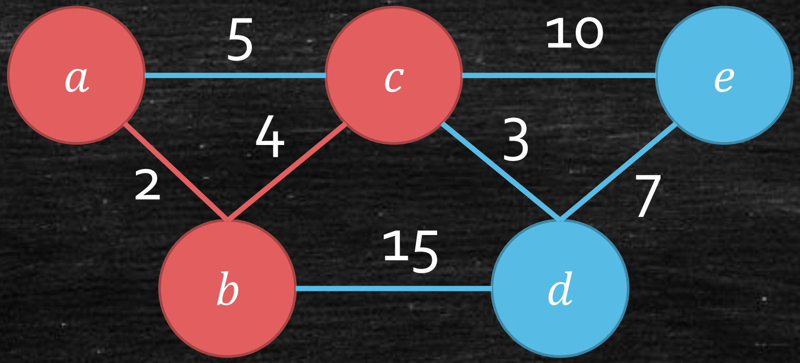
Sample Run



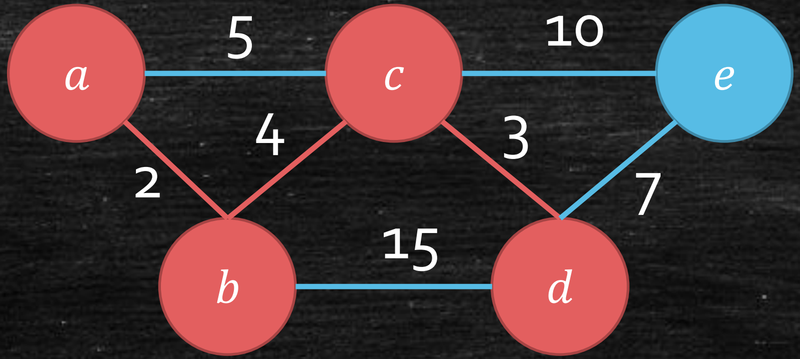
Sample Run



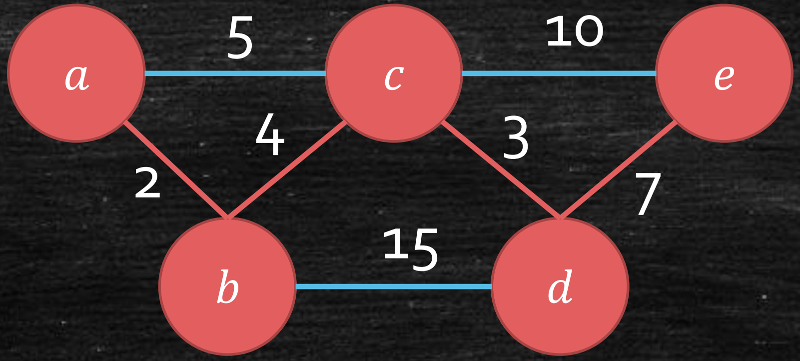
Sample Run



Sample Run



Sample Run



Running Time

- I believe you know how to analyze it:
- We can do it in $O(|E| + |V| \log|V|)$.

Kruskal Algorithm [Kruskal 1956]

- Another Greedy!

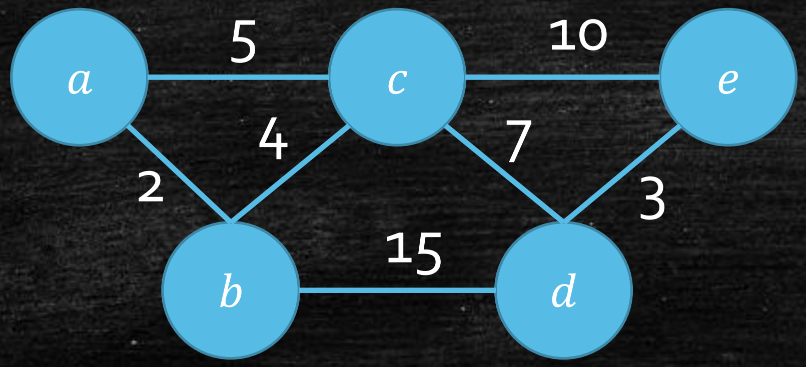
Kruskal$(G = (V, E))$

- Sort the edge set E to descending order.
- For each $e \in E$ in descending order
  - If e do not create a **cycle**, then choose it.

Kruskal Algorithm

Kruskal$(G = (V, E))$

- Sort the edge set E to ascending order.
- For each $e \in E$ in ascending order
  - If e do not create a **cycle**, then choose it.

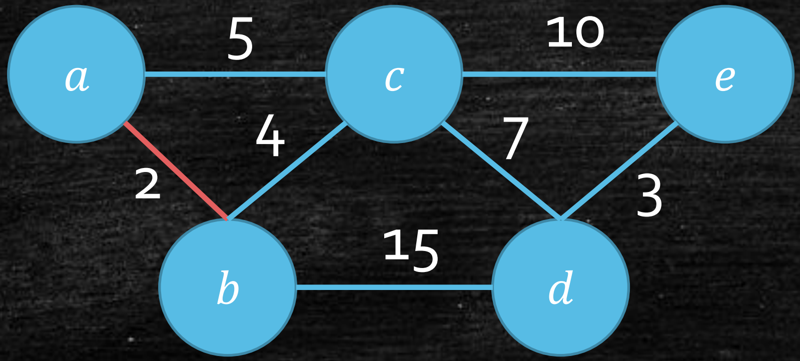


2	3	4	5	7	10	15

Kruskal Algorithm

Kruskal$(\boldsymbol{G} = (\boldsymbol{V}, \boldsymbol{E}))$

- Sort the edge set E to ascending order.
- For each $e \in E$ in ascending order
  - If e do not create a **cycle**, then choose it.

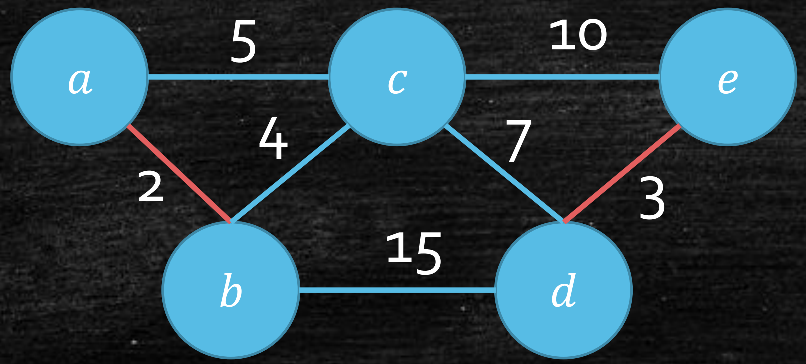


2	3	4	5	7	10	15

Kruskal Algorithm

Kruskal$(G = (V, E))$

- Sort the edge set E to ascending order.
- For each $e \in E$ in ascending order
  - If e do not create a **cycle**, then choose it.

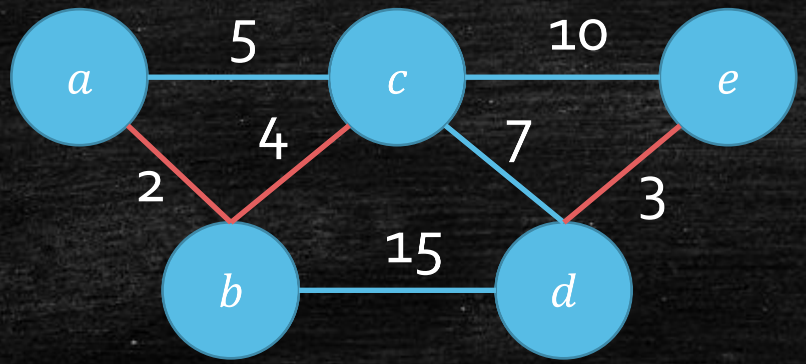


2	3	4	5	7	10	15

Kruskal Algorithm

Kruskal$(G = (V, E))$

- Sort the edge set E to ascending order.
- For each $e \in E$ in ascending order
  - If e do not create a **cycle**, then choose it.

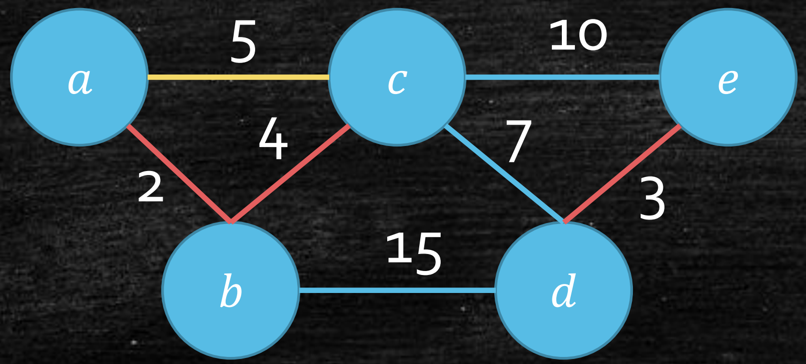


2	3	4	5	7	10	15

Kruskal Algorithm

Kruskal$(\boldsymbol{G} = (\boldsymbol{V}, \boldsymbol{E}))$

- Sort the edge set E to ascending order.
- For each $e \in E$ in ascending order
  - If e do not create a **cycle**, then choose it.

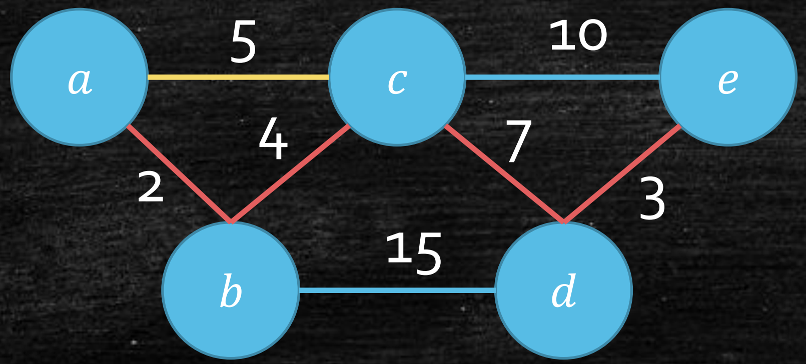


2	3	4	5	7	10	15

Kruskal Algorithm

Kruskal$(G = (V, E))$

- Sort the edge set E to ascending order.
- For each $e \in E$ in ascending order
  - If e do not create a **cycle**, then choose it.

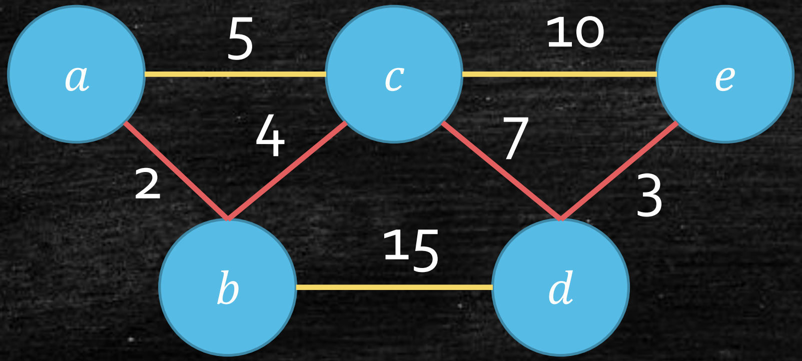


2	3	4	5	7	10	15

Kruskal Algorithm

Kruskal$(G = (V, E))$

- Sort the edge set E to ascending order.
- For each $e \in E$ in ascending order
  - If e do not create a **cycle**, then choose it.

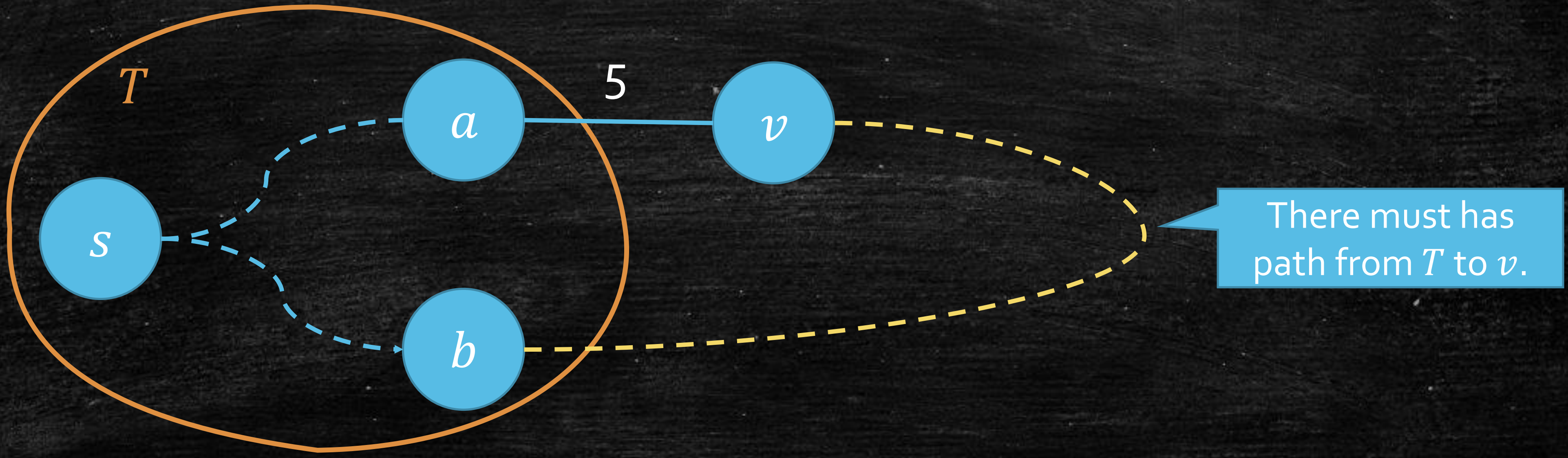


2	3	4	5	7	10	15

Correctness of Prim's Growing idea

- Let's say T^* is the complete MST that contains T, and suppose $(\mathrm{a}, \mathrm{v}) \notin T^*$.

