Computer Science and Information Engineering
National Chi Nan University

Chapter 6
Tree

§ 6.1 Max. Matching of Trees

6.1 Max. Matching of Trees

- **Def:**

① A connected graph $T = (V, E)$ is called a **Tree** if $|E| = |V| - 1$.

② A vertex x of a tree is called a **leaf** iff $\deg(x) = 1$.

- **Thm:**

① $T = (V, E)$ is a tree and $|E| \geq 1$ (or $|V| \geq 2$)
$\Rightarrow \exists$ at least one leaf x, $T - x$ is a tree.

6.1 Max. Matching of Trees

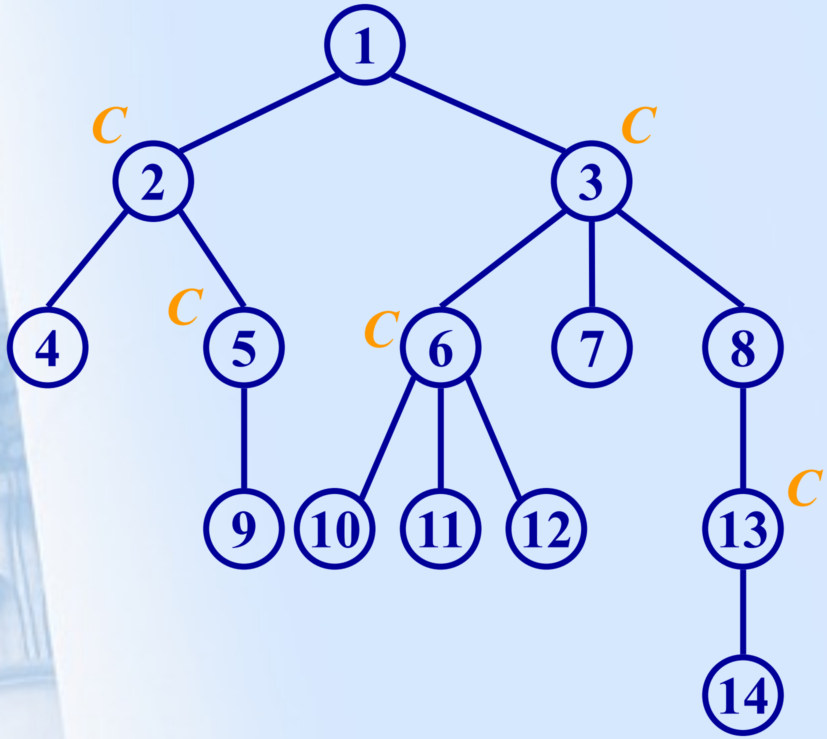
- **Algorithm**: **Max. Matching for Trees $T = (V, E)$**

$$
\begin{array}{l}
M \leftarrow \varnothing; \\
C \leftarrow \varnothing; \\
\underline{\textbf{while}}\ (|V| \geq 2) \\
\qquad \textbf{choose a leaf } x \textbf{ adjacent to } y \\
\qquad \underline{\textbf{if}}\ \{x, y\} \cap C = \varnothing\ \underline{\textbf{then}} \\
\qquad\quad \begin{cases} M \leftarrow M \cup \{xy\}; \\ C \leftarrow C \cup \{y\}; \end{cases} \\
\qquad V \leftarrow V - x; \\
\qquad E \leftarrow E - \{xy\}; \\
\underline{\textbf{end}}
\end{array}
$$

- **Time Complexity: $O(|V|)$**

6.1 Max. Matching of Trees

- **ex:**



6.1 Max. Matching of Trees

- **<justify Max. Matching Algorithm of Tree>**

  Assume M^* is the final output M, and C^* is the final output C.

  <u>Claim ①</u>: M^* is a matching.

  <u>Claim ②</u>: C^* is a vertex cover.

  <u>Claim ③</u>: $|M^*| = |C^*|$.

  Then $|M^*| \leq \max_M |M| \leq \min_C |C| \leq |C^*| = |M^*|$

  ∴ all "≤" are "=".

  ⇒ ①′ M^* is a max. matching.

  ②′ C^* is a min. vertex cover.

  ③′ $\max_M |M| = \min_C |C|$.

