信州大学
大学院総合理工学研究科

修士論文

複数の符号表を用いた符号の一意復号可能性と最大復号遅延を調べるアルゴリズム

指導教員　西新 幹彦 准教授

学科　電子情報システム工学科
学籍番号　23W2003C
氏名　秋好 伸之輔

2025 年 2 月 6 日

目次

1 はじめに 1

2 一意復号可能符号と判定方法 2
2.1 一意復号可能符号 2
2.2 サーディナス-パターソン・アルゴリズム 3

3 復号遅延と最大復号遅延を調べる方法 5
3.1 復号遅延 5
3.2 最大復号遅延の有限値の求め方 7
3.3 最大復号遅延を調べるアルゴリズム 9

4 2 つの符号表を使った符号 12
4.1 符号の表現 12
4.2 一意復号可能性 13
4.3 復号遅延 14

5 2 つの符号表を使った符号における符号評価アルゴリズム 14
5.1 一意復号可能性を調べる方法 15
5.2 最大復号遅延を調べる方法 16
5.3 符号評価アルゴリズム 17
5.4 考察 17

6 まとめ 21

謝辞 21

参考文献 21

付録 A ソースコード 22
A.1 1 つの符号表を使った符号において一意復号可能性と最大復号遅延を調べるプログラム 22
A.2 2 つの符号表を使った符号において一意復号可能性と最大復号遅延を調べるプログラム 25

1　はじめに

世の中の通信システムにおいて，データは「符号」という形式に変換されて伝送される．実際，テレビの映像や携帯電話の音声などの莫大なデータは，全て符号に変換されて伝送される．そのため符号化する際には，データを誤りなく，いかに小さくまとめて伝送できるかが問題である．

復号誤りが生じない符号は「一意復号可能符号」と言い，データ系列に対する符号語系列が一意に定まる．しかし，最初の一語を復号するために，符号語系列の後の部分を調べないと復号できないことがある．何個先の部分を調べないと復号できないか，という概念を復号遅延と言い，一意復号可能符号は「復号遅延のない符号」と「復号遅延のある符号」の 2 つの符号に分類できる．

従来研究より与えられた符号に対する一意復号可能性はサーディナス-パターソン・アルゴリズム [1] を用いて判定することができる．このアルゴリズムは，復号遅延の有無に関わらず，一意復号可能符号を判別することができる．しかし，復号遅延のない符号の一意復号可能性は符号の性質上それぞれの符号語が，どの符号語の語頭にもなっていないという条件で判別することもできる．そして復号遅延の無い符号は，各符号語の最終ビットを調べた時点で瞬時に符号語の復号ができる点から瞬時符号と呼ばれる．瞬時符号を構成するための制約条件としてクラフトの不等式がある．これは，不等式を満たす符号語長の組を持つ符号しか瞬時符号になり得ないという条件である．そして，瞬時符号を構成する代表的な方法としてハフマン符号 [2] があり，瞬時符号の中で平均符号語長が最小になる符号化アルゴリズムとして知られている．一方，復号遅延のある符号を構成する方法として AIFV 符号 [3] がある．AIFV 符号は，2 つの符号表を用いており，最大復号遅延が 2 ビットになるが，ハフマン符号よりも小さい平均符号語長を達成できる符号である．一般的に復号遅延のある符号は，大きな復号遅延が生じる可能性があり実用的ではないが，AIFV 符号は復号遅延が 2 ビット以下になるように作られており，この特性から準瞬時符号と呼ばれる．そして，復号遅延のある符号の中には，最大復号遅延が無限になる符号が存在する．復号遅延が無限になるということは，その符号語系列に終わりが無い限り，復号することができない状況のことを言う．つまり，最大復号遅延が無限になる符号は，系列の最後の 1 ビットを調べるまで復号できないという性質を持っている．この復号のタイミングをコントロールできる特性を，セキュリティ分野で応用できるのではないかと考える．

本研究ではサーディナス-パターソン・アルゴリズムを拡張し，最大復号遅延が無限になるか有限になるか調べる方法を提案し，さらに複数の符号表を用いた符号の一意復号可能性と最大復号遅延を調べる方法を提案する．

2　一意復号可能符号と判定方法

前章で述べたように，データは符号に変換されて伝送される．しかし，データを符号に変換する際，どんな方法でも良いわけではない．この章では，従来の結果にある一意復号可能符号の定義と，その判定方法を述べる．

2.1　一意復号可能符号

まず，一意復号可能符号を定義する．

定義 1　D 個の要素からなる符号アルファベットを $\mathcal{D}$ で表し，$\mathcal{D}$ の要素を任意の個数並べた系列の全体を $\mathcal{D}^*$ と書くことにする．情報源アルファベットを $\mathcal{X}$ とするとき，情報源符号 C は，$\mathcal{X}$ から符号アルファベット上の系列の集合 $\mathcal{D}^*$ への写像で定義される．このとき，$C(x)$ はデータ $x \in \mathcal{X}$ に対する符号語を表す．

定義 2　符号 C において，すべての情報源記号 $x \in \mathcal{X}$ がそれぞれ異なる符号語 $C(x) \in \mathcal{D}^*$ に変換されるとき，すなわち任意の $x, x' \in \mathcal{X}$ に対して，

$$x \neq x' \Longrightarrow C(x) \neq C(x') \tag{1}$$

が成り立つとき，C は正則な符号であるという．

定義 3　符号 C の拡大符号とは，情報源アルファベット $\mathcal{X}$ の要素の有限個の系列から，任意の有限長の系列の集合 $\mathcal{D}^*$ への写像として定義され，

$$C^*(x_1 x_2 \cdots x_n) = C(x_1)C(x_2)\cdots C(x_n) \tag{2}$$

で与えられる．ここで，$C(x_1)C(x_2)\cdots C(x_n)$ は，それぞれの符号語を順に連接したものを表している．

定義 4　符号 C の拡大符号が正則であるとき，C を一意復号可能符号と呼ぶ．

以上のように一意復号可能符号は定義される．以下では，実際どのような符号が一意復号可能符号なのかを具体例を用いて述べる．

情報源アルファベットを $\mathcal{X} \stackrel{\text{def}}{=} \{a, b, c, d\}$，符号アルファベットを $\mathcal{D} \stackrel{\text{def}}{=} \{0, 1\}$ とし，符号 C が $C(a) = 0, C(b) = 001, C(c) = 101, C(d) = 11$ で与えられているとする．復号する文字列を符号語系列と呼び，001011 を復号すると，一意に bad と復号される．しかし，最初の符号語 0 が $C(a) = 0$ なのか，$C(b) = 001$ なのかは符号語系列の後の部分を調べないと判別できない．図 1 を見ると，00101 つまり aac までは復号することができるが，最後の 1 という

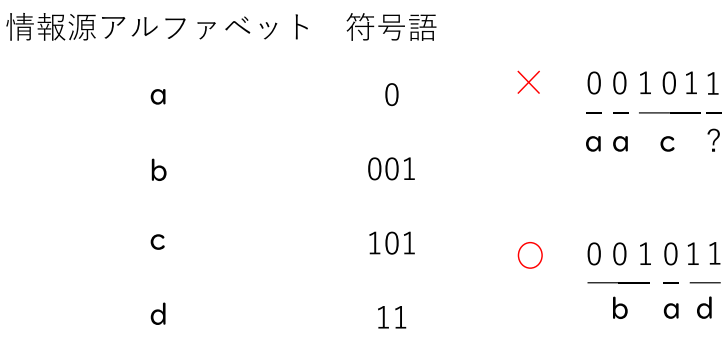

図 1　符号 C が $C(a)=0, C(b)=001, C(c)=101, C(d)=11$ で与えられており，符号語系列 001011 を復号する場合

文字列を復号することができないことがわかる．よって最初の符号語を $C(a)=0$ で復号することはできないため，最初の符号語が $C(b)=001$ と定まる．次に符号語 001 が削除された符号語系列 011 の復号を考えると，最初の符号語 0 が $C(a)=0$ なのか，$C(b)=001$ なのかは符号語系列の後の部分を調べないと判別できないが，01 まで調べた時点で $C(b)=001$ で復号することができないため，符号語系列 0 は $C(a)=0$ で復号される．そして残りの文字列 11 は $C(d)=11$ で復号される．このように符号語系列 001011 に対しては，一意に復号することができる．しかし，異なる符号語系列に対して一意に復号できなければ，この符号は復号誤りが生じる符号となり，符号化に使うことはできない．つまり，どんな符号語系列に対しても一意に復号できることを確認できれば，この符号は一意復号可能符号と分かる．しかし，全ての符号語系列に対して一意に復号できるかを確認するのは困難である．そこで，与えられた符号の一意復号可能性を調べるアルゴリズムとして，サーディナス-パターソン・アルゴリズムがある．

2.2　サーディナス-パターソン・アルゴリズム

まず，アルゴリズムを述べるために必要な概念である語頭と差分を定義する．

定義 5　系列 $u_1u_2\cdots u_n$ の語頭とは，その系列の任意の長さ $m(m\leq n)$ の先頭部分 $u_1u_2\cdots u_m$ のことをいう．

定義 6　2 つの系列 a, b において，a が b の語頭になっているとき，系列 b から語頭 a の部分を取り除いた系列を a と b の差分という．また，2 つの集合 A と B が与えられたとき，集合 A B の任意の要素 $a\in A$ と $b\in B$ において，a が b の語頭あるいは b が a の語頭になっているとき，求められた差分の集合を集合 A と B の差分という．

