人の言を通す

通信ひとわたり

達哉ん

2018年6月執筆開始

はじめに

気づけば、C 言語の教科書を書いてから 5 年、問題集を書いてから 2 年が経とうとしている。”弟子”として 1 から教えた若人も手を離れ、弟子も他の知己も含め、時折尋ねに来た時に応ずるばかりとなった。今は、新たに学んだ日本茶でも淹れてほっと一息つきながら、最近はじめた落語に興じている日々である。

変わったのは何もプライベートばかりではない。時はすべてを連れて行くもので、仕事も住まいも知人も大きく変わった。それは、弟子や知己からの質問も例外ではない。ネットワークやシステムに関する質問が増えたのだ。仕事もまた然り。面白いことには、質問も仕事も、ネットワークは抽象化した理論をお話することが主で、システムは個々の具象を扱うことが多いというのが、共通していることである。

個々のシステムについては、各々をマニュアル的に記すことで満足な仕上がりが達せられる。DNS,LDAP,DHCP,POP,SMTP など各々のサービスの設定は言うに及ばず、ルータの設定コマンドなども、マニュアルで対応すれば良いところであろう。だが、それらの設定の意味を理解することまでは、マニュアルに求められまい。殊に、汎論の範疇と呼ぶべき部分までマニュアルに記載しているようでは、あまりに冗長となってしまう。また、ネットワーク一般の理論もマニュアルに記すべきものではない。そこで、これらネットワークの汎論を統べる役目の一冊を用意したいと考えた。周辺の方への説明の際などに便利でもあろうし、無論、自身の記憶を掘り起こす良い復習の契機にもなろう…こうして、この本を著すこととした。

新たに記すときに悩むのは、その性格付けである。コンピュータネットワークの世界には、大部の聖典もあればよく纏まった入門書もある。同じ方向性で書いたとしても、陳腐なものにしかなりはしない。いかなる性格付けにするか、その悩みに鍵をくれたのは幸いに私を慕ってくれる優秀で賢明なる若き友であった。彼らの根問の中で、私は文字に象られている通りの「人に言を通す」という通信こそが大切であると考えたのである。口伝から始まり現代ではインターネットで多くの通信が行われる。その中で、私は非常に多くの通信方法を経験し、また利用している。趣味で演る落語は口伝が基本だ。生で見る落語は、肉声か、あるいはマイクを通す程度である。一方、レコードや CD、DVD で落語を見ることもある。手紙を丁寧に書いて送るのは実に楽しいもので、時折私書が郵便箱にあるとそれだけでとても嬉しい。一方、友人とのやり取りは電子メールや SNS、チャットアプリなどが主力となっている。これだけ多く「人の言を伝える」ことに接する現代、まずは口伝、それから手紙と俯瞰し、そこからアナログ通信・デジタル通信へと発展させていくのはどうだろうか、と考えた。

といっても、演劇の声の出し方や手紙の書きかたなどを丁寧に書くものではない。落語の稽古風景や郵便制度の話を詳しく書くのは土俵が異なるだろう。そうではなく、各分野の経験を元に、「人に言を通す」ことを掬い出して話す、そんなテキストを執筆することとした。歴史に通信を学び、そこから電気通信はどうなっていったのか、どういう必要性で技術が生

まれたのか、経験から考察を加えて執筆していこうと思った。自身の学んだ電気通信を、経験に従って解釈しなおし、書き直す。地味であり、新規性も乏しいかも知れない。だが、何より話好きな自分の教え方と書くテキストに一貫性を入れるなら、言を通す、それを置いて他になかろう。オーソドックスかもしれないが、様々な分野の経験が何がしかのシナジーをおこし、説明に彩りを加えてくれればと思う。

本書の導入として、我々人類が物事を伝えるために用いた手段には如何なるものがあるのか、それは現代でどのように生きてあるいは滅びたのかという章を設けた。電気通信と程遠く見えるこれらの伝達には、しかし電気通信の要件を定めるような人類のアイディアが詰まっている。そのアイディアの最も元となる部分には「記録」があり、第 I 部ではその記録技術や信号処理技術についての最小限を記した。ここで記録された情報をまずは電気的に伝えることを考えるのが第 II 部である。ラジオ・電話・FAX と言った現代の「デジタル信号」からすると一代前の原理であるが、これを知ることは TCP/IP 世界にも関わる低レイヤー技術の理解に十分に役立つと考える。第 III 部からはいよいよ本丸、コンピュータ・ネットワークへと論を進める。階層化モデルに従った理解と代表的なプロトコルを、各々第 III 部・第 IV 部として本書の大部分を用いて解説する。付章的であるが、第 V 部として、冗長化・セキュリティ・トラブルシューティングなど、構築や管理といったより実践的な情報の基盤となる、最初の一歩を記述した。

各章の記述においては、多少の数学を要求する箇所もあるが、高校程度の数学が取り扱えれば読めることを基本とした。このため、前提知識の解説を随時【補足】と記して用意した。一方、厳密な取り扱いが本論にとっては不要であったり、あるいは難解である場合、その部分についてはまず厳密な取り扱いをせずに論を展開し、後の節で詳細を補うこととした。この詳述部分には【補遺】と記し、読み飛ばしても大局に影響しないよう心がけた。また、汎論に対する実例の紹介という意味付けで、各章で出来うる限り演習問題を出題し、巻末にその解答を記述した。汎論の解説ではあっても、手ずから具体例を解釈することは理解の深化に繋がると信ずる。

最後に、この本・これまでの本や、著者と付き合ってくれたこれまでの友人たち、付き合うこととなるこれからの読者たちにもお礼を申し上げたい。南天竺には赤梅檀という立派な木があり、その周には難莚草という草が蔓延ると聞く。この 2 つの植物は”有無相持ち”、赤梅檀の降ろす露が難莚草に双無き水となり、難莚草の盛衰は赤梅檀に栄養をもたらすという。友人と自身、読者と著者…携わる人々がそんな関係であることを祈り、この本を捧げる。

桂米朝師の「伝」に畏敬を払いながら
達哉ん

目 次

第 0 章

「伝える」歴史

現代で通信という言葉を聞いた時、連想される多くはインターネットを始めとした様々な電気通信だろう。だが、電気通信ができる以前より、人は様々な意思疎通…通信を行ってきた。本章では、電気通信に至るまでの人の意思疎通の営みを見ていく。これらの営みの中で、人間は通信に必要とされる要件を見出してきた。通信とは何か、何が必要なのか。まずは歴史からその要件を学び取っていこう。

0.1 声による伝達

動物には種によって様々な対話方法がある。蝙蝠なら超音波を使い、鳥ならば鳴き声で対話を行う。動物の鳴き声のモノマネで知られる江戸家小猫氏によれば、人間に最も近いゴリラなどは 10 種類の鳴き声により様々な対話を行うという。例えば、我々がゴリラの鳴き声と聞く「ウホウホ」という鳴き声は、特に喜ばしい時、幼いゴリラが使うことが多い表現なのだそうだ。我々人間にその差異はなかなかわかりづらいが、しかしながら種ごとの鳴き声による伝達は、我々人間の発話による伝達に同じことなのだという。

0.1.1 人間の発話

自らの声帯により何らかの音を出し、これに意味を持たせて伝達する。我々人間が知る多くの動物の鳴き声にはそれほど多くの種類の伝達はなく、概ね自らの感情の伝達が中心となっているようではあるが、人間の発話による伝達もそれと同様である。生まれてすぐの赤子は、泣くという動作を伴った発話により自身が不快であることを伝える。成長し、感情が分化する (図 0.1) につれ、何が不快なのかなどもその泣き方に入ってくるようで、親はその泣き方で状況を察することも少なくない。やがて、言葉を覚え話すようになると、それらの言葉の意味を経験から学ぶようになる。家族から広がるコミュニティの中で、共通した意味の言葉を使うようになり、人との伝達・コミュニケーションを始めていくのである。

