Overview

logistics:
pset1 out, due R5 — can submit in groups of \( \leq 3 \)

last lecture:
- recursion
- memoization
- dynamic programming
- fibonacci numbers
- edit distance
- knapsack

today:
- dynamic programming on trees
- maximum independent set
- dominating set
logistics:
logistics:

- pset1 out,
Overview

logistics:
- pset1 out, due R5
Overview

**logistics:**
- pset1 out, due R5 — can submit in *groups* of $\leq 3$
Overview

logistics:
- pset1 out, due R5 — can submit in groups of $\leq 3$

last lecture:
Overview

logistics:
- pset1 out, due R5 — can submit in *groups* of ≤ 3

last lecture:
- recursion
Overview

**logistics:**
- pset1 out, due R5 — can submit in *groups* of $\leq 3$

**last lecture:**
- recursion
- memoization
Overview

logistics:
- pset1 out, due R5 — can submit in groups of ≤ 3

last lecture:
- recursion
- memoization
- dynamic programming
Overview

logistics:
- pset1 out, due R5 — can submit in groups of $\leq 3$

last lecture:
- recursion
- memoization
- dynamic programming
  - fibonacci numbers
Overview

logistics:
- pset1 out, due R5 — can submit in *groups* of $\leq 3$

last lecture:
- recursion
- memoization
- dynamic programming
  - fibonacci numbers
  - edit distance
Overview

**logistics:**
- pset1 out, due R5 — can submit in *groups* of ≤ 3

**last lecture:**
- recursion
- memoization
- dynamic programming
  - fibonacci numbers
  - edit distance
  - knapsack
Overview

**logistics:**
- pset1 out, due R5 — can submit in *groups* of ≤ 3

**last lecture:**
- recursion
- memoization
- dynamic programming
  - fibonacci numbers
  - edit distance
  - knapsack

**today:**
Overview

**logistics:**
- pset1 out, due R5 — can submit in *groups* of \( \leq 3 \)

**last lecture:**
- recursion
- memoization
- dynamic programming
  - fibonacci numbers
  - edit distance
  - knapsack

**today:**
- dynamic programming
Overview

**logistics:**
- pset1 out, due R5 — can submit in *groups* of $\leq 3$

**last lecture:**
- recursion
- memoization
- dynamic programming
  - fibonacci numbers
  - edit distance
  - knapsack

**today:**
- dynamic programming *on trees*
Overview

**logistics:**
- pset1 out, due R5 — can submit in *groups* of $\leq 3$

**last lecture:**
- recursion
- memoization
- dynamic programming
  - fibonacci numbers
  - edit distance
  - knapsack

**today:**
- dynamic programming *on trees*
- maximum independent set
Overview

**logistics:**
- pset1 out, due R5 — can submit in *groups* of \( \leq 3 \)

**last lecture:**
- recursion
- memoization
- dynamic programming
  - fibonacci numbers
  - edit distance
  - knapsack

**today:**
- dynamic programming *on trees*
- maximum independent set
- dominating set
Dynamic Programming

- develop recursive algorithm
- understand structure of subproblems
- names of subproblems
- number of subproblems
- dependency graph amongst subproblems
- memoize (implicitly, or explicitly)
- analysis (time, space)
- further optimization

Remarks:
- memoizing a recursive algorithm does not necessarily lead to an efficient algorithm (e.g., knapsack problem)
- recognizing that dynamic programming applies to a problem can be non-obvious
Dynamic Programming

dynamic programming:

dynamic programming:
- develop recursive algorithm
Dynamic Programming

dynamic programming:

- develop recursive algorithm
- understand structure of subproblems

Remarks: memoizing a recursive algorithm does not necessarily lead to an efficient algorithm (e.g., knapsack problem) — you need the right recursion. Recognizing that dynamic programming applies to a problem can be non-obvious.
dynamic programming:
- develop recursive algorithm
- understand structure of subproblems
  - names of subproblems
dynamic programming:
- develop recursive algorithm
- understand structure of subproblems
  - names of subproblems
  - number of subproblems
Dynamic Programming

dynamic programming:
- develop recursive algorithm
- understand structure of subproblems
  - names of subproblems
  - number of subproblems
  - dependency graph amongst subproblems

analysis (time, space)

further optimization

remarks: memoizing a recursive algorithm does not necessarily lead to an efficient algorithm (e.g., knapsack problem) — you need the right recursion.

recognizing that dynamic programming applies to a problem can be non-obvious.
Dynamic Programming

dynamic programming:

- develop recursive algorithm
- understand structure of subproblems
  - names of subproblems
  - number of subproblems
  - dependency graph amongst subproblems
- memoize

Further optimization

Remarks:

Memoizing a recursive algorithm does not necessarily lead to an efficient algorithm (e.g., knapsack problem).

Recognizing that dynamic programming applies to a problem can be non-obvious.
**Dynamic Programming**

**dynamic programming:**

- develop recursive algorithm
- understand structure of subproblems
  - names of subproblems
  - number of subproblems
  - dependency graph amongst subproblems
- memoize (implicitly,
Dynamic Programming

dynamic programming:

- develop recursive algorithm
- understand structure of subproblems
  - names of subproblems
  - number of subproblems
  - dependency graph amongst subproblems
- memoize (implicitly, or explicitly)
Dynamic Programming

dynamic programming:
- develop recursive algorithm
- understand structure of subproblems
  - names of subproblems
  - number of subproblems
  - dependency graph amongst subproblems
- memoize (implicitly, or explicitly)
- analysis

Remarks: memoizing a recursive algorithm does not necessarily lead to an efficient algorithm (e.g., knapsack problem). Recognizing that dynamic programming applies to a problem can be non-obvious.
Dynamic Programming

**dynamic programming:**
- develop recursive algorithm
- understand structure of subproblems
  - names of subproblems
  - number of subproblems
  - dependency graph amongst subproblems
- memoize (implicitly, or explicitly)
- analysis (time,
**Dynamic Programming**

**dynamic programming:**
- develop recursive algorithm
- understand structure of subproblems
  - names of subproblems
  - number of subproblems
  - dependency graph amongst subproblems
- memoize (implicitly, or explicitly)
- analysis (time, space)

Remarks:
- Memoizing a recursive algorithm does not necessarily lead to an efficient algorithm (e.g., knapsack problem).
- Recognizing that dynamic programming applies to a problem can be non-obvious.
Dynamic Programming

dynamic programming:
- develop recursive algorithm
- understand structure of subproblems
  - names of subproblems
  - number of subproblems
  - dependency graph amongst subproblems
- memoize (implicitly, or explicitly)
- analysis (time, space)
- further optimization

remarks:
- memoizing a recursive algorithm does not necessarily lead to an efficient algorithm (e.g., knapsack problem)
- recognizing that dynamic programming applies to a problem can be non-obvious
Dynamic Programming

dynamic programming:
- develop recursive algorithm
- understand structure of subproblems
  - names of subproblems
  - number of subproblems
  - dependency graph amongst subproblems
- memoize (implicitly, or explicitly)
- analysis (time, space)
- further optimization

remarks:
Dynamic Programming

**dynamic programming:**
- develop recursive algorithm
- understand structure of subproblems
  - names of subproblems
  - number of subproblems
  - dependency graph amongst subproblems
- memoize (implicitly, or explicitly)
- analysis (time, space)
- further optimization

**remarks:**
- memoizing a recursive algorithm does not necessarily lead to an efficient algorithm
Dynamic Programming

dynamic programming:

- develop recursive algorithm
- understand structure of subproblems
  - names of subproblems
  - number of subproblems
  - dependency graph amongst subproblems
- memoize (implicitly, or explicitly)
- analysis (time, space)
- further optimization

remarks:

- memoizing a recursive algorithm does not necessarily lead to an efficient algorithm (e.g., knapsack problem)
Dynamic Programming

dynamic programming:
- develop recursive algorithm
- understand structure of subproblems
  - names of subproblems
  - number of subproblems
  - dependency graph amongst subproblems
- memoize (implicitly, or explicitly)
- analysis (time, space)
- further optimization

remarks:
- memoizing a recursive algorithm does not necessarily lead to an efficient algorithm (e.g., knapsack problem) — you need the right recursion
dynamic programming:

- develop recursive algorithm
- understand structure of subproblems
  - names of subproblems
  - number of subproblems
  - dependency graph amongst subproblems
- memoize (implicitly, or explicitly)
- analysis (time, space)
- further optimization

remarks:

- memoizing a recursive algorithm does not necessarily lead to an efficient algorithm (e.g., knapsack problem) — you need the right recursion
- recognizing that dynamic programming applies to a problem can be non-obvious
Fact: many computational problems ask to optimize an objective over a graph. Many graph optimization problems are $\text{NP}$-hard, yet many $\text{NP}$-hard graph optimization problems can be efficiently solved when the graph is a tree.

Remarks: dynamic programming over graphs often relies on decomposing the graph into subgraphs, but there are many subgraphs and they relate to each other in complicated ways. Trees can be easily decomposed into sub-trees, which are easily related to each other. $\implies$ Trees are amenable to divide and conquer, and dynamic programming more generally. Dynamic programming on trees often generalizes to graphs that have low treewidth.
fact:
fact:

- many computational problems ask to optimize an objective over a graph
fact:
- many computational problems ask to optimize an objective over a graph
- many graph optimization problems are NP-hard
**fact:**
- many computational problems ask to optimize an objective over a graph
- many graph optimization problems are NP-hard
- *yet:*

trees can be easily decomposed into sub-trees, which are easily related to each other.
fact:

- many computational problems ask to optimize an objective over a graph
- many graph optimization problems are NP-hard
- yet: many NP-hard graph optimization problems can be efficiently solved
fact:

- many computational problems ask to optimize an objective over a graph
- many graph optimization problems are NP-hard
- yet: many NP-hard graph optimization problems can be efficiently solved when the graph is a tree
**fact:**
- many computational problems ask to optimize an objective over a graph
- many graph optimization problems are NP-hard
- *yet:* many NP-hard graph optimization problems can be efficiently solved when the graph is a *tree*

**remarks:**
fact:
- many computational problems ask to optimize an objective over a graph
- many graph optimization problems are NP-hard
- yet: many NP-hard graph optimization problems can be efficiently solved when the graph is a tree

remarks:
- dynamic programming over graphs
fact:
- many computational problems ask to optimize an objective over a graph
- many graph optimization problems are NP-hard
- *yet*: many NP-hard graph optimization problems can be efficiently solved when the graph is a *tree*

remarks:
- dynamic programming over graphs often relies on decomposing the graph into subgraphs,
fact:
- many computational problems ask to optimize an objective over a graph
- many graph optimization problems are NP-hard
- *yet:* many NP-hard graph optimization problems can be efficiently solved when the graph is a *tree*

remarks:
- dynamic programming over graphs often relies on decomposing the graph into subgraphs, but there are many subgraphs
fact:
- many computational problems ask to optimize an objective over a graph
- many graph optimization problems are NP-hard
- yet: many NP-hard graph optimization problems can be efficiently solved when the graph is a tree

remarks:
- dynamic programming over graphs often relies on decomposing the graph into subgraphs, but there are many subgraphs and they relate to each other in complicated ways
fact:
- many computational problems ask to optimize an objective over a graph
- many graph optimization problems are NP-hard
- *yet*: many NP-hard graph optimization problems can be efficiently solved when the graph is a *tree*

remarks:
- dynamic programming over graphs often relies on decomposing the graph into subgraphs, but there are many subgraphs and they relate to each other in complicated ways
- trees can be easily decomposed into sub-trees,
Trees

**fact:**

- many computational problems ask to optimize an objective over a graph
- many graph optimization problems are NP-hard
- *yet:* many NP-hard graph optimization problems can be efficiently solved when the graph is a *tree*

**remarks:**

- dynamic programming over graphs often relies on decomposing the graph into subgraphs, but there are many subgraphs and they relate to each other in complicated ways
- trees can be easily decomposed into sub-trees, which are easily related to each other
Trees

**fact:**
- many computational problems ask to optimize an objective over a graph
- many graph optimization problems are NP-hard
- *yet:* many NP-hard graph optimization problems can be efficiently solved when the graph is a *tree*

**remarks:**
- dynamic programming over graphs often relies on decomposing the graph into subgraphs, but there are many subgraphs and they relate to each other in complicated ways
- trees can be easily decomposed into sub-trees, which are easily related to each other $\implies$ trees are amenable to divide and conquer,
fact:
- many computational problems ask to optimize an objective over a graph
- many graph optimization problems are NP-hard
- yet: many NP-hard graph optimization problems can be efficiently solved when the graph is a *tree*

remarks:
- dynamic programming over graphs often relies on decomposing the graph into subgraphs, but there are many subgraphs and they relate to each other in complicated ways
- trees can be easily decomposed into sub-trees, which are easily related to each other $\implies$ trees are amenable to divide and conquer, and dynamic programming more generally
fact:
- many computational problems ask to optimize an objective over a graph
- many graph optimization problems are NP-hard
- yet: many NP-hard graph optimization problems can be efficiently solved when the graph is a tree

remarks:
- dynamic programming over graphs often relies on decomposing the graph into subgraphs, but there are many subgraphs and they relate to each other in complicated ways
- trees can be easily decomposed into sub-trees, which are easily related to each other \(\implies\) trees are amenable to divide and conquer, and dynamic programming more generally
- dynamic programming on trees often generalizes to graphs that have low treewidth
Maximum Independent Set

Definition

Let $G = (V, E)$ be an undirected (simple) graph. An independent set of $G$ is a subset $S \subseteq V$ such that there are no edges in $G$ between vertices in $S$. That is, for all $u, v \in S$ that $(u, v) \not\in E$.

Example:

\[
\begin{array}{cccc}
  a & b & c & d & e & f \\
\end{array}
\]

Independent sets include $\emptyset$, \{a, c\}, and \{b, e, f\}.
### Definition

Let $G = (V, E)$ be an undirected (simple) graph. An independent set of $G$ is a subset $S \subseteq V$ such that there are no edges in $G$ between vertices in $S$. That is, for all $u, v \in S$ that $(u, v) \not\in E$.

Examples include $\emptyset$, \{a, c\}, and \{b, e, f\}.
Definition

Let $G = (V, E)$ be an undirected (simple) graph.
Maximum Independent Set

**Definition**

Let $G = (V, E)$ be an undirected (simple) graph. An **independent set of** $G$ is a subset $S \subseteq V$ such that there are no edges in $G$ between vertices in $S$. That is, for all $u, v \in S$ that $(u, v) \not\in E$.

Examples:
- $\emptyset$
- $\{a, c\}$
- $\{b, e, f\}$
Maximum Independent Set

Definition

Let $G = (V, E)$ be an undirected (simple) graph. An **independent set** of $G$ is a subset $S \subseteq V$.
Definition

Let $G = (V, E)$ be an undirected (simple) graph. An **independent set of** $G$ is a subset $S \subseteq V$ such that there are no edges in $G$ between vertices in $S$. 

ex: 

```
 a  b  c  d  e  f
```

Independent sets include $\emptyset$, \{a, c\}, and \{b, e, f\}. 


Definition

Let $G = (V, E)$ be an undirected (simple) graph. An **independent set of** $G$ is a subset $S \subseteq V$ such that there are no edges in $G$ between vertices in $S$. That is, for all $u, v \in S$.
Definition

Let $G = (V, E)$ be an undirected (simple) graph. An **independent set of** $G$ is a subset $S \subseteq V$ such that there are no edges in $G$ between vertices in $S$. That is, for all $u, v \in S$ that $(u, v) \notin E$. 

Ex: 

```
a b
c d
e f
```
Maximum Independent Set

Definition

Let $G = (V, E)$ be an undirected (simple) graph. An **independent set of** $G$ is a subset $S \subseteq V$ such that there are no edges in $G$ between vertices in $S$. That is, for all $u, v \in S$ that $(u, v) \notin E$.

**ex:**
Maximum Independent Set

Definition

Let $G = (V, E)$ be an undirected (simple) graph. An **independent set of** $G$ is a subset $S \subseteq V$ such that there are no edges in $G$ between vertices in $S$. That is, for all $u, v \in S$ that $(u, v) \notin E$.

ex:
Maximum Independent Set

**Definition**

Let $G = (V, E)$ be an undirected (simple) graph. An **independent set** of $G$ is a subset $S \subseteq V$ such that there are no edges in $G$ between vertices in $S$. That is, for all $u, v \in S$ that $(u, v) \notin E$.

**ex:**

![Graph Example](image)

Independent sets include $\emptyset$,
Definition

Let $G = (V, E)$ be an undirected (simple) graph. An independent set of $G$ is a subset $S \subseteq V$ such that there are no edges in $G$ between vertices in $S$. That is, for all $u, v \in S$ that $(u, v) \notin E$.

ex:

Independent sets include $\emptyset$, $\{a, c\}$,
Maximum Independent Set

Definition

Let $G = (V, E)$ be an undirected (simple) graph. An **independent set of** $G$ is a subset $S \subseteq V$ such that there are no edges in $G$ between vertices in $S$. That is, for all $u, v \in S$ that $(u, v) \notin E$.

**ex:**

![Diagram of an undirected graph with vertices a, b, c, d, e, f and edges connecting a to b, b to c, c to d, d to e, e to f, and f to a.]

Independent sets include $\emptyset$, $\{a, c\}$, and $\{b, e, f\}$.
Maximum Independent Set (II)

**Definition**

The maximum independent set (MIS) problem is to, given a undirected (simple) graph $G = (V, E)$, output the size of the largest independent set in $G$. That is, output $\alpha(G) := \max_{S \subseteq V, S \text{ independent set of } G} |S|$. 

ex:

```
    a   b   c   d   e   f
```

$\alpha(G) = 3$
The maximum independent set (MIS) problem is to, given an undirected (simple) graph $G = (V, E)$, output the size of the largest independent set in $G$. That is, output $\alpha(G) := \max \{ |S| \mid S \subseteq V, S \text{ independent set of } G \}$. 

Example:

```
a b c d e f
```

$\alpha(G) = 3$
Maximum Independent Set (II)

Definition

The **maximum independent set (MIS)** problem is to,
Maximum Independent Set (II)

Definition

The **maximum independent set (MIS)** problem is to, given a undirected (simple) graph \( G = (V, E) \), output the size of the largest independent set in \( G \).

That is, output \( \alpha(G) := \max \{ |S| \mid S \subseteq V, S \text{ independent set of } G \} \).
Maximum Independent Set (II)

Definition

The **maximum independent set (MIS)** problem is to, given a undirected (simple) graph $G = (V, E)$ output the size of the largest independent set in $G$. 
The **maximum independent set (MIS)** problem is to, given a undirected (simple) graph $G = (V, E)$ output the size of the largest independent set in $G$. That is, output

$$\alpha(G) := \max_{S \subseteq V, S \text{ independent set of } G} |S|.$$
Definition

The **maximum independent set (MIS)** problem is to, given a undirected (simple) graph $G = (V,E)$ output the size of the largest independent set in $G$. That is, output

$$\alpha(G) := \max_{S \subseteq V, S \text{ independent set of } G} |S|.$$
The **maximum independent set (MIS)** problem is to, given a undirected (simple) graph \( G = (V, E) \) output the size of the largest independent set in \( G \). That is, output

\[
\alpha(G) := \max_{S \subseteq V, S \text{ independent set of } G} |S|.
\]

**ex:**
The **maximum independent set (MIS)** problem is to, given a undirected (simple) graph $G = (V, E)$ output the size of the largest independent set in $G$. That is, output

$$\alpha(G) := \max_{S \subseteq V, S \text{ independent set of } G} |S|.$$
The maximum independent set (MIS) problem is to, given a undirected (simple) graph \( G = (V, E) \) output the size of the largest independent set in \( G \). That is, output

\[
\alpha(G) := \max_{S \subseteq V, S \text{ independent set of } G} |S|.
\]

**ex:**

\[
\alpha(G) = 3
\]
Definition

The **maximum independent set (MIS)** problem is to, given a undirected (simple) graph $G = (V, E)$ output the size of the largest independent set in $G$. That is, output

$$\alpha(G) := \max_{S \subseteq V, S \text{ independent set of } G} |S|.$$  

**ex:**

$$\alpha(G) = 3$$
The maximum weight independent set problem is to, given a undirected (simple) graph $G = (V, E)$ and a weight function $w: V \rightarrow \mathbb{N}$, output the weight of the maximum weight independent set in $G$. That is, output $\max_{S \subseteq V, S \text{ independent set of } G} \sum_{v \in S} w(v)$. 
The maximum weight independent set problem is to, given a undirected (simple) graph $G = (V, E)$ and a weight function $w: V \rightarrow \mathbb{N}$, output the weight of the maximum weight independent set in $G$. That is, output $\max_{S \subseteq V} \sum_{v \in S} w(v)$. 
The **maximum weight independent set** problem is to,
The **maximum weight independent set** problem is to, given a undirected (simple) graph \( G = (V, E) \)
Maximum Independent Set (III)

Definition

The **maximum weight independent set** problem is to, given a undirected (simple) graph $G = (V, E)$ and a weight function $w : V \rightarrow \mathbb{N}$, output the weight of the maximum weight independent set in $G$.

$$\sum_{v \in S} w(v).$$
Definition

The **maximum weight independent set** problem is to, given a undirected (simple) graph \( G = (V, E) \) and a weight function \( w : V \rightarrow \mathbb{N} \), output the weight of the maximum weight independent set in \( G \).
Maximum Independent Set (III)

Definition

The **maximum weight independent set** problem is to, given a undirected (simple) graph $G = (V, E)$ and a weight function $w : V \rightarrow \mathbb{N}$, output the weight of the maximum weight independent set in $G$. That is, output

$$\max_{S \subseteq V} \sum_{v \in S} w(v).$$

$S$ independent set of $G$
Maximum Independent Set (III)

