Dynamic Programming on Trees

Lecture 4
September 2, 2016
What is Dynamic Programming?

Every recursion can be memoized. Automatic memoization does not help us understand whether the resulting algorithm is efficient or not.

**Dynamic Programming:**
A recursion that when memoized leads to an *efficient* algorithm.

**Key Questions:**
- Given a recursive algorithm, how do we analyze the complexity when it is memoized?
- How do we recognize whether a problem admits a dynamic programming based efficient algorithm?
- How do we further optimize time and space of a dynamic programming based algorithm?
Dynamic Programming Template

1. Come up with a recursive algorithm to solve problem
2. Understand the structure/number of the subproblems generated by recursion
3. Memoize the recursion
   - set up compact notation for subproblems
   - set up a data structure for storing subproblems
4. Iterative algorithm
   - Understand dependency graph on subproblems
   - Pick an evaluation order (any topological sort of the dependency dag)
5. Analyze time and space
6. Optimize
Dynamic Programming on Trees

**Fact:** Many graph optimization problems are **NP-Hard**

**Fact:** The same graph optimization problems are in \( P \) on trees.

Why?
Dynamic Programming on Trees

**Fact:** Many graph optimization problems are **NP-Hard**

**Fact:** The same graph optimization problems are in **P** on trees.

Why?

**A significant reason:** DP algorithm based on *decomposability*

Powerful methodology for graph algorithms via a formal notion of decomposability called **treewidth** (beyond the scope of this class)
Maximum Independent Set in a Graph

**Definition**

Given undirected graph $G = (V, E)$ a subset of nodes $S \subseteq V$ is an independent set (also called a stable set) if for there are no edges between nodes in $S$. That is, if $u, v \in S$ then $(u, v) \notin E$.

Some independent sets in graph above: $\{D\}, \{A, C\}, \{B, E, F\}$
Maximum Independent Set Problem

**Input** Graph $G = (V, E)$

**Goal** Find maximum sized independent set in $G$
Maximum Weight Independent Set Problem

Input  Graph $G = (V, E)$, weights $w(v) \geq 0$ for $v \in V$

Goal  Find maximum weight independent set in $G$
No one knows an efficient (polynomial time) algorithm for this problem.

Problem is NP-Hard and it is believed that there is no polynomial time algorithm.

**Brute-force algorithm:**

Try all subsets of vertices.
A Recursive Algorithm

Let $V = \{v_1, v_2, \ldots, v_n\}$.
For a vertex $u$ let $N(u)$ be its neighbors.
A Recursive Algorithm

Let \( V = \{v_1, v_2, \ldots, v_n\} \).

For a vertex \( u \) let \( N(u) \) be its neighbors.

Observation

\( v_1 \): vertex in the graph.

One of the following two cases is true

Case 1 \( v_1 \) is in some maximum independent set.

Case 2 \( v_1 \) is in no maximum independent set.

We can try both cases to “reduce” the size of the problem
A Recursive Algorithm

Let $V = \{v_1, v_2, \ldots, v_n\}$.
For a vertex $u$ let $N(u)$ be its neighbors.

Observation
$v_1$: vertex in the graph.
One of the following two cases is true

Case 1 $v_1$ is in some maximum independent set.
Case 2 $v_1$ is in no maximum independent set.

We can try both cases to “reduce” the size of the problem

$G_1 = G - v_1$ obtained by removing $v_1$ and incident edges from $G$
$G_2 = G - v_1 - N(v_1)$ obtained by removing $N(v_1) \cup v_1$ from $G$

$\text{MIS}(G) = \max\{\text{MIS}(G_1), \text{MIS}(G_2) + w(v_1)\}$
A Recursive Algorithm

\begin{center}
\begin{tabular}{|l|}
\hline
\textbf{RecursiveMIS}(G):
\textbf{if} $G$ is empty \textbf{then} Output 0
\textit{a} = RecursiveMIS($G - v_1$)
\textit{b} = $w(v_1)$ + RecursiveMIS($G - v_1 - N(v_1)$)
Output $\text{max}(a, b)$
\hline
\end{tabular}
\end{center}
Recursive Algorithms
..for Maximum Independent Set

Running time:

\[ T(n) = T(n - 1) + T\left(n - 1 - \deg(v_1)\right) + O(1 + \deg(v_1)) \]

where \( \deg(v_1) \) is the degree of \( v_1 \). \( T(0) = T(1) = 1 \) is base case.

