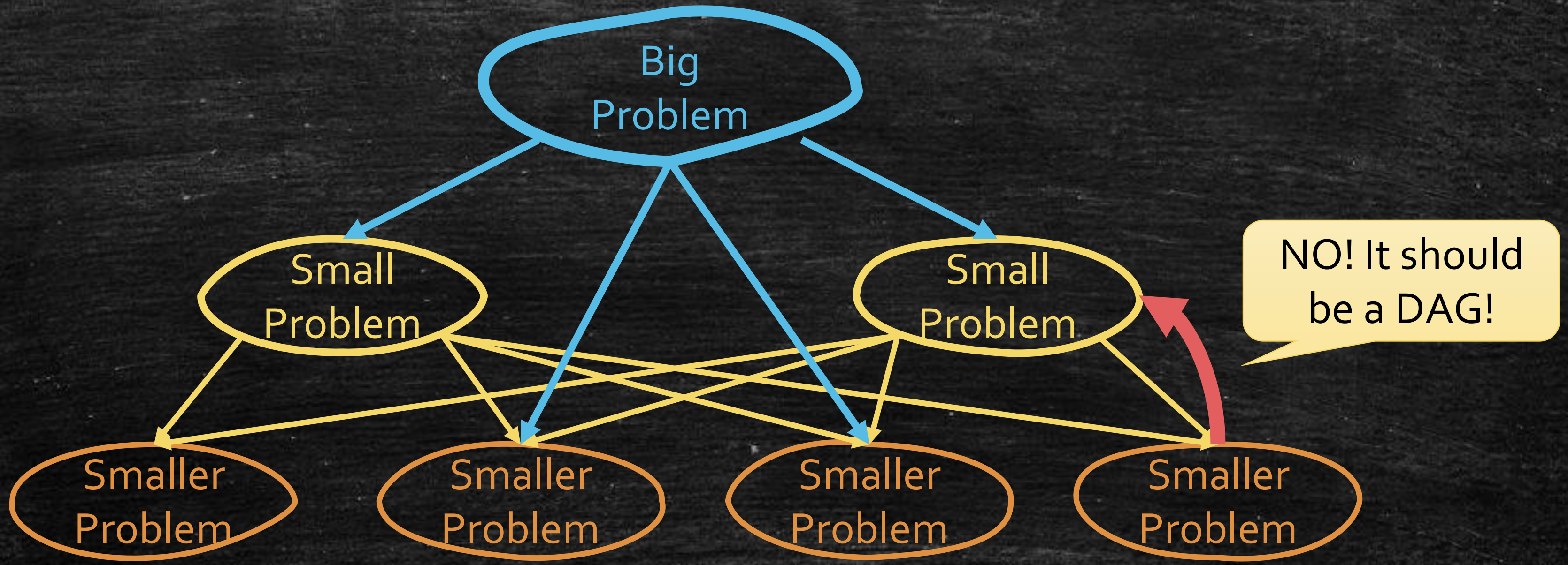
Dynamic Programming

Not So Efficient DP

Dynamic Programming

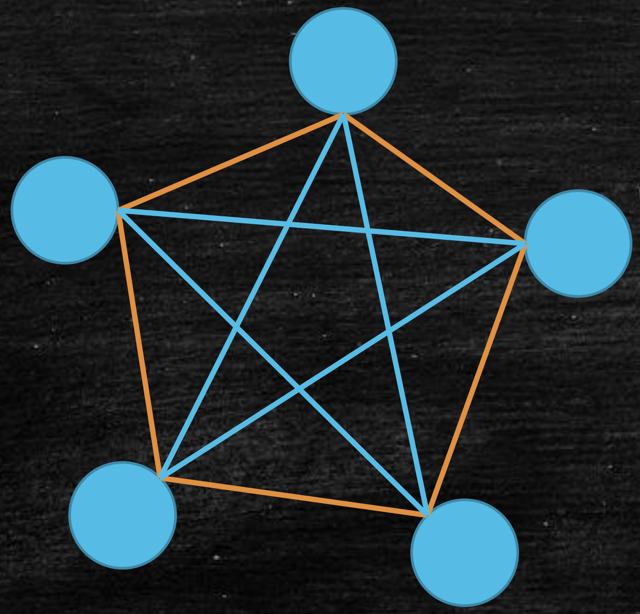


A simpler guideline

- Find subproblems.
- Check whether we are in a **DAG** and find the **topological order** of this DAG. (Usually, by hand.)
- Solve & store the subproblems by the topological order.

Traveling Salesman Problem (TSP)

- **Input:** A complete weighted undirected graph G, such that $d(u, v) > 0$ for each pair u, v $(u \neq v)$.
- **Output:** the cycle of n vertices with the minimum weight.

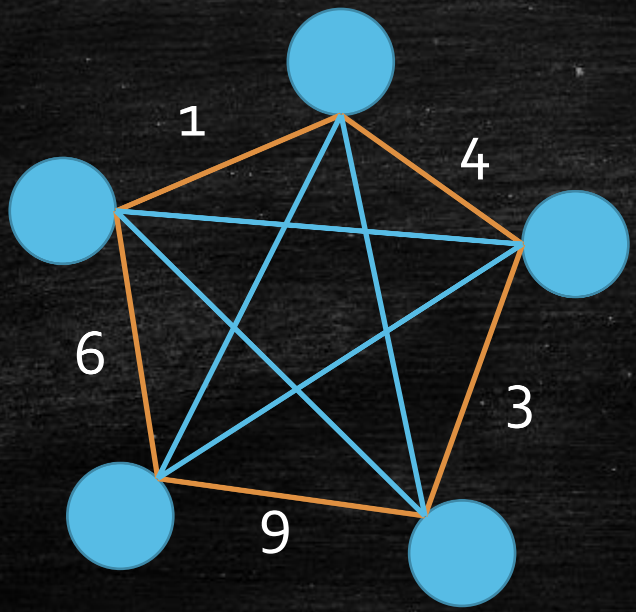


How to brute-force

What is the time complexity?

TSP vs. Shortest Path

- TSP
  - **Output:** the cycle of n vertices with the minimum weight.
- All Pair Shortest Path
  - **Output:** the minimum weight path from u to v.

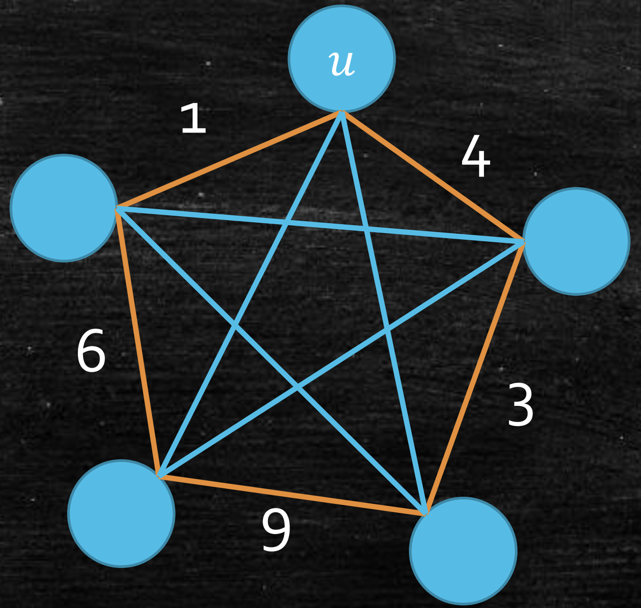


Subproblems in Shortest Path Problem

- $f[k, u, v]$
  - The shortest path from u to v, with inter-vertex chosen in $v_1 \dots v_k$.
- What we should do now?
- We can directly try this subproblem!

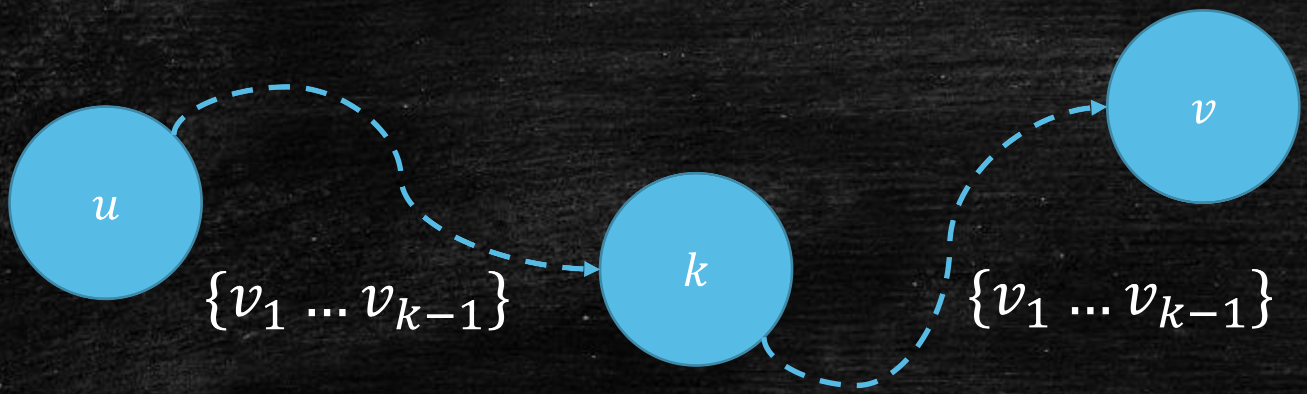
Plan A

- $f[k, u, v]$
  - The shortest path from u to v with inter-vertex **exactly** $v_1 \dots v_k$ except u and v.
- Hot to solve TSP?
  - $\min_u f[|V|, u, u]$ is what we want!
- How to solve $f[k, u, v]$?



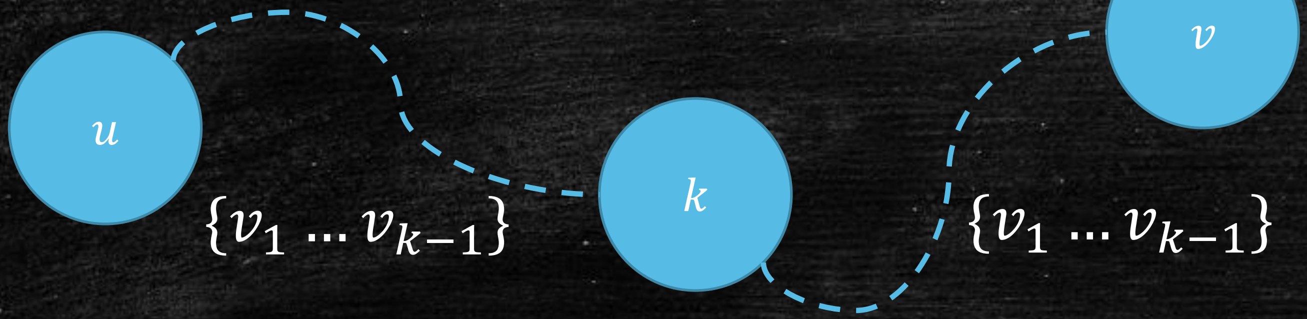
Floyd-Warshall: Solving Subproblems

- $dist[k, u, v]$: the shortest distance from u to v that only **across inter-vertices in $\{v_1 \dots v_k\}$.**
- Solve $dist[k, u, v]$ (give addition power k to all pairs)
  - Case 1: the shortest path do not go across k.
  - Case 2: the shortest path go across k.
  - $dist[k, u, v] = \min\{dist[k-1, u, v], dist[k-1, u, k] + dist[k-1, k, v]\}$



Plan A: Subproblem Definition

- $f[k, u, v]$
  - The shortest path from u to v with inter-vertex **exactly** $v_1 \dots v_k$ except u and v.
  - $\min_u f[|V|, u, u]$ is what we want!
- How to solve $f[k, u, v]$?
- What is the problem now?



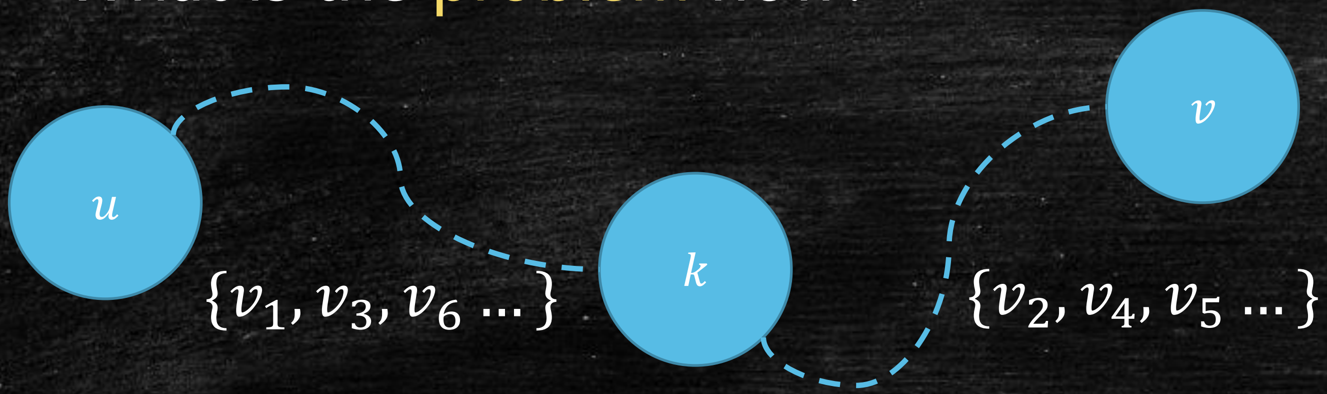
Two sub paths can not contain same vertices.

Plan A: Why it is not enough?

- $f[k, u, v]$
  - The shortest path from u to v with inter-vertex **exactly** $v_1 \dots v_k$ except u and v.
  - $\min_u f[|V|, u, u]$ is what we want!
- How to solve $f[k, u, v]$?
- What is the problem now?