コミュニティの違いによる差異こそあるものの、我々人間の”鳴き声”は言語を為し、その共通認識により対話、伝達が始まったのである。この言語は感情にとどまらず、具象抽象の垣根を超えて叙事や論証、または歌や言葉遊びと言った様々な知的活動の礎となった。動物の鳴き声による対話と対応した人間の会話は、最も原始的な伝達の方法であるものの、様々に技術が発達した今なおコミュニケーションの基礎を為している。通信の始まりは、このような”発話による伝達”であったといえよう。

0.1.2 発話の制約

しかし、単純な発声による発話では、声が届く範囲にしかその意味が届かない。この時の届く範囲とは、空間的な範囲もあれば時間的な範囲でもある。落語会が行われていた会場に、

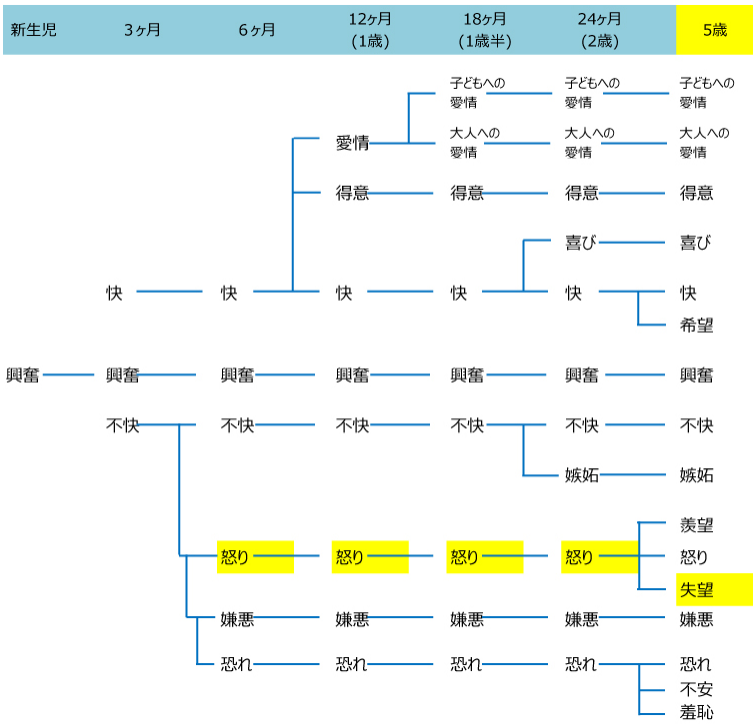


図 0.1: ブリッジスの情緒分化図 日本病児保育学会ブログ記事より引用

終わった後に入っても落語を聞くことはできないし、また、同じ時間に遠く離れた場所にいても落語を聞くことはできない。肉声の落語を聞くことができるのは、その時その場所で演者と同じ空気を共有していた者に限られる。逆に、その空気を共有している者の中では、選択的に伝えたり聞いたりするということも難しい。高座の上で演者が下座に何らかの指示を出す時、声を使うとすれば客席の幾人かにはどうしても聞こえてしまう。同じ会場で鼾をかいて眠る客がいたとして、これだけを聞かないというのもなかなか無理がある。

先の鼾の例のような雑音があった場合などは特にそうだが、発話によるコミュニケーションでは聞き取れない・追いつかないと言った問題も発生する。マンツーマンのゆっくりした対話であれば言い直してもらえばすむだけの話であるが、先の落語のような例では、そういうわけにも行かない。音自体が聞こえなかった、音は聞こえたけれども判別がつかなかったと言ったレベルから、言語的な問題…イントネーションの差異、語彙にない単語の使用、そもそも言語が異なる、勘違い…と言った知識や思考の問題まで、発話の時には伝達の誤りが起こりうる。

発話による伝達は、原始的であるがゆえに手軽ではあれど制約や問題も内包している。通信・伝達技術というのは、これらの制約や問題を如何に打破するのかという人間の飽くなき挑戦の成果であるとも言える。

0.1.3 口伝という方法

発話の制約を最も単純に外したのは**口伝**である。現在でも民話や民謡など、様々なものが口伝により伝えられているが、これは人間の記憶を介することによって時間や空間の制約を緩和したものといえよう。最も身近なところでは、伝言がそうであろう。もっとも、伝言ゲー

ムに見られる通り、伝達の誤りという問題は解決されているとは言いがたい。「JPCZ による擾乱の影響で山陰地方を中心に荒天となる」などと専門の用語が入った言を伝言したところで、途中に知らぬ人が入れば伝言がうまく行く可能性は低いだろう。また、伝言には意識的・無意識的な取捨選択もあり、伝達の誤りという観点ではむしろ増えているかもしれない。

しかしながら、人間以外の資源が必要ないこの口伝という方法は、最も初期から存在すると同時に、現代まで続いている伝達方法である。口伝の国内における最も古い例としては、稗田阿礼が挙げられる。古事記には彼 (彼女？) の誦により伝えられたものが筆録されている。その後の歴史にも、説法や講釈と言った例はほぼ口伝であったようだ。一方、現代で行われている口伝の例としては、先にも持ちだした落語を例に挙げたい。昭和の爆笑王、桂枝雀師は著書「枝雀とヨメはんと七人の弟子」の中で次のように書いておられる。

”一番最初は「ちょっとやるから聞いてなさい」と言って、一字一句口移しで教えることから出発いたします。やらせてみて、ちょっと違うとイントネーションから直す。それが段々お稽古やっていくうちに、五分なら五分、流れを教えるようになる。”

これは、平成のはじめ頃に書かれた文であるし、枝雀師は 99 年にお隠れになったから、いま他の噺家が同様のやり方をしていると言い切ることはできない。だが、筆者が、この枝雀師の三番弟子、文之助師に”つる”を教わった時は、ほとんどこの通りの教え方であった。ただ、カルチャースクールでのことであるから、録音し、自分で原稿に起こし、その上で直してもらいながらという指導であった。とはいえ、これも口伝には違いない。多くの伝達法がある中で、わざわざこの原始的な口伝という方法で伝承をするのはなぜだろうか。

それは、口伝によってしか伝えられない、空気や情感、発声、身振り手振りの細かな違いなどを伝えられるというところであろう。現代の技術をもってすれば、双方向で似たような指導を遠隔地間で行うことも可能であるように見える。しかし、それでも微妙な空気感や息の間といったものは伝えきれない。落語という芸能は高座の落語家と客席が一体となって作り上げる空間が生命であるから、なるほど、その部分を伝えるためには口伝を取るよりほかないのであろう。また、落語は時代により変化していくものであるから、口伝という形態での変化は逆に追い風となるのであろう。これは、落語の伝承が「ネタの原稿を伝える」のみでないことを示しているといえる。

現代の視点からすれば口伝は古臭く、非効率な手法に見えることも多い。だが、それを意図して取り込んでいる芸に学べる通り、口伝でこそ伝わることもある。通信・伝達技術の多くは、重要と考えにくいものや伝えづらいものを取捨選択しているのである。逆に、その捨てられたものを拾う価値があるシーンや、そこまでの効率を要求しないシーンがあるからこそ、一見古臭く見える通信・伝達の方法も現代に息づいているのである。

0.2 筆写による記録

口伝は人間の記憶を媒介にして時間や空間の制約を打ち破った。しかしながら、当然全ての人間が稗田阿礼のような記憶力を持つはずもないし、伝達も記憶も個人の状況に依存することとなる。これを打破するのは日本へと輸入された文字、そして筆記媒体であった。

