In this chapter we develop the sequential algorithm to check whether a tree is perfect IVDP or not. First we do the negativity testing. After passing the negativity testing we do the trim operation. The sequential algorithm is explained with examples.

4.1 Negativity Testing

Let $T$ be a tree with root $r$. $C[i]$ and $LC[i]$ denote the number of children and the number of leaf children for node $i$.

1. If $LC[i] > 4$ for a node $i$ then $T$ does not have a perfect IVDP matching.

2. If $T$ is an even tree and $LC[i] > 3$ for a node $i$, then $T$ does not have a perfect IVDP matching.

3. If $T$ is an odd tree and there are two distinct nodes $i$ and $j$ with $LC[i] = LC[j] = 4$ then $T$ does not have a perfect IVDP matching.
4.2 Trim Operation

If $T$ is a given tree, the following is defined as *trim* operation on $T$.

1. If there is a node $i$ with $LC[i] = C[i] = 1$ or $LC[i] = C[i] = 3$ then remove $i$ from the tree together with all its children.

2. If there is a node $i$ with $LC[i] = C[i] = 2$ then remove the children of $i$ from the tree.

In step 1 of the trim operation, we identify nodes having 3 leaf children but no non-leaf child and remove it with its children. In step 2 we identify nodes with 2 leaf children but no non-leaf child and remove only the children. The above concepts are explained with the following example.

Consider a tree shown in Figure 4.1. Its parent child array along with the number of children and number of leaf children are given in the Table 4.1.
Figure 4.1. A tree $T$
<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>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
</tr>
</thead>
<tbody>
<tr>
<td>P[i]</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>29</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>42</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>C[i]</td>
<td>5</td>
<td>3</td>
<td>3</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>LC[i]</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>i</th>
<th>15</th>
<th>16</th>
<th>17</th>
<th>18</th>
<th>19</th>
<th>20</th>
<th>21</th>
<th>22</th>
<th>23</th>
<th>24</th>
<th>25</th>
<th>26</th>
<th>27</th>
<th>28</th>
</tr>
</thead>
<tbody>
<tr>
<td>P[i]</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>17</td>
<td>17</td>
<td>18</td>
<td>3</td>
<td>3</td>
<td>13</td>
<td>14</td>
</tr>
<tr>
<td>C[i]</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>LC[i]</td>
<td>0</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>i</th>
<th>29</th>
<th>30</th>
<th>31</th>
<th>32</th>
<th>33</th>
<th>34</th>
<th>35</th>
<th>36</th>
<th>37</th>
<th>38</th>
<th>39</th>
<th>40</th>
<th>41</th>
<th>42</th>
<th>43</th>
</tr>
</thead>
<tbody>
<tr>
<td>P[i]</td>
<td>32</td>
<td>29</td>
<td>29</td>
<td>15</td>
<td>32</td>
<td>33</td>
<td>32</td>
<td>35</td>
<td>32</td>
<td>37</td>
<td>37</td>
<td>32</td>
<td>40</td>
<td>40</td>
<td>14</td>
</tr>
<tr>
<td>C[i]</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>LC[i]</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 4.1. Child, Parent Array of tree $T$ in Figure 4.1
4.3 Trim operation 1 on tree $T$

Consider the tree $T$ in the Figure 4.1. The iterations are explained below:

**Iteration 1**

1. Node 3 has 3 leaf children 4, 25, 26 and no non-leaf child. So remove 3, and its children 4, 25, 26 from the tree.

2. Node 13 has one leaf child 27 and no non-leaf child. So remove 13 and its leaf child 27 from the tree.

3. Node 14 has 2 leaf children 28 and 43 and no non-leaf child. So remove the children 28 and 43.


5. Node 17 has two leaf children 22 and 23 and no non-leaf child. So remove the leaf children 22 and 23.

7. Node 5 has 3 leaf children 6, 7 and 8 and no non-leaf child. So remove 5 and its leaf children 6, 7 and 8.

8. Node 33 has one leaf child 34 and no non-leaf child. So remove 33 and its leaf child 34.


10. Node 37 has two leaf children 38 and 39 and no non-leaf child. So remove the leaf children 38 and 39.

11. Node 9 has three leaf children 10, 11 and 12 and no non-leaf child. So remove the node 9 and its leaf children 10, 11 and 12.
The resultant tree after the first trim operation is shown in the Figure 4.2.
Consider the tree after the first trim operation in Figure 4.2. On this tree we can do the trim operation again. The resulting tree will be as shown in Figure 4.3. The following removals will take place.