We need to know

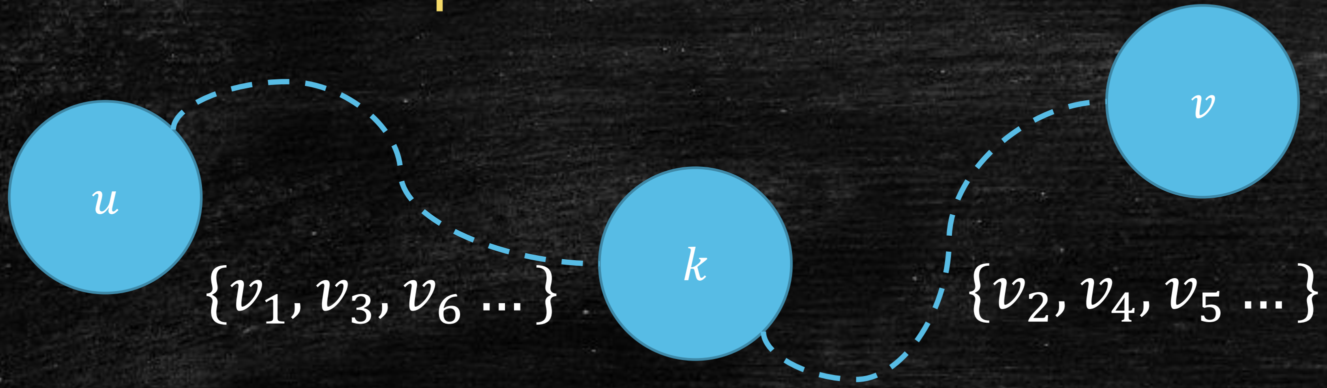
- what vertices $u \to k$ use?
- what vertices $k \to v$ use?



Plan A: Why it is not enough?

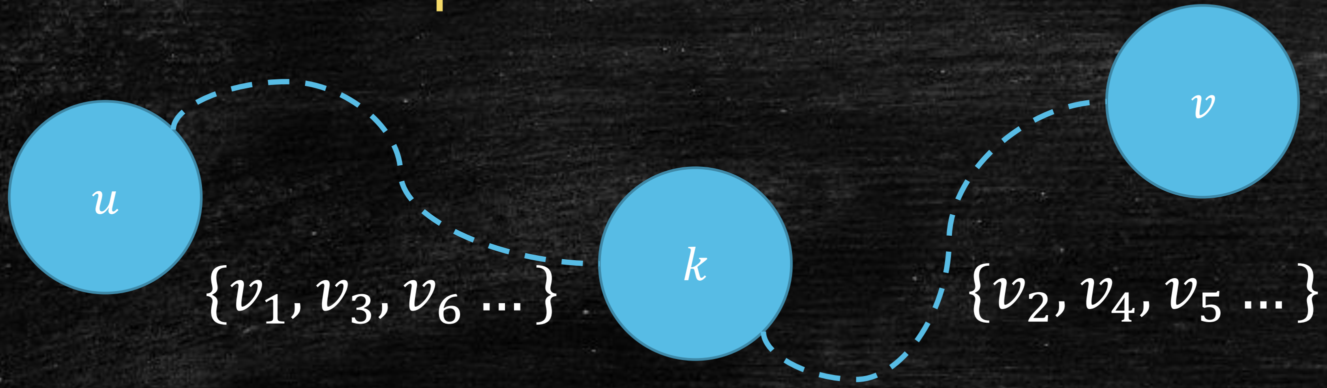
- $f[k, u, v]$
  - The shortest path from u to v with inter-vertex **exactly** $v_1 \dots v_k$ except u and v.
  - $\min_u f[|V|, u, u]$ is what we want!
- How to solve $f[k, u, v]$?
- What is the problem now?

We need to know
- what vertices $u \to k$ use?
- what vertices $k \to v$ use?



Plan A: Why it is not enough?

- $f[k, u, v]$
  - The shortest path from u to v with inter-vertex **exactly** $v_1 \dots v_k$ except u and v.
  - $\min\limits_{u} f[|V|, u, u]$ is what we want!
- How to solve $f[k, u, v]$?
- What is the problem now?



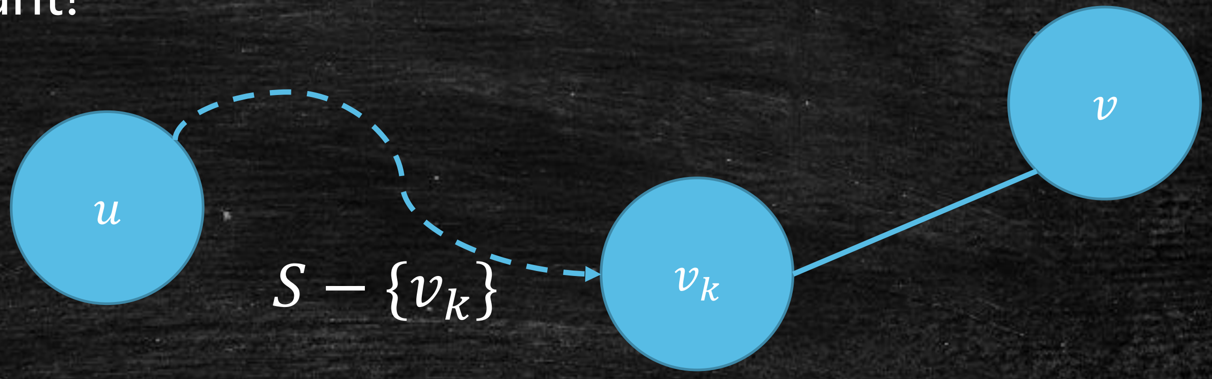
We need to know
- what vertices $u \to k$ use?
- what vertices $k \to v$ use?

We do not solve the subproblem $u \to k$ with $\{v_1, v_3, v_6 \dots\}$

Do you know how to fix?

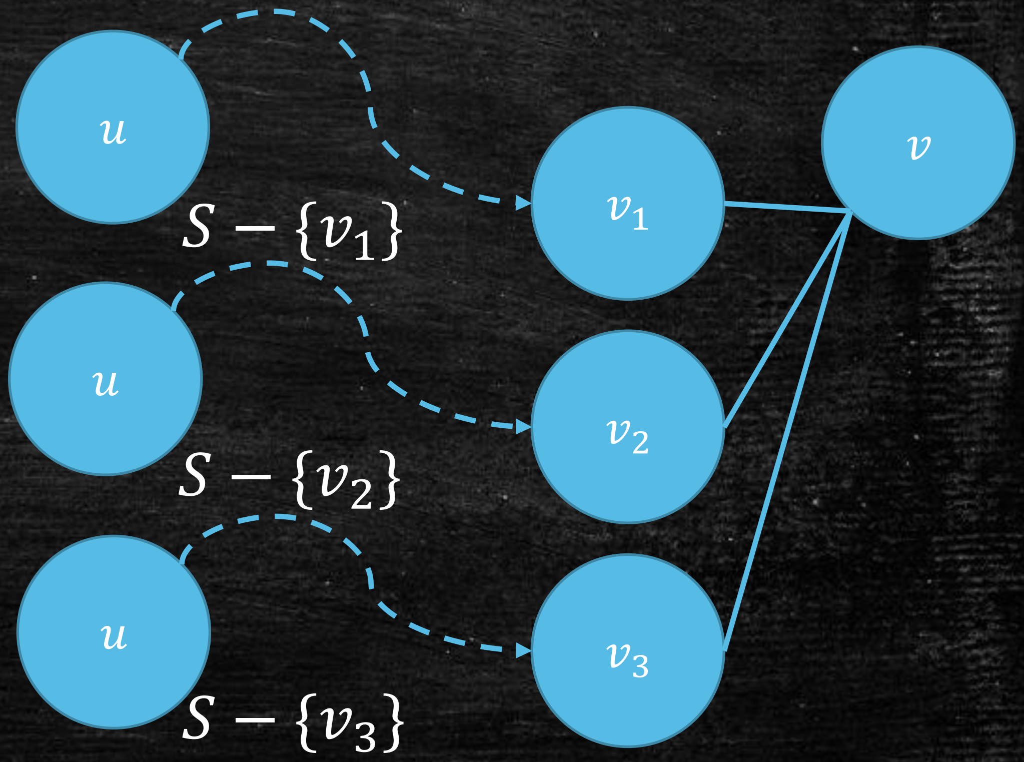
Plan B: Subproblem Definition

- $f[\boldsymbol{S}, u, v]$
  - The shortest path from u to v with inter-vertex **exactly** $\boldsymbol{S \subset V}$ except u and v.
  - $\min_u f[V, u, u]$ is what we want!
- How to solve $f[\boldsymbol{S}, u, v]$?



Plan B: Solving Subproblems

- $f[\boldsymbol{S}, u, v]$
  - The shortest path from u to v with inter-vertex **exactly** $\boldsymbol{s} \subset \boldsymbol{V}$ except u and v.
  - $\min_{u} f[V, u, u]$ is what we want!
- How to solve $f[\boldsymbol{S}, u, v]$?
- $f[S, u, v] = \min_{k \in V} f[S - \{k\}, u, k] + d(k, v)$

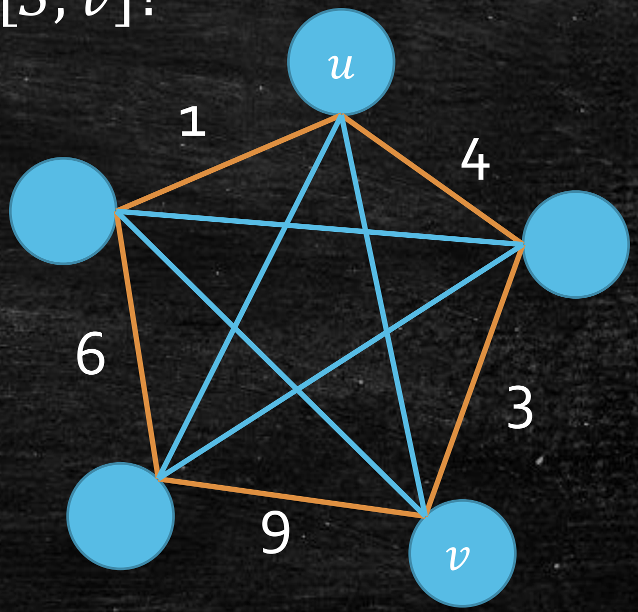


Summarize

- $f[\mathbf{S}, u, v]$
  - The shortest path from u to v with inter-vertex **exactly** $\boldsymbol{s \subset V}$ except u and v.
  - $\min_u f[V, u, u]$ is what we want!
- $f[S, u, v] = \min_{k \in V} f[S - \{k\}, u, k] + d(k, v)$
- Do you know the topological order of the DP?

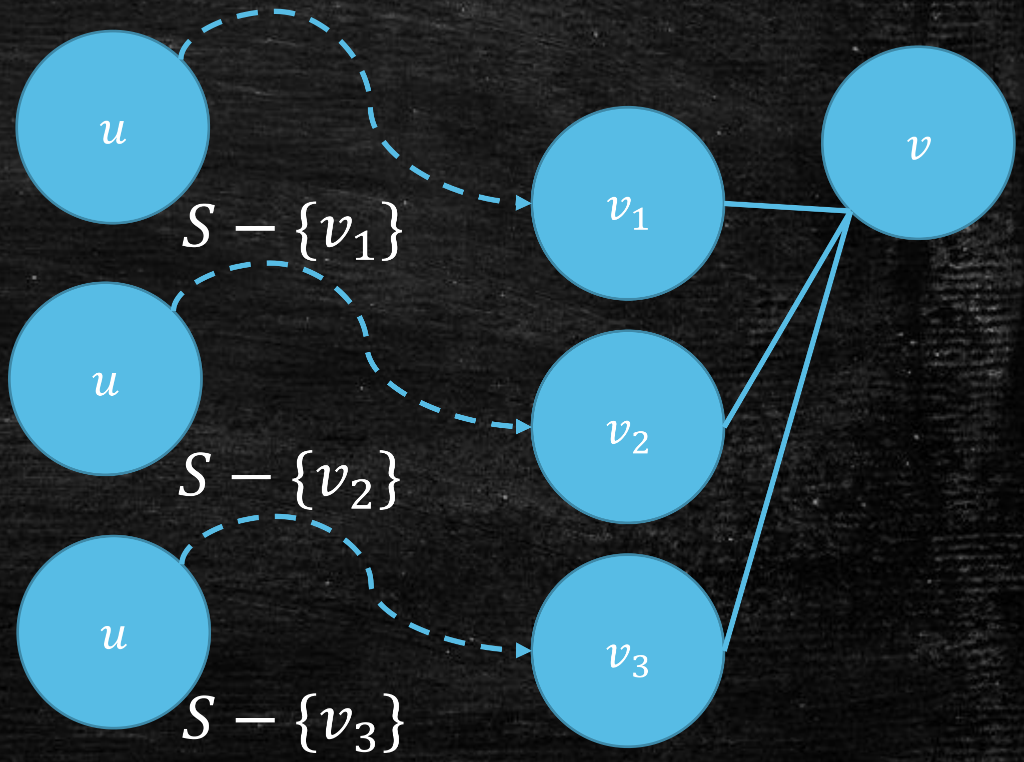
A little improvement:

- $\forall u, v, f[V, u, u] = f[V, v, v]$!
- We only need to know one fixed $f[V, u, u]$.
- Can we fix an arbitrary u and only solve $f[S, v]$?



Solve $f[S, v]$!

- $f[S, u, v] = \min_{k \in V} f[S - \{k\}, u, k] + d(k, v)$
- $f[S, u, v]$ only comes from $f[S - \{k\}, u, k]$.
- It is enough for us to only record $f[S, v]$.
- $f[S, v] = \min_{k \in V} f[S - \{k\}, k] + d(k, v)$.



