信州大学工学部

学士論文

遅延特性改善のための
分節木上の符号語の配置に関する一考察

指導教員　西新 幹彦 准教授

学科　電気電子工学科
学籍番号　09T2022J
氏名　荻野 真志

2013 年 3 月 31 日

目次

1 序章 1

1.1 研究の背景 1

1.2 論文の構成 2

2 伝送システムと遅延 2

3 符号 3

3.1 従来研究 3

3.2 符号器のための分節木と符号語 4

3.3 復号器のための分節木と符号語 4

4 実験と考察 5

5 まとめと今後の課題 9

謝辞 9

参考文献 9

付録 A ポアソン過程 10

付録 B ソースコード 10

B.1 伝送実験のプログラム 10

1 序章

1.1 研究の背景

スマートフォンをはじめ, 最近の情報分野では日々技術が進歩してきておりその情報量もますます莫大になってきている．さらに, スマートフォン端末などを使ったアプリ開発の市場が大きくなり, ますます莫大なデータをスマートに送受信できる伝送システムの開発が必要となる．伝送システムを用いて情報を送る場合，そこには必ず遅延が生じる．しかし遅延は少ないほうが情報を送るほうにとっても, 受け取るほうにとっても良いと言えるはずである．よって遅延がより小さいシステムは優れたシステムということになる．そこで本研究では，遅延の小さいデータ伝送の実現方法について考察する．

このような問題に対して従来研究 [1] では，算術符号を用いる方法が提案されている．算術符号は優れたデータ圧縮性能をもつが，遅延は必ずしも小さくはない．これは，確率の大きなシンボルが連続して到着したときなど，シンボルが到着しても符号語の生成が進まないことがあるためである．従来研究 [1] では，情報源のアルファベットとは別に「フィードバックシンボル」と呼ばれるシンボルを考え，符号語の生成がしばらく進まないときにフィードバックシンボルを符号器に入力することによって符号語の生成を促すことが提案されている．算術符号から見れば情報源のシンボルとフィードバックシンボルに特別な区別はない．よって復号器ではフィードバックシンボルも復号されるが，受信者には不要のため，それらは捨てられる．したがって符号語の生成が進む反面，フィードバックシンボルを送る分だけ符号語長が長くなり，その結果送信にかかる時間も長くなる．従来研究 [1] では，フィードバックシンボルを用いることで，全体として遅延を小さくできることが示されている．

また，従来研究 [2] では入力系列を可変長のブロックに分節して符号化するというモデルで遅延を小さくすることが考えられている．入力系列を分節するタイミングは分節木で表現される．通常，分節された系列に符号語を割り当てて符号化するが，これは分節木の葉に符号語が対応していることに相当する．このモデルで遅延が発生する原因は，分節木の葉に到達するまで符号語が生成されないことにある．このことから従来研究 [2] では分節木の中間ノードにも符号語を割り当て，分節木の葉に到達する時間が長く，しばらく符号語が生成されないとき，中間ノードに割り当てられた符号語を送信することが提案されている．このことを「送信機からのフィードバックにより符号語が出力される」という．この方法では分節木の葉とは別に中間ノードにも符号語を用意するため，符号語の数が多くなる．中間ノードの符号語は葉の符号語とは独立に（語頭条件を満たしたもとで）自由に選ぶことができる．葉の符号語と中間ノードの符号語のどちらが用いられるかはタイミングによるが，復号される系列は同一であるため，必然的に符号語は冗長になる．これは符号語の合計長さが長くなることを意味し，やは

り送信にかかる時間が増加する．従来研究 [2] では，中間ノードに符号語を割り当てる方法によって遅延を小さくできることが示されている．

本研究でも分節木を用いた符号器を用い，分節木の中間ノードに符号語を割り当てることによって遅延を小さくする方法を考察する．しかし，従来研究 [2] とは異なり，本研究では本質的に符号語を増やさない．したがって，中間ノードに符号語が割り当てられていない場合に比べて符号語の合計長さは増加しない．その代わり，中間ノードに割り当てる符号語は自由に選ぶことはできず，葉の符号語にしたがって一意に決定される．本研究では，いくつかの条件のもとで遅延の計測実験を行い，提案法の有効性を検証する．

1.2　論文の構成

本論文は次のような構成をとる．第 2 章では本研究で想定した伝送システムの概要と伝送遅延の定義づけを行う．第 3 章では符号器と復号器の仕組みに焦点を当て, 従来研究と本研究の位置づけについて述べる．第 4 章では実験内容と考察を述べる．第 5 章ではまとめと今後の課題を述べる．論文の最後に付録として情報源の到着過程で使用したポアソン過程についての説明と実験で使用したソースコードを掲載した．