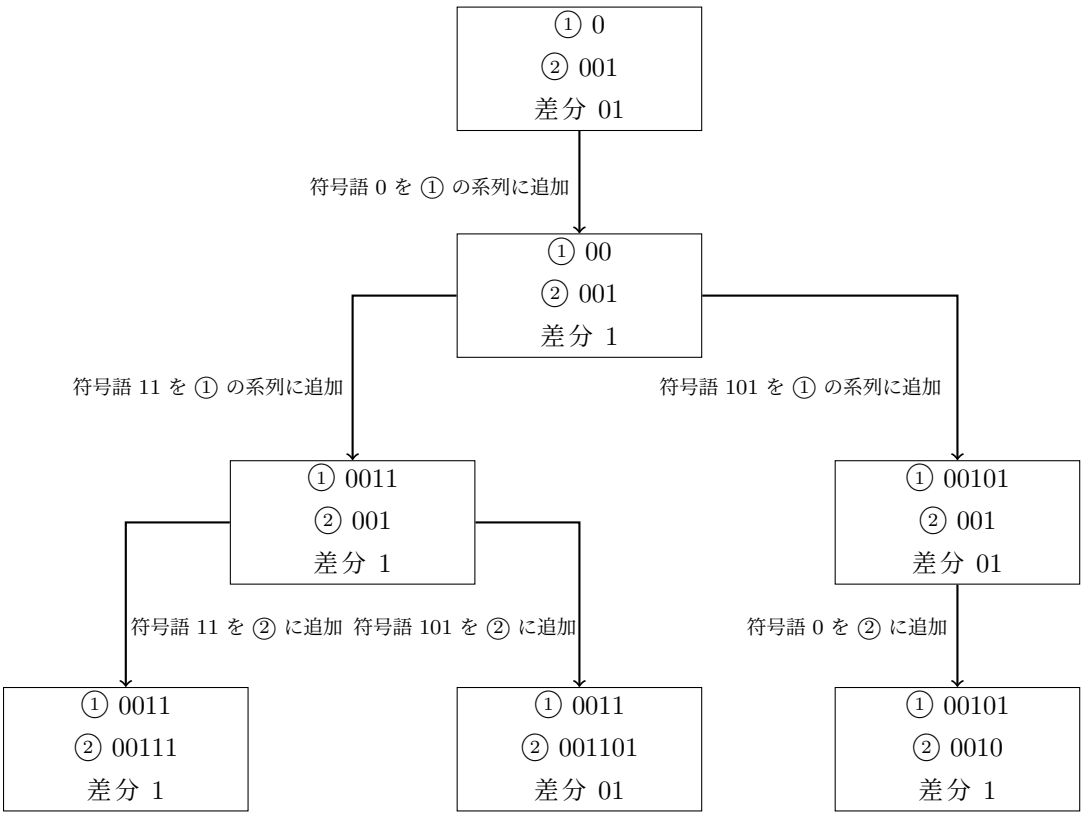

図 2　符号 $C = \{0, 001, 101, 11\}$ において先頭が符号語 0 と 001 から始まり，同じ文字列で構成される 2 つの符号語系列を作るときの遷移

前節の例で用いた符号について語頭と差分を考えると，符号語 0 は符号語 001 の語頭になっており，001 から 語頭の 0 を取り除いた系列 01 が差分である．サーディナス-パターソン・アルゴリズムでは，差分を使って一意復号可能性を調べる．

図 2 は，符号 $C = \{0, 001, 101, 11\}$ において，先頭が符号語 0 と 001 から始まり，同じ文字列で構成される 2 つの符号語系列を作るときの遷移，つまり 2 つの復号パターンがある符号語系列を作るときの遷移を表している．まず，符号語 0 と 001 で差分 01 が取れる．仮に，符号 C に 01 という符号語が含まれていた場合，001 という符号語系列に対して，2 通りの復号パターンがあるため，一意復号可能ではない．つまり，符号語同士の差分が符号語に含まれていれば，一意復号可能ではないと分かる．次に差分 01 と符号語で差分が取れるか調べると，符号語 0 との差分 1 が取れる．このとき ① の系列に符号語 0 を加え，符号語系列 00 を作る．もし，符号 C に符号語 1 が含まれていた場合，001 という符号語系列に対して，2 通りの復号パターンがあるため，一意復号可能ではない．つまり，出てきた差分と符号語の差分が符

号語に含まれていれば，一意復号可能ではないと分かる．ここで図 2 を見ると，この先の符号語系列を作るときの遷移で，差分 01 と 差分 1 しか出てこないことが確認できる．これまでに差分 01 と差分 1 が符号語に含まれていないことを確かめていたため，これ以上 2 つの復号パターンがある符号語系列を考える必要は無い．したがって，符号 $C = \{0, 001, 101, 11\}$ は一意復号可能符号であるとわかる．このように，符号語同士の差分，そして出てきた差分と符号語の差分が符号語に含まれているかで，一意復号可能性を調べることができる．これがサーディナス-パターソン・アルゴリズムであり，図 3 のようになる．

では実際に，アルゴリズムを使って一意復号可能かどうか確かめる．符号 $C = \{0, 001, 101, 11\}$ に図 3 のアルゴリズムを適応する．まず，$\delta_0 = \{01\}$ が取れ，$S_1 = \{01\}$ となる．ここで，C と S_1 の差分を取ると， $\delta_1 = \{1\}$ となるので，$S_2 = \{01, 1\}$ となる．さらに，C と S_2 の差分を取ると，$\delta_2 = \{1, 01\}$ となるので，$S_3 = \{1, 01\}$ となる．ここで，$S_3 = S_2$ となり，反復処理は終了する．得られた集合 S_2 に C の要素は 1 つも含まれていない．よって，C は一意復号可能符号である．

3　復号遅延と最大復号遅延を調べる方法

一意復号可能符号には，各符号語の最終ビットを調べた時点で復号できず，復号する際に遅延が生じてしまう符号が存在する．本章では復号遅延を定義し，最大何ビット先を調べないと復号できないか，という概念である最大復号遅延を調べる方法について述べる．

3.1　復号遅延

まず，復号遅延について，2.1 節の例を用いて述べる．図 1 を見ると，符号語系列 001011 において，最初の符号語を $C(b) = 001$ と復号するまでに符号語系列 001011 の最後の系列 1 まで調べた．つまり 001 より 3 つ後の部分を調べて復号することができたので，このとき b の復号遅延が 3 であると定義する．さらに，その後の $C(a) = 0$ を復号するために 0 よりも 2 つ後の部分を調べて復号できたので，このとき a の復号遅延が 2 となる．そして，その後の d の復号遅延は 0 となる．このように復号遅延はその符号語を復号するために，符号語系列のいくつ後の部分を調べたかによって決まる．そして，この符号語系列では一番大きい復号遅延が 3 ビットになっている．ここで，全ての符号語系列の復号を考えた場合に最も大きくなる復号遅延を最大復号遅延と定義する．

同じ符号を用いて，符号語系列 0010101 を復号しようとすると，一意に bac と復号される．しかし前例と同様に，最初の符号語 0 が $C(a) = 0$ なのか，$C(b) = 001$ なのかは符号語系列の後の部分を調べないと判別できない．図 4 を見ると，001010，つまり $aaca$ までは復号できるが，最後の 1 を復号することができないため，最初の符号語を $C(a) = 0$ で復号することは