6.1 Max. Matching of Trees

- **Proof of Claim.**

  ① **In any iteration, M is <u>matching</u>**

  (o.w. $\exists \{x_1y_1, x_2y_2\} \subseteq M$ s.t. $\{x_1y_1\} \cap \{x_2y_2\} \neq \varnothing$)

  $\Rightarrow M^*$ is matching.

  ② **$\forall\, xy \in E$, either $\{x, y\} \cap C \neq \varnothing$ or $\{x, y\} \cap C = \varnothing$**

  $\Rightarrow C = C \cup \{y\}$.

  $\because$ <u>if</u>

  $\therefore \{x, y\} \cap C^* \neq \varnothing$.

  i.e. C^* is a vertex cover.

  ③ **In any iteration, $|M| = |C|$. ($\because$ <u>if then</u>)**

  $\therefore |M^*| = |C^*|$.

Computer Science and Information Engineering
National Chi Nan University

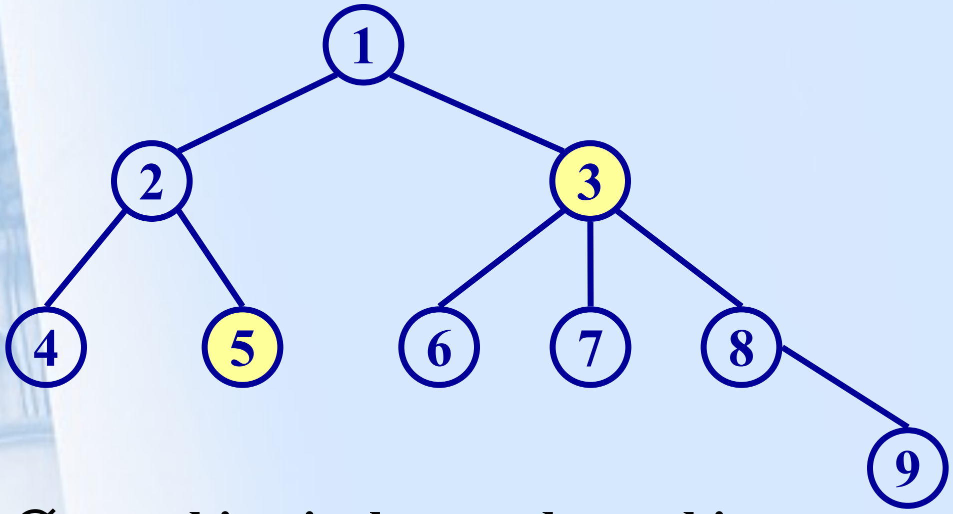
Chapter 6
Tree

§ 6.2 Dynamic Programming for Max. *C*-Matching on Trees

6.2 Dynamic Programming for Max. *C*-Matching on Trees

- **<u>Def</u>: $T = (V, E)$ is a tree, given $C \subseteq V$. A *C*-matching of T is a matching M of T s.t. not exist $xy \in M$ with $\{x, y\} \cap C \neq \varnothing$. (i.e. $\forall\ xy \in M$, $\{x, y\} \cap C = \varnothing$)**

- **<u>ex</u>:**



- **<u>Remark</u>: $\varnothing$-matching is the usual matching.**

- **<u>Def</u>: $\alpha'(T, C)$ = max$\{|M|$: M is a C-matching of $T\}$.**

6.2 Dynamic Programming for Max. *C*-Matching on Trees

- **Thm:** T is a tree and x is a leaf adjacent to y.

① $\{x, y\} \cap C \neq \varnothing \Rightarrow \alpha'(T, C) = \alpha'(T - x, C - \{x\})$.

② $\{x, y\} \cap C = \varnothing \Rightarrow \alpha'(T, C) = \alpha'(T - x, C \cup \{y\}) + 1$.

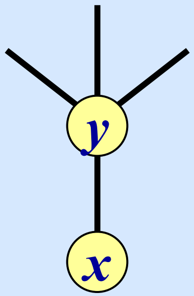
Proof. (1/3)

① Choose a max. C-matching M of T, i.e. $|M| = \alpha'(T, C)$.