0.2.1 文字と図画

文字とそれを記録する媒体は、ヒエログリフやパピルスと言った著名な例があるとおり古代エジプト文明の時代にまで遡る。楔形文字や漢字と言った様々な文字が青銅器時代に生まれたのである。日本へ文字が入ってくるのは弥生時代とされ、それからの後、8 世紀頃には先の稗田阿礼の言を元にした古事記 (図 0.2) や風土記、日本書紀といった書物が発行された。

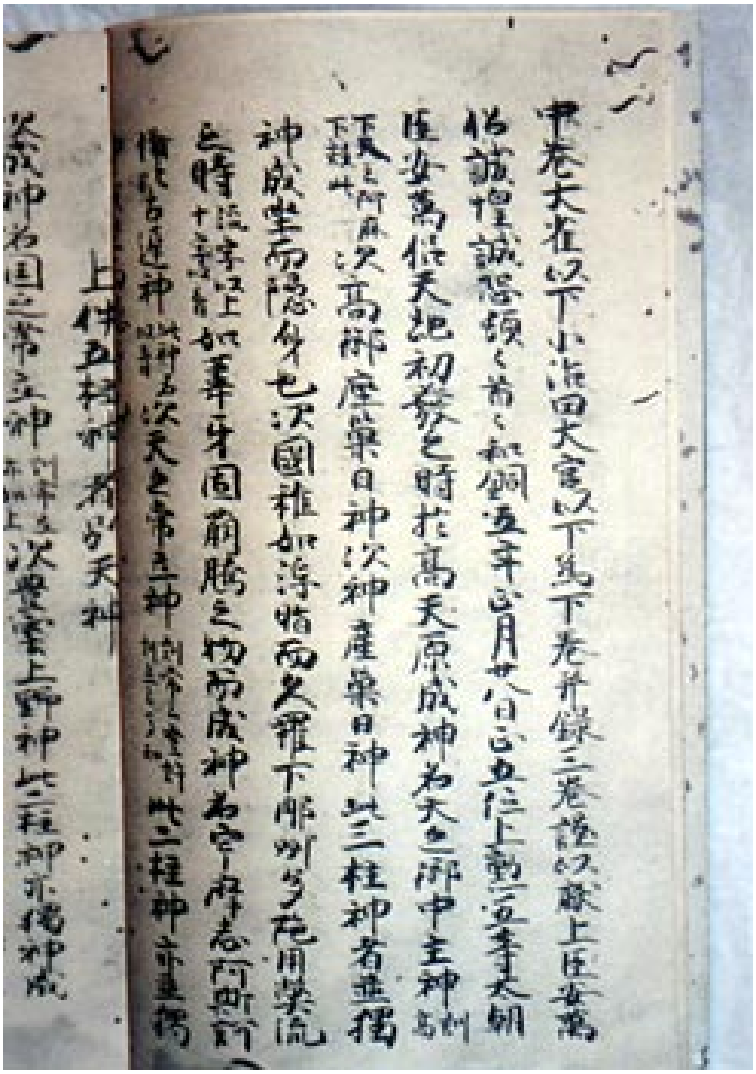


図 0.2: 真福寺収蔵の国宝・『古事記』 Wikipedia より引用

文字の輸入に伴い、離れた場所に文字を書いた媒体を送るという形式での伝送も行われるようになっていった。それらの書物の中には何百年という時を経てなお現代で読むことができるものも多い。これは、文字とその記録媒体が時間と空間の制約を打ち破ったということである。著名な数学者の書簡や日記が見つかることもあれば、伝わるつもり無く詠まれた”この世をば 我が世とぞ思う 望月の 欠けたることも なしと思えば”が現代にまで伝わっている (道長本人は記録に残さなかったが、その場に居合わせた人が日記に記したために現代まで伝わっている) など、文字は言語によって表現可能なものの時間的・空間的制約を打ち破ったのである。汚損や焼失といったリスクこそあるものの、口伝や発話の段階からすれば大きく進歩したと言えるのは明らかであろう。

また、これらの媒体は図画の伝達も可能とした。これは、発話や口伝で伝えられなかったものを伝えられるという利点をもたらした。紙媒体は、文字を媒介に発話を記録すると共に、図画として描けるものも記録する媒体なのである。

0.2.2 写本から複写へ

主として紙媒体に記述された文字による通信は、複写されて更に伝えられることにより、口伝やうわさ話と同様に広く伝わる手段となった。写本はその最も原始的な例である。紀元前3世紀、アレクサンドリア図書館では既に組織的な写本が行われていたとされる。別には、ギルガメシュ叙事詩もその頃の写本を最初とし、原本こそわからぬものの現代まで継がれている。日本においても多くの写本がある。特に源氏物語などは、既に写本が遠距離の伝達の役割を担っていたことが、菅原孝標女が更級日記に記述した例などからわかる。

先に述べたとおり、筆記にまつわる媒体には確かに汚損や焼失というリスクがある。これ以外にも、数十年・数百年という時の流れの中で退色するようなものもあれば(現代に流通しているボールペンインクの多くや、万年筆のインクなどは必ずしも長期保存に適するものばかりではない)、言語や文字が滅びて内容を理解できないというものもある。しかし、複写による流通は、これらのリスク解決に対し、”冗長性”という解決策を与えた。つまり、写本の1冊が残れば伝わる緒は紡ぐことができ、翻訳により別言語へと訳されていれば滅びた言語を用いずとも内容を理解することができる。

現代においてバックアップと呼ばれる手法も、原理としては複写と同等である。その観点から言えば、複写は多方向への同時期の通信・流通を可能にしたのみならず、その安全性において起源的なものといえる。紙と図画・文字による伝達は、複写によって通信の時間的・空間的制約を打ち破るとともにその冗長性などを（意識的かどうかは不明ではあるが）確保したのである。

無論、人間の手による複写は手間であるし、また誤りもある。そのため、短時間で多くの複写を行う印刷技術が発達した。8世紀には既に木版印刷の技術が生まれていたようで、その後も活版(凸版)印刷、銅版(凹版)印刷、平版印刷と多くの印刷技術が追随していくこととなる。広い意味では、印鑑などもその一例とも言え、承認の意を当人しか原版を持たない複写によって表しているとも考えられる。このように、複写・印刷の技術は、通信の幅を大きく広げた、その代表的なものと言える。

0.2.3 筆記媒体が捨てたもの

一方、筆記媒体・印刷媒体は視覚によって通信できないものを取捨選択している。音や匂い、手触りや味といったものを、筆記媒体・印刷媒体のみで完全に送ることは出来ない。勿論、これらが必ずしも必要だとは限らない。ただ、筆記媒体は、図画と言語によって伝えられるもののみを選択し、結果として他を捨てることで、通信に様々なメリットを生み出したのである。

具体的な例を見てみよう。先に、ゴリラには10種の鳴き声があると書いた。これらの鳴き声を、我々は正確に伝わるように書き分けられるだろうか。あるいは、その各々の声のシ

チュエーションを解説するとき、十二分な語彙・表現があると言い切れるだろうか。おそらく、この問に Yes と答えられる人は少数であろう。では、この鳴き声の差異が現実に必要となるような例はあるだろうか？その道の専門家でもない限り、これを文字で正確に伝えなければ困るというケースは考えにくい。十分な語彙・表現と図画を以って伝達するとき、筆記媒体が捨てた要素は確かに存在するのであるが、一方でその要素が必要なケースでなければ、筆記媒体による通信によりその利便性のみを享受できるのである。

