Chapter 2

BACKGROUND

In this chapter we review some fundamental definitions and notations on graph theory that are relevant to the topic of this thesis. Given an undirected graph \( G = (V, E) \), a matching is a set of edges no two of which have a vertex in common. Each edge in the matching connects two vertices \( u, v \) and we say that \( u, v \) are matched. One is usually interested in finding maximum matching that is, matching having a maximum number of edges. Sometimes the edges have associated weights and one is interested in finding maximum weight matchings. Problems involving matching occur in many situations. Workers may be matched to jobs, machines to parts, players to teams etc.

A perfect matching is a matching in which atmost one vertex is left unmatched. Not all graphs have a perfect matching. Sun Wu and Udi Manber [WM 92] have generalized the concept of matchings and introduced the concept of path matching. They have given a
sequential algorithm to find a perfect Min-max EDP-matching of a tree.

Xavier proved that Sun Wu and Udi Manber’s algorithm is incorrect. He showed that their algorithm may not always output the perfect Min-max EDP-matching when the tree contains odd number of vertices. He developed [XA96] two correct parallel algorithms for the same problem on trees. The first algorithm processes odd and even trees separately. It works in a bottom up manner and finds the cost of the perfect Min-max EDP-matching. The second algorithm converts an odd tree into an even tree by removing a suitable vertex and then solves the problem on the even tree.

2.1 Trees

The concept of a tree is probably the most important in graph theory. It finds its applications to simple situations such as puzzles and games deferring the application to more complete and scientific problems.

A tree is a connected graph without any cycles. The following are equivalent.
i. $G$ is a tree

ii. There is exactly one path between any two vertices of $G$.

iii. $G$ is connected and it contains $n$ vertices and $n-1$ edges.

iv. $G$ is minimally connected

v. $G$ has no cycles and $G$ has $n$ vertices and $n-1$ edges.

The graph in Figure 2.1 is a tree.

It follows immediately from the definition that a tree has to be a *simple graph* that is, having neither a self loop nor parallel edges.

![Figure 2.1 A tree](image-url)
Trees appear in numerous instances. The genealogy of a family is often represented by means of a tree. A river with its tributaries and subtributaries can be represented by a tree. The sorting of mail according to zip code and the sorting of punched cards are done according to a tree.

Consider the tree given in Figure 2.2(a). If we designate 2 as the root of the tree, the tree can be redrawn as in Figure 2.2(b). For a rooted tree, the level numbers can be defined to each of the vertices as follows:
1. The root is assigned level number 0.

2. The vertices adjacent to the root are called the children of the root and they are assigned the level number 1. The root is called the parent of its children.

   In our tree 1 and 3 are the children of 2.

3. If a vertex \( v \) is at level \( i \), any other adjacent vertex \( u \) which is not the parent of \( v \) is assigned level \( i+1 \). Such a node will be called a child of \( v \). The height or depth of a tree is defined to be the maximum level of any node in the tree.

A connected subgraph of a tree is called a subtree. A rooted tree is usually represented by its parent relation. If \( T = (V, E) \) is a rooted tree with root \( r \), it is represented by the array Parent(1:n) where \( n \) is the number of vertices.

Parent relation is defined as follows:

\[ \text{PARENT} (i) = \text{the parent of } i, \text{ if } i \text{ is not the root} \]

\[ \text{PARENT} (r) = r \text{ where } r \text{ is the root} \]
Figure 2.2(b)  A rooted tree T

For the tree in figure 2.2(b) the parent array is shown below

<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>
</tr>
</thead>
<tbody>
<tr>
<td>Parent</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>5</td>
<td>5</td>
<td>7</td>
<td>6</td>
<td>5</td>
</tr>
</tbody>
</table>