Definition

The **maximum weight independent set** problem is to, given a undirected (simple) graph $G = (V, E)$ and a weight function $w : V \rightarrow \mathbb{N}$, output the weight of the maximum weight independent set in $G$. That is, output

$$\max_{S \subseteq V} \sum_{v \in S} w(v).$$

$S$ independent set of $G$.
Maximum Independent Set (III)

Definition

The **maximum weight independent set** problem is to, given a undirected (simple) graph $G = (V, E)$ and a weight function $w : V \rightarrow \mathbb{N}$, output the weight of the maximum weight independent set in $G$. That is, output

$$\max_{S \subseteq V} \sum_{v \in S} w(v).$$

$S$ independent set of $G$.
Maximum Independent Set (IV)

Remarks:

- Maximum (weight) independent set (MIS) is solvable via brute force: try all possible subsets ⇒ solvable in time $O(n^{O(1)}2^n)$
- No efficient algorithm currently known
- MIS is $\text{NP}$-hard ⇒ an efficient algorithm not expected to exist
- MIS is efficiently solvable if the underlying graph is a tree
Maximum Independent Set (IV)

**remarks:**

- The maximum (weight) independent set (MIS) is solvable via brute force: try all possible subsets $\Rightarrow$ solvable in time $O(n^{O(1)^2})$.
- No efficient algorithm currently known.
- MIS is $\text{NP}$-hard $\Rightarrow$ an efficient algorithm not expected to exist.
- MIS is efficiently solvable if the underlying graph is a tree.
Maximum Independent Set (IV)

remarks:

- maximum (weight) independent set (MIS) is solvable via brute force:
maximum (weight) independent set (MIS) is solvable via brute force: try all possible subsets
Maximum Independent Set (IV)

remarks:

- maximum (weight) independent set (MIS) is solvable via brute force: try all possible subsets $\implies$ solvable in time $O(n^{O(1)}2^n)$
Maximum Independent Set (IV)

**remarks:**

- maximum (weight) independent set (MIS) is solvable via brute force: try *all* possible subsets $\implies$ solvable in time $O(n^{O(1)}2^n)$
- no efficient algorithm *currently* known
Maximum Independent Set (IV)

**remarks:**

- maximum (weight) independent set (MIS) is solvable via brute force: try *all* possible subsets $\Rightarrow$ solvable in time $O(n^{O(1)}2^n)$
- no efficient algorithm *currently* known
- MIS is NP-hard
Maximum Independent Set (IV)

**remarks:**

- maximum (weight) independent set (MIS) is solvable via brute force: try *all* possible subsets $\implies$ solvable in time $O(n^{O(1)}2^n)$
- no efficient algorithm *currently* known
- MIS is NP-hard $\implies$ an efficient algorithm *not* expected to exist
Maximum Independent Set (IV)

**remarks:**

- maximum (weight) independent set (MIS) is solvable via brute force: try *all* possible subsets $\Rightarrow$ solvable in time $O(n^{O(1)}2^n)$
- no efficient algorithm *currently* known
- MIS is NP-hard $\Rightarrow$ an efficient algorithm *not* expected to exist
- MIS is efficiently solvable if the underlying graph is a *tree*
Maximum Independent Set (V)

For vertex \( v \), let \( N(v) \) denote the subset \( S \subseteq V \) of neighbors of \( v \).

**Lemma**

\[ G = (V, E), w : V \to N, \]

with \( |V| \geq 1 \).

Then for any \( v \in V \),

\[ \text{MIS}(G) = \max \{ \text{MIS}(G - v), \text{MIS}(G - v - N(v)) + w(v) \} \]

**Proof.**

For any set \( S \) independent in \( G \), either \( v \not\in S \) or \( v \in S \).

\( G - v \): any set \( T \subseteq V \setminus \{ v \} \) independent in \( G - v \) has \( T \subseteq V \) independent in \( G \).

\( G - v - N(v) \): any set \( T \subseteq V \setminus (\{ v \} \cup N(v)) \) independent in \( G - v - N(v) \) has \( T \cup \{ v \} \subseteq V \) independent in \( G \).

Any set \( S \) independent in \( G \) must be of the above two cases.

Now maximize.
For vertex \( v \),
For vertex $v$, let $N(v)$ denote the subset $S \subseteq V$ of neighbors of $v$. 
Maximum Independent Set (V)

For vertex $v$, let $N(v)$ denote the subset $S \subseteq V$ of neighbors of $v$.

Lemma
Maximum Independent Set (V)

For vertex $v$, let $N(v)$ denote the subset $S \subseteq V$ of neighbors of $v$.

Lemma

$G = (V, E)$. 
For vertex $v$, let $N(v)$ denote the subset $S \subseteq V$ of neighbors of $v$.

Lemma

$G = (V, E), w : V \rightarrow \mathbb{N},$
Maximum Independent Set (V)

For vertex $v$, let $N(v)$ denote the subset $S \subseteq V$ of neighbors of $v$.

Lemma

$G = (V, E), \ w : V \rightarrow \mathbb{N}, \ with \ |V| \geq 1.$
For vertex $v$, let $N(v)$ denote the subset $S \subseteq V$ of neighbors of $v$.

**Lemma**

$G = (V, E), w : V \rightarrow \mathbb{N}, \text{ with } |V| \geq 1$. Then for any $v \in V$,

$$\text{MIS}(G) =$$
For vertex $v$, let $N(v)$ denote the subset $S \subseteq V$ of neighbors of $v$.

**Lemma**

$G = (V, E), w : V \rightarrow \mathbb{N},$ with $|V| \geq 1$. Then for any $v \in V$,

$$\text{MIS}(G) = \max \left\{ \right.$$

9 / 29
Maximum Independent Set (V)

For vertex $v$, let $N(v)$ denote the subset $S \subseteq V$ of neighbors of $v$.

**Lemma**

$G = (V, E)$, $w : V \to \mathbb{N}$, with $|V| \geq 1$. Then for any $v \in V$,

$$\text{MIS}(G) = \max \left\{ \text{MIS}(G - v), \text{MIS}(G - v - N(v)) + w(v) \right\}$$
Maximum Independent Set (V)

For vertex $v$, let $N(v)$ denote the subset $S \subseteq V$ of neighbors of $v$.

**Lemma**

$G = (V, E), w : V \rightarrow \mathbb{N},$ with $|V| \geq 1$. Then for any $v \in V$,

$$\text{MIS}(G) = \max \left\{ \text{MIS}(G - v), \text{MIS}(G - v - N(v)) + w(v) \right\}.$$
Maximum Independent Set (V)

For vertex $v$, let $N(v)$ denote the subset $S \subseteq V$ of neighbors of $v$.

Lemma

$G = (V, E), w : V \rightarrow \mathbb{N}$, with $|V| \geq 1$. Then for any $v \in V$,

$$\text{MIS}(G) = \max \left\{ \text{MIS}(G - v), \text{MIS}(G - v - N(v)) + w(v) \right\}.$$ 

Proof.
Maximum Independent Set (V)

For vertex $v$, let $N(v)$ denote the subset $S \subseteq V$ of *neighbors* of $v$.

**Lemma**

$G = (V, E)$, $w : V \rightarrow \mathbb{N}$, with $|V| \geq 1$. Then for any $v \in V$,

$$\text{MIS}(G) = \max \left\{ \text{MIS}(G - v), \text{MIS}(G - v - N(v)) + w(v) \right\}.$$  

**Proof.**

For any set $S$ independent in $G$, 


For vertex $v$, let $N(v)$ denote the subset $S \subseteq V$ of neighbors of $v$.

**Lemma**

$G = (V, E), w : V \to \mathbb{N}$, with $|V| \geq 1$. Then for any $v \in V$,

$$\text{MIS}(G) = \max \left\{ \text{MIS}(G - v), \text{MIS}(G - v - N(v)) + w(v) \right\}.$$

**Proof.**

For any set $S$ independent in $G$, either $v \not\in S$
Maximum Independent Set (V)

For vertex $v$, let $N(v)$ denote the subset $S \subseteq V$ of neighbors of $v$.

**Lemma**

$G = (V, E), w : V \rightarrow \mathbb{N}$, with $|V| \geq 1$. Then for any $v \in V$,

$$\text{MIS}(G) = \max \left\{ \text{MIS}(G - v), \text{MIS}(G - v - N(v)) + w(v) \right\}.$$  

**Proof.**

For any set $S$ independent in $G$, either $v \notin S$ or $v \in S$.  

Maximum Independent Set (V)

For vertex $v$, let $N(v)$ denote the subset $S \subseteq V$ of neighbors of $v$.

**Lemma**

$G = (V, E), w : V \rightarrow \mathbb{N},$ with $|V| \geq 1$. Then for any $v \in V$,

$$\text{MIS}(G) = \max \left\{ \text{MIS}(G - v), \text{MIS}(G - v - N(v)) + w(v) \right\}.$$

**Proof.**

For any set $S$ independent in $G$, either $v \notin S$ or $v \in S$.

- $G - v$: 
  

For vertex $v$, let $N(v)$ denote the subset $S \subseteq V$ of neighbors of $v$.

**Lemma**

$G = (V, E)$, $w : V \rightarrow \mathbb{N}$, with $|V| \geq 1$. Then for any $v \in V$,

$$\text{MIS}(G) = \max \left\{ \text{MIS}(G - v), \text{MIS}(G - v - N(v)) + w(v) \right\}.$$  

**Proof.**

For any set $S$ independent in $G$, either $v \notin S$ or $v \in S$.

- $G - v$: any set $T \subseteq V \setminus \{v\}$ independent in $G - v$
Maximum Independent Set (V)

For vertex $v$, let $N(v)$ denote the subset $S \subseteq V$ of neighbors of $v$.

**Lemma**

$G = (V, E), w : V \rightarrow \mathbb{N}$, with $|V| \geq 1$. Then for any $v \in V$,

$$\text{MIS}(G) = \max \left\{ \text{MIS}(G - v), \text{MIS}(G - v - N(v)) + w(v) \right\}.$$  

**Proof.**

For any set $S$ independent in $G$, either $v \notin S$ or $v \in S$.

- $G - v$: any set $T \subseteq V \setminus \{v\}$ independent in $G - v$ has $T \subseteq V$ independent in $G$
Maximum Independent Set (V)

For vertex \( v \), let \( N(v) \) denote the subset \( S \subseteq V \) of neighbors of \( v \).

Lemma

\[ G = (V, E), \ w : V \to \mathbb{N}, \ with \ |V| \geq 1. \ Then \ for \ any \ v \in V, \]

\[ \text{MIS}(G) = \max \left\{ \text{MIS}(G - v), \ \text{MIS}(G - v - N(v)) + w(v) \right\}. \]

Proof.

For any set \( S \) independent in \( G \), either \( v \notin S \) or \( v \in S \).

- \( G - v \): any set \( T \subseteq V \setminus \{v\} \) independent in \( G - v \) has \( T \subseteq V \) independent in \( G \)
- \( G - v - N(v) \):

Maximum Independent Set (V)

For vertex $v$, let $N(v)$ denote the subset $S \subseteq V$ of neighbors of $v$.

**Lemma**

$G = (V, E)$, $w: V \to \mathbb{N}$, with $|V| \geq 1$. Then for any $v \in V$,

$$\text{MIS}(G) = \max \left\{ \text{MIS}(G - v), \text{MIS}(G - v - N(v)) + w(v) \right\}.$$

**Proof.**

For any set $S$ independent in $G$, either $v \notin S$ or $v \in S$.

- $G - v$: any set $T \subseteq V \setminus \{v\}$ independent in $G - v$ has $T \subseteq V$ independent in $G$
- $G - v - N(v)$: any set $T \subseteq V \setminus (\{v\} \cup N(v))$ independent in $G - v - N(v)$
Maximum Independent Set (V)

For vertex $v$, let $N(v)$ denote the subset $S \subseteq V$ of neighbors of $v$.

**Lemma**

$G = (V, E), w : V \rightarrow \mathbb{N}$, with $|V| \geq 1$. Then for any $v \in V$,

$$\text{MIS}(G) = \max \left\{ \text{MIS}(G - v), \text{MIS}(G - v - N(v)) + w(v) \right\}.$$ 

**Proof.**

For any set $S$ independent in $G$, either $v \notin S$ or $v \in S$.

- $G - v$: any set $T \subseteq V \setminus \{v\}$ independent in $G - v$ has $T \subseteq V$ independent in $G$
- $G - v - N(v)$: any set $T \subseteq V \setminus (\{v\} \cup N(v))$ independent in $G - v - N(v)$ has $T \cup \{v\} \subseteq V$ independent in $G$
Maximum Independent Set (V)

For vertex $v$, let $N(v)$ denote the subset $S \subseteq V$ of neighbors of $v$.

**Lemma**

$G = (V, E), w : V \to \mathbb{N}$, with $|V| \geq 1$. Then for any $v \in V$,

$$\text{MIS}(G) = \max \left\{ \text{MIS}(G - v), \text{MIS}(G - v - N(v)) + w(v) \right\}.$$ 

**Proof.**

For any set $S$ independent in $G$, either $v \not\in S$ or $v \in S$.

- $G - v$: any set $T \subseteq V \setminus \{v\}$ independent in $G - v$ has $T \subseteq V$ independent in $G$
- $G - v - N(v)$: any set $T \subseteq V \setminus (\{v\} \cup N(v))$ independent in $G - v - N(v)$ has $T \cup \{v\} \subseteq V$ independent in $G$

Any set $S$ independent in $G$ must be of the above two cases.
Maximum Independent Set (V)

For vertex \( v \), let \( N(v) \) denote the subset \( S \subseteq V \) of neighbors of \( v \).

**Lemma**

\[ G = (V, E), \ w : V \rightarrow \mathbb{N}, \text{with } |V| \geq 1. \text{ Then for any } v \in V, \]

\[ \text{MIS}(G) = \max \left\{ \text{MIS}(G - v), \text{MIS}(G - v - N(v)) + w(v) \right\}. \]

**Proof.**

For any set \( S \) independent in \( G \), either \( v \notin S \) or \( v \in S \).

- \( G - v \): any set \( T \subseteq V \setminus \{v\} \) independent in \( G - v \) has \( T \subseteq V \) independent in \( G \)
- \( G - v - N(v) \): any set \( T \subseteq V \setminus (\{v\} \cup N(v)) \) independent in \( G - v - N(v) \) has \( T \cup \{v\} \subseteq V \) independent in \( G \)

Any set \( S \) independent in \( G \) must be of the above two cases. Now maximize. \( \square \)
Maximum Independent Set (VI)

\[ \text{MIS}(G) = \max \{ \text{MIS}(G - v) \} \]

\[ \text{MIS}(G - v - \text{N}(v)) + w(v) \]
Maximum Independent Set (VI)

\[ \text{MIS}(G) = \max \begin{cases} \text{MIS}(G - v) \\ \text{MIS}(G - v - N(v)) + w(v) \end{cases} \]
Maximum Independent Set (VI)

\[ \text{MIS}(G) = \max \left\{ \begin{array}{ll}
\text{MIS}(G - \nu) \\
\text{MIS}(G - \nu - N(\nu)) + w(\nu)
\end{array} \right\} \]
Maximum Independent Set (VI)

\[ \text{MIS}(G) = \max \left\{ \begin{array}{l}
\text{MIS}(G - v) \\
\text{MIS}(G - v - N(v)) + w(v)
\end{array} \right\} \]
Maximum Independent Set (VI)

\[
\text{MIS}(G) = \max \left\{ \text{MIS}(G - v), \text{MIS}(G - v - N(v)) + w(v) \right\}
\]
Maximum Independent Set (VI)

$$\text{MIS}(G) = \max \begin{cases} 
\text{MIS}(G - v) \\
\text{MIS}(G - v - N(v)) + w(v) 
\end{cases}$$
Maximum Independent Set (VI)

\[ \text{MIS}(G) = \max \begin{cases} 
\text{MIS}(G - v) \\
\text{MIS}(G - v - N(v)) + w(v) 
\end{cases} \]
Maximum Independent Set (VI)

\[ \text{MIS}(G) = \max \left\{ \begin{array}{l} \text{MIS}(G - v) \\ \text{MIS}(G - v - N(v)) + w(v) \end{array} \right\} \]
Maximum Independent Set (VI)

\[ \text{MIS}(G) = \max \left\{ \begin{array}{l}
\text{MIS}(G - v) \\
\text{MIS}(G - v - N(v)) + w(v)
\end{array} \right\} \]
Maximum Independent Set (VI)

\[ \text{MIS}(G) = \max \left\{ \text{MIS}(G - v), \text{MIS}(G - v - N(v)) + w(v) \right\} \]
Maximum Independent Set (VI)

\[ \text{MIS}(G) = \max \begin{cases} \text{MIS}(G - v) \\ \text{MIS}(G - v - N(v)) + w(v) \end{cases} \]
Maximum Independent Set (VI)

$$MIS(G) = \max \begin{cases} 
MIS(G - v) \\
MIS(G - v - N(v)) + w(v)
\end{cases}$$
Maximum Independent Set (VI)

\[ \text{MIS}(G) = \max \begin{cases} \text{MIS}(G - v) \\ \text{MIS}(G - v - N(v)) + w(v) \end{cases} \]
Maximum Independent Set (VI)

\[ \text{MIS}(G) = \max \begin{cases} 
\text{MIS}(G - v) \\
\text{MIS}(G - v - N(v)) + w(v)
\end{cases} \]
Maximum Independent Set (VI)

\[ \text{MIS}(G) = \max \begin{cases} 
\text{MIS}(G - v) \\
\text{MIS}(G - v - N(v)) + \text{w}(v) 
\end{cases} \]
Maximum Independent Set (VII)

recursive-MIS($G = (V, E), w: V \rightarrow N$):

if $V = \emptyset$ return 0

choose $v \in V$

return $\max(\text{recursive-MIS}(G - v), \text{recursive-MIS}(G - v - N(v)) + w(v))$

Correctness:

Complexity:

$n := |V|$

$T(0), T(1) \geq \Omega(1)$.

$T(n) \geq T(n - 1) + T(n - 1 - \deg(v)) \geq 2T(n - 2) \geq \cdots \geq 2^n \cdot T(1) \geq \Omega(2^n)$.

When $G$ has no edges then clearly $\text{MIS}(G) = |V|$, but this worst-case runtime is hard to avoid.