$\because \{x, y\} \cap C \neq \varnothing$, $\{xy\} \notin M$

$\therefore$ M is a $(C - \{x\})$-matching in $T - x$.

$\Rightarrow \alpha'(T, C) = |M| \leq \alpha'(T - x, C - \{x\})$ …(1)



On the other hand,

suppose M^* is a max. $(C - \{x\})$-matching in $T - x$.

It is easy to see that M^* is a C-matching of T. ($\because \{xy\} \notin M^*$)

$\therefore \alpha'(T, C) \geq |M^*| = \alpha'(T - x, C - \{x\})$ …(2)

$\therefore$ By (1)(2), $\alpha'(T, C) = \alpha'(T - x, C - \{x\})$.

6.2 Dynamic Programming for Max. C-Matching on Trees

- **Thm**: T is a tree and x is a leaf adjacent to y.

① $\{x, y\} \cap C \neq \varnothing \Rightarrow \alpha'(T, C) = \alpha'(T - x, C - \{x\})$.

② $\{x, y\} \cap C = \varnothing \Rightarrow \alpha'(T, C) = \alpha'(T - x, C \cup \{y\}) + 1$.

Proof. (2/3)

② Choose a max. C-matching M of T, i.e. $|M| = \alpha'(T, C)$.

Let $M' = M - \{yz: yz \in M\}$.

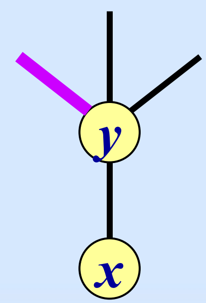
(such yz exists, otherwise $M \cup \{xy\}$ is a C-matching of size greater then $|M|$. $\rightarrow\leftarrow$ ($\because \{x, y\} \cap C = \varnothing$))

It is easy to see that M' is a $(C \cup \{y\})$-matching in $T - x$.

$\therefore |M'| \leq \alpha'(T - x, C \cup \{y\})$.

$\because |M'| = |M| - 1 = \alpha'(T, C) - 1$.

$\therefore \alpha'(T, C) \leq \alpha'(T - x, C \cup \{y\}) + 1$. …(1)



6.2 Dynamic Programming for Max. C-Matching on Trees

- **Thm:** T is a tree and x is a leaf adjacent to y.

① $\{x, y\} \cap C \neq \varnothing \Rightarrow \alpha'(T, C) = \alpha'(T - x, C - \{x\})$.

② $\{x, y\} \cap C = \varnothing \Rightarrow \alpha'(T, C) = \alpha'(T - x, C \cup \{y\}) + 1$.

Proof. (3/3)

② On the other hand,

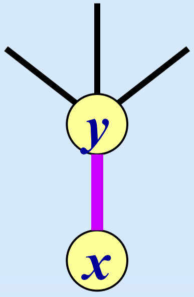
suppose M^* is a max. $(C \cup \{y\})$-matching in $T - x$.

Let $M^{**} = M^* \cup \{xy\}$.

It is easy to see that M^{**} is a C-matching of T. ($\because \{x, y\} \cap C = \varnothing$)

$\therefore \alpha'(T, C) \geq |M^{**}| = |M^*| + 1 = \alpha'(T - x, C \cup \{y\}) + 1$…(2)

$\therefore$ By (1)(2), $\alpha'(T, C) = \alpha'(T - x, C \cup \{y\}) + 1$.