0.3 遠距離の高速通信

筆記媒体によって破られた制約は多いが、しかし、筆記媒体にも難点があった。通じるのに時間がかかるという点である。

0.3.1 持ち運ぶことの難点

郵便碁という遊びを考える。現代において、遠く離れた場所にいる二人が囲碁を打ちたいとすれば、インターネット上のゲームサイトにアクセスするなどすれば、ほぼリアルタイムで遊ぶことができる。しかし、それがなかった明治の頃、ハガキに一手ずつ手を書いて送る”郵便碁”が生まれた。現代の郵便制度を持ってしても、1 局に 1 年から 2 年はかかる。通常は一度に 4 つの手を送り、4 局同時に進めたそうであるが、それでも非常に多くの時間がかかる。日本郵便碁愛好会に所属されているようなじっくり楽しみたい方はともかくとして、遠方の友人と会話がてら軽く打ちたいという時に気軽に楽しめるとは言いづらい。

郵便制度の発達の前には伝書鳩や飛脚により同様の伝達が行われていた。しかし、時間がかかるという点はいずれも変わらないようである。明石飛脚という噺では足に自身のある男が大阪から明石までの約 15 里 (現代で言うと約 59km 相当) の道のりを一日かけて片道だったという。無論、交通手段が未発達であったという時代背景もあろうが、現代でも郵便は大抵遅い手段として知られる[1]。

0.3.2 視覚による通信

マラトンの戦いの例がもっとも古いものとして知られるが、戦況を知らせるような伝令は内容こそ多くなくとも速度が要求される。そこで、視覚的に伝えられるものに何らかの意味を定めておき、遠距離でも素早く通じるようにした。狼煙や手旗信号 (図 0.3) がそれである。現代でも、野球でベンチから監督が出すサインはこれに当たるだろう。また、筆記媒体も介していることとなるが、アドバルーンもその一例と言える。

これら視覚による通信は視線を確保することが要求されるが、通常人間が見える範囲においてはタイムラグなしに通信を行うことができる。また、中継も比較的容易い。手旗信号や筆記媒体など、細部を見る必要がある手段だと距離こそ短くなるものの、伝達にかかる速度自体は高速で、音では届かない距離でも伝達できることには変わりない。一方、その意味付けの少なさから、多くの情報を伝えるには不向きである。

[1] これは非常に特殊な例であるが、10TB 容量の HDD400 台を詰めた箱を 2 日かけて車などで運べば、185Gbps という驚異的な速度になる。読み書きの時間は無視しているし、それだけ大容量のデータを運ぶ必要もないだろうが、データ量が著しく多い場合は郵便が最速になることも考えられる例としてここに記した。

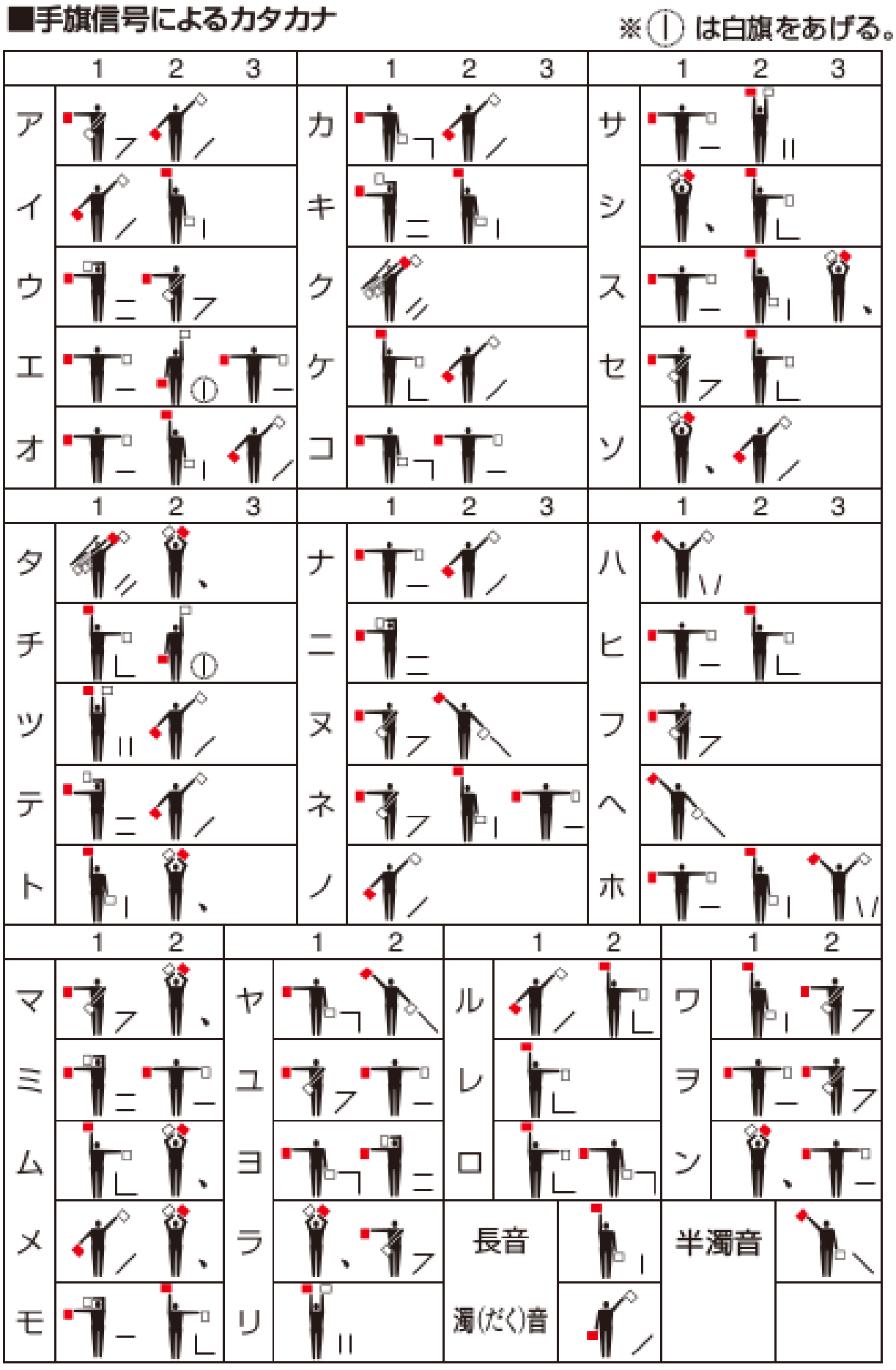


図 0.3: カタカナの手旗信号 海上自衛隊・海の手帳より引用

0.3.3 視覚の通信が捨てたもの

視覚による通信は、速度を重視したがために、伝えられるものが非常に限定的なものとなった。手旗信号や狼煙であれば、前もって決められた意味付けの範疇に属するものしか伝えられないのである。また、狼煙は多くの場合一方向的でもあった。逆に、双方向的にも可能な手旗信号は、音やものに比べ受信者が気づきづらいか、あるいは受信者を常にさく必要があった。

インターネット技術の標準化を推進する任意団体**IETF**(Internet Engineering Task Force)は、その標準技術仕様等を RFC(Request For Comments) という形で公開しているが、この中の RFC4824 では、手旗信号を用いた IP データグラムの転送を記している。これは**ジョーク RFC** として出された仕様ではあるが、手旗信号について（電気通信を元に）実に愉快な考察がなされている。以下、例を挙げてみよう。