2　伝送システムと遅延

図 1 は本研究で想定している伝送システムのモデル図である．図の情報源は二元アルファベット $\{a, b\}$ をもつ．情報源から出力されたシンボル列を情報源系列という．情報源系列は到着レート λ をもつポアソン過程にしたがって符号器に入力される．ポアソン過程についての詳細は付録に記述する. 符号器は入力されたシンボルにしたがって符号語を出力する．符号語は送信機に入る前に一旦，バッファに蓄えられる．そしてバッファからひとシンボルづつに送信機によって通信路に送信される．簡単のため，ひとつのシンボルの送信にかかる時間を 1 秒とする．送信された符号語は復号器で受信される．通信路による誤りは発生しないと仮定する．復号器は受け取った符号語にしたがって元の情報源系列を復元する．以上が本研究で考える伝送システムの全体像である．

一つのシンボルが符号器に入力されて符号語となり送信バッファに送られて，そこから復号器に送られシンボルが復元されるまでに要する時間を遅延という．情報源系列に対応して得られる遅延の列は確率過程となるが，本研究では各シンボルの遅延の算術平均によって伝送システムを評価する．

図 1　本研究で想定した伝送システム

3　符号

本研究の説明に入る前に 3.1 節で従来研究 [2] の概要を説明し, 3.2 節と 3.3 節で本研究の提案法との差異を比較することで本研究について理解を深めたい.

3.1　従来研究

従来研究 [2] に遅延改善のための分節木と符号語に関する実験的考察と題した研究がある. 分節木とは情報源から入力された入力系列を可変長ブロックにそれぞれ区切っていき, そのブロックを木の構造で表現したものである. 分節木の葉に符号語を割り当てることで情報源系列を符号化することができるが, このとき葉に状態遷移するまで符号化は完了せず, これが遅延の原因となる.

情報源の到着レートが小さい場合, つまり符号器にシンボルがなかなか入力されない場合, 分節木を用いた符号化ではなかなか符号化が進まず遅延が生じる. その遅延は次のバッファ, 送信機へも影響を及ぼす. 送信機の送信レートが大きい場合, 送信機が使われていない時間が増える. 従来研究 [2] ではこれら 2 点を想定した場合の遅延改善について研究を行なっている.

これらに着目した遅延改善のために従来研究 [2] では, バッファ・送信機内の送信されるべき符号語が底を突くと, 送信機はフィードバック信号を送り, 送信機は符号器から強制的に符号語を出力させ符号語を補充するシステムを考えている. このようなシステムを実現するために重要となるのが符号器に内蔵された分節木の改良である. 従来研究 [2] では符号器の内部の分節木の中間ノードにも符号語を割り当てるという改良を加えた. この改良により符号器はフィードバック信号を受け取ると中間ノードにあっても符号語を出力できるようになった. 従来研究 [2] ではこのシステムを用いて実験を重ね, 遅延改善に繋がったことを確認している.

復号器では符号語と元あった分節されたブロックとの一対一の対応が記述された表が内蔵されている. この表に従い復号化され元あった入力系列として復号される. 中間ノードに符号語を割り当てた場合でも, 元あった入力系列に復号され情報誤りは生じない.

3.2　符号器のための分節木と符号語

説明を進める前に次のことを定義する．ある中間ノードを「親」と決めると, 共通の親をもつふたつのノードは「兄弟」である．本研究では従来研究 [2] と同様に遅延改善のために分節木上の中間ノードに符号語を割り当てる．本研究と従来研究 [2] では中間ノードへの符号語の割り当て方が異なっている．従来研究 [2] は分節木の中間ノードに接頭符号の中から選び割り当てている．それ対して、本研究では分節木の葉に語頭符号の中から符号語を割り当て, さらに兄弟間の符号語に着目しその両者に共通した語頭があれば、その語頭を両者から抜き出しそれをそのまま親のノードに割り当てる．兄弟は葉に限らないことから中間ノードにおいても提案法は適用できる．

このように符号語を割り当てると符号化法は次のとおりになる．分節木の状態が符号語の割り当てられている中間ノードにたどり着くと, その割り当てられた符号語をバッファ・送信機に送る．　状態が葉に辿りつくと葉に割り当てられた符号語をバッファ・送信機に送り，分節木の状態は初期状態に戻る．

従来法 [2] も提案法も，中間ノードに符号語が割り当てられる．しかし中間ノードに割り当てられた符号語が実際に送信されるのは，従来法ではフィードバックがあったときに限られていたのに対し，提案法では常に送信される．従来法の中間ノードの符号語は符号器の状態を早く伝えるために導入されたものであって，一意復号のためには冗長である．一方，提案法では中間ノードの符号語は一意復号のために必要である．したがって提案法ではフィードバックを必要とせず，符号器が自ら符号語を小出しに出力しているのである．

また, 従来研究 [2] と提案法とでは平均符号語長が異なっている．従来研究 [2] では符号語を新たに割り当てたのに対して, 提案法では本質的な符号語の数は変わっていないつまり符号語の合計長さはかわっていないという理由から従来研究 [2] に比べ提案法では平均符号長が小さくなる．このことにより遅延は改善されるはずである．

3.3　復号器のための分節木と符号語

従来研究 [2] の復号器では, 符号器が持っている分節木のノードと符号語の対応を復号表として用い，, 受け取った符号語に対応した情報源系列を出力するものだった．それに対し, 本研究では復号器として分節木を採用すると, 提案法を適用でき, 遅延の改善に繋がると考えられるので, 復号器側にも分節木を採用しさらに提案法を適用した．

