GRAPH
ALGORITHMS

SHIMON EVEN
Technion Institute

Computer Science Press
Chapter 4

4. ORDERED TREES

4.1 UNIQUELY DECIPHERABLE CODES

Let $\Sigma = \{0, 1, \ldots, \sigma - 1\}$. We call $\Sigma$ an alphabet and its elements are called letters; the number of letters in $\Sigma$ is $\sigma$. (Except for this numerical use of $\sigma$, the “numerical” value of the letters is ignored; they are just “meaningless” characters. We use the numerals just because they are convenient characters.) A finite sequence $a_1a_2 \cdots a_l$, where $a_i$ is a letter, is called a word whose length is $l$. We denote the length of a word $w$ by $l(w)$. A set of (non-empty and distinct) words is called a code. For example, the code $\{102, 21, 00\}$ consists of three code-words: one code-word of length 3 and two code-words of length 2; the alphabet is $\{0, 1, 2\}$ and consists of three letters. Such an alphabet is called ternary.

Let $c_1, c_2, \ldots, c_k$ be code-words. The message $c_1c_2 \cdots c_k$ is the word resulting from the concatenation of the code-word $c_1$ with $c_2$, etc. For example, if $c_1 = 00$, $c_2 = 21$ and $c_3 = 00$, then $c_1c_2c_3 = 002100$.

A code $C$ over $\Sigma$ (that is, the code-words of $C$ consist of letters in $\Sigma$) is said to be uniquely decipherable (UD) if every message constructed from code-words of $C$ can be broken down into code-words of $C$ in only one way. For example, the code $\{01, 0, 10\}$ is not UD because the message 010 can be parsed in two ways: 0, 10 and 01, 0.

Our first goal is to describe a test for deciding whether a given code $C$ is UD. This test is an improvement of a test of Sardinas and Patterson [1] and can be found in Gallager's book [2].

If $s$, $p$ and $w$ are words and $ps = w$ then $p$ is called a prefix of $w$ and $s$ is called a suffix of $w$. We say that a word $w$ is non-empty if $l(w) > 0$.

A non-empty word $t$ is called a tail if there exist two messages $c_1c_2 \cdots c_m$ and $c_1'c_2' \cdots c_n'$ with the following properties:

1. $c_i, 1 \leq i \leq m$, and $c_j', 1 \leq j \leq n$ are code-words and $c_1 \neq c_1'$;
2. $t$ is a suffix of $c_n'$;
3. $c_1c_2 \cdots c_mt = c_1'c_2' \cdots c_n'$.
Lemma 4.1: A code \( C \) is UD if and only if no tail is a code-word.

**Proof:** If a code-word \( c \) is a tail then by definition there exist two messages \( c_1c_2 \ldots c_m \) and \( c_1'c_2' \ldots c_n' \) which satisfy \( c_1c_2 \ldots c_mc = c_1'c_2' \ldots c_n' \), while \( c_1 \neq c_1' \). Thus, there are two different ways to parse this message, and \( C \) is not UD.

If \( C \) is not UD then there exist messages which can be parsed in more than one way. Let \( \mu \) be such an ambiguous message whose length is minimum: \( \mu = c_1c_2 \ldots c_k = c_1'c_2' \ldots c_n' \); i.e. all the \( c_i \)-s and \( c_j \)-s are code-words and \( c_1 \neq c_1' \). Now, without loss of generality we can assume that \( c_k \) is a suffix of \( c_n' \) (or change sides). Thus, \( c_k \) is a tail.

Q.E.D.

The algorithm generates all the tails. If a code-word is a tail, the algorithm terminates with a negative answer.

**Algorithm for UD:**

1. For every two code-words, \( c_i \) and \( c_j \) (\( i \neq j \)), do the following:
   1.1 If \( c_i = c_j \), halt; \( C \) is not UD.
   1.2 If for some word \( s \), either \( c_is = c_j \) or \( c_j = c_is \), put \( s \) in the set of tails.

2. For every tail \( t \) and every code-word \( c \) do the following:
   2.1 If \( t = c \), halt; \( C \) is not UD.
   2.2 If for some word \( s \) either \( ts = c \) or \( cs = t \), put \( s \) in the set of tails.