Memoization does not obviously help — subproblems correspond to subgraphs, of which there are possibly exponentially many.
Maximum Independent Set (VII)

\[
\text{recursive-MIS}(G = (V, E), w : V \rightarrow \mathbb{N}):
\]

\[
\text{if } V = \emptyset \text{ return } 0.
\]

\[
\text{choose } v \in V.
\]

\[
\text{return } \max(\text{recursive-MIS}(G - v), \text{recursive-MIS}(G - v - \text{neighbors}(v)) + w(v)).
\]
Maximum Independent Set (VII)

\[
\text{recursive-MIS}(G = (V, E), w : V \rightarrow \mathbb{N}) : \\
\quad \text{if } V = \emptyset \\
\]

recursive-MIS\((G = (V, E), w : V \rightarrow \mathbb{N})\):
    if \(V = \emptyset\)
        return 0
Maximum Independent Set (VII)

\[
\text{recursive-MIS}(G = (V, E), w : V \rightarrow \mathbb{N}): \\
\text{if } V = \emptyset \\
\quad \text{return } 0 \\
\quad \text{choose } v \in V
\]
Maximum Independent Set (VII)

recursive-MIS(G = (V, E), w : V → N):
  if V = ∅
    return 0
  choose v ∈ V
  return max (recursive-MIS(G − v), recursive-MIS(G − v − N(v)) + w(v))
recursive-MIS($G = (V, E), w : V \rightarrow \mathbb{N}$):
   if $V = \emptyset$
      return 0
   choose $v \in V$
   return $\max \left( \text{recursive-MIS}(G - v), \text{recursive-MIS}(G - \text{N}(v)) \right) + w(v)$
Maximum Independent Set (VII)

recursive-MIS\((G = (V, E), w : V \rightarrow \mathbb{N})\):

if \(V = \emptyset\)

return 0

choose \(v \in V\)

return \(\max\left(\text{recursive-MIS}(G - v), \text{recursive-MIS}(G - v - N(v)) + w(v)\right)\)
recursive-MIS\((G = (V, E), w : V \rightarrow \mathbb{N})\):

\[
\begin{align*}
\text{if } V &= \emptyset \\
\text{return } 0 \\
\text{choose } v \in V \\
\text{return } \max \left( \text{recursive-MIS}(G - v), \text{recursive-MIS}(G - v - N(v)) + w(v) \right)
\end{align*}
\]
Maximum Independent Set (VII)

recursive-MIS\((G = (V, E), w : V \rightarrow \mathbb{N})\):

\[
\text{if } V = \emptyset \\
\quad \text{return } 0 \\
\text{choose } v \in V \\
\text{return } \max \left( \text{recursive-MIS}(G - v), \text{recursive-MIS}(G - v - N(v)) + w(v) \right)
\]

correctness:
Maximum Independent Set (VII)

```
recursive-MIS(G = (V, E), w : V → N):
    if V = ∅
        return 0
    choose v ∈ V
    return max (recursive-MIS(G − v), recursive-MIS(G − v − N(v)) + w(v))
```

correctness: clear
recursive-MIS\((G = (V, E), w : V \to \mathbb{N})\):

\[
\begin{align*}
&\text{if } V = \emptyset \\
&\quad \text{return } 0 \\
&\text{choose } v \in V \\
&\quad \text{return } \max \left( \text{recursive-MIS}(G - v), \text{recursive-MIS}(G - v - N(v)) + w(v) \right)
\end{align*}
\]

correctness: clear

complexity:
Maximum Independent Set (VII)

```plaintext
recursive-MIS(G = (V, E), w : V → N):
    if V = ∅
        return 0
    choose v ∈ V
    return max (recursive-MIS(G − v), recursive-MIS(G − v − N(v)) + w(v))
```

correctness: clear
complexity: \( n := |V| \)
Maximum Independent Set (VII)

recursive-MIS\(G = (V, E), w : V \to \mathbb{N}):\)

\[
\text{if } V = \emptyset \\
\quad \text{return } 0 \\
\text{choose } v \in V \\
\quad \text{return } \max\left(\text{recursive-MIS}(G - v), \text{recursive-MIS}(G - v - N(v)) + w(v)\right)
\]

**correctness:** clear

**complexity:** \(n := |V|\)

- \(T(0), T(1) \geq \Omega(1)\).
Maximum Independent Set (VII)

\[
\text{recursive-MIS}(G = (V, E), w : V \to \mathbb{N}): \\
\text{if } V = \emptyset \\
\quad \text{return } 0 \\
\text{choose } v \in V \\
\text{return } \max \left( \text{recursive-MIS}(G - v), \text{recursive-MIS}(G - v - N(v)) + w(v) \right)
\]

correctness: clear
complexity: \( n := |V| \)

- \( T(0), T(1) \geq \Omega(1). \) \( T(n) \geq T(n - 1) + T(n - 1 - \deg(v)) \)
Maximum Independent Set (VII)

```plaintext
recursive-MIS(G = (V, E), w : V → N):
  if V = ∅
    return 0
  choose v ∈ V
  return max(recursive-MIS(G − v), recursive-MIS(G − v − N(v)) + w(v))
```

correctness: clear

complexity: $n := |V|
  - $T(0), T(1) \geq \Omega(1)$. $T(n) \geq T(n − 1) + T(n − 1 − \text{deg}(v))$
  - silly case:
Maximum Independent Set (VII)

recursive-MIS\((G = (V, E), w : V \rightarrow \mathbb{N})\):

if \(V = \emptyset\)
    return 0

choose \(v \in V\)

return \(\max\left(\text{recursive-MIS}(G - v), \text{recursive-MIS}(G - v - N(v)) + w(v)\right)\)

**correctness**: clear

**complexity**: \(n := |V|\)

- \(T(0), T(1) \geq \Omega(1). \ T(n) \geq T(n - 1) + T(n - 1 - \deg(v))\)
- silly case: \(G\) has no edges
Maximum Independent Set (VII)

recursive-MIS\((G = (V, E), w : V \to \mathbb{N})\):

\[
\text{if } V = \emptyset \\
\quad \text{return } 0 \\
\text{choose } v \in V \\
\quad \text{return } \max \left( \text{recursive-MIS}(G - v), \text{recursive-MIS}(G - v - N(v)) + w(v) \right)
\]

correctness: clear

complexity: \( n := |V| \)

- \( T(0), T(1) \geq \Omega(1) \). \( T(n) \geq T(n - 1) + T(n - 1 - \text{deg}(v)) \)
- silly case: \( G \) has no edges \( \implies \) for all \( v \), \( \text{deg}(v) = 0 \)
Maximum Independent Set (VII)

recursive-MIS\((G = (V, E), w : V \rightarrow \mathbb{N})\):

\[
\text{if } V = \emptyset \\
\quad \text{return } 0 \\
\text{choose } v \in V \\
\quad \text{return } \max \left( \text{recursive-MIS}(G - v), \text{recursive-MIS}(G - v - N(v)) + w(v) \right)
\]

correctness: clear

complexity: \(n := |V|\)

\(T(0), T(1) \geq \Omega(1). \quad T(n) \geq T(n - 1) + T(n - 1 - \deg(v))\)

silly case: \(G\) has no edges \(\implies\) for all \(v\), \(\deg(v) = 0\)

\(\implies T(n) \geq 2T(n - 1)\)
Maximum Independent Set (VII)

recursive-MIS($G = (V, E), w : V \rightarrow \mathbb{N}$):
  - if $V = \emptyset$
    - return 0
  - choose $v \in V$
  - return $\max \left( \text{recursive-MIS}(G - v), \text{recursive-MIS}(G - v - N(v)) + w(v) \right)$

**correctness:** clear

**complexity:** $n := |V|$

- $T(0), T(1) \geq \Omega(1)$. $T(n) \geq T(n - 1) + T(n - 1 - \deg(v))$
- silly case: $G$ has no edges $\implies$ for all $v$, $\deg(v) = 0$
  $\implies$ $T(n) \geq 2T(n - 1) \geq 4T(n - 2)$
Maximum Independent Set (VII)

recursive-MIS(\(G = (V, E), w : V \rightarrow \mathbb{N}\)):

\[
\text{if } V = \emptyset \\
\quad \text{return 0} \\
\text{choose } v \in V \\
\quad \text{return max} \left( \text{recursive-MIS}(G - v), \text{recursive-MIS}(G - v - N(v)) + w(v) \right)
\]

**correctness:** clear

**complexity:** \(n := |V|\)

- \(T(0), T(1) \geq \Omega(1)\). \(T(n) \geq T(n - 1) + T(n - 1 - \text{deg}(v))\)
- silly case: \(G\) has no edges \(\implies\) for all \(v\), \(\text{deg}(v) = 0\)

\(\implies T(n) \geq 2T(n - 1) \geq 4T(n - 2) \geq \cdots \geq \)
Maximum Independent Set (VII)

recursive-MIS\((G = (V, E), w : V \rightarrow \mathbb{N}):\)

\[
\text{if } V = \emptyset \\
\quad \text{return } 0 \\
\text{choose } v \in V \\
\quad \text{return } \max \left( \text{recursive-MIS}(G - v), \text{recursive-MIS}(G - v - N(v)) + w(v) \right)
\]

**correctness:** clear

**complexity:** \(n := |V|\)

- \(T(0), T(1) \geq \Omega(1). \ T(n) \geq T(n - 1) + T(n - 1 - \deg(v))\)
- silly case: \(G\) has no edges \(\implies\) for all \(v, \ \deg(v) = 0\)
  \(\implies T(n) \geq 2T(n - 1) \geq 4T(n - 2) \geq \cdots \geq 2^n \cdot T(1)\)
recursive-MIS\((G = (V, E), w : V \rightarrow \mathbb{N})\):

\[
\text{if } V = \emptyset \quad \text{return } 0 \\
\text{choose } v \in V \\
\text{return } \max \left( \text{recursive-MIS}(G - v), \text{recursive-MIS}(G - v - N(v)) + w(v) \right)
\]

correctness: clear

complexity: \(n := |V|\)

- \(T(0), T(1) \geq \Omega(1)\). \(T(n) \geq T(n - 1) + T(n - 1 - \text{deg}(v))\)
- silly case: \(G\) has no edges \(\implies\) for all \(v\), \(\text{deg}(v) = 0\)

\(\implies\) \(T(n) \geq 2T(n - 1) \geq 4T(n - 2) \geq \cdots \geq 2^n \cdot T(1) \geq \Omega(2^n)\).
Maximum Independent Set (VII)

recursive-MIS($G = (V,E), w : V \rightarrow \mathbb{N}$):
  
  if $V = \emptyset$
    return 0
  
  choose $v \in V$
  
  return $\max \left( \text{recursive-MIS}(G - v), \text{recursive-MIS}(G - v - N(v)) + w(v) \right)$

**correctness:** clear

**complexity:** $n := |V|$

- $T(0), T(1) \geq \Omega(1)$. $T(n) \geq T(n - 1) + T(n - 1 - \deg(v))$
- silly case: $G$ has no edges \(\implies\) for all $v$, $\deg(v) = 0$
- \(\implies\) $T(n) \geq 2T(n - 1) \geq 4T(n - 2) \geq \cdots \geq 2^n \cdot T(1) \geq \Omega(2^n)$.
- when $G$ has no edges then clearly MIS($G$) = $|V|$,
Maximum Independent Set (VII)

**recursive-MIS** \((G = (V, E), w : V \to \mathbb{N})\): 

- if \(V = \emptyset\)
  - return 0
- choose \(v \in V\)
  - return \(\max(\text{recursive-MIS}(G - v), \text{recursive-MIS}(G - v - N(v)) + w(v))\)

**correctness:** clear

**complexity:** \(n := |V|\)

- \(T(0), T(1) \geq \Omega(1). \ T(n) \geq T(n - 1) + T(n - 1 - \deg(v))\)
- silly case: \(G\) has no edges \(\implies\) for all \(v\), \(\deg(v) = 0\)
  \(\implies\) \(T(n) \geq 2T(n - 1) \geq 4T(n - 2) \geq \cdots \geq 2^n \cdot T(1) \geq \Omega(2^n)\).
- when \(G\) has no edges then clearly \(\text{MIS}(G) = |V|\), but this worst-case runtime is hard to avoid
Maximum Independent Set (VII)

```
recursive-MIS(G = (V, E), w : V → N):
    if V = ∅
        return 0
    choose v ∈ V
    return max (recursive-MIS(G − v), recursive-MIS(G − v − N(v)) + w(v))
```

correctness: clear

complexity: \( n := |V| \)

- \( T(0), T(1) \geq \Omega(1) \). \( T(n) \geq T(n − 1) + T(n − 1 − \text{deg}(v)) \)
- silly case: \( G \) has no edges \( \implies \) for all \( v \), \( \text{deg}(v) = 0 \)

\( \implies T(n) \geq 2T(n − 1) \geq 4T(n − 2) \geq \cdots \geq 2^n \cdot T(1) \geq \Omega(2^n) \).

- when \( G \) has no edges then clearly \( \text{MIS}(G) = |V| \), but this worst-case runtime is hard to avoid
- memoization does not obviously help
Maximum Independent Set (VII)

recursive-MIS(G = (V, E), w : V → N):
  if V = ∅
    return 0
  choose v ∈ V
  return max (recursive-MIS(G − v), recursive-MIS(G − v − N(v)) + w(v))

correctness: clear
complexity: n := |V|
  - T(0), T(1) ≥ Ω(1). T(n) ≥ T(n − 1) + T(n − 1 − deg(v))
  - silly case: G has no edges ⇒ for all v, deg(v) = 0
    ⇒ T(n) ≥ 2T(n − 1) ≥ 4T(n − 2) ≥ ... ≥ 2^n · T(1) ≥ Ω(2^n).
  - when G has no edges then clearly MIS(G) = |V|, but this worst-case runtime is
    hard to avoid
  - memoization does not obviously help — subproblems correspond to subgraphs,
Maximum Independent Set (VII)

\[
\text{recursive-MIS}(G = (V, E), w : V \to \mathbb{N}) :
\]
\[
\text{if } V = \emptyset \\
\quad \text{return } 0 \\
\text{choose } v \in V \\
\text{return } \max \left( \text{recursive-MIS}(G - v), \text{recursive-MIS}(G - v - N(v)) + w(v) \right)
\]

correctness: clear

complexity: \( n := |V| \)

\begin{itemize}
  \item \( T(0), T(1) \geq \Omega(1) \). \( T(n) \geq T(n - 1) + T(n - 1 - \deg(v)) \)
  \item silly case: \( G \) has no edges \( \implies \) for all \( v \), \( \deg(v) = 0 \)
  \begin{align*}
  \implies T(n) &\geq 2T(n - 1) \geq 4T(n - 2) \geq \cdots \geq 2^n \cdot T(1) \geq \Omega(2^n). \\
  \text{when } G \text{ has no edges then clearly } &\text{MIS}(G) = |V|, \text{ but this worst-case runtime is hard to avoid} \\
  \text{memoization does not obviously help — subproblems correspond to subgraphs, of which there are possibly exponentially many}
  \end{align*}
\end{itemize}
Maximum Independent Set, in Trees

Question: maximum weight independent set, in trees?

Question: how to bound the number of subproblems in recursive algorithm? how to pick which vertex \( v \in V \) to eliminate?
question:
question: maximum weight independent set,
question: maximum weight independent set, in trees?
question: maximum weight independent set, in trees?
question: maximum weight independent set, in trees?
Maximum Independent Set, in Trees

**question:** maximum weight independent set, in trees?

![Tree diagram](image)

**question:** how to bound the number of subproblems in recursive algorithm?
**Maximum Independent Set, in Trees**

**question:** maximum weight independent set, in trees?

![Tree Diagram]

**question:**
- how to bound the number of subproblems in recursive algorithm?
- how to pick which vertex \( v \in V \) to eliminate?
Maximum Independent Set, in Trees (II)

\[ \text{MIS}(G) = \max \{ \text{MIS}(G - v) \} \]

\[ \text{MIS}(G - v - N(v)) + w(v) \]

\[ r \ a \ c \ h \]

\[ i \]

\[ b \ f \]

\[ j \ a \]

\[ c \ d \]

\[ h \]

\[ e \]

\[ b \ f \]

\[ g \]

\[ j \]

\[ c \ d \]

\[ h \]

\[ i \]

\[ b \ f \]

\[ g \]

\[ j \]
Maximum Independent Set, in Trees (II)

\[ \text{MIS}(G) = \max \begin{cases} \text{MIS}(G - v) \\ \text{MIS}(G - v - N(v)) + w(v) \end{cases} \]

Diagram: A tree structure with vertices labeled as follows: r, a, b, c, d, e, f, g, h, i, j.
Maximum Independent Set, in Trees (II)

\[
\text{MIS}(G) = \max \left\{ \text{MIS}(G - v), \text{MIS}(G - v - N(v)) + w(v) \right\}
\]
Maximum Independent Set, in Trees (II)

\[
\text{MIS}(G) = \max \begin{cases} 
\text{MIS}(G - v) \\
\text{MIS}(G - v - N(v)) + w(v)
\end{cases}
\]
Maximum Independent Set, in Trees (II)

\[
\text{MIS}(G) = \max \begin{cases} 
\text{MIS}(G - v) \\
\text{MIS}(G - v - N(v)) + w(v)
\end{cases}
\]
Maximum Independent Set, in Trees (II)

\[ \text{MIS}(G) = \max \left\{ \text{MIS}(G - v), \text{MIS}(G - v - N(v)) + w(v) \right\} \]
Lemma

Let $T = (V, E)$ be a tree, with root $v \in V$. Then $T - v$ is a forest, with each tree associated to a child $u$ of $v$. $T - v - N(v)$ is a forest, with each tree associated to a grandchild $w$ of $v$.

Proof.
Lemma

Let $T = (V, E)$ be a tree, with root $v \in V$. Then $T - v$ is a forest, with each tree associated to a child $u$ of $v$. $T - v - N(v)$ is a forest, with each tree associated to a grandchild $w$ of $v$.

Proof.
Lemma

Let $T = (V, E)$ be a tree,
Lemma

Let $T = (V, E)$ be a tree, with root $v \in V$. Then $T - v$ is a forest, with each tree associated to a child $u$ of $v$. $T - v - N(v)$ is a forest, with each tree associated to a grandchild $w$ of $v$. 

Proof.
Lemma

Let $T = (V, E)$ be a tree, with root $v \in V$. Then

$T - v$ is a forest, with each tree associated to a child $u$ of $v$.

$T - v - N(v)$ is a forest, with each tree associated to a grandchild $w$ of $v$. 

Proof.
Lemma

Let $T = (V, E)$ be a tree, with root $v \in V$. Then

- $T - v$ is a forest,
Lemma

Let $T = (V, E)$ be a tree, with root $v \in V$. Then

- $T - v$ is a forest, with each tree associated to a child $u$ of $v$. 
Lemma

Let $T = (V, E)$ be a tree, with root $v \in V$. Then

- $T - v$ is a forest, with each tree associated to a child $u$ of $v$.
- $T - v - N(v)$ is a forest,
Lemma

Let $T = (V, E)$ be a tree, with root $v \in V$. Then

- $T - v$ is a forest, with each tree associated to a child $u$ of $v$.
- $T - v - N(v)$ is a forest, with each tree associated to a grandchild $w$ of $v$. 
Lemma

Let $T = (V, E)$ be a tree, with root $v \in V$. Then

- $T - v$ is a forest, with each tree associated to a child $u$ of $v$.
- $T - v - N(v)$ is a forest, with each tree associated to a grandchild $w$ of $v$.

Proof.
Lemma

Let \( T = (V, E) \) be a tree, with root \( v \in V \). Then

- \( T - v \) is a forest, with each tree associated to a child \( u \) of \( v \).
- \( T - v - N(v) \) is a forest, with each tree associated to a grandchild \( w \) of \( v \).

Proof.
Lemma

Let $T = (V, E)$ be a tree, with root $v \in V$. Then $T - v$ is a forest, with each tree associated to a child $u$ of $v$.

$T - v - N(v)$ is a forest, with each tree associated to a grandchild $w$ of $v$.

Corollary

Let $T = (V, E)$ be a tree.

Pick a root $r \in V$ for $T$ to create the rooted tree $(T, r)$.

Running recursive-MIS on $T$ and eliminating nodes closest to $r$ in $T$, then the result subproblems exactly correspond to forests of rooted subtrees of $(T, r)$, and disjoint rooted subtrees can be solved independently $\Rightarrow \leq |V|$ subproblems $\Rightarrow$ memoized recursive algorithm is efficient.
Lemma

Let $T = (V, E)$ be a tree, with root $v \in V$. Then

- $T - v$ is a forest, with each tree associated to a child $u$ of $v$.
- $T - v - N(v)$ is a forest, with each tree associated to a grandchild $w$ of $v$. 
Lemma

Let $T = (V, E)$ be a tree, with root $v \in V$. Then

- $T - v$ is a forest, with each tree associated to a child $u$ of $v$.
- $T - v - N(v)$ is a forest, with each tree associated to a grandchild $w$ of $v$.

Corollary

Let $T = (V, E)$ be a tree.

Lemma

Let $T = (V, E)$ be a tree, with root $v \in V$. Then

- $T - v$ is a forest, with each tree associated to a child $u$ of $v$.
- $T - v - N(v)$ is a forest, with each tree associated to a grandchild $w$ of $v$.

Corollary

Let $T = (V, E)$ be a tree. Pick a root $r \in V$ for $T$ to create the rooted tree $(T, r)$.
**Lemma**

Let $T = (V, E)$ be a tree, with root $v \in V$. Then

- $T - v$ is a forest, with each tree associated to a child $u$ of $v$.
- $T - v - N(v)$ is a forest, with each tree associated to a grandchild $w$ of $v$.

**Corollary**

Let $T = (V, E)$ be a tree. Pick a root $r \in V$ for $T$ to create the rooted tree $(T, r)$. Running recursive-MIS on $T$
Lemma

Let $T = (V, E)$ be a tree, with root $v \in V$. Then

- $T - v$ is a forest, with each tree associated to a child $u$ of $v$.
- $T - v - N(v)$ is a forest, with each tree associated to a grandchild $w$ of $v$.

Corollary

Let $T = (V, E)$ be a tree. Pick a root $r \in V$ for $T$ to create the rooted tree $(T, r)$. Running recursive-MIS on $T$ and eliminating nodes closest to $r$ in $T$, ...
Lemma

Let $T = (V, E)$ be a tree, with root $v \in V$. Then

- $T - v$ is a forest, with each tree associated to a child $u$ of $v$.
- $T - v - N(v)$ is a forest, with each tree associated to a grandchild $w$ of $v$.

Corollary

Let $T = (V, E)$ be a tree. Pick a root $r \in V$ for $T$ to create the rooted tree $(T, r)$. Running recursive-MIS on $T$ and eliminating nodes closest to $r$ in $T$, then the result subproblems exactly correspond to forests of rooted subtrees of $(T, r)$.
Lemma

Let \( T = (V, E) \) be a tree, with root \( v \in V \). Then

- \( T - v \) is a forest, with each tree associated to a child \( u \) of \( v \).
- \( T - v - N(v) \) is a forest, with each tree associated to a grandchild \( w \) of \( v \).

Corollary

Let \( T = (V, E) \) be a tree. Pick a root \( r \in V \) for \( T \) to create the rooted tree \( (T, r) \). Running recursive-MIS on \( T \) and eliminating nodes closest to \( r \) in \( T \), then the result subproblems exactly correspond to forests of rooted subtrees of \( (T, r) \), and disjoint rooted subtrees can be solved independently.
Lemma

Let $T = (V, E)$ be a tree, with root $v \in V$. Then

- $T - v$ is a forest, with each tree associated to a child $u$ of $v$.
- $T - v - N(v)$ is a forest, with each tree associated to a grandchild $w$ of $v$.

Corollary

Let $T = (V, E)$ be a tree. Pick a root $r \in V$ for $T$ to create the rooted tree $(T, r)$. Running recursive-MIS on $T$ and eliminating nodes closest to $r$ in $T$, then the result subproblems exactly correspond to forests of rooted subtrees of $(T, r)$, and disjoint rooted subtrees can be solved independently

$\Rightarrow \leq |V|$ subproblems
Lemma

Let $T = (V, E)$ be a tree, with root $v \in V$. Then

- $T - v$ is a forest, with each tree associated to a child $u$ of $v$.
- $T - v - N(v)$ is a forest, with each tree associated to a grandchild $w$ of $v$.

Corollary

Let $T = (V, E)$ be a tree. Pick a root $r \in V$ for $T$ to create the rooted tree $(T, r)$. Running recursive-MIS on $T$ and eliminating nodes closest to $r$ in $T$, then the result subproblems exactly correspond to forests of rooted subtrees of $(T, r)$, and disjoint rooted subtrees can be solved independently

$\Rightarrow \leq |V|$ subproblems

$\Rightarrow$ memoized recursive algorithm is efficient
For a rooted tree $T$ with root $r$, for $v \in V$ define $T(v)$ to be the subtree of $T$ descending from $v$.

