Chapter 3

TREES OF HEIGHT 1 AND 2

In this chapter we define Perfect Internally Vertex Disjoint Path Matching (Perfect IVDP) and establish the necessary and sufficient conditions for the existence of Perfect IVDP matching for the trees of height 1 and 2. Trees of height 1 and 2 are simple and easy to handle. So first we discuss the existence of Perfect IVDP matching for trees of height 1 and 2. We develop sequential and parallel algorithms for determining the existence. Here odd and even trees are treated separately.

3.1. Introduction

![Figure 3.1. A Graph G](image)

Figure 3.1. A Graph G
Two paths are said to be *vertex disjoint* if they don’t have any vertex in common. They are internally vertex disjoint if no vertex is an internal vertex of both the paths.

Consider the paths $P_1$ and $P_2$ of the graph shown in Figure 3.1 where,

$$P_1 = (a, b, c, d, e)$$

$$P_2 = (c, f, g)$$

$P_1$ and $P_2$ are *Internally Vertex Disjoint* but they are not *vertex disjoint*. From the definition, *every pair of vertex disjoint paths are internally vertex disjoint*. The converse is not true. For example consider the paths $P_1$ and $P_2$ shown in Figure 3.1. They are internally vertex disjoint but not vertex disjoint. A set of paths $P$ in a graph $G$, is said to be an *internally vertex disjoint path matching* (IVDP) if it is a path matching and every pair of paths in $P$ are *Internally Vertex Disjoint*. 
3.2. IVDP Matching in Trees

There are trees having no perfect IVDP matchings. For example, the Tree shown in Figure 3.2 has no perfect IVDP matching. Theorem 3.2.1 and 3.2.2 establish the equivalent conditions for the existence of perfect IVDP matching for even and odd trees respectively.

Figure 3.2. A tree $T$ without IVDP matching.
**Theorem 3.2.1**

Let $T$ be an even tree (tree with even number of nodes). If a node $u$ of $T$ has more than three leaf children, then $T$ doesn't have a Perfect IVDP matching.

**Proof**

Assume $T$ is an even tree with a node $u$ having more than 3 leaf children.

Suppose $T$ has an IVDP matching $P$. Let $v_1, v_2, v_3, v_4$ be leaf children of $u$, as shown in Figure 3.3.

Let $P_1, P_2, P_3, P_4$ denote the matching paths of $v_1, v_2, v_3$ and $v_4$. Therefore, $u$ belongs to $P_i (1 \leq i \leq 4)$
Case 1

Suppose $u$ is matched to some $v_i$ ($1 \leq i \leq 4$).

For certainty, without loss of generality assume that $u$ is matched to $v_1$. Then $u$ is an internal vertex of $P_2$, $P_3$ and $P_4$. Among $P_2$, $P_3$ and $P_4$ at the most two may be identical. So, $u$ is an internal vertex of at least two paths. This contradicts the definition of IVDP matching.

Case 2

Suppose $u$ is not a match to $v_i$ ($1 \leq i \leq 4$). $u$ is an internal vertex of $P_1$, $P_2$, $P_3$ and $P_4$. Among $P_1$, $P_2$, $P_3$ and $P_4$ at least two are non-
identical. So \( u \) is an internal vertex of at least two paths. This contradicts the definition of IVDP matching.

**Theorem 3.2.2**

Let \( T \) be an odd tree (tree with odd number of nodes). If a node \( u \) of \( T \) has more than four leaf children, then \( T \) doesn't have a perfect IVDP matching.

**Proof**

Assume \( T \) is a tree with odd number of nodes such that \( T \) has a node \( u \) having more than 4 children which are leaf nodes.

Suppose \( T \) has an IVDP matching \( P \). Let, \( v_1, v_2, v_3, v_4, v_5 \) be the leaf nodes of \( T \) which are children of \( u \), as shown in the Figure 3.4. Let \( P_1, P_2, P_3, P_4, P_5 \) denote the matching paths of \( v_1, v_2, v_3, v_4 \) and \( v_5 \). Therefore \( u \) belongs to \( P_i (1 \leq i \leq 5) \)
Figure 3.4. A tree $T$ having more than 4 leaf children

**Case 1**

Suppose $u$ is matched to $v_i$ ($1 \leq i \leq 5$). Without loss of generality assume $u$ is matched to $v_1$. Then, $u$ is an internal vertex of $P_2$, $P_3$, $P_4$ and $P_5$. Among $P_2$, $P_3$, $P_4$ and $P_5$ at the most two may be identical. So, $u$ is an internal vertex of at least two paths. This contradicts the definition of IVDP matching.

**Case 2**

Suppose $u$ is not a match to $v_i$ ($1 \leq i \leq 5$).

Since every $v_i$ can be matched by a path passing through $u$, $u$ is an internal vertex of $P_1$, $P_2$, $P_3$, $P_4$ and $P_5$. Among $P_1$, $P_2$, $P_3$, $P_4$ and $P_5$
at least two are distinct paths. So, \( u \) is an internal vertex of at least two paths. This contradicts the definition of IVDP matching.