Worst case is when \( \deg(v_1) = 0 \) when the recurrence becomes

\[ T(n) = 2T(n - 1) + O(1) \]

Solution to this is \( T(n) = O(2^n) \).
Memoization

We can memoize the recursive algorithm.

**Question:** Does it lead to an efficient algorithm?
Memoization

We can memoize the recursive algorithm.

**Question:** Does it lead to an efficient algorithm?

What is number of subproblems if started on graph with $n$ nodes?

**Exercise:** Show that even when $G$ is a cycle the number of subproblems is exponential in $n$. 
Part I

Maximum Weighted Independent Set in Trees
Maximum Weight Independent Set in a Tree

Input: Tree $T = (V, E)$ and weights $w(v) \geq 0$ for each $v \in V$

Goal: Find maximum weight independent set in $T$

Maximum weight independent set in above tree: ??
A Recursive Algorithm

For an arbitrary graph $G$:
1. Number vertices as $v_1, v_2, \ldots, v_n$
2. Find recursively optimum solutions without $v_n$ (recurse on $G - v_n$) and with $v_n$ (recurse on $G - v_n - N(v_n)$ & include $v_n$).
3. Saw that if graph $G$ is arbitrary there was no good ordering that resulted in a small number of subproblems.

What about a tree?
A Recursive Algorithm

For an arbitrary graph $G$:

1. Number vertices as $v_1, v_2, \ldots, v_n$

2. Find recursively optimum solutions without $v_n$ (recurse on $G - v_n$) and with $v_n$ (recurse on $G - v_n - N(v_n)$ & include $v_n$).

3. Saw that if graph $G$ is arbitrary there was no good ordering that resulted in a small number of subproblems.

What about a tree? Natural candidate for $v_n$ is root $r$ of $T$?
Towards a Recursive Solution

Natural candidate for $v_n$ is root $r$ of $T$? Let $O$ be an optimum solution to the whole problem.

Case $r \notin O$: Then $O$ contains an optimum solution for each subtree of $T$ hanging at a child of $r$. 
Towards a Recursive Solution

Natural candidate for \( v_n \) is root \( r \) of \( T \)? Let \( O \) be an optimum solution to the whole problem.

Case \( r \notin O \) : Then \( O \) contains an optimum solution for each subtree of \( T \) hanging at a child of \( r \).

Case \( r \in O \) : None of the children of \( r \) can be in \( O \). \( O - \{r\} \) contains an optimum solution for each subtree of \( T \) hanging at a grandchild of \( r \).
Towards a Recursive Solution

Natural candidate for $v_n$ is root $r$ of $T$? Let $O$ be an optimum solution to the whole problem.

Case $r \notin O$ : Then $O$ contains an optimum solution for each subtree of $T$ hanging at a child of $r$.

Case $r \in O$ : None of the children of $r$ can be in $O$. $O - \{r\}$ contains an optimum solution for each subtree of $T$ hanging at a grandchild of $r$.

Subproblems? Subtrees of $T$ rooted at nodes in $T$. 
Towards a Recursive Solution

Natural candidate for $v_n$ is root $r$ of $T$? Let $O$ be an optimum solution to the whole problem.

Case $r \not\in O$: Then $O$ contains an optimum solution for each subtree of $T$ hanging at a child of $r$.

Case $r \in O$: None of the children of $r$ can be in $O$. $O - \{ r \}$ contains an optimum solution for each subtree of $T$ hanging at a grandchild of $r$.

Subproblems? Subtrees of $T$ rooted at nodes in $T$.

How many of them?
Towards a Recursive Solution

Natural candidate for $v_n$ is root $r$ of $T$? Let $O$ be an optimum solution to the whole problem.

Case $r \notin O$: Then $O$ contains an optimum solution for each subtree of $T$ hanging at a child of $r$.

Case $r \in O$: None of the children of $r$ can be in $O$. $O - \{r\}$ contains an optimum solution for each subtree of $T$ hanging at a grandchild of $r$.

Subproblems? Subtrees of $T$ rooted at nodes in $T$.