The recursive formula is then:

$$\text{MIS}(T) = \max\left\{ \sum_{v \in N(v)} \text{MIS}(T(v)) \right\}$$

dependency graph:

subproblems are rooted subtrees of $(T, r)$

a subtree $T(v)$ depends on all of subtrees $T(u)$ where $u$ is a descendent of $v$.

$= \Rightarrow$ iterating over $V$ in post-order traversal of $T$ will satisfy the dependency graph.
For a rooted tree $T$ with root $r$, define $T(v)$ to be the subtree of $T$ descending from $v$.

The recursive formula is then:

$$\text{MIS}(T) = \max \left\{ \sum_{v \in N(v)} \text{MIS}(T(v)) \right\} + w(v)$$
For a rooted tree $T$ with root $r$, for $v \in V$ define $T(v)$ to be the subtree of $T$ descending from $v$. 
For a rooted tree \( T \) with root \( r \), for \( v \in V \) define \( T(v) \) to be the subtree of \( T \) descending from \( v \). The recursive formula is then:

\[
MIS(T) = \max \left\{ \sum_{v \in N(v)} MIS(T(v)) \right\} \sum_{v \in N(v)} MIS(T(v)) + w(v)
\]

dependency graph: subproblems are rooted subtrees of \((T, r)\) a subtree \( T(v) \) depends on all of subtrees \( T(u) \) where \( u \) is a descendent of \( v \) = \( \Rightarrow \) iterating over \( V \) in post-order traversal of \( T \) will satisfy the dependency graph
For a rooted tree $T$ with root $r$, for $v \in V$ define $T(v)$ to be the subtree of $T$ descending from $v$. The recursive formula is then:

$$\text{MIS}(T) = \max \left\{ \sum_{v \in N(v)} \text{MIS}(T(v)) \right\}$$
For a rooted tree $T$ with root $r$, for $v \in V$ define $T(v)$ to be the subtree of $T$ descending from $v$. The recursive formula is then:

$$MIS(T) = \max \left\{ \sum_{v \in N(v)} MIS(T(v)) \right\}$$

$$\left( \sum_{v \in N(v)} MIS(T(v)) \right) + w(v)$$
For a rooted tree $T$ with root $r$, for $v \in V$ define $T(v)$ to be the subtree of $T$ descending from $v$. The recursive formula is then:

$$\text{MIS}(T) = \max \left\{ \sum_{v \in N(v)} \text{MIS}(T(v)) \right\}$$

dependency graph:

- Subproblems are rooted subtrees of $(T, r)$
- A subtree $T(v)$ depends on all subtrees $T(u)$ where $u$ is a descendent of $v$

- Iterating over $V$ in post-order traversal of $T$ will satisfy the dependency graph
For a rooted tree $T$ with root $r$, for $v \in V$ define $T(v)$ to be the subtree of $T$ descending from $v$. The recursive formula is then:

$$
MIS(T) = \max \left\{ \sum_{v \in N(v)} MIS(T(v)) \left( \sum_{v \in N(N(v))} MIS(T(v)) \right) + w(v) \right\}
$$

**dependency graph:**
- subproblems are rooted subtrees of $(T, r)$
For a rooted tree $T$ with root $r$, for $v \in V$ define $T(v)$ to be the subtree of $T$ descending from $v$. The recursive formula is then:

$$
\text{MIS}(T) = \max \left\{ \sum_{v \in N(v)} \text{MIS}(T(v)) \right\} + w(v)
$$

**dependency graph:**
- subproblems are rooted subtrees of $(T, r)$
- a subtree $T(v)$ depends on all of subtrees $T(u)$ where $u$ is a descendent of $v$
For a rooted tree $T$ with root $r$, for $v \in V$ define $T(v)$ to be the subtree of $T$ descending from $v$. The recursive formula is then:

$$\text{MIS}(T) = \max \left\{ \sum_{v \in N(v)} \text{MIS}(T(v)) \right\} + w(v)$$

**dependency graph:**

- subproblems are rooted subtrees of $(T, r)$
- a subtree $T(v)$ depends on all of subtrees $T(u)$ where $u$ is a descendent of $v$

$\Rightarrow$ iterating over $V$ in post-order traversal of $T$ will satisfy the dependency graph
Maximum Independent Set, in Trees (V)

iterative algorithm:

\[
\text{iter-MIS-tree} \left( T = (V, E), w \right):
\]

let \(v_1, v_2, \ldots, v_n\) be a post-order traversal of nodes of \(T\)

\[
M[i] = \max \left\{ \sum_{j: v_j \in N(v_i)} M[j] + \sum_{j: v_j \in N(N(v_i))} M[j] + w(v_i) \right\}
\]

return \(M[n]\)

\(\text{correctness:}\)

\(\text{complexity:}\)

\(O(n)\) space to store \(M[·]\)

\(O(n)\) time per node,

\(n\) nodes \(\Rightarrow\) \(O(n^2)\) time

\(\text{better:}\)

each node \(v_j\) has its \(M[j]\) value read by parent, and by grandparent

\(\Rightarrow\) \(O(1)\) work per \(n\) nodes \(\Rightarrow\) \(O(n)\) time
iterative algorithm:

\[
\begin{align*}
\text{let } v_1, v_2, \ldots, v_n \text{ be a post-order traversal of nodes of } T &= \Rightarrow v_n \text{ is the root for } 1 \leq i \leq n \\
M[i] &= \max \left\{ \sum_{j : v_j \in N(v_i)} M[j] \left( \sum_{j : v_j \in N(N(v_i)))} M[j] \right) + w(v_i) \right\} \\
\text{return } M[n]
\end{align*}
\]
iterative algorithm:

\texttt{iter-MIS-tree}(T = (V, E), w : V \rightarrow \mathbb{N}):
iterative algorithm:

iter-MIS-tree\(T = (V, E), w : V \rightarrow \mathbb{N})\):

let \(v_1, v_2, \ldots, v_n\) be a post-order traversal of nodes of \(T\)
Maximum Independent Set, in Trees (V)

iterative algorithm:

iter-MIS-tree(T = (V, E), w : V → N):
  let v₁, v₂, ..., vₙ be a post-order traversal of nodes of T
  \( v_n \) is the root
iterative algorithm:

iter-MIS-tree($T = (V, E), w : V \rightarrow \mathbb{N}$):

let $v_1, v_2, \ldots, v_n$ be a post-order traversal of nodes of $T$

$\Rightarrow v_n$ is the root

for $1 \leq i \leq n$

\[ M[i] = \max \{ \sum_{j : v_j \in N(v_i)} M[j] \mid \sum_{j : v_j \in N(N(v_i))} M[j] \} + w(v_i) \]
Maximum Independent Set, in Trees (V)

**iterative algorithm:**

```
iter-MIS-tree(T = (V, E), w : V → N):
    let v₁, v₂, ..., vₙ be a post-order traversal of nodes of T
    ⇒ vₙ is the root
    for 1 ≤ i ≤ n
        M[i] = max { ...
```
iterative algorithm:

iter-MIS-tree\((T = (V, E), w : V \rightarrow \mathbb{N})\):

let \(v_1, v_2, \ldots, v_n\) be a post-order traversal of nodes of \(T\)

\(
\Rightarrow v_n \text{ is the root}
\)

for \(1 \leq i \leq n\)

\[
M[i] = \max \left\{ \sum_{j : v_j \in N(v_i)} M[j] \right\}
\]

correctness:

complexity:

\(O(n)\) space to store \(M[i]\)

naive:

\(O(n)\) time per node, \(n\) nodes \(\Rightarrow O(n^2)\) time

better:

each node \(v_j\) has its \(M[j]\) value read by parent, and by grandparent \(\Rightarrow O(n)\) work per \(n\) nodes \(\Rightarrow O(n)\) time
Maximum Independent Set, in Trees (V)

iterative algorithm:

\[
\text{iter-MIS-tree}(T = (V, E), w : V \rightarrow \mathbb{N}): \\
\text{let } v_1, v_2, \ldots, v_n \text{ be a post-order traversal of nodes of } T \\
\Rightarrow v_n \text{ is the root} \\
\text{for } 1 \leq i \leq n \\
M[i] = \max \left\{ \sum_{j : v_j \in N(v_i)} M[j] \left( \sum_{j : v_j \in N(N(v_i))} M[j] \right) + w(v_i) \right\}
\]
Maximum Independent Set, in Trees (V)

iterative algorithm:

\[ \text{iter-MIS-tree}(T = (V, E), w : V \rightarrow \mathbb{N}) : \]
\[
\begin{align*}
\text{let } v_1, v_2, \ldots, v_n \text{ be a post-order traversal of nodes of } T \\
\quad \Longrightarrow v_n \text{ is the root} \\
\text{for } 1 \leq i \leq n \\
\quad M[i] = \max \left\{ \sum_{j : v_j \in N(v_i)} M[j] \right. \\
\quad \left. \left( \sum_{j : v_j \in N(N(v_i))} M[j] \right) + w(v_i) \right\} \\
\text{return } M[n]
\end{align*}
\]
iterative algorithm:

\[
\text{iter-MIS-tree}(T = (V, E), w : V \rightarrow \mathbb{N}) :
\]

let \( v_1, v_2, \ldots, v_n \) be a post-order traversal of nodes of \( T \)
\( \implies v_n \) is the root

for \( 1 \leq i \leq n \)

\[
M[i] = \max \left\{ \sum_{j : v_j \in N(v_i)} M[j] \mid (\sum_{j : v_j \in N(N(v_i))} M[j]) + w(v_i) \right\}
\]

return \( M[n] \)
iterative algorithm:

iter-MIS-tree\((T = (V, E), w : V \rightarrow \mathbb{N})\):

let \(v_1, v_2, \ldots, v_n\) be a post-order traversal of nodes of \(T\)

\[ v_n \] is the root

for \(1 \leq i \leq n\)

\[ M[i] = \max \left\{ \sum_{j : v_j \in N(v_i)} M[j] \right\} \left( \sum_{j : v_j \in N(N(v_i))} M[j] \right) + w(v_i) \]

return \(M[n]\)

correctness:
Maximum Independent Set, in Trees (V)

**iterative algorithm:**

\[
\text{iter-MIS-tree}(T = (V, E), w : V \rightarrow \mathbb{N}) :
\]

let \( v_1, v_2, \ldots, v_n \) be a post-order traversal of nodes of \( T \)
\[ \implies v_n \text{ is the root} \]

for \( 1 \leq i \leq n \)

\[
M[i] = \max \left\{ \sum_{j : v_j \in N(v_i)} M[j] \left( \sum_{j : v_j \in N(N(v_i))} M[j] \right) + w(v_i) \right\}
\]

return \( M[n] \)

**correctness:** clear
Maximum Independent Set, in Trees (V)

iterative algorithm:

\[
\text{iter-MIS-tree}(T = (V, E), w: V \rightarrow \mathbb{N}): \\
\text{let } v_1, v_2, \ldots, v_n \text{ be a post-order traversal of nodes of } T \\
\quad \Rightarrow v_n \text{ is the root} \\
\text{for } 1 \leq i \leq n \\
\quad M[i] = \max \left\{ \sum_{j: v_j \in N(v_i)} M[j] \left( \sum_{j: v_j \in N(N(v_i))} M[j] \right) + w(v_i) \right\} \\
\text{return } M[n]
\]

correctness: clear

complexity:
iterative algorithm:

```
iter-MIS-tree(T = (V, E), w : V → N):
    let v₁, v₂, ..., vₙ be a post-order traversal of nodes of T
        ⇒ vₙ is the root
    for 1 ≤ i ≤ n
        M[i] = max \left\{ \sum_{j : v_j ∈ N(v_i)} M[j], \left( \sum_{j : v_j ∈ N(N(v_i))} M[j] \right) + w(v_i) \right\}
    return M[n]
```

correctness: clear

complexity:
- \( O(n) \) space to store \( M[.] \)
iterative algorithm:

```plaintext
iter-MIS-tree(T = (V, E), w : V → N):
  let v₁, v₂, ..., vₙ be a post-order traversal of nodes of T
  ⇒ vₙ is the root
  for 1 ≤ i ≤ n
    M[i] = \max \left\{ \sum_{j : v_j \in N(v_i)} M[j] \right. 
    \left. \left( \sum_{j : v_j \in N(N(v_i))} M[j] \right) + w(v_i) \right\}
  return M[n]
```

correctness: clear

complexity:
- \(O(n)\) space to store \(M[\cdot]\)
- time
iterative algorithm:

```plaintext
iter-MIS-tree(T = (V, E), w : V → N):
    let v_1, v_2, ..., v_n be a post-order traversal of nodes of T
    \[ v_n \text{ is the root}\]
    for 1 ≤ i ≤ n
    \[ M[i] = \max \left\{ \sum_{j : v_j \in N(v_i)} M[j] \right\} \]
    return M[n]
```

correctness: clear

complexity:
- \( O(n) \) space to store \( M[\cdot] \)
- time
  - naive:
Maximum Independent Set, in Trees (V)

iterative algorithm:

\[
\text{iter-MIS-tree}(T = (V, E), w : V \to \mathbb{N}): \\
\text{let } v_1, v_2, \ldots, v_n \text{ be a post-order traversal of nodes of } T \\
\implies v_n \text{ is the root} \\
\text{for } 1 \leq i \leq n \\
M[i] = \max \left\{ \sum_{j : v_j \in N(v_i)} M[j] \right\} \\
\quad \left( \sum_{j : v_j \in N(N(v_i))} M[j] \right) + w(v_i) \\
\text{return } M[n]
\]

correctness: clear

complexity:

- \(O(n)\) space to store \(M[\cdot]\)
- time
  - naive: \(O(n)\) time per node,
iterative algorithm:

\[
\text{iter-MIS-tree}(T = (V, E), w : V \rightarrow \mathbb{N}): \\
\text{let } v_1, v_2, \ldots, v_n \text{ be a post-order traversal of nodes of } T \\
\quad \implies v_n \text{ is the root} \\
\text{for } 1 \leq i \leq n \\
\quad M[i] = \max \left\{ \sum_{j : v_j \in N(v_i)} M[j], \left( \sum_{j : v_j \in N(N(v_i))} M[j] \right) + w(v_i) \right\} \\
\text{return } M[n]
\]

**correctness:** clear

**complexity:**
- \(O(n)\) space to store \(M[\cdot]\)
- time
  - naive: \(O(n)\) time per node, \(n\) nodes
Maximum Independent Set, in Trees (V)

iterative algorithm:

```plaintext
iter-MIS-tree(T = (V, E), w : V → N):
    let v₁, v₂, ..., vₙ be a post-order traversal of nodes of T
    ⇒ vₙ is the root
    for 1 ≤ i ≤ n
        \[ M[i] = \max \left\{ \sum_{j : v_j \in N(v_i)} M[j] \middle| \left( \sum_{j : v_j \in N(N(v_i))} M[j] \right) + w(v_i) \right\} \]
    return M[n]
```

correctness: clear

complexity:
- \( O(n) \) space to store \( M[·] \)
- time
  - naive: \( O(n) \) time per node, \( n \) nodes \( ⇒ O(n^2) \)
iterative algorithm:

\[
\text{iter-MIS-tree}(T = (V, E), w : V \rightarrow \mathbb{N}) : \\
\text{let } v_1, v_2, \ldots, v_n \text{ be a post-order traversal of nodes of } T \\
\quad \Rightarrow \quad v_n \text{ is the root} \\
\text{for } 1 \leq i \leq n \\
\quad M[i] = \max \left\{ \sum_{j : v_j \in N(v_i)} M[j] \right\} \\
\quad \left( \sum_{j : v_j \in N(N(v_i))} M[j] \right) + w(v_i) \\
\text{return } M[n]
\]

correctness: clear

complexity:
- \(O(n)\) space to store \(M[\cdot]\)
- time
  - naive: \(O(n)\) time per node, \(n\) nodes \(\Rightarrow\) \(O(n^2)\)
  - better:
Maximum Independent Set, in Trees (V)

**iterative algorithm:**

```
iter-MIS-tree(T = (V, E), w : V → N):
  let v₁, v₂, ..., vₙ be a post-order traversal of nodes of T
  => vₙ is the root
  for 1 ≤ i ≤ n
    M[i] = max \left\{ \sum_{j : v_j \in N(v_i)} M[j] \right\}
    \left( \sum_{j : v_j \in N(N(v_i))} M[j] \right) + w(v_i)
  return M[n]
```

**correctness:** clear

**complexity:**

- $O(n)$ space to store $M[.]$
- time
  - *naive:* $O(n)$ time per node, $n$ nodes $\implies O(n^2)$
  - *better:* each node $v_j$ has its $M[j]$ value read by
Maximum Independent Set, in Trees (V)

iterative algorithm:

\[
\text{iter-MIS-tree}(T = (V, E), w : V \rightarrow \mathbb{N}): \\
\text{let } v_1, v_2, \ldots, v_n \text{ be a post-order traversal of nodes of } T \\
\quad \Rightarrow v_n \text{ is the root} \\
\text{for } 1 \leq i \leq n \\
\quad M[i] = \max \left\{ \sum_{j : v_j \in N(v_i)} M[j] \right\} \\
\quad \left( \sum_{j : v_j \in N(N(v_i))} M[j] \right) + w(v_i) \\
\text{return } M[n]
\]

correctness: clear

complexity:
- \(O(n)\) space to store \(M[\cdot]\)
- time
  - naive: \(O(n)\) time per node, \(n\) nodes \(\Rightarrow\) \(O(n^2)\)
  - better: each node \(v_j\) has its \(M[j]\) value read by parent,
Maximum Independent Set, in Trees (V)

**iterative algorithm:**

```plaintext
iter-MIS-tree(T = (V, E), w : V → N):
    let v₁, v₂, . . . , vₙ be a post-order traversal of nodes of T
    ⇒ vₙ is the root
    for 1 ≤ i ≤ n
    M[i] = max \{ \sum_{j : v_j \in N(v_i)} M[j] \}
         \left( \sum_{j : v_j \in N(N(v_i))} M[j] \right) + w(v_i)
    return M[n]
```

**correctness:** clear

**complexity:**

- \(O(n)\) space to store \(M[.]\)
- time
  - naive: \(O(n)\) time per node, \(n\) nodes \(⇒\) \(O(n^2)\)
  - better: each node \(v_j\) has its \(M[j]\) value read by parent, and by grandparent
Maximum Independent Set, in Trees (V)

iterative algorithm:

\[
\text{iter-MIS-tree}(T = (V, E), w : V \rightarrow \mathbb{N}) :
\]
\[
\text{let } v_1, v_2, \ldots, v_n \text{ be a post-order traversal of nodes of } T \\
\implies v_n \text{ is the root} \\
\text{for } 1 \leq i \leq n \\
M[i] = \max \left\{ \sum_{j : v_j \in N(v_i)} M[j] \\
\left( \sum_{j : v_j \in N(N(v_i))} M[j] \right) + w(v_i) \right\} \\
\text{return } M[n]
\]

correctness: clear

complexity:

- \(O(n)\) space to store \(M[\cdot]\)
- time
  - naive: \(O(n)\) time per node, \(n\) nodes \(\implies\) \(O(n^2)\)
  - better: each node \(v_j\) has its \(M[j]\) value read by parent, and by grandparent \(\implies\) \(O(1)\) work per \(n\) nodes
Maximum Independent Set, in Trees (V)

iterative algorithm:

```text
iter-MIS-tree(T = (V, E), w : V → N):
    let v_1, v_2, ..., v_n be a post-order traversal of nodes of T
    ⇒ v_n is the root
    for 1 ≤ i ≤ n
        \[ M[i] = \max \left\{ \sum_{j : v_j \in N(v_i)} M[j] \right\} \]
        \[ \left( \sum_{j : v_j \in N(N(v_i))} M[j] \right) + w(v_i) \]
    return M[n]
```

correctness: clear

complexity:

- \( O(n) \) space to store \( M[\cdot] \)
- time
  - naive: \( O(n) \) time per node, \( n \) nodes \( \Rightarrow \) \( O(n^2) \)
  - better: each node \( v_j \) has its \( M[j] \) value read by parent, and by grandparent \( \Rightarrow \) \( O(1) \) work per \( n \) nodes \( \Rightarrow \) \( O(n) \) time
Dynamic Programming, in Trees

Definition

\( G = (V, E) \).

A set of nodes \( S \subseteq V \) is a separator for \( G \) if \( G - S \) has at least two connected components, that is, \( G - S \) is disconnected.

\( S \) is a balanced separator if each connected component of \( G - S \) has at most \( \frac{2}{3} \cdot |V| \) vertices.

E.g., in trees, every vertex is a separator, but not all are balanced.

Remarks:

Every tree \( T \) has a balanced separator consisting of a single node.

Dynamic programming + small balanced separators \( \Rightarrow 2O(\sqrt{n}) \)-time MIS algorithm for planar graphs.
question:

Definition

\[ G = (V, E) \]

A set of nodes \( S \subseteq V \) is a separator for \( G \) if \( G - S \) has \( \geq 2 \) connected components, that is, \( G - S \) is disconnected.

\( S \) is a balanced if each connected component of \( G - S \) has \( \leq \frac{2}{3} \cdot |V| \) vertices.

E.g., in trees, every vertex is a separator, but not all are balanced.

Remarks:

Every tree \( T \) has a balanced separator consisting of a single node.