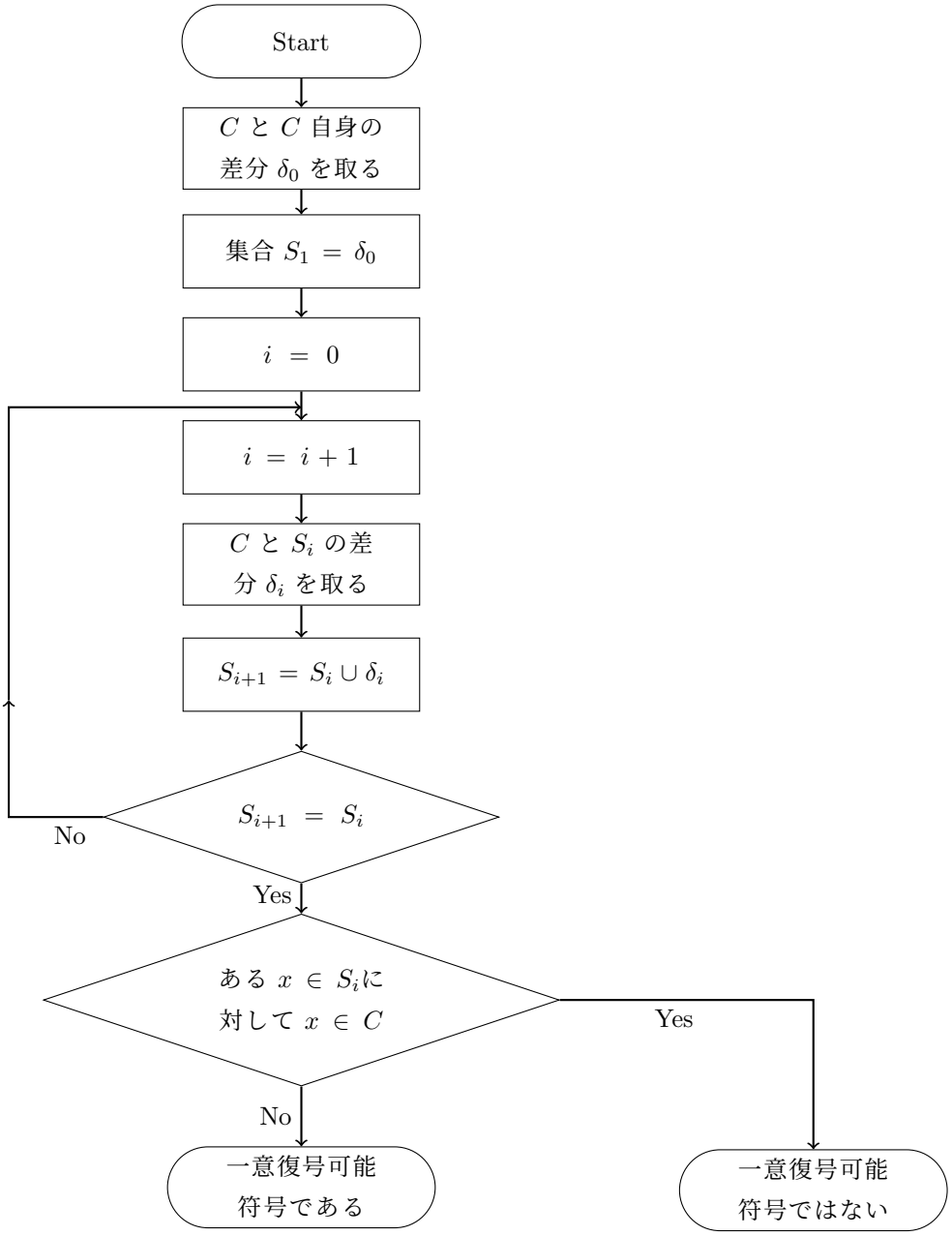

図 3　サーディナス-パターソン・アルゴリズム

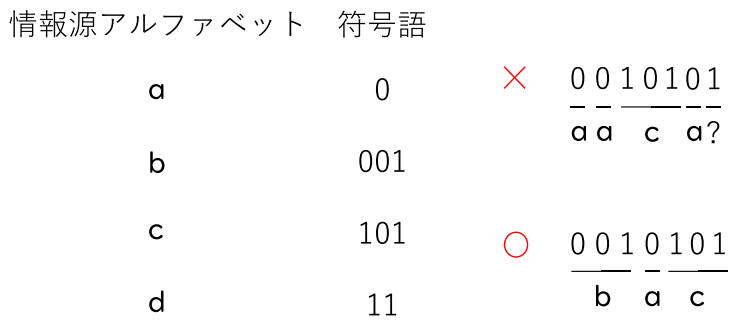

図 4　符号 C が $C(a)=0, C(b)=001, C(c)=101, C(d)=11$ で与えられており，符号語系列 0010101 を復号する場合

できない．よって，最初の符号語が $C(b)=001$ と定まる．次に符号語 001 が削除された符号語系列 0101 の復号を考えると，最初の符号語 0 が $C(a)=0$ なのか，$C(b)=001$ なのかは符号語系列の後の部分を調べないと判別できないが，01 まで調べた時点で $C(b)=001$ で復号することができないため，$C(a)=0$ と復号される．そして残りの文字列 101 は $C(c)=101$ で復号される．このように符号語系列 0010101 に対して，一意に復号することができる．ここで注目して欲しいのが 0101 という系列である．もし与えられた符号語系列の後半部分が 01 で繰り返される 0010̇1̇ だった場合，$C(a)=0$ と $C(c)=101$ を使って繰り返し復号できてしまうため，この系列に終わりを設けない限り，復号を確定させることができない．つまり，この符号の最大復号遅延は無限大になる．

3.2　最大復号遅延の有限値の求め方

本節では，最大復号遅延の有限値の求め方を述べる．図 5 のように，符号 C が $C(a)=0, C(b)=001, C(c)=110$ で与えられており，符号語系列 00110 を復号する場合を考える．まず，最初の符号語 0 が $C(a)=0$ なのか，$C(b)=001$ なのかは符号語系列の後の部分を調べないと判別できない．先頭を $C(b)=001$ で復号するパターンで，符号語系列 001 が削除された符号語系列 10 の復号を考えたとき，最後の 0 まで調べた時点で $C(c)=110$ では復号できないことが確定し，最初の符号語が $C(a)=0$ と定まる．つまり，$C(a)=0$ と復号するまでに符号語系列 00110 の最後の系列 0 まで調べたため，a の復号遅延が 4 となる．

一方，図 6 は，符号 $C=\{0, 001, 110\}$ において，先頭が符号語 0 と 001 から始まり，同じ文字列で構成される 2 つの符号語系列を作るときの遷移，つまり 2 つの復号パターンがある符号語系列を作るときの遷移を表している．まず，① の符号語系列 0 と ② の符号語系列 001 の差分 01 が取れる．次に差分 01 と差分が取れる符号語 0 を ① の系列に追加し，差分 1 が

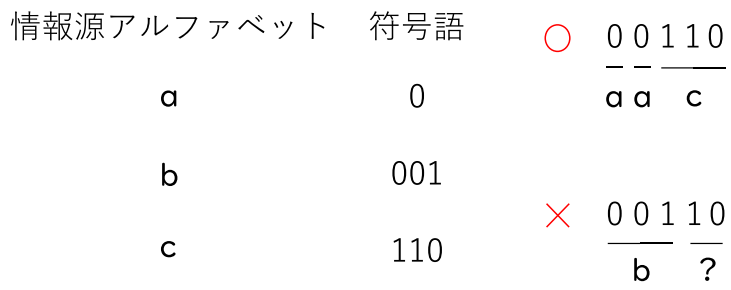

図 5　符号 C が $C(a) = 0, C(b) = 001, C(c) = 110$ で与えられており，符号語系列 00110 を復号する場合

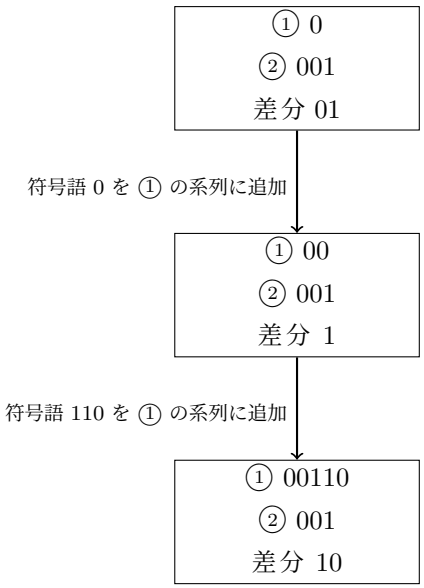

図 6　符号 $C = \{0, 001, 110\}$ において先頭が符号語 0 と 001 から始まり，同じ文字列で構成される 2 つの符号語系列を作るときの遷移

取れる．次に差分 1 と差分が取れる符号語 110 を ① の系列に追加し，差分 10 が取れる．最後に差分 10 と差分が取れる符号語がないため，これ以上 2 つの復号パターンがある符号語系列を作ることができない．つまり差分が取れなくなった時，符号語系列 00110 の復号パターンが 1 通りに定まるため，符号語系列 00110 を作るまでに取れた差分の系列の総長が，この符号の最大復号遅延と考えることができる．しかし，ここで注目したいのが，差分 01 と符号語 0 の差分 1 である．この差分 1 は 差分 01 の系列から取れた差分であるため，差分 1 を復号遅延としてカウントしてしまうと差分 01 と重複してしまう．すなわち符号語よりも短い差

分のみをカウントする必要がある．したがって，符号語 0 と符号語 001 の差分 01 と，差分 1 と符号語 110 の差分 10 の総長 4 が，この符号の最大復号遅延となる．次節では，与えられた符号の最大復号遅延を調べる方法について述べる．

3.3 最大復号遅延を調べるアルゴリズム

最大復号遅延が有限であるか無限であるかを調べるには，サーディナス-パターソン・アルゴリズムと同様に差分を使う．図 2 において ① の符号語系列 00101 を作るときの遷移を見てみると，まず ① の符号語系列 0 に符号語 0 を追加し，次に ① の符号語系列 00 に符号語 101 を追加し，符号語系列 00101 を作っている．この先の符号語系列を考えたとき，② の符号語系列 001 にも同様に符号語 0 と 101 を加えることで，2 つの復号パターンが存在する符号語系列を無限に作ることができる．そもそも符号語系列を作る際には，差分と符号語を比較して差分が取れる符号語を系列に追加している．つまり，既出の差分 01 が出てきた時点で，この符号の最大復号遅延が無限になるとわかる．したがって，最大復号遅延は 2 つの復号パターンが存在する符号語系列において，同じ差分があるなら最大復号遅延が無限になり，もし，2 つの復号パターンが存在する全ての符号語系列において，同じ差分が無い場合には最大復号遅延は有限となる．本研究では，サーディナス-パターソン・アルゴリズムを拡張し，与えられた符号について一意復号可能性と最大復号遅延の両方を調べるアルゴリズムを提案する．これを本論文では符号評価アルゴリズムと呼ぶことにし，そのアルゴリズムは図 7 のようになる．

では実際に，アルゴリズムを使って一意復号可能性と最大復号遅延を求める．符号 $C = \{0, 001, 101, 11\}$ に図 7 のアルゴリズムを適用する．まず，符号語 0 と符号語 001 の差分 01 が取れ，$S = \{01\}$ となり，$L = 2$ となる．次に，$f(01)$ として次のアルゴリズムに移る．そして差分 01 と C の差分を取ると，符号語 0 との差分 1 が取れ，$S = \{01, 1\}$ となる．ここで，(差分 01 の系列長) $>$ (符号語 0 の符号語長) より $L = 2$ のままである．次に差分 1 と符号語を比較すると，符号語 101 との差分 01 が取れるが，$\{01\} \in S$ であるため，$L_{max} = \infty$ となる．さらに差分 1 は符号語 11 との差分 1 を取ることができるが，これも $\{1\} \in S$ であり，反復処理は終了する．$L_{max} = \infty$ より，この符号の最大復号遅延は無限である．さらに差分と符号語が一致したことは一度もないため，この符号は一意復号可能符号である．

このように，差分と符号語を比較して一意復号可能性を調べるサーディナス-パターソン・アルゴリズムに，差分同士を比較するプロセスを加えることで，一意復号可能性と最大復号遅延の両方を調べることができる．前章と本章では，1 つの符号表を使ったものを符号と呼び，それについて一意復号可能性と最大復号遅延を調べる方法を述べた．次章からは 2 つの符号表を使ったものを符号と呼び，それについて一意復号可能性と最大復号遅延を調べる方法について述べる．