Table 2.1 Parent array of the tree
2.2 Matching Problems

Given a graph $G = (V,E)$ a **matching** is a set of edges $M$ such that no two edges in $M$ incident on the same vertex. If an edge $(u,v) \in M$, we say that $u$ and $v$ are matched in $M$. A collection of paths $P$ is called a **path matching** if no vertex of $G$ is an end vertex of more than one path in $P$. If any two paths of $P$ are edge disjoint, $P$ is called **edge disjoint path matching (EDP − Matching)**. If vertices $u$ and $v$ are end vertices of a path in $P$, we say that $u$ is matched to $v$. When $G$ is an edge weighted graph the sum of weight of edges of a path in $G$ is called the **cost of the path**. If $P$ is a path matching, the maximum cost of a path in $P$ is called the **cost of $P$**. A perfect EDP matching with minimum cost is called a **perfect Min-max EDP matching**. A **maximal matching** is a matching which is not properly contained in another matching. A matching with maximum number of edges is called a **maximum matching**. A **perfect matching** of $G = (V, E)$ is a matching $M$ which matches all the vertices of $G$ except possibly one. That is, $M$ is a perfect matching if and only if $|M| = \lfloor n/2 \rfloor$. A perfect matching does not exist for some graphs. When a perfect matching exists, one may be interested in finding the
perfect matching with minimum cost (or maximum cost). The matching problem has applications in several situations. Workers may be matched to jobs, machines to parts, players to teams etc.

Sun Wu and Udi Manber [WM 92] have introduced a generalization of the matching problem. Let $G = (V, E)$ be a weighted graph. A path matching in $G$ is a set of simple paths with distinct end vertices. Thus a matching is a special case of a path-matching in which each path is of length one.

A path matching $P$ of $G$ is said to be perfect if $|P| = \lfloor n/2 \rfloor$. If $G$ contains odd number of vertices, a perfect path matching leaves exactly one element unmatched. If $G$ has even number of vertices, all the elements of $G$ are matched in a perfect path matching. A pathmatching $P$ of $G$ in which any two matching paths are edge disjoint is called an edge disjoint path matching (EDP matching).

Sun Wu and Udi Manber have proved that

**Lemma 2.11 [WM 92]** Every undirected graph has at least one edge disjoint perfect path matching.
As an application of path matching let us assume that $G$ models a network of computers such that each vertex corresponds to a computer and each edge corresponds to a link of communication. Each link is associated with a cost (eg. load, tariff, delay). Suppose further that we want to organize a tournament among the computers such that each computer is paired with another one and they perform some competition together. The competition may correspond to some computation task that both computers are involved in. A path represents the pairing of the computers corresponding to the end vertices. We now list some variations of the path-matching problem.

### 2.2.1 Min-sum Path Matching

Let $G=(V,E)$ be a weighted graph and $P$ be a perfect path matching of $G$. The cost of a path is the sum of the weights of all the edges in the path. The sum of all the costs of the paths in $P$ is called the cost-sum of $P$. The problem of finding the perfect path matching with minimum cost-sum is called the **Min-Sum Path Matching Problem**. [WM 88] gives an $O(n)$ sequential algorithm to find the min-sum path matching for trees.
For general graphs, there is a min-sum path matching that contains only paths with at most two edges each. This can be obtained by computing all shortest paths (with at most two edges) and finding a minimum matching.

2.2.2 Min-max Path Matching

The max-cost of a perfect path matching $M$ is the maximum cost of the paths in $M$. The perfect path matching which has minimum max-cost is called the min-max perfect path matching. This problem is also called bottle-neck problem. We call the edge disjoint perfect path matching, EDP-matching. Sun Wu and Udi Manber [WM 92] have given an $O(n \log d \log w)$ time sequential algorithm to find the min-max EDP-matching for integer weighted trees, where $d$ is the maximum degree of a vertex and $w$ is the maximum cost of an edge. They have given another $O(n^2)$ time algorithm for finding min-max DP-matching for trees with arbitrary costs. Xavier [XA 96] proved that both the algorithm of Sun Wu and Udi Manber [WM 92] are incorrect. He gave a counter-example to show that their algorithm does not find the min-max EDP-matching always. He also
developed correct parallel algorithms for finding the min-max EDP-matchings.