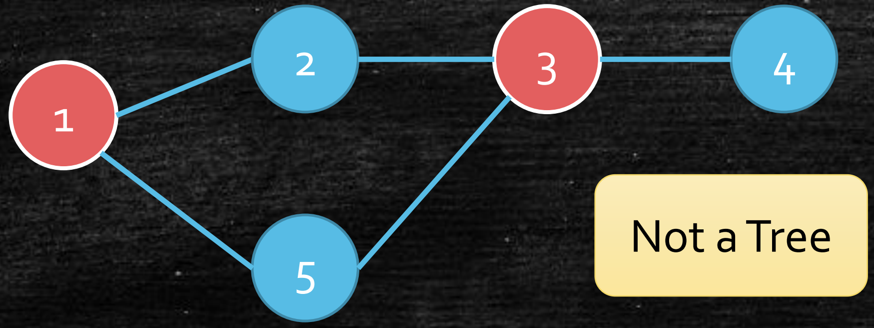
Time Complexity

- Time complexity $(n = |V|)$?
  - $O(n2^n)$ subproblems.
  - $O(n)$ solving.
  - $\boldsymbol{O(n^2 2^n)}$ totally!
- Comparing to Brute-force
  - Brute-force: $\boldsymbol{O(n!)}$
  - Do you know why $O(n^2 2^n)$ is better than $O(n!)$?
  - Do you know why DP is better than brute-force?
- Do you know how to implement $f[S, v]$?

Solve Problems on Trees

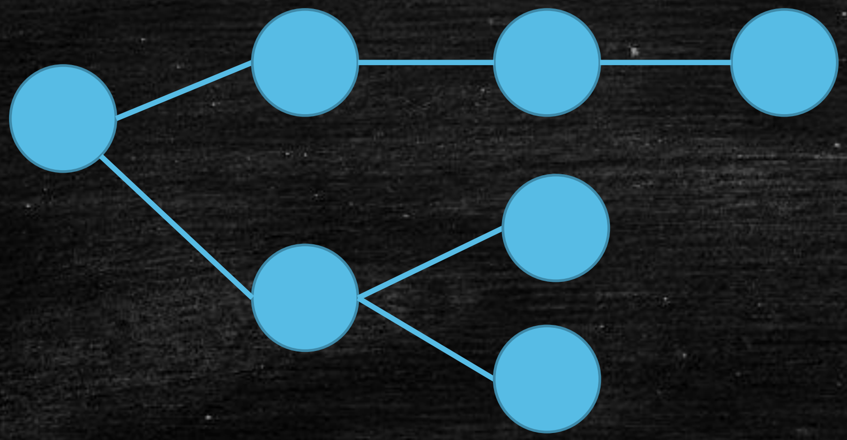
Maximize Independent Set on Trees

- **Input:** an undirected tree $G = (V, E)$.
- **Output:** an independent set with maximum cardinality (number of vertices)
- **Independent Set:** a set S of vertices:
  - $\forall u, v \in S$, we have $(u, v) \notin E$



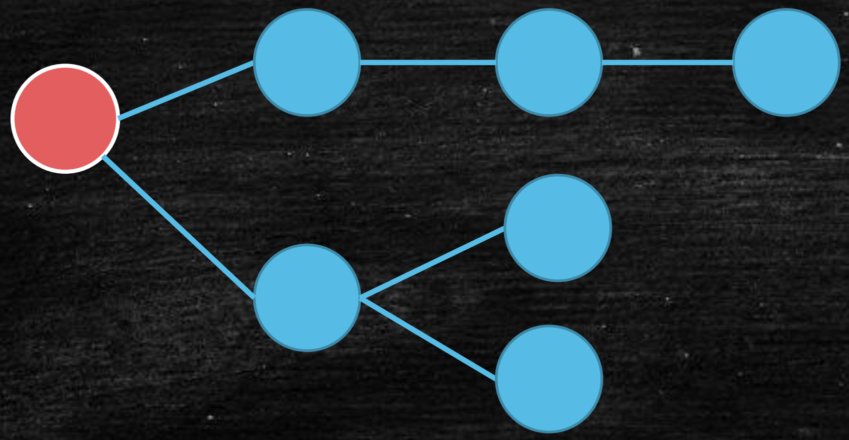
Maximize Independent Set ~~on Trees~~

- Maximize Independent Set ~~on Trees~~ is NP-hard.
- Is the tree special case easier?



Solve it Recursively

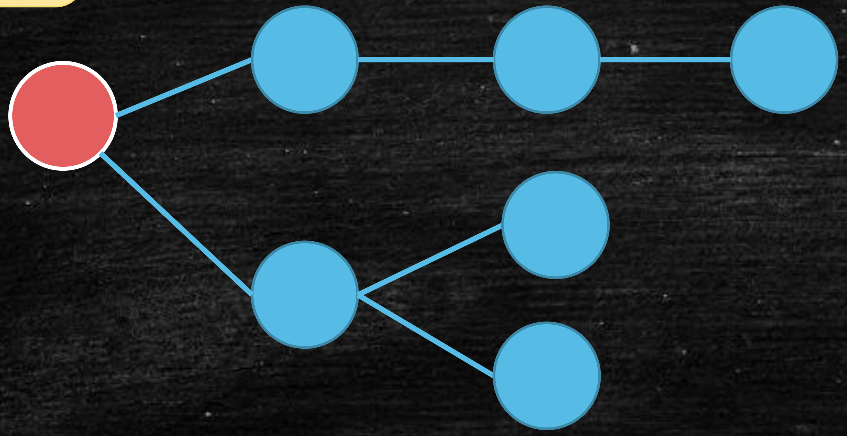
- Start from the root
  - Case 1: We choose the root, what happens?
  - Case 2: We do not choose the root, what happens?



Start from recursive

- Start from the root
  - Case 1: We choose the root, what happens?
  - Case 2: We do not choose the root, what happens?

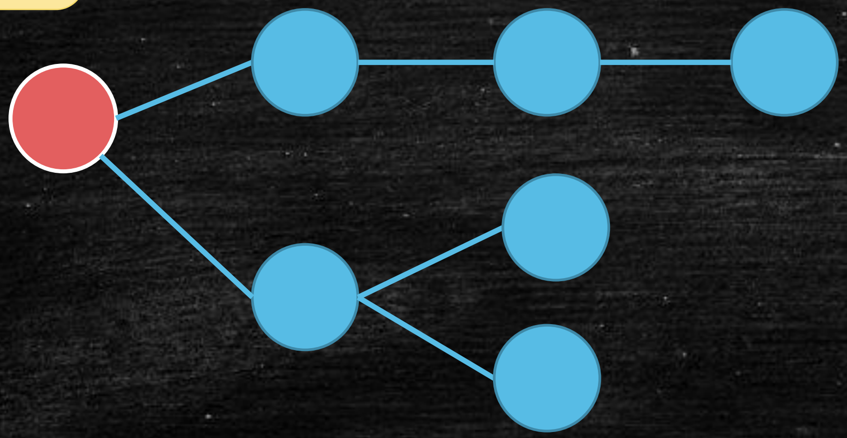
Case 1: We can not choose its children.



Start from recursive

- Start from the root
  - Case 1: We choose the root, what happens?
  - Case 2: We do not choose the root, what happens?

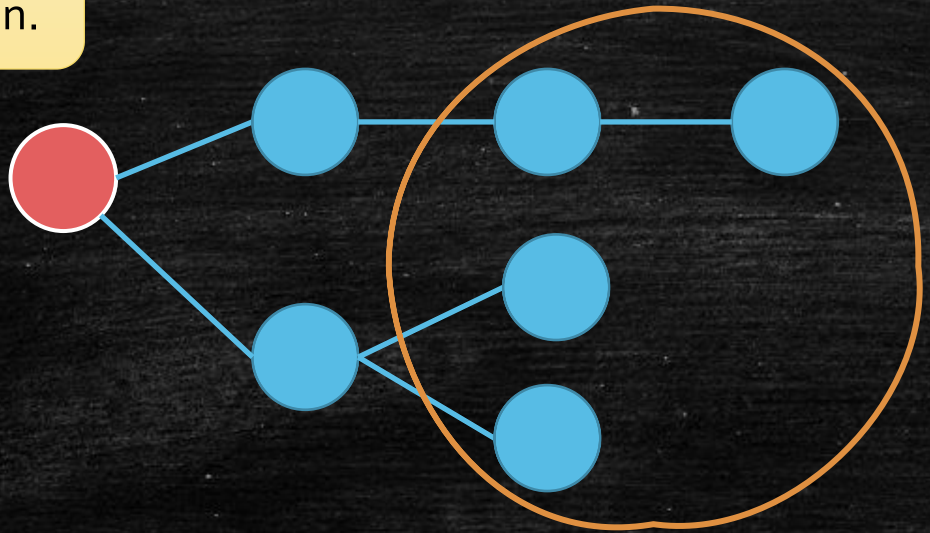
Case 2: We can choose its children.



What subproblems do we need to solve?

We need to know the max independent set here.

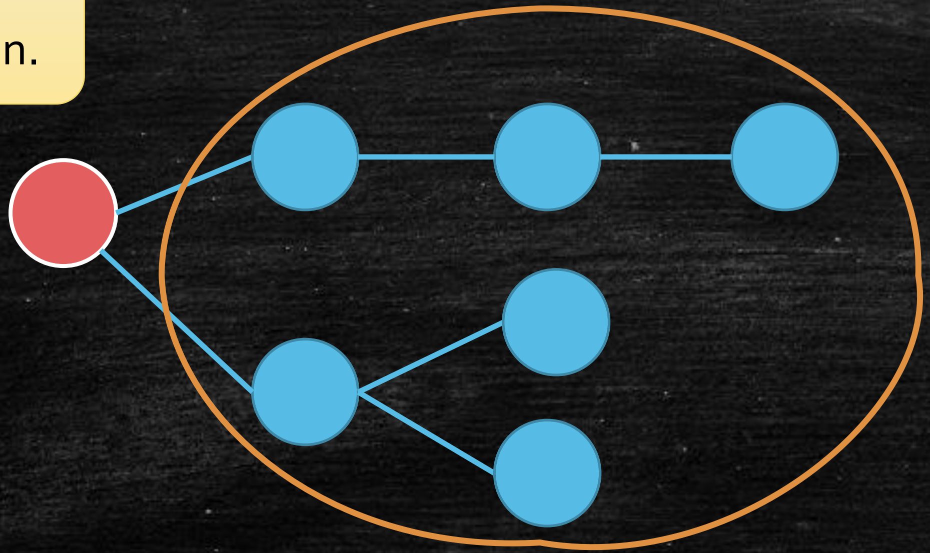
Case 1: We can not choose its children.



What subproblems do we need to solve?

We need to know the max independent set here.

Case 2: We can choose its children.

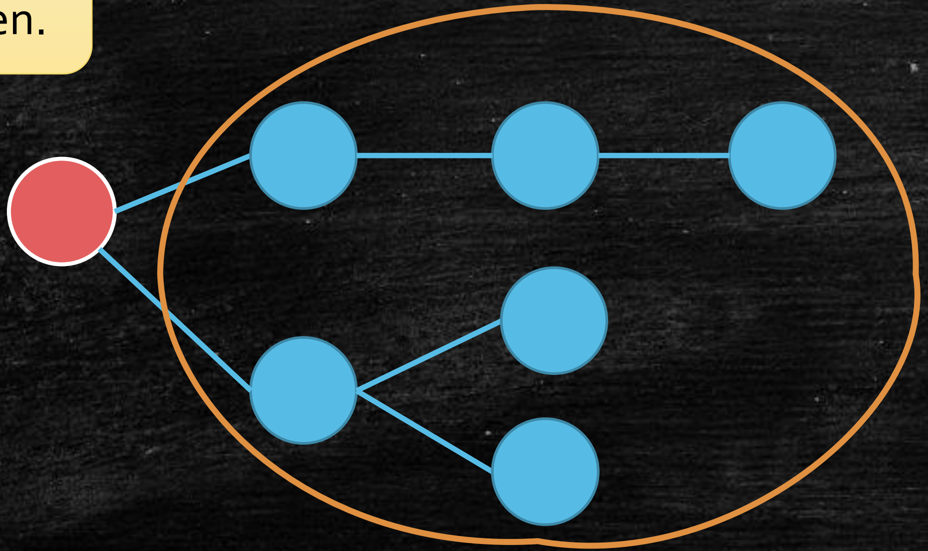


How to define subprobelms?

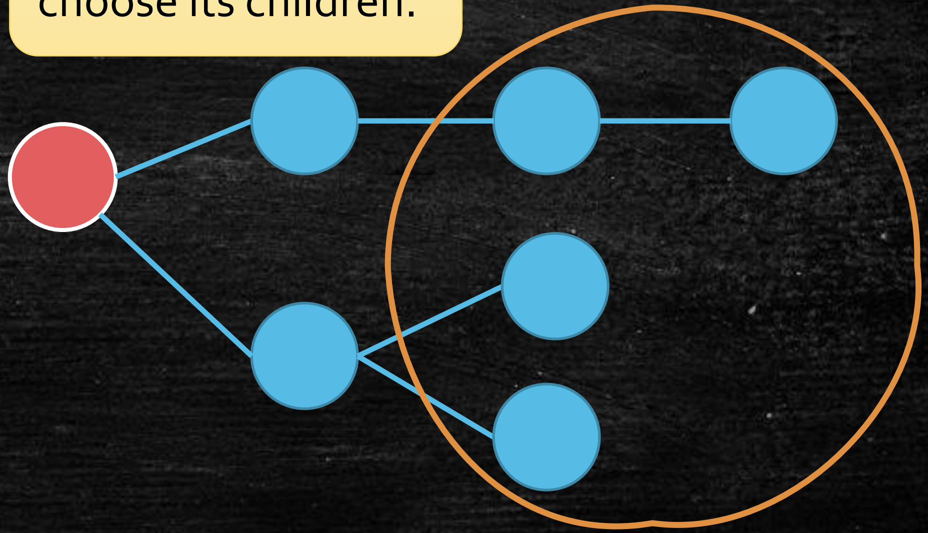
Subproblem Definition

- Subproblem $f[v]$: the maximized size of independent set of the subtree rooted at v.

Case 2: We can choose its children.