How many of them? $O(n)$
Example

```
    r
   /|
  a  b
 /|
c d e
 /|
h i
 /|
   j
```

Nodes: a, b, c, d, e, f, g, h, i, j

Weights: 4, 5, 6, 7, 8, 9, 10, 11
A Recursive Solution

\( T(u) \): subtree of \( T \) hanging at node \( u \)

\( \text{OPT}(u) \): max weighted independent set value in \( T(u) \)

\[
\text{OPT}(u) = \max (\text{OPT}(\text{child of } u), w(u) + \text{OPT}(\text{grandchild of } u))
\]
A Recursive Solution

\( T(u) \): subtree of \( T \) hanging at node \( u \)

\( OPT(u) \): max weighted independent set value in \( T(u) \)

\[
OPT(u) = \max \left\{ \sum_{v \text{ child of } u} OPT(v), \quad w(u) + \sum_{v \text{ grandchild of } u} OPT(v) \right\}
\]
Iterative Algorithm

1. Compute $OPT(u)$ bottom up. To evaluate $OPT(u)$ need to have computed values of all children and grandchildren of $u$.

2. What is an ordering of nodes of a tree $T$ to achieve above?
Iterative Algorithm

1. Compute $OPT(u)$ bottom up. To evaluate $OPT(u)$ need to have computed values of all children and grandchildren of $u$.

2. What is an ordering of nodes of a tree $T$ to achieve above? Post-order traversal of a tree.
Iterative Algorithm

\textbf{MIS-Tree} \((T)\):

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

\begin{verbatim}
for i = 1 to n do 
  \(M[v_i] = \max(\sum_{v_j \text{ child of } v_i} M[v_j], \ w(v_i) + \sum_{v_j \text{ grandchild of } v_i} M[v_j])\)
\end{verbatim}

return \(M[v_n]\) (* Note: \(v_n\) is the root of \(T\) *)

\textbf{Space:} \(O(n)\) to store the value at each node of \(T\)

\textbf{Running time:}

1. Naive bound: \(O(n^2)\) since each \(M[v_i]\) evaluation may take \(O(n)\) time and there are \(n\) evaluations.

2. Better bound: \(O(n)\). Value \(M[v_j]\) is accessed only by its parent and grand parent.
Iterative Algorithm

**MIS-Tree**($T$):

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

for $i = 1$ to $n$ do

$$M[v_i] = \max \left( \sum_{v_j \text{ child of } v_i} M[v_j], \quad w(v_i) + \sum_{v_j \text{ grandchild of } v_i} M[v_j] \right)$$

return $M[v_n]$ (* Note: $v_n$ is the root of $T$ *)

Space:
MIS-Tree\( (T) \):

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

\[
\text{for } i = 1 \text{ to } n \text{ do}
\]

\[
M[v_i] = \max \left( \sum_{v_j \text{ child of } v_i} M[v_j], w(v_i) + \sum_{v_j \text{ grandchild of } v_i} M[v_j] \right)
\]

\text{return } M[v_n] \quad \text{(* Note: } v_n \text{ is the root of } T \text{ *)}

Space: \( O(n) \) to store the value at each node of \( T \)

Running time:
Iterative Algorithm

**MIS-Tree** $(T)$:

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

for $i = 1$ to $n$ do

\[
M[v_i] = \max \left( \sum_{v_j \text{ child of } v_i} M[v_j], \quad w(v_i) + \sum_{v_j \text{ grandchild of } v_i} M[v_j] \right)
\]

return $M[v_n]$ (* Note: $v_n$ is the root of $T$ *)

**Space:** $O(n)$ to store the value at each node of $T$

**Running time:**

- Naive bound: $O(n^2)$ since each $M[v_i]$ evaluation may take $O(n)$ time and there are $n$ evaluations.
**MIS-Tree**($T$):

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

for $i = 1$ to $n$ do

\[
M[v_i] = \max \left( \sum_{v_j \text{ child of } v_i} M[v_j], \ w(v_i) + \sum_{v_j \text{ grandchild of } v_i} M[v_j] \right)
\]

return $M[v_n]$ (* Note: $v_n$ is the root of $T$ *)

---

**Space:** $O(n)$ to store the value at each node of $T$

**Running time:**

1. Naive bound: $O(n^2)$ since each $M[v_i]$ evaluation may take $O(n)$ time and there are $n$ evaluations.

2. Better bound: $O(n)$. A value $M[v_j]$ is accessed only by its parent and grand parent.
Why did DP work on trees?

Each node (including the root) is a separator!

Definition

Given a graph $G = (V, E)$ a set of nodes $S \subset V$ is a separator for $G$ if $G - S$ has at least two connected components.
Why did DP work on trees?

Each node (including the root) is a *separator*!

**Definition**

Given a graph $G = (V, E)$ a set of nodes $S \subseteq V$ is a *separator* for $G$ if $G - S$ has at least two connected components.