3.3. Trees of height 1

In this section we will construct a perfect IVDP matching for trees of height 1.

**Theorem 3.3.1**

An even tree \( T \) of height 1 has a perfect IVDP matching if and only if \( T \) has at the most 4 nodes.

**Proof:**

Let \( T_i \) denote the tree of height 1 with \( i \) nodes. Trivially \( T_2 \) has IVDP matching. In \( T_4 \) shown in Figure 3.5, \( \{ (a, b), (c, a, d) \} \) is a perfect IVDP matching. By theorem 3.2.1 trees \( T_6, T_8, \ldots \) do not have perfect IVDP matching.
**Theorem 3.3.2**

Let $T$ be an odd tree of height 1. If the tree $T$ has seven nodes then $T$ has no perfect IVDP matching.

**Proof**

The proof of this theorem is similar to the proof of Theorem 3.3.1.

**Theorem 3.3.3**

Let $T$ be an odd tree in which there exists two nodes $u$ and $v$ each having four leaf children. Then $T$ has no IVDP matching.
\textbf{Proof}

Let \( u \) and \( v \) have the leaf children \( v_1, v_2, v_3, v_4 \) and \( v_5, v_6, v_7, v_8 \) respectively.

Suppose \( T \) has an IVDP matching. As per the definition of IVDP matching, \( u \) is not an internal vertex of more than one matching paths. This is possible only if \( u \) is matched to \( v_1 \), and \( v_2 \) to \( v_3 \). This forces \( v_4 \) to be left unmatched in \( T \).

In the odd tree \( T \), the only node left unmatched is \( v_4 \). Now consider the nodes \( v, v_5, v_6, v_7, v_8 \). Among these vertices nothing can be left out. Let \( P_5, P_6, P_7, P_8 \) be the matching paths of \( v_5, v_6, v_7, v_8 \) respectively. Each path \( P_5, P_6, P_7, P_8 \) passes through \( v \). Since \( v \) can be an end vertex of at the most one path, \( v \) will be an internal vertex of more than two paths. This contradicts the definition of IVDP matching. Hence \( T \) does not have IVDP matching.

\textbf{Theorem 3.3.4}

A tree of height 1 has IVDP matching if and only if it has at the most five nodes.
Proof

Let $T_n$ denote the tree of height 1 with $n$ nodes.

$T_n$ has a IVDP matching for $n=1,2,3$ trivially. When $n=4$, the matching is done as shown in Figure 3.6(a).

When $n=5$, let $r$ be the root node and $a,b,c,d$ be the children. $a$ is matched to $b$ and $r$ to $c$. $d$ is the unmatched node.

Conversely let $T$ be a tree of height 1 with more than 5 nodes. Let the leaf nodes be $v_1, v_2, v_3, v_4$ and $v_5$. Let $P_1,P_2,P_3,P_4,P_5$ be the matching paths of $v_1,v_2,v_3,v_4$ and $v_5$ respectively. Since $r$ is the root all the above paths pass through $r$. Among $P_1,P_2,P_3,P_4$ and $P_5$ at least three are nonidentical. Hence $r$ will be an internal vertex of at least two matching paths. Therefore $T$ cannot have IVDP matching.

A tree with a perfect IVDP matching is called an IVDP tree. Even and odd trees of height 1 with perfect IVDP matching is given in the Figure 3.6(a) and Figure 3.6(b).
Figure 3.6(a) Even trees of height 1 having IVDP matching

Figure 3.6(b) Odd trees of height 1 having IVDP matchings

In this section we mention the algorithm to check if a tree of height 1 is IVDP. This can be implemented in $O(1)$ time.
3.4 Algorithm for tree of height 1

The following is a very simple algorithm to find if a tree of height 1 is IVDP.

Algorithm IsIVDPHtT (T)

Input: Tree T of height 1 in the form of parent array. \( n \) is the number of nodes. The nodes are numbered from 1 to \( n \). \( p[i] \) is the parent of node \( i \). The parent of root is itself. It is given that the height of the tree is 1.

Output: A Boolean value \( result \) to say if the tree is IVDP.

1. If \( n \leq 5 \) the tree is IVDP

   \[ \text{So } result = \text{ true} \]

   else

   The tree is not IVDP,

   \[ \text{So } result = \text{ false} \]
Complexity Analysis

Since \( n \) itself is given as an input, this can be done in \( O(1) \) time.

This leads to the following theorem:

**Theorem 3.4.1**

In a tree \( T \) of height 1 verification of the existence of a perfect IVDP can be done in \( O(1) \) time.

3.5. Even Trees of Height 2

In this section we develop algorithm to verify the existence of perfect IVDP matching in trees of height 2. Let \( T \) be an even tree of height 2. Let \( r \) be the root of \( T \). Let \( n_{odd} \) denotes the number of odd children, \( n_{even} \) denotes the number of even children and \( l \) denotes the number of leaf nodes of the root node \( r \).
Example

Consider the tree shown in Figure 3.7. \( r \) is the root. \( a \) is a child of \( r \).