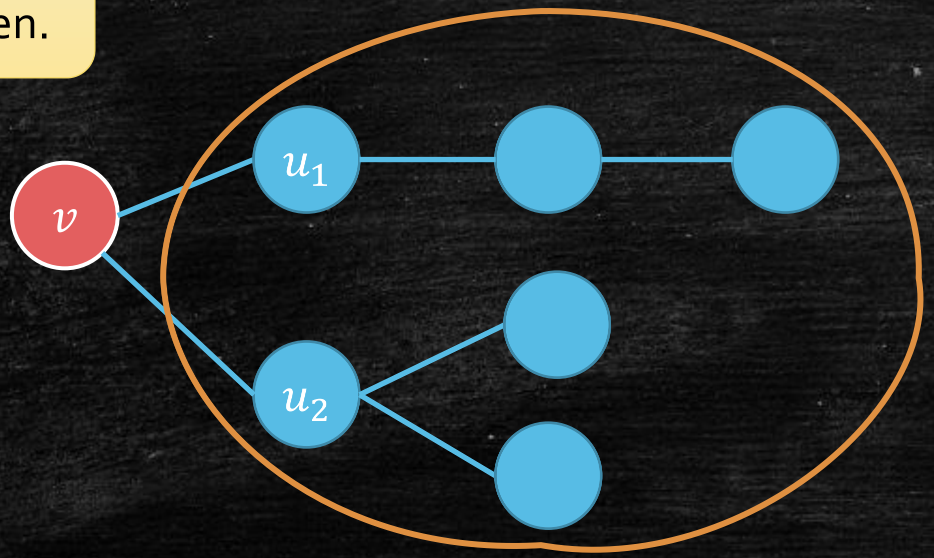
Case 1: We can not choose its children.



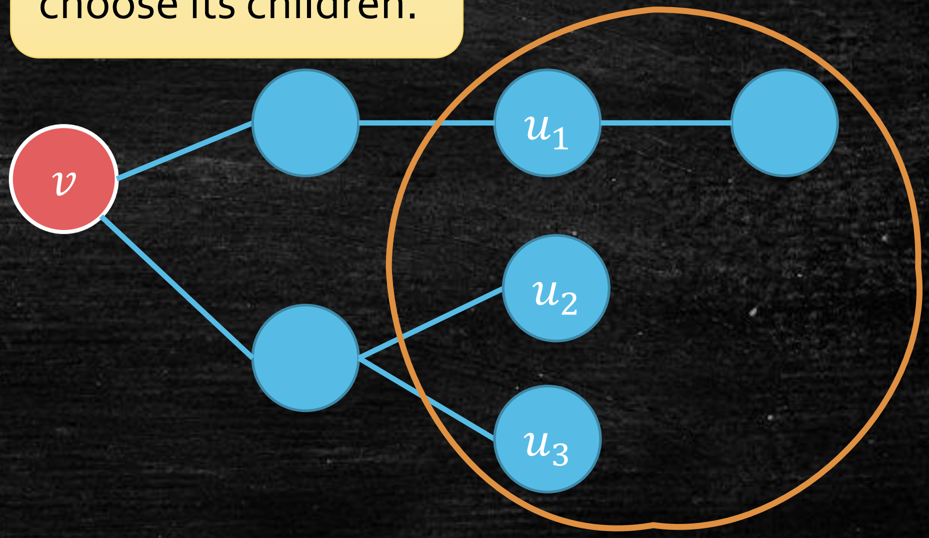
Subproblem Solving

- Subproblem $f[v]$: the maximized size of independent set of the subtree rooted at v.
- $f[v] = \max\{\sum_{u \in children(v)} f[u], \sum_{u \in grandchildren(v)} f[u] + 1\}$

Case 2: We can choose its children.



Case 1: We can not choose its children.



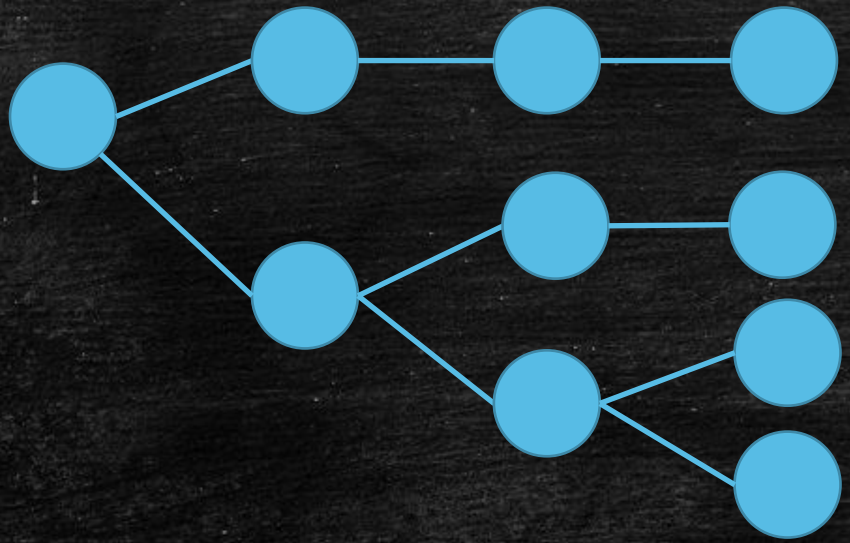
Running Time

- $f[v] = \max\{\sum_{u \in children(v)} f[u], \sum_{u \in grandchildren(v)} f[u] + 1\}$
- Looks $O(n^2)$.
  - We have n subproblems.
  - Each take $O(n)$ times.
- But it is $O(n)$!
  - Each of its children and its grandchildren cost one.
  - On other words, each vertex only need to pay one for its **parent** and one for its **grandparent**.
  - Totally $O(n)$.
  - Question: how to find a bottom-up order?

What is the topological order of the DP?

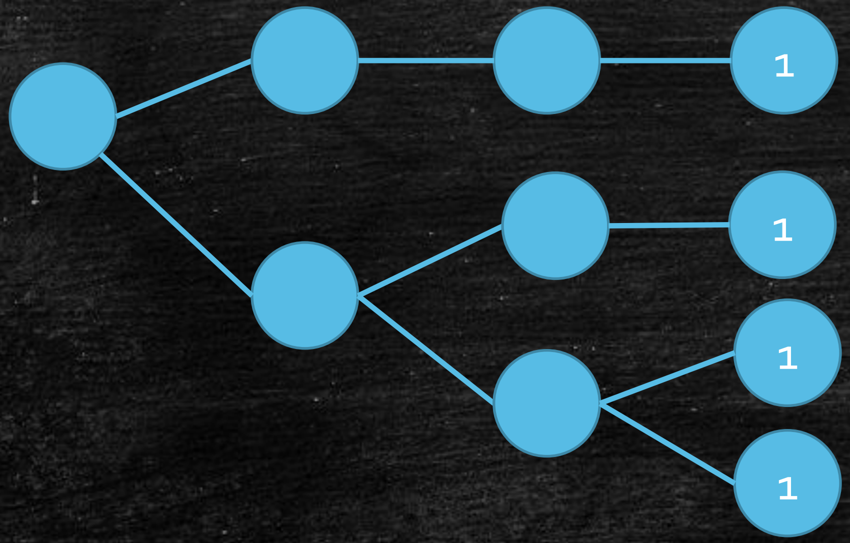
It is also a greedy algorithm!

- Try to solve it bottom-up!



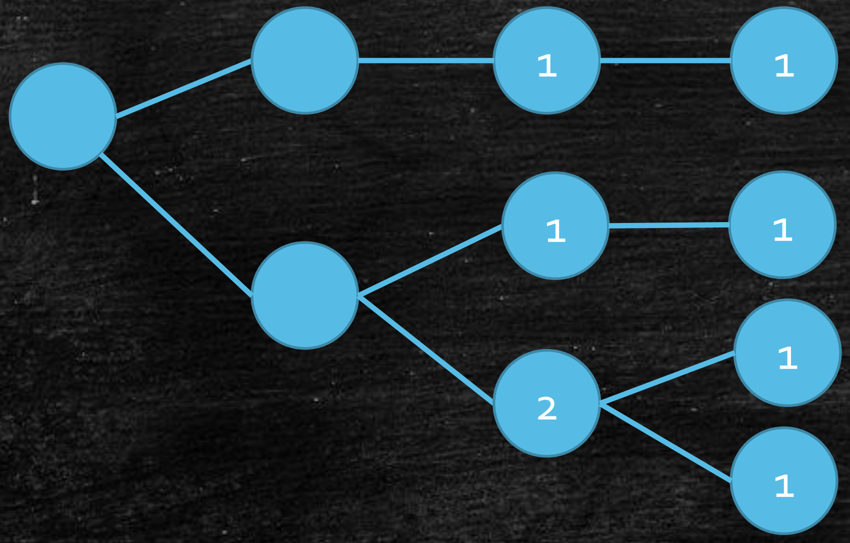
It is also a greedy algorithm!

- Try to solve it bottom-up!



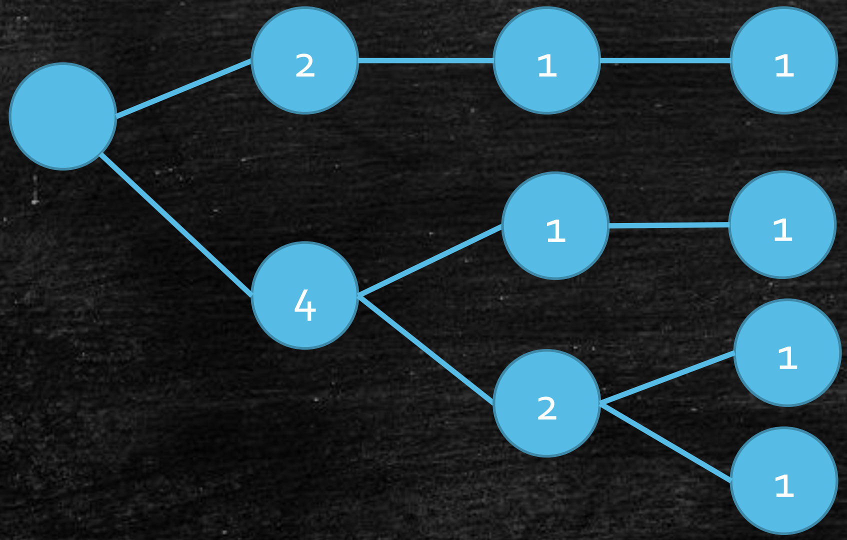
It is also a greedy algorithm!

- Try to solve it bottom-up!



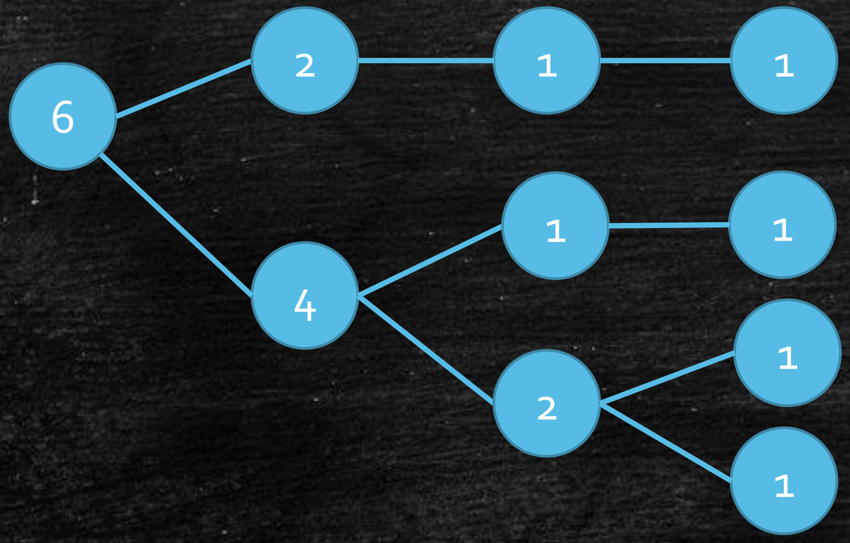
It is also a greedy algorithm!

- Try to solve it bottom-up!



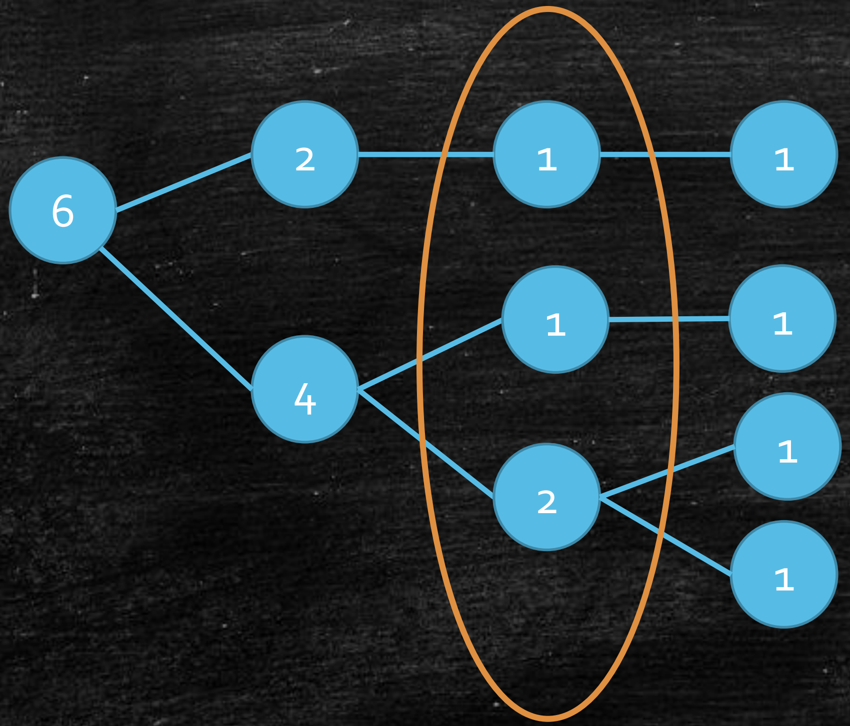
It is also a greedy algorithm!

- Try to solve it bottom-up!



It is also a greedy algorithm!