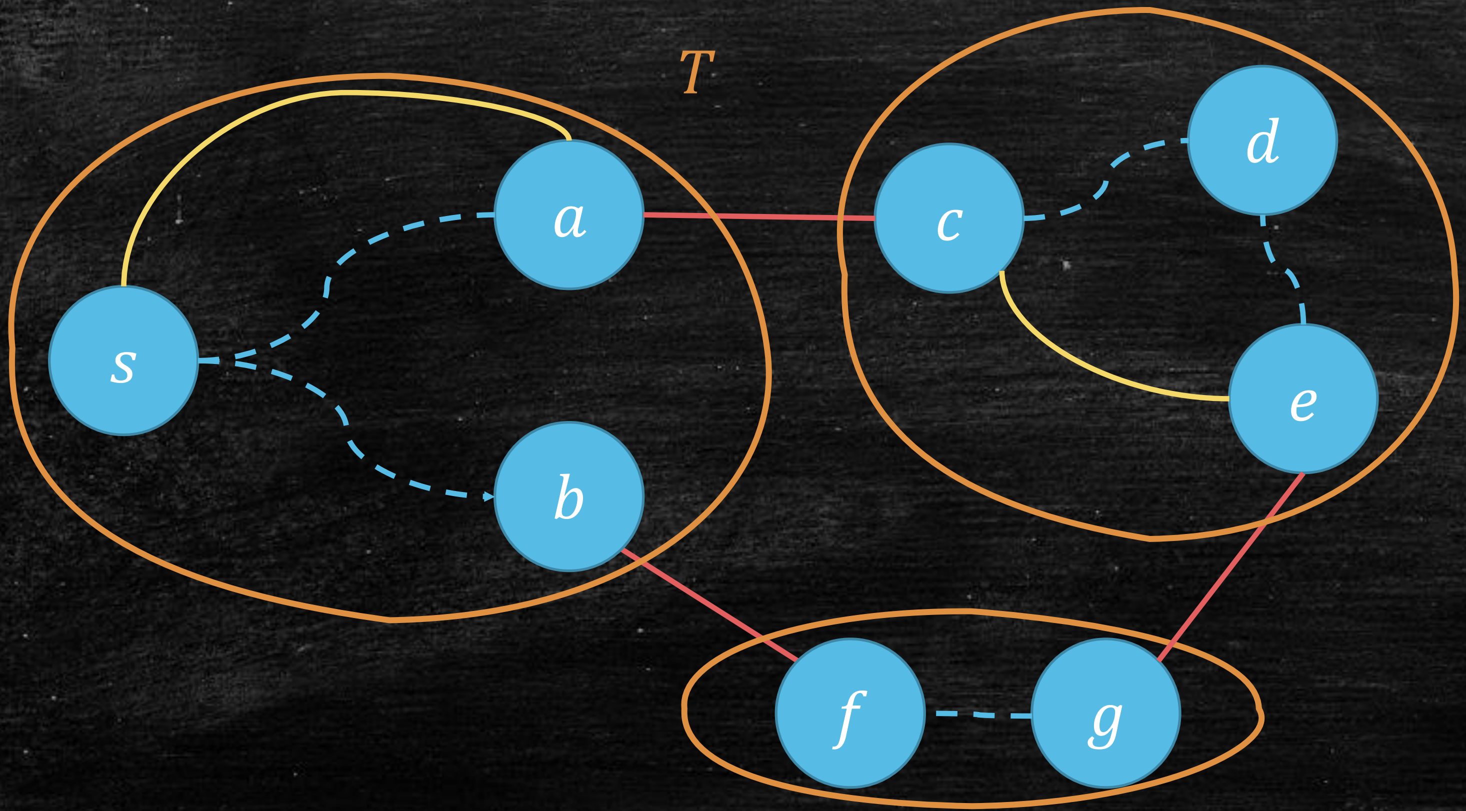
- **Given:** a small P-MST T.
- **Want:** a larger P-MST.
- Can we explore v (smallest cost) into T?



Correctness of Kruskal's Growing idea

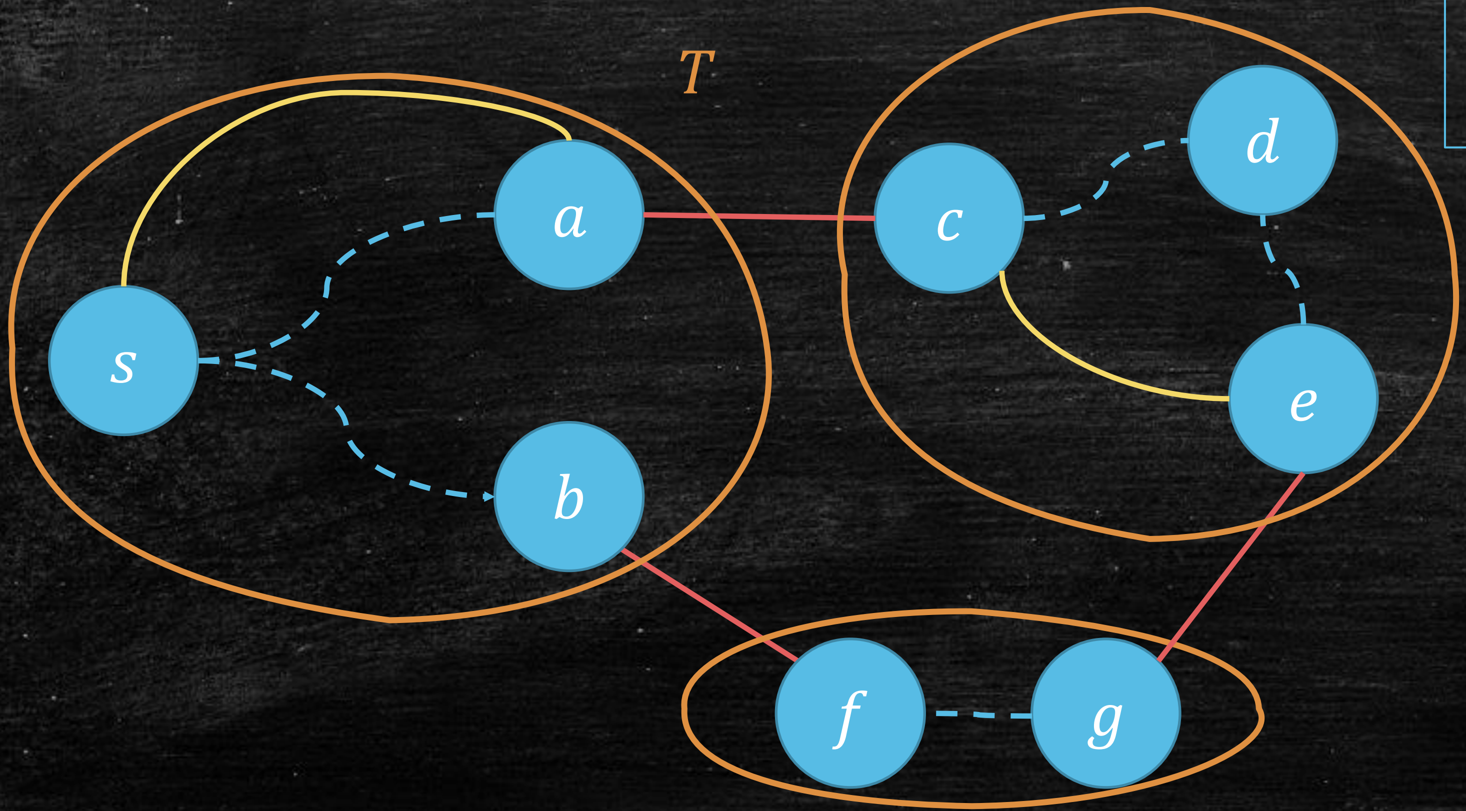
- Let's say T^* is the complete MST that contains T, and suppose $(\mathrm{a}, \mathrm{v}) \notin T^*$.

- **Given:** a small P-MST T.
- **Want:** a larger P-MST.



Correctness of Kruskal's Growing idea

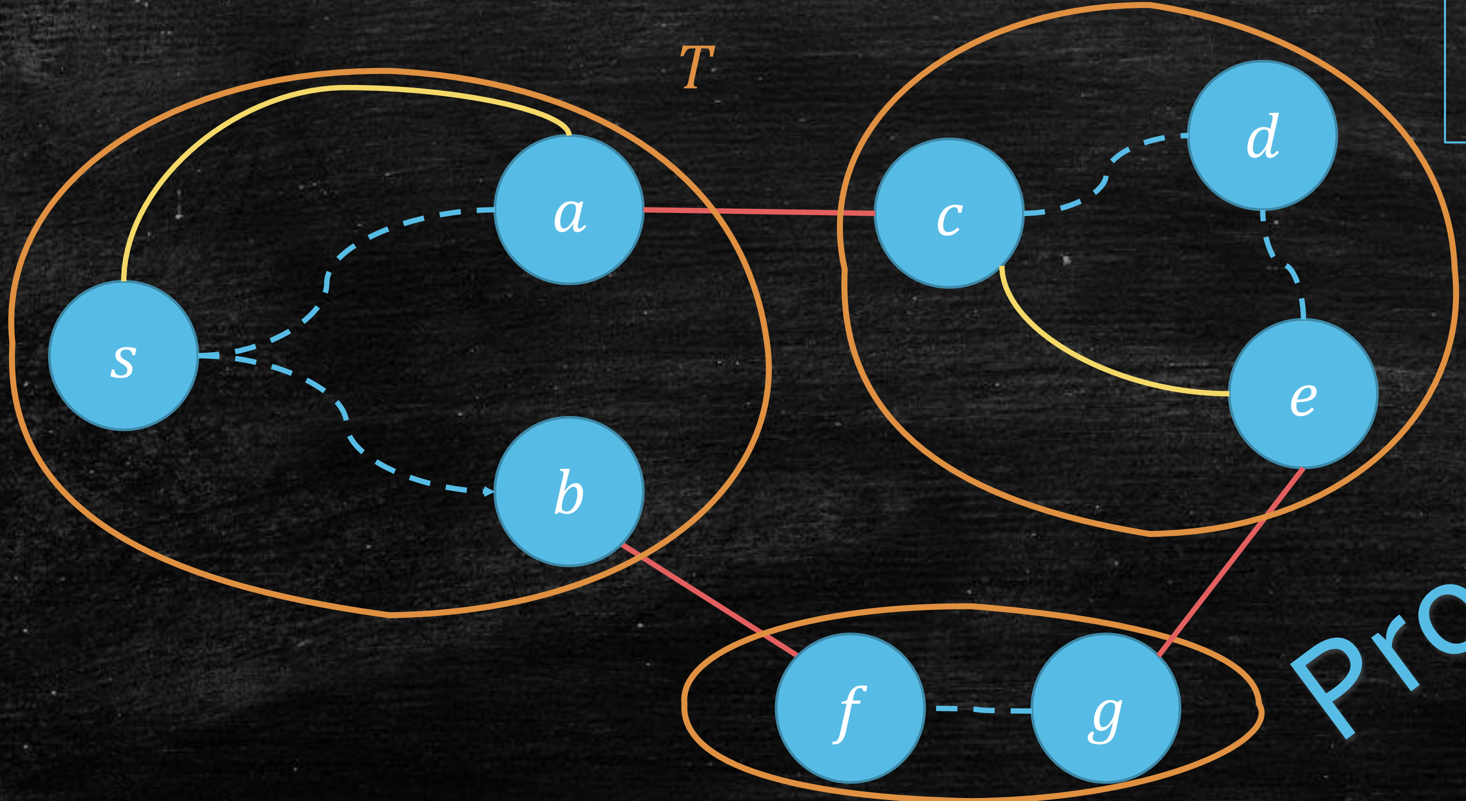
- Let's say T^* is the complete MST that contains T, and suppose $(\mathrm{a}, \mathrm{v}) \notin T^*$.