3. Halt; \( C \) is UD.

Clearly, in Step (1), the words declared to be tails are indeed tails. In Step (2), since \( t \) is already known to be a tail, there exist code-words \( c_1, c_2, \ldots, c_m \) and \( c_1', c_2', \ldots, c_n' \) such that \( c_1c_2 \ldots c_mt = c_1'c_2' \ldots c_n' \). Now, if \( ts = c \) then \( c_1c_2 \ldots c_MC = c_1'c_2' \ldots c_n's \), and therefore \( s \) is a tail; and if \( cs = t \) then \( c_1c_2 \ldots c_Mcs = c_1'c_2' \ldots c_n' \) and \( s \) is a tail.

Next, if the algorithm halts in (3), we want to show that all the tails have been produced. Once this is established, it is easy to see that the conclusion that \( C \) is UD follows; Each tail has been checked, in Step (2.1), whether it is equal to a code-word, and no such equality has been found; by Lemma 4.1, the code \( C \) is UD.
For every \( t \) let \( m(t) = c_1c_2 \cdots c_m \) be a shortest message such that \( c_1c_2 \cdots c_m = c_1'c_2' \cdots c_n' \), and \( t \) is a suffix of \( c_n' \). We prove by induction on the length of \( m(t) \) that \( t \) is produced. If \( m(t) = 1 \) then \( t \) is produced by (1.2), since \( m = n = 1 \).

Now assume that all tails \( p \) for which \( m(p) < m(t) \) have been produced. Since \( t \) is a suffix of \( c_n' \), we have \( pt = c_n' \). Therefore, \( c_1c_2 \cdots c_m = c_1'c_2' \cdots c_{n-1}'p \).

If \( p = c_m \) then \( c_m = c_n \) and \( t \) is produced in Step (1).

If \( p \) is a suffix of \( c_m \) then, by definition, \( p \) is a tail. Also, \( m(p) \) is shorter than \( m(t) \). By the inductive hypothesis \( p \) has been produced. In Step (2.2), when applied to the tail \( p \) and code-word \( c_n' \), by \( pt = c_n' \), the tail \( t \) is produced.

If \( c_m \) is a suffix of \( p \), then \( c_m \) is a suffix of \( c_n' \), and therefore, \( c_m \) is a tail. \( m(c_m) = c_1c_2 \cdots c_{m-1} \), and is shorter than \( m(t) \). By the inductive hypothesis \( c_m \) has been produced. In Step (2.2), when applied to the tail \( c_m \) and code-word \( c_m \), the tail \( t \) is produced.

This proves that the algorithm halts with the right answer.

Let the code consists of \( n \) words and \( l \) be the maximum length of a code-word. Step (1) takes at most \( O(n^2 \cdot l) \) elementary operations. The number of tails is at most \( O(n \cdot l) \). Thus, Step (2) takes at most \( O(n^2l^2) \) elementary operations. Therefore, the whole algorithm is of time complexity \( O(n^2l^2) \). Other algorithms of the same complexity can be found in References 3 and 4; these tests are extendible to test for additional properties [5, 6, 7].

**Theorem 4.1:** Let \( C = \{c_1, c_2, \ldots, c_n\} \) be a UD code over an alphabet of \( \sigma \) letters. If \( l_i = l(c_i), i = 1, 2, \ldots, n \), then

\[
\sum_{i=1}^{n} \sigma^{-l_i} \leq 1. \tag{4.1}
\]

The left hand side of (4.1) is called the *characteristic sum* of \( C \); clearly, it characterizes the vector \((l_1, l_2, \ldots, l_n)\), rather than \( C \). The inequality (4.1) is called the *characteristic sum condition*. The theorem was first proved by McMillan [8]. The following proof is due to Karush [9].

**Proof:** Let \( e \) be a positive integer

\[
\left( \sum_{i=1}^{n} \sigma^{-l_i} \right)^e = \sum_{i_{e-1}} \sum_{i_{e-1}} \cdots \sum_{i_{e-1}} \sigma^{-l_i_{e-1}+l_i_{e-1}^2+\ldots+l_i_{e-1}^e}. 
\]
There is a unique term, on the right hand side, for each of the \( n^e \) messages of \( e \) code-words. Let us denote by \( N(e, j) \) the number of messages of \( e \) code-words whose length is \( j \). It follows that

\[
\sum_{i_1}^{n} \sum_{i_2}^{n} \cdots \sum_{i_e}^{n} \sigma^{-l_{i_1}+l_{i_2}+\cdots+l_{i_e}} = \sum_{j=e}^{e} N(e, j) \cdot \sigma^{-j}
\]

where \( \hat{l} \) is the maximum length of a code-word. Since \( C \) is UD, no two messages can be equal. Thus, \( N(e, j) \leq \sigma^j \). We now have,

\[
\sum_{j=e}^{e} N(e, j) \cdot \sigma^{-j} \leq \sum_{j=e}^{e} \sigma^j \cdot \sigma^{-j} \leq e \cdot \hat{l}.
\]