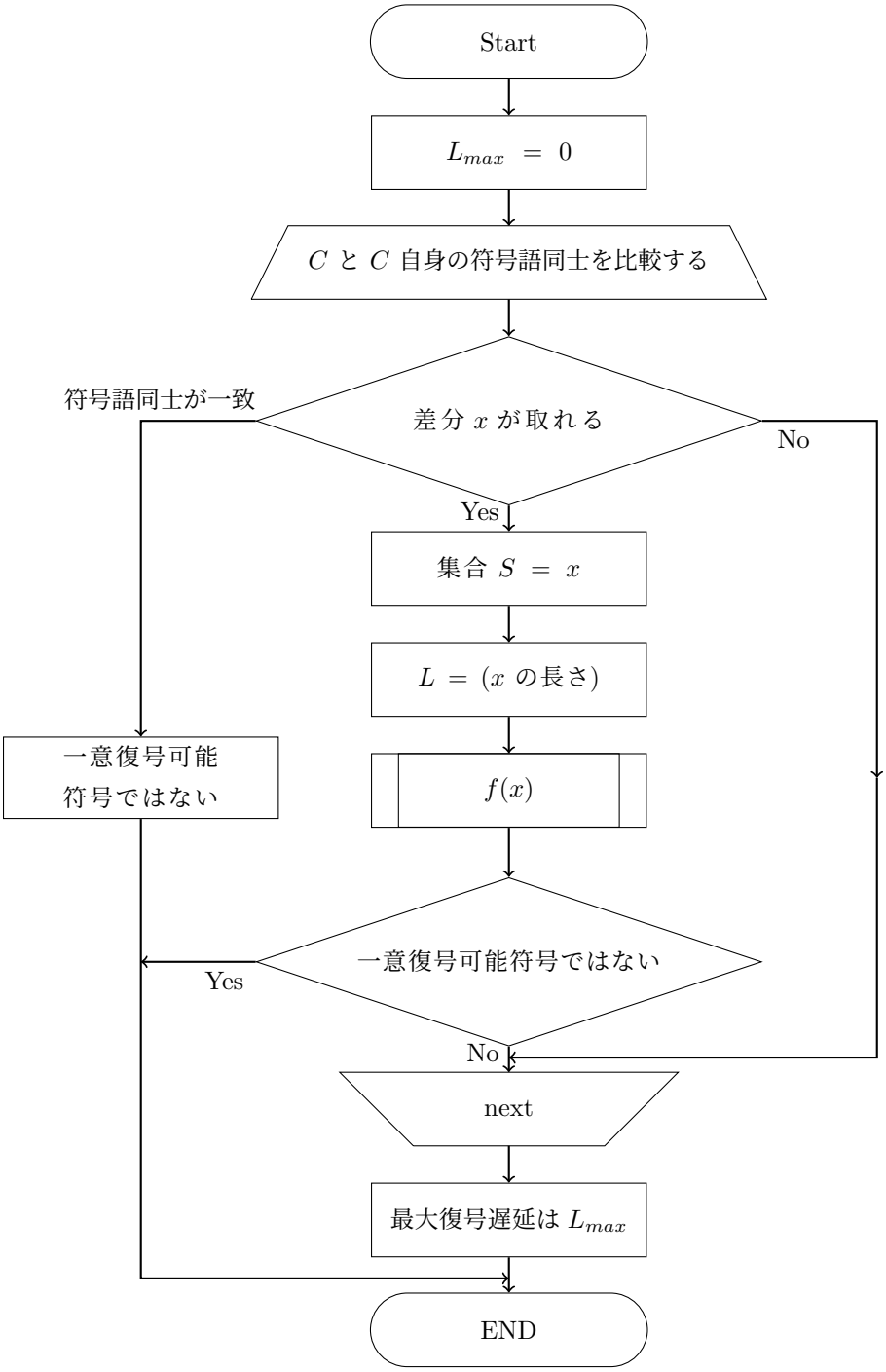

図 7-1　符号評価アルゴリズム

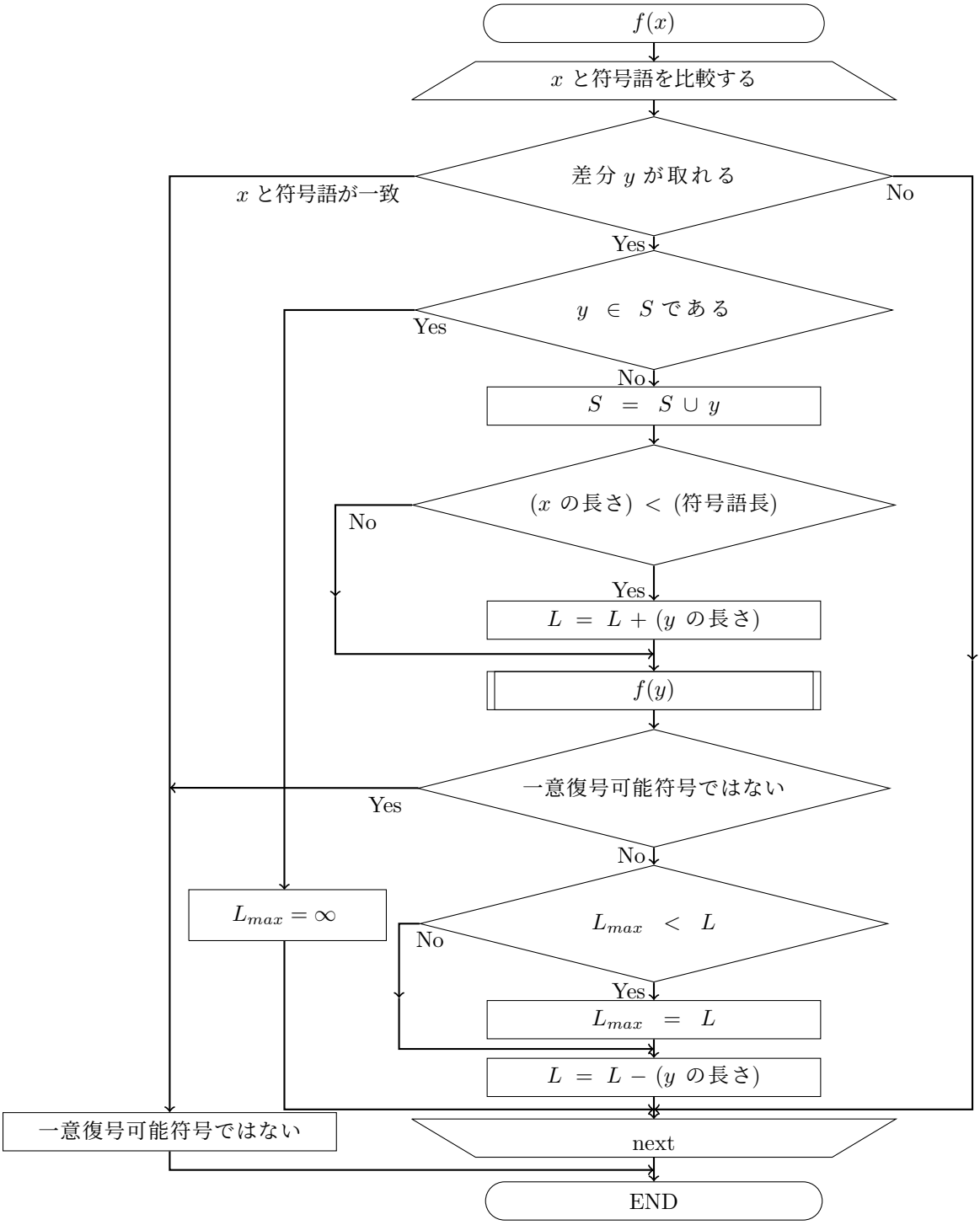

図 7-2　符号評価アルゴリズム

4　2 つの符号表を使った符号

1 章で述べた AIFV 符号は，2 つの符号表を使った符号である．最大復号遅延が 2 ビットになる代わりにハフマン符号よりも小さい平均符号語長を達成できる．よって，最大復号遅延が有限である符号について平均符号語長を調べたら，AIFV 符号の他にもハフマン符号よりも小さい平均符号語長を達成できる符号が存在するかもしれない．本章からは，2 つの符号表を使ったものを符号と呼び，まずは符号化方法と一意復号可能，復号遅延の定義を拡張する．

4.1　符号の表現

まず，AIFV 符号を例にして本章から考える符号について述べる．AIFV 符号は 2 つの符号表 T_0 と T_1 で構成され，情報源記号 $x_i \in \mathcal{X}$ が葉または中間ノードで符号化された時，x_{i+1} はそれぞれ T_0 または T_1 で符号化される．例えば，情報源アルファベットが $\mathcal{X} \stackrel{\text{def}}{=} \{a, b, c, d\}$，符号 T が $T_0 = \{0, 10, 11, 1100\}$ と $T_1 = \{01, 10, 11, 1100\}$ で与えられる場合，図 8 を見ると，黒く塗りつぶされている点が符号語で，中間ノードにある符号語は四角，葉にある符号語は丸で表されている．データ系列が $cbcd$ で与えられ，最初の文字が符号表 T_0 を用いて符号化されるとき，符号語系列と符号表の遷移は図 9 のようになる．

このように 2 つの符号表を使った符号では符号化する際に，各符号語について次の符号化に用いる符号表が定められていなければならない．そこで，本研究では，符号を表 1 のように表す．