- **Given:** a small P-MST T.
- **Want:** a larger P-MST.
- Add the smallest red edge get a larger P-MST.

Correctness of Kruskal's Growing idea

- Let's say T^* is the complete MST that contains T, and suppose $(\mathrm{a}, \mathrm{v}) \notin T^*$.



- **Given:** a small P-MST T.
- **Want:** a larger P-MST.
- Add the smallest red edge get a larger P-MST

Prove by yourself!

Running Time

Kruskal$(\boldsymbol{G} = (\boldsymbol{V}, \boldsymbol{E}))$

- Sort the edge set E to ascending order.
- For each $e \in E$ in ascending order
  - If e do not create a **cycle**, then choose it.

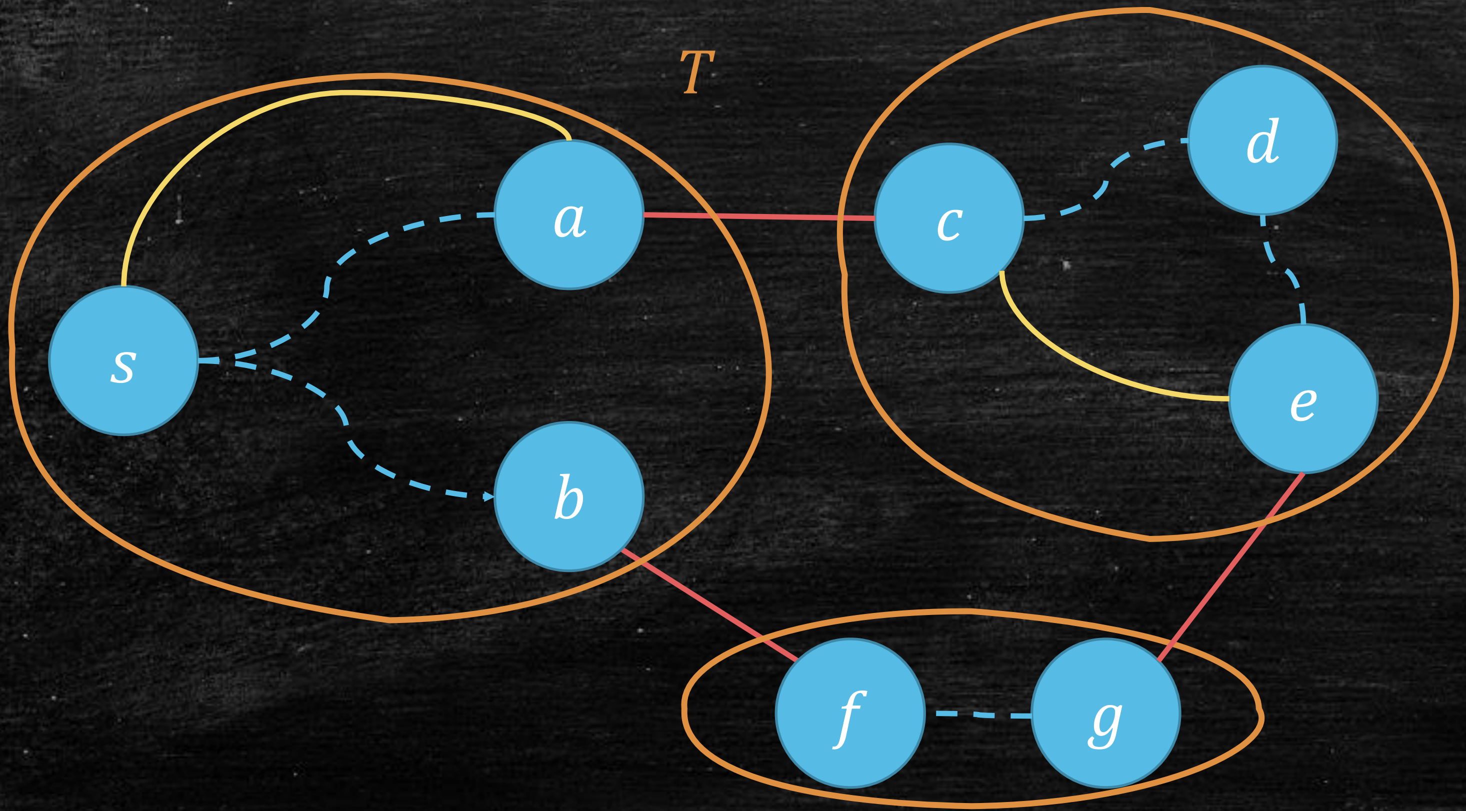
- $O(|E|\log|E|)$ for sorting.
- $|E|$ round: check cycle!

Recall DFS

- When an edge is a back edge (to marked vertices),
- It forms a cycle.

During Kruskal

- When an edge connect the same group vertices,
- It forms a cycle.



Kruskal (refine)

Kruskal$(\boldsymbol{G} = (\boldsymbol{V}, \boldsymbol{E}))$

- Sort the edge set E to ascending order.
- For each $(u, v) \in E$ in ascending order
  - If $group(u) != group(v)$
    - Choose (u, v).
    - $union(group(u), group(v))$

Running Time: Kruskal (refine)

Kruskal$(\boldsymbol{G = (V, E)})$

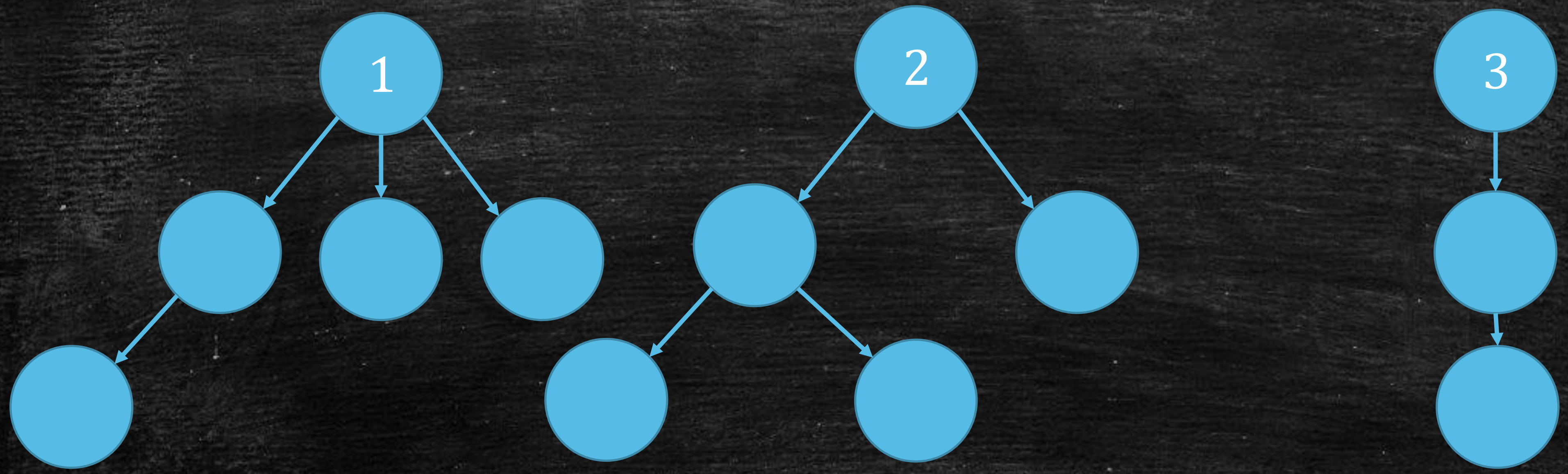
- Sort the edge set E to ascending order.
- For each $(u, v) \in E$ in ascending order
  - If $group(u)! = group(v)$
    - Choose (u, v).
    - $union(group(u), group(v))$

- $O(|E| \log|E|)$ for sorting.
- $2|E|$ round: check group
- $|V|$ round: union group

Union-Find Set

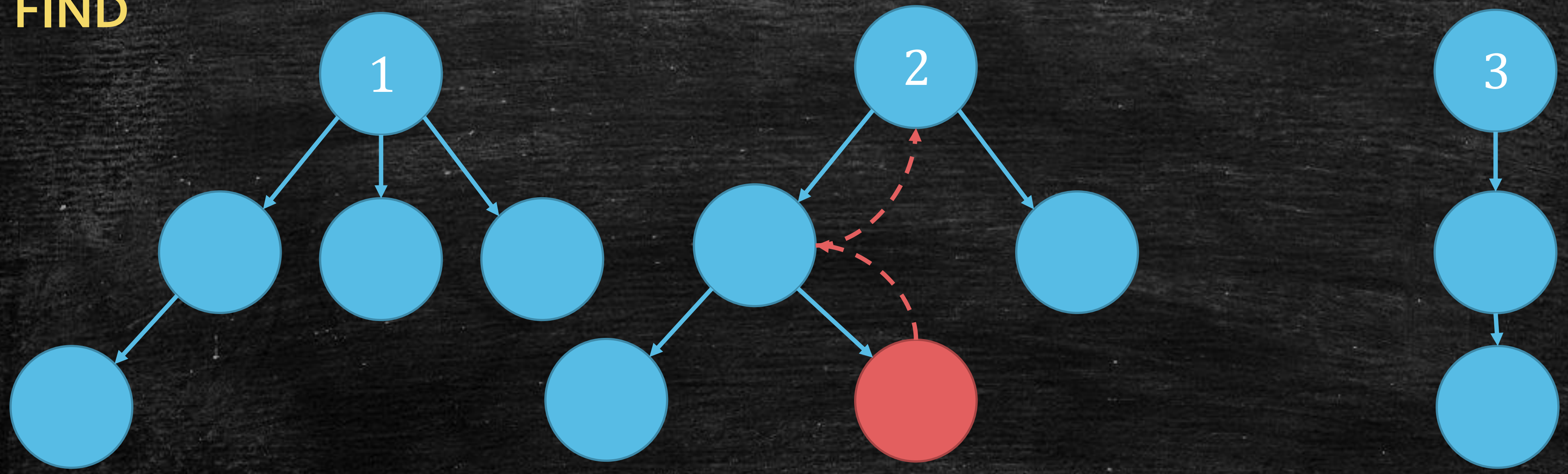
- Recall Union-Find Set
  - Find: $O(\log n)$
  - Union: $O(1)$
- Kruskal
  - $O(|E|\log|E|)$ for sorting.
  - $2|E|$ round: check group
  - $|V|$ round: union group
  - $O(|E|\log|E|) = O(|E|\log|V|)$