Dynamic Programming + small balanced separators \( \Rightarrow 2^{O(\sqrt{n})} \)-time MIS algorithm for planar graphs
question: why does dynamic programming work on trees?
question: why does dynamic programming work on trees?

Definition

A set of nodes $S \subseteq V$ is a separator for $G$ if $G - S$ has at least 2 connected components, that is, $G - S$ is disconnected.

$S$ is a balanced if each connected component of $G - S$ has at most $2^{\frac{2}{3} \cdot |V|}$ vertices.

e.g., in trees, every vertex is a separator, but not all are balanced.

remarks: every tree $T$ has a balanced separator consisting of a single node.

dynamic-programming + small balanced separators $\Rightarrow$ $2^{O(\sqrt{n})}$-time MIS algorithm for planar graphs.
Dynamic Programming, in Trees

**question:** why does dynamic programming work on trees?

**Definition**

\[ G = (V, E). \]
question: why does dynamic programming work on trees?

Definition

\( G = (V, E) \). A set of nodes \( S \subseteq V \) is a **separator for** \( G \).
question: why does dynamic programming work on trees?

Definition

$G = (V, E)$. A set of nodes $S \subseteq V$ is a separator for $G$ if $G - S$ has at $\geq 2$ connected components,
question: why does dynamic programming work on trees?

Definition

$G = (V, E)$. A set of nodes $S \subseteq V$ is a separator for $G$ if $G - S$ has at $\geq 2$ connected components, that is,
question: why does dynamic programming work on trees?

Definition

$G = (V, E)$. A set of nodes $S \subseteq V$ is a separator for $G$ if $G - S$ has at least 2 connected components, that is, $G - S$ is disconnected.
question: why does dynamic programming work on trees?

**Definition**

$G = (V, E)$. A set of nodes $S \subseteq V$ is a **separator for** $G$ if $G - S$ has at $\geq 2$ connected components, that is, $G - S$ is disconnected.

e.g., in trees,
question: why does dynamic programming work on trees?

Definition

$G = (V, E)$. A set of nodes $S \subseteq V$ is a separator for $G$ if $G - S$ has at $\geq 2$ connected components, that is, $G - S$ is disconnected.

e.g., in trees, every vertex is a separator,
question: why does dynamic programming work on trees?

**Definition**

\[ G = (V, E) \]. A set of nodes \( S \subseteq V \) is a **separator for** \( G \) if \( G - S \) has at \( \geq 2 \) connected components, that is, \( G - S \) is disconnected. 

\( S \) is a **balanced** if each connected component of \( G - S \) has \( \leq \frac{2}{3} \cdot |V| \) vertices.

e.g., in trees, every vertex is a separator,
Dynamic Programming, in Trees

**question:** why does dynamic programming work on trees?

**Definition**

$G = (V, E)$. A set of nodes $S \subseteq V$ is a **separator for** $G$ if $G - S$ has at $\geq 2$ connected components, that is, $G - S$ is disconnected. $S$ is a **balanced** if each connected component of $G - S$ has $\leq \frac{2}{3} \cdot |V|$ vertices.

e.g., in trees, *every* vertex is a separator, but not all are *balanced*. 
question: why does dynamic programming work on trees?

Definition

\( G = (V, E) \). A set of nodes \( S \subseteq V \) is a separator for \( G \) if \( G - S \) has at \( \geq 2 \) connected components, that is, \( G - S \) is disconnected.

\( S \) is a balanced if each connected component of \( G - S \) has \( \leq \frac{2}{3} \cdot |V| \) vertices.

e.g., in trees, every vertex is a separator, but not all are balanced.

remarks:
Dynamic Programming, in Trees

**question:** why does dynamic programming work on trees?

**Definition**

\( G = (V, E) \). A set of nodes \( S \subseteq V \) is a **separator** for \( G \) if \( G - S \) has at \( \geq 2 \) connected components, that is, \( G - S \) is disconnected.

\( S \) is a **balanced** if each connected component of \( G - S \) has \( \leq \frac{2}{3} \cdot |V| \) vertices.

e.g., in trees, every vertex is a separator, but not all are balanced.

**remarks:**

- every tree \( T \) has a balanced separator consisting of a single node
question: why does dynamic programming work on trees?

Definition

$G = (V, E)$. A set of nodes $S \subseteq V$ is a separator for $G$ if $G - S$ has at $\geq 2$ connected components, that is, $G - S$ is disconnected.

$S$ is a balanced if each connected component of $G - S$ has $\leq \frac{2}{3} \cdot |V|$ vertices.

e.g., in trees, every vertex is a separator, but not all are balanced.

remarks:

- every tree $T$ has a balanced separator consisting of a single node
- dynamic-programming
**question:** why does dynamic programming work on trees?

**Definition**

\[ G = (V, E). \] A set of nodes \( S \subseteq V \) is a **separator for** \( G \) if \( G - S \) has at \( \geq 2 \) connected components, that is, \( G - S \) is disconnected.

\( S \) is a **balanced** if each connected component of \( G - S \) has \( \leq \frac{2}{3} \cdot |V| \) vertices.

**remarks:**

- every tree \( T \) has a balanced separator consisting of a single node
- dynamic-programming + small balanced separators

---

e.g., in trees, *every* vertex is a separator, but not all are *balanced.*
Dynamic Programming, in Trees

**question:** why does dynamic programming work on trees?

**Definition**

\[ G = (V, E). \] A set of nodes \( S \subseteq V \) is a **separator for** \( G \) if \( G - S \) has at \( \geq 2 \) connected components, that is, \( G - S \) is disconnected.

\( S \) is a **balanced** if each connected component of \( G - S \) has \( \leq \frac{2}{3} \cdot |V| \) vertices.

e.g., in trees, *every* vertex is a separator, but not all are balanced.

**remarks:**

- every tree \( T \) has a balanced separator consisting of a single node
- dynamic-programming + small balanced separators \( \implies 2^{O(\sqrt{n})}\)-time MIS algorithm for *planar* graphs
Minimum Dominating Set

Definition

Let $G = (V, E)$ be an undirected (simple) graph. A dominating set of $G$ is a subset $S \subseteq V$ such that for all $v \in V$, either $v \in S$, or $v$ has a neighbor $u \in N(v)$ with $u \in S$.

Example:

Dominating sets include \{a, b, c, d, e, f\}, \{e, c, f\}, and \{a, b, f\}.
Definition

Let $G = (V, E)$ be an undirected (simple) graph. A dominating set of $G$ is a subset $S \subseteq V$ such that for all $v \in V$, either $v \in S$, or $v$ has a neighbor $u \in N(v)$ with $u \in S$.

Examples:
- $\{a, b, c, d, e, f\}$
- $\{e, c, f\}$
- $\{a, b, f\}$
Definition

Let $G = (V, E)$ be an undirected (simple) graph.
Minimum Dominating Set

Definition

Let $G = (V, E)$ be an undirected (simple) graph. A **dominating set of** $G$
Minimum Dominating Set

<table>
<thead>
<tr>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Let ( G = (V, E) ) be an undirected (simple) graph. A <strong>dominating set of</strong> ( G ) is a subset ( S \subseteq V ).</td>
</tr>
</tbody>
</table>

Dominating sets include \{a, b, c, d, e, f\}, \{e, c, f\}, and \{a, b, f\}. 
**Definition**

Let $G = (V, E)$ be an undirected (simple) graph. A **dominating set of** $G$ is a subset $S \subseteq V$ such that for all $v \in V$, every vertex in $V$ either belongs to $S$ or is adjacent to a vertex in $S$.

\begin{itemize}
  \item Dominating sets include: 
    \begin{itemize}
      \item \{a, b, c, d, e, f\}
      \item \{e, c, f\}
      \item \{a, b, f\}
    \end{itemize}
\end{itemize}
Minimum Dominating Set

**Definition**

Let $G = (V, E)$ be an undirected (simple) graph. A **dominating set of** $G$ is a subset $S \subseteq V$ such that for all $v \in V$, either $v \in S$,
Minimum Dominating Set

**Definition**

Let $G = (V, E)$ be an undirected (simple) graph. A **dominating set of** $G$ is a subset $S \subseteq V$ such that for all $v \in V$, either $v \in S$, or $v$ has neighbor $u \in N(v)$ with $u \in S$. 

Dominating sets include $\{a, b, c, d, e, f\}$, $\{e, c, f\}$, and $\{a, b, f\}$. 
Minimum Dominating Set

**Definition**
Let \( G = (V, E) \) be an undirected (simple) graph. A **dominating set** of \( G \) is a subset \( S \subseteq V \) such that for all \( v \in V \), either \( v \in S \), or \( v \) has neighbor \( u \in N(v) \) with \( u \in S \).

**ex:**
Definition

Let $G = (V, E)$ be an undirected (simple) graph. A **dominating set of** $G$ is a subset $S \subseteq V$ such that for all $v \in V$, either $v \in S$, or $v$ has neighbor $u \in N(v)$ with $u \in S$.

**ex:**

```
  a  b  c
  |
  a  e  d
  |
  f
```
Definition

Let $G = (V, E)$ be an undirected (simple) graph. A **dominating set of** $G$ is a subset $S \subseteq V$ such that for all $v \in V$, either $v \in S$, or $v$ has neighbor $u \in N(v)$ with $u \in S$.

**ex:**

![Graph diagram](image)

Dominating sets include $\{a, b, c, d, e, f\}$.
Definition

Let $G = (V, E)$ be an undirected (simple) graph. A **dominating set of** $G$ is a subset $S \subseteq V$ such that for all $v \in V$, either $v \in S$, or $v$ has neighbor $u \in N(v)$ with $u \in S$.

**ex:**

Dominating sets include \{a, b, c, d, e, f\}, \{e, c, f\},
Definition

Let $G = (V, E)$ be an undirected (simple) graph. A **dominating set of** $G$ is a subset $S \subseteq V$ such that for all $v \in V$, either $v \in S$, or $v$ has neighbor $u \in N(v)$ with $u \in S$.

**ex:**

![Graph](image)

Dominating sets include $\{a, b, c, d, e, f\}$, $\{e, c, f\}$, and $\{a, b, f\}$.
Minimum Dominating Set (II)

Definition

The minimum weight dominating set problem is to, given a undirected (simple) graph $G = (V, E)$ and a weight function $w: V \to \mathbb{N}$, output the weight of the minimum weight dominating set in $G$. That is, output $\max S \subseteq V$ dominating set of $G$ $\sum_{v \in S} w(v)$.
Minimum Dominating Set (II)

**Definition**

The minimum weight dominating set problem is to, given a undirected (simple) graph $G = (V, E)$ and a weight function $w : V \rightarrow \mathbb{N}$, output the weight of the minimum weight dominating set in $G$. That is, output $\max_{S \subseteq V} \sum_{v \in S} w(v)$.
Definition

The *minimum weight dominating set* problem is to,
Definition

The **minimum weight dominating set** problem is to, given a undirected (simple) graph $G = (V, E)$, output the weight of the minimum weight dominating set in $G$.

$$\sum_{v \in S} w(v)$$
Definition

The **minimum weight dominating set** problem is to, given a undirected (simple) graph $G = (V, E)$ and a weight function $w : V \rightarrow \mathbb{N}$,
Definition

The **minimum weight dominating set** problem is to, given a undirected (simple) graph $G = (V, E)$ and a weight function $w : V \rightarrow \mathbb{N}$, output the weight of the minimum weight dominating set in $G$. 
Definition

The **minimum weight dominating set** problem is to, given a undirected (simple) graph $G = (V, E)$ and a weight function $w : V \to \mathbb{N}$, output the weight of the minimum weight dominating set in $G$. That is, output

$$\max_{S \subseteq V} \sum_{v \in S} w(v).$$

$S$ dominating set of $G$
The **minimum weight dominating set** problem is to, given a undirected (simple) graph $G = (V, E)$ and a weight function $w : V \rightarrow \mathbb{N}$, output the weight of the minimum weight dominating set in $G$. That is, output

$$\max_{S \subseteq V} \sum_{v \in S} w(v).$$
Definition

The **minimum weight dominating set** problem is to, given a undirected (simple) graph $G = (V, E)$ and a weight function $w : V \rightarrow \mathbb{N}$, output the weight of the minimum weight dominating set in $G$. That is, output

$$\max_{S \subseteq V, \text{ dominating set of } G} \sum_{v \in S} w(v).$$
Minimum Dominating Set (III)

- The minimum (weight) dominating set problem can be solved via brute force: try all possible subsets. This leads to a time complexity of $O(2^n n)$.

- Currently, no efficient algorithm is known for the minimum weight dominating set problem.

- It is expected that an efficient algorithm for this problem does not exist.

- If the underlying graph is a tree, the minimum weight dominating set problem can be efficiently solved.
Minimum Dominating Set (III)

**Remarks:**

- Minimum (weight) dominating set is solvable via brute force: try *all* possible subsets $\Rightarrow$ solvable in time $O(n^{O(1)}2^n)$
- No efficient algorithm *currently* known
- Minimum weight dominating set is NP-hard $\Rightarrow$ an efficient algorithm *not* expected to exist
- Minimum weight dominating set is efficiently solvable if the underlying graph is a *tree*
Let $T(v)$ denote the subtree rooted at $v \in V$, and let $S(v)$ be any minimum weight dominating set for $T(v)$.

To build $S(r)$:

- $r \in S$:
  - could take any $S(a) \cup S(b) \cup \{r\}$
- $r \not\in S$:
  - could try to take any $S(a) \cup S(b)$, but how to dominate $r$?
  - need a "extra" dominating set from one of $T(a)$ and $T(b)$.

Question: how to parameterize these subproblems?
Let $T(v)$ denote the subtree rooted at $v \in V$, and let $S(v)$ be any minimum weight dominating set for $T(v)$.

Building $S(r)$:

- $r \in S(a) \cup S(b) \cup \{r\}$ could take any $S(a) \cup S(b) \cup \{r\}$.

If we cover $r$, then $a, b$ do not need to be covered — only need a "mostly" dominating set on $T(a)$ and $T(b)$.

$r \not\in S(a) \cup S(b)$ could try to take any $S(a) \cup S(b)$, but how to dominate $r$?

Need a "extra" dominating set from one of $T(a)$ and $T(b)$.

Question: how to parameterize these subproblems?
Minimum Dominating Set, in Trees

**question**: copy&paste from MIS on trees?

Let $T(v)$ denote the subtree rooted at $v \in V$, and let $S(v)$ be any minimum weight dominating set for $T(v)$. Building $S(r)$:

- $r \in S$:
  - could take any $S(a) \cup S(b) \cup \{r\}$

- If we cover $r$ then $a, b$ do not need to be covered — only need a "mostly" dominating set on $T(a)$ and $T(b)$

$r/ \in S$:

- could try to take any $S(a) \cup S(b)$, but how to dominate $r$?
  - need a "extra" dominating set from one of $T(a)$ and $T(b)$

**question**: how to parameterize these subproblems?
question: copy & paste from MIS on trees?
question: copy & paste from MIS on trees?

Let $T(v)$ denote the subtree rooted at $v \in V$,
question: copy&paste from MIS on trees?

Let $T(v)$ denote the subtree rooted at $v \in V$, and let $S(v)$ be any minimum weight dominating set for $T(v)$.
question: copy & paste from MIS on trees?

building $S(r)$:

Let $T(v)$ denote the subtree rooted at $v \in V$, and let $S(v)$ be any minimum weight dominating set for $T(v)$. 

```plaintext
r
  /   \
 a     b
  / \
 c   d e   f
 /
 h i  g j
```
question: copy & paste from MIS on trees?

building $S(r)$:
- $r \in S$:

Let $T(v)$ denote the subtree rooted at $v \in V$, and let $S(v)$ be any minimum weight dominating set for $T(v)$. 
Minimum Dominating Set, in Trees

question: copy & paste from MIS on trees?

building $S(r)$:

- $r \in S$:
  - could take any $S(a) \cup S(b) \cup \{r\}$

Let $T(v)$ denote the subtree rooted at $v \in V$, and let $S(v)$ be any minimum weight dominating set for $T(v)$. 
Let $T(v)$ denote the subtree rooted at $v \in V$, and let $S(v)$ be any minimum weight dominating set for $T(v)$.

**question:** copy & paste from MIS on trees?

**building $S(r)$:**
- $r \in S$:
  - could take any $S(a) \cup S(b) \cup \{r\}$
  - *better*: if we cover $r$ then $a, b$ do not need to be covered

Let $T(v)$ denote the subtree rooted at $v \in V$, and let $S(v)$ be any minimum weight dominating set for $T(v)$. 
**question:** copy & paste from MIS on trees?

Let $T(v)$ denote the subtree rooted at $v \in V$, and let $S(v)$ be any minimum weight dominating set for $T(v)$.

**building $S(r)$:**

- $r \in S$:
  - could take any $S(a) \cup S(b) \cup \{r\}$
  - better: if we cover $r$ then $a, b$ do not need to be covered — only need a “mostly” dominating set on $T(a)$ and $T(b)$
question: copy & paste from MIS on trees?

Let $T(v)$ denote the subtree rooted at $v \in V$, and let $S(v)$ be any minimum weight dominating set for $T(v)$.

building $S(r)$:

- **$r \in S$:**
  - could take any $S(a) \cup S(b) \cup \{r\}$
  - better: if we cover $r$ then $a, b$ do not need to be covered — only need a “mostly” dominating set on $T(a)$ and $T(b)$

- **$r \notin S$:**

$$
\text{building } S(r):
\begin{align*}
  &r \in S: \\
  &\quad \text{could take any } S(a) \cup S(b) \cup \{r\} \\
  &\quad \text{better: if we cover } r \text{ then } a, b \text{ do not need to be covered — only need a “mostly” dominating set on } T(a) \text{ and } T(b) \\
  &r \notin S: 
\end{align*}
$$
Minimum Dominating Set, in Trees

**question:** copy & paste from MIS on trees?

**building** $S(r)$:

- $r \in S$:
  - could take any $S(a) \cup S(b) \cup \{r\}$
  - better: if we cover $r$ then $a, b$ do not need to be covered — only need a “mostly” dominating set on $T(a)$ and $T(b)$

- $r \notin S$:
  - could try to take any $S(a) \cup S(b)$,

Let $T(v)$ denote the subtree rooted at $v \in V$, and let $S(v)$ be any minimum weight dominating set for $T(v)$. 
question: copy & paste from MIS on trees?

Let $T(\nu)$ denote the subtree rooted at $\nu \in V$, and let $S(\nu)$ be any minimum weight dominating set for $T(\nu)$.

building $S(r)$:

- $r \in S$:
  - could take any $S(a) \cup S(b) \cup \{r\}$
  - better: if we cover $r$ then $a, b$ do not need to be covered — only need a “mostly” dominating set on $T(a)$ and $T(b)$

- $r \notin S$:
  - could try to take any $S(a) \cup S(b)$, but how to dominate $r$?
question: copy & paste from MIS on trees?

Let $T(v)$ denote the subtree rooted at $v \in V$, and let $S(v)$ be any minimum weight dominating set for $T(v)$.

building $S(r)$:

- $r \in S$:
  - could take any $S(a) \cup S(b) \cup \{r\}$
  - better: if we cover $r$ then $a, b$ do not need to be covered — only need a “mostly” dominating set on $T(a)$ and $T(b)$

- $r \notin S$:
  - could try to take any $S(a) \cup S(b)$, but how to dominate $r$?
  - need a “extra” dominating set from one of $T(a)$ and $T(b)$
**question:** copy & paste from MIS on trees?

Let $T(v)$ denote the subtree rooted at $v \in V$, and let $S(v)$ be any minimum weight dominating set for $T(v)$.

**building $S(r)$:**

- $r \in S$:
  - could take any $S(a) \cup S(b) \cup \{r\}$
  - *better:* if we cover $r$ then $a, b$ do not need to be covered — only need a “mostly” dominating set on $T(a)$ and $T(b)$

- $r \notin S$:
  - could try to take any $S(a) \cup S(b)$, but how to dominate $r$?
  - need a “extra” dominating set from one of $T(a)$ and $T(b)$

**question:**
Minimum Dominating Set, in Trees

**question:** copy & paste from MIS on trees?

**building** $S(r)$:

- $r \in S$:
  - could take any $S(a) \cup S(b) \cup \{r\}$
  - *better:* if we cover $r$ then $a, b$ do not need to be covered — only need a “mostly” dominating set on $T(a)$ and $T(b)$

- $r \notin S$:
  - could try to take any $S(a) \cup S(b)$, but how to dominate $r$?
  - need a “extra” dominating set from one of $T(a)$ and $T(b)$

**question:** how to parameterize these subproblems?

---

Let $T(v)$ denote the subtree rooted at $v \in V$, and let $S(v)$ be any minimum weight dominating set for $T(v)$.
Minimum Dominating Set, in Trees (II)

Definition
Let $T = (V, E)$ be a rooted tree with root $r$. A type-0 dominating set for $T$ is an actual dominating set. A type-1 dominating set for $T$ is an actual dominating set $S$ where $r \in S$. A type-2 dominating set for $T$ is a subset $S \subseteq V$ such that for all $v \in V \setminus \{r\}$, either $v \in S$ or $v$ has a neighbor $u \in N(v)$ with $u \in S$.

For $b \in \{0, 1, 2\}$, define $OPT_b$ to be the minimum weight dominating set for $T$ of $b$-type. Define $OPT_b(v)$ to be the $OPT_b$ for the subtree of $T$ rooted at $v$.

base case: $T$ has no vertices $\Rightarrow OPT_b(T) = 0$

extends gracefully by the following conventions:
for $S = \emptyset$, $\sum_{v \in S} f(v) = 0$
for $S = \emptyset$, $\min_{v \in S} f(v) = \infty$
Definition

Let $T = (V, E)$ be a rooted tree with root $r$. A type-0 dominating set for $T$ is an actual dominating set. A type-1 dominating set for $T$ is an actual dominating set $S$ where $r \in S$. A type-2 dominating set for $T$ is a subset $S \subseteq V$ such that for all $v \in V \setminus \{r\}$, either $v \in S$ or $v$ has a neighbor $u \in N(v)$ with $u \in S$.