実装の手順としてまず, 復号器は符号器から提案法適用前の符号語を確認する．そして, 受け取ったある 1 つの符号語の各シンボルを確認する．シンボルが 0 なら左に枝を伸ばし 1 なら右に枝を伸ばす．これを葉に到るまで繰り返す．葉に元あった情報源系列を割り当てる．そして,

次の符号語に移り, 同様に作業を行なう．これを確認したすべての符号語に対して行なう．すると, 分節木ができあがる．ここで, 提案法が適用され, 復号器ではこの提案法が適用された分節木が実装されている．

本研究では符号器の分節木と復号器の分節木とを区別するために, それぞれ「符号器のための分節木」と「復号器のための分節木」と呼ぶ．

4　実験と考察

提案法は，分節木の葉に割り当てられた符号語を中間のノードを含めて付け替えるアルゴリズムである．提案法の遅延に関する効果を定量的に評価するためにプログラムによるシミュレーション実験を行なった．プログラムのソースコードは付録に掲載する．

本論文の実験の目的は，提案法の適用前後の比較をすることである．提案法の適用後に遅延が最も小さくなるような分節木や符号語は何かという問題は，本論文では扱わない．したがって適用前の分節木や符号語の選び方には大きな意味はない．図 2 と図 3 はそれぞれ実験で使用した提案法適用前と適用後の分節木である．

次に本実験環境について記述する．図 1 の伝送モデルをプログラムを用いて再現した．情報源シンボルはポアソン過程で 100 万回到着させた．ポアソン過程の到着レートを 1[シンボル/秒], 送信機の送信レートを 1[bit/秒] と一定とした．情報源シンボル a, b の生起確率をそれぞれ p_a, p_b と表す．シンボルの生起確率 (p_a, p_b) を (0.5,0.5) , (0.6,0.4) , (0.7,0.3) , (0.8,0.2) と変えた．

次に実験結果と考察について報告する．

表 1 はシンボルの生起確率 (p_a, p_b) を (0.5,0.5) , (0.6,0.4) , (0.7,0.3) , (0.8,0.2) と変えたときの提案法の適用前と後での平均遅延時間と適用前に対する適用後の平均値延時間の改善率を表にまとめたものである．改善率は次の式で定義される．

$$\text{改善率} = \frac{\text{適用前 } [s] - \text{適用後 } [s]}{\text{適用前 } [s]}$$

提案法の適用後ではすべてのパターンにおいて遅延時間の改善がみられた．改善率でもシンボルの生起確率 (0.5,0.5) の場合, 0.20 と大幅な改善がみられた．

図 4〜図 7 はそれぞれシンボルの生起確率が (0.5,0.5) , (0.6,0.4) , (0.7,0.3) , (0.8,0.2) のときの提案法の適用前と後とで比較をした相対頻度図である．実線が提案法適用前を表し, 点線が提案法適用後を表している．どの図においても提案法適用前では遅延時間の大きい側に頻度が集中し, 提案法適用後では遅延時間の小さい側に頻度が集中していることが見てとれ, 遅延時間の改善が見られたことがわかる．

また, どの図においても小さい整数の上に突出した頻度があることが見てとれる．整数値の遅延が発生する原因は, サーバーが空状態時に符号語がサーバーに到着すると, 送信レートは

