Overview

**logistics:**
- pset1 out, due W10 (next week) — can submit in *groups* of $\leq 3$
- if you are waiting to enroll: post private note in piazza with name, netid, major by *today* — we have a limited number of additional spots in the online section and will prioritize enrollment

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

**today:**
- dynamic programming *on trees*
- maximum independent set
- dominating set
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
- recognizing that dynamic programming applies to a problem can be non-obvious
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 $\Rightarrow$ 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 $(u, v) \notin E$.

ex:

![Graph](image)

Independent sets include $\emptyset$, {$A, C$}, and {$B, E, F$}. 
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

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

**Definition**

Maximum Independent Set (III)
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
- 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**

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

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

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

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

**correctness:** clear

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

- \( T(0), T(1) \geq \Omega(1). \) \( T(n) \geq T(n - 1) + T(n - 1 - \deg(v)) \)
- 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
- memoization does not obviously help — subproblems correspond to subgraphs, of which there are possibly exponentially many
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?
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 $v$.

Corollary

Let $T = (V, E)$ be a tree. Pick a root $r \in V$ for $T$ to create the rooted tree $(T, r)$. If you run \texttt{recursive-MIS} on $T$ and always eliminate the nodes who were closest to $r$ in $T$, then the result subproblems exactly correspond to rooted subtrees of $(T, r)$

$\implies \quad \leq |V|$ subproblems

$\implies \quad$ memoized recursive algorithm is efficient
Maximum Independent Set, in Trees (IV)

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 $v$

$\implies$ iterating over $V$ in post-order traversal of $T$ will satisfy the dependency graph.
iterative algorithm:

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

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

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 \(\implies O(n)\) time
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
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 $\{A, B, C, D, E, F\}$, $\{E, C, F\}$, and $\{A, B, F\}$. 
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

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

S dominating set of G $v \in S$
Minimum Dominating Set (III)

remarks:

- minimum (weight) dominating set is solvable via brute force: try all possible subsets $\implies$ solvable in time $O(n^{O(1)}2^n)$
- no efficient algorithm currently known
- minimum weight dominating set is NP-hard $\implies$ an efficient algorithm not expected to exist
- minimum weight dominating set is efficiently solvable if the underlying graph is a tree
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 $S(a) \cup S(b) \cup \{r\}$
  - *but can 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?
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)

$T$ 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**

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

**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 (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**

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

**Proof.**

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

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

**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)

$T$ 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:**

- $OPT_0(r) = \min \left\{ \left( \sum_{v \in N(r)} OPT_2(v) \right) + w(r) \right\}$
- $OPT_1(r) = \left( \sum_{v \in N(r)} OPT_2(v) \right) + w(r)$
- $OPT_2(r) = \min \left\{ \left( \sum_{v \in N(r)} OPT_2(v) \right) + w(r), \sum_{v \in N(r)} OPT_0(v) \right\}$

$OPT_0(r)$ is 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
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*
Overview (II)

**logistics:**
- pset1 out, due W10 (next week) — can submit in *groups* of $\leq 3$
- if you are waiting to enroll: post private note in piazza with name, netid, major by *today* — we have a limited number of additional spots in the online section and will prioritize enrollment

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

**next time:**
- *more* dynamic programming