1. Node 2 has one leaf child 14 and no non-leaf child. So remove node 2 and its leaf child 14.

3. Node 40 has two leaf children 41 and 42 and no non-leaf child. So remove the leaf children 41 and 42.

The resultant tree is shown in Figure 4.3.

**Theorem 4.3.1**

Let $T$ be a tree successfully passing negativity test. Let $T^*$ be the tree got by doing trim operation on $T$. $T^*$ has a perfect IVDP matching if and only if $T$ has a perfect IVDP matching.

**Proof**

Suppose $T$ has perfect IVDP matching, $T^*$ is got by removing nodes in three categories.

**Category 1**

Let $u$ be a node such that $v$ is its only child and $v$ is a leaf as shown in Figure 4.4. Here $u$ and $v$ are both removed. In this case, in any
perfect IVDP matching of $P$, $u$ must be matched to $v$. So, the removal of $u$ and $v$ will not damage the perfect IVDP matching of $P$.

$$\begin{array}{c}
\text{Figure 4.4. A node with a child}
\end{array}$$

**Category 2**

Let $u$ be a node such that $v_1$, $v_2$, $v_3$ are the leaf children as shown in Figure 4.5. In this case $u$, $v_1$, $v_2$, $v_3$ are removed. In this case in any perfect IVDP matching of $P$, $u$ must be matched to any one of its child nodes $v_1$, $v_2$ and $v_3$ and other two nodes are matched to each other through $u$. So this removal of $u$, $v_1$, $v_2$, $v_3$ will not damage the perfect IVDP matching of $P$. 

104
Let $u$ be a node such that $v_1$ and $v_2$ are the leaf children of $u$ and $u$ has no other children as shown in the Figure 4.6. In this case in any perfect IVDP matching of $T$, two of the nodes $u$, $v_1$, $v_2$ are matched to each other and one node is matched to some other node. Without loss of generality, we can assume that $v_1$ and $v_2$ are matched to each other and $u$ is matched to some other node. So removal of $v_1$ and $v_2$ will not damage any perfect IVDP matching.
Because of the above arguments, removal of nodes in trim operation will not damage the perfect IVDP matching of $T$. So if $T$ has a perfect IVDP matching, $T^*$ also has.

Conversely, assume that $T^*$ has a perfect IVDP matching. The matching paths for the removed vertices can be defined in $T$ as follows.

**Category 1**

$u$ is matched to $v$, as shown in Figure 4.7.
Figure 4.7. A node $u$ with the child $v$

**Category 2**

$u$ is matched to $v_1$. $v_2$ is matched to $v_3$, as shown in Figure 4.8.

Figure 4.8. A node $u$ with child nodes $v_1$, $v_2$ and $v_3$

**Category 3**

$v_1$ is matched to $v_2$, as shown in Figure 4.9.
Clearly the addition of matching paths defined above will not violate any of the characteristics of a perfect IVDP matching. So $T$ also has a perfect IVDP matching.

If a node is not removed it is said to be in *active state* $A$. If it is removed it is said to be *removed state* $R$. $\text{STATUS}[]$ is an array which represents whether a node is in active state or not.

$\text{AC}[]$ is an array which denotes the number of active children of node $i$.

$\text{ALC}[]$ is an array which denotes the number of active leaf children of node $i$. 
In a subtree with five nodes, all the nodes are matched except one unmatched node. That node must be avoided. We define a Boolean variable AVOIDED to indicate that in \( t_5 \) such node is avoided or not.