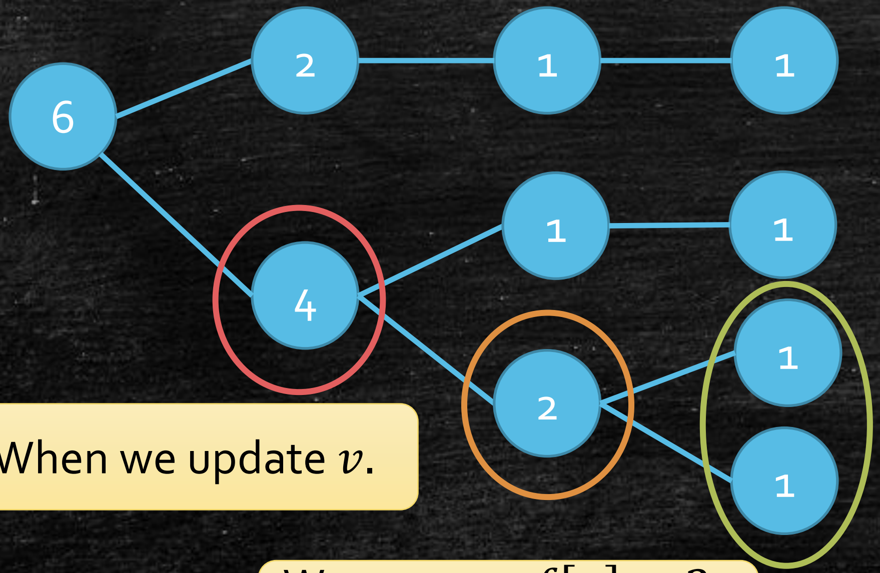
- Try to solve it bottom-up!



They are useless!

It is also a greedy algorithm!

- Try to solve it bottom-up!



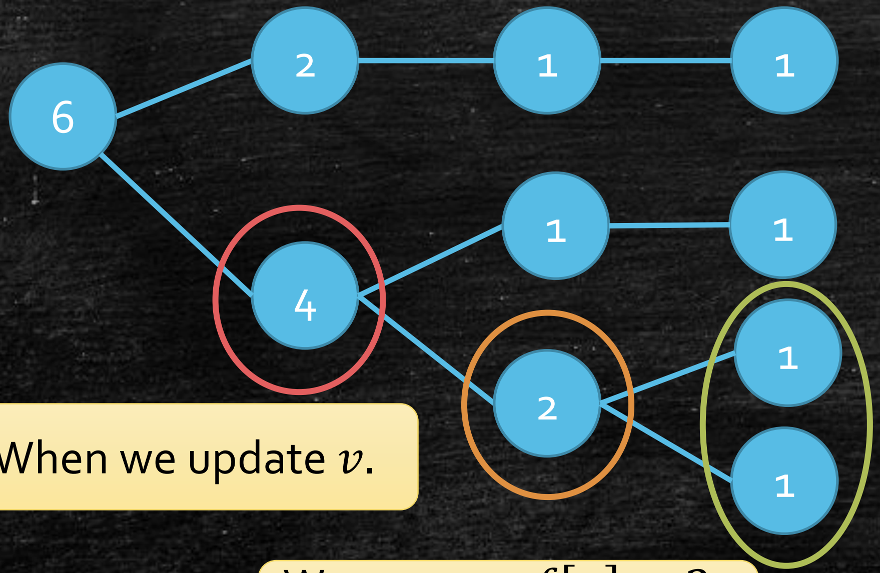
When we update v.

We may use $f[u] = 2$, with restriction.

Why not use $f[u_1] + f[u_2]$ without restriction?

It is also a greedy algorithm!

- Try to solve it bottom-up!
- $\sum_{u \in children(v)} f[u]$



When we update v.

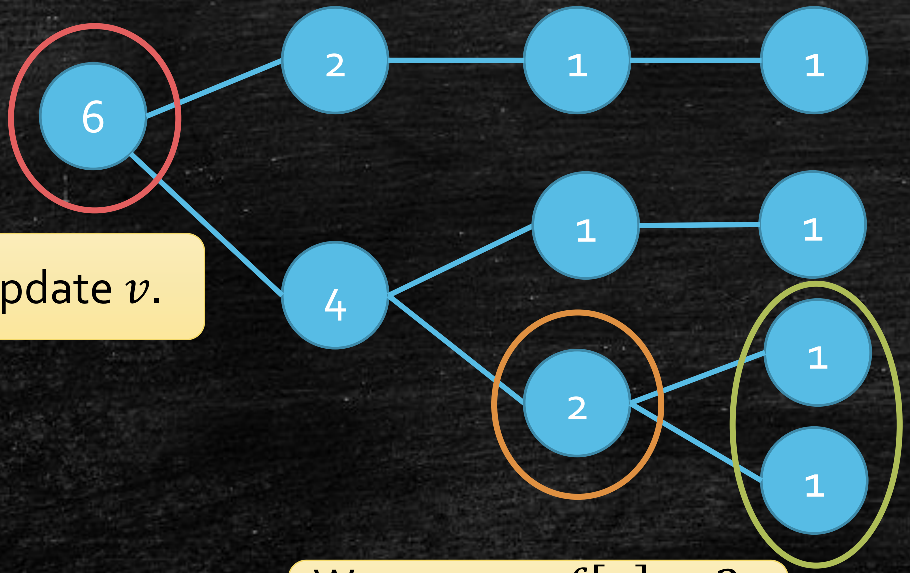
We may use $f[u] = 2$, with **restriction**.

Why not use $f[u_1] + f[u_2]$ **without restriction**?

It is also a greedy algorithm!

- Try to solve it bottom-up!
- $\sum_{u \in grandchildren(v)} f[u] + 1$

When we update v.

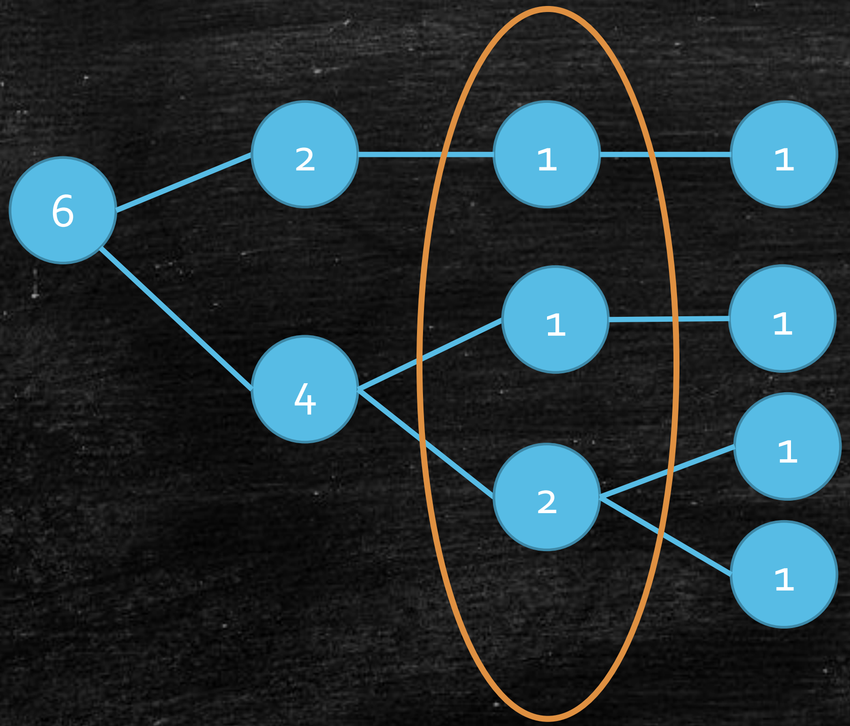


Why not use $f[u_1] + f[u_2]$ **without restriction**?

We may use $f[u] = 2$, **without restriction**.

It is also a greedy algorithm!

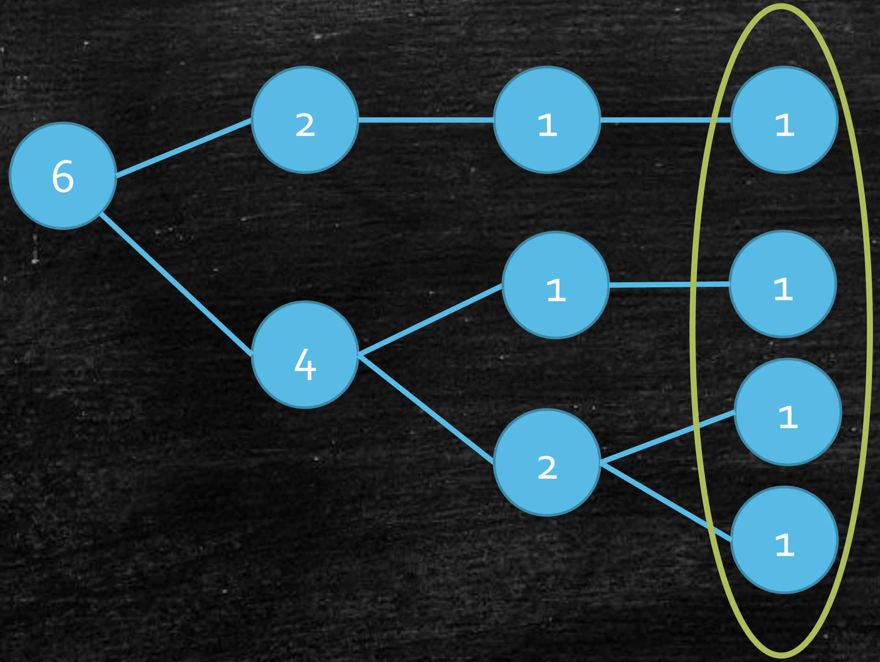
- Try to solve it bottom-up!



They are useless!

It is also a greedy algorithm!

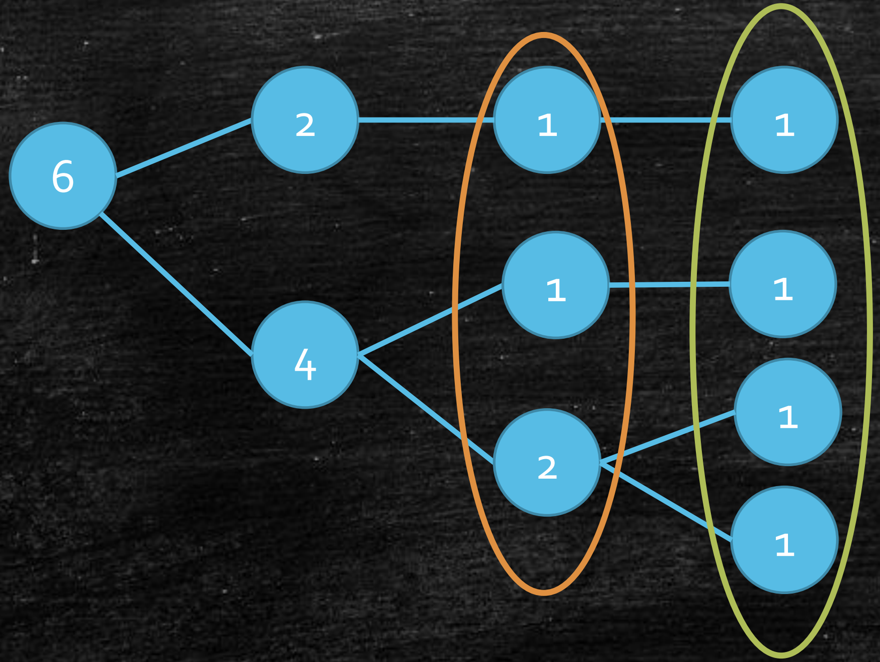
- Try to solve it bottom-up!



They are super useful!
They can update their ancestor **without restriction**!

It is also a greedy algorithm!

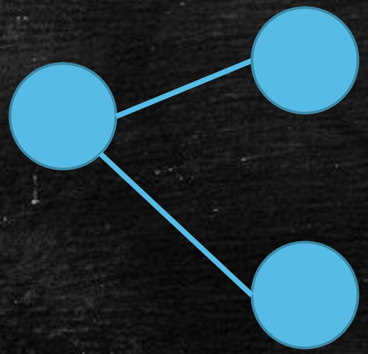
- Try to solve it bottom-up!

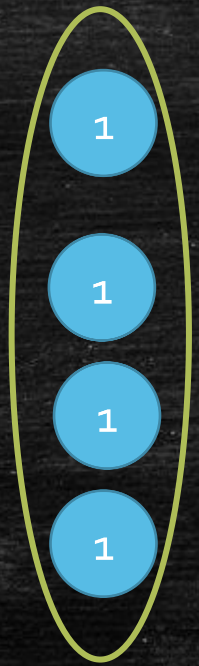


It is same to say, we choose Green and remove Orange.

It is also a greedy algorithm!

- Try to solve it bottom-up!

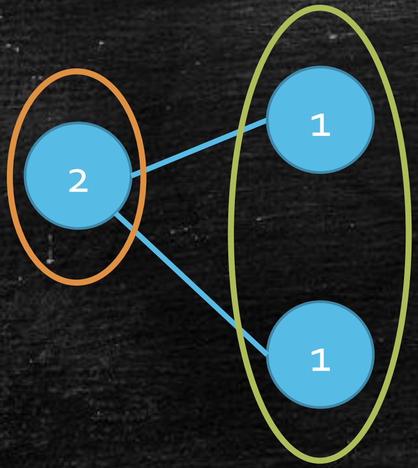




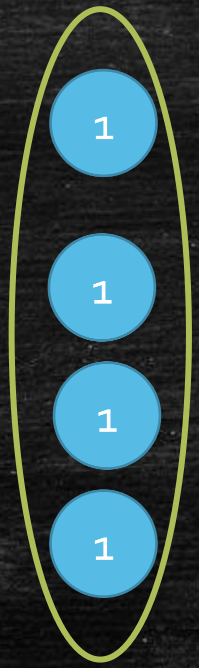
It is same to say, we choose Green and remove Orange.

It is also a greedy algorithm!

- Try to solve it bottom-up!