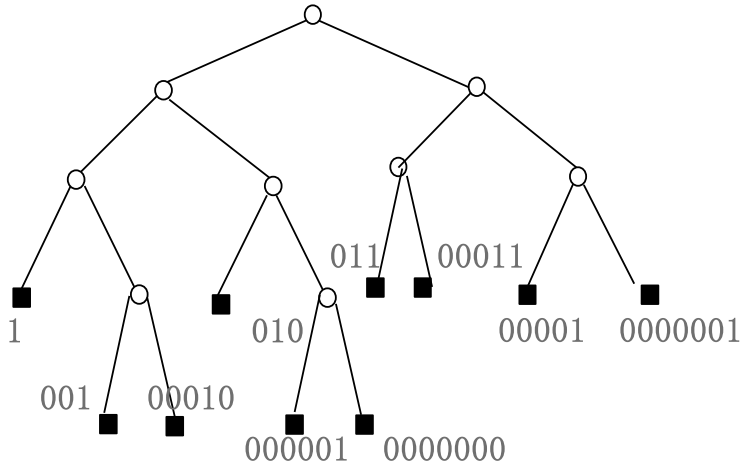

図 2　実験で使用した提案法適用前の分節木

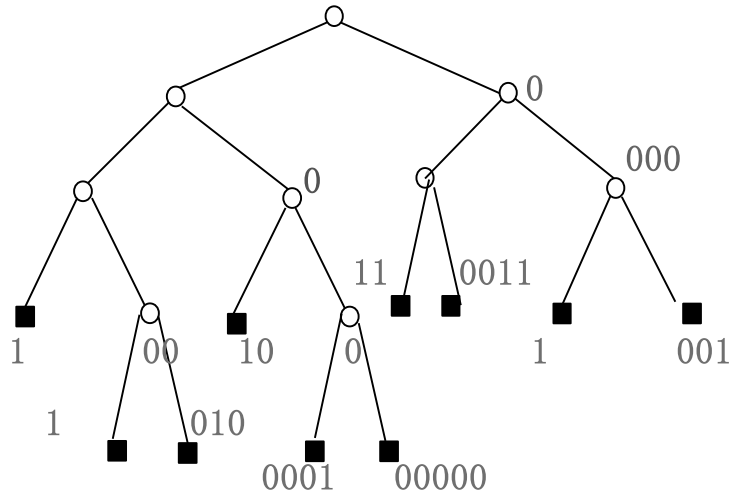

図 3　実験で使用した提案法適用後の分節木

1[bit/second] なので符号語のシンボル数だけの送信時間がかかることである. 加えて, この現象が頻発するため頻度が突出するという頻度図になるわけである.

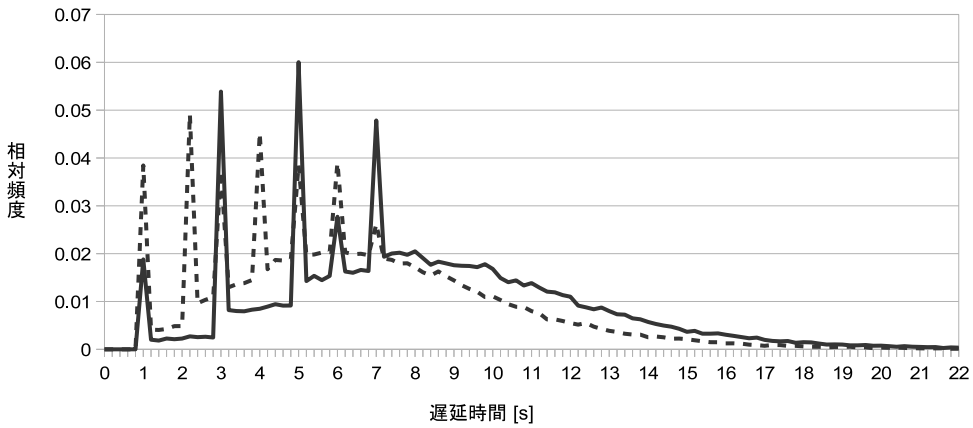

図 4　(a,b)=(0.5,0.5)

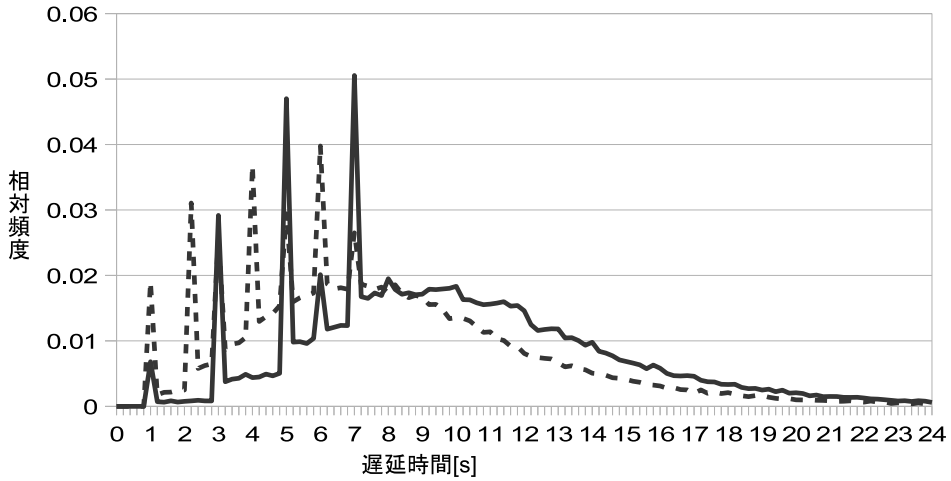

図 5　(a,b)=(0.6,0.4)

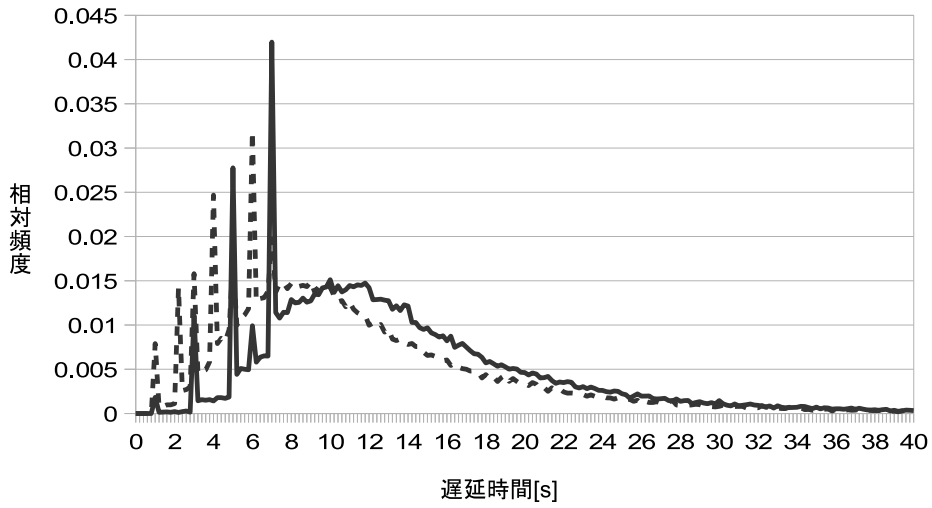

図 6　(a,b)=(0.7,0.3)