## 4.4 Sequential algorithm for trees of arbitrary height

In this section we develop sequential algorithm to check whether a tree \( T \) is perfect IVDP. It is represented in the form of parent array \( p[i] \). First we find the number of children \( C[i] \) and number of leaf children \( LC[i] \) for each node \( i \). Then we do the negativity test. We do the trim operation iteratively until we get the boolean value result which is true or false. The sequential algorithm is given below.

**Algorithm SequentialIVDP**

**Input**: i) A tree \( T \) with root \( r \) represented in the form of parent array.

**Output**: A Boolean variable \( result \) which indicates whether \( T \) has an IVDP matching.

Global variable AVOIDED = false
0. Define an array STATUS(i) which indicates whether the node i exists in the tree or has been removed. It may have A to denote that the node is active and R to denote that the node is removed.

1. Find the array C[i], i = 1 to n where C[i] is the number of children for the node i.

2. Find the array LC[i], i = 1 to n where LC[i] is the number of leaf children for the node i.

3. AC[i] = C[i], i = 1 to n

4. ALC[i] = LC[i], i = 1 to n

newn = n

Negativity test

5. For i = 1 to n

   5.1 If ALC[i] > 4 then result = false; exit

   5.2 If newn is even and ALC[i] > 3 then

       result = false; exit

6. For i = 1 to n and j = 1 to n and i ≠ j

   if ALC[i] = ALC[j] = 4 then

       result = false; exit.
7. For i = 1 to n

    if ALC[i] == ALC[j] == 4 and AVOIDED = True then

    result = false; exit.

Process

//Removing K2 leaves//

8. For i = 1 to n

8.1 If (STATUS[i] = A)

And (AC[p[i]] = ALC[p[i]] = 1)

{

    STATUS[i] = R
    STATUS[p[i]] = R
    AC[p[i]] = AC[p[i]]-1
    ALC[p[i]] = ALC[p[i]]-1
    AC[p[p[i]]] = AC[p[i]]-1
    ALC[p[p[i]]] = ALC[p[p[i]]]-1
    newn = newn - 2

}

9. For i = 1 to n

9.1 If (STATUS[i] = A)
and \((AC[p[i]] = ALC[p[i]] = 3)\)

{  
//remove i, its siblings and \(p[i]\)//

for \(j = 1\) to \(n\)

{
  
  if \(p[j] = p[i]\)
  
  STATUS\([j]\) = R

}

STATUS\([p[i]\]) = R

\(AC[p[i]] = AC[p[i]] - 3\)

\(ALC[p[i]] = ALC[p[i]] - 3\)

\(AC[p[p[i]]] = AC[p[p[i]]] - 1\)

\(ALC[p[p[i]]] = ALC[p[p[i]]] - 1\)

\(newn = newn - 4\)

}

10. For \(i = 1\) to \(n\)

10.1 If \((STATUS\([i]\) = A)\)

and \((AC[p[i]] = ALC[p[i]] = 2)\)

{  

//remove i and its sibling

for j = 1 to n
{
    if p[j] = p[i]
    STATUS [j] = R
}

STATUS [p[i]] = R
AC[p[i]] = AC [p[i]] - 2
ALC[p[i]] = ALC [p[i]] - 2
AC [p[p[i]]] = AC [p[p[i]]] - 1
ALC [p[p[i]]] = ALC [p[p[i]]] - 1
newn = newn - 2
}

11. For i = 1 to n

11.1 If (STATUS [i] = = A)
    and (AC[p[i]] = ALC [p[i]] = -4)
{
    //Avoid i and remove i, siblings of i and p[i]
    for j = 1 to n
\{ 
  if \( p[j] = p[i] \)
  \textit{STATUS}[j] = R
\}

\textit{STATUS}[p[i]] = R
\textit{AC}[p[i]] = \textit{AC}[p[i]] - 4
\textit{ALC}[p[i]] = \textit{ALC}[p[i]] - 4
\textit{AC}[p[p[i]]] = \textit{AC}[p[p[i]]] - 1
\textit{ALC}[p[p[i]]] = \textit{ALC}[p[p[i]]] - 1
\textit{AVOIDED} = \textsc{true}
newn = newn - 5
\}

12. \( n = \textit{newn} \)
13. \textit{If} \( n > 1 \) \textit{goto step 3 for negativity test}

\textit{else result} = \textsc{true}; \textit{exit}
The above algorithm's flow is explained below:

As a required data structure, first find the number of children for every node $i$ as $C[i]$. We also need the number of leaf children for each node. This is also found as $LC[i]$.