The maximal subtree with \( a \) as the root has the nodes \( a, h, \) and \( k \). So, \( a \) is an odd child of \( r \). \( b \) is also an odd child of \( r \). \( c, d \) and \( e \) are even children of \( r \). Hence in this case

\[
\begin{align*}
n &= 17 \\
n_{\text{odd}} &= 2 \\
n_{\text{even}} &= 3 \\
l &= 2
\end{align*}
\]

Figure 3.7 A tree T
Theorem 3.5.1

Let $T$ be an even tree with root $r$ of height 2. $T$ has a perfect IVDP matching if and only if the following two conditions are satisfied.

1. The sum of the number of leaf children of $r$ and the number of odd children of $r$ is at the most 3.

2. Each odd and even subtree of $r$ has a perfect IVDP matching.

Proof

Let $T$ be an even tree of height 2 such that the condition 1 and 2 are satisfied. Each subtree has perfect IVDP matching. In even subtree all the nodes are matched within the subtree itself.

In the case of odd subtrees all the nodes are matched within the subtree except one unmatched node. This unmatched node has to be matched either to the root $r$ or to any other node of another subtree through the root $r$. This leads to the fact that totally $(n_{odd} + l)$ nodes are to be matched to either $r$ or through $r$. Hence $(n_{odd} + l)$ must be less than or equal to 3.
Conversely suppose one of the two conditions fail.

In the case of odd subtrees all the nodes are matched within the subtree except one unmatched node. This unmatched node has to be matched either to the root $r$ or to a node through the root $r$. This leads to the fact that totally $(n_{odd} + l)$ nodes are to be matched either to $r$ or through $r$. So if $(n_{odd} + l) > 3$, perfect IVDP matching cannot be done for these $(n_{odd} + l)$ nodes and $r$.

If any of the odd or even subtrees doesn’t have a perfect IVDP matching, then the tree $T$ doesn’t have a perfect IVDP matching.

This proves the theorem.

**Theorem 3.5.2**

Let $T$ be an even tree of height 2. $T$ has a perfect IVDP matching if it satisfies the following three conditions

1) $(n_{odd} + l) \leq 3$
2) The root does not have a subtree isomorphic to $T_5$, where $T_5$ is a tree of height 1 with 5 nodes.

3) Each subtree of $r$ has a perfect IVDP matching.

**Proof**

Assume that the three conditions hold.

Let $r$ be the root. By hypothesis each of the even subtrees has an IVDP matching. We have to identify the matching paths of $r$, the leaf nodes and the odd subtrees.

**Case 1**

\[ n_{\text{odd}} = 0 \quad \text{and} \quad l = 3. \]

Let $v_1, v_2, v_3$ be the leaf children of $r$. $r$ is matched to $v_1$ and $v_2$ is matched to $v_3$. This is shown in the Figure 3.8.
Figure 3.8. A tree with $n_{odd} = o$ and $l = 3$

\[ P = \{(r, v_1), (v_2, r, v_3)\} \]

Case 2

$n_{odd} = 1$ and $l = 2$

Figure 3.9. A tree with $n_{odd} = l$ and $l = 2$

\[ P = \{(a, v_1, b)(v_1, r), (v_2, r, v_3)\} \]
By condition 2, the odd subtree can only be $T_3$. Let $v_1$, be the root node of this odd subtree and $a,b$ its children. $a$ will be matched to $b$. $v_l$ will be matched to $r$. The two leaf children of $r$ are matched to each other through $r$ as shown in the Figure 3.9.

**Case 3**

$n_{odd}=2 \quad l=1$

By condition 2 these odd subtrees can only be $T_3$. Let $v_2,v_3$ be the root nodes of these $T_3$'s and $c_1,d_1$ and $c_2,d_2$ be its children respectively. Consider the tree shown in the Figure 3.10.

![Figure 3.10. A tree with $n_{odd} = 2$ and $l = 1$](image)

$$P = \{(c_1, v_2, d_1)(c_2, v_3, d_2), (v_2, r, v_3), (r, v_4)\}$$
$c_1$ and $d_1$ are matched to each other through $v_2$. $c_2$ and $d_2$ are matched to each other through $v_3$. The nodes $v_2$ and $v_3$ are matched to each other through $r$. $r$ is matched to its leaf child $v_4$.

Case 4

$n_{odd}=3 \quad l=0$

By condition 2 these odd subtrees can only be $T_3$. Let $v_4, v_5, v_6$ be the root nodes of these $T_3's$ and $e_1, f_1, e_2, f_2, e_3, f_3$ are their children. The nodes $e_1$ and $f_1$ are matched to each other through $v_4$. $e_2$ and $f_2$ are matched to each other through $v_5$. $e_3$ and $f_3$ are matched to each other through $v_6$. $v_4$ and $v_5$ are matched to each other through $r$. $v_6$ and $r$ are matched to each other as shown in the Figure 3.11.