For $b \in \{0, 1, 2\}$, define $OPT_b$ to be the minimum weight dominating set for $T$ of $b$-type. Define $OPT_b(v)$ to be the $OPT_b$ for the subtree of $T$ rooted at $v$.

base case: $T$ has no vertices $\Rightarrow OPT_b(T) = 0$

extends gracefully by the following conventions:

for $S = \emptyset$, $\sum_{v \in S} f(v) = 0$

for $S = \emptyset$, $\min_{v \in S} f(v) = \infty$
Definition

Let \( T = (V, E) \) be a rooted tree with root \( r \).
Minimum Dominating Set, in Trees (II)

Definition

Let \( T = (V, E) \) be a rooted tree with root \( r \).

- A **type-0** dominating set for \( T \) is an actual dominating set.

For \( b \in \{0, 1, 2\} \), define \( \text{OPT}_b \) to be the minimum weight dominating set for \( T \) of \( b \)-type.

Define \( \text{OPT}_b(v) \) to be the \( \text{OPT}_b \) for the subtree of \( T \) rooted at \( v \).

**base case:** \( T \) has no vertices \( \Rightarrow \text{OPT}_b(T) = 0 \)

extends gracefully by the following conventions:

for \( S = \emptyset \), \( \sum_{v \in S} f(v) = 0 \)

for \( S = \emptyset \), \( \min_{v \in S} f(v) = \infty \)
Minimum Dominating Set, in Trees (II)

Definition

Let $T = (V, E)$ be a rooted tree with root $r$.

- A **type-0** dominating set for $T$ is an actual dominating set.
- A **type-1** dominating set for $T$ is an actual dominating set $S$ where $r \in S$.

For $b \in \{0, 1, 2\}$, define $\text{OPT}_b$ to be the minimum weight dominating set for $T$ of $b$-type.

Define $\text{OPT}_b(v)$ to be the $\text{OPT}_b$ for the subtree of $T$ rooted at $v$.

**base case:** $T$ has no vertices $\Rightarrow \text{OPT}_b(T) = 0$

extends gracefully by the following conventions:

- for $S = \emptyset$, $\sum_{v \in S} f(v) = 0$
- for $S = \emptyset$, $\min_{v \in S} f(v) = \infty$
Definition

Let $T = (V, E)$ be a rooted tree with root $r$.

- A type-0 dominating set for $T$ is an actual dominating set.
- A type-1 dominating set for $T$ is an actual dominating set $S$ where $r \in S$.
- A type-2 dominating set for $T$ is a subset $S \subseteq V$ such that for all $v \in V \setminus \{r\}$, either $v \in S$ or $v$ has a neighbor $u \in N(v)$ with $u \in S$. 

For $b \in \{0, 1, 2\}$, define $OPT^b$ to be the minimum weight dominating set for $T$ of $b$-type.

$OPT^b(v)$ to be the $OPT^b$ for the subtree of $T$ rooted at $v$.

base case: $T$ has no vertices $\Rightarrow OPT^b(T) = 0$

extends gracefully by the following conventions:

- for $S = \emptyset$, $\sum_{v \in S} f(v) = 0$
- for $S = \emptyset$, $\min_{v \in S} f(v) = \infty$
Minimum Dominating Set, in Trees (II)

Definition

Let $T = (V, E)$ be a rooted tree with root $r$.

- A **type-0** dominating set for $T$ is an actual dominating set.
- A **type-1** dominating set for $T$ is an actual dominating set $S$ where $r \in S$.
- A **type-2** dominating set for $T$ is a subset $S \subseteq V$.
Definition

Let $T = (V, E)$ be a rooted tree with root $r$.

- A **type-0** dominating set for $T$ is an actual dominating set.
- A **type-1** dominating set for $T$ is an actual dominating set $S$ where $r \in S$.
- A **type-2** dominating set for $T$ is a subset $S \subseteq V$ such that for all $v \in V$,
Minimum Dominating Set, in Trees (II)

Definition

Let $T = (V, E)$ be a rooted tree with root $r$.

- A **type-0** dominating set for $T$ is an actual dominating set.
- A **type-1** dominating set for $T$ is an actual dominating set $S$ where $r \in S$.
- A **type-2** dominating set for $T$ is a subset $S \subseteq V$ such that for all $v \in V \setminus \{r\}$, either $v \in S$ or $v$ has a neighbor $u \in N(v)$ with $u \in S$. 

For $b \in \{0, 1, 2\}$, define $OPT_b$ to be the minimum weight dominating set for $T$ of $b$-type. Define $OPT_b(v)$ to be the $OPT_b$ for the subtree of $T$ rooted at $v$.

**base case:** $T$ has no vertices $\Rightarrow OPT_b(T) = 0$

Extends gracefully by the following conventions:

- for $S = \emptyset$, $\sum_{v \in S} f(v) = 0$
- for $S = \emptyset$, $\min_{v \in S} f(v) = \infty$
Definition

Let $T = (V, E)$ be a rooted tree with root $r$.

- A **type-0** dominating set for $T$ is an actual dominating set.
- A **type-1** dominating set for $T$ is an actual dominating set $S$ where $r \in S$.
- A **type-2** dominating set for $T$ is a subset $S \subseteq V$ such that for all $v \in V \setminus \{r\}$, either $v \in S$ or $v$ has a neighbor $u \in N(v)$ with $u \in S$. 

For $b \in \{0, 1, 2\}$, define $\text{OPT}_b$ to be the minimum weight dominating set for $T$ of $b$-type. Define $\text{OPT}_b(v)$ to be the $\text{OPT}_b$ for the subtree of $T$ rooted at $v$.

**Base case:** $T$ has no vertices $\Rightarrow \text{OPT}_b(T) = 0$

Extends gracefully by the following conventions:

- For $S = \emptyset$, $\sum_{v \in S} f(v) = 0$
- For $S = \emptyset$, $\min_{v \in S} f(v) = \infty$
Definition

Let $T = (V, E)$ be a rooted tree with root $r$.

- A **type-0** dominating set for $T$ is an actual dominating set.
- A **type-1** dominating set for $T$ is an actual dominating set $S$ where $r \in S$.
- A **type-2** dominating set for $T$ is a subset $S \subseteq V$ such that for all $v \in V \setminus \{r\}$, either $v \in S$ or $v$ has a neighbor $u \in N(v)$ with $u \in S$.

For $b \in \{0, 1, 2\}$,
Definition

Let $T = (V, E)$ be a rooted tree with root $r$.

- A **type-0** dominating set for $T$ is an actual dominating set.
- A **type-1** dominating set for $T$ is an actual dominating set $S$ where $r \in S$.
- A **type-2** dominating set for $T$ is a subset $S \subseteq V$ such that for all $v \in V \setminus \{r\}$, either $v \in S$ or $v$ has a neighbor $u \in N(v)$ with $u \in S$.

For $b \in \{0, 1, 2\}$, define $\text{OPT}_b$ to be the minimum weight dominating set for $T$ of $b$-type.
### Definition

Let $T = (V, E)$ be a rooted tree with root $r$.

- A **type-0** dominating set for $T$ is an actual dominating set.
- A **type-1** dominating set for $T$ is an actual dominating set $S$ where $r \in S$.
- A **type-2** dominating set for $T$ is a subset $S \subseteq V$ such that for all $v \in V \setminus \{r\}$, either $v \in S$ or $v$ has a neighbor $u \in N(v)$ with $u \in S$.

For $b \in \{0, 1, 2\}$, define $\text{OPT}_b$ to be the minimum weight dominating set for $T$ of $b$-type. Define $\text{OPT}_b(v)$ to be the $\text{OPT}_b$ for the subtree of $T$ rooted at $v$. 

Definition

Let $T = (V, E)$ be a rooted tree with root $r$.

- A **type-0** dominating set for $T$ is an actual dominating set.
- A **type-1** dominating set for $T$ is an actual dominating set $S$ where $r \in S$.
- A **type-2** dominating set for $T$ is a subset $S \subseteq V$ such that for all $v \in V \setminus \{r\}$, either $v \in S$ or $v$ has a neighbor $u \in N(v)$ with $u \in S$.

For $b \in \{0, 1, 2\}$, define $\text{OPT}_b$ to be the minimum weight dominating set for $T$ of $b$-type. Define $\text{OPT}_b(v)$ to be the $\text{OPT}_b$ for the subtree of $T$ rooted at $v$.

**base case:**
Minimum Dominating Set, in Trees (II)

<table>
<thead>
<tr>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Let ( T = (V, E) ) be a rooted tree with root ( r ).</td>
</tr>
<tr>
<td>- A <strong>type-0</strong> dominating set for ( T ) is an actual dominating set.</td>
</tr>
<tr>
<td>- A <strong>type-1</strong> dominating set for ( T ) is an actual dominating set ( S ) where ( r \in S ).</td>
</tr>
<tr>
<td>- A <strong>type-2</strong> dominating set for ( T ) is a subset ( S \subseteq V ) such that for all ( v \in V \setminus {r} ), either ( v \in S ) or ( v ) has a neighbor ( u \in N(v) ) with ( u \in S ).</td>
</tr>
</tbody>
</table>

For \( b \in \{0, 1, 2\} \), define \( \text{OPT}_b \) to be the minimum weight dominating set for \( T \) of \( b \)-type. Define \( \text{OPT}_b(v) \) to be the \( \text{OPT}_b \) for the subtree of \( T \) rooted at \( v \).

**base case:**
- \( T \) has no vertices
Minimum Dominating Set, in Trees (II)

### Definition

Let $T = (V, E)$ be a rooted tree with root $r$.

- A **type-0** dominating set for $T$ is an actual dominating set.
- A **type-1** dominating set for $T$ is an actual dominating set $S$ where $r \in S$.
- A **type-2** dominating set for $T$ is a subset $S \subseteq V$ such that for all $v \in V \setminus \{r\}$, either $v \in S$ or $v$ has a neighbor $u \in N(v)$ with $u \in S$.

For $b \in \{0, 1, 2\}$, define $\text{OPT}_b$ to be the minimum weight dominating set for $T$ of $b$-type. Define $\text{OPT}_b(v)$ to be the $\text{OPT}_b$ for the subtree of $T$ rooted at $v$.

**base case:**

- $T$ has no vertices $\implies \text{OPT}_b(T) = 0$
Definition

Let $T = (V, E)$ be a rooted tree with root $r$.

- A **type-0** dominating set for $T$ is an actual dominating set.
- A **type-1** dominating set for $T$ is an actual dominating set $S$ where $r \in S$.
- A **type-2** dominating set for $T$ is a subset $S \subseteq V$ such that for all $v \in V \setminus \{r\}$, either $v \in S$ or $v$ has a neighbor $u \in N(v)$ with $u \in S$.

For $b \in \{0, 1, 2\}$, define $\text{OPT}_b$ to be the minimum weight dominating set for $T$ of $b$-type. Define $\text{OPT}_b(v)$ to be the $\text{OPT}_b$ for the subtree of $T$ rooted at $v$.

**base case:**

- $T$ has no vertices $\implies \text{OPT}_b(T) = 0$
- extends gracefully by the following conventions:
Minimum Dominating Set, in Trees (II)

Definition

Let $T = (V, E)$ be a rooted tree with root $r$.

- A **type-0** dominating set for $T$ is an actual dominating set.
- A **type-1** dominating set for $T$ is an actual dominating set $S$ where $r \in S$.
- A **type-2** dominating set for $T$ is a subset $S \subseteq V$ such that for all $v \in V \setminus \{r\}$, either $v \in S$ or $v$ has a neighbor $u \in N(v)$ with $u \in S$.

For $b \in \{0, 1, 2\}$, define $\text{OPT}_b$ to be the minimum weight dominating set for $T$ of $b$-type. Define $\text{OPT}_b(v)$ to be the $\text{OPT}_b$ for the subtree of $T$ rooted at $v$.

**base case:**

- $T$ has no vertices $\implies \text{OPT}_b(T) = 0$
- extends gracefully by the following conventions:
  - for $S = \emptyset$, 


Minimum Dominating Set, in Trees (II)

Definition

Let $T = (V, E)$ be a rooted tree with root $r$.

- A **type-0** dominating set for $T$ is an actual dominating set.
- A **type-1** dominating set for $T$ is an actual dominating set $S$ where $r \in S$.
- A **type-2** dominating set for $T$ is a subset $S \subseteq V$ such that for all $v \in V \setminus \{r\}$, either $v \in S$ or $v$ has a neighbor $u \in N(v)$ with $u \in S$.

For $b \in \{0, 1, 2\}$, define $\text{OPT}_b$ to be the minimum weight dominating set for $T$ of $b$-type. Define $\text{OPT}_b(v)$ to be the $\text{OPT}_b$ for the subtree of $T$ rooted at $v$.

**base case:**

- $T$ has no vertices $\implies \text{OPT}_b(T) = 0$
- extends gracefully by the following conventions:
  - for $S = \emptyset$, $\sum_{v \in S} f(v) = 0$
Minimum Dominating Set, in Trees (II)

Definition

Let $T = (V, E)$ be a rooted tree with root $r$.

- A **type-0** dominating set for $T$ is an actual dominating set.
- A **type-1** dominating set for $T$ is an actual dominating set $S$ where $r \in S$.
- A **type-2** dominating set for $T$ is a subset $S \subseteq V$ such that for all $v \in V \setminus \{r\}$, either $v \in S$ or $v$ has a neighbor $u \in N(v)$ with $u \in S$.

For $b \in \{0, 1, 2\}$, define $OPT_b$ to be the minimum weight dominating set for $T$ of $b$-type. Define $OPT_b(v)$ to be the $OPT_b$ for the subtree of $T$ rooted at $v$.

**base case:**

- $T$ has no vertices $\implies OPT_b(T) = 0$
- extends gracefully by the following conventions:
  - for $S = \emptyset$, $\sum_{v \in S} f(v) = 0$
  - for $S = \emptyset$, $\min_{v \in S} f(v) = \infty$
Minimum Dominating Set, in Trees (II)

**Definition**

Let $T = (V, E)$ be a rooted tree with root $r$.

- A **type-0** dominating set for $T$ is an actual dominating set.
- A **type-1** dominating set for $T$ is an actual dominating set $S$ where $r \in S$.
- A **type-2** dominating set for $T$ is a subset $S \subseteq V$ such that for all $v \in V \setminus \{r\}$, either $v \in S$ or $v$ has a neighbor $u \in N(v)$ with $u \in S$.

For $b \in \{0, 1, 2\}$, define $\text{OPT}_b$ to be the minimum weight dominating set for $T$ of $b$-type. Define $\text{OPT}_b(v)$ to be the $\text{OPT}_b$ for the subtree of $T$ rooted at $v$.

**base case:**

- $T$ has no vertices $\implies \text{OPT}_b(T) = 0$
- extends gracefully by the following conventions:
  - for $S = \emptyset$, $\sum_{v \in S} f(v) = 0$
  - for $S = \emptyset$, $\min_{v \in S} f(v) = \infty$
Minimum Dominating Set, in Trees (III)

A rooted tree with root $r$. $T(v)$ is subtree rooted at $v$.

**Type-0:**

- Regular dominating set

**Type-1:**

- Dominating set which includes root $r$

**Type-2:**

- Dominating set which is relaxed at root $r$

**Lemma:**

$$\text{OPT}_0(r) = \min \left\{ \sum_{v \in N(r)} \text{OPT}_2(v) \right\}$$

$$+ w(r)$$

$$\min_{v \in N(r)} \left( \text{OPT}_1(v) + \sum_{u \in N(r)} \{v\} \text{OPT}_0(u) \right).$$

**Proof.**

- In optimum $S$, $r \in S$
- In optimum $S$, $r \not\in S$ and $r$ dominated by child $v \in S$. 

24 / 29
Minimum Dominating Set, in Trees (III)

$T$ rooted tree with root $r$. 

Lemma

$$\text{OPT}_0(r) = \min \left\{ \sum_{v \in N(r)} \text{OPT}_2(v) \right\} + w(r) \quad \min_{v \in N(r)} \left( \text{OPT}_1(v) + \sum_{u \in N(r)} \{v\} \text{OPT}_0(u) \right)$$
Minimum Dominating Set, in Trees (III)

$T$ rooted tree with root $r$. $T(v)$ is subtree rooted at $v$. 

$OPT_0(r) = \min \left\{ \sum_{v \in N(r)} OPT_2(v) + w(r), \min_{v \in N(r)} (OPT_1(v) + \sum_{u \in N(r)} \{v\} OPT_0(u)) \right\}$.
Minimum Dominating Set, in Trees (III)

\( T \) rooted tree with root \( r \). \( T(v) \) is subtree rooted at \( v \).

- **type-0:**
Minimum Dominating Set, in Trees (III)

$T$ rooted tree with root $r$. $T(v)$ is subtree rooted at $v$.

- **type-0**: regular dominating set

Proof.
in optimum $S$, $r \in S$
in optimum $S$, $r \not\in S$ and $r$ dominated by child $v \in S$
Minimum Dominating Set, in Trees (III)

$T$ rooted tree with root $r$. $T(v)$ is subtree rooted at $v$.

- **type-0**: regular dominating set
- **type-1**: 

\[
\text{Lemma} \quad \text{OPT}_0(r) = \min \left\{ \sum_{v \in N(r)} \text{OPT}_2(v) + w(r), \min_{v \in N(r)} \left( \text{OPT}_1(v) + \sum_{u \in N(r) \setminus \{v\}} \text{OPT}_0(u) \right) \right\}.
\]

Proof.
Minimum Dominating Set, in Trees (III)

$T$ rooted tree with root $r$. $T(v)$ is subtree rooted at $v$.

- **type-0**: regular dominating set
- **type-1**: dominating set which includes root $r$

**Lemma**

$\text{OPT}_0(r) = \min \left\{ \sum_{v \in N(r)} \text{OPT}_2(v) + w(r), \min_{v \in N(r)} (\text{OPT}_1(v) + \sum_{u \in N(r) \setminus \{v\}} \text{OPT}_0(u)) \right\}$.

**Proof.**

In optimum $S$, $r \in S$ in optimum $S$, $r \not\in S$ and $r$ dominated by child $v \in S$. 

24 / 29
Minimum Dominating Set, in Trees (III)

$T$ rooted tree with root $r$. $T(v)$ is subtree rooted at $v$.

- **type-0**: regular dominating set
- **type-1**: dominating set which includes root $r$
- **type-2**: 

Lemma

$$\text{OPT}_0(r) = \min \left\{ \sum_{v \in N(r)} \text{OPT}_2(v) + w(r), \min_{v \in N(r)} \left( \text{OPT}_1(v) + \sum_{u \in N(r) \setminus \{v\}} \text{OPT}_0(u) \right) \right\}$$

Proof.

In optimum $S$, $r \in S$.

In optimum $S$, $r \notin S$ and $r$ dominated by child $v \in S$. 


Minimum Dominating Set, in Trees (III)

$T$ rooted tree with root $r$. $T(v)$ is subtree rooted at $v$.

- **type-0**: regular dominating set
- **type-1**: dominating set which includes root $r$
- **type-2**: dominating set which is relaxed at root $r$

Lemma

$$\text{OPT}_0(r) = \min \begin{cases} 
\sum_{v \in N(r)} \text{OPT}_2(v) + w(r) \\
\min_{v \in N(r)} \left( \text{OPT}_1(v) + \sum_{u \in N(r) \setminus \{v\}} \text{OPT}_0(u) \right)
\end{cases}$$

Proof.
in optimum $S$, $r \in S$
in optimum $S$, $r \not\in S$ and $r$ dominated by child $v \in S$
Minimum Dominating Set, in Trees (III)

Let $T$ be a rooted tree with root $r$. $T(v)$ is the subtree rooted at $v$.

- **type-0**: regular dominating set
- **type-1**: dominating set which includes root $r$
- **type-2**: dominating set which is relaxed at root $r$

**Lemma**

$$\text{OPT}_0(r) = \min \left\{ \sum_{v \in N(r)} \text{OPT}_2(v) + w(r), \min_{v \in N(r)} \left( \text{OPT}_1(v) + \sum_{u \in N(r) \setminus \{v\}} \text{OPT}_0(u) \right) \right\}$$
Minimum Dominating Set, in Trees (III)

$T$ rooted tree with root $r$. $T(v)$ is subtree rooted at $v$.
- **type-0**: regular dominating set
- **type-1**: dominating set which includes root $r$
- **type-2**: dominating set which is relaxed at root $r$

Lemma

$$\text{OPT}_0(r) = \min$$
Minimum Dominating Set, in Trees (III)

$T$ rooted tree with root $r$. $T(v)$ is subtree rooted at $v$.

- **type-0**: regular dominating set
- **type-1**: dominating set which includes root $r$
- **type-2**: dominating set which is relaxed at root $r$

**Lemma**

\[
\text{OPT}_0(r) = \min \left\{ \left( \sum_{v \in N(r)} \text{OPT}_2(v) \right) + w(r) \right\} 
\]

Proof. In optimum $S$, $r \in S$ in optimum $S$, $r \notin S$ and $r$ dominated by child $v \in S$. 

\[
\text{OPT}_0(r) = \min \left\{ \left( \sum_{v \in N(r)} \text{OPT}_1(v) \right) + \sum_{u \in N(r) \setminus \{v\}} \text{OPT}_0(u) \right\}
\]
T rooted tree with root r. T(ν) is subtree rooted at ν.