これより先では，符号アルファベットの並びと遷移先をまとめたものを符号語と呼び，

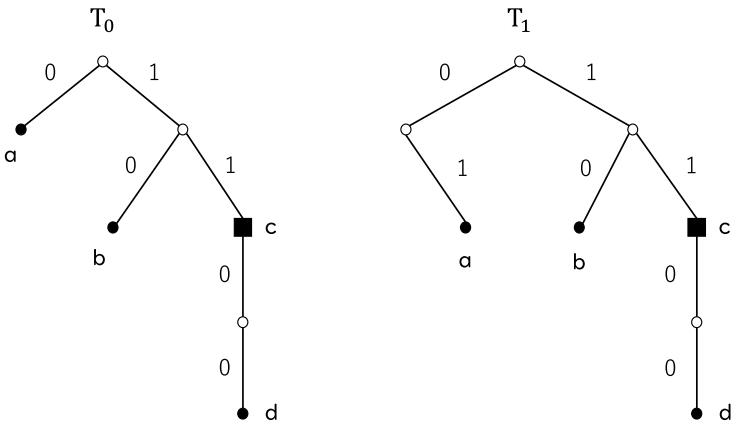

図 8　AIFV 符号の符号木

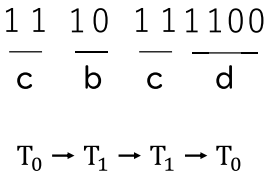

図 9　AIFV 符号においてデータ系列 *cbcd* を符号化したときの符号語系列と符号表の遷移

T_0	
アルファベット	符号語
a	$0T_0$
b	$10T_0$
c	$11T_1$
d	$1100T_0$

T_1	
アルファベット	符号語
a	$01T_0$
b	$10T_0$
c	$11T_1$
d	$1100T_0$

表 1　AIFV 符号

符号アルファベットの並びを出力系列と呼ぶ．そして符号語系列を考える際には，遷移先を反映させた符号アルファベットの並びを考える．例えば，図 8 の AIFV 符号は $T_0 = \{0T_0, 10T_0, 11T_1, 1100T_0\}$ と，$T_1 = \{01T_0, 10T_0, 11T_1, 1100T_0\}$ で構成されており，符号語系列は 011101100 と表す．

4.2　一意復号可能性

2.1 節で定義した一意復号可能符号の定義を，2 つの符号表を用いた符号に拡張する．

定義 7　情報源アルファベット $\mathcal{X}$ の要素と，それを符号語に変換するために用いた符号表から，次に遷移する符号表が決まるため，任意の情報源記号 $x_n \in \mathcal{X}$ の符号表 C_n は

$$C_n = f(C_{n-1}, x_{n-1}) \tag{3}$$

で与えられる．

定義 8　符号 T の拡大符号とは，情報源アルファベット $\mathcal{X}$ の要素の有限個の系列から，任意の有限長の系列の集合 $\mathcal{D}^*$ への写像として定義され，

$$T^*(x_1 x_2 \cdots x_n) = T_{C_1}(x_1) T_{C_2}(x_2) \cdots T_{C_n}(x_n) \tag{4}$$

で与えられる．ここで，x_1 を符号化するために用いた符号表 C_1 は元から与えられているものとする．

定義 9　2 つの符号表を用いた符号 T の拡大符号が正則であるとき，T を一意復号可能符号と呼ぶ．

4.3　復号遅延

復号遅延の考え方は，2 つの符号表を使った符号でも変わらない．表 1 の AIFV 符号において符号語系列が 0110110 で与えられ，最初の文字の復号に符号表 T_0 を用いるとき，図 10 のように復号される．まず，最初の系列 0 は 符号表 T_0 を見ると a でしか復号できないため，このとき a の復号遅延が 0 であると定義する．次に出力系列 0 が削除された符号語系列 110110 の復号を考えると，1101 まで調べた時点で，出力系列 1100 の d ではなく，出力系列 11 の c が復号されるため，c の復号遅延が 2 になる．そして出力系列 11 が削除された符号語系列 0110 の復号を考えると，系列 01 は復号遅延 0 で a と復号され，残りの系列 10 も復号遅延 0 で b と復号される．このように各符号語に遷移先が付いていても，復号遅延は出力系列を見て判断する．2 つの符号表を用いた符号についても前節と同様に，最大復号遅延が有限になるか無限になるかに着目する．

5　2 つの符号表を使った符号における符号評価アルゴリズム

3 章のアルゴリズムと同様に，一意復号可能性と最大復号遅延の両方を調べるアルゴリズムを符号評価アルゴリズムと呼び，どのように調べるかを述べる．

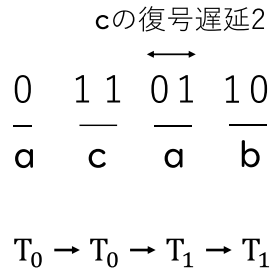

図 10　AIFV 符号において符号語系列 0110110 の復号

5.1 一意復号可能性を調べる方法

まず，新たに差分を定義する．

定義 10 2 つの系列 aC_a, bC_b において，a が b の語頭になっているとき，差分は

$$(b-a)C_b \tag{5}$$

と表す．ここで，$(b-a)$ を差分系列，C_a を差分の遷移先と呼ぶ．そして，差分と遷移先をまとめた $(b-a)C_b, C_a$ を $bC_b - aC_a$ と表す．

一意復号可能性は，サーディナス-パターソン・アルゴリズムと同じように差分を使って調べることができる．図 11 は，符号 T が $T_0 = \{0T_1, 001T_0, 100T_0\}$ と，$T_1 = \{10T_0, 011T_0\}$ で構成され，先頭が符号語 $0T_1$ と $001T_0$ から始まり，同じ文字列で構成される 2 つの符号語系列を作るときの遷移，つまり 2 つの復号パターンがある符号語系列を作るときの遷移を表している．まず，符号語 $0T_1$ と $001T_0$ で差分 $01T_0$ と差分の遷移先 T_1 が出てくる．仮に，符号表 T_1 の符号語に 01 という出力系列が含まれていた場合，001 という符号語系列に対して，2 通りの復号パターンがあるため，一意復号可能ではない．つまり，符号語同士の差分系列が，遷移先の符号語の出力系列と一致していれば，一意復号可能ではないと分かる．次に差分 $01T_0$ と T_1 の符号語で差分が取れるか調べると，符号語 $011T_0$ との差分 $1T_0$ と差分の遷移先 T_0 が取れる．このとき ① の系列に符号語 $011T_0$ を加え，系列 $0011T_0$ を作る．もし，符号表 T_0 の符号語に出力系列 1 が含まれていた場合，0011 という符号語系列に対して，2 通りの復号パターンがあるため，一意復号可能ではない．つまり，出てきた差分と符号語の差分系列が，遷移先の符号語の出力系列と一致していれば，一意復号可能ではないと分かる．さらに，差分 $1T_0$ と T_0 の符号語で差分が取れるか調べると，符号語 $100T_0$ との差分 $00T_0$ と差分の遷移先 T_0 が取れる．このとき ② の系列に符号語 $100T_0$ を加え，系列 $001100T_0$ を作る．ここで，差分系列 00 が遷移先の符号表 T_0 の符号語の出力系列と一致してないことがわかる．このような手順で符号 T の一意復号可能性を調べると，図 11 の左側のパスにおいて ① の系列 $0011011T_0$ を作った時に，差分 $11T_0$ が出てきたが，遷移先の符号表 T_0 の符号語と差分を取ることができないため，これ以上 2 つの復号パターンがある符号語系列を作ることはできない．一方，右側のパスを見ると，差分系列 1 と遷移先の符号表 T_0 の符号語の比較と，差分系列 00 と遷移先の符号表 T_0 の符号語の比較を繰り返しており，この先の符号語系列を考えても同じプロセスの繰り返しになってしまうため，これ以上 2 つの復号パターンがある符号語系列を考える必要はない．したがって，符号 T は，差分系列と遷移先の符号語の出力系列が一致することがないため一意復号可能符号となる．このように符号語同士の差分系列，そして出てきた差分と符号語の差分系列が，遷移先の符号語の出力系列と一致しているかで一意復号可