- 送信・受信に各 1 名以上ずつの、2 名以上の組によりひとつのインターフェイス (送受信設備：この場合は人) がなされる。
- 通信速度は送受信者の経験に依存する。
- なりすまし防止の為の自己紹介が推奨される。
- タイムアウト、通信が時間切れとなる感覚はデフォルトで 60 秒で、手を休めることができる程度になっている。
- 転送量が多く長時間の通信となる場合は代替のインターフェイスを用意する必要がある。
- 時間とともに転送速度 (帯域幅) が変わり、概ね減少していく傾向にある。
- リンク内に別のインターフェイスがあった場合受信できるため、セキュリティ的に安全ではない。
- インターフェイスが送受信したデータを記憶する必要があり、生存時間が定まっていない。
- インターフェイスを通過したデータであっても、その記憶に残っていることがある。

これらの点は、現代の電気通信からすればその多くが特異なものであり、実用上はほぼデメリットと捉えられるものであろう。換言すれば、これほどまでに多くのものを犠牲にしてでも、何よりも速度が必要になる時、初めて手旗信号など視覚の通信に頼る必要が出てくるのである。

0.4 通信の秘匿

先に書いた藤原道長の歌は、本人の意図によらず後世へと伝わったものである。この歌であれば（元々が酒席等の話であったようなので）笑い話で済むが、意図しないところに伝わるのは望ましくないことも多い。対面の方法であれば、本人であることを認識するか、ある

いは初対面であれば本人確認をするなどの方法で当人同士とし、他に誰もいない場所で (音漏れがなく、盗聴器なども仕掛けられていない前提であるが) 話すなどの方法がある。だが、遠距離の場合、送受信者双方の目から離れる時間が必ず存在する。その時に、秘密を守るため、やはり様々な手法が用いられてきた。

0.4.1 未開封証明

中世のヨーロッパで主に用いられていた方法が未開封証明、封蝋 (シーリングワックス, 図 0.4) である。蝋を垂らし、多くの場合は家紋などを象った、当人オリジナルの封印を施して郵送した。洋形封筒の場合、このシーリングワックスを破損することなく開封して再度元のとおりに戻すのは至難の業である。したがって、全きシーリングワックスの未開封の書状が届くことは、途中で第三者に読まれあるいは改ざんされていないことの証明となった。

図 0.4: 筆者の封蝋の例

だが、これはあくまでも手紙が届いた場合の話にしか過ぎない。その内容を読むことを目的とする場合、書状を手に入れてしまえば内容は平文であるため、易々と読むことができる。本来の受信者に渡す必要さえなければ、そのまま内々に破棄してしまうことで受信者には何もわからない。これは、現代の郵便制度でも同様であり、心ない郵便局員が一般郵便を持ち帰り、内容を読んだり届けなかったりしたという事件もあった[2]。

0.4.2 暗号化

そこで、途中で奪われたり盗聴されたりしてもその内容を把握できないよう、**暗号**が用いられるようになった。

古来の暗号として特に有名なのは**シーザー暗号**である。古代ローマの軍人、ガイウス・ユリウス・カエサル（英語読みでジュリアス・シーザーとなるため、これにちなんでシーザー暗号と呼ぶ) は自身が秘匿としたい文を送るとき、それぞれの文字を”シフト”して送信したという。例えば、a を c に、d を f に、x を z に、y を a に、z を b にと、それぞれ（サイク

[2]少なくとも日本においてはこれが事件として報道される程度に珍しいことであるので、郵便が信用ならないと言っているのではない。寧ろ日本の郵便送達率は 9 割を超え、世界に冠たる信頼の郵便制度と自負すべきところである。

リックに見て）2 文字後のアルファベットへと変えて送信したのである。この規則を知らない（また気づかない）者には、内容を把握することは出来ない。当時の教育水準や、カエサルに敵対していた者の教養程度からして、これは効果的な手法であったと思われる。

一方、暗号化がうまく行かなかった例もある。世界大戦中の日本軍は、早口の薩摩弁で伝令を行うことにより、当地を故郷とする者にしか内容がわからないよう（即席の、ある種の暗号として）秘匿していたという。ただ、これは敵軍に捉えられた捕虜に読ませることにより、本来読まれるべきでない敵軍にも内容が伝わってしまったようである。

このように、奪われないこと・途中で第三者が読まないことを必ずしも期待できない状況においては、難読・容易に内容を把握させないようにすることのほうが重要となっていった。今日の電気信号においても、暗号化は非常に重要な役割を果たしている。

0.5 記録・通信媒体の増加

ここまでの例を見てもわかる通り、記録媒体や通信媒体は時代とともに増加してきた。個別にその例を全て論ずることは出来ないが、いくつかを取り上げてみよう。

0.5.1 増加した媒体

先までの例の後に増えた媒体は、特に光を用いたものが多い。

写真 (ここで言うのはアナログ写真・銀塩写真) は、受光によって化学変化を起こす物質を用い、受光状態を保存する媒体である。実際には、受光の記録とその定着に当たる現像の工程を経ることとなるが、絵よりも手軽かつ多くの場合精密にその光景を記録した。これを連続的に撮影したものが初期の動画である。だが、この方法では絵のみしか送ることが出来ない。そこで、その内容は別の台本にするなどし、活弁士と呼ばれる人々が画に合わせて内容を語っていたという。これが今日で言う映画の原形、活動写真 (無声映画) である（同じものを絵に変えた紙芝居はそれより以前からあった。)。

レコードは盤面に音=振動に応じた溝を掘ることにより録音を可能にした。これが音をそのまま伝達できる最初の記憶媒体であった。これもまた先の写真と同じく、その状況—ここでは音—をそのままに記録するという手法であった。このアナログな録音方法は、本稿を書いている 2018 年現在、再び売れ行き 100 万枚に返り咲き、およそ 30 年ぶりに生産が再開される会社も出てくるなど、見直しが進んできている。その主たる論調は、ここまで学んできた事実「通信は何かを取捨選択して他の制約を破ってきた」という中での”取捨選択”の部分を捨てるべきでないというものである。デジタルの音とアナログの音は違う。同様にデジタルの写真とアナログの写真は違う。実用の観点からは捨てられた部分だったかもしれないが、それを価値として見出すことによって、これらアナログの手法は今日でも利用されていると言える。

記録媒体を上げてきたが、通信媒体で言うとモールス信号が挙げられる。これは 2 つの長さの音あるいは光によって構成され、いくつかの音によって文字を伝達する手法である。ここまでであれば手旗信号と変わらないように思えるが、光源を用いたモールス信号は手旗信

号の使えない闇夜でも利用できるし、人にかかる負担もそれほどのものではなかったようである。

このように、記録媒体や通信媒体が発達し、多くのやり取りができるようになったのが、概ね 1800 年台から 1900 年台であった。その頃、人々はこれまでの火や水に加え、電気の利用を知る。多くのものが電気に取って代わられる中で、通信もまたその波に乗った。この同じ時代は電気通信の幕開けでもあった。

0.5.2 そして電気通信へ

次章からいよいよ本題の電気通信であるが、歴史に学んだことを振り返っておこう。

口伝えから始まった通信は、以下のような様々な要求の度合いに応じて伝達事項を取捨選択し、種々の発展を見せていった。

- 共通認識による正確な伝達 (使う言葉の認識や狼煙の意味付け)
- 時間や空間を超えた伝達 (様々な媒体による記録)
- 通信の速度への要求 (光を用いた通信)
- 同時の複数の通信 (印刷技術や郵便碁の多重化など)
- 伝達に気づくということ (手旗信号で捨てたもの)
- 秘匿性 (封蝋やシーザー暗号)
- 冗長性 (記憶に頼らないということ、印刷技術)

電気を用いた通信では、これらの考えや要件をどのように取り込み、実現しているのであろうか。

電気通信の技術も、ここまでに学んだ通信と同様多岐にわたるものである。だが、通信とは字に書いたとおり、人の言を通じさせることであり、電気を用いたとしてもこの基本に変わりはない。個々の技術の理解の根底に、これら「伝える」要件を見出すことで、技術が活きるのだと信じる。電気通信は現代に発展した技術であり、あまり枯れているようには見えないかもしれないが、人類の経験と歴史に基づいた大いなる叡智でもある。このことを常に意識し、これから電気通信のお話にお付き合い願う次第である。