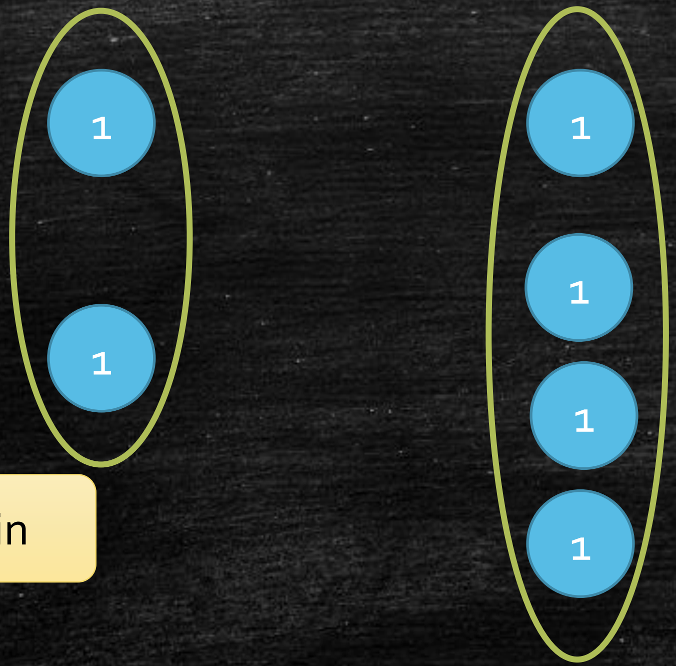
Do it again



It is same to say, we choose Green and remove Orange.

It is also a greedy algorithm!

- Try to solve it bottom-up!



Do it again

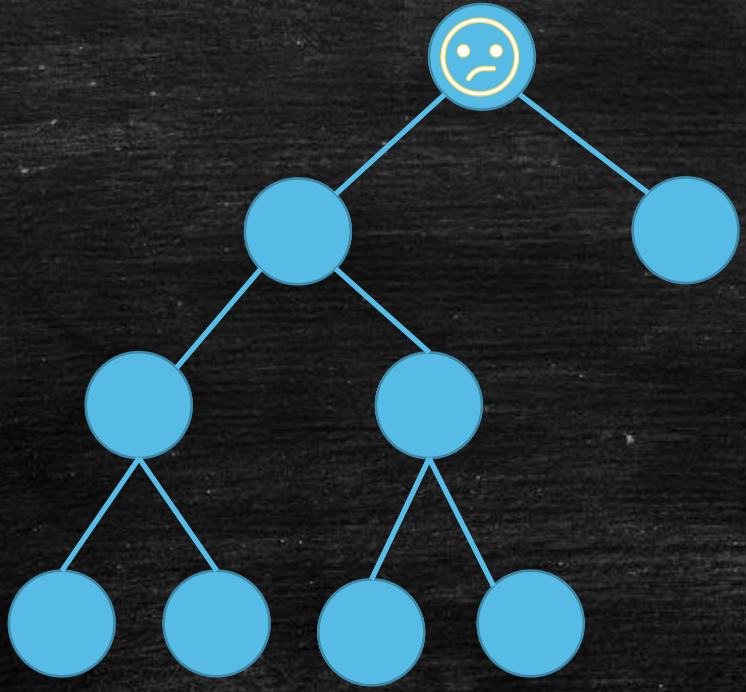
It is same to say, we choose Green and remove Orange.

The Greedy Algorithm

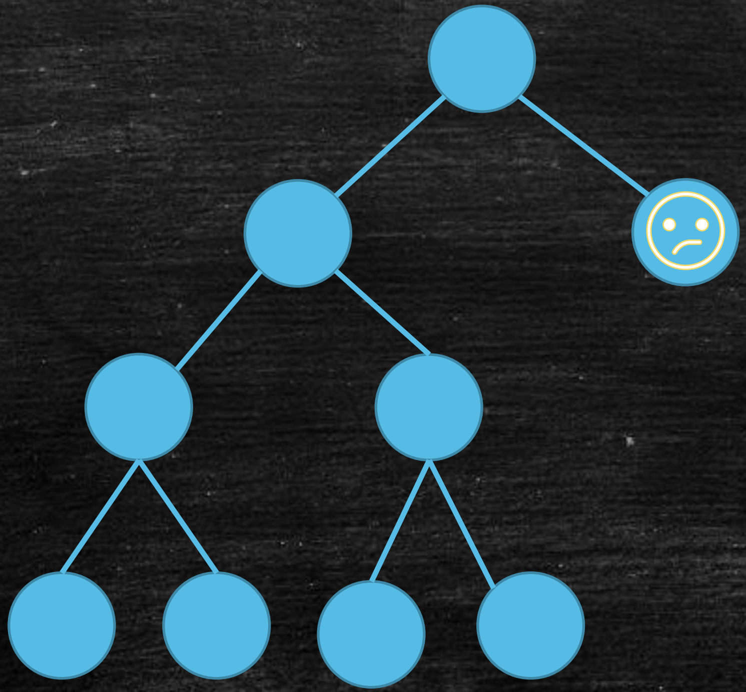
1. Choose all Leaves.
2. Remove all leaves' parents.
3. Repeat 1 again.

- How to implement it in $O(|V|)$?