![Figure 3.11. A tree with $n_{odd} = 3$ and $l = 0$](image)

$P = \{(e_1, v_4, f_1), (e_2, v_5, f_2), (e_3, v_6, f_3), (v_4, r, v_5), (v_6, r)\}$
Conversely let $T$ be a tree of height 2 having a perfect IVDP matching $P$. We will prove that the three conditions hold.

**Case 1**

Suppose condition (i) Fails. Therefore $(n_{odd} + l) \geq 4$. For each odd subtree all but one vertex are matched among the subtree itself. The leftover node must be matched to another node through the root $r$. Similarly any leaf child of $r$ must be matched through $r$. This means that $r$ is a node in the matching paths of at least 4 other nodes. Hence $r$ is an internal node for at least 2 non-identical matching paths. So $T$ is not perfect IVDP.
Case 2

Suppose there exists a subtree isomorphic to $T_5$. Let $a$ be the root of $T_5$ and $b, c, d, e$ be its leaf nodes. In $T_5$, the perfect IVDP matching leaves one leaf node (say $e$) unmatched. The other matchings are $(a, b), (c, a, d)$. Since $T$ is an even subtree $e$ must be matched to a node outside $T_5$. The node $a$ will become an internal node of this matching path of $e$. Already $a$ is an internal node of $(c, a, d)$. Therefore the matching is not perfect IVDP.

Case 3

Suppose a subtree is not a perfect IVDP. Then $T$ also is not perfect IVDP.
3.6 Odd trees of height 2

Theorem 3.6.1

Let $T$ be an odd tree of height 2. $T$ has a perfect IVDP matching if it satisfies the following three conditions.

1) The root has at the most one subtree isomorphic to $T_5$.

Where $T_5$ is the tree of height 1 with 5 nodes.

2) If $T_5$ is present then $n_{odd} + l \leq 3$

else $n_{odd} + l \leq 4$

3) Each subtree of $r$ has a perfect IVDP matching.

Proof

Sufficient condition:

Case 1

Let $T$ doesnot have a subtree isomorphic to $T_5$. For each even subtree the nodes are matched among itself. For each odd subtree, one vertex
is left unmatched. These unmatched vertices must be matched to another vertex through the root $r$. This unmatched vertex is always the root of the subtree. So there are at the most four vertices to be matched through $r$. Among these four vertices, $a, b, c, d$ one must be left unmatched (say $d$). $a$ must be matched to the root $r$ and $b$ matched to $c$. This defines a perfect IVDP matching.

Case 2

There is one subtree isomorphic to $T_3$.

In $T_3$, remove one leafnode. Now it becomes $T_4$ and the whole tree becomes an even subtree satisfying the conditions of the theorem 3.6.

Hence it has a perfect IVDP matching.
Necessary condition:

Case 1

Suppose condition 2 fails

\[(n_{\text{odd}} + l) \geq 5.\]
For certainty assume \((n_{\text{odd}} + l) = 5.\)

Case 1(a)

\[n_{\text{odd}} = 0 \quad \text{and} \quad l = 5\]

Let \(v_1, v_2, v_3, v_4, v_5\) be the leaf of children of \(r.\) \(r\) is matched to any one of its leaf children. This means that \(r\) is an internal node for at least 2 different matching paths. So \(T\) is not perfect IVDP. This is shown in the Figure 3.12.

![Figure 3.12. A tree T](image-url)
Case 1(b)

\[ n_{\text{odd}} = 1 \quad \text{and} \quad l = 4 \]

Let \( v_1, v_2, v_3, v_4 \) be the leaf children of \( r \) and there may exist one odd subtree of \( r \). Let \( v_5 \) denote the node of the odd subtree that is matched outside. \( r \) can be matched to any one of the leaf children. All the other nodes \( v_2, v_3, v_4, v_5 \) must be matched through \( r \). This also leads to \( r \) becoming an internal node for more than one path. Therefore the matching is not perfect IVDP. This is shown in the Figure 3.13.

**Figure 3.13.** A tree \( T \) with one odd subtree
Case 1(c)

\[ n_{\text{odd}} = 2 \quad \text{and} \quad l = 3 \]

Let \( v_1, v_2 \) be the root nodes of the odd subtrees. In the odd subtrees all the other nodes may be matched except one. So ultimately 5 leaf nodes are left unmatched. One of its leaf node is matched to \( r \) and all the other four leaf nodes are matched through \( r \).

This leads to more than two distinct paths through \( r \). Hence the tree \( T \) is not perfect IVDP. This is shown in the Figure 3.14.