2.3 Path Matching on Trees

Sun Wu and Udi Manber[WM 92] have given algorithms to find the min-max EDP-matching of trees. Xavier showed by a counter example that the algorithm of Sun Wu and Udi Manber does not work for trees with odd number of vertices. He gave a correct parallel algorithm for this problem which works in $O(h \log s)$ time using $O(h n' \log s)$ processors in CREW PRAM where $h$ is the height of the tree, $n'$ the maximum number of nodes at any level and $s$ the maximum number of children for any node.

2.4 Euler Circuits

Let $T = (V, E)$ be a graph. Let $T'$ be the directed graph obtained by replacing each edge $(u, v)$ of $T$ by two directed arcs $<u, v>$ and $<v, u>$. The directed graph $T'$ is Eulerian. The Euler circuit of $T'$ is said to be the Euler circuit of $T$. Assume that a tree $T$ is represented
in terms of its adjacency list. For example the tree in Figure 2.3 is shown in the Table 2.2

We represent the Euler circuit of the tree as a list of directed arcs.

The Euler circuit is \( \{<1, 2>, <2, 3>, <3, 2>, <2, 4>, <4, 5>, <5, 6>, <6, 5>, <5, 7>, <7, 5>, <5, 8>, <8, 9>, <9, 8>, <8, 10>, <10, 8>, <8, 12>, <12, 11>, <11, 12>, <12, 8>, <8, 5>, <5, 4>, <4, 2>, <2, 1>\}.

![Figure 2.3. A tree T](image)
<table>
<thead>
<tr>
<th>Vertex (V)</th>
<th>Adj (v)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>1, 3, 4</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>2, 5</td>
</tr>
<tr>
<td>5</td>
<td>4, 6, 7, 8</td>
</tr>
<tr>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>7</td>
<td>5</td>
</tr>
<tr>
<td>8</td>
<td>5, 9, 10, 12</td>
</tr>
<tr>
<td>9</td>
<td>8</td>
</tr>
<tr>
<td>10</td>
<td>8</td>
</tr>
<tr>
<td>11</td>
<td>12</td>
</tr>
<tr>
<td>12</td>
<td>8, 11</td>
</tr>
</tbody>
</table>

**Table 2.2. Adjacency list of the above tree in Figure 2.3**

The list is represented in the form of its successor function. That is successor is a bijection from the set of directed arcs of $T'$ to itself. The successor function completely specifies the Euler circuit.

Consider the vertex 5. The vertices adjacent to 5 are 4, 6, 7 and 8.

If we see the Euler’s circuit, we find that
Successor (<4, 5>) = <5, 6>
Successor (<6, 5>) = <5, 7>
Successor (<7, 5>) = <5, 8>
Successor (<8, 5>) = <5, 4>

In general, let $v$ be a vertex and $u_0, u_1, u_2 \ldots u_{d-1}$ be the vertices adjacent to $v$ where the degree of $v$ is $d$. Then we have

Successor (<$u_0, v$>) = <$v, u_1$
Successor (<$u_1, v$>) = <$v, u_2$
Successor (<$u_2, v$>) = <$v, u_3$

Successor (<$u_{d-1}, v$>) = <$v, u_0$
Successor (<$u_i, v$>) = <$v, $u_{i+1(mod \ d)}$

Consider the tree represented by its adjacency list with some additional pointers. For any arc <$u_i, v$> we can find <$v, u_{i+1(mod \ d)}$> in $O(1)$ time. Hence we have the following parallel algorithm to find the Euler Circuit of a tree which works in $O(1)$ time using $O(n)$ processors in EREW PRAM.
Algorithm Euler Circuit [XI 98]

**Input** : A tree $T$ represented by its adjacency list with some additional pointers.

**Output** : Successors $(u, v)$ for every arc $(u, v)$

1. For every arc $(u, v)$ do step 2 in parallel

2. Successor $(u, v) = (v, w)$

Where $w$ occurs next to $u$ in the ordered list of vertices adjacent to $v$.

If $u$ appears last in the list of vertices adjacent to $v$ then $w$ is the first node in the list.

Having developed an algorithm to find the Euler circuit of a tree in $O(1)$ time, we can solve several interesting problems on trees using Euler circuits.