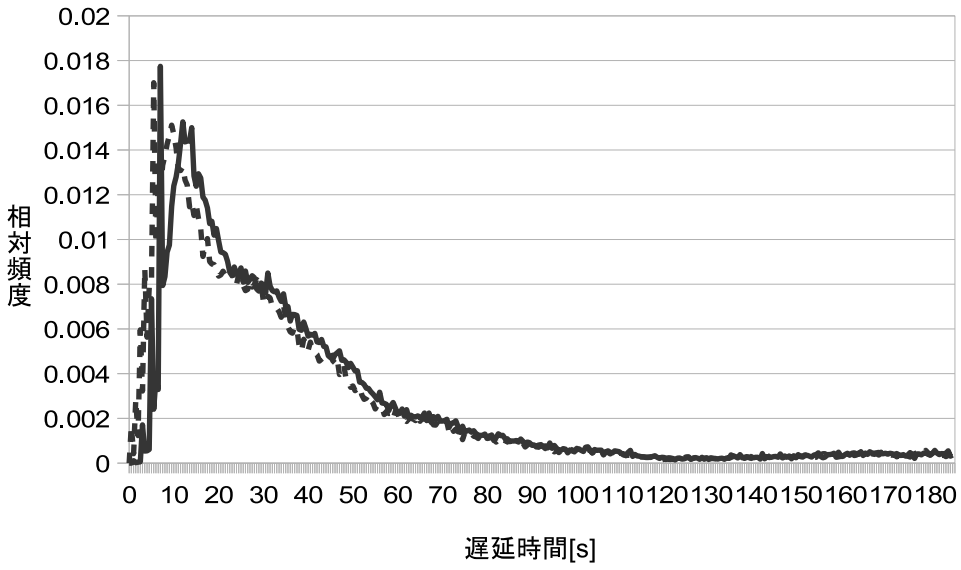

図 7　(a,b)=(0.8,0.2)

表 1　情報源生起確率を変化させたときの平均値延時間

情報源生起確率	適用前	適用後	改善率
(0.5,0.5)	8.21 [s]	6.53 [s]	0.20
(0.6,0.4)	10.08 [s]	8.14 [s]	0.19
(0.7,0.3)	13.92 [s]	11.73 [s]	0.16
(0.8,0.2)	36.53 [s]	34.11 [s]	0.07

5　まとめと今後の課題

本研究ではプログラムによるシミュレーション実験により提案法適用前と適用後との比較を行なった．その結果, 適用後では実験したすべてのパターンにおいて遅延の改善が見られた．さらに大きいものでは約二割の遅延削減につながったものもあった．また, 遅延における相対頻度図からも遅延の改善に繋がったことを確認した．

今後の課題として, 従来研究 [2] との比較が行なえるようにプログラムを構築し, それを用いて定量的に検討したい．

謝辞

本研究を終えるにあたり, 終始一貫して丁寧なご指導を賜わりました指導教員の西新幹彦先生に心より感謝の意を申し上げます．

参考文献

[1] 森田圭一,「ポアソン到着シンボルに対する伝送システムとその遅延特性」, 信州大学大学院工学系研究科, 修士論文, 2010.

[2] 泉陽一, 「遅延特性改善のための分節木と符号語に関する実験的考察」, 信州大学大学院工学系研究科, 修士論文, 2012.

[3] 西田俊夫，待ち行列の理論と応用，朝倉書店，1971.

[4] 杉原辰徳, 「算術符号を用いた伝送システムにおけるポアソン到着シンボルの遅延」, 信州大学工学部, 学士論文，2008.

付録 A　ポアソン過程

ポアソン到着とは，ランダムに到着するシンボルを表す確率過程のひとつである．ポアソン到着のシンボルの到着間隔は指数分布にしたがう．

ポアソン到着 [2][3] は，シンボルの到着が次の 3 つの条件を満たすと仮定した 1 つの到着の仕方のモデルである．到着間隔は指数分布に従う．ポアソン到着は以下の性質をもつ．

- 定常性
  $t > 0$，$x \leq 0$ なる任意の実数 t，任意の整数 x に対し，時間間隔 $(a,\ a+t)$ の間に x 人到着する確率は，全ての a について同一で，$v_x(t)$ と表すことができる．つまり，時間区間の幅が同じであれば，どこをとってもシステムの確率的状態は同じである．この性質を定常性という．