![Figure 3.14. A tree T with 2 odd subtrees](image-url)
Case 1(d)

\[ n_{odd} = 3 \quad \text{and} \quad l = 2 \]

Let \( v_1, v_2, v_3 \) be the root nodes of the odd subtrees and \( v_4, v_5 \) be the leaf nodes. In the odd subtrees all the other nodes may be matched except one. So five leaf nodes are left unmatched. One of its leaf node is matched to \( r \) and all the other four nodes are matched through \( r \). This leads to more than two distinct paths through \( r \). Hence the tree is not perfect IVDP. This is shown in the Figure 3.15.

![Figure 3.15. A tree T with 3 odd subtrees](image-url)
Case 1(e)

\[ n_{\text{odd}} = 4 \quad \text{and} \quad l = 1 \]

Let \( v_1, v_2, v_3, v_4 \), be the root nodes of the odd subtrees and \( v_5 \) be the leaf node. In the odd subtrees all the other nodes may be matched except one. So five leaf nodes are left unmatched. One of its leaf node is matched to \( r \) and all the other four nodes are matched through \( r \). This leads to more than two distinct paths through \( r \). Hence the tree is not perfect IVDP. This is shown in the Figure 3.16.

![Diagram showing a tree with 4 odd subtrees and a leaf node unmatched](image)

**Figure 3.16. A tree T with 4 odd subtrees**
Case 1(f)

\[ n_{odd} = 5 \quad \text{and} \quad l = 0 \]

Let \( v_1, v_2, v_3, v_4 \) and \( v_5 \) be the root nodes of the odd subtrees. In the odd subtrees all the other nodes may be matched except one. So five leaf nodes are left unmatched. One of its leaf node is matched to \( r \) and all the other four nodes are matched through \( r \). This leads to more than two distinct paths through \( r \). Hence the tree is not perfect IVDP. This is shown in the Figure 3.17.

![Figure 3.17. A tree T with 5 odd subtrees](image)
Case 2

In $T_5$, the perfect IVDP matching is done as follows.

i) The root is matched to a leaf

ii) Two leaves are matched to each other

iii) One leaf is left unmatched.

Since there are two subtrees isomorphic to $T_5$, there are two leaf nodes left unmatched. At least one of them must be matched to some other node. When this is done, the root of $T_5$ becomes an internal node of more than one matching paths. Therefore $T$ is not perfect IVDP.

Case 3

Suppose a subtree does not have perfect IVDP matching. Then $T$ is also not IVDP. This proves the converse of the theorem.
The odd tree of height 2 with perfect IVDP matching and odd tree of height 2 without perfect IVDP matching are given in Figure 3.18, Figure 3.19 and Figure 3.20. Also the even tree of height 2 with perfect IVDP matching and even tree of height 2 without perfect IVDP matching are given in Figure 3.21, Figure 3.22 and Figure 3.23.
Figure 3.18. Odd tree with IVDP matching

The matching is given by \{ (a, b), (c, d), (e, f), (g, f, h), (i, j), (k, j, l), (m, n, o) (p,q,s), (t, q), (n, r) \}
Figure 3.19.
Odd tree without IVDP matching because of the presence of two $T_5$'s
Figure 3.20.
Odd tree without IVDLP matching

\[ n_{\text{even}} = 2 \]
\[ n_{\text{odd}} = 2 \]
\[ l = 4 \]
\[ n_{\text{odd}} + l = 6 > 4 \]
Figure 3.21. Even tree of height 2 with IVDP matching

The matching is given by \{ (a, b), (c, d), (e, f), (g, f, h), (i, j), (k, j, l), (m, n, o), (n, r), (p, r, q) \}
Even tree without IVDP matching
3.7 Algorithm for Trees of height 2

Theorem 3.5.2 and 3.6.1 gives the necessary and sufficient conditions for the existence of perfect IVDP matching for trees of height 2. In this section we develop the sequential and parallel algorithms for determining the existence of perfect IVDP matching for trees of height 2.

3.7.1 To find the root

When the tree is represented in the form of the parent array $p[i]$, we can identify the root as follows:

**Algorithm FindRoot** ($p$, $n$)

```
{
For $i = 1$ to $n$
If $p[i] = i$ then root = $i$
}
```

In sequential algorithm, this can be implemented in $O(n)$ time.
3.7.2 To count the number of leaf children for each node

Let $T$ be the tree with root $r$. The tree is represented in the form of parent array $p[i]$. The algorithm to find the number of leaf children is given below.

**Algorithm FindNoOfLeafChildren (p, n)**

**Input** : $p[i]$ Parent array

**Output** : 1) $C[i]$ number of children for $i$

2) $LC[i]$ number of leaf child for $i$

**Step 0** : Initialize $C[i] = 0$ and $LC[i] = 0$ for $i = 1$ to $n$.

**Step 1** : For $i = 1$ to $n$

{};

$C[p[i]] = C[p[i]] + 1$

}
Step 2: For $i = 1$ to $n$

{ 

If $C[i] = 0$

$LC[p[i]] = LC[p[i]] + 1$

}

Example

Consider the tree shown in the Figure 3.24.