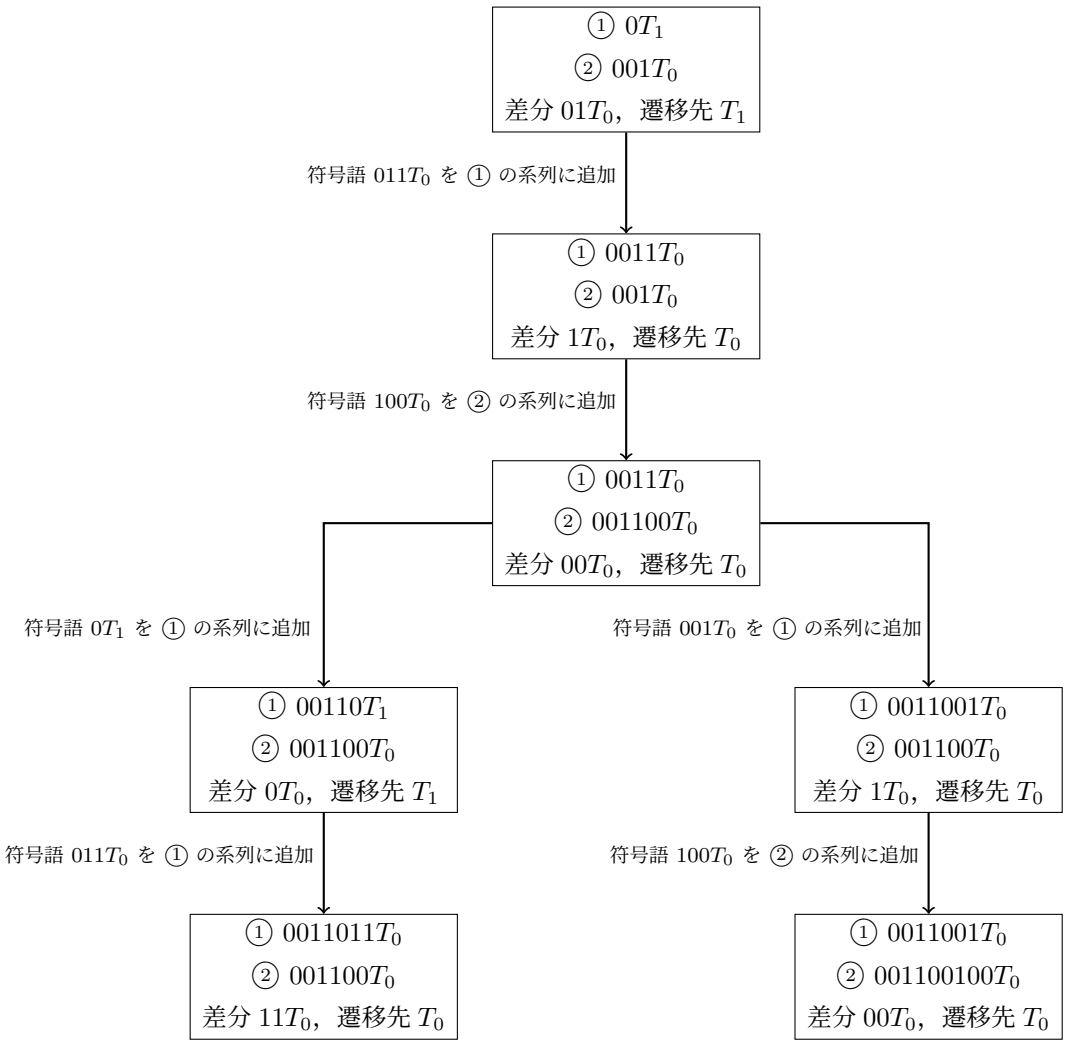

図 11　符号 T が $T_0 = \{0T_1, 001T_0, 100T_0\}$ と，$T_1 = \{10T_0, 011T_0\}$ で構成され，先頭が符号語 $0T_1$ と $001T_0$ から始まり，同じ文字列で構成される 2 つの符号語系列を作るときの遷移

能性を調べることができる．

5.2　最大復号遅延を調べる方法

最大復号遅延を調べるには，一意復号可能性を調べるのと同様に差分を使う．図 11 において ① の系列 $0011T_0$ から ① の系列 $0011001T_0$ を作るときの遷移を見てみると，まず，②

の符号語系列 001 に遷移先 T_0 の符号語 $100T_0$ を追加し，次に ① の符号語系列 0011 に遷移先 T_0 の符号語 $001T_0$ を追加し，系列 $0011001T_0$ を作っている．この先の符号語系列を考えたとき，② の符号語系列に符号語 $100T_0$ を追加し，次に ① の符号語系列に符号語 $001T_0$ を追加することで，2 つの復号パターンが存在する符号語系列を無限に作ることができる．そもそも符号語系列を作る際には，差分と遷移先の符号語を比較して差分が取れる符号語を系列に追加している．つまり，既出の差分 $1T_0$ と遷移先 T_0 が出てきた時点で，この符号の最大復号遅延が無限になるとわかる．したがって，最大復号遅延は 2 つの復号パターンが存在する符号語系列において，同じ $bC_b - aC_a$ があるなら最大復号遅延が無限になり，2 つの復号パターンが存在する全ての符号語系列において，同じ $bC_b - aC_a$ が無い場合には最大復号遅延は有限となる．このようにして最大復号遅延を調べることができ，符号評価アルゴリズムは図 12 のようになる．

5.3 符号評価アルゴリズム

では実際にアルゴリズムを使って一意復号可能性と最大復号遅延を求める．符号 T が $T_0 = \{0T_1, 01T_0, 11T_0\}$, $T_1 = \{0T_0, 11T_0\}$ で構成されている時，符号 T に図 12 のアルゴリズムを適用すると図 13 のようなプロセスになる．まず，符号語 $0T_1$ と $01T_0$ を比較すると，差分 $1T_0$ と遷移先 T_1 が取れて，$S = \{1T_0, T_1\}$，$L = 1$ となる．その後，$f(1T_0, T_1)$ として図 12-2 のアルゴリズムに進み，差分 $1T_0$ と 遷移先 T_1 の符号語を比較すると，符号語 $11T_0$ との差分 $1T_0$ と遷移先 T_0 が取れる． ここで，$\{1T_0, T_0\} \notin S$ なので $S = \{(1T_0, T_1)$, $(1T_0, T_0)\}$, $L = 2$ となる．さらに，$f(1T_0, T_0)$ として差分 $1T_0$ と 遷移先 T_0 の符号語を比較すると，符号語 $11T_0$ との差分 $1T_0$ と遷移先 T_0 が取れる．ここで，$\{1T_0, T_0\} \in S$ なので $L_{max} = \infty$ となる．符号語 $11T_0$ 以外に，差分 $1T_0$ と 遷移先 T_0 の符号語で差分を取ることができないため，反復処理は終了する．$L_{max} = \infty$ より，符号 T の最大復号遅延は無限であり，差分系列と符号語の出力系列が一度も一致しなかったので，一意復号可能符号である．

5.4 考察

2 つの符号表を用いた符号について一意復号可能性と最大復号遅延を調べるアルゴリズムを提案した．アルゴリズムでは，差分系列と遷移先の符号語の出力系列を比較して，一意復号可能性を調べている．そして遷移先が同じである差分同士を比較して，最大復号遅延を調べている．本研究では，AIFV 符号を例にして 2 つの符号表を用いた符号に着目した．しかし，アルゴリズムの性質上，符号語の遷移先を指定すれば 3 つ以上の符号表を使った符号にも応用できる．このアルゴリズムを拡張すれば，3 つ以上の符号表を使った符号を作ることも可能であると考える．