Review Union-Find Set



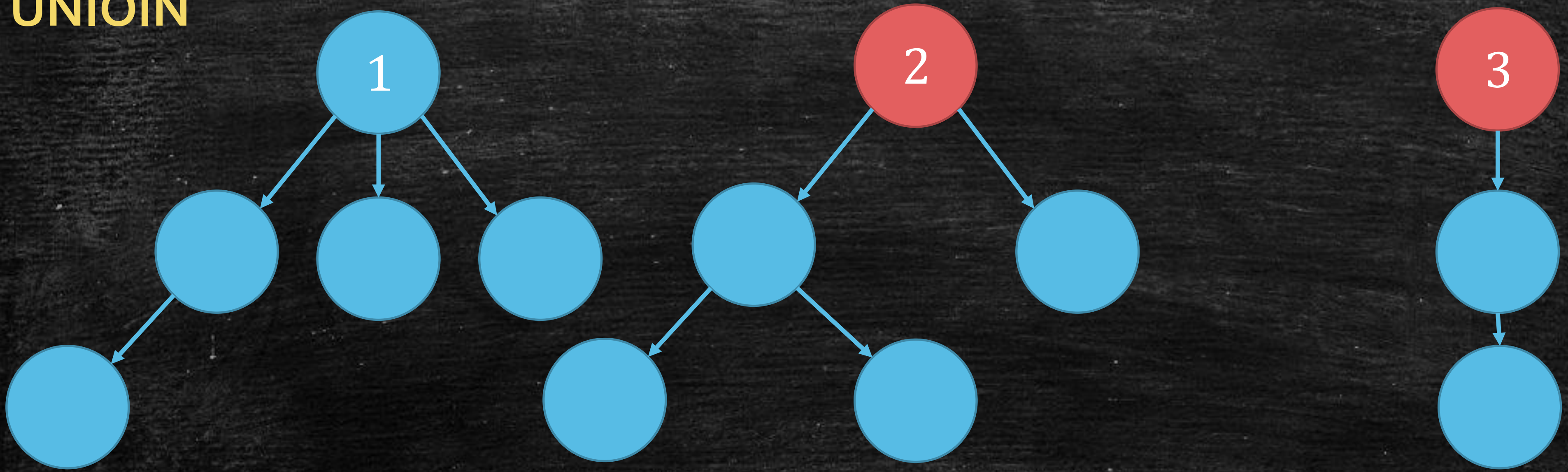
Review Union-Find Set

FIND



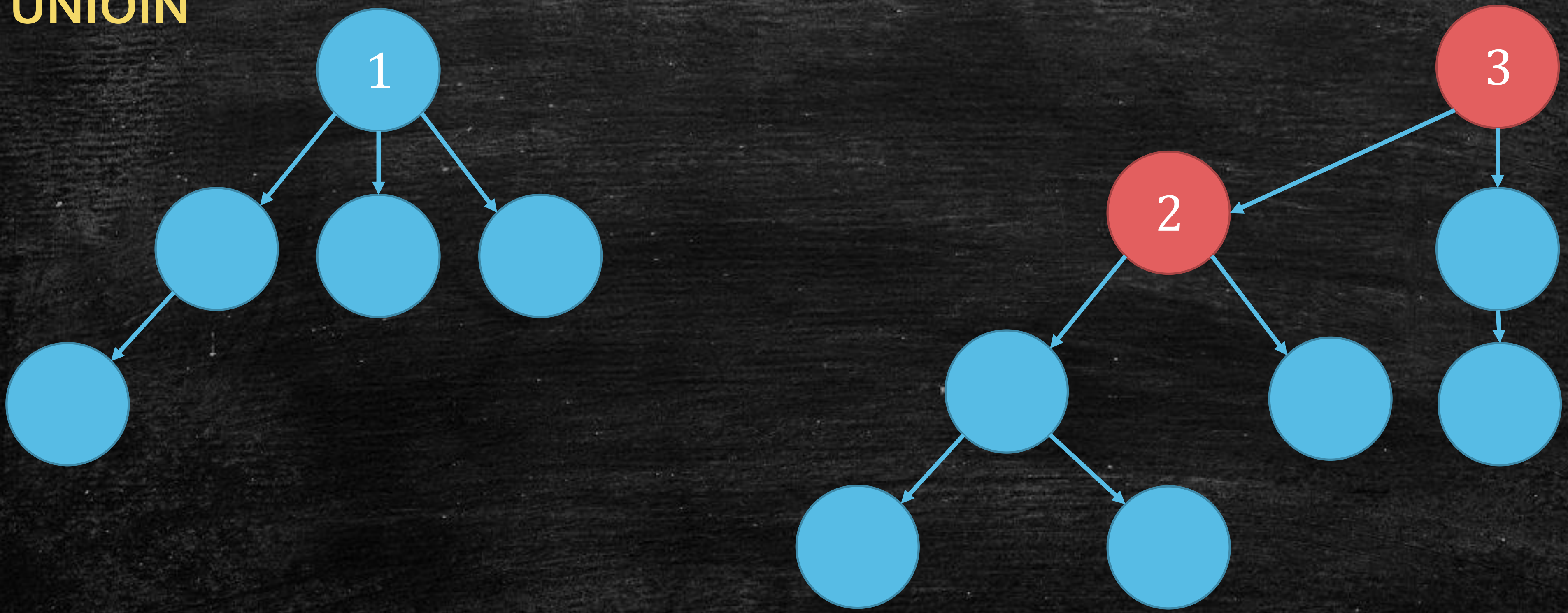
Review Union-Find Set

UNIOIN



Review Union-Find Set

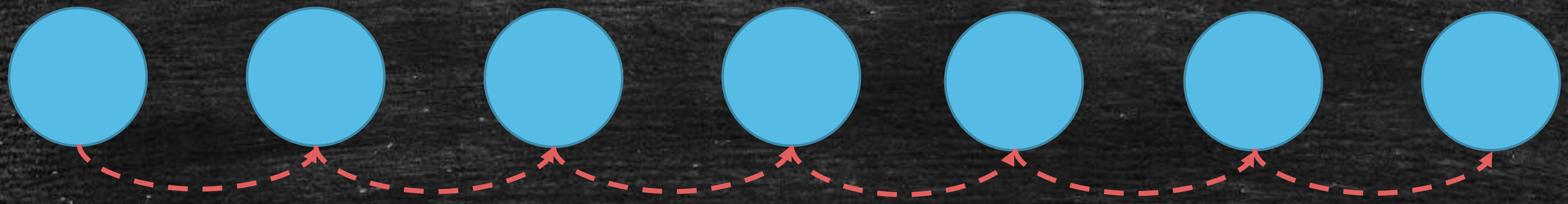
UNIOIN



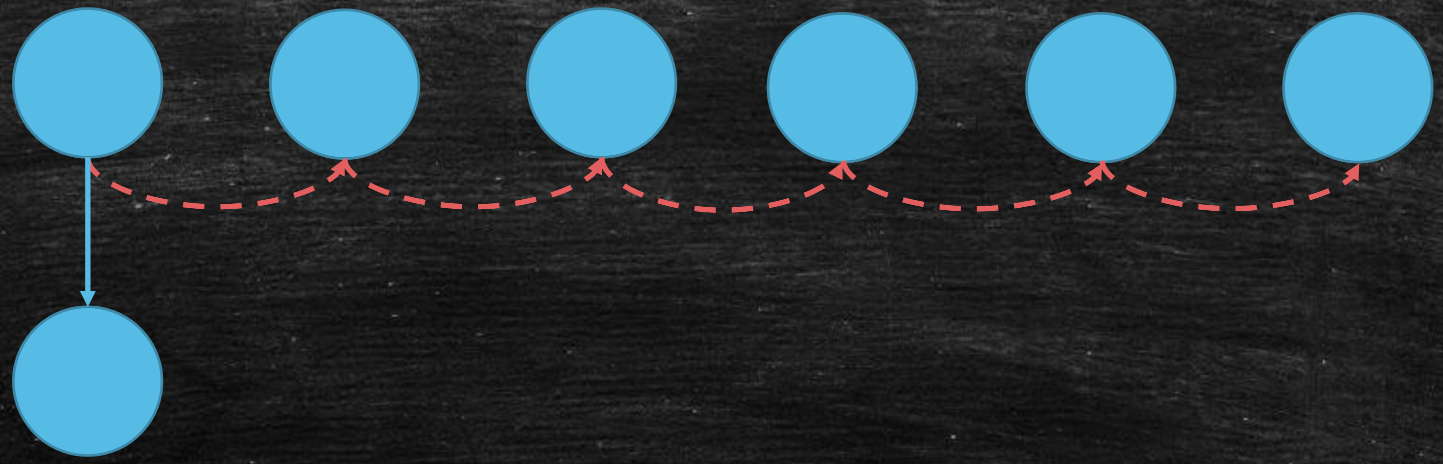
Time Complexity

- Find
  - $O(\max\{Tree\ height\})$
- Union
  - $O(1)$

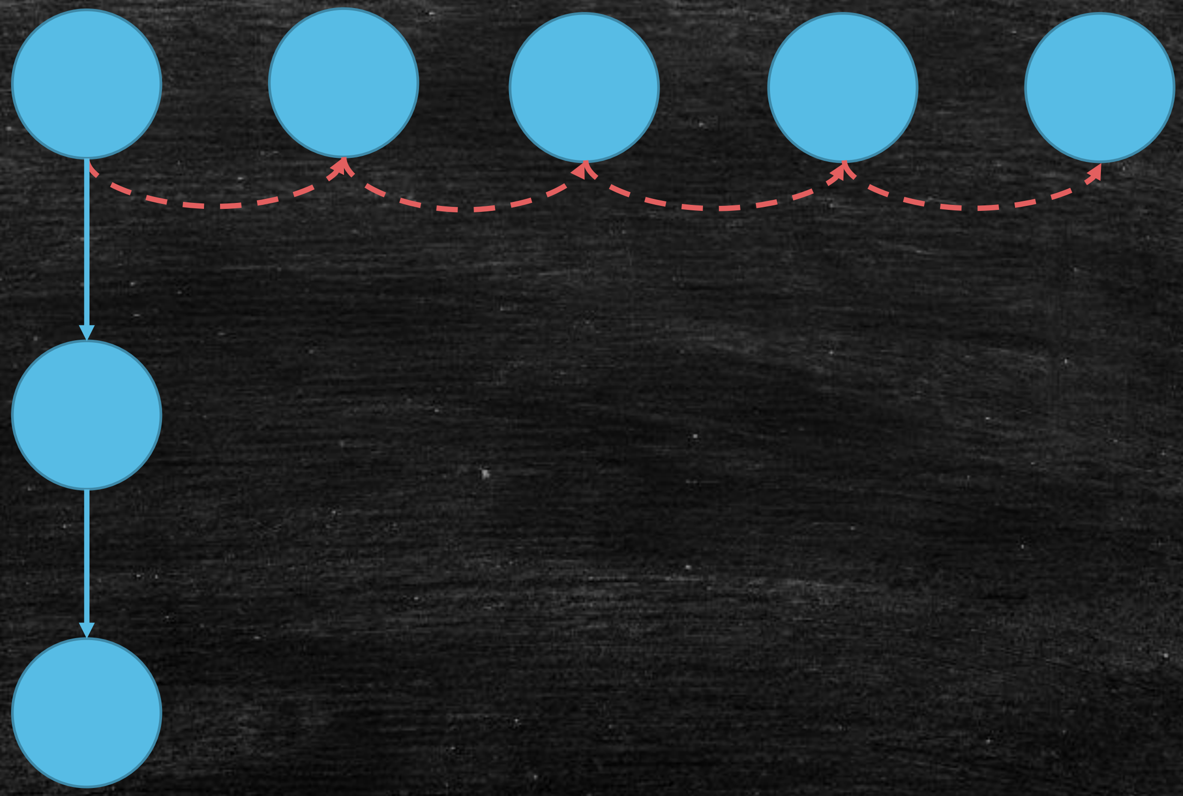
A bad Case



A bad Case

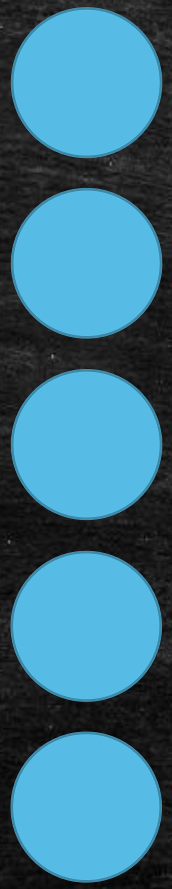


A bad Case



A bad Case

O(n) tree height

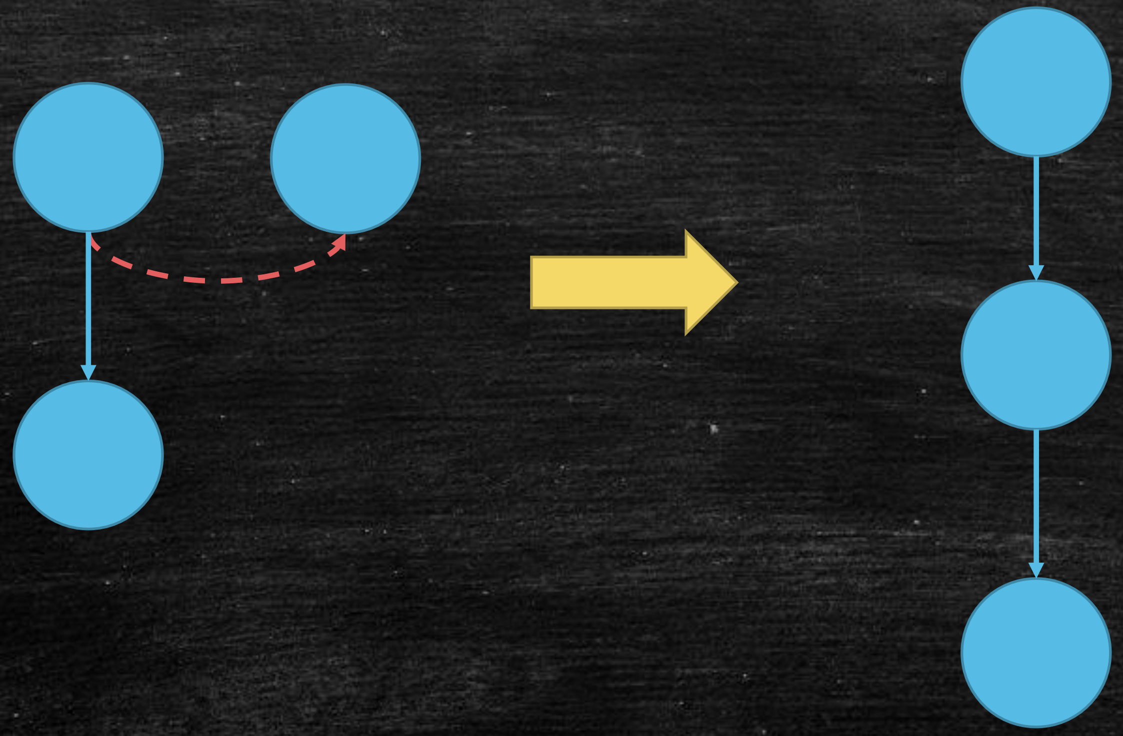


How to improve

- Find
  - $O(\max\{Tree\ height\})$
  - $O(n)$!
- Union
  - $O(1)$
- To Do
  - Reduce Tree Height