6.2 Dynamic Programming for Max. C-Matching on Trees

- **Algorithm: Dynamic Programming for Max. C-Matching on Trees**

  $T = (V, E)$

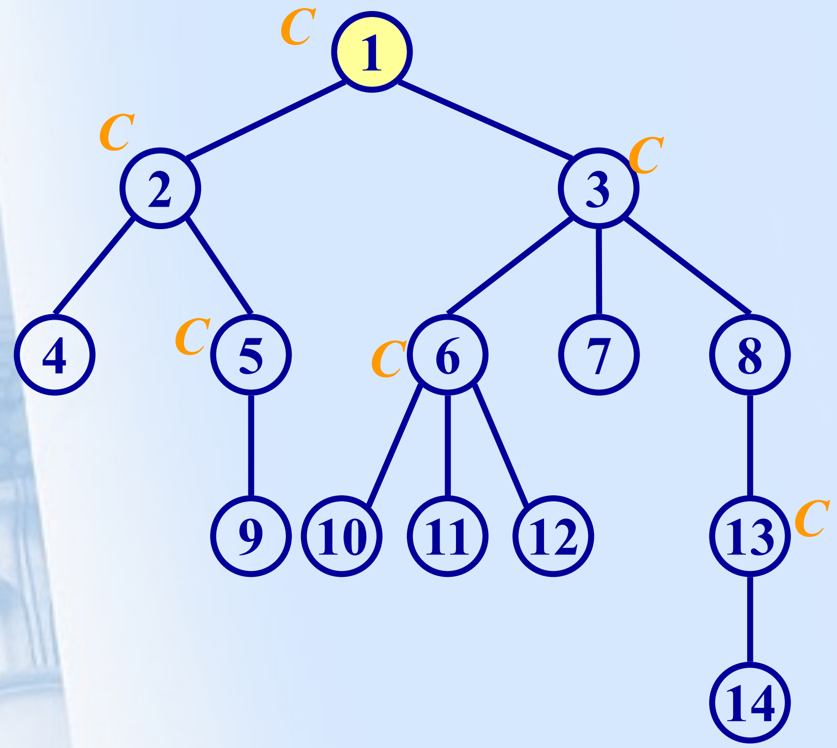
$M \leftarrow \varnothing$; $\alpha'(T, C)$;

Procedure $\alpha'(T, C)$
if ($|V| > 1$) **then**
    choose a leaf x adjacent to y
    if ($\{x, y\} \cap C \neq \varnothing$) **then**
        return $\alpha'(T - x, C - \{x\})$;
    else
        $M \leftarrow M \cup \{xy\}$;
        return $\alpha'(T - x, C \cup \{y\}) + 1$;
else
    return 0;

- **Time Complexity:** $O(|V|)$

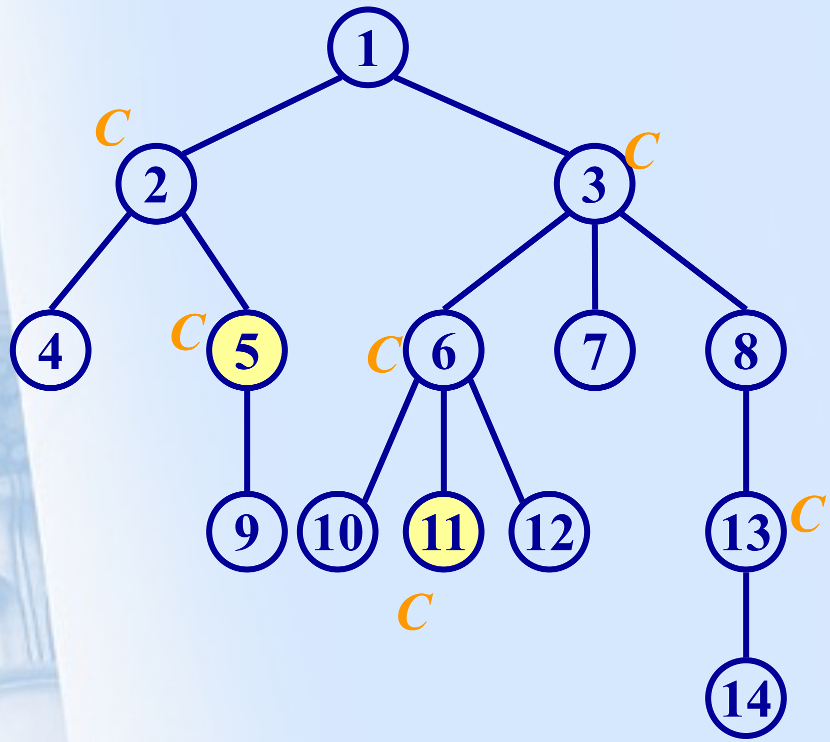
6.2 Dynamic Programming for Max. *C*-Matching on Trees

- ex:



6.2 Dynamic Programming for Max. *C*-Matching on Trees

- **ex:**



Computer Science and Information Engineering
National Chi Nan University

Chapter 6
Tree

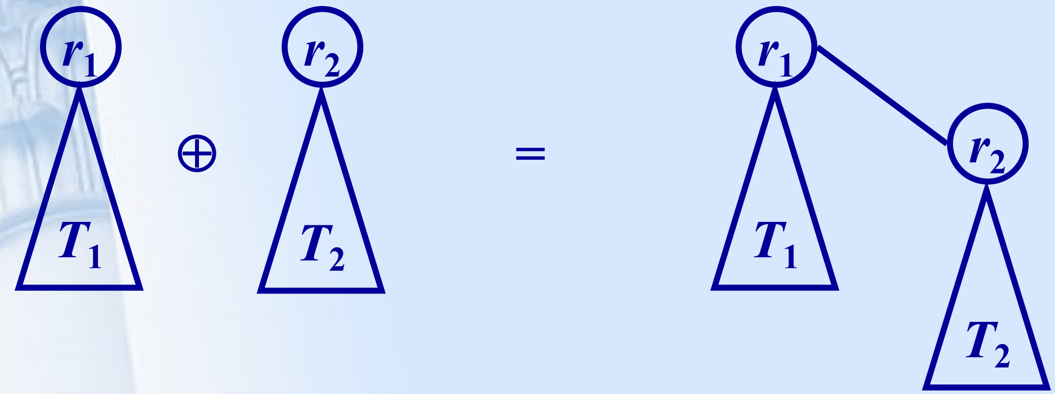
§ 6.3 Dynamic Programming for Matching on Trees

6.3 Dynamic Programming for Matching on Trees

- **Thm:**

② **A tree can be constructed by a sequence of "composition" operation from K_1, where "composition" operation denoted by $\oplus$, $T_1(r_1) \oplus T_2(r_2) = (V(T_1) \cup V(T_2), E(T_1) \cup E(T_2) \cup \{r_1r_2\})$ and root of $T_1 \oplus T_2$ is r_1, where r_i is the root of T_i, $i = 1, 2$.**

- **Ex:**



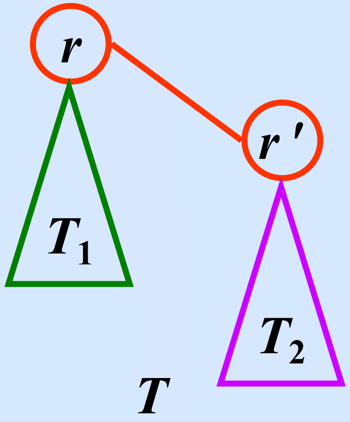
6.3 Dynamic Programming for Matching on Trees

- **Def**: T: rooted at r, $\alpha'(T, r) = \max\{|M|: M \text{ is a matching of } T\}$,
  $\alpha_0'(T, r) = \max\{|M|: M \text{ is a matching of } T - r\}$.

- **Thm**: $T(r) = T_1(r) \oplus T_2(r')$ then

① $\alpha'(T, r) = \max\{\alpha'(T_1, r) + \alpha'(T_2, r'), \alpha_0'(T_1, r) + \alpha_0'(T_2, r') + 1\}$,

② $\alpha_0'(T, r) = \alpha_0'(T_1, r) + \alpha'(T_2, r')$.

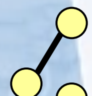
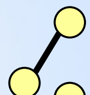


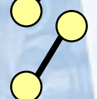
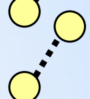
6.3 Dynamic Programming for Matching on Trees

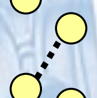
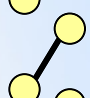
- **Thm:** $T = T_1(r) \oplus T_2(r')$

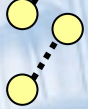
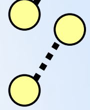
  $\Rightarrow \alpha'(T, r) = \alpha'(T_1, r) + \alpha'(T_2, r') + 1 - \lceil(\alpha'(T_1, r) - \alpha_0'(T_1, r) + \alpha'(T_2, r') - \alpha_0'(T_2, r')) / 2\rceil.$