- **type-0**: regular dominating set
- **type-1**: dominating set which includes root r
- **type-2**: dominating set which is relaxed at root r

**Lemma**

\[
\text{OPT}_0(r) = \min \left\{ \left( \sum_{ν \in N(r)} \text{OPT}_2(ν) \right) \right\}
\]
Minimum Dominating Set, in Trees (III)

$T$ rooted tree with root $r$. $T(v)$ is subtree rooted at $v$.

- **type-0**: regular dominating set
- **type-1**: dominating set which includes root $r$
- **type-2**: dominating set which is relaxed at root $r$

**Lemma**

$$
\text{OPT}_0(r) = \min \left\{ \left( \sum_{v \in N(r)} \text{OPT}_2(v) \right) + w(r) \right\}
$$
$T$ rooted tree with root $r$. $T(v)$ is subtree rooted at $v$.

- **type-0**: regular dominating set
- **type-1**: dominating set which includes root $r$
- **type-2**: dominating set which is relaxed at root $r$

**Lemma**

$$
OPT_0(r) = \min \left\{ \left( \sum_{v \in N(r)} OPT_2(v) \right) + w(r), \right.
\left. \min_{v \in N(r)} \right\}
$$
Minimum Dominating Set, in Trees (III)

$T$ rooted tree with root $r$. $T(v)$ is subtree rooted at $v$.

- **type-0**: regular dominating set
- **type-1**: dominating set which includes root $r$
- **type-2**: dominating set which is relaxed at root $r$

**Lemma**

$$\text{OPT}_0(r) = \min \left\{ \left( \sum_{v \in N(r)} \text{OPT}_2(v) \right) + w(r), \min_{v \in N(r)} \left( \text{OPT}_1(v) \right) \right\}$$
Minimum Dominating Set, in Trees (III)

$T$ rooted tree with root $r$. $T(v)$ is subtree rooted at $v$.

- **type-0**: regular dominating set
- **type-1**: dominating set which includes root $r$
- **type-2**: dominating set which is relaxed at root $r$

**Lemma**

$$OPT_0(r) = \min \begin{cases} 
\left( \sum_{v \in N(r)} OPT_2(v) \right) + w(r) \\
\min_{v \in N(r)} \left( OPT_1(v) + \sum_{u \in N(r) \setminus \{v\}} OPT_0(u) \right)
\end{cases}$$
Minimum Dominating Set, in Trees (III)

$T$ rooted tree with root $r$. $T(v)$ is subtree rooted at $v$.

- **type-0**: regular dominating set
- **type-1**: dominating set which includes root $r$
- **type-2**: dominating set which is relaxed at root $r$

**Lemma**

$$
\text{OPT}_0(r) = \min \left\{ \left( \sum_{v \in N(r)} \text{OPT}_2(v) \right) + w(r) \right. \\
\left. \min_{v \in N(r)} \left( \text{OPT}_1(v) + \sum_{u \in N(r) \setminus \{v\}} \text{OPT}_0(u) \right) \right\}.
$$

Proof.
in optimum $S$, $r \in S$
in optimum $S$, $r \notin S$ and $r$ dominated by child $v \in S$
Minimum Dominating Set, in Trees (III)

Let $T$ be a rooted tree with root $r$. $T(v)$ is the subtree rooted at $v$.

- **type-0**: regular dominating set
- **type-1**: dominating set which includes root $r$
- **type-2**: dominating set which is relaxed at root $r$

**Lemma**

$$\text{OPT}_0(r) = \min \left\{ \left( \sum_{v \in N(r)} \text{OPT}_2(v) \right) + w(r) \right\}$$

$$\min_{v \in N(r)} \left( \text{OPT}_1(v) + \sum_{u \in N(r) \setminus \{v\}} \text{OPT}_0(u) \right)$$

**Proof.**
Minimum Dominating Set, in Trees (III)

$T$ rooted tree with root $r$. $T(v)$ is subtree rooted at $v$.

- **type-0**: regular dominating set
- **type-1**: dominating set which includes root $r$
- **type-2**: dominating set which is relaxed at root $r$

**Lemma**

$$OPT_0(r) = \min \begin{cases} \left( \sum_{v \in N(r)} OPT_2(v) \right) + w(r) \\ \min_{v \in N(r)} \left( OPT_1(v) + \sum_{u \in N(r) \setminus \{v\}} OPT_0(u) \right) \end{cases}.$$  

**Proof.**

- in optimum $S$, $r \in S$
Minimum Dominating Set, in Trees (III)

$T$ rooted tree with root $r$. $T(v)$ is subtree rooted at $v$.

- **type-0**: regular dominating set
- **type-1**: dominating set which includes root $r$
- **type-2**: dominating set which is relaxed at root $r$

**Lemma**

$$OPT_0(r) = \min \begin{cases} \left( \sum_{v \in N(r)} OPT_2(v) \right) + w(r) \\ \min_{v \in N(r)} \left( OPT_1(v) + \sum_{u \in N(r) \setminus \{v\}} OPT_0(u) \right) \end{cases}.$$  

**Proof.**

- in optimum $S$, $r \in S$
- in optimum $S$, $r \notin S$
Minimum Dominating Set, in Trees (III)

$T$ rooted tree with root $r$. $T(v)$ is subtree rooted at $v$.

- **type-0**: regular dominating set
- **type-1**: dominating set which includes root $r$
- **type-2**: dominating set which is relaxed at root $r$

**Lemma**

$$\text{OPT}_0(r) = \min \left\{ \left( \sum_{v \in N(r)} \text{OPT}_2(v) \right) + w(r) \right\}$$

**Proof.**

- in optimum $S$, $r \in S$
- in optimum $S$, $r \notin S$ and $r$ dominated by child $v \in S$
Minimum Dominating Set, in Trees (IV)

T is a rooted tree with root r. T(v) is subtree rooted at v.

- **type-0**: regular dominating set
- **type-1**: dominating set which includes root r
- **type-2**: dominating set which is relaxed at root r

**Lemma**

$$\text{OPT}_1(r) = \sum_{v \in N(r)} \text{OPT}_2(v) + w(r).$$

**Proof.**

In optimum S, r ∈ S.
Minimum Dominating Set, in Trees (IV)

$T$ rooted tree with root $r$. $T(v)$ is subtree rooted at $v$.

- **type-0**: regular dominating set
- **type-1**: dominating set which includes root $r$
- **type-2**: dominating set which is relaxed at root $r$

Lemma

$OPT_1(r) = \sum_{v \in N(r)} OPT_2(v) + w(r)$.

Proof.

In optimum $S$, $r \in S$. 

25 / 29
$T$ rooted tree with root $r$. $T(v)$ is subtree rooted at $v$.

- **type-0**: regular dominating set
- **type-1**: dominating set which includes root $r$
- **type-2**: dominating set which is relaxed at root $r$

Lemma

$\text{OPT}_1(r) = \left( \sum_{v \in \text{N}(r)} \text{OPT}_2(v) \right) + w(r)$.
Minimum Dominating Set, in Trees (IV)

$T$ rooted tree with root $r$. $T(v)$ is subtree rooted at $v$.

- **type-0**: regular dominating set
- **type-1**: dominating set which includes root $r$
- **type-2**: dominating set which is relaxed at root $r$

**Lemma**

$$\text{OPT}_1(r) =$$
Minimum Dominating Set, in Trees (IV)

$T$ rooted tree with root $r$. $T(v)$ is subtree rooted at $v$.

- **type-0**: regular dominating set
- **type-1**: dominating set which includes root $r$
- **type-2**: dominating set which is relaxed at root $r$

**Lemma**

\[
\text{OPT}_1(r) = \left( \sum_{v \in N(r)} \right)
\]
Minimum Dominating Set, in Trees (IV)

$T$ rooted tree with root $r$. $T(v)$ is subtree rooted at $v$.

- **type-0**: regular dominating set
- **type-1**: dominating set which includes root $r$
- **type-2**: dominating set which is relaxed at root $r$

**Lemma**

$$\text{OPT}_1(r) = \left( \sum_{v \in N(r)} \text{OPT}_2(v) \right)$$
Minimum Dominating Set, in Trees (IV)

$T$ rooted tree with root $r$. $T(v)$ is subtree rooted at $v$.

- **type-0**: regular dominating set
- **type-1**: dominating set which includes root $r$
- **type-2**: dominating set which is relaxed at root $r$

**Lemma**

$$\text{OPT}_1(r) = \left( \sum_{v \in N(r)} \text{OPT}_2(v) \right) + w(r).$$
Minimum Dominating Set, in Trees (IV)

$T$ rooted tree with root $r$. $T(v)$ is subtree rooted at $v$.

- **type-0**: regular dominating set
- **type-1**: dominating set which includes root $r$
- **type-2**: dominating set which is relaxed at root $r$

**Lemma**

$$\text{OPT}_1(r) = \left( \sum_{v \in N(r)} \text{OPT}_2(v) \right) + w(r).$$

**Proof.**

In optimum $S$, $r \in S$. 

Minimum Dominating Set, in Trees (V)

A rooted tree with root $r$. $T(v)$ is the subtree rooted at $v$.

**Type-0**: Regular dominating set

**Type-1**: Dominating set which includes root $r$

**Type-2**: Dominating set which is relaxed at root $r$

**Lemma**

$$\text{OPT}_2(r) = \min \left\{ \sum_{v \in N(r)} \text{OPT}_2(v) + w(r) \sum_{v \in N(r)} \text{OPT}_0(v) \right\}.$$ 

**Proof.**

In optimum $S$, $r \in S$, in optimum $S$, $r / \in S$ and $r$ does not need to be dominated by children.
Minimum Dominating Set, in Trees (V)

$T$ rooted tree with root $r$. $T(v)$ is subtree rooted at $v$.

- **type-0**: regular dominating set
- **type-1**: dominating set which includes root $r$
- **type-2**: dominating set which is relaxed at root $r$

$$\text{Lemma} \quad \text{OPT}_2(r) = \min \left\{ \sum_{v \in N(r)} \text{OPT}_2(v) + w(r) \sum_{v \in N(r)} \text{OPT}_0(v) \right\}.$$ 

Proof.

in optimum $S$, $r \in S$
in optimum $S$, $r \not\in S$
and $r$ does not need to be dominated by children
Minimum Dominating Set, in Trees (V)

$T$ rooted tree with root $r$. $T(v)$ is subtree rooted at $v$.

- **type-0**: regular dominating set
- **type-1**: dominating set which includes root $r$
- **type-2**: dominating set which is relaxed at root $r$

**Lemma**

$$\text{OPT}_2(r) = \min \{ \sum_{v \in N(r)} \text{OPT}_2(v) + w(r), \sum_{v \in N(r)} \text{OPT}_0(v) \}.$$
Minimum Dominating Set, in Trees (V)

$T$ rooted tree with root $r$. $T(v)$ is subtree rooted at $v$.

- **type-0**: regular dominating set
- **type-1**: dominating set which includes root $r$
- **type-2**: dominating set which is relaxed at root $r$

**Lemma**

$OPT_2(r) = \min \left\{ \sum_{v \in N(r)} OPT_2(v) + w(r), \sum_{v \in N(r)} OPT_0(v) \right\}$

Proof.
in optimum $S$, $r \in S$
in optimum $S$, $r \not\in S$ and $r$ does not need to be dominated by children
Minimum Dominating Set, in Trees (V)

$T$ rooted tree with root $r$. $T(v)$ is subtree rooted at $v$.
- **type-0**: regular dominating set
- **type-1**: dominating set which includes root $r$
- **type-2**: dominating set which is relaxed at root $r$

**Lemma**

$$\text{OPT}_2(r) = \min \left\{ \left(\sum_{v \in N(r)} \right) \right\}$$
Minimum Dominating Set, in Trees (V)

$T$ rooted tree with root $r$. $T(v)$ is subtree rooted at $v$.

- **type-0**: regular dominating set
- **type-1**: dominating set which includes root $r$
- **type-2**: dominating set which is relaxed at root $r$

**Lemma**

\[
\text{OPT}_2(r) = \min \left\{ \left( \sum_{v \in N(r)} \text{OPT}_2(v) \right) \right\}
\]
Minimum Dominating Set, in Trees (V)

$T$ rooted tree with root $r$. $T(v)$ is subtree rooted at $v$.

- **type-0**: regular dominating set
- **type-1**: dominating set which includes root $r$
- **type-2**: dominating set which is relaxed at root $r$

**Lemma**

$$\text{OPT}_2(r) = \min \left\{ \left( \sum_{v \in N(r)} \text{OPT}_2(v) \right) + w(r) \right\}$$
Minimum Dominating Set, in Trees (V)

$T$ rooted tree with root $r$. $T(v)$ is subtree rooted at $v$.

- **type-0**: regular dominating set
- **type-1**: dominating set which includes root $r$
- **type-2**: dominating set which is relaxed at root $r$

**Lemma**

\[
\text{OPT}_2(r) = \min \left\{ \left( \sum_{v \in N(r)} \text{OPT}_2(v) \right) + w(r) \right\}
\]
Minimum Dominating Set, in Trees (V)

$T$ rooted tree with root $r$. $T(v)$ is subtree rooted at $v$.

- **type-0**: regular dominating set
- **type-1**: dominating set which includes root $r$
- **type-2**: dominating set which is relaxed at root $r$

**Lemma**

\[
\text{OPT}_2(r) = \min \left\{ \left( \sum_{v \in N(r)} \text{OPT}_2(v) \right) + w(r), \sum_{v \in N(r)} \text{OPT}_0(v) \right\}.
\]
Minimum Dominating Set, in Trees (V)

$T$ rooted tree with root $r$. $T(v)$ is subtree rooted at $v$.

- **type-0**: regular dominating set
- **type-1**: dominating set which includes root $r$
- **type-2**: dominating set which is relaxed at root $r$

**Lemma**

\[ \text{OPT}_2(r) = \min \left\{ \left( \sum_{v \in N(r)} \text{OPT}_2(v) \right) + w(r), \sum_{v \in N(r)} \text{OPT}_0(v) \right\}. \]

**Proof.**
Minimum Dominating Set, in Trees (V)

$T$ rooted tree with root $r$. $T(v)$ is subtree rooted at $v$.

- **type-0**: regular dominating set
- **type-1**: dominating set which includes root $r$
- **type-2**: dominating set which is relaxed at root $r$

**Lemma**

$$\text{OPT}_2(r) = \min \left\{ \left( \sum_{v \in N(r)} \text{OPT}_2(v) \right) + w(r) \right\}.$$

**Proof.**

- in optimum $S$, $r \in S$
Minimum Dominating Set, in Trees (V)

A rooted tree $T$ with root $r$. $T(v)$ is subtree rooted at $v$.

- **type-0**: regular dominating set
- **type-1**: dominating set which includes root $r$
- **type-2**: dominating set which is relaxed at root $r$

**Lemma**

$$\text{OPT}_2(r) = \min \left\{ \left( \sum_{v \in N(r)} \text{OPT}_2(v) \right) + w(r), \sum_{v \in N(r)} \text{OPT}_0(v) \right\}.$$ 

**Proof.**

- in optimum $S$, $r \in S$
- in optimum $S$, $r \notin S$
Minimum Dominating Set, in Trees (V)

$T$ rooted tree with root $r$. $T(v)$ is subtree rooted at $v$.

- **type-0**: regular dominating set
- **type-1**: dominating set which includes root $r$
- **type-2**: dominating set which is relaxed at root $r$

Lemma

$$\text{OPT}_2(r) = \min \left\{ \left( \sum_{v \in N(r)} \text{OPT}_2(v) \right) + w(r), \sum_{v \in N(r)} \text{OPT}_0(v) \right\}.$$ 

Proof.

- in optimum $S$, $r \in S$
- in optimum $S$, $r \notin S$ and $r$ does not need to be dominated by children
Minimum Dominating Set, in Trees (VI)

A rooted tree with root $r$.

Subproblems:
- Type-0: Regular dominating set
- Type-1: Dominating set which includes root $r$
- Type-2: Dominating set which is relaxed at root $r$

Recursion:

$$
\text{OPT}_0(r) = \min \begin{cases} 
\left( \sum_{v \in N(r)} \text{OPT}_2(v) \right) + w(r) \\
\min_{v \in N(r)} \left( \text{OPT}_1(v) + \sum_{u \in N(r) \setminus \{v\}} \text{OPT}_0(u) \right)
\end{cases}
$$

$$
\text{OPT}_1(r) = \left( \sum_{v \in N(r)} \text{OPT}_2(v) \right) + w(r)
$$

$$
\text{OPT}_2(r) = \min \begin{cases} 
\left( \sum_{v \in N(r)} \text{OPT}_2(v) \right) + w(r) \\
\sum_{v \in N(r)} \text{OPT}_0(v)
\end{cases}
$$

$\text{OPT}_0(r)$ is the desired answer.

Recursive algorithm:
- 3 $n$ subproblems
- Can implicitly memoize
- Naively $O(n)$ work per node
- Can optimize to $O(n)$ total work as with MIS on trees

Iterative algorithm:
- Follow post-order traversal of rooted tree to satisfy dependencies
- Optimize analysis to obtain $O(n)$ total work
- Details are an exercise
Minimum Dominating Set, in Trees (VI)

$T$ rooted tree with root $r$. 

...
$T$ rooted tree with root $r$.

**subproblems:**

\[
\text{OPT}_0(r) = \min \left\{ \sum_{v \in N(r)} \text{OPT}_2(v) + w(r), \min_{v \in N(r)} (\text{OPT}_1(v) + \sum_{u \in N(r)} \left\{ v \right\} \text{OPT}_0(u)) \right\}
\]

\[
\text{OPT}_1(r) = \left( \sum_{v \in N(r)} \text{OPT}_2(v) \right) + w(r)
\]

\[
\text{OPT}_2(r) = \min \left\{ \sum_{v \in N(r)} \text{OPT}_2(v) + w(r), \sum_{v \in N(r)} \text{OPT}_0(v) \right\}
\]

**recursive algorithm:**

- 3·$n$ subproblems can implicitly memoize
- naively $O(n)$ work per node, can optimize to $O(n)$ total work as with MIS on trees

**iterative algorithm:**

- follow post-order traversal of rooted tree to satisfy dependencies
- optimize analysis to obtain $O(n)$ total work

details are an exercise
Minimum Dominating Set, in Trees (VI)

$T$ rooted tree with root $r$. 

**subproblems:**

- **type-0**: regular dominating set
- **type-1**: dominating set which includes root $r$
- **type-2**: dominating set which is relaxed at root $r$
Minimum Dominating Set, in Trees (VI)

$T$ rooted tree with root $r$.

subproblems:
- **type-0**: regular dominating set
- **type-1**: dominating set which includes root $r$
- **type-2**: dominating set which is relaxed at root $r$

recursion:

\[
\begin{align*}
\text{OPT}_0(r) &= \min \left\{ \sum_{v \in N(r)} \text{OPT}_2(v) + w(r) \right\} \\
\text{OPT}_1(r) &= \left( \sum_{v \in N(r)} \text{OPT}_2(v) \right) + w(r) \\
\text{OPT}_2(r) &= \min \left\{ \sum_{v \in N(r)} \text{OPT}_0(v) \right\}
\end{align*}
\]

OPT$_0$($r$) is desired answer.

recursive algorithm:
- 3 · $n$ subproblems can implicitly memoize
- can optimize to $O(n)$ work per node, as with MIS on trees

iterative algorithm:
- follow post-order traversal of rooted tree to satisfy dependencies
- optimize analysis to obtain $O(n)$ total work

details are an exercise
Minimum Dominating Set, in Trees (VI)

$T$ rooted tree with root $r$.

**subproblems:**

- **type-0**: regular dominating set
- **type-1**: dominating set which includes root $r$
- **type-2**: dominating set which is relaxed at root $r$

**recursion:**

- $\text{OPT}_0(r) = \text{min}$
Minimum Dominating Set, in Trees (VI)

$T$ rooted tree with root $r$.

**subproblems:**

- **type-0**: regular dominating set
- **type-1**: dominating set which includes root $r$
- **type-2**: dominating set which is relaxed at root $r$

**recursion:**

$$\text{OPT}_0(r) = \min \left\{ \left( \sum_{v \in N(r)} \right) \right\}$$
Minimum Dominating Set, in Trees (VI)

$T$ rooted tree with root $r$.

**subproblems:**
- **type-0**: regular dominating set
- **type-1**: dominating set which includes root $r$
- **type-2**: dominating set which is relaxed at root $r$

**recursion:**

$$
\text{OPT}_0(r) = \min \left\{ \left( \sum_{v \in N(r)} \text{OPT}_2(v) \right) \right\}
$$
Minimum Dominating Set, in Trees (VI)

$T$ rooted tree with root $r$.

**subproblems:**
- **type-0**: regular dominating set
- **type-1**: dominating set which includes root $r$
- **type-2**: dominating set which is relaxed at root $r$

**recursion:**

\[
\text{OPT}_0(r) = \min \left\{ \left( \sum_{v \in N(r)} \text{OPT}_2(v) \right) + w(r) \right\}
\]

\[
\text{OPT}_1(r) = \left( \sum_{v \in N(r)} \text{OPT}_2(v) \right)
\]

\[
\text{OPT}_2(r) = \min_{v \in N(r)} \text{OPT}_0(v)
\]

Recursive algorithm:
- $3n$ subproblems
- Can implicitly memoize
- Naively $O(n)$ work per node
- Can optimize to $O(n)$ total work as with MIS on trees

Iterative algorithm:
- Follow post-order traversal of rooted tree to satisfy dependencies
- Optimize analysis to obtain $O(n)$ total work

Details are an exercise.
Minimum Dominating Set, in Trees (VI)

A rooted tree with root $r$.