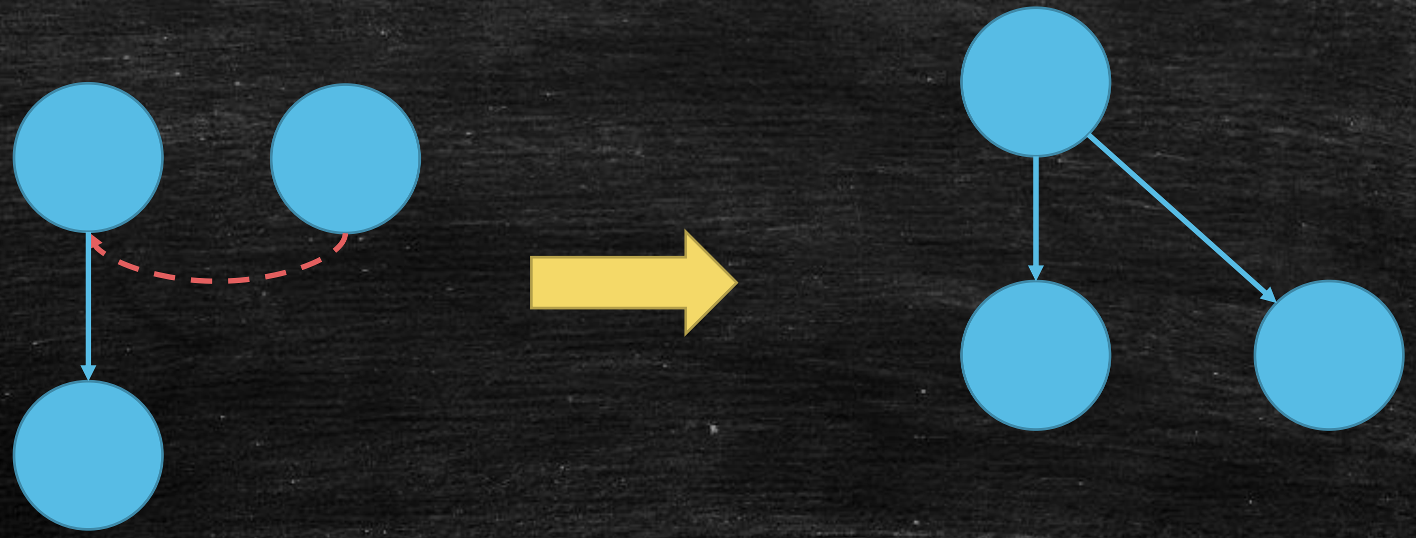
Intuition

BAD



Intuition

GOOD



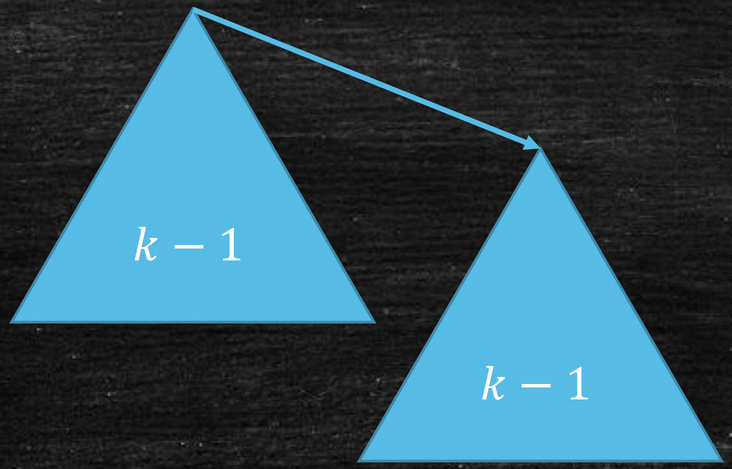
We should merge to a same root!
We should merge short tree to high tree!

Implement

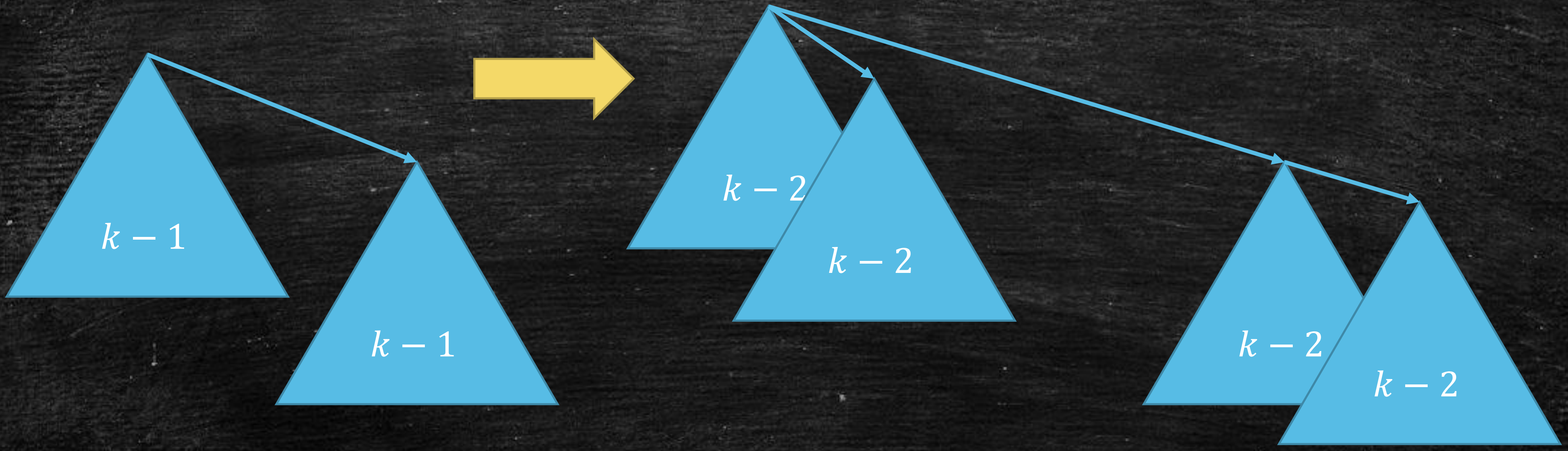
- Record Tree's height (rank).
- $rank[v]$: the rank of tree rooted at v.
- Union: u and v.
  - Rooted at u: if $rank[u] \geq rank[v]$
  - Rooted at v: if $rank[u] < rank[v]$
  - Update $rank[u]++$: if $rank[u] = rank[v]$
- We make it hard to build a large rank tree!

How to build a rank k tree?

How to build a rank k tree?



How to build a rank k tree?



We should at least use **2^k nodes!**

Max tree height

- Build a rank k tree: We should at least use **2^k nodes!**
- What is the max tree height (rank)?
- $\boldsymbol{O(\log n)}$
- Find
  - $O(\max\{Tree\ height\})$
  - $\boldsymbol{O(\log n)}$!
- Union (rank based)
  - $O(1)$

Union-Find Set

- Recall Union-Find Set
  - Find: $O(\log n)$
  - Union: $O(1)$
- Kruskal
  - $O(|E|\log|E|)$ for sorting.
  - $2|E|$ round: check group
  - $|V|$ round: union group
  - $O(|E|\log|E|) = O(|E|\log|V|)$

Can we do better?

- Karger-Klein Tarjan (1995)
  - $O(m)$ randomized algorithm.
- Chazelle (2000)
  - $O(m \cdot \alpha(n))$ deterministic algorithm.
  - $\alpha(n)$ is the inverse Ackermann function $\alpha(9876!) \leq 5$.
  - Ackermann function: $A(4,4) \approx 2^{2^{2^{2^{16}}}}$.
- Pettie-Ramachandran (2002)
  - O(optimal #comparison to determine solution)
  - We know #comparison $= \Omega(n) = O(m \cdot \alpha(n))$

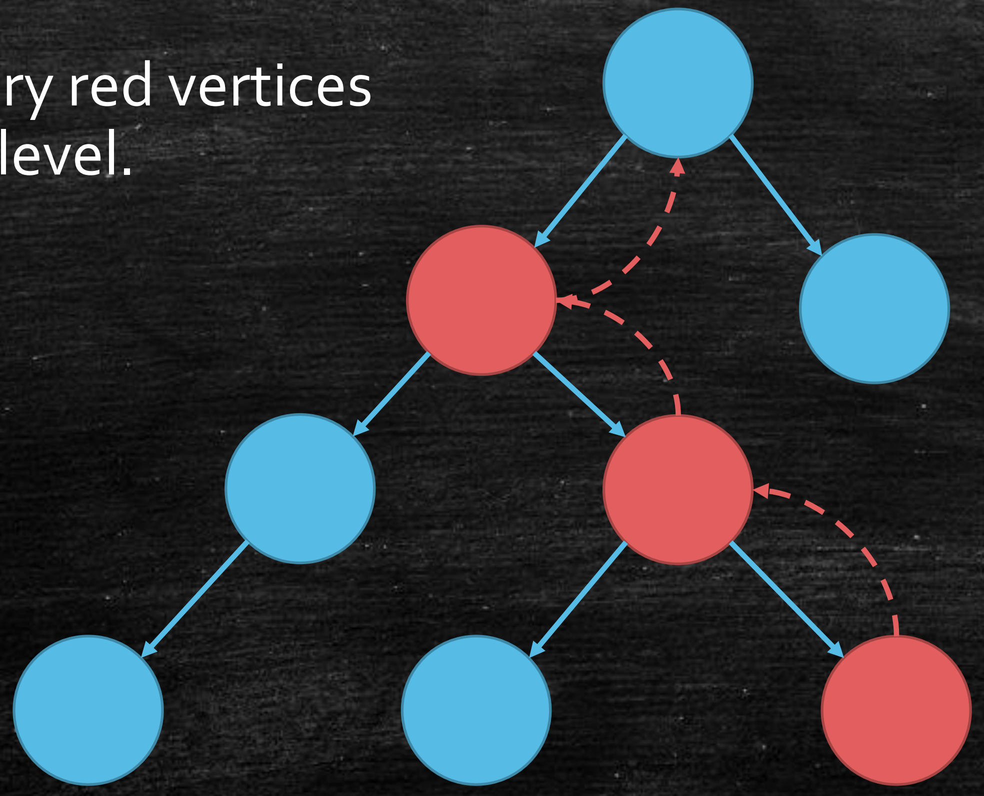
Can we do better for Union-Find Set?

- Have you heard **Path Compression**?