**Definition**

$S$ is a *balanced* separator if each connected component of $G - S$ has at most $2|V(G)|/3$ nodes.
Why did DP work on trees?

Each node (including the root) is a separator!

**Definition**

Given a graph $G = (V, E)$ a set of nodes $S \subseteq V$ is a separator for $G$ if $G - S$ has at least two connected components.

**Definition**

$S$ is a balanced separator if each connected component of $G - S$ has at most $2|V(G)|/3$ nodes.

**Exercise:** Prove that every tree $T$ has a balanced separator consisting of a single node.
Part II

Minimum Dominating Set in Trees
Definition

Given undirected graph $G = (V, E)$ a subset of nodes $S \subseteq V$ is a dominating set if for all $v \in V$, either $v \in S$ or a neighbor of $v$ is in $S$.

Some dominating sets in graph above: \{A, B, C, D, E, F\},
**Input**  
Graph $G = (V, E)$, weights $w(v) \geq 0$ for $v \in V$

**Goal**  
Find minimum weight dominating set in $G$
**Input**  Graph $G = (V, E)$, weights $w(v) \geq 0$ for $v \in V$

**Goal**  Find minimum weight dominating set in $G$

**NP-Hard** problem
Minimum Weight Dominating Set in a Tree

**Input**  
Tree $T = (V, E)$ and weights $w(v) \geq 0$ for each $v \in V$

**Goal**  
Find minimum weight dominating set in $T$

Minimum weight dominating set in above tree: ??
Recursive Algorithm

$r$ is root of $T$. Let $O$ be an optimum solution for $T$.

Case $r \not\in O$ : Then $O$ must contain some child of $r$. Which one?
Recursive Algorithm

\( r \) is root of \( T \). Let \( O \) be an optimum solution for \( T \).

Case \( r \notin O \): Then \( O \) must contain some child of \( r \). Which one?

Case \( r \in O \): None of the children of \( r \) *need* to be in \( O \) because \( r \) can dominate them. However, they may have to be.
Recursive Algorithm

\( r \) is root of \( T \). Let \( \mathcal{O} \) be an optimum solution for \( T \).

Case \( r \notin \mathcal{O} \): Then \( \mathcal{O} \) must contain some child of \( r \). Which one?

Case \( r \in \mathcal{O} \): None of the children of \( r \) need to be in \( \mathcal{O} \) because \( r \) can dominate them. However, they may have to be.

In both cases it is not feasible to express \( |\mathcal{O}| \) easily as optimum solution values of children or descendants of \( r \).

Removing \( r \) decomposes \( T \) into subtrees rooted at children of \( r \). However, not easy to decompose problem structure recursively. Problems at children of \( r \) are dependent. Need to introduce additional variable(s).
Let $u_1, u_2, \ldots, u_k$ be children of root $r$ of $T$

What “information” do $T_{u_1}, \ldots, T_{u_k}$ need to know about $r$'s status in an optimum solution in order to become “independent”
Let $u_1, u_2, \ldots, u_k$ be children of root $r$ of $T$

What “information” do $T_{u_1}, \ldots, T_{u_k}$ need to know about $r$'s status in an optimum solution in order to become “independent”

- Whether $r$ is included in the solution
- If $r$ is not included then which of the children is going to dominate it. Equivalently, $T_{u_i}$ needs to know whether it should cover $r$ or some other child will.
Recursive Algorithm: Introducing Variables

- **u**: node in tree
- **pi**: boolean variable to indicate whether parent is in solution. 
  - $pi = 0$ means parent is not included. $pi = 1$ means it is included.
- **cp**: boolean variable to indicate whether node needed to cover parent. 
  - $cp = 1$ means parent needs to be covered. $cp = 0$ means not needed.

$OPT(u, pi, cp)$: value of minimum dominating set in $T_u$ with booleans $pi$ and $cp$ with meaning above.

$OPT(r, 0, 0)$: value of minimum dominating set in $T$

$OPT(u, 1, 0)$: value of min dominating set in $T_u$ with parent of $u$ included no need to cover parent.
Recursive Solution

Can we express $OPT(u, pi, cp)$ recursively via children of $u$?

First consider $OPT(u, 0, 0)$ which is the value of a minimum dominating set in $T_u$ where we assume that $u$'s parent is not included and $u$ does not need to cover its parent. Let $C_u$ be children of $u$.