- 独立性
  上記に記した $v_x(t)$ は，時刻 a までにどれだけきたか，またいつ来たかには無関係である．もし前に到着したシンボルの時刻や個数によってこの確率 $v_x(t)$ が影響を受けるならば，前のシンボルはあとに残る影響，つまり残留効果をもつことになるが，そういうことがないというのが，この独立性である．

- 希少性
  $\psi(t) = \sum_{x=2}^{\infty} v_x(t)$ と置くとき，t が十分小さければ $\psi(t)$ は無視してよい．つまり，

  $$\psi(t) = o(t) \qquad (t \to 0)$$

  が成り立つ．この式は，

  $$\lim_{t \to 0} \frac{\psi(t)}{t} = 0$$

  を意味する．$o(t)$ とは t が十分小さければ無視してよい程度であることを示している．つまり $\psi(t) = o(t)$ は，2 つ以上のシンボルが同時に連れだってこないことを意味する．

付録 B　ソースコード

B.1　伝送実験のプログラム

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

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


#define MRND 1000000000L
static int jrand;
static long ia[56];

static void irn55(void)
{
    int i;
    long j;

    for(i=1;i<=24;i++){
        j=ia[i]-ia[i+31];
        if(j<0) j+=MRND;
        ia[i]=j;
    }
    for(i=25;i<=55;i++){
        j=ia[i]-ia[i-24];
        if(j<0) j+=MRND;
        ia[i]=j;
    }
}

void init_rnd(unsigned long seed)
{
    int i,ii;
    long k;

    ia[55]=seed;
    k=1;
    for(i=1;i<=54;i++){
        ii=(21*i)%55;
        ia[ii]=k;
        k=seed-k;
        if(k<0) k+=MRND;
        seed=ia[ii];
    }
    irn55(); irn55(); irn55();
    jrand=55;
}

long irnd(void)
{
    if(++jrand>55){irn55(); jrand=1;}
    return ia[jrand];
}

double rnd(void)
{
    return (1.0/MRND)*irnd();
}

double ramda_x = 0.5;
double rand_exp(void)
{
return (-(1/ramda_x)*log(1-(rnd())));
}

#define CODEWORDLEN 20

struct node {
    int ch[2];
    char word[CODEWORDLEN + 1];
};

int num_en_node;
int num_de_node;
```

```
struct node encode_tree[256];
struct node decode_tree[256];

#define BUFFLEN 4096
int
load_en_tree(FILE *fp, int n)
{
    int isleaf;
    int m;
    char buff[BUFFLEN];

    fgets(buff, BUFFLEN, fp);
    //printf("buff=%s", buff);
    sscanf(buff, "%d : %s\n" , &isleaf, encode_tree[n].word);

    switch (isleaf){
    case 0:
        m = n;
        encode_tree[m].ch[0] = n + 1;
        n = load_en_tree(fp, n + 1);
        encode_tree[m].ch[1] = n;
        n = load_en_tree(fp, n);
        break;
    case 1:
        encode_tree[n].ch[0] = -1;
        encode_tree[n].ch[1] = -1;
        n++;
        break;
    default:
        printf("[%d]", __LINE__);
        exit(1);
        break;
    }
    return(n);
}

void
sprout_de_tree(char *codeword, char *src)
{
    int i;
    int node;

    for (i = 0, node = 0; codeword[i] != '\0'; i++){
        if (codeword[i] != '0' && codeword[i] != '1'){
            printf("[%d] \n", __LINE__);
            exit(1);
        }
        if (decode_tree[node].ch[codeword[i] - '0'] < 0){
            node = decode_tree[node].ch[codeword[i] - '0'] = num_de_node++;
            decode_tree[node].ch[0] = -1;
            decode_tree[node].ch[1] = -1;
            decode_tree[node].word[0] = '\0';
        } else {
            node = decode_tree[node].ch[codeword[i] - '0'];
            if (decode_tree[node].word[0] != '\0'){
                printf("[%d]", __LINE__);
                exit(1);
            }
        }
    }
    strcpy(decode_tree[node].word, src);
    return;
}

void
make_de_tree_(int node)
{
    static char path[CODEWORDLEN + 1];
    static int idx = 0;
```

```
    if (encode_tree[node].word[0] == '\0'){
        idx++;
        path[idx - 1] = 'a';
        make_de_tree_(encode_tree[node].ch[0]);
        path[idx - 1] = 'b';
        make_de_tree_(encode_tree[node].ch[1]);
        idx--;
    } else {
        path[idx] = '\0';
        // printf("sprout(\"%s\", \"%s\");\n", encode_tree[node].word, path);
        sprout_de_tree(encode_tree[node].word, path);
    }
    return;
}

void
make_de_tree(void)
{
    num_de_node = 1;
    decode_tree[0].ch[0] = -1;
    decode_tree[0].ch[1] = -1;
    decode_tree[0].word[0] = '\0';
    make_de_tree_(0);
    return;
}

void
extract_prefix(struct node tree[], int i)
{
    int j, k;

    if (tree[i].ch[0] < 0 || tree[i].ch[1] < 0){
        return;
    }
    extract_prefix(tree, tree[i].ch[0]);
    extract_prefix(tree, tree[i].ch[1]);