Path Compression

FIND

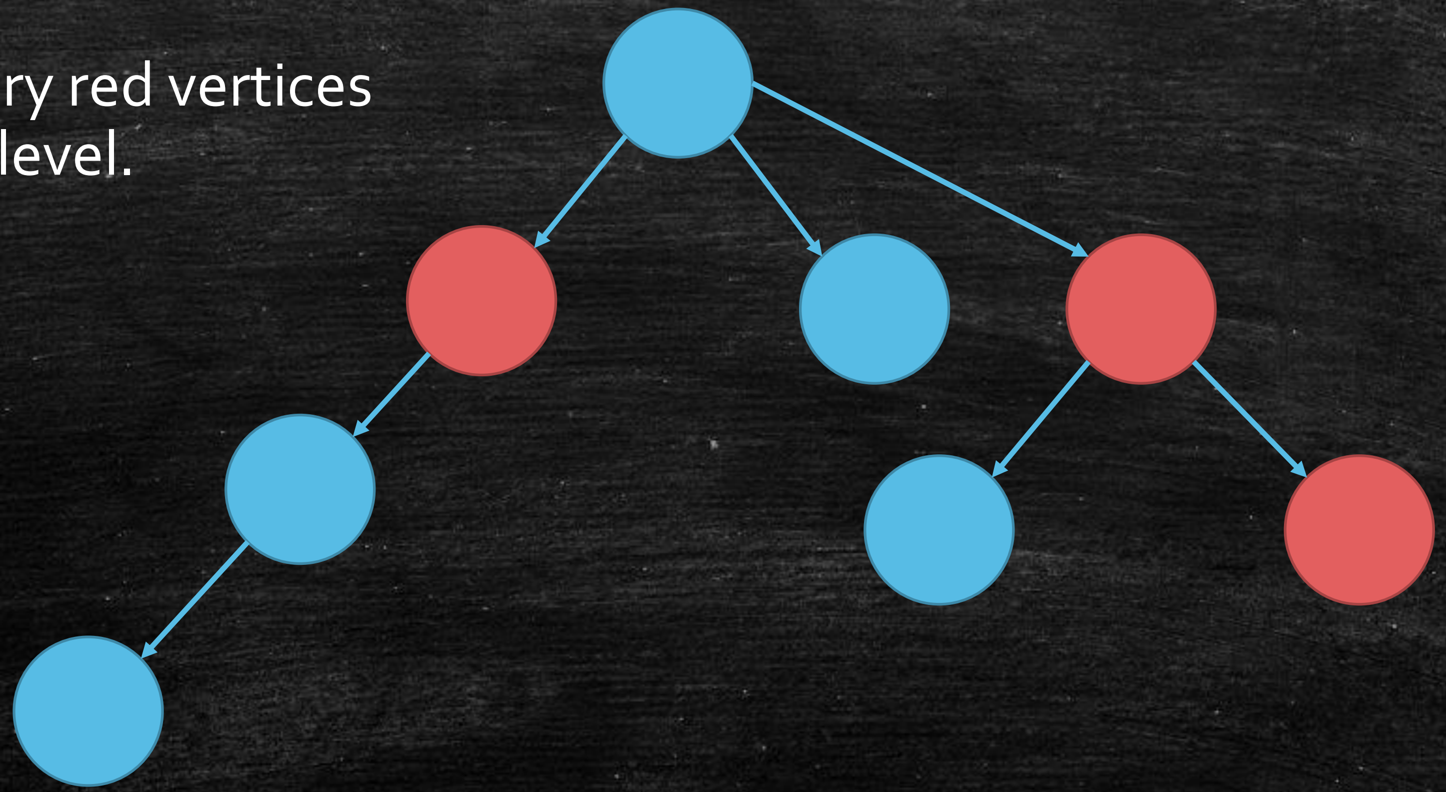
We put every red vertices
to the first level.



Path Compression

FIND

We put every red vertices
to the first level.

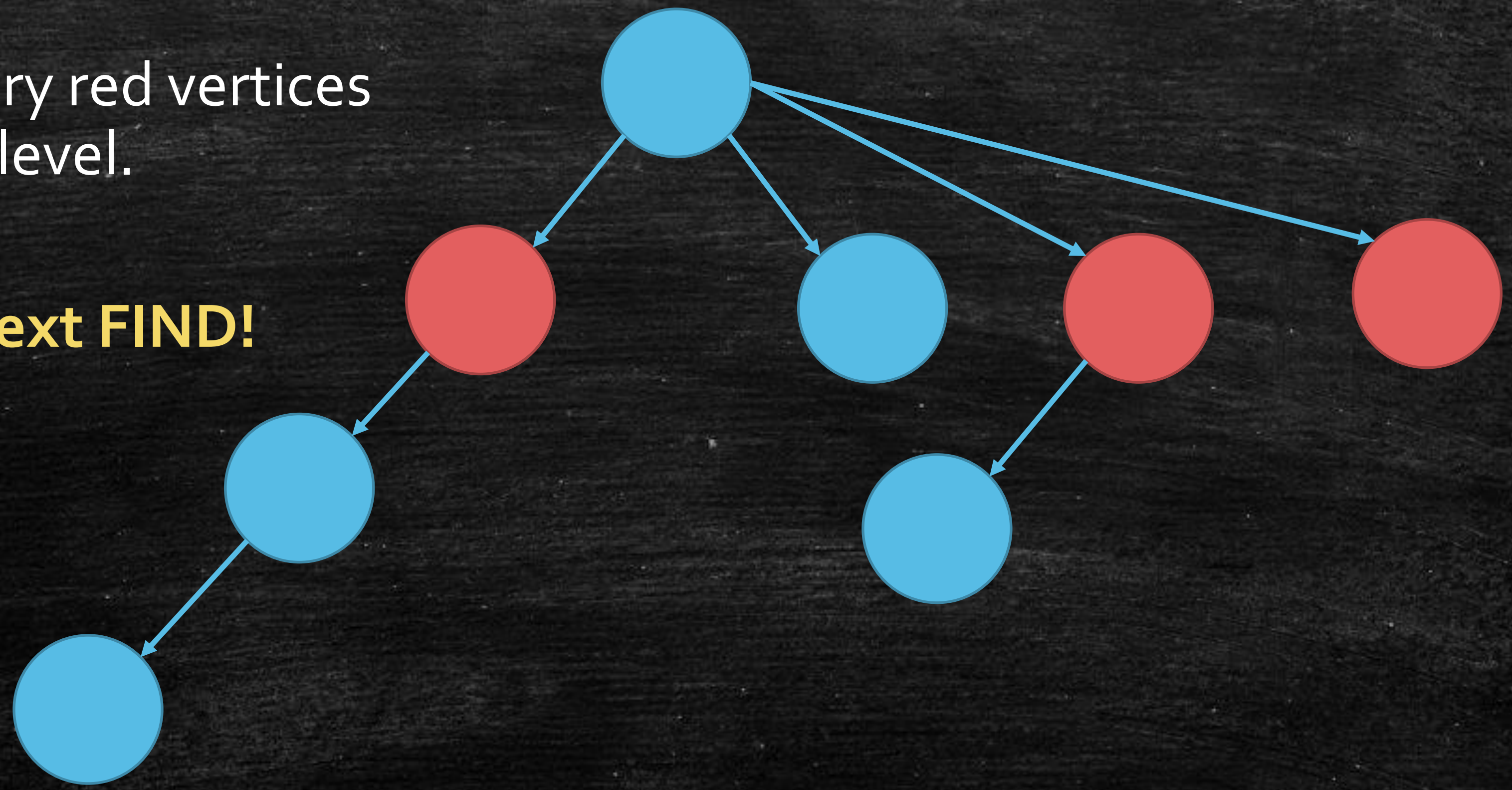


Path Compression

FIND

We put every red vertices to the first level.

Good for next FIND!



You know what the next step!

Amortized Analysis

Time Complexity

- Find (Path Compression)
  - $O(\log^* n)$ [Hopcroft & Ullman 1973]
  - $\log^*(2^{2^{2^{2^2}}}) = \log^*(2^{65536}) = 5$
  - $O(\alpha(n))$ [Tarjan 1975]
  - $\alpha(n)$ is the inverse Ackermann function $\alpha(9876!) = 5$.
- Union (rank based)
  - $O(1)$

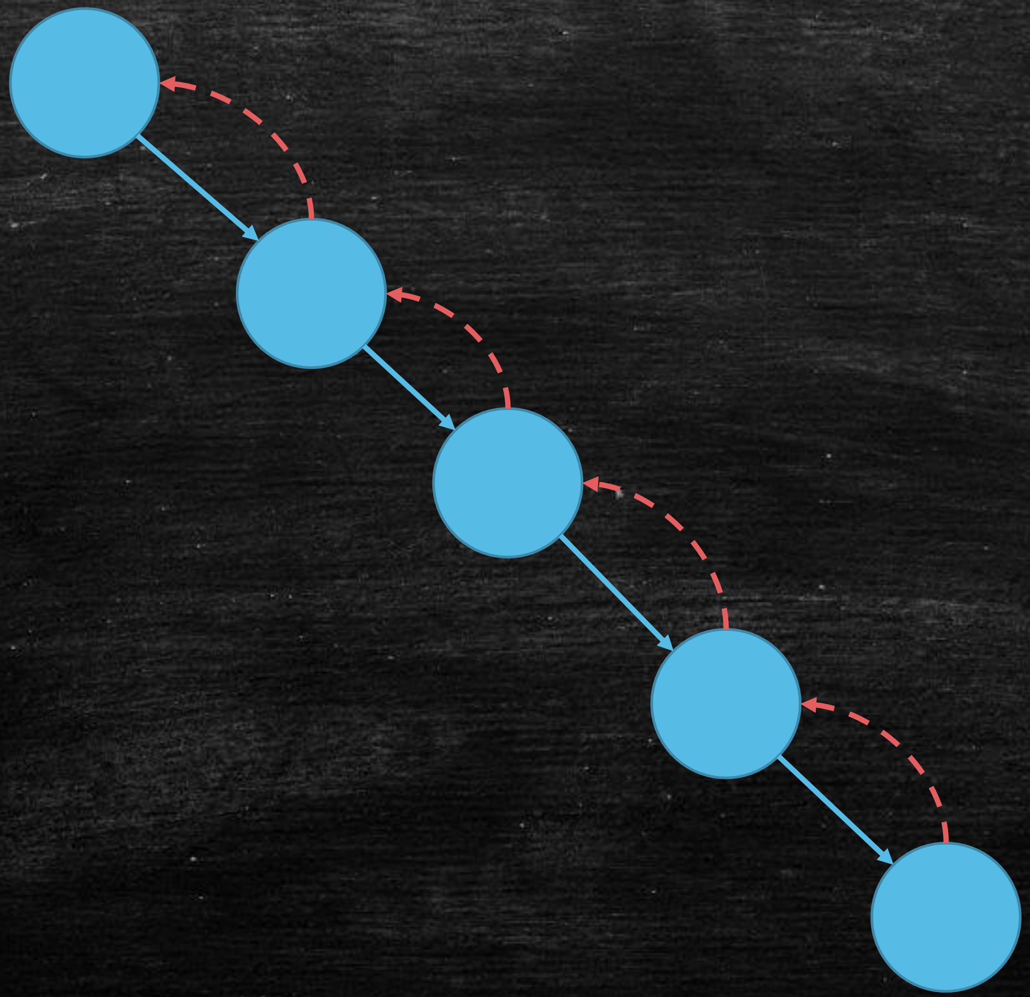
Rank Based Union + Find with Path Compression

- It is still an amortized analysis
- We prove:
  - m find operation, totally cost $O(m \log^* n)$.

Analysis

FIND

Cost=number of **red edges.**

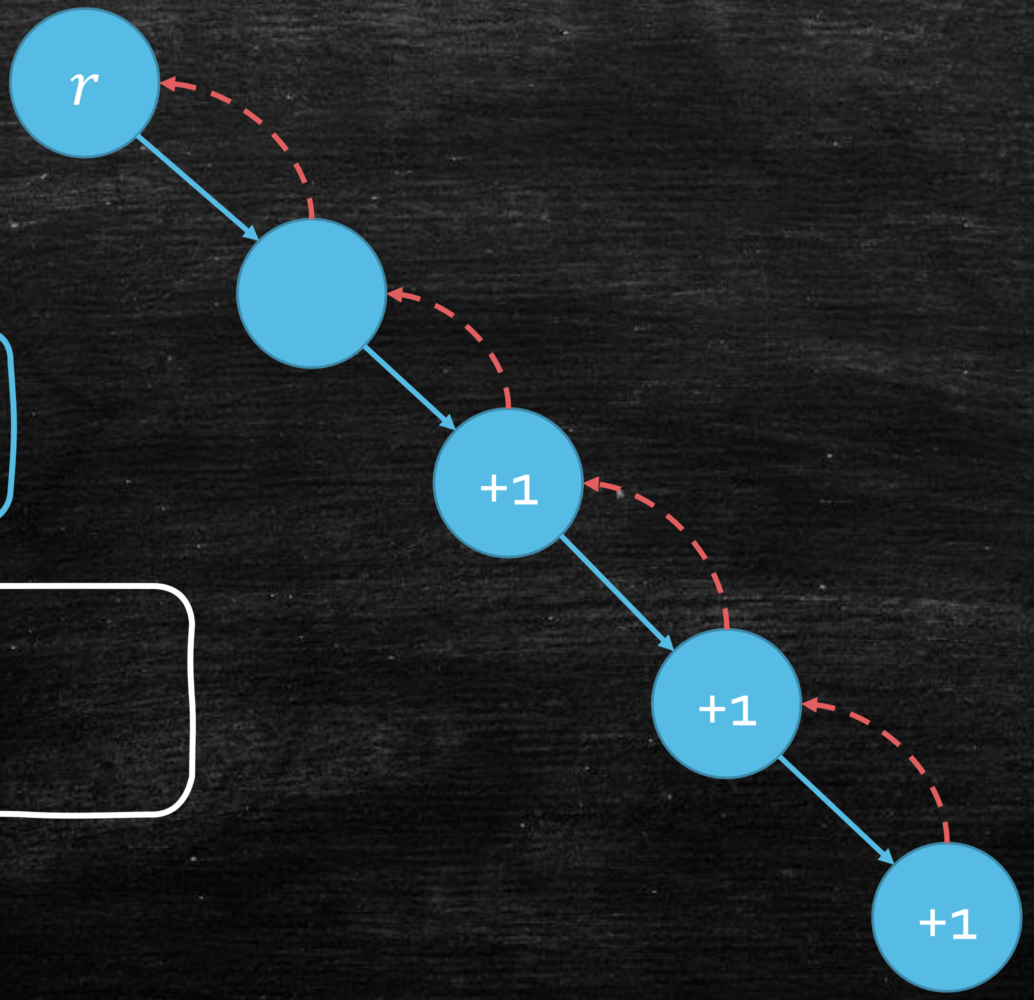


Key Idea: Charge Cost to Vertices

FIND

Cost=number of **red edges.**

Red edge charge cost to child vertex.