We conclude that for all \( e \geq 1 \)

\[
\left( \sum_{i=1}^{n} \sigma^{-l_i} \right)^e \leq e \cdot \hat{l}.
\]

This implies (4.1).

Q.E.D.

A code \( C \) is said to be prefix if no code-word is a prefix of another. For example, the code \{00, 10, 11, 100, 110\} is not prefix since 10 is a prefix of 100; the code \{00, 10, 11, 010, 011\} is prefix. A prefix code has no tails, and is therefore UD. In fact it is very easy to parse messages: As we read the message from left to right, as soon as we read a code-word we know that it is the first code-word of the message, since it cannot be the beginning of another code-word. Therefore, in most applications, prefix codes are used. The following theorem, due to Kraft [10], in a sense, shows us that we do not need non-prefix codes.

**Theorem 4.2:** If the vector of integers, \((l_1, l_2, \ldots, l_n)\), satisfies

\[
\sum_{i=1}^{n} \sigma^{-l_i} \leq 1 \quad (4.2)
\]

then there exists a prefix code \( C = \{c_1, c_2, \ldots, c_n\} \), over the alphabet of \( \sigma \) letters, such that \( l_i = l(c_i) \).
Proof: Let \( \lambda_1 < \lambda_2 < \cdots < \lambda_m \) be integers such that each \( l_i \) is equal to one of the \( \lambda_j \)'s and each \( \lambda_i \) is equal to at least one of the \( l_i \)'s. Let \( k_j \) be the number of \( l_i \)'s which are equal to \( \lambda_j \). We have to show that there exists a prefix code \( C \) such that the number of code-words of length \( \lambda_j \) is \( k_j \).

Clearly, (4.2) implies that

\[
\sum_{j=1}^{m} k_j \sigma^{-\lambda_j} \leq 1 \tag{4.3}
\]

We prove by induction on \( r \) that for every \( 1 \leq r \leq m \) there exists a prefix code \( C_r \), such that, for every \( 1 \leq j \leq r \), the number of its code-words of length \( \lambda_j \) is \( k_j \).

First assume that \( r = 1 \). Inequality (4.3) implies that \( k_1 \sigma^{-\lambda_1} \leq 1 \), or \( k_1 \leq \sigma^{\lambda_1} \). Since there are \( \sigma^{\lambda_1} \) distinct words of length \( \lambda_1 \), we can assign any \( k_1 \) of them to constitute \( C_1 \).

Now, assume \( C_r \) exists. If \( r < m \) then (4.3) implies that

\[
\sum_{j=1}^{r+1} k_j \sigma^{-\lambda_j} \leq 1.
\]

Multiplying both sides by \( \sigma^{\lambda_{r+1}} \) yields

\[
\sum_{j=1}^{r+1} k_j \sigma^{\lambda_{r+1}-\lambda_j} \leq \sigma^{\lambda_{r+1}},
\]

which is equivalent to

\[
k_{r+1} \leq \sigma^{\lambda_{r+1}} - \sum_{j=1}^{r} k_j \sigma^{\lambda_{r+1}-\lambda_j}. \tag{4.4}
\]

Out of the \( \sigma^{\lambda_{r+1}} \) distinct words of length \( \lambda_{r+1} \), \( k_j \cdot \sigma^{\lambda_{r+1}-\lambda_j} \), \( 1 \leq j \leq r \), have prefixed of length \( \lambda_j \) as code-words of \( C_r \). Thus, (4.4) implies that enough are left to assign \( k_{r+1} \) words of length \( \lambda_{r+1} \), so that none has a prefix in \( C_r \). The enlarged set of code-words is \( C_{r+1} \).

Q.E.D.

This proof suggests an algorithm for the construction of a code with a given vector of code-word length. We shall return to the question of prefix code construction, but first we want to introduce positional trees.