演習問題

1 歴史的なシーザー暗号では、アルファベットを 3 文字後ろにずらして (a,d,x,y,z をそれぞれ d,g,a,b,c に変えるなど) 暗号化していた。1024 文字以下 1 行の半角文字列が入力されるとき、それをシーザー暗号化・復号化するプログラムを作成せよ。

2 糸電話もひとつの通信の方法である。糸電話の通信にはどのような特徴があり、何を取捨選択したのか、本章の他の方法を見た時と同様に考察してみよ。

第Ⅰ部

情報の記録

第 1 章

デジタルデータの記録

電気を用いない通信から見てきた通り、通信と記録は密接な関係にある。また、記録の際には記録するべき要素の取捨選択が行われる。人間が直接運ぶにせよ光や音を使うにせよ電気通信にせよ、記録されたものを受け渡しするのが通信であるということに立ち戻れば、記録と通信は表裏一体の関係にあることは明らかであろう。

本章では、離散的な「デジタルデータ」をいかにして表現・記録するかという点について学んでいく。

1.1 二元状態とデジタルデータの特徴

伝言ゲームでは、なかなか情報が正確に伝わらない。この例を見てもわかる通り、通信は記録と伝達手段 (伝送路) の双方において正確・確実であり、途中で内容が変化しないことが求められる。

1.1.1 記録の条件と二元状態

記録、すなわち表現において正確性や確実性を担保するためには、次のような条件が必要となろう。

- いくつかの状態に明確に分かれることあるいは定量的であること。
- 内容によらず表現方法が統一されていること。
- 伝送や処理に適した、単純な形態であること。

この条件を満たす最も端的な表現として、**二元状態**：明確な二つの状態でそれ以外の状態のないもの があり、実際のデータ伝送において広く用いられている。通信において使われる二元状態は、2 進数 (0/1) のほか、論理 (T/F)、極性 (+-)、電流の有無、スイッチの on/off などが代表的だが、センサの通信などにも目を向ければ動静 (動きの有無)、音声 (声の有無) なども通信に使われている二元状態といえよう。この二元状態が通信に使われる理由としては、以下のようなものが挙げられる。

二元状態の通信における利点

- はっきりと状態の区別がつき、検出も容易で、誤りも生じにくい
- 多くの電気的状態に一致し、情報の伝送や記憶に有効である
- 数字の 0/1 に対応させて 2 進数表現することで四則・ビット演算を適用できる

1.1.2 デジタルデータの特徴

デジタルとはよく聞く言葉であるが、この訳語「離散量」あるいは「計数」はそれほど馴染みがない言葉かもしれない。デジタルというのは幾つかの、飛び飛びの状態を取る量という意味であり、対してアナログは連続的に値が定まる。例えば電車の線路が始点からどれだけの距離にあるかはアナログであるが、現在居る駅という情報はデジタルと言える。針が連続に動くアナログ時計は（現実に読み取れるか、その精度はどうかというのは別にして）10.232 秒など小数点以下の秒数も表しているが、デジタル時計では数字として表示される桁数以上の情報は捨てられている。

このように、デジタルデータは離散的な有限個の状態に最初からデータが分かれているため、各々に 2 進数値を対応させることで二元状態の羅列によりデータを表現できる。また、状態が有限個であり、2 進数値に対応付けるときに誤差が生じることがない。この例としては、ある文字を音声と書き文字の双方で伝える場合を考えてみよう。文字を音声で伝える際、例えば”し”と”ひ”がごく近い発音だとどちらの文字かひどく伝わりづらいことがある（実際に、江戸言葉や上方言葉ではこれらが混同した発音あるいは入れ替わることがある）。このため、「執事」と「羊」や「しなびた」と「ひなびた」など、”し”と”ひ”しか違いのない 2 つの言葉は時として文脈などから推定せざるを得ないこととなる。だが、書いている文字であればどちらを指しているのかは一目瞭然であり誤解の恐れはない。ここで挙げたのは極端な例であるが、デジタルデータは最初から状態が明確に分かれていることがアドバンテージとなっているということである。

1.1.3 【補足】2 進数・記数法について

> 本節は前提知識の補足である。
> 既知の読者におかれては飛ばして次節を読まれたい。

先の解説で 2 進数を用いたが、記数法について慣れない読者に向け、補足解説を記す。

我々が物を数える際には手の 10 本の指を用いる。ここから物を数える位取りとして、指がいっぱいになったら位を 1 増やすということで **10 進法**が生まれた。10 進法とは、10(十、ten) 毎に位がひとつ上がるような数え方であり、我々が普段使っている数の表記法である。10 進法により表された数のことを **10 進数**と呼ぶ。

一方、先の二元状態は 2 つの指を持つといえる。そこで、2 毎に位がひとつ上がり、0,1,10,11,... と数える **2 進法**を使う。

これらのような、数の表記にあたりどのようにして位をとるかの方法を**位取り記数法** (positional notation) あるいは単に**記数法**と呼ぶ。また、その位取りの基準となる数 (10 進法なら 10、2 進法なら 2) を**基数**と呼ぶ。

記数法の一般論

先までの例と同様に、指の本数から考えよう。指が $n(\geq 2)$ 本しかないと仮定する。このときは、n 本がいっぱいになったら次の位に移る。たとえば、指が 3 本しかなければ、3 になったら次の位に移るようにして数えることができる。

一般の自然数 n に対して、n 進法とは、n を基数にする方法である。つまり、数えていって、n に達する毎に上の位に移るという事である。

例えば、123 という数字を考えよう。これは、123 個の 1 であるが、同時に 12 個の 10 と 3 個の 1 である。更に分ければ、1 個の 100 と 2 個の 10 と 3 個の 1 と分かれる。つまり、10^k の束がいくつあるかを数え、それが 10 個に達したら 10^{k+1} を 1 増やす。これによって、各位は 10^{k+1} の束にならない余りとなる。

上記に従って位取りをしてみよう。10 進法では、小数点の直上を 10^0 とし、その上の桁を $10^1, 10^2, \cdots$、小数点以下を $10^{-1}, 10^{-2}, \cdots$ としている。この 10 を n に変えたのが n 進法である。たとえば、2 進法における 11111 は $2^4 \times 1 + \cdots + 2^0 \times 1 = 31$ である。

n 進法で使われる数字は、n 種類ある。2 進法なら 0 と 1 だし、10 進法なら 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 であり、16 進法の場合は 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, a, b, c, d, e, f である[1]。数字だけでは何進法であるかわからない場合もあるため、これを明確にするときには、数字の右下に $33_{(n)}$ のように記す。たとえば、$101.1_{(2)} = 5.5_{(10)}$ である。

記数法の変換

n 進法から 10 進法への変換はどのように行うだろうか。これは n 進法の仕組みを考えてみればわかる。

n^k の位が a_k であるような数字 A の 10 進法での値は定義から次のように求められる。

n 進法から 10 進法への変換

n^k の位が a_k であるような数字 $A_{(n)}$ は、10 進法において

$$A_{(10)} = \sum_k a_k \times n^k$$

と計算できる。

逆に、10 進法から n 進法に変える場合は、どうすればよいだろうか。

1 つには、単に「取り尽くす」方法がある。つまり、n^k のうち、元の数を超えない最大のものを見つけ、その数で元の数を割った商を n^k の位にし、余りについて n^{k-1} で割り…と繰り返すのである。特に、小数点がある場合はこの方法を利用すると楽である。