Total Cost of m FIND
- $O(m)(to\ root)$
- Total Charging

How much each vertex will be charge?

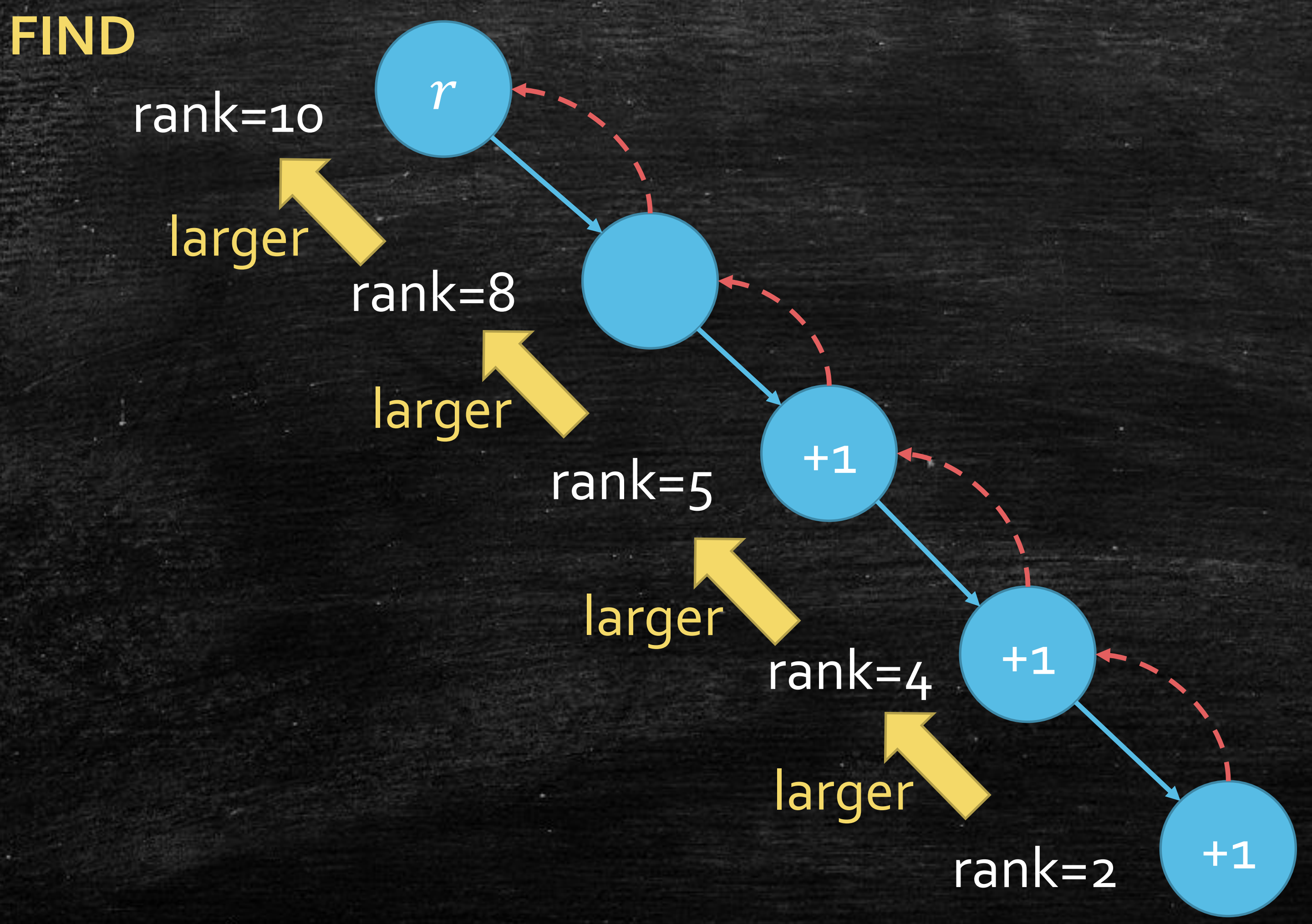
What is rank now?

- No path compression
  - $rank[v]$: the max height of the subtree rooted at v.
- With path compression
  - The max height of the subtree rooted at v can **be changed**.
  - But we still have a rank for each v. So, we call it rank but not height :)
- Recall $rank[v]$
  - Originally $rank[v] = 1$.
  - When u is merged to v, and $rank[v] > rank[u]$, nothing changed.
  - When u is merged to v, and $rank[v] = rank[u]$, $rank[v] + +$
  - When v is merged to any other vertices, $rank[v]$ will not be changed.

Property of rank

- Parent’s rank is strictly larger than the child.
- Because
  - We only merge small rank to large rank。
  - If we merge two same root, the new root’s rank will +1.
  - Even if we do path compression, v’s parent become stronger (rank larger).

Key Fact in FIND



Group Vertices

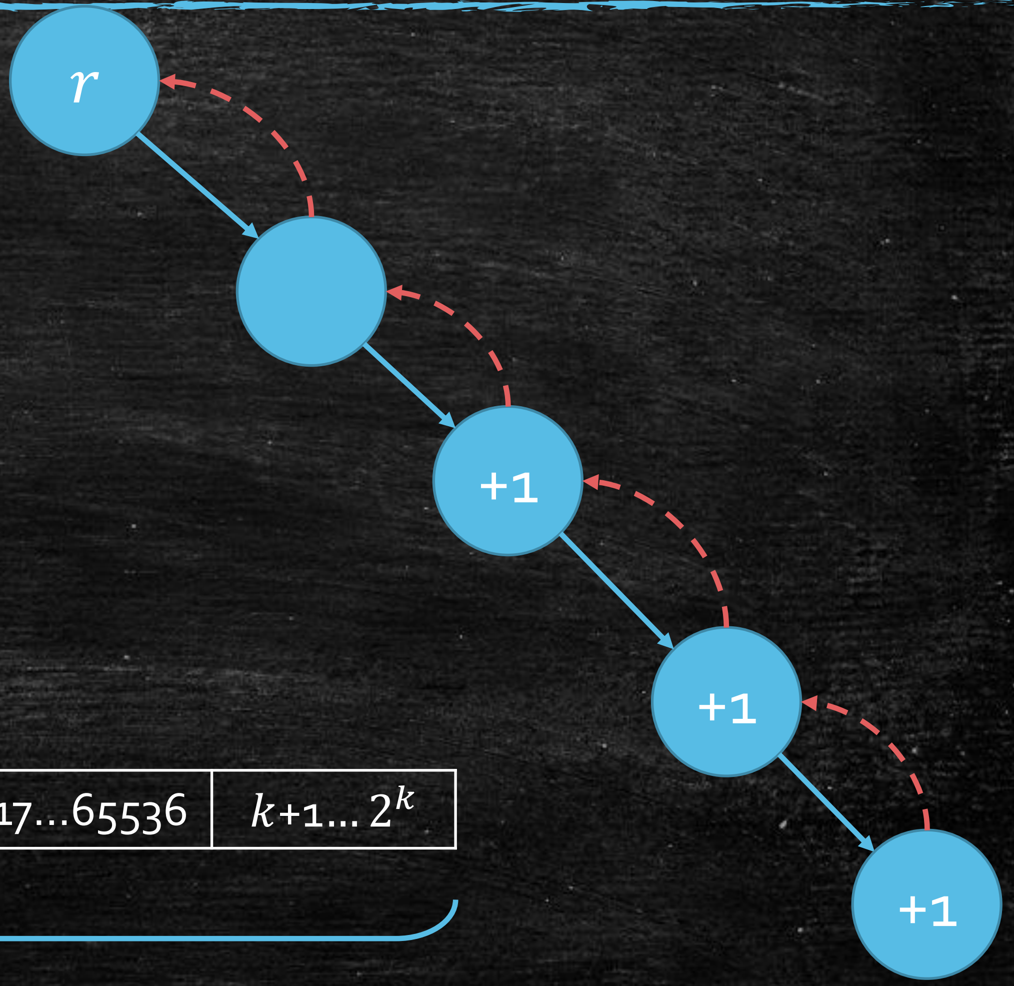
- Group vertices by rank
  - Group 1: $k_1 = 0$
  - Group 2: $k_2 = 1$
  - Group i: $k_i = 2^{k_{i-1}}$
- Property: group $[k+1, 2^k]$ at most have $n/2^k$ vertices.
- Recall: build rank k need 2^k vertices. (Think why it is still right when we use Path Compression.)

1…1	2…2	3…4	5…16	17…65536	k+1… 2^k

$\log^* n$

Different Type Charging

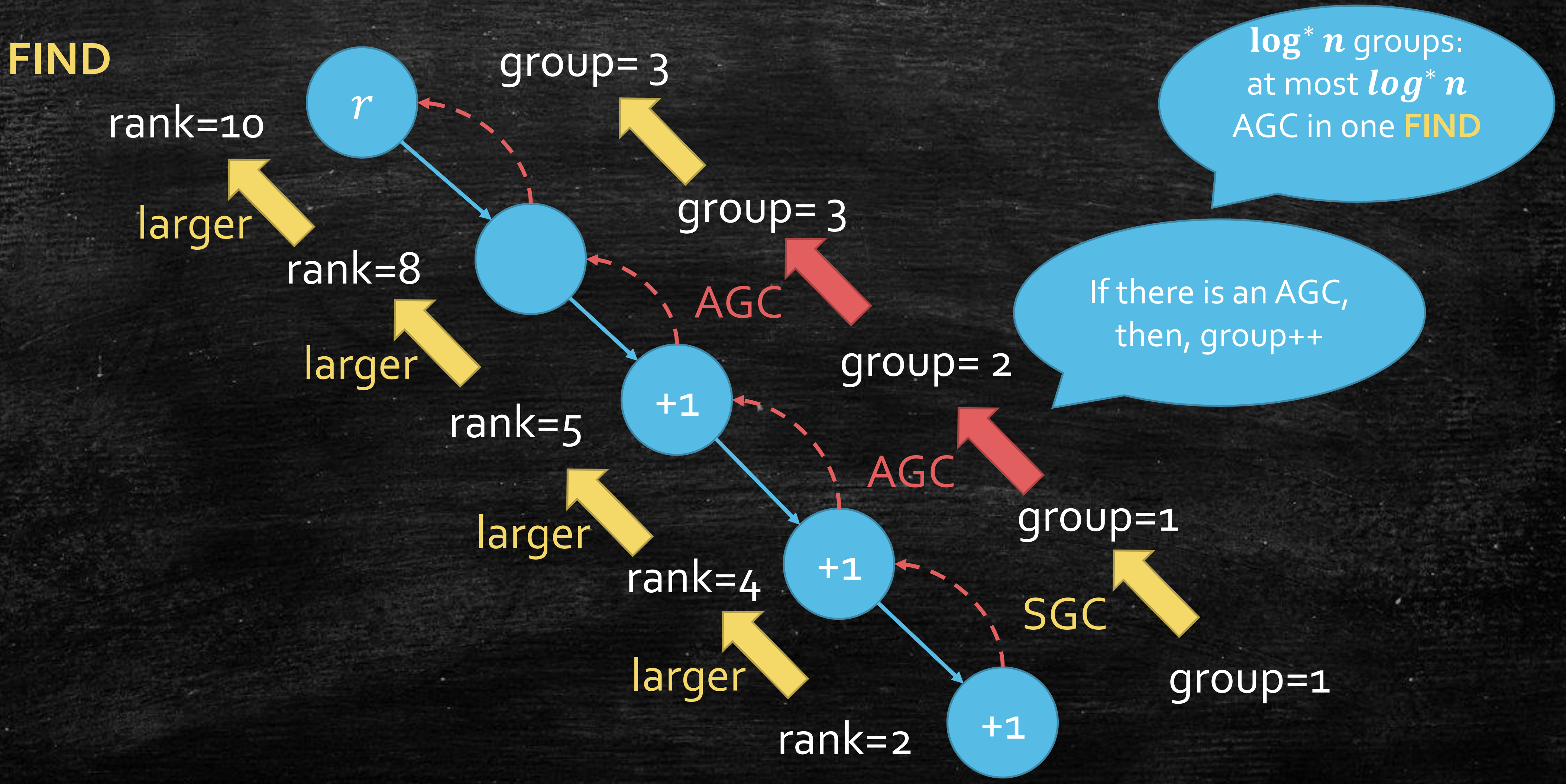
- Two kind of charging
  - Same Group Charging (SGC)
  - Across Group Charing (AGC)
- AGC for all vertices: $m \cdot \log^* n$
  - m FIND
  - Each FIND at most $\log^* n$ AGC