    if (tree[tree[i].ch[0]].word[0] == '\0' || tree[tree[i].ch[1]].word[0] == '\0'){
        tree[i].word[0] = '\0';
        return;
    }
    for (j = 0; tree[tree[i].ch[0]].word[j] == tree[tree[i].ch[1]].word[j]; j++){
        tree[i].word[j] = tree[tree[i].ch[0]].word[j];
    }
    tree[i].word[j] = '\0';
    if (j > 0){
        for (k = 0; tree[tree[i].ch[0]].word[k + j] != '\0'; k++){
            tree[tree[i].ch[0]].word[k] = tree[tree[i].ch[0]].word[k + j];
        }
        tree[tree[i].ch[0]].word[k] = '\0';
        for (k = 0; tree[tree[i].ch[1]].word[k + j] != '\0'; k++){
            tree[tree[i].ch[1]].word[k] = tree[tree[i].ch[1]].word[k + j];
        }
        tree[tree[i].ch[1]].word[k] = '\0';
    }

    return;
}

void
extract_prefix_en(void)
{
    int i;

    for (i = 0; i <num_en_node; i++){
        if (encode_tree[encode_tree[i].ch[0]].word[0] != '\0' && encode_tree[encode_tree[i].ch[1]].word[0] != '\0'){
            int j, k;
```

```
            for (j = 0; encode_tree[encode_tree[i].ch[0]].word[j] == encode_tree[encode_tree[i].ch[1]].word[j]; j++){
                encode_tree[i].word[j] = encode_tree[encode_tree[i].ch[0]].word[j];
            }
            encode_tree[i].word[j] = '\0';
            if (j > 0){
                for (k = 0; encode_tree[encode_tree[i].ch[0]].word[k + j] != '\0'; k++){
                        encode_tree[encode_tree[i].ch[0]].word[k] = encode_tree[encode_tree[i].ch[0]].word[k + j];
                }
                encode_tree[encode_tree[i].ch[0]].word[k] = '\0';
                for (k = 0; encode_tree[encode_tree[i].ch[1]].word[k + j] != '\0'; k++){
                        encode_tree[encode_tree[i].ch[1]].word[k] = encode_tree[encode_tree[i].ch[1]].word[k + j];
                }
                encode_tree[encode_tree[i].ch[1]].word[k] = '\0';
            }
        }
    }
    return;
}

void
extract_prefix_de(void)
{
    int i;

    for (i = 0; i <num_de_node; i++){
        if (decode_tree[decode_tree[i].ch[0]].word[0] != '\0' && decode_tree[decode_tree[i].ch[1]].word[0] != '\0'){
            int j = 0, k;
            printf("[%d]", __LINE__);
            while (decode_tree[decode_tree[i].ch[0]].word[j] == decode_tree[decode_tree[i].ch[1]].word[j]){
                decode_tree[i].word[j] = decode_tree[decode_tree[i].ch[0]].word[j];
                j++;
                                printf("[%d]", __LINE__);
            }
            decode_tree[i].word[j] = '\0';
            if (j > 0){
                for (k = 0; decode_tree[decode_tree[i].ch[0]].word[k + j] != '\0'; k++){
                        decode_tree[decode_tree[i].ch[0]].word[k] = decode_tree[decode_tree[i].ch[0]].word[k + j];
                }
                decode_tree[decode_tree[i].ch[0]].word[k] = '\0';
                for (k = 0; decode_tree[decode_tree[i].ch[1]].word[k + j] != '\0'; k++){
                        decode_tree[decode_tree[i].ch[1]].word[k] = decode_tree[decode_tree[i].ch[1]].word[k + j];
                }
                decode_tree[decode_tree[i].ch[1]].word[k] = '\0';
            }
        }
        printf("num_de_node=%d 子 1=%d 子 2=%d 復号語=%s\n",
                        i, decode_tree[i].ch[0], decode_tree[i].ch[1], decode_tree[i].word);
    }
}



int symbol_queue_count = 0;
int symbol_queue_in = 0;
int symbol_queue_out = 0;
#define SYMBOL_QUEUE_MAX 1000000
double symbol_queue_time[SYMBOL_QUEUE_MAX];
char symbol_queue_symbol[SYMBOL_QUEUE_MAX];

void
put_symbol_queue(double time, char symbol)
{
    if (symbol_queue_count >= SYMBOL_QUEUE_MAX){
        printf("[%d]\n", __LINE__);
        exit(1);
    }
    symbol_queue_time[symbol_queue_in] = time;
    symbol_queue_symbol[symbol_queue_in] = symbol;
```