![Figure 3.24. A Tree T](image-url)
Table 3.1 The node, parent, number of children and number of leaf children arrays of Figure 3.24

<table>
<thead>
<tr>
<th>i</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>p[i]</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>C[i]</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>LC[i]</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

The node $i$, for a parent $p[i]$, and the corresponding child nodes and number of leaf children are given in the Table 3.1.

**Complexity Analysis**

From the above algorithm, the time complexity is $O(n)$. So given a tree represented in the form of parent array $p[i]$, the number of children $C[i]$ and the number of leaf children $LC[i]$ for each node $i$, can be determined in $O(n)$ time. This leads to the following theorem.
Theorem 3.7.1

In a tree $T$ the sequential algorithm to count the number of children $C[i]$ and number of leaf children $LC[i]$ can be determined in linear time.

3.7.3 To count the number of odd and even children of the root

Let $T$ be the tree with root $r$. Let $i$ be a child of $r$. If $i$ is the root of the subtree, with odd number of children, $i$ is called an odd child of $r$. Similarly we can define the even child of $r$. Let $n_{\text{odd}}$ and $n_{\text{even}}$ be the number of odd and even children of $r$. Assume that $T$ is represented in the form of its parent array.

Algorithm \textit{FindOddAndEvenChildrenOfRoot} ($p$, $n$, root)

1) $\text{Find } C[i]$

2) $n_{\text{odd}} = 0$

3) $n_{\text{even}} = 0$

4) for $i = 1$ to $n$
4a) If \((p[i] = \text{root}) \text{ and } (i \neq \text{root})\)

        if \(C[i]\) is odd

            \(n_{\text{even}}++\)

        else

            \(n_{\text{odd}}++\)

        endif

From the above algorithm, the number of odd and even children can be determined in \(O(n)\) time.

3.7.4 Sequential algorithm for even trees of height 2

Let \(T\) be an even tree, of height 2 represented in the form of parent array \(p[i]\). Let \(n\) be the number of nodes. The sequential algorithm to check whether the tree is perfect IVDP is given below.
**Algorithm EvenTwo()**

**Input** : i) A tree $T$ of height 2

          ii) Parent array $p[i]$

          iii) Number of nodes $n$

**Output** : A Boolean value $result$ which indicates whether $T$ is perfect IVDP

1. Find $n_{odd} = \text{number of odd subtrees of } r$.

   $l = \text{number of leaf children of } r$

2. If $n_{odd} + l > 3$ then $result = \text{false};$ exit

   else proceed to step 3

3. For every subtree of $r$ verify if it has a perfect IVDP matching.

   If any of them doesn't have a perfect IVDP matching then

   $result = \text{false};$ exit
If all the even sub trees have perfect IVDP matching

proceed to step 4.

4. result = True

Complexity Analysis

In a tree $T$ of height 2, the sequential algorithm EvenTwo( ) checks whether the tree is IVDP which is determined as follows.

Step 1 can be found out in $O(n)$ time. Steps 2, 3 check for the existence of IVDP matching. Step 4 gives the Boolean value result which is true or false. So the algorithm is implemented in $O(n)$ time.

3.7.5 Algorithm for odd tree of height 2

Let $T$ be an odd tree of height 2 represented in the form of parent array $p[i]$. Let $n$ be the number of nodes. Then the sequential algorithm to determine the perfect IVDP matching is given below.
Algorithm OddTwo()

Input:

i) A tree $T$ of height 2

ii) Parent array $p(i)$

iii) Number of nodes $n$

Output:

A Boolean value $\text{result}$ which indicates whether $T$ is perfect IVDP

1. Find $n_{\text{odd}} =$ number of odd subtrees of $r$

   $l =$ number of leaf children of $r$

   $t_5 =$ number of subtrees with 5 nodes

2. If $t_5 > 1$ then $\text{result} = \text{false}$; exit.

3. If $(t_5 = 1)$ and $(n_{\text{odd}} + l) > 3$ then

   $\text{result} = \text{false}$; exit.

4. If $(t_5 = 0)$ and $(n_{\text{odd}} + l) > 4$ then
result = false; exit

5. For every subtree \( T \), verify if the subtree is perfect IVDP.

   If any one is not perfect IVDP,

   then result = false; exit

   If all the subtrees are perfect IVDP,

   proceed to step 6.

6. result = True

**Complexity Analysis**

The complexity of the algorithm OddTwo( ) is same as the complexity of the algorithm EvenTwo( ).
3.8. Parallel Algorithms for Trees of height 2

In this section we develop a parallel implementation of the algorithm given in the previous section. Consider a tree of height 2 represented in the form of its parent array. To implement the algorithm in parallel machines, consider the two dimensional array, \( \text{child}(i, j) \) which consists of \( n \) rows and \( n \) columns. Where \( n \) is the number of nodes, defined as follows:

\[
\text{child}(i, j) = \begin{cases} 
1 & \text{if } i \text{ is the child of } j \text{ and } i \neq j \\
0 & \text{otherwise}
\end{cases}
\]

Example

Consider the tree in Figure 3.25 with 8 nodes.