1…1	2…2	3…4	5…16	17…65536	k+1… 2^k

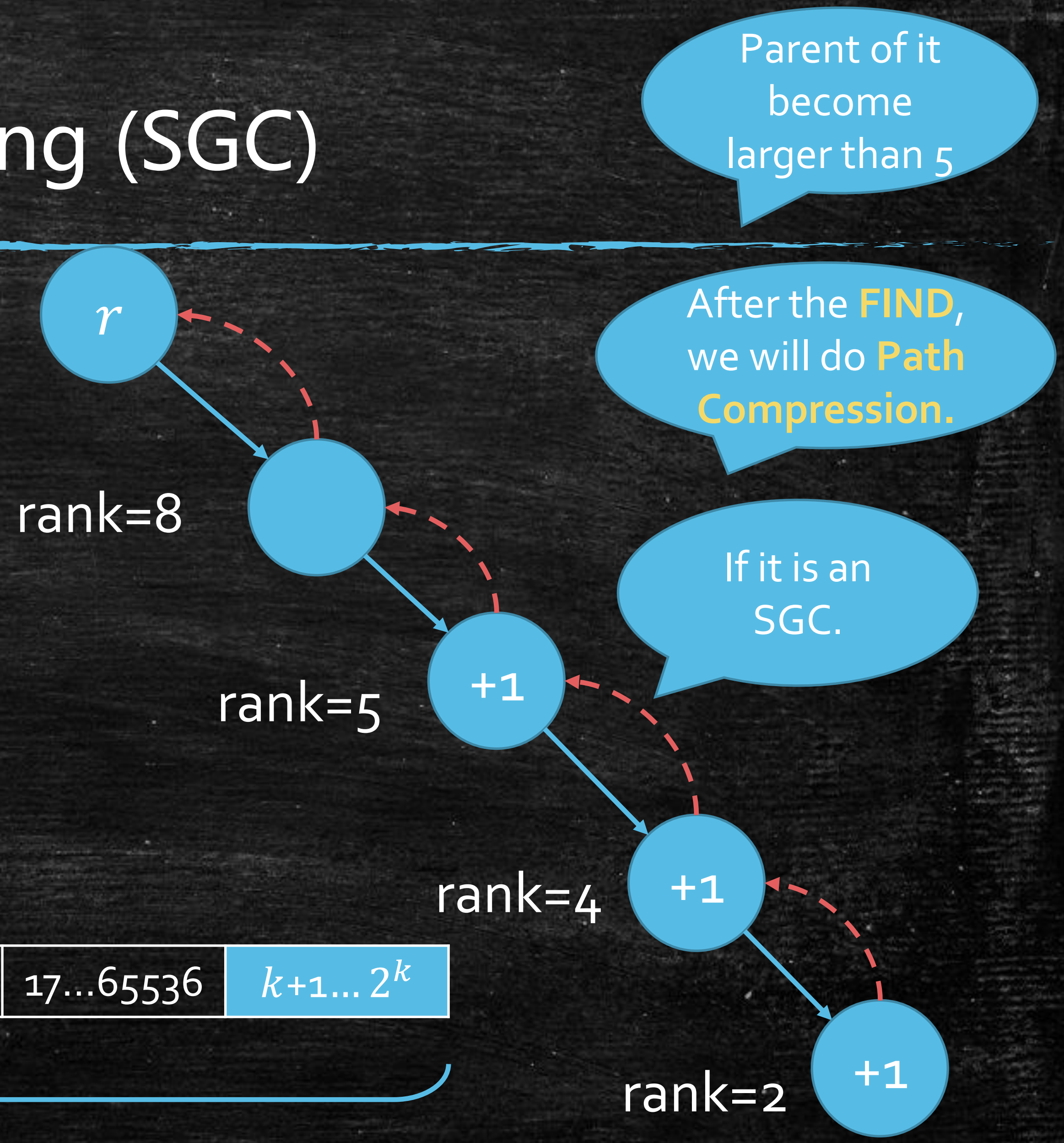
$\log^* n$

Cost of Across Group Charing (AGC)



Bound Same Group Charing (SGC)

- SGC for v
- After m FIND
  - v can be SGC many times
  - $u_1, u_2, \ldots$ is each SGC's parent (in the same group).
  - Path Compression: $rank[u_3] > rank[u_2] > rank[u_1] > rank[v]$.
  - At most $2^{\mathrm{k}} - (\mathrm{k}+1) < 2^{\mathrm{k}}$ SGC for v.



1…1	2…2	3…4	5…16	17…65536	k+1… 2^k

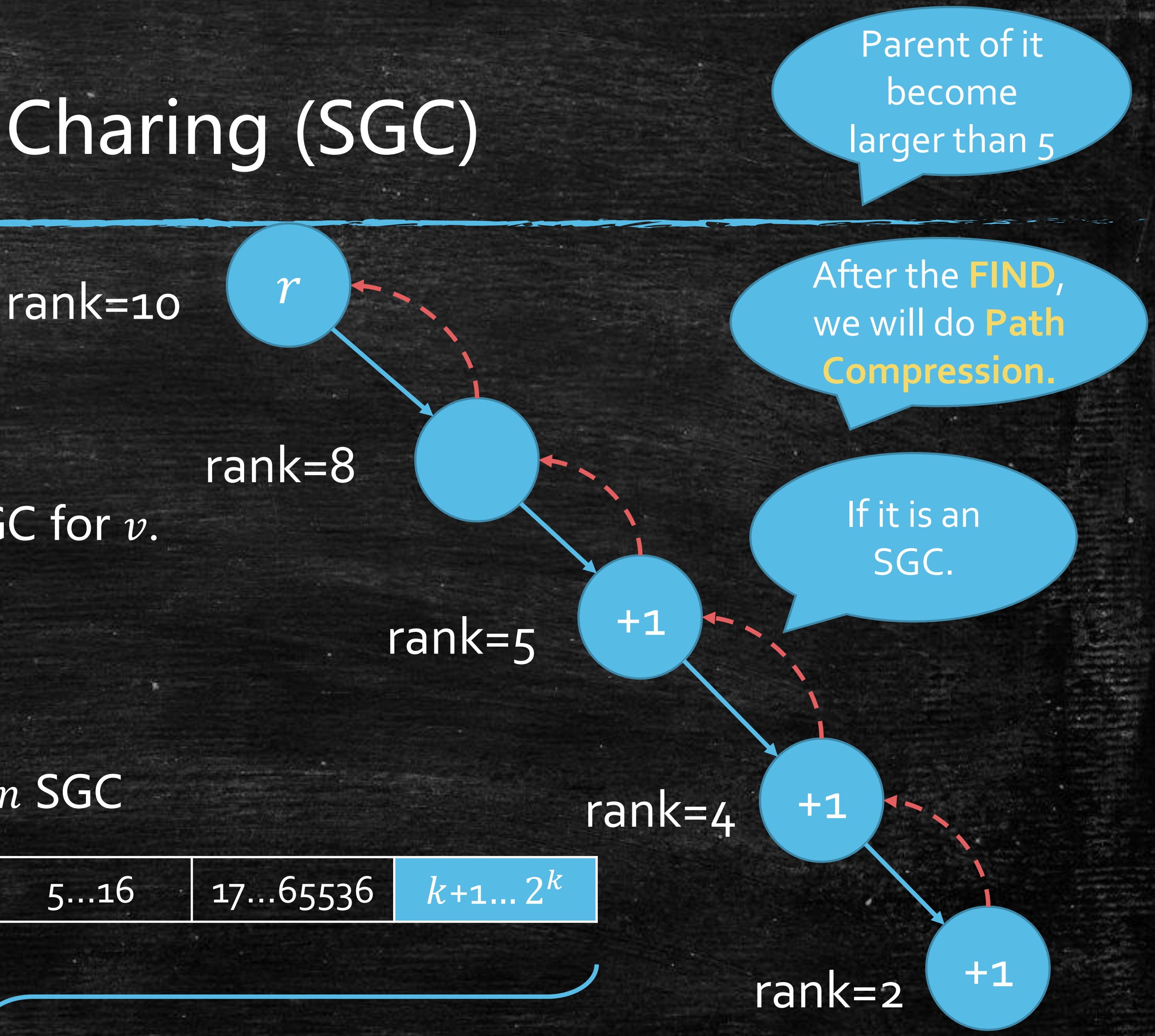
$\log^* n$

Bound Same Group Charing (SGC)

- SGC for v
- After m FIND
  - v can be SGC many times
  - At most $2^{\mathrm{k}} - (\mathrm{k}+1) < 2^{\mathrm{k}}$ SGC for v.
  - In a group $[k+1 \ldots 2^k]$
    - $n/2^k$ vertices
    - Each 2^k SGC
    - Totally n SGC
  - Totally: $\log^* n$ groups, $n \log^* n$ SGC

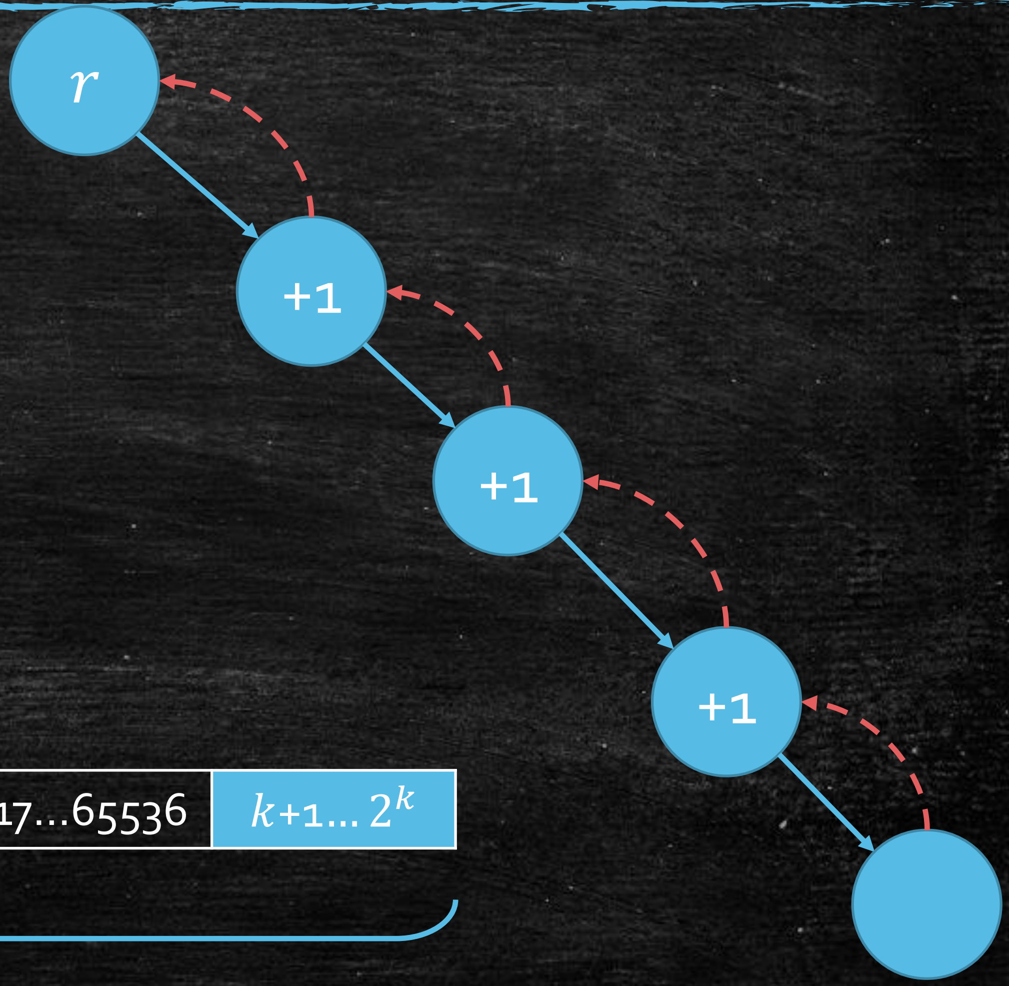
1…1	2…2	3…4	5…16	17…65536	k+1… 2^k

$\log^* n$



Bound Total cost

- Total Cost of m FIND
  - $O(m)(to\ root)$
  - Total AGC
    - $m \cdot \log^* n$
  - Total SGC
    - $n \cdot \log^* n$
  - Total : $m \log^* n$



1…1	2…2	3…4	5…16	17…65536	k+1… 2^k

$\log^* n$

Today’s goal

- Learn what is **Greedy**!
- Learn to use **Greedy** to finish homework!
- Learn Prim and Kruskal!
- Again, how to use **Data Structure** to improve **Algorithms**.
- Review Union-Find Set!
- Learn another **Amortized Analysis**!