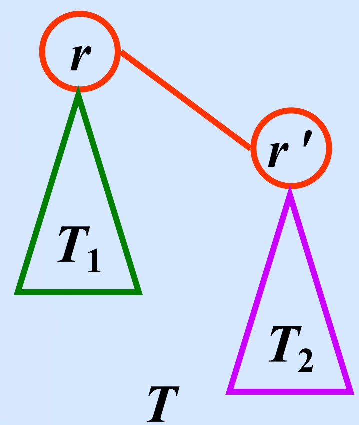
- **Ex:**

  If T_1:  , T_2:  $\Rightarrow 1 - \lceil(1 + 1) / 2\rceil = 0.$

  If T_1:  , T_2:  $\Rightarrow 1 - \lceil(1 + 0) / 2\rceil = 0.$

  If T_1:  , T_2:  $\Rightarrow 1 - \lceil(0 + 1) / 2\rceil = 0.$

  If T_1:  , T_2:  $\Rightarrow 1 - \lceil(0 + 0) / 2\rceil = 1.$



6.3 Dynamic Programming for Matching on Trees

- **Algorithm**: Dynamic Programming for Max. Matching on Trees $T = (V, E)$

$\alpha'(T, r)$;

Procedure $\alpha'(T, r)$
if $(|V| > 1)$ **then**
  find $T_1(r)$, $T_2(r')$ s.t.
    $T = T_1(r) \oplus T_2(r')$;
  $a = \alpha'(T_1, r)$;
  $b = \alpha'(T_2, r')$;
  $c = \alpha_0'(T_1, r)$;
  $d = \alpha_0'(T_2, r')$;
  return $a + b + 1 - \lceil (a - c + b - d) / 2 \rceil$;
else
  return 0;

Procedure $\alpha_0'(T, r)$
if $(|V| > 1)$ **then**
  find $T_1(r)$, $T_2(r')$ s.t.
    $T = T_1(r) \oplus T_2(r')$;
  return $\alpha_0'(T_1, r) + \alpha'(T_2, r')$;
else
  return 0;

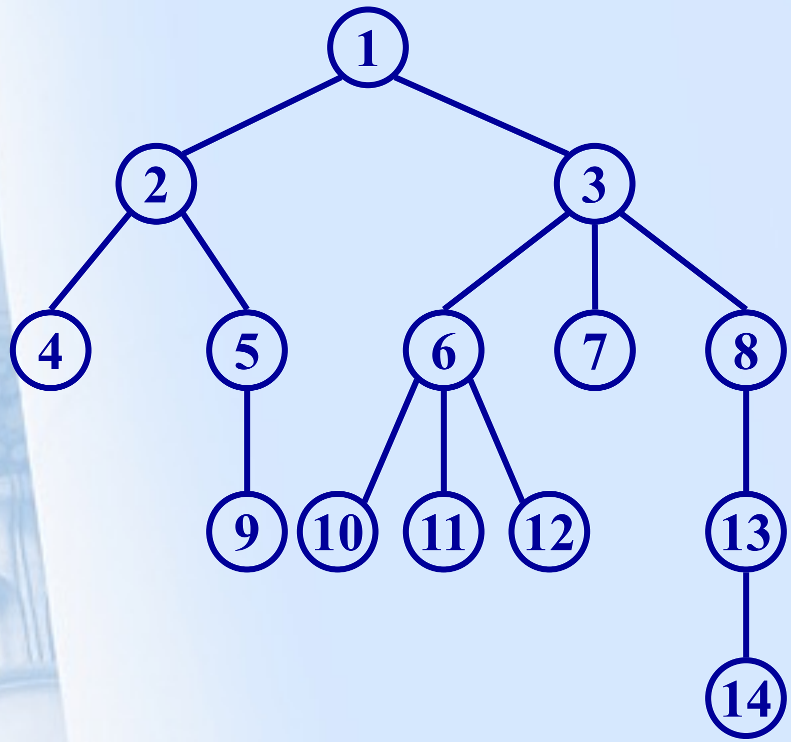
Time Complexity: $O(|V|)$

$\alpha'(T, r) = a + b + 1 - \lceil (a - c + b - d) / 2 \rceil$

$\alpha_0'(T, r) = \alpha_0'(T_1, r) + \alpha'(T_2, r')$

6.3 Dynamic Programming for Matching on Trees

- ex:



	14	13	12	11	10	9	8	7	6	5	4	3	2	1
α'														
α_0'														