Case $u$ is included: Then $u$ does not need to covered by any child and we recurse on children with $u$ being included.

$OPT(u, 0, 0) = w(u) + \sum_{v \in C_u} OPT(v, 1, 0)$

Case $u$ is not included: Then $u$ needs to be covered by some child. We do a min over all children.

$OPT(u, 0, 0) = \min_{v \in C_u} (OPT(v, 0, 1) + \sum_{v_0 \in C_u} OPT(v_0, 0, 0))$

Since one of these cases has to be true, we take the min of the values in the above two cases to compute $OPT(u, 0, 0)$. 
Recursive Solution

Can we express $OPT(u, pi, cp)$ recursively via children of $u$?

First consider $OPT(u, 0, 0)$ which is the value of a minimum dominating set in $T_u$ where we assume that $u$’s parent is not included and $u$ does not need to cover its parent. Let $C_u$ be children of $u$.

Case $u$ is included: Then $u$ does not need to be covered by any child and we recurse on children with $u$ being included.

$$OPT(u, 0, 0) = w(u) + \sum_{v \in C_u} OPT(v, 1, 0)$$

Case $u$ is not included: Then $u$ needs to be covered by some child. We do a min over all children.

$$OPT(u, 0, 0) = \min_{v \in C_u} (OPT(v, 0, 1) + \sum_{v' \in C_u - v} OPT(v', 0, 0))$$

Since one of these cases has to be true, we take the min of the values in the above two cases to compute $OPT(u, 0, 0)$. 

Chandra & Ruta (UIUC)  
CS473  31  
Fall 2016  31 / 46
Recursive Solution

Consider $OPT(u, 1, 0)$ which is the value of a minimum dominating set in $T_u$ where we assume that $u$'s parent is included and $u$ does not need to cover its parent. Let $C_u$ be children of $u$.

Case $u$ is included: Then $u$ does not need to be covered by any child and we recurse on children with $u$ being included.

$$OPT(u, 1, 0) = w(u) + \sum_{v \in C_u} OPT(v, 1, 0)$$

Case $u$ is not included: Since $u$'s parent is included $u$ does not need to be covered by its children. Thus we have,

$$OPT(u, 1, 0) = \sum_{v \in C_u} OPT(v, 0, 0)$$

Since one of these cases has to be true, we take the min of the values in the above two cases to compute $OPT(u, 0, 0)$. 

Caution: Not including $u$ may appear to be always advantageous but it is not true.
Recursive Solution

Consider $OPT(u, 1, 0)$ which is the value of a minimum dominating set in $T_u$ where we assume that $u$'s parent is included and $u$ does not need to cover its parent. Let $C_u$ be children of $u$.

Case $u$ is included: Then $u$ does not need to be covered by any child and we recurse on children with $u$ being included.

$$OPT(u, 1, 0) = w(u) + \sum_{v \in C_u} OPT(v, 1, 0)$$

Case $u$ is not included: Since $u$'s parent is included $u$ does not need to be covered by its children. Thus we have,

$$OPT(u, 1, 0) = \sum_{v \in C_u} OPT(v, 0, 0)$$

Since one of these cases has to be true, we take the min of the values in the above two cases to compute $OPT(u, 0, 0)$.

Caution: Not including $u$ may appear to be always advantageous but it is not true.
Recursive Solution

Consider \( OPT(u, 0, 1) \) which is the value of a minimum dominating set in \( T_u \) where we assume that \( u \)’s parent is not included and \( u \) needs to cover its parent. Let \( C_u \) be children of \( u \).

**Case** \( u \) **is included**: Then \( u \) does not need to be covered by any child and we recurse on children with \( u \) being included.

\[
OPT(u, 0, 1) = w(u) + \sum_{v \in C_u} OPT(v, 1, 0)
\]

**Case** \( u \) **is not included**: This does not arise because \( u \) has to cover its parent.
Recursive Solution

Consider $OPT(u, 1, 1)$ which is the value of a minimum dominating set in $T_u$ where we assume that $u$’s parent is included and $u$ needs to cover its parent.

This subproblem does not make sense since if $u$’s parent is included then $u$ does not need to cover it. In other words it suffices to only consider the subproblems $OPT(u, 0, 0), OPT(u, 1, 0), OPT(u, 0, 1)$. 
Base Cases

Leaves are base cases. If $u$ is a leaf.