In step 3, we initialize an array to indicate whether a node is active or removed. Initially all the nodes are active. So assign the children of each node $i$ which is $C[i]$ to active children of $i$ which is also a single dimensional array $AC[i]$.

In step 4, initially all the leaf children are active leaf children. So assign the single dimensional array leaf children of each node $i$ $LC[i]$ to active leaf children of $i$ which is also a single dimensional array $ALC[i]$. Initially we assign $newn$ as $n$ since no node is removed. Each time when nodes are removed, the value of $newn$ is decreased.

In an odd tree we can avoid one node. For that we use the flag AVOIDED to indicate that a node is avoided or not.
Step 5 to 7 explains the negativity test discussed earlier.

Step 8 gives the node with a single child. So the STATUS of child and parent were changed to removed state $R$. Then $\text{newn}$ value is decreased by 2 since we remove 2 nodes. As we remove the nodes the Active leaf children and active children of $i$ and its parent were decreased by 1.

The same process is done for subtrees with 3 children as explained in step 9. Here $i$ and its siblings are removed.

Step 10 explains the subtree with 2 children. In this case only the child nodes are removed except the parent.

Step 11 explains the subtree with 5 nodes. In an odd tree one such $T_5$ can exist. We can avoid one vertex from $T_5$ making the value of $\text{AVOIDED} = \text{TRUE}$ and we remove the remaining 3 children along with their parent. The same process is repeated until the $\text{newn}$ is greater than 1. Finally we arrive at the result that whether the tree $T$ has IVDP matching or not.
Complexity Analysis

In step 1, the number of children $C[i]$ can be found in $O(n)$ time. In step 2 the number of leaf children $LC[i]$ can be found out in $O(n)$ time. In step 3, the number of active children can be found out in $O(n)$ time. In step 4 the number of active leaf children can be found out in $O(n)$ time. Step 7 is implemented in $O(n)$ time. In the process part steps 8, 9 and 10 are implemented in $O(n)$ time. Steps 9, 10, 11 are implemented in $O(n^2)$ time. So the algorithm to check the existence of IVDP matching in a tree is implemented in $O(n^2)$ time.

4.5 Illustration

Consider the tree shown in Figure 4.1. For each node $i$, the parent $p[i]$, the number of children $C[i]$ and number of leaf children $LC[i]$ are found out. Also the active children $AC[i]$ for each $i$, the active leaf children $ALC[i]$ for each node is given in table 4.2(a).
<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>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
</tr>
</thead>
<tbody>
<tr>
<td>p[i]</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>29</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>42</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>C[i]</td>
<td>5</td>
<td>3</td>
<td>3</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>LC[i]</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AC[i]</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>ALC[i]</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>STATUS</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>i</th>
<th>15</th>
<th>16</th>
<th>17</th>
<th>18</th>
<th>19</th>
<th>20</th>
<th>21</th>
<th>22</th>
<th>23</th>
<th>24</th>
<th>25</th>
<th>26</th>
<th>27</th>
<th>28</th>
</tr>
</thead>
<tbody>
<tr>
<td>p[i]</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>17</td>
<td>17</td>
<td>18</td>
<td>3</td>
<td>3</td>
<td>13</td>
<td>14</td>
</tr>
<tr>
<td>C[i]</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>LC[i]</td>
<td>0</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>AC[i]</td>
<td>0</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>ALC[i]</td>
<td>0</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>STATUS</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
</tr>
</tbody>
</table>