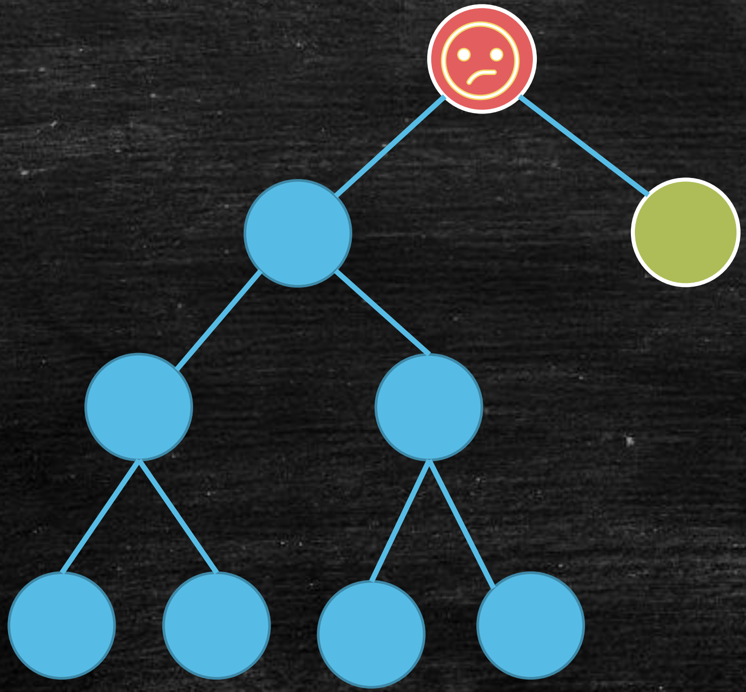
Greedy Algorithm Implementation



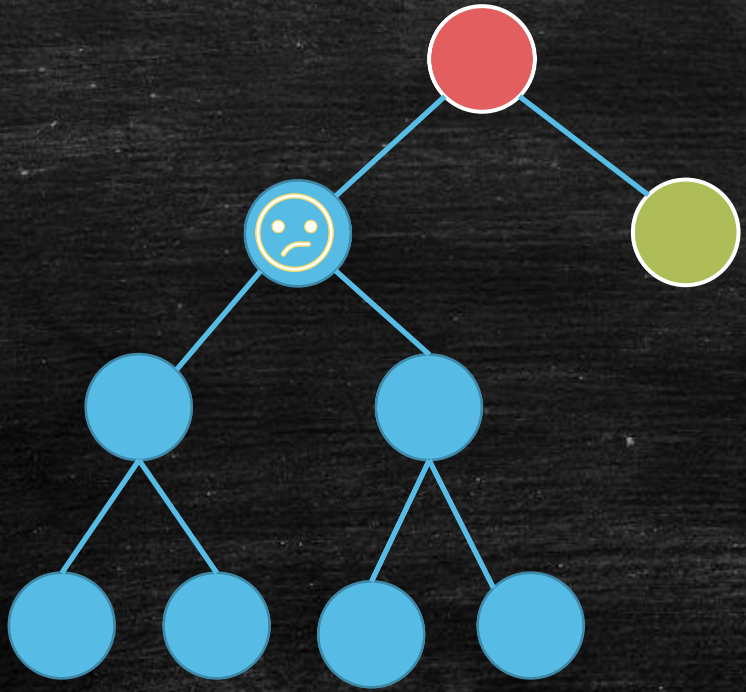
Greedy Algorithm Implementation



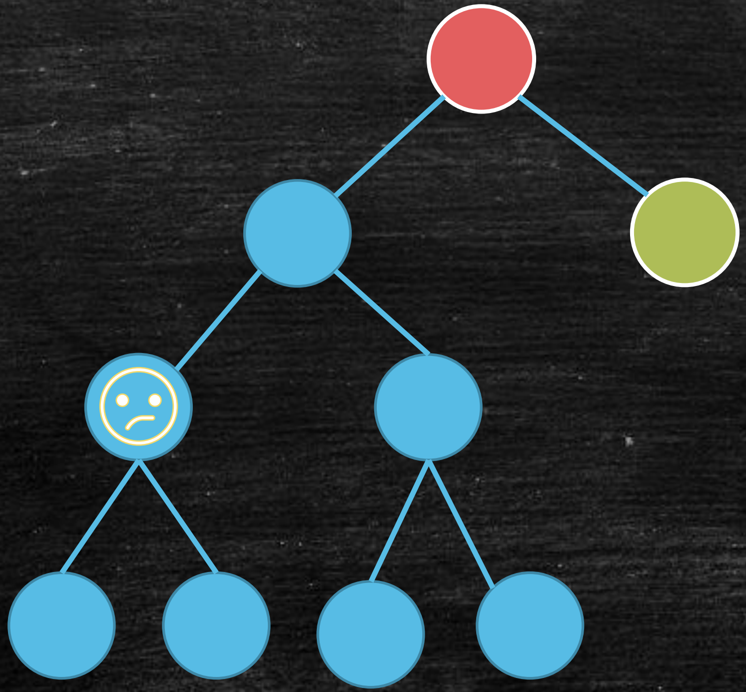
Greedy Algorithm Implementation



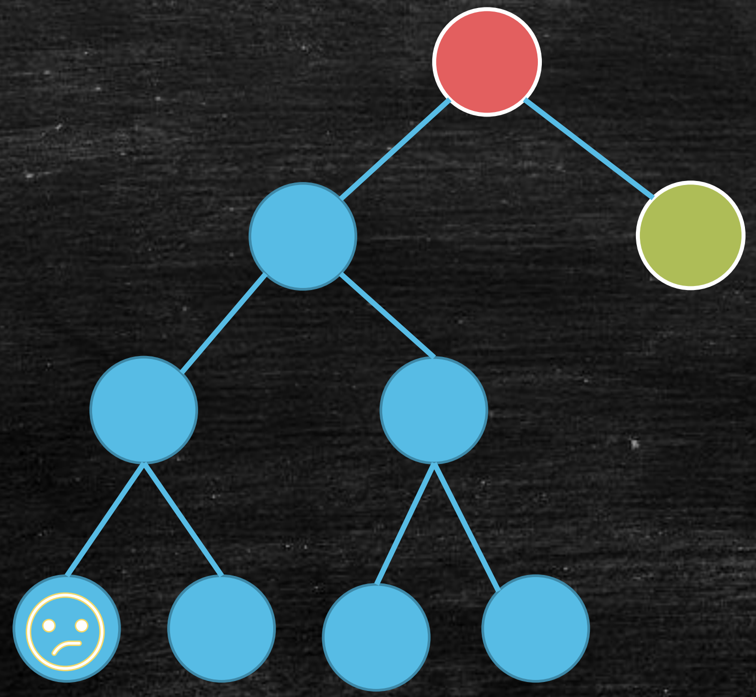
Greedy Algorithm Implementation



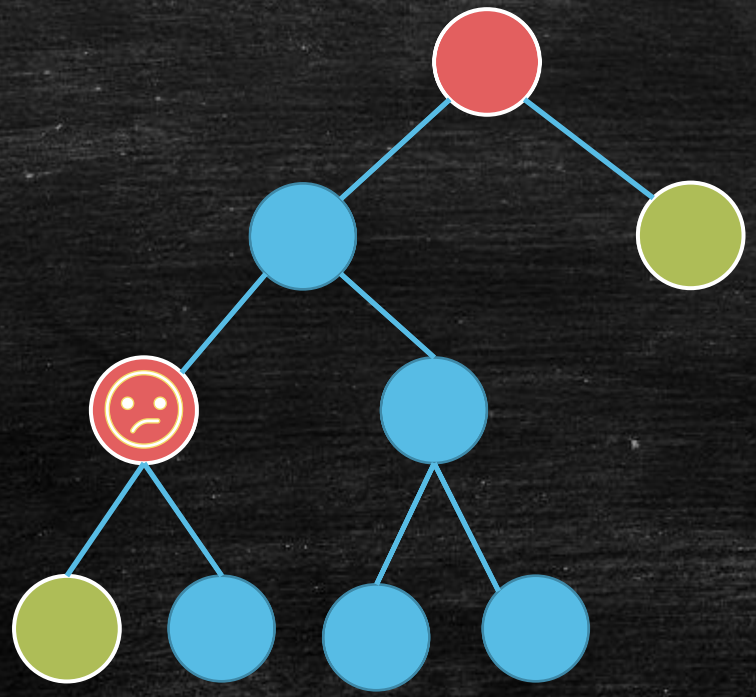
Greedy Algorithm Implementation



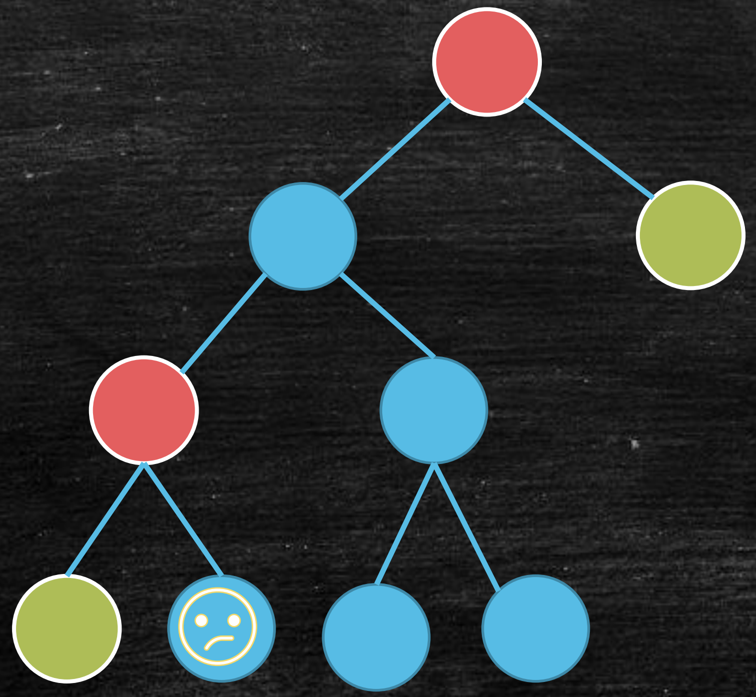
Greedy Algorithm Implementation



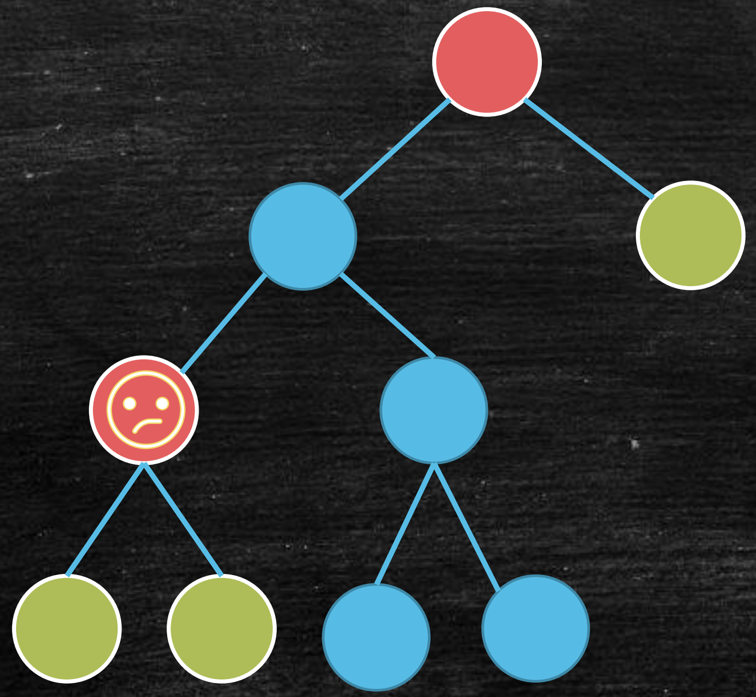
Greedy Algorithm Implementation



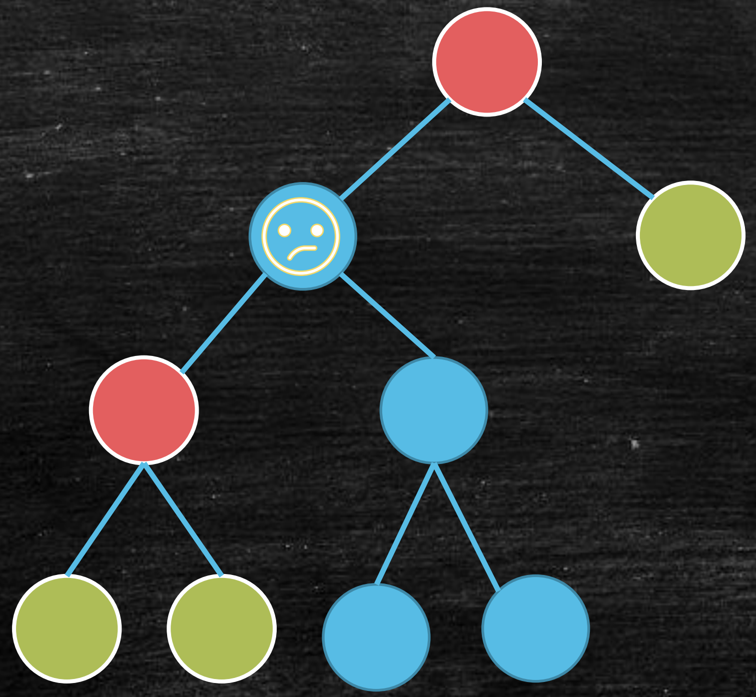
Greedy Algorithm Implementation



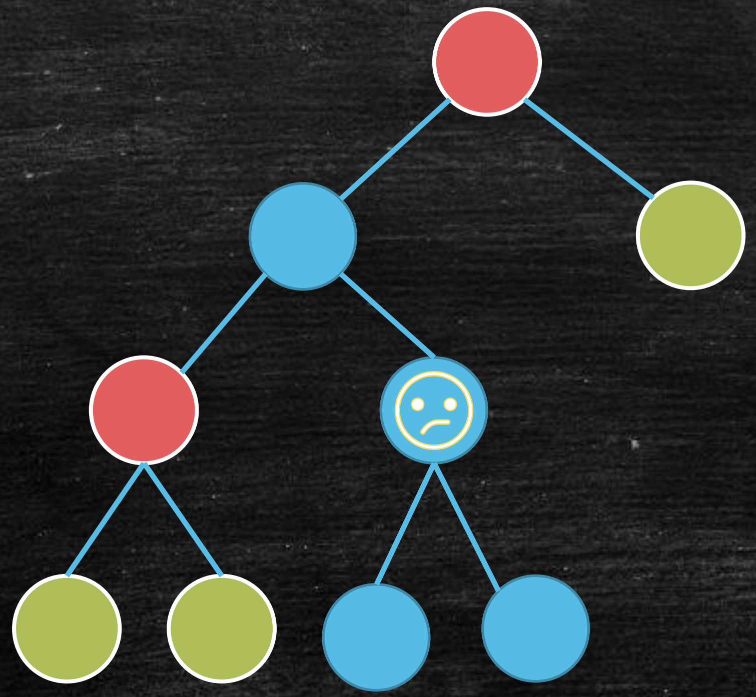
Greedy Algorithm Implementation



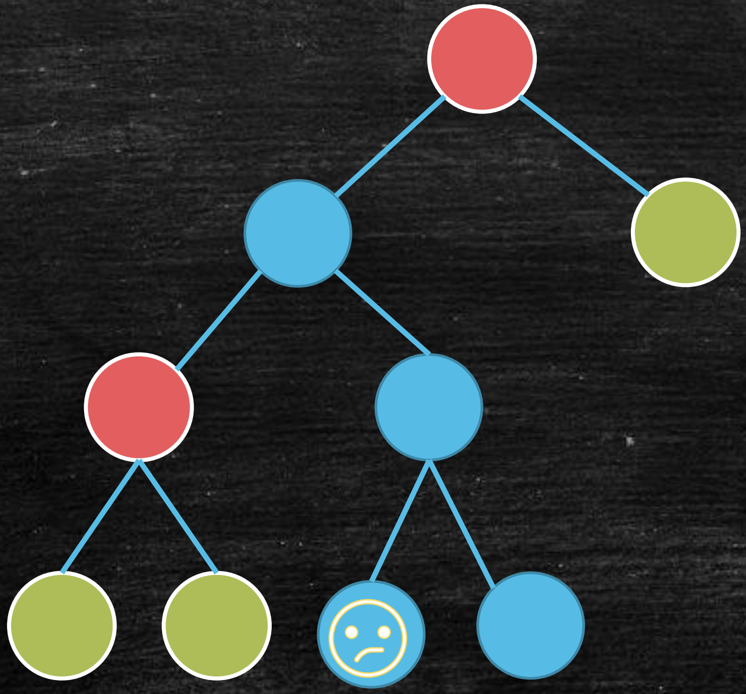
Greedy Algorithm Implementation



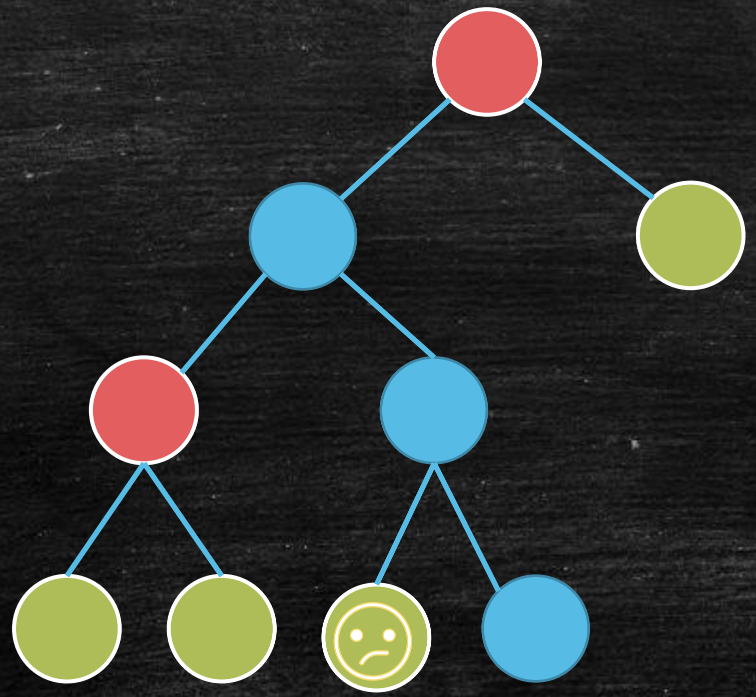
Greedy Algorithm Implementation



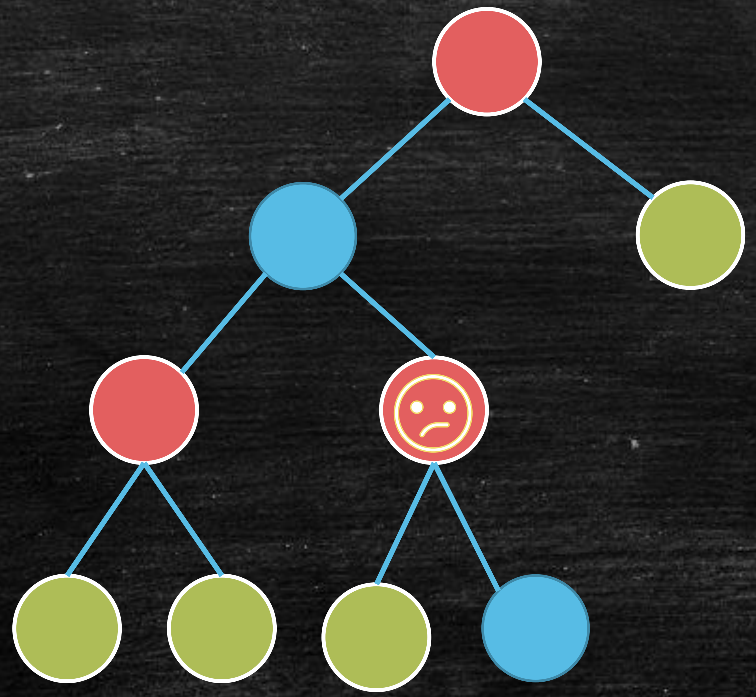
Greedy Algorithm Implementation



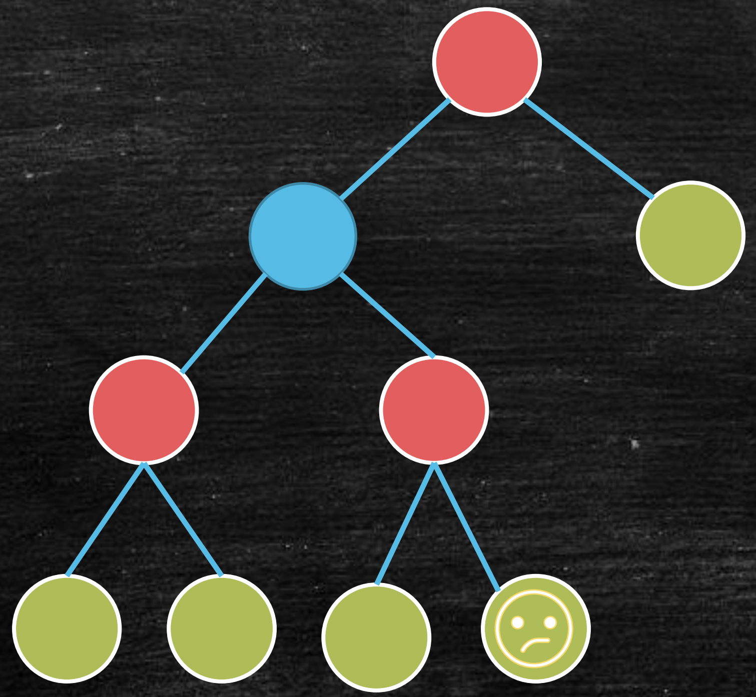
Greedy Algorithm Implementation



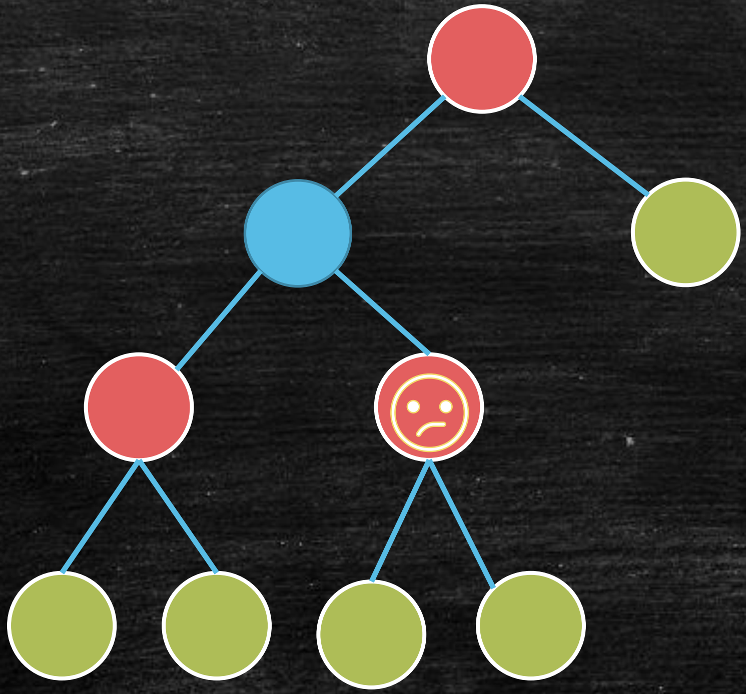
Greedy Algorithm Implementation



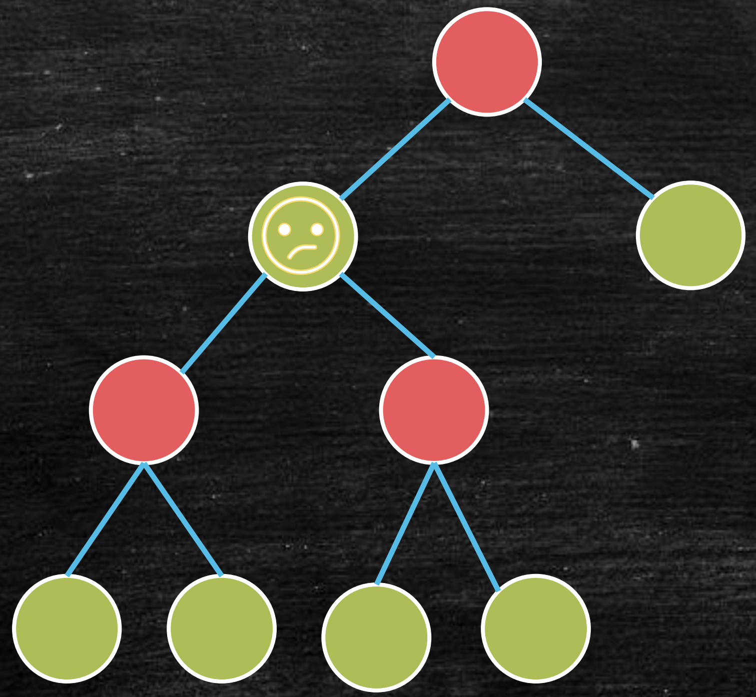
Greedy Algorithm Implementation



Greedy Algorithm Implementation

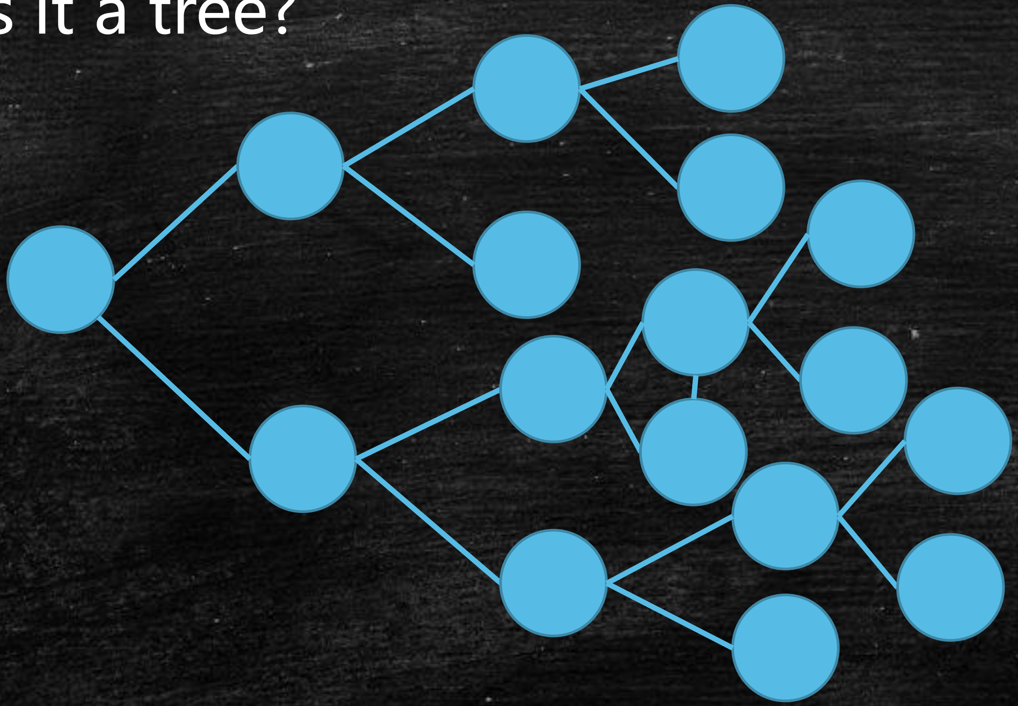


Greedy Algorithm Implementation



TSP on General Graphs

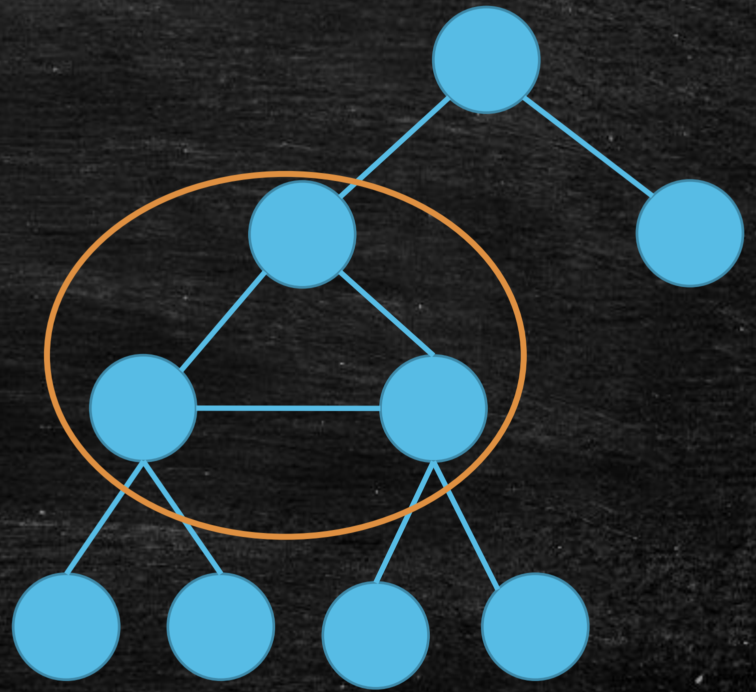
- Look at the following graph.
- Is it a tree?



It looks like a tree!

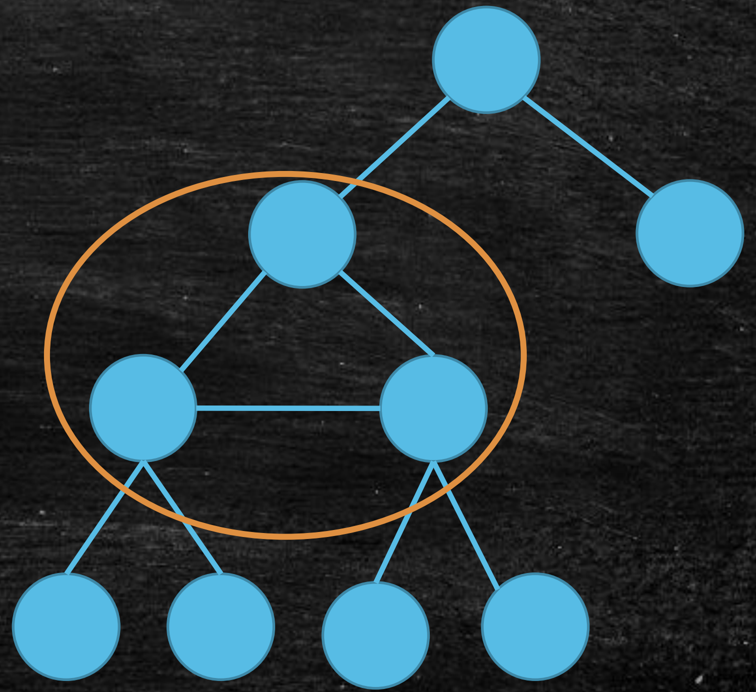
A Tree-like Graph

- It is a tree is we view the triangle as a **super node**.
- Question: Can we still use DP?