- $OPT(u, 0, 0) = w(u)$
- $OPT(u, 1, 0) = 0$
- $OPT(u, 0, 1) = w(u)$
Minimum weight dominating set value in $T$ is $OPT(r, 0, 0)$

To compute $OPT(r, 0, 0)$ we need to compute recursively $OPT(u, 0, 0)$, $OPT(u, 1, 0)$, $OPT(u, 0, 1)$ for all $u \in T$. Thus number of subproblems is $O(n)$.

Can do this bottom up from leaves to root.
DominatingSet-Tree($T$):

Let $v_1, v_2, \ldots, v_n$ be a post-order traversal of nodes of $T$.
Allocate array $M[1..n, 0..1, 0..1]$ to store $OPT(v_i, pi, cp)$ values.

for $i = 1$ to $n$ do

Compute $OPT(v_i, 0, 0)$, $OPT(v_i, 1, 0)$ and $OPT(v_i, 0, 1)$ using values of children of $v_i$ stored in $M$,
or via base cases if $v_i$ is leaf.

Store computed values in $M$ for use by parent of $v_i$.

return $OPT(v_n, 0, 0)$ (* Note: $v_n$ is the root of $T$ *)

Exercise: Work out details and prove an $O(n)$ time implementation.
Recap

- To obtain recursive solution we introduced additional variables based on “information” needed to decompose.
- Decomposition depends both on structure (trees decompose via separators) and objective function.
- Subproblems and recursion are almost defined hand in hand.
Part III

Maximum Independent Set in Planar Graphs
Planar Graphs

Definition

(Informal): A graph $G = (V, E)$ is planar if it can be drawn in the plane without edges crossing.

(More formal): $G$ is planar if there is a mapping of $\phi : V \rightarrow \mathbb{R}^2$ to distinct points in the 2-dimensional Euclidean plane and a mapping of each edge $uv \in E$ to a non-crossing curve $\psi(uv)$ that connects $\phi(u)$ to $\phi(v)$ such that the curves corresponding to different edges intersect only at $\phi(V)$.

Planar graphs are very important in both theory and practice.
Planar Graph Theorems

Many beautiful properties and theorems.

- Euler’s theorem. \(|E| \leq 3|V| - 6\) for every planar graph. Hence there is always a node of degree at most 5. Thus planar graphs are 5-degenerate.

- Kuratowski’s theorem: \(G\) is planar iff it “excludes” \(K_5\) or \(K_{3,3}\).

- 4-Color theorem. Every planar graph is 4-colorable.

- Planar Separator Theorem: Every planar graph \(G\) on \(n\) nodes has a balance separator of size \(O(\sqrt{n})\).

Properties are exploited to develop algorithms.
Maximum Independent Set in Planar Graphs

MIS in general graphs: best known algorithm runs in time $O(1.889^n)$.

Believed that MIS in general graphs requires $c^n$ time for some fixed $c > 1$.

However, can solve MIS in planar graphs in $2^{O(\sqrt{n})}$ time.
Recursive Decomposition via Planar Separator

Theorem

\text{RecursiveDecomp}(G):

If \(|V(G)| \leq n_0\)

Output tree \(T\) with single root node containing \(V(G)\)

Else

Compute balanced separator \(S\) of \(G\) of size \(O(\sqrt{|V(G)|})\)

For each component \(G_i\) of \(G - S\) do

\(T_i = \text{RecursiveDecomp}(G_i)\)

Create \(T\) with root \(r\) containing \(S\)

Make each \(T_i\) a child of \(r\)

Output \(T\)

Using linear-time algorithm to compute planar separator, algorithms run time is \(O(n)\).
\[ |S| \leq \sqrt{n} \]
Properties of Decomposition Tree $T$

Let $T$ be the decomposition tree for planar $G$ with $n$ nodes

- $T$ is a rooted tree where each tree-node $v \in V(T)$ there is a set $S_v \subseteq V(G)$ of $G$’s vertices associated with it.
  $|S_v| = O(\sqrt{n})$ for all $v \in V(T)$.
- $S_v \cap S_u = \emptyset$ for $u \neq v$ and $\bigcup_{v \in V(T)} S_v = V(G)$.
- For node $v$ let $S'_v = \bigcup_{v \in T_v} S_v$ be the nodes in sub-tree of $T$ rooted at $v$. Let $G_v = G[S'_v]$ be the sub-graph of $G$ induced by $S'_v$. Then $S_v$ is a balanced separator for $G_v$.
- Depth of $T$ is $O(\log n)$. $|V(T)| = O(n)$.