118
<table>
<thead>
<tr>
<th>i</th>
<th>29 30 31 32 33 34 35 36 37 38 39 40 41 42 43</th>
</tr>
</thead>
<tbody>
<tr>
<td>p[i]</td>
<td>32 29 29 15 32 33 32 35 32 37 37 32 40 40 14</td>
</tr>
<tr>
<td>C[i]</td>
<td>3 0 0 5 1 0 1 0 2 0 0 2 0 1 0</td>
</tr>
<tr>
<td>LC[i]</td>
<td>2 0 0 0 1 0 1 0 2 0 0 1 0 0 0</td>
</tr>
<tr>
<td>AC[i]</td>
<td>3 0 0 5 1 0 1 0 2 0 0 2 0 1 0</td>
</tr>
<tr>
<td>ALC[i]</td>
<td>2 0 0 0 1 0 1 0 2 0 0 1 0 0 0</td>
</tr>
<tr>
<td>STATUS</td>
<td>A A A A A A A A A A A A A A A A A A</td>
</tr>
</tbody>
</table>

Table 4.2(a) Initial Array of tree $T$ in Figure 4.1.

**Iteration - 1**

After executing the first iteration of the algorithm, the values of arrays are changed and the Tree is shown in Figure 4.2 and given in Table 4.2(b).
<table>
<thead>
<tr>
<th>i</th>
<th>1 2 3 4 5 6 7 8 9 10 11 12 13 14</th>
</tr>
</thead>
<tbody>
<tr>
<td>p[i]</td>
<td>1 1 2 3 29 5 5 5 42 9 9 9 2 2</td>
</tr>
<tr>
<td>C[i]</td>
<td>5 3 3 0 3 0 0 0 3 0 0 0 1 2</td>
</tr>
<tr>
<td>LC[i]</td>
<td>0 0 3 0 3 0 0 0 3 0 0 0 1 2</td>
</tr>
<tr>
<td>AC[i]</td>
<td>3 1 0 0 0 0 0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>ALC[i]</td>
<td>1 1 0 0 0 0 0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>STATUS</td>
<td>A A R R R R R R R R R R R A</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>i</th>
<th>15 16 17 18 19 20 21 22 23 24 25 26 27 28</th>
</tr>
</thead>
<tbody>
<tr>
<td>p[i]</td>
<td>1 1 1 1 16 16 16 17 17 18 3 3 13 14</td>
</tr>
<tr>
<td>C[i]</td>
<td>1 3 2 1 0 0 0 0 0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>LC[i]</td>
<td>0 3 2 1 0 0 0 0 0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>AC[i]</td>
<td>1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>ALC[i]</td>
<td>0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>STATUS</td>
<td>A R A R R R R R R R R R R R R</td>
</tr>
<tr>
<td></td>
<td>29 30 31 32 33 34 35 36 37 38 39 40 41 42 43</td>
</tr>
<tr>
<td>---</td>
<td>---------------------------------------------</td>
</tr>
<tr>
<td>p[i]</td>
<td>32 29 29 15 32 33 32 35 32 37 37 32 40 40 14</td>
</tr>
<tr>
<td>C[i]</td>
<td>3 0 0 5 1 0 1 0 2 0 0 2 0 1 0</td>
</tr>
<tr>
<td>LC[i]</td>
<td>2 0 0 0 1 0 1 0 2 0 0 1 0 0 0</td>
</tr>
<tr>
<td>AC[i]</td>
<td>2 0 0 3 0 0 0 0 0 0 0 2 0 0 0</td>
</tr>
<tr>
<td>ALC[i]</td>
<td>2 0 0 1 0 0 0 0 0 0 0 2 0 0 0</td>
</tr>
<tr>
<td>STATUS</td>
<td>A A A A R R R R A R R A A A R</td>
</tr>
</tbody>
</table>

**Table 4.2(b). The arrays after one iteration**

Repeating the same process, all the entries of STATUS become remove state $R$. This is got as a result of trim operation in $T$. The result indicates whether the tree is IVDP or not.