[1] アルファベットは大文字を用いることもある。

だが、この計算はやや面倒に感じることもあろう。そこで、主に整数向けであるが、先に書いた位取りの考えを用いる方法がある。

n^k の位が a_k であるという事は、n^k で割った後に、小数点以下を切り捨て、n に対する剰余を取ると a_k になる、という事である。例えば、10 進法で 100 の位の数字を出したいとすれば、100 で割って 10 による余りを取れば良い。これと同じ性質が一般の n 進法にも成立する。この性質を利用し、次のような手順を踏んで記数法を変換できる。

10 進法から n 進法への変換

1. 変換したい 10 進数を n で割り、その余りを別におく。
2. これを、商が 0 になるまで繰り返す。
3. 別においた余りを逆順に並べると n 進数での表記になる。

いくつかの数を変換してみればわかるが、10 進整数は他の記数法でも整数である。一方、ある記数法で有限小数であるからと言って他の記数法で有限小数であるとは限らない。例えば、1/3 は、10 進法では無限小数だが、3 進法であれば $0.1_{(3)}$ とシンプルに表せる。

1.2 情報の符号化

先にデジタルデータを 2 進数化して表現することを論じた。情報内容を変えることなく、この 2 進数化のように用途 (記録・通信 etc...) に応じて変換を加えることを**符号化**あるいは**エンコード**という。必ずしも 1/0 で表すわけではなく、文字を手旗信号で表すのも符号化の一例といえる。

ここでは整数や文字と言った典型的なデジタルデータの 2 進数による表現方法 (=符号化) を説明する。また、やや歴史的な符号化の例としてモールス符号を、デジタルでアナログを近似している例として小数の表現方法を紹介する。

なお、これ以降の議論については、二元状態として **bit**(binary digit の略で二進数の意) を用いるものとし、8bit である場合の表現として 1**byte(バイト)** あるいは（歴史的経緯により）特に 8bit であることを強調したい際には 1**octet(オクテット)** を用いる[2]。

1.2.1 整数の表現

2 進数での表現として最も単純なのは符号なし整数であろう。符号なしの整数をそのまま 2 進数に変換し、これをビットパターンとする方法である。Nbit により、0 以上 2^N 未満の整数が表現される。

[2]2008 年まで、1byte=8bit であることは主流でこそあったものの明確な定義として定まっておらず、1byte が 8bit でない環境も存在していた。2008 年に ISO（国際標準化機構）や IEC（国際電気標準会議）において 1byte=1octet が正式に定まったため、現在では 1byte=8bit として考えて問題はないが、追随できていない環境などでは 1byte が 8bit でない可能性もありうる。このことから、本書においては 8bit であることが特に重要な場合には旧来の 1octet という表記を用いることとした。

一方、符号付き整数を扱う場合にはどうだろうか。例えば、先頭の 1 ビットを符号に割り当てるなどの方法が考えられる。だが、この方法では+0 と-0 が出る、符号の扱いの都合で CPU の実装等が複雑になるなどの難点がある。これらの理由の回避のため、近年の電子計算機では **2 の補数**表現が用いられることが多い。

この表現は、次のようなものである。

まず、0 を基準とする。10 進法における 0 を、2 進法における 0 と対応させるのは自然な考えであろう。さて、これから 1 を引いた数、すなわち-1 をどう表すべきであろうか ??

4 ビットで考えることとしよう。1=0001 から 1 を引くと 0=0000 になる。では、それより 1 小さいものはどうすればいいか。これは、1111 とすれば良いのである。ビットがあふれる部分への繰り上がりあるいはその部分からの繰り下がりを無視して考えれば、0000-1=1111 であるし、足し算で考えても 1111+0001=0000, 0000+0001=0001 となる。また、嬉しいことに、この方法を用いた場合でも、先頭ビット 0 は+または符号なしを、1 は-を示す。このように、-a のビット表現を、0 から a を引いたものとして表現するのが 2 の補数表現である。

参考として、表 1.1 に、2 の補数表現を用いた 4 ビットの場合の符号なし整数・符号あり整数一覧を示す。

表 1.1: 4bit 符号なし整数・符号あり整数の一覧

ビット列	符号なし整数	符号あり整数	ビット列	符号なし整数	符号あり整数
0000	0	0	1000	8	-8
0001	1	1	1001	9	-7
0010	2	2	1010	10	-6
0011	3	3	1011	11	-5
0100	4	4	1100	12	-4
0101	5	5	1101	13	-3
0110	6	6	1110	14	-2
0111	7	7	1111	15	-1

この例を見てもわかるように、(桁溢れの繰り上がりを無視すれば)2 進数・10 進数とも順に並んでいるのが 2 の補数表現の特徴である。また、符号を用いた表現と違い 0 が 1 つになっており、-8 から 7 までと、負の側は絶対値が 1 大きい値まで収まる。

【補遺】補数の意味

本節は大筋に影響しない。
難解あるいは興味索然たるものと感じる折には
飛ばして次節を読まれたい。

先の解説では 2 の補数表現について補数とは何か説明せずに仕組みのみを述べた。本節では、関心ある読者諸賢に向け、**補数**の意味について解説する。

ある基数bにおける自然数aの補数とは、aに足した時に桁が1増えるような最小の自然数cのことをいう[3]。10進法における74に対する補数は26である。2進数において$1100_{(2)} = 12_{(10)}$の補数は$100_{(2)} = 4_{(10)}$である。

先に出てきた2の補数表現とは、2進数におけるaの補数を$-a$として使う$(a > 0)$という考えである。但し、一般の補数では直上の1桁が増えるかどうか見るところ、この表現においては扱うサイズで桁溢れが起きるかどうかにより判断をする。換言すれば、「符号なし整数の最大値+1からaを引いた値」といえる。先に挙げた4bitにおける5(0101)の補数は、4bitが桁溢れを起こすような最小の自然数、つまり16(10000)-5(0101)=11(1011)である。そこで、1011という表現を-5の表現とする。同じように、aの補数の符号なし整数表現(2進数表現)を$-a$として使う$(a > 0)$というのが、2の補数表現である。

1.2.2 文字の表現

デジタルデータとして文字を表現する場合には、それぞれの2進数値に対応した文字を定める。これを**文字コード**と呼ぶ。例えば、最も一般的な文字コードと言えるASCII(American Standard Code for Information Interchange)の場合、数字の9は57となり、文字Aは65に割り当てられている(表1.2)。ここではASCIIの例を出したが、文字コードには様々な種類があり、それによって同じ数字でも違う文字に割り当てられることがある。

1オクテットでは256通りの文字しか表わせないため、日本ではひらがな･カタカナ･漢字を表すのに不十分である。そのため、日本語を扱う際などには、それらにも対応したShift_JISやEUC-JP(Extended UNIX Code Packed Format for Japanese)、UTF-8などの日本語の文字コードを使うことになる。Webページなどを閲覧している時に文字化けして見えることがあるが、この多くは異なる文字コードを使っていることによるものである。異なる文字コードであれば当然違う文字が割り当てられているため、本来と違うコードによって解釈されてしまうと意図した文字と異なる文字として扱われるということである。

1.2.3 モールス符号

文字列の符号化として多くの制御文字があるASCIIより歴史ある例として、**モールス符号**があげられる。これは短点(・。日本語ではトンと呼ばれる)と長点(−。同じく日本語ではツーと呼ばれる)の2元状態を用いた符号である。英文中の発生頻度の高いアルファベットには短い符号、発生頻度の低いアルファベットには比較的長い符号を割り当てることで、平均的な符号長を短縮したものである(表1.3参照)。