**Subproblems:**

- **type-0**: regular dominating set
- **type-1**: dominating set which includes root $r$
- **type-2**: dominating set which is relaxed at root $r$

**Recursion:**

$$OPT_0(r) = \min \left\{ \left( \sum_{v \in N(r)} OPT_2(v) \right) + w(r), \min_{v \in N(r)} \left( OPT_1(v) \right) \right\}$$
Minimum Dominating Set, in Trees (VI)

A rooted tree $T$ with root $r$.

**Subproblems:**

- **type-0**: regular dominating set
- **type-1**: dominating set which includes root $r$
- **type-2**: dominating set which is relaxed at root $r$

**Recursion:**

\[
\begin{align*}
\text{OPT}_0(r) &= \min \left\{ \left( \sum_{v \in N(r)} \text{OPT}_2(v) \right) + w(r) \right. \\
& \quad \left. \min_{v \in N(r)} \left( \text{OPT}_1(v) + \sum_{u \in N(r) \setminus \{v\}} \text{OPT}_0(u) \right) \right\}
\end{align*}
\]
Minimum Dominating Set, in Trees (VI)

$T$ rooted tree with root $r$.

subproblems:
- **type-0**: regular dominating set
- **type-1**: dominating set which includes root $r$
- **type-2**: dominating set which is relaxed at root $r$

recursion:

- $\text{OPT}_0(r) = \min \left\{ \left( \sum_{v \in N(r)} \text{OPT}_2(v) \right) + w(r) \right\}$
- $\text{OPT}_0(r) = \min_{v \in N(r)} \left( \text{OPT}_1(v) + \sum_{u \in N(r) \setminus \{v\}} \text{OPT}_0(u) \right)$
- $\text{OPT}_1(r) = \left( \sum_{v \in N(r)} \text{OPT}_2(v) \right) + w(r)$
Minimum Dominating Set, in Trees (VI)

A rooted tree with root $r$.

**Subproblems:**
- **type-0**: regular dominating set
- **type-1**: dominating set which includes root $r$
- **type-2**: dominating set which is relaxed at root $r$

**Recursion:**

- $\text{OPT}_0(r) = \min \left\{ \left( \sum_{v \in N(r)} \text{OPT}_2(v) \right) + w(r), \min_{v \in N(r)} \left( \text{OPT}_1(v) + \sum_{u \in N(r) \setminus \{v\}} \text{OPT}_0(u) \right) \right\}$
- $\text{OPT}_1(r) = \left( \sum_{v \in N(r)} \text{OPT}_2(v) \right) + w(r)$
- $\text{OPT}_2(r) = \min$
Minimum Dominating Set, in Trees (VI)

A rooted tree with root $r$.

Subproblems:
- **type-0**: regular dominating set
- **type-1**: dominating set which includes root $r$
- **type-2**: dominating set which is relaxed at root $r$

Recursion:

$$\text{OPT}_0(r) = \min \left\{ \left( \sum_{v \in N(r)} \text{OPT}_2(v) \right) + w(r), \min_{v \in N(r)} \left( \text{OPT}_1(v) + \sum_{u \in N(r) \setminus \{v\}} \text{OPT}_0(u) \right) \right\}$$

$$\text{OPT}_1(r) = \left( \sum_{v \in N(r)} \text{OPT}_2(v) \right) + w(r)$$

$$\text{OPT}_2(r) = \min \left\{ \left( \sum_{v \in N(r)} \text{OPT}_2(v) \right) + w(r) \right\}$$
Minimum Dominating Set, in Trees (VI)

A rooted tree with root \( r \).

**subproblems:**

- **type-0:** regular dominating set
- **type-1:** dominating set which includes root \( r \)
- **type-2:** dominating set which is relaxed at root \( r \)

**recursion:**

- \( \text{OPT}_0(r) = \min \left\{ \left( \sum_{v \in N(r)} \text{OPT}_2(v) \right) + w(r) \right\} \)
- \( \text{OPT}_1(r) = \left( \sum_{v \in N(r)} \text{OPT}_2(v) \right) + w(r) \)
- \( \text{OPT}_2(r) = \min \left\{ \left( \sum_{v \in N(r)} \text{OPT}_2(v) \right) + w(r) \right\} \)

\( \text{OPT}_0(r) \) is desired answer.

**recursive algorithm:**

- 3\( \cdot n \) subproblems can implicitly memoize
- naively \( O(n) \) work per node,
- can optimize to \( O(n) \) total work as with MIS on trees

**iterative algorithm:**

- follow post-order traversal of rooted tree to satisfy dependencies
- optimize analysis to obtain \( O(n) \) total work
- details are an exercise
Minimum Dominating Set, in Trees (VI)

$T$ rooted tree with root $r$.

**subproblems:**

- type-0: regular dominating set
- type-1: dominating set which includes root $r$
- type-2: dominating set which is relaxed at root $r$

**recursion:**

- $\text{OPT}_0(r) = \min \left\{ \left( \sum_{v \in N(r)} \text{OPT}_2(v) \right) + w(r), \min_{v \in N(r)} \left( \text{OPT}_1(v) + \sum_{u \in N(r) \setminus \{v\}} \text{OPT}_0(u) \right) \right\}$
- $\text{OPT}_1(r) = \left( \sum_{v \in N(r)} \text{OPT}_2(v) \right) + w(r)$
- $\text{OPT}_2(r) = \min \left\{ \left( \sum_{v \in N(r)} \text{OPT}_2(v) \right) + w(r), \sum_{v \in N(r)} \text{OPT}_0(v) \right\}$

$\text{OPT}_0(r)$ is desired answer
Minimum Dominating Set, in Trees (VI)

Let $T$ be a rooted tree with root $r$.  

**Subproblems:**

- **type-0**: regular dominating set
- **type-1**: dominating set which includes root $r$
- **type-2**: dominating set which is relaxed at root $r$

**Recursion:**

- \[ \text{OPT}_0(r) = \min \left\{ \left( \sum_{v \in N(r)} \text{OPT}_2(v) \right) + w(r) \right\} \]
- \[ \text{OPT}_1(r) = \left( \sum_{v \in N(r)} \text{OPT}_2(v) \right) + w(r) \]
- \[ \text{OPT}_2(r) = \min \left\{ \left( \sum_{v \in N(r)} \text{OPT}_2(v) \right) + w(r) \right\} \]

$\text{OPT}_0(r)$ is desired answer

**Recursive algorithm:**

- 3 \cdot n subproblems can implicitly memoize naively $O(n)$ work per node, can optimize to $O(n)$ total work as with MIS on trees

**Iterative algorithm:**

- Follow post-order traversal of rooted tree to satisfy dependencies
- Optimize analysis to obtain $O(n)$ total work

Details are an exercise
Minimum Dominating Set, in Trees (VI)

$T$ rooted tree with root $r$.

**subproblems:**
- **type-0**: regular dominating set
- **type-1**: dominating set which includes root $r$
- **type-2**: dominating set which is relaxed at root $r$

**recursion:**

- $\text{OPT}_0(r) = \min \left\{ \left( \sum_{v \in N(r)} \text{OPT}_2(v) \right) + w(r) \right\}$
- $\min_{v \in N(r)} \left( \text{OPT}_1(v) + \sum_{u \in N(r) \setminus \{v\}} \text{OPT}_0(u) \right)$
- $\text{OPT}_1(r) = \left( \sum_{v \in N(r)} \text{OPT}_2(v) \right) + w(r)$
- $\text{OPT}_2(r) = \min \left\{ \left( \sum_{v \in N(r)} \text{OPT}_2(v) \right) + w(r) \right\}$
- $\sum_{v \in N(r)} \text{OPT}_0(v)$

$\text{OPT}_0(r)$ is desired answer

**recursive algorithm:**
- $3 \cdot n$ subproblems
Minimum Dominating Set, in Trees (VI)

$T$ rooted tree with root $r$.

**subproblems:**
- **type-0**: regular dominating set
- **type-1**: dominating set which includes root $r$
- **type-2**: dominating set which is relaxed at root $r$

**recursion:**
- \[ \text{OPT}_0(r) = \min \left\{ \left( \sum_{v \in N(r)} \text{OPT}_2(v) \right) + w(r) \right\} \]
- \[ \text{OPT}_1(r) = \left( \sum_{v \in N(r)} \text{OPT}_2(v) \right) + w(r) \]
- \[ \text{OPT}_2(r) = \min \left\{ \left( \sum_{v \in N(r)} \text{OPT}_2(v) \right) + w(r) \right\} \]

\[ \sum_{v \in N(r)} \text{OPT}_0(v) \]

\text{OPT}_0(r) is desired answer

**recursive algorithm:**
- 3 $\cdot$ $n$ subproblems
- can implicitly memoize

**iterative algorithm:**
- follow post-order traversal of rooted tree to satisfy dependencies
- optimize analysis to obtain $O(n)$ total work
- details are an exercise
Minimum Dominating Set, in Trees (VI)

$T$ rooted tree with root $r$.

**subproblems:**
- **type-0:** regular dominating set
- **type-1:** dominating set which includes root $r$
- **type-2:** dominating set which is relaxed at root $r$

**recursion:**

- $\text{OPT}_0(r) = \min \left\{ \sum_{v \in N(r)} \text{OPT}_2(v) + w(r), \min_{v \in N(r)} \left( \text{OPT}_1(v) + \sum_{u \in N(r) \setminus \{v\}} \text{OPT}_0(u) \right) \right\}$
- $\text{OPT}_1(r) = \sum_{v \in N(r)} \text{OPT}_2(v) + w(r)$
- $\text{OPT}_2(r) = \min \left\{ \sum_{v \in N(r)} \text{OPT}_2(v) + w(r), \sum_{v \in N(r)} \text{OPT}_0(v) \right\}$

$\text{OPT}_0(r)$ is desired answer

**recursive algorithm:**
- 3 · $n$ subproblems
- can implicitly memoize
- naively $O(n)$ work per node,

**iterative algorithm:**
- follow post-order traversal of rooted tree to satisfy dependencies
- optimize analysis to obtain $O(n)$ total work

details are an exercise
Minimum Dominating Set, in Trees (VI)

A rooted tree with root $r$.

**Subproblems:**
- **type-0:** regular dominating set
- **type-1:** dominating set which includes root $r$
- **type-2:** dominating set which is relaxed at root $r$

**Recursion:**

\[
\begin{align*}
\text{OPT}_0(r) &= \min \left\{ \left( \sum_{v \in N(r)} \text{OPT}_2(v) \right) + w(r) \right. \\
& \quad \quad \left. \min_{v \in N(r)} \left( \text{OPT}_1(v) + \sum_{u \in N(r) \setminus \{v\}} \text{OPT}_0(u) \right) \right\} \\
\text{OPT}_1(r) &= \left( \sum_{v \in N(r)} \text{OPT}_2(v) \right) + w(r) \\
\text{OPT}_2(r) &= \min \left\{ \left( \sum_{v \in N(r)} \text{OPT}_2(v) \right) + w(r) \right. \\
& \quad \quad \left. \sum_{v \in N(r)} \text{OPT}_0(v) \right\}
\end{align*}
\]

$\text{OPT}_0(r)$ is desired answer

**Recursive algorithm:**
- $3 \cdot n$ subproblems
- can implicitly memoize
- naively $O(n)$ work per node, can optimize to $O(n)$ total work
Minimum Dominating Set, in Trees (VI)

Let $T$ be a rooted tree with root $r$.

**Subproblems:**
- **type-0**: regular dominating set
- **type-1**: dominating set which includes root $r$
- **type-2**: dominating set which is relaxed at root $r$

**Recursion:**

\[
\begin{align*}
\text{OPT}_0(r) &= \min \left\{ \left( \sum_{v \in N(r)} \text{OPT}_2(v) \right) + w(r), \quad \min_{v \in N(r)} \left( \text{OPT}_1(v) + \sum_{u \in N(r) \setminus \{v\}} \text{OPT}_0(u) \right) \right\} \\
\text{OPT}_1(r) &= \left( \sum_{v \in N(r)} \text{OPT}_2(v) \right) + w(r) \\
\text{OPT}_2(r) &= \min \left\{ \left( \sum_{v \in N(r)} \text{OPT}_2(v) \right) + w(r), \quad \sum_{v \in N(r)} \text{OPT}_0(v) \right\}
\end{align*}
\]

$\text{OPT}_0(r)$ is the desired answer.

**Recursive algorithm:**
- $3 \cdot n$ subproblems
- Can implicitly memoize
- Naively $O(n)$ work per node, can optimize to $O(n)$ total work as with MIS on trees

**Iterative algorithm:**
- Follow post-order traversal of rooted tree to satisfy dependencies
- Optimize analysis to obtain $O(n)$ total work
- Details are an exercise
Minimum Dominating Set, in Trees (VI)

A rooted tree with root $r$.

Subproblems:
- **type-0**: regular dominating set
- **type-1**: dominating set which includes root $r$
- **type-2**: dominating set which is relaxed at root $r$

Recursion:

- \[
  \text{OPT}_0(r) = \min \left\{ \left( \sum_{v \in N(r)} \text{OPT}_2(v) \right) + w(r), \min_{v \in N(r)} \left( \text{OPT}_1(v) + \sum_{u \in N(r) \setminus \{v\}} \text{OPT}_0(u) \right) \right\}
  \]

- \[
  \text{OPT}_1(r) = \left( \sum_{v \in N(r)} \text{OPT}_2(v) \right) + w(r)
  \]

- \[
  \text{OPT}_2(r) = \min \left\{ \left( \sum_{v \in N(r)} \text{OPT}_2(v) \right) + w(r), \sum_{v \in N(r)} \text{OPT}_0(v) \right\}
  \]

$\text{OPT}_0(r)$ is desired answer

**Recursive algorithm:**
- 3 \cdot n subproblems
- Can implicitly memoize
- Naively $O(n)$ work per node, can optimize to $O(n)$ total work as with MIS on trees

**Iterative algorithm:**
- Follow post-order traversal of rooted tree to satisfy dependencies
- Optimize analysis to obtain $O(n)$ total work
- Details are an exercise
Minimum Dominating Set, in Trees (VI)

$T$ rooted tree with root $r$.

**subproblems:**
- **type-0**: regular dominating set
- **type-1**: dominating set which includes root $r$
- **type-2**: dominating set which is relaxed at root $r$

**recursion:**

\[
\begin{align*}
\text{OPT}_0(r) &= \min \left\{ \left( \sum_{v \in N(r)} \text{OPT}_2(v) \right) + w(r) \right\} \\
&= \min_{v \in N(r)} \left( \text{OPT}_1(v) + \sum_{u \in N(r) \setminus \{v\}} \text{OPT}_0(u) \right) \\
\text{OPT}_1(r) &= \left( \sum_{v \in N(r)} \text{OPT}_2(v) \right) + w(r) \\
\text{OPT}_2(r) &= \min \left\{ \left( \sum_{v \in N(r)} \text{OPT}_2(v) \right) + w(r) \right\} \\
&= \sum_{v \in N(r)} \text{OPT}_0(v)
\end{align*}
\]

**recursive algorithm:**
- 3 · $n$ subproblems
- can implicitly memoize
- naively $O(n)$ work per node, can optimize to $O(n)$ total work as with MIS on trees

**iterative algorithm:**
- follow post-order traversal of rooted tree to satisfy dependencies

$\text{OPT}_0(r)$ is desired answer
**Minimum Dominating Set, in Trees (VI)**

A rooted tree $T$ with root $r$.

**Subproblems:**
- **type-0**: regular dominating set
- **type-1**: dominating set which includes root $r$
- **type-2**: dominating set which is relaxed at root $r$

**Recursion:**

- $\text{OPT}_0(r) = \min \left\{ \left( \sum_{v \in N(r)} \text{OPT}_2(v) \right) + w(r) \right\}$
- $\text{OPT}_1(r) = \left( \sum_{v \in N(r)} \text{OPT}_2(v) \right) + w(r)$
- $\text{OPT}_2(r) = \min \left\{ \left( \sum_{v \in N(r)} \text{OPT}_2(v) \right) + w(r) \right\}$

$\text{OPT}_0(r)$ is the desired answer.

**Recursive algorithm:**
- $3 \cdot n$ subproblems
- Can implicitly memoize
- Naively $O(n)$ work per node, can optimize to $O(n)$ total work as with MIS on trees

**Iterative algorithm:**
- Follow post-order traversal of rooted tree to satisfy dependencies
- Optimize analysis to obtain $O(n)$ total work
T rooted tree with root r.

**subproblems:**
- **type-0**: regular dominating set
- **type-1**: dominating set which includes root r
- **type-2**: dominating set which is relaxed at root r

**recursion:**
- \( \text{OPT}_0(r) = \min \left\{ \left( \sum_{v \in N(r)} \text{OPT}_2(v) \right) + w(r) \right\} \)
- \( \min_{v \in N(r)} \left( \text{OPT}_1(v) + \sum_{u \in N(r) \setminus \{v\}} \text{OPT}_0(u) \right) \)
- \( \text{OPT}_1(r) = \left( \sum_{v \in N(r)} \text{OPT}_2(v) \right) + w(r) \)
- \( \text{OPT}_2(r) = \min \left\{ \left( \sum_{v \in N(r)} \text{OPT}_2(v) \right) + w(r) \right\} \)
- \( \sum_{v \in N(r)} \text{OPT}_0(v) \)

OPT\(_0(r)\) is desired answer

**recursive algorithm:**
- 3 \( \cdot \) n subproblems
- can implicitly memoize
- naively \( O(n) \) work per node, can optimize to \( O(n) \) total work as with MIS on trees

**iterative algorithm:**
- follow post-order traversal of rooted tree to satisfy dependencies
- optimize analysis to obtain \( O(n) \) total work
details are an exercise
Dynamic Programming, in Trees (II)

Remarks:

Dynamic programming is about finding the correct recursion, and the correct recursion is intimately tied to understanding the structure and number of subproblems. Trees can be easily decomposed into a small number of subtrees, this allows a small number of resulting subproblems. Dynamic programming on trees can often be generalized to graphs of small treewidth.
Dynamic Programming, in Trees (II)

remarks:
Dynamic Programming, in Trees (II)

Remarks:
- Dynamic program is about finding the *correct* recursion,
Dynamic Programming, in Trees (II)

**remarks:**

- dynamic program is about finding the *correct* recursion, and the correct recursion is intimately tied to understand the *structure* and *number* of subproblems
Dynamic Programming, in Trees (II)

**remarks:**

- dynamic program is about finding the *correct* recursion, and the correct recursion is intimately tied to understand the *structure* and *number* of subproblems
- trees can be easily decomposed into a (small) number of subtrees,
Dynamic Programming, in Trees (II)

**Remarks:**

- Dynamic program is about finding the *correct* recursion, and the correct recursion is intimately tied to understand the *structure* and *number* of subproblems.

- Trees can be easily decomposed into a (small) number of subtrees, this allows a small number of resulting subproblems.
Dynamic Programming, in Trees (II)

**remarks:**

- dynamic program is about finding the *correct* recursion, and the correct recursion is intimately tied to understand the *structure* and *number* of subproblems
- trees can be easily decomposed into a (small) number of subtrees, this allows a small number of resulting subproblems
- dynamic programming on trees can often be generalized to graphs of small *treewidth*
today:
- dynamic programming on trees
- maximum independent set
- dominating set

next lecture:
- more dynamic programming

logistics:
- pset1 out, due R5 — can submit in groups of \( \leq 3 \)
<table>
<thead>
<tr>
<th></th>
<th>Title</th>
<th></th>
<th>Maximum Independent Set, in Trees (III)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Overview</td>
<td></td>
<td>Maximum Independent Set, in Trees (IV)</td>
</tr>
<tr>
<td></td>
<td>Dynamic Programming</td>
<td></td>
<td>Maximum Independent Set, in Trees (V)</td>
</tr>
<tr>
<td>4</td>
<td>Trees</td>
<td></td>
<td>Dynamic Programming, in Trees</td>
</tr>
<tr>
<td>5</td>
<td>Maximum Independent Set</td>
<td></td>
<td>Minimum Dominating Set</td>
</tr>
<tr>
<td>6</td>
<td>Maximum Independent Set (II)</td>
<td></td>
<td>Minimum Dominating Set (II)</td>
</tr>
<tr>
<td>7</td>
<td>Maximum Independent Set (III)</td>
<td></td>
<td>Minimum Dominating Set (III)</td>
</tr>
<tr>
<td>8</td>
<td>Maximum Independent Set (IV)</td>
<td></td>
<td>Minimum Dominating Set, in Trees</td>
</tr>
<tr>
<td>9</td>
<td>Maximum Independent Set (V)</td>
<td></td>
<td>Minimum Dominating Set, in Trees (II)</td>
</tr>
<tr>
<td>10</td>
<td>Maximum Independent Set (VI)</td>
<td></td>
<td>Minimum Dominating Set, in Trees (III)</td>
</tr>
<tr>
<td>11</td>
<td>Maximum Independent Set (VII)</td>
<td></td>
<td>Minimum Dominating Set, in Trees (IV)</td>
</tr>
<tr>
<td>12</td>
<td>Maximum Independent Set, in Trees</td>
<td></td>
<td>Minimum Dominating Set, in Trees (V)</td>
</tr>
<tr>
<td>13</td>
<td>Maximum Independent Set, in Trees (II)</td>
<td></td>
<td>Minimum Dominating Set, in Trees (VI)</td>
</tr>
<tr>
<td>15</td>
<td></td>
<td></td>
<td>Overview (II)</td>
</tr>
</tbody>
</table>

29 / 29