### 2.4.1 Post order Numbering

The post order traversal method is an order to visit the nodes of the tree. The post order traversal of a tree $T$ with root $r$ consists of the post order traversal of the subtrees of $r$ from left to right followed by the root $r$. For example consider the tree shown in the Figure 2.4.
Here 4 is the root. The children of 4 are 2, 3 and 5. The post order traversal of this tree is 1, 2, 3, 6, 7, 11, 10, 8, 9, 5, 4.

The post order numbering is a function which gives the rank of the vertex in the post order traversal sequence. For example, the post order numbering is given by
Post (1) = 1
Post (2) = 2
Post (3) = 3
Post (6) = 4
Post (7) = 5
Post (11) = 6
Post (10) = 7
Post (8) = 8
Post (9) = 9
Post (5) = 10
Post (4) = 11

In the Euler circuit whenever we travel along the arc \( <v, p(v)> \), we have just traversed the vertex \( v \). This is achieved by the following steps.

1. For every arc \( <u, v> \) if \( u \) is the parent of \( v \), then assign weight 0 to \( <u, v> \) and if \( v \) is the parent of \( u \) assign weight 1 to \( <u, v> \).

2. Perform the prefix sum of weights of the arcs as per the list specified by the successor function of the Euler circuit.
3. For every vertex \( v \), \( post(v) \) is the prefix sum of the arc 
\(<v, p(v)>\).

4. Post order numbering of the root is \( n \), where \( n \) is the number of vertices in the tree.

This is given in the Table 2.3.

<table>
<thead>
<tr>
<th>( v )</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>
</tr>
</thead>
<tbody>
<tr>
<td>Post(( v ))</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>11</td>
<td>10</td>
<td>4</td>
<td>5</td>
<td>8</td>
<td>9</td>
<td>7</td>
<td>6</td>
</tr>
</tbody>
</table>

**Table 2.3. Post order Numbering**

Euler’s path, weights, prefix sum are shown in the Table 2.4. We present the algorithm to find the post order numbering as given in [XI 98]
<table>
<thead>
<tr>
<th>Euler’s Path</th>
<th>Weight</th>
<th>Prefix sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;4, 2&gt;</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>&lt;2, 1&gt;</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>&lt;1, 2&gt;</td>
<td>1</td>
<td>1 post (1) = 1</td>
</tr>
<tr>
<td>&lt;2, 4&gt;</td>
<td>1</td>
<td>2 post (2) = 2</td>
</tr>
<tr>
<td>&lt;4, 3&gt;</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>&lt;3, 4&gt;</td>
<td>1</td>
<td>3 post (3) = 3</td>
</tr>
<tr>
<td>&lt;4, 5&gt;</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>&lt;5, 6&gt;</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>&lt;6, 5&gt;</td>
<td>1</td>
<td>4 post (6) = 4</td>
</tr>
<tr>
<td>&lt;5, 7&gt;</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>&lt;7, 5&gt;</td>
<td>1</td>
<td>5 post (7) = 5</td>
</tr>
<tr>
<td>&lt;5, 8&gt;</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>&lt;8, 10&gt;</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>&lt;10, 11&gt;</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>&lt;11, 10&gt;</td>
<td>1</td>
<td>6 post (11) = 6</td>
</tr>
<tr>
<td>&lt;10, 8&gt;</td>
<td>1</td>
<td>7 post (10) = 7</td>
</tr>
<tr>
<td>&lt;8, 5&gt;</td>
<td>1</td>
<td>8 post (8) = 8</td>
</tr>
<tr>
<td>&lt;5, 9&gt;</td>
<td>0</td>
<td>8</td>
</tr>
<tr>
<td>&lt;9, 5&gt;</td>
<td>1</td>
<td>9 post (9) = 9</td>
</tr>
<tr>
<td>&lt;5, 4&gt;</td>
<td>1</td>
<td>10 post (5) = 10</td>
</tr>
</tbody>
</table>

Table 2.4. Post order numbering as prefix sum
Algorithm Postorder

Input

1. The rooted tree \( T = (V, E) \) with root \( r \) given by the parent relation \( p(v) \)
2. The Euler circuit of \( T \).