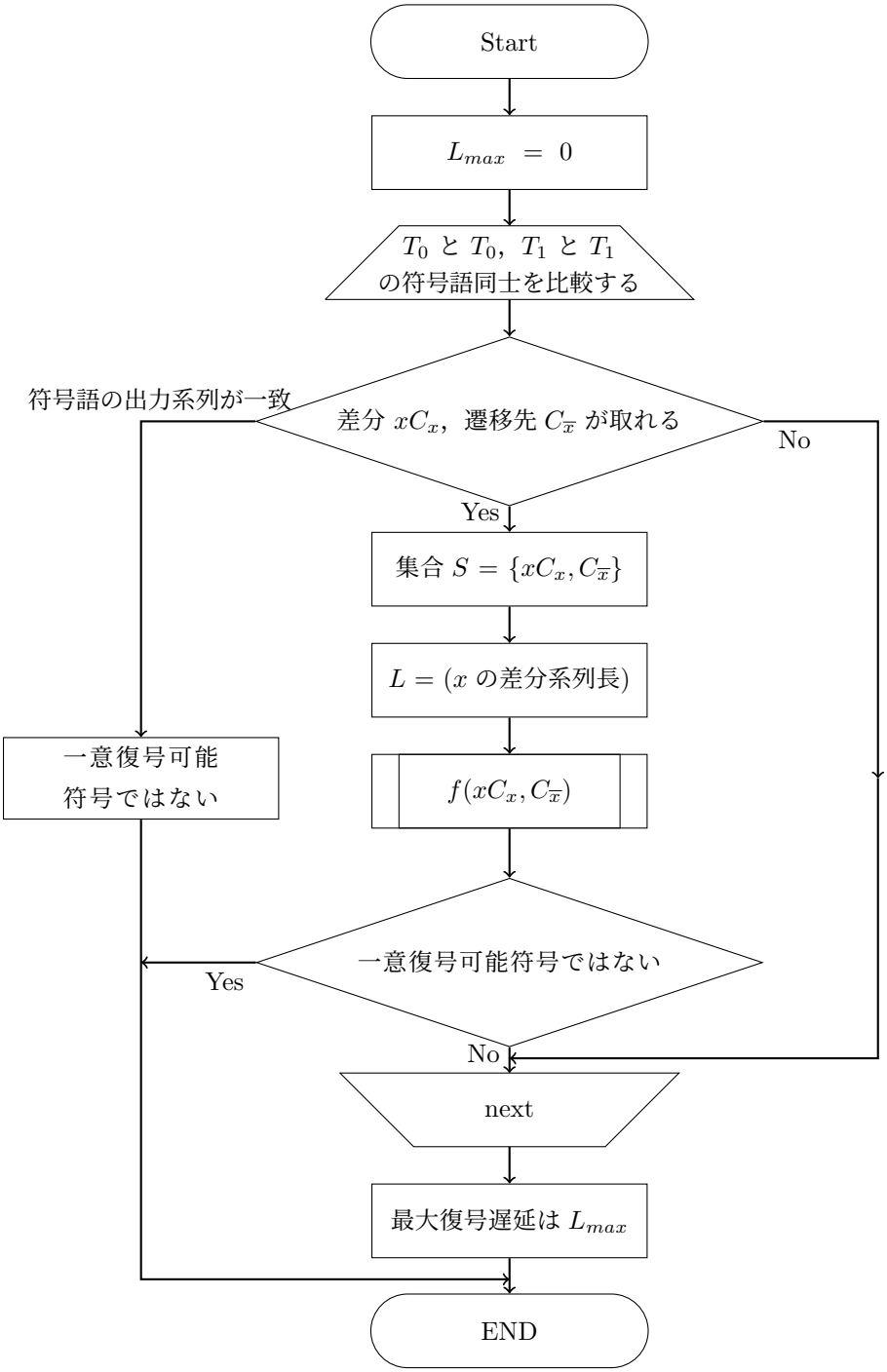

図 12-1　2 つの符号表を用いた符号の符号評価アルゴリズム

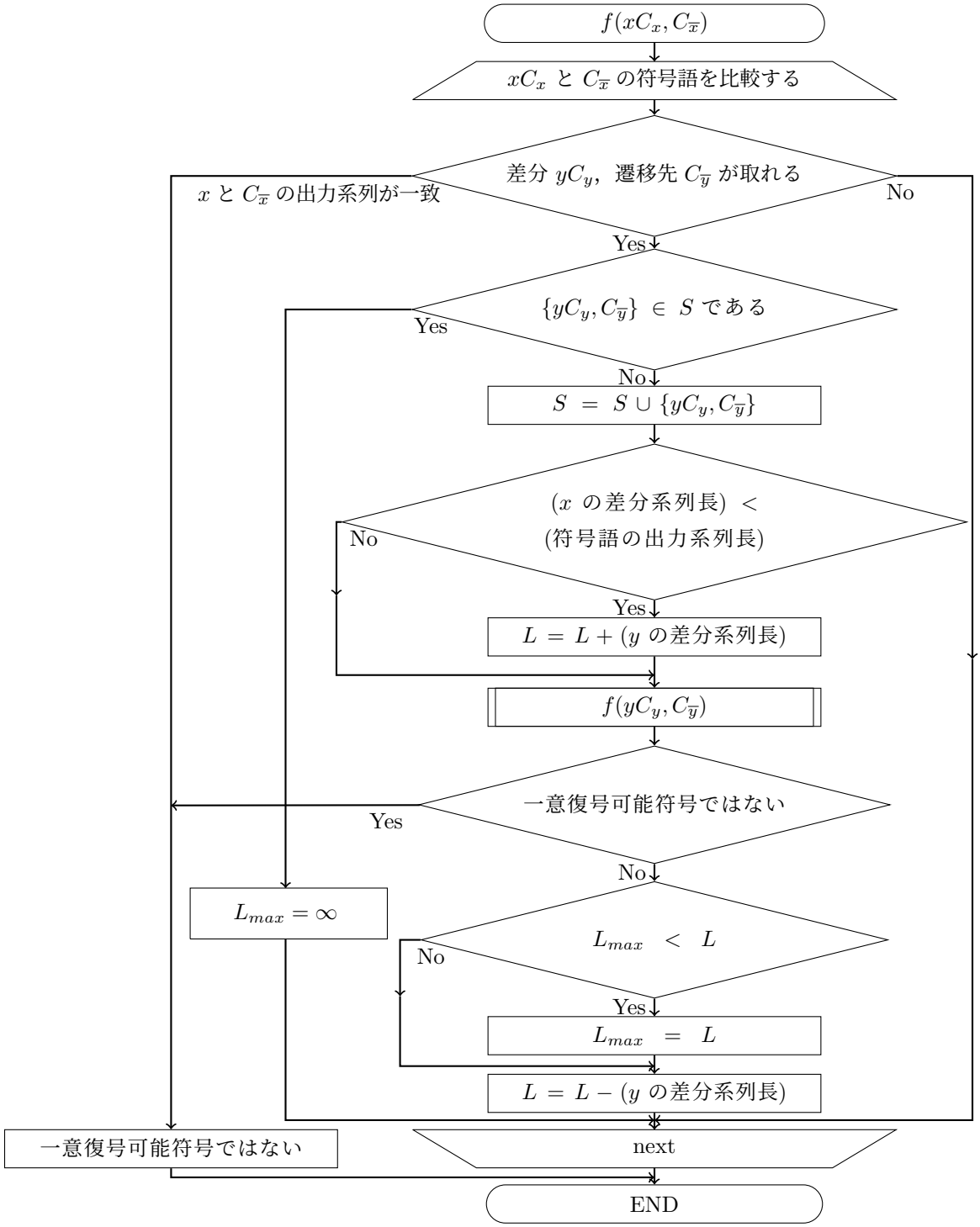

図 12-2　2 つの符号表を用いた符号の符号評価アルゴリズム

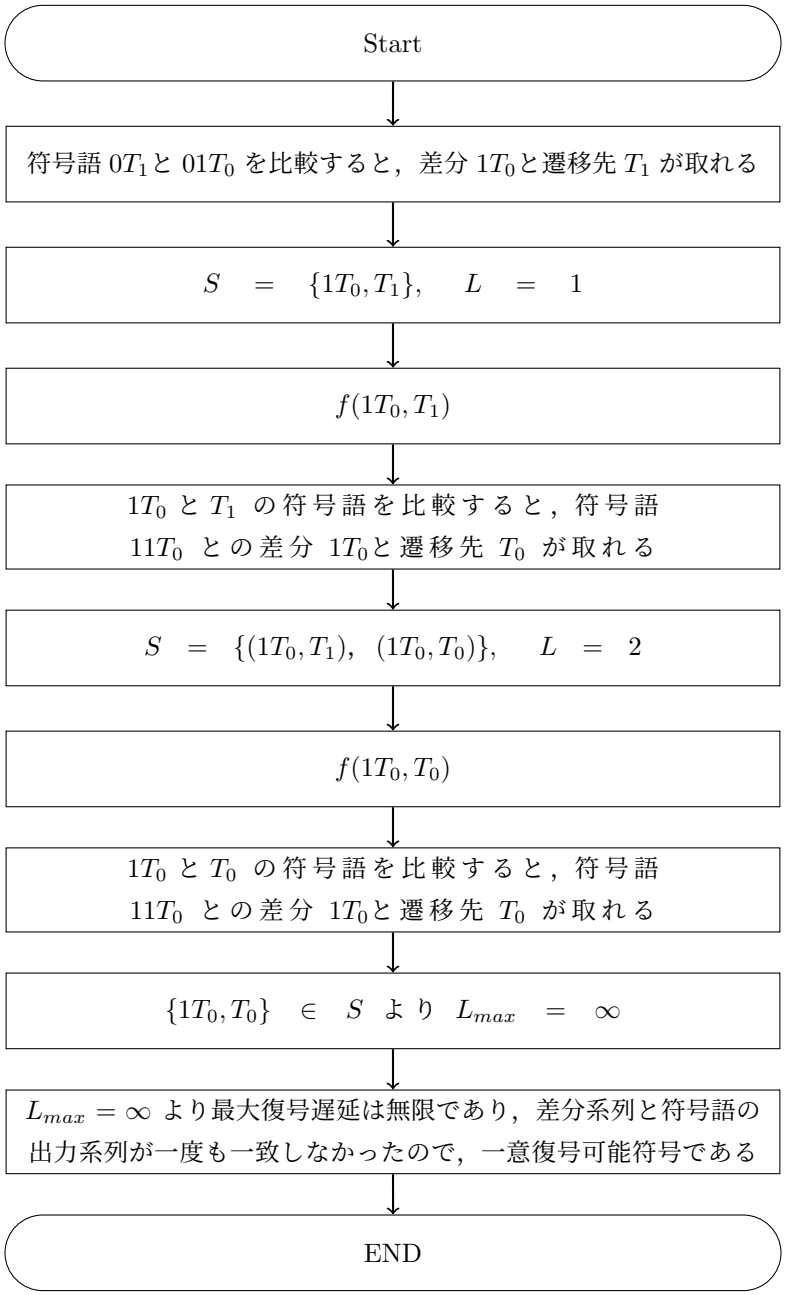

図 13　符号 T が $T_0 = \{0T_1, 01T_0, 11T_0\}$, $T_1 = \{0T_0, 11T_0\}$ で構成されている時の一意復号可能性と最大復号遅延を調べるプロセス

6 まとめ

本研究では，1 つの符号表を使った符号に対して，最大復号遅延を調べる方法を提案し，2 つの符号表を使った符号に対して，一意復号可能性と最大復号遅延を調べる方法を提案した．今後の課題として，符号語数や符号語長による計算量の評価や，最大復号遅延が有限である符号に関する平均符号語長の評価があげられる．さらに最大復号遅延が無限になる符号について，最終 1 ビットまで系列を見なければ復号を確定させることができない性質をセキュリティ分野で応用できると考える．

謝辞

本研究にあたり，細かく指導してくださった指導教員の西新幹彦准教授に感謝の意を表する．

参考文献

[1] Gautam Bandyopadhyay, “A Simple Proof of the Decipherability Criterion of Sardinas and Patterson,” Information and Control, vol.6, no.4, pp.331-336, Dec. 1963.

[2] 小嶋徹也，はじめての情報理論，近代科学社，2011.

[3] Hirosuke Yamamoto, Masato Tsuchihashi, Junya Honda, “Almost Instantaneous Fixed-to-Variable Length Codes,” IEEE Transactions on Information Theory, vol.61, no.12, pp.6432-6443, Dec. 2015.