```
    symbol_queue_in = (symbol_queue_in + 1) % SYMBOL_QUEUE_MAX;
    symbol_queue_count++;

    return;
}

void
get_symbol_queue(double *time, char *symbol)
{
    if (symbol_queue_count == 0){
        printf("[%d]\n", __LINE__);
        exit(1);
    }
    *time = symbol_queue_time[symbol_queue_out];
    *symbol = symbol_queue_symbol[symbol_queue_out];
    symbol_queue_out = (symbol_queue_out + 1) % SYMBOL_QUEUE_MAX;
    symbol_queue_count --;
    return;
}

int server_queue_count = 0;
int server_queue_in = 0;
int server_queue_out = 0;
#define SERVER_QUEUE_MAX 1000000
char server_queue[SERVER_QUEUE_MAX];

void
put_server_queue(char symbol)
{
    if (server_queue_count >= SERVER_QUEUE_MAX){
        printf("[%d]\n", __LINE__);
        exit(1);
    }
    server_queue[server_queue_in] = symbol;
    server_queue_in = (server_queue_in + 1) % SERVER_QUEUE_MAX;
    server_queue_count++;

    return;
}

int
get_server_queue()
{
    char symbol;

    if (server_queue_count == 0){
        printf("[%d]\n", __LINE__);
        exit(1);
    }
    symbol = server_queue[server_queue_out];
    server_queue_out = (server_queue_out + 1) % SERVER_QUEUE_MAX;
    server_queue_count--;

    return (symbol);
}

double current_time;
double arrival_time;
double departure_time;
#define CUSTOMER_MAX 1000000
int customers = 0;
int arrival_mode = 0;        /* 0:通常の到着 1:余分な到着 2:到着停止 */


void
encode()
{
    static int current_node = 0;
    int next_node;
```

```
    int length;

    if (rnd() > 0.8){
        next_node = encode_tree[current_node].ch[0];
        put_symbol_queue(current_time, 'a');
    //  printf("arrive: a\n");
    } else {
        next_node = encode_tree[current_node].ch[1];
        put_symbol_queue(current_time, 'b');
        //printf("arrive: b\n");
    }

    length = strlen(encode_tree[next_node].word);
    if (length > 0){
        int i;
        if (server_queue_count == 0){
            departure_time = current_time + 1.0;
        }
        for (i = 0; i < length; i++){
            put_server_queue(encode_tree[next_node].word[i]);
        //  printf("queue: %c\n", encode_tree[next_node].word[i]);
        }
    }

    if (encode_tree[next_node].ch[0] == -1) {
        current_node = 0;
        if (arrival_mode == 1){
            arrival_mode = 2;
        }
    } else {
        current_node = next_node;
    }

    customers++;
    if (arrival_mode == 0 && customers >= CUSTOMER_MAX){
        arrival_mode = 1;
    }
    if (arrival_mode <= 1){
        arrival_time = current_time + rand_exp();
    }

    return;
}


void
decode(FILE *fo)
{
    static int current_node = 0;
    double a_time;
    char symbol;
    int next_node;
    int i;
    double n;

    if (get_server_queue() == '0'){
        next_node = decode_tree[current_node].ch[0];
        //printf("server: 0 (%d)\n", next_node);
    } else {
        next_node = decode_tree[current_node].ch[1];
        //printf("server: 1 (%d)\n", next_node);
    }

    for (i = 0; decode_tree[next_node].word[i] != '\0'; i++){
        get_symbol_queue(&a_time, &symbol);

        if(fo){
        fprintf(fo, "%f \n",current_time - a_time);
        } else {
```

```
            printf("%d\n", __LINE__);
            exit(1);
        }

        n = current_time - a_time;
        if (decode_tree[next_node].word[i] != symbol){
            printf("%d\n", __LINE__);
            exit(1);
        }
    }


    if (decode_tree[next_node].ch[0] == -1) {
        current_node = 0;
    } else {
        current_node = next_node;
    }

    if (server_queue_count > 0){
        departure_time = current_time + 1.0;
    }

    return;
}

int
main(void)
{
    FILE *fp, *fo;
    char filename[80];
    int i;

    init_rnd(2345);

    printf("ファイル名=");
    gets(filename);

    if ((fp = fopen(filename, "r")) == NULL) {
        printf ("ファイルをオープンできません.\n");
        exit(1);
    }

        if ((fo = fopen("seiki0802_prefix.ods", "a")) == NULL) {
        printf ("ファイルをオープンできません.\n");
        exit(1);
    }

    fscanf(fp, "%d ", &num_en_node);
    load_en_tree(fp, 0);
    fclose(fp);

    make_de_tree();
    extract_prefix(encode_tree, 0);
    extract_prefix(decode_tree, 0);


    for (i = 0; i < num_en_node; i++){
        printf("i=%d ch_0=%d ch_1=%d en_word=[%s]\n",
                                    i, encode_tree[i].ch[0], encode_tree[i].ch[1], encode_tree[i].word);
    }
    for (i = 0; i < num_de_node; i++){
        printf("i=%d ch_0=%d ch_1=%d de_word=[%s]\n",
                                     i, decode_tree[i].ch[0], decode_tree[i].ch[1], decode_tree[i].word);
    }


    while (arrival_mode <= 1 || server_queue_count > 0){
        if (arrival_mode <= 1 && ((server_queue_count == 0) || (arrival_time <= departure_time))){
            current_time = arrival_time;
```

```
            encode();
        } else {
            current_time = departure_time;
            decode(fo);
        }
    }
    printf("%d, %d\n",symbol_queue_count, server_queue_count);
    fclose(fo);

    return(0);
}
```