![Figure 3.25. A Tree T](image-url)
The parent relation of the above tree is given in Table 3.2

<table>
<thead>
<tr>
<th>$i$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p(i)$</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

Table 3.2. Parent relation of tree $T$

The two dimensional array $\text{child}(i, j)$ is given in Table 3.3. The column sum gives the number of child nodes

<table>
<thead>
<tr>
<th>Child</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>Column Sum</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 3.3. Number of child nodes for each $i$ of Tree $T$ in Figure 3.25
The parallel algorithm to find the number of child nodes is given below.

### 3.8.1 Parallel Algorithm to count the number of children

**Algorithm CountNumberOfChildren**

- **Input**: Tree $T$
- **Output**: $C[i]$ Number of children for $i$

1) For $i = 1$ to $n$ do in parallel

   If $i \neq p[i]$

   $Child[i, p[i]] = 1$

2) For each column $j$ do in parallel

   Find column sum in child matrix

   $C[j] = Column\ sum\ of\ j^{th}\ column$

**Complexity Analysis**

The above algorithm can be implemented in $O(\log n)$ time using $O(n^2)$ processors in EREW PRAM
**Theorem 3.8.1**

In a tree $T$ with root $r$, the number of children for each node $i$ can be determined in $O(\log n)$ time using $O(n^2)$ processors in EREW PRAM.

**Proof**

Step 1 can be implemented in $O(1)$ time with $O(n)$ processors in EREW PRAM. In order to implement step 2, $n$ processors are needed for each column. So, for computing the $n$ columns in parallel we need $O(n^2)$ processors. Since sum of $n$ numbers can be found in $O(\log n)$ time using $O(n)$ processors [XI 98] step 2 can be implemented in $O(\log n)$ time using $O(n^2)$ processors in EREW PRAM. So the algorithm can be implemented in $O(\log n)$ time using $O(n^2)$ processors in EREW PRAM.
3.8.2 Parallel Algorithm to find the number of leaf children

To find the number of leaf children in parallel, consider another two-dimensional array \( LChild[i, j] \) with \( n \) rows and \( n \) Columns, where

\[
LChild(i, j) = \begin{cases} 
1 & \text{if } i \text{ is a leaf child of } j \\
0 & \text{otherwise} 
\end{cases}
\]

Example

Consider the tree given in the Figure 3.25. Also consider the following arrays \( p[i] \) and \( C[i] \) as given in Table 3.4.

<p>| | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>p[i]</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>C[i]</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 3.4. The child, Parent, number of children of Tree \( T \) in Figure 3.25

84
The two dimensional array $LChild[i, j]$ for this tree is given in Table 3.5. The column sum of the table gives the number of leaf children for each node $i$.

<table>
<thead>
<tr>
<th>LChild</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Column sum</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 3.5. Number of leaf children for each node $i$ of Tree $T$ in Figure 3.25
The parallel algorithm to find out the number of leaf children is given below:

Algorithm CountLeafChildren

**Input**
1) Tree $T$
2) parent array $p[i]$
3) Array $C[i]$ which gives the number of children of each node $i$

**Output**
number of leaf children $LC[i]$

1) For $i = 1$ to $n$ do in parallel

   if $C[i] = 0$ then

   $LChild [i, p [i]] = 1$

2) For each column $j$ do in parallel

   Find column sum in child matrix

   $LC[j] = \text{Column sum of } j^{th} \text{ column of } LChild \text{ matrix}$
**Complexity Analysis**

The above algorithm can be implemented in $O(\log n)$ time using $O(n^2)$ processors in EREW PRAM which leads to the following theorem.

**Theorem 3.8.2**

In a tree $T$ with root $r$ the algorithm CountLeafChildren for each node $i$ can be determined in $O(\log n)$ time using $O(n^2)$ processors in EREW PRAM.

**3.8.3 Algorithm to find the number of odd and Even subtrees of root.**

Consider two arrays EVEN [i] and ODD [i]

\[
\text{EVEN [i]} = \begin{cases} 
1 & \text{if } i \text{ is a child of root and } i \text{ is the root of an even subtree} \\
0 & \text{otherwise}
\end{cases}
\]

\[
\text{ODD [i]} = \begin{cases} 
1 & \text{if } i \text{ is a child of root and } i \text{ is the root of an odd subtree} \\
0 & \text{otherwise}
\end{cases}
\]
Algorithm CountEvenOddSubtreesOfRoot \((p, n, \text{root}, C[i], L C[i])\)

**Input**: 1) Tree \(T\) of height 2

2) Parent array \(p[i]\)

3) Root of \(T\)

4) Array \(C[i]\) which gives the number of children for each node \(i\)

5) Array \(L C[i]\) which gives the number of leaf children for each node \(i\)

**Output**: Number of odd subtrees and number of even subtrees

1. For \(i = 1\) to \(n\) do in parallel

   1.1 If \((p[i] = \text{root} \text{ and } C[i] = \text{odd})\) then

      \[\text{EVEN}[i] = 1;\]

   1.2 If \((p[i] = \text{root} \text{ and } C[i] = \text{even})\) then

      \[\text{ODD}[i] = 1\]

2. Find the sum of the arrays

   \[n_{\text{odd}} = \text{sum of the array ODD}[i]\]

   \[n_{\text{even}} = \text{sum of the array EVEN}[i]\]
**Theorem 3.8.3**

The algorithm CountEvenOddSubtreesOfRoot correctly determines the value of $n_{odd}$ and $n_{even}$.