付録 A　ソースコード

A.1　1 つの符号表を使った符号において一意復号可能性と最大復号遅延を調べるプログラム

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <math.h>

// 入力

char *cw[4] = {"01", "010", "110", "1011"};
int num = 4; //符号語数

//

//深さ
int d;
//差分のポインタ
char *s[100];
//符号語の番号を格納
int cwnum[100];
//復号遅延判定
int d_flag = 0;
int flag = 0; //先頭の符号語を格納するため
//差分の値を入れる
char *x;
char *y;
char *z;
//遅延のカウント
int count = 0;
//最大遅延
int max_delay = 0;
//遅延の文字を格納
char *de_char[100];
//遅延の文字数
int de_count = 0;
//最大遅延を出す符号語を格納
char *cw_max[2];
//無限遅延の文字列を格納
char *inf_char[100];
//無限遅延と判定した時の文字列の文字数
int inf_count = 0;
//無限遅延を出す符号語を格納
char *cw_inf[2];



//差分を取る
char *sabun(char *a,char *b){
    while (*a==*b && *a!='\0' && *b!='\0'){
        a++;
        b++;
    }
    if (*b == '\0'){
        return(a);
    }
    if (*a == '\0'){
        return(b);
    }
    return(NULL);
}
```

```
//文字数カウント
int delcount(char *q){
    int r = 0;
    while (*(q + r) != '\0'){
        r++;
    }
    return(r);
}

//差分判定
char* process(char *p){
    if ((y = sabun(p, cw[cwnum[d]])) == NULL){
        return(NULL);
    } else if (*y == '\0'){
        printf("not decodable code");
        exit(0);
    }
    //差分がでた時
    //同じ差分がある確認
    for (int i=0;i<(d+1);i++){
        if (strcmp(s[i],y) == 0){
            d_flag = 1; //遅延が無限と判定
            for (int j=0;j<(d+1);j++){
                //遅延の文字を格納
                if (j < 99){
                    inf_char[j] = s[j];
                } else if (j >= 99){
                    printf("inf_char[] over");
                    exit(0);
                }
            }
            inf_count = d+1;
            inf_char[inf_count++] = y;
            return(p);
        }
    }
    //同じ差分が出なかった時
    if (d < 100){
        s[d+1] = y;
        cwnum[d+1]= -1;
        //差分の文字数カウント
        if (delcount(p) < delcount(cw[cwnum[d]])){
            de_count = de_count + delcount(y);
        }
        d++;
        return(y);
    } else {
        printf("s[] over");
        exit(0);
    }
}

int main(void){
    for (int i=0;i<num;i++){
        for (int j=i+1;j<num;j++){
            if ((z = sabun(cw[i], cw[j])) == NULL){
                continue;
            } else if (*z == '\0'){
                printf("not decodable code");
                exit(0);
            }
            s[0] = z;
            de_count = delcount(z);
            cwnum[0] = -1;
            d = 0;
            while(d >= 0 && d < 100){
                if (cwnum[d] == (num - 1)){
                    d--;
                    //最後の差分の文字数分遅延を減らす
```

```
                    de_count = de_count - delcount(s[d+1]);
                    z = s[d];
                    continue;
                }
                cwnum[d]++;
                x = process(z);
                if (x == NULL){
                    if (max_delay < de_count){
                        max_delay = de_count;
                        cw_max[0] = cw[i];
                        cw_max[1] = cw[j];
                        for (int k=0;k<(d+1);k++){
                            //遅延の文字を格納
                            if (k < 100){
                                de_char[k] = s[k];
                            } else if (k >= 100){
                                printf("de_char[] over");
                                exit(0);
                            }
                        }
                        count = d+1;
                    }
                    continue;
                }
                if (*x == '\0'){
                    printf("not decodable code");
                    exit(0);
                }
                if (flag = 1){
                    cw_inf[0] = cw[i];
                    cw_inf[1] = cw[j];
                    flag = 0;
                }
                z = x;
            }
            if (d >= 100){
                printf("s[] over");
                exit(0);
            }
        }
    }
    printf("uniquely decodable code\n");
    if (d_flag == 1){
        printf("infinite delay\n");
        printf("first codeword\n");
        printf("%s , %s\n",cw_inf[0],cw_inf[1]);
        printf("codeword diff :\n");
        for (int p=0;p<inf_count;p++){
            printf("%s",inf_char[p]);
            printf("\n");
        }
    } else {
        printf("finite delay\n");
        printf("max_delay = %d\n",max_delay);
        printf("first codeword\n");
        printf("%s , %s\n",cw_max[0],cw_max[1]);
        printf("codeword diff :\n");
        for (int p=0;p<count;p++){
            printf("%s",de_char[p]);
            printf("\n");
        }
    }
    return(0);
}
```

A.2 2 つの符号表を使った符号において一意復号可能性と最大復号遅延を調べるプログラム

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <math.h>

// 入力

char *cwA[4] = {"01B", "010A", "1011A", "110A"}; //'A' はツリー 1,'B' はツリー 2 に遷移
int numA = 4; //符号 1 の符号語数

char *cwB[3] = {"00B", "0011A", "111A"};
int numB = 3; //符号 2 の符号語数

//

//深さ
int d;
//遷移先のツリー
char T;
//ツリーの情報を格納
char tr[100];
//差分のポインタ
char *s[100];
//符号語の番号を格納
int cwnum[100];
//復号遅延判定
int d_flag = 0;
int flag = 0; //先頭の符号語を格納するため
//差分の値を入れる
char *x;
char *y;
char *z;
//遅延のカウント
int count = 0;
//最大遅延
int max_delay = 0;
//遅延の文字列を格納
char *de_char[100];
//遅延の文字数
int de_count = 0;
//最大遅延を出す符号語を格納
char *cw_max[2];
//有限遅延の差分のツリーの情報を保存
char de_tr[100];
//無限遅延の文字列を格納
char *inf_char[100];
//無限遅延と判定した時の文字列の文字数
int inf_count = 0;
//無限遅延を出す符号語を格納
char *cw_inf[2];
//無限遅延の差分のツリーの情報を保存
char inf_tr[100];



//差分を取る
char *sabun(char *a,char *b){
    while (*a==*b && *a!='A' && *a!='B' && *b!='A' && *b!='B'){
        a++;
        b++;
```

```
    }
    if (*b == 'A'){
        T = 'A';
        return(a);
    }
    if (*b == 'B'){
        T = 'B';
        return(a);
    }
    if (*a == 'A'){
        T = 'A';
        return(b);
    }
    if (*a == 'B'){
        T = 'B';
        return(b);
    }
    return(NULL);
}

//文字数カウント
int delcount(char *q){
    int r = 0;
    while (*(q + r) != '\0'){
        r++;
    }
    return(r-1);
}

//差分判定
char* process(char *p, char t){
    char *cw;
    if (t == 'A'){
        cw = cwA[cwnum[d]];
    }
    if (t == 'B'){
        cw = cwB[cwnum[d]];
    }
    if ((y = sabun(p, cw)) == NULL){
        return(NULL);
    } else if (*y == 'A' || *y == 'B'){
        printf("not decodable code");
        exit(0);
    }
    //差分がでた時
    //同じ差分がある確認
    for (int i=0;i<(d+1);i++){
        if (T == tr[i]){
            if (strcmp(s[i],y) == 0){
                d_flag = 1; //遅延が無限と判定
                flag = 1;
                for (int j=0;j<(d+1);j++){
                    //遅延の文字を格納
                    if (j < 99){
                        inf_char[j] = s[j];
                        inf_tr[j] = tr[j];

                    } else if (j >= 99){
                        printf("inf_char[] over");
                        exit(0);
                    }
                }
                inf_count = d+1;
                inf_char[inf_count] = y;
                inf_tr[inf_count++] = T;
                return(p);
            }
        }
    }
```

```
    //同じ差分が出なかった時
    if (d < 100){
        s[d+1] = y;
        tr[d+1] = T;
        cwnum[d+1]= -1;
        //差分の文字数カウント
        if (delcount(p) < delcount(cw)){
            de_count = de_count + delcount(y);
        }
        d++;
        return(y);
    } else {
        printf("s[] over");
        exit(0);
    }
}

//探索
int search(char *c[], int n){
    for (int i=0;i<n;i++){
        for (int j=i+1;j<n;j++){
            if ((z = sabun(c[i], c[j])) == NULL){
                continue;
            } else if (*z == 'A' || *z == 'B'){
                printf("not decodable code");
                exit(0);
            }
            s[0] = z;
            de_count = delcount(z);
            if (T == 'A'){
                tr[0] = 'A';
            }
            if (T == 'B'){
                tr[0] = 'B';
            }
            cwnum[0] = -1;
            d = 0;
            while(d >= 0 && d < 100){
                if ((tr[d] == 'A' && cwnum[d] == (numA - 1)) || (tr[d] == 'B' && cwnum[d] == (numB - 1))){
                    d--;
                    //最後の差分の文字数分遅延を減らす
                    de_count = de_count - delcount(s[d+1]);
                    z = s[d];
                    continue;
                }
                cwnum[d]++;
                x = process(z, tr[d]);
                if (x == NULL){
                    if (max_delay < de_count){
                        max_delay = de_count;
                        cw_max[0] = c[i];
                        cw_max[1] = c[j];
                        for (int k=0;k<(d+1);k++){
                            //遅延の文字を格納
                            if (k < 100){
                                de_char[k] = s[k];
                                de_tr[k] = tr[k];
                            } else if (k >= 100){
                                printf("de_char[] over");
                                exit(0);
                            }
                        }
                        count = d+1;
                    }
                    continue;
                }
                if (*x == 'A' || *x == 'B'){
                    printf("not decodable code");
                    exit(0);
```

```
                }
                if (flag = 1){
                    cw_inf[0] = c[i];
                    cw_inf[1] = c[j];
                    flag = 0;
                }
                z = x;
            }
            if (d >= 100){
                printf("s[] over");
                exit(0);
            }
        }
    }
    return(0);
}

int main(void){
    search(cwA,numA);
    search(cwB,numB);
    printf("uniquely decodable code\n");
    if (d_flag == 1){
        printf("infinite delay\n");
        printf("first codeword\n");
        printf("%s , %s\n",cw_inf[0],cw_inf[1]);
        printf("codeword diff , Tree :\n");
        for (int p=0;p<inf_count;p++){
            printf("%s  '%c'",inf_char[p],inf_tr[p]);
            printf("\n");
        }
    } else {
        printf("finite delay\n");
        printf("max_delay = %d\n",max_delay);
        printf("first codeword\n");
        printf("%s , %s\n",cw_max[0],cw_max[1]);
        printf("codeword diff , Tree :\n");
        for (int p=0;p<count;p++){
            printf("%s  '%c'",de_char[p],de_tr[p]);
            printf("\n");
        }
    }
    return(0);
}
```