Output: For every vertex \( v \), the post order numbering \( post(v) \)

1. For every arc \( <u, v> \) do in parallel
   
   If \( <v, p(v)> \) assign weight 1 else
   
   Assign the weight 0 to the arc \( <u, v> \).

2. Find the prefix sum of the list of the weights specified by the successor function.

3. For every vertex \( v \) do in parallel
   
   \( post(v) = \) prefix sum of the arc \( <v, p(v)> \)
   
   end parallel

4. \( post(r) = n \)

5. End Post order numbering
Complexity Analysis

Step 1 of the algorithm can be done in constant time. Step 2 can be done in $O(\log n)$ time using $O(n)$ processors. Step 3 and 4 needs only constant time using $O(n)$ processors. So the algorithm can be implemented in $O(\log n)$ time with $O(n)$ processors in CREW PRAM.

2.4.2 Number of Descendants

The number of descendants for each vertex can be obtained from the prefix sum of the weights of arcs determined in the post order numbering algorithm. The number of descendants of a vertex $v$ is the number of vertices in the maximal subtree with $v$ as the root. It is the difference between prefix sum of $<p(v), v>$ and the prefix sum of $<v, p(v)>$. For example from the table of prefix sum of $<4, 5>$ and $<5, 4>$ are 3 and 10 respectively. So the number of descendants of vertex 5 are $10 - 3 = 7$. That is the maximal subtree with 5 as root has 7 vertices. Similarly the prefix sum of the arcs $<5, 8>$ and $<8, 5>$ are 5 and 8 respectively. So the number of descendants for 8 is 3. Similarly the number of descendants for each node can be determined. The algorithm to find the number of descendants is given below.
Algorithm Numberofdescendants

**Input**

1. The rooted tree $T = (V, E)$ with root $r$
given by the parent relation $p(v)$
2. The Euler circuit of $T$.

**Output** : For every vertex $v$, the number of descendants

1. For every arc $<u, v>$ do in parallel
   
   if $(v, p(v))$ assign weightage = 1 else
   
   assign weight = 0

2. Find the prefix sum of the list of weights specified by the successor function.

3. For every vertex $v$ do in parallel
   
   $post(v) =$ prefix sum of arc $<v, p(v)>$
   
   end parallel

4. For every vertex $v$ do in parallel
   
   $Number \text{ of descendants } (v) = (post <v, p(v)> ) - (post <p(v), v>)$
   
   End parallel

**Theorem 2.1**

Algorithm Numberofdescendants correctly finds the number of descendants of every node $v$ of $T$. 

28
Proof

The post order numbering $post(v)$ is correctly found in step 3 of the algorithm. We have assigned the weights 0 and 1 to the arcs in such a way that arcs of the type $<v, p(v)>$ get 1 weight and the arcs of the type $<p(v), v>$ get 0 weight. When the prefix sum is evaluated, a value $l$ will get added only when we move upwards. So, the difference between the prefix sum of $<v, p(v)>$ and $<p(v), v>$ gives the number of times we move upwards in the Euler’s circuit between $<p(v), v>$ to $<v, p(v)>$ which is the number of descendants of $v$ (including $v$). Hence the proof.

Figure 2.5. A tree $T$
Theorem 2.2

The above algorithm counts the number of descendants of each node \( v \) of a tree \( T \) in \( O(\log n) \) time using \( O(n) \) processors.

Proof

Step 1 of the above algorithm is implemented in \( O(1) \) time using \( O(n) \) processors. Step 2 of the algorithm is implemented in \( O(\log n) \) time using \( O(n) \) processors. Step 3 can be implemented using \( O(1) \) time using \( O(n) \) processors.

To determine the number of descendants step 4 is implemented in \( O(\log n) \) time using \( O(n) \) processors.

Hence the above algorithm is implemented in \( O(\log n) \) time using \( O(n) \) processors in EREW PRAM.