**Proof**

If $C[i]$ is odd and $p[i] = \text{root}$, then $i$ is a child of the root and $i$ has odd number of children. If $i$ is having odd number of children then $i$ is the root of a subtree with even number of nodes. Hence by step 1.1 of the algorithm EVEN $[i] = 1$ indicates that $i$ is the root of an even subtree. As per similar argument by step 1.2 of the algorithm ODD$[i] = 1$ indicates that $i$ is the root of an odd subtree. Hence $n_{odd}$ and $n_{even}$ are correctly determined.

**Complexity Analysis**

Step 1 can be implemented in $O(1)$ time using $O(n)$ processors. As sum of $n$ numbers can be computed in $O(\log n)$ time using $O(n)$ processors in EREW PRAM. Step 2 can be implemented in $O(\log n)$ time using $O(n)$ processors. So, the above algorithm can be
implemented in $O(\log n)$ time using $O(n)$ processors in EREW PRAM. This leads to the following result.

**Theorem 3.9.4**

In a tree $T$ of height 2 with root $r$, the number of odd subtrees $n_{odd}$ and the number of even subtrees $n_{even}$ can be determined in $O(\log n)$ time using $O(n)$ processors in EREW PRAM.

3.8.4 Parallel Algorithm for even trees of height 2.

**Algorithm EvenIVDPhight2**

**Input**: i) A tree $T$ of height 2

ii) Parent array $p[i]$

iii) Number of nodes $n$

**Output**: A Boolean value $result$ which indicates whether $T$ is perfect IVDP
1. Find $n_{odd} = \text{number of odd subtrees of } r$

$$l = \text{number of leaf children of } r$$

2. If $n_{odd} + l > 3$ then result = false; exit

else proceed to step 3

3. For each subtree $T_i$ of $r$ do in parallel

3.1 Check if $T_i$ is perfect IVDP. If $T_i$ is not perfect IVDP,

result = false: exit.

4. result = true

**Complexity Analysis**

In step 1, the number of odd subtrees $n_{odd}$ can be found in $O(\log n)$ time using $O(n)$ processors. The number of leaf children $l$ for each node $i$ can be found in $O(\log n)$ time using $O(n^2)$ processors.

Step 3 can be implemented in $O(1)$ time using $O(n)$ processors. So the above algorithm can be implemented in $O(\log n)$ time using $O(n^2)$ processors. This leads to the following result.
Theorem 3.8.5

If $T$ is an even tree of height 2, we can verify if $T$ has a perfect IVDP matching in $O(\log n)$ time using $O(n^2)$ processors in EREW PRAM.

3.8.5 Parallel Algorithm for odd trees of height 2

Algorithm OddIVDPheight2

Input:  

i) A tree $T$ of height 2

ii) Parent array $p[i]$ 

iii) Number of nodes $n$

Output: A Boolean value $result$ which indicates whether $T$ is perfect IVDP

1. Find

\[ n_{\text{odd}} = \text{number of odd subtrees of } r \]

\[ l = \text{number of leaf children of } r \]
\[ t_5 = \text{number of subtrees with 5 nodes} \]

2. If \( t_5 > 1 \) then result = false; exit

3. if \((t_5 = 1) \text{ and } (n_{odd} + l) > 3\) then

   \[ \text{result} = \text{false; exit} \]

4. if \((t_5 = 0) \text{ and } (n_{odd} + l) > 4\) then

   \[ \text{result} = \text{false; exit} \]

5. For every subtree \( T_i \) do in parallel

   5.1 Check if \( T_i \) is perfect IVDP. If \( T_i \) is not perfect IVDP,

   \[ \text{result} = \text{false; exit} \]

6. result = true

**Complexity Analysis**

In step 1, the number of odd subtrees \( n_{odd} \) can be found in \( O(\log n) \) time using \( O(n) \) processors. The number of leaf children \( l \) for each node \( i \) can be foundout in \( O(\log n) \) time using \( O(n^2) \) processors. \( t_5 \) is implemented in \( O(l) \) time using \( O(n) \) processors.
Step 5 can be implemented in $O(1)$ time using $O(n)$ processors. So the above algorithm can be implemented using $O(\log n)$ using $O(n^2)$ processors. This leads to the following result.

**Theorem 3.8.6**

If $T$ is an odd tree of height 2, we can verify if $T$ has perfect IVDP matching in $O(\log n)$ time using $O(n^2)$ processors in EREW PRAM.