- Consider the subtree rooted at the **orange super node**.
  - Case 1: We choose the **super node**, what happens?
  - Case 2: We do not choose the **super node**, what happens?



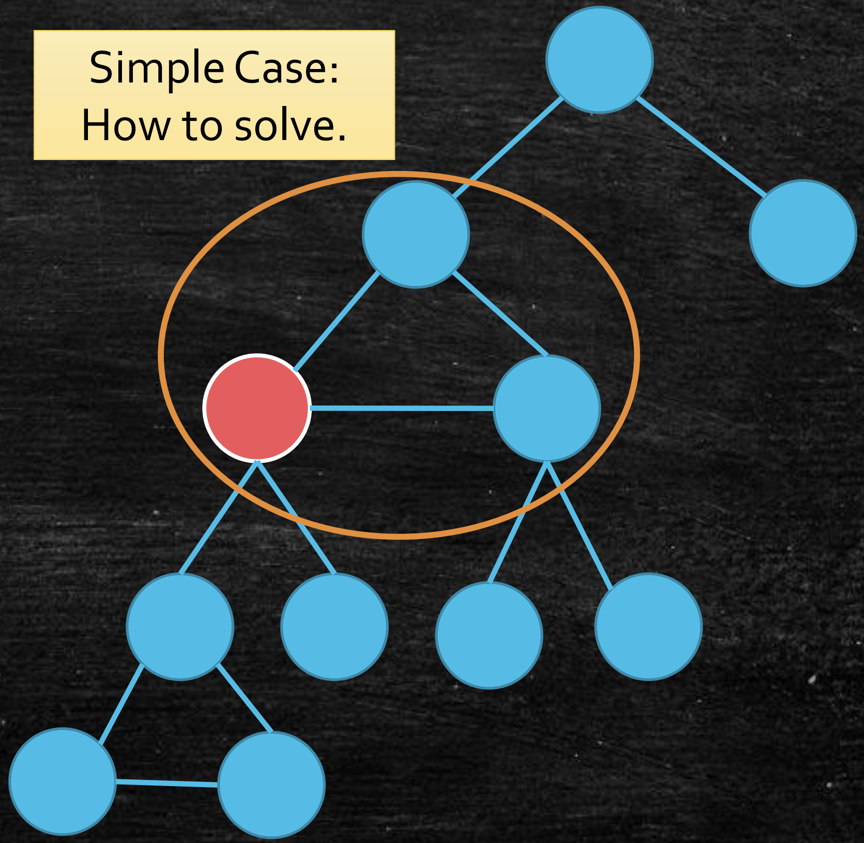
"Choose" a Super Node.

- "Choose" a Super Node.
- How many cases?
- We have at most 2^3 possible way!
- Different way means different restriction for next level selection.



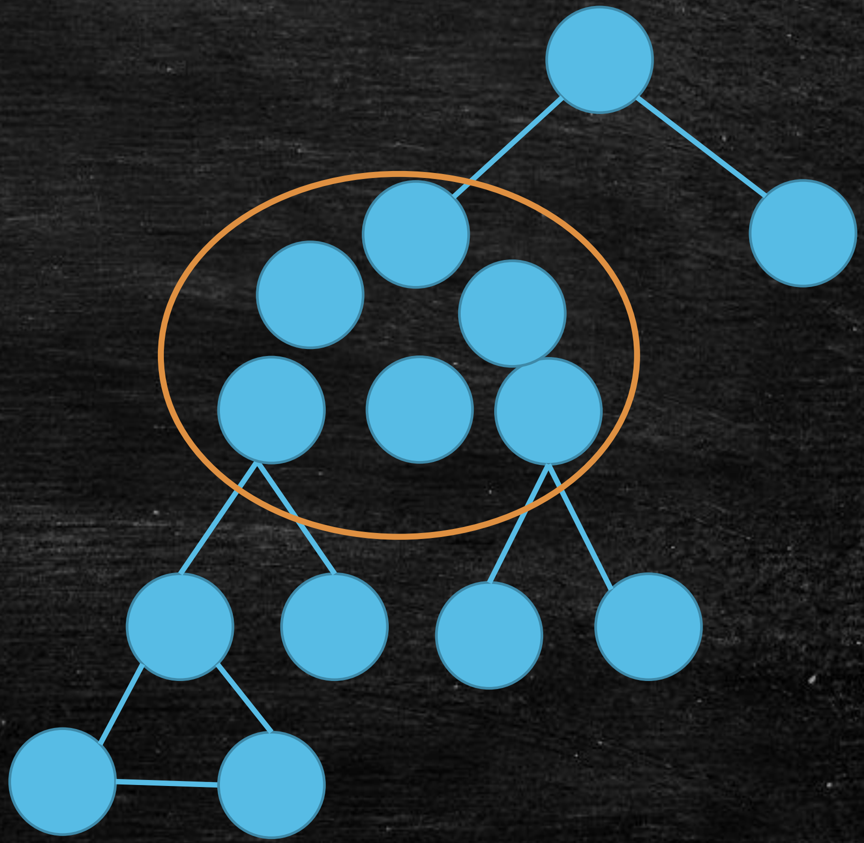
“Choose” a Super Node.

- “Choose” a Super Node.
- How many cases?
- We have at most 2^3 possible way!
- Different way means different restriction for next level selection.
- We can design a DP for $f[i, way]$.
- Time: $O(2^{3\times 3} \cdot n)$



"Choose" a Super Node.

- What if super node has k vertices?
- We have at most 2^k possible way!
- Time: $O\left(2^{O(k)} \cdot n\right) \rightarrow O(2^k \cdot n)$
- It hold when the largest super node has k vertices!



Tree-width (Idea)

- The best way to make a graph to tree-like!
- Best: minimize the number of vertices (k) in the largest super node.
- Tree-width: $k-1$
  - Cycle: 2
  - Clique: n-1
  - Tree: 1 (special)
  - series-parallel graphs: ≤ 2
- Many Optimization Problem in these graphs can use tree DP to get $O\left(2^{O(k)} \cdot poly(n)\right)$.

Fixed-Parameter Tractable

- $O(f(k) \cdot n^c)$
- $f(k)$ do not need to be polynomial.
- Many Optimization Problem in these graphs can use tree DP to get $O\left(2^{O(k)} \cdot poly(n)\right)$!
- They are FPT in terms of the treewidth!
- Compare to Approximation Algorithms!