長点は短点の3倍の長さと定まっており、また点と点の間には短点1つ分の間隔を空ける。文字間隔は短点3つ分、語間隔は短点7つ分と定められている。

1.2.4 小数の表現

小数は(無限小数なども含めれば)文字や整数と違い、連続的な(アナログな)データである。このため、先に挙げたデジタルデータの利点である「元から離散的で有限個」という性

[3]厳密にはこのcを(b進法における)基数の補数(bの補数)と、cを1小さくしたものを(b進法における)減基数の補数($b-1$の補数)と呼ぶ。本書で出てくるのは全て前者である

表 1.2: ASCII コード一覧 (16 進接頭辞 0x は省いた)

10 進	16 進	文字	10 進	16 進	文字	10 進	16 進	文字
0	00	NUL(null 文字)	16	10	DLE(伝送制御拡張)	32	20	(空白)
1	01	SOH(ヘッダ開始)	17	11	DC1(装置制御 1)	33	21	!
2	02	STX(テキスト開始)	18	12	DC2(装置制御 2)	34	22	"
3	03	ETX(テキスト終了)	19	13	DC3(装置制御 3)	35	23	#
4	04	EOT(転送終了)	20	14	DC4(装置制御 4)	36	24	$
5	05	ENQ(照会)	21	15	NAK(受信失敗)	37	25	%
6	06	ACK(受信 OK)	22	16	SYN(同期)	38	26	&
7	07	BEL(警告)	23	17	ETB(転送ブロック終了)	39	27	’
8	08	BS(後退)	24	18	CAN(とりけし)	40	28	(
9	09	HT(水平タブ)	25	19	EM(メディア終了)	41	29	)
10	0a	LF(改行)	26	1a	SUB(置換)	42	2a	*
11	0b	VT(垂直タブ)	27	1b	ESC(エスケープ)	43	2b	+
12	0c	FF(改頁)	28	1c	FS(フォーム区切り)	44	2c	,
13	0d	CR(復帰)	29	1d	GS(グループ区切り)	45	2d	-
14	0e	SO(シフトアウト)	30	1e	RS(レコード区切り)	46	2e	.
15	0f	SI(シフトイン)	31	1f	US(ユニット区切り)	47	2f	/

10 進	16 進	文字	10 進	16 進	文字	10 進	16 進	文字	10 進	16 進	文字
48	30	0	68	44	D	88	58	X	108	6c	l
49	31	1	69	45	E	89	59	Y	109	6d	m
50	32	2	70	46	F	90	5a	Z	110	6e	n
51	33	3	71	47	G	91	5b	[	111	6f	o
52	34	4	72	48	H	92	5c	\	112	70	p
53	35	5	73	49	I	93	5d	]	113	71	q
54	36	6	74	4a	J	94	5e	^	114	72	r
55	37	7	75	4b	K	95	5f	_	115	73	s
56	38	8	76	4c	L	96	60	‘	116	74	t
57	39	9	77	4d	M	97	61	a	117	75	u
58	3a	:	78	4e	N	98	62	b	118	76	v
59	3b	;	79	4f	O	99	63	c	119	77	w
60	3c	<	80	50	P	100	64	d	120	78	x
61	3d	=	81	51	Q	101	65	e	121	79	y
62	3e	>	82	52	R	102	66	f	122	7a	z
63	3f	?	83	53	S	103	67	g	123	7b	{
64	40	@	84	54	T	104	68	h	124	7c	|
65	41	A	85	55	U	105	69	i	125	7d	}
66	42	B	86	56	V	106	6a	j	126	7e	~
67	43	C	87	57	W	107	6b	k	127	7f	DEL(削除)

表 1.3: 国際モールス符号のアルファベット

英字	出現率	符号	英字	出現率	符号	英字	出現率	符号
A	6.42%	・－	J	0.08%	・－－－	S	5.14%	⋯
B	1.27%	－⋯	K	0.49%	－・－	T	**7.96%**	－
C	2.18%	－・－・	L	3.21%	・－‥	U	2.28%	‥－
D	3.17%	－‥	M	1.98%	－－	V	0.83%	⋯－
E	**10.31%**	・	N	5.74%	－・	W	1.75%	・－－
F	2.08%	‥－・	O	6.32%	－－－	X	0.13%	－‥－
G	1.52%	－－・	P	1.52%	・－－・	Y	1.64%	－・－－
H	4.67%	⋯⋯	Q	0.08%	－－・－	Z	0.05%	－－‥
I	5.75%	‥	R	4.84%	・－・	空白	**18.59%**	

質がなく、デジタル表現のために「最も近い表現可能な値」に丸める必要がある。これは次章のアナログデータの表現で出てくる量子化というステップであり、これによる誤差 (プログラミングの世界では一般に丸め誤差という) は量子化雑音に相当する。

コンピュータ等で扱う小数は、数学等の小数と違い、表せる個数が限られたデジタルデータである。本来はアナログデータである小数を出来る限り近い値で表せるように考えられたデジタル表現とも言える。その表現方法には**固定小数点数表現**と**浮動小数点数表現**がある[4]。

固定小数点数

ある定まったビットまでを整数部として、そこから下の部を小数部としてみなすことでしょうすうを表現する方法を、固定小数点数表現という。例えば、16 ビットある時に、上位 8 ビットがその数の整数部を示し、下位 8 ビットが小数部を示すようにすれば良い。この時、$2^0 = 1$ を示すビットのすぐ下のビットは 2^{-1} を、その下は 2^{-2} を $\cdots$ というように、桁が連続的に定まっていくのが普通である。

この方式は表現できる数の範囲はごく限られるものの (例えば先の 16bit の例ならば、10 進数で 1/256 刻みの値しか扱えない)、定められた範囲においては、誤差を出すことなく高速に計算できる。10 の累乗で表現するような値が頻出する科学計算では利用しづらいが、桁数がある程度定まっている上に誤差が大きな問題となる経済計算などで用いられる。

浮動小数点数

浮動小数点数表現は、小数点の場所を上手く変えながら (浮動) 表す方法である。浮動小数点数では、表したい数をまず以下の形式で表現する。

$$(\text{符号})(\text{仮数}) \times (\text{基数})^{(\text{指数})}$$

この内、基数は通常 2 が用いられる。また、仮数は 1 以上 2 未満の数である (これにより、上記の形式での表現が一意に定まる)。そして、符号 (1bit)、指数 (符号付き整数)、仮数の小数

[4] それぞれ、固定-小数点-数、浮動-小数点-数と区切る。対応する英語も、fixed point number,floating point number である。

部 (固定小数点数) を各々保存する。例えば、-4.8 という数字を浮動小数点数で表現することを考えよう。$-4.8 = -1.2 \times 2^2$ である。したがって、符号部には-が、仮数部には 0.2 が、指数部には 2 が格納されることになる。なお、指数・仮数を何 bit とするかは規格によって異なるが、近年のコンピュータでは IEEE754 という規格に従っていることが多い。

固定小数点数型に比べ、指数部の表現によって科学計算などで用いるような大小様々な値に比較的近い値を表現できるが、誤差が出たり、計算の時間がかかったりといった問題点もある。固定小数点数型は狭い範囲を等間隔に区切っている表現であるが、浮動小数点数は広い範囲をその絶対値に応じてスケーリングして区切っているという見方もできる。

1.3 符号化効率の改善と圧縮

データの記録や通信の資源は限られているため、少しでも効率よく記録・通信したい。先に挙げたモールス信号は光あるいは音を人間が認識して行うことから伝達に時間がかかる。そのため、少しでも短縮するために文字の出現頻度を考慮して符号の長さが決まっていた。同様に、何らかの手法を用いてデータを**圧縮**して符号化する、あるいは符号化したものを圧縮する方法が考えられた。

1.3.1 符号化効率の改善

先に扱った例では、取りうる値各々に定長を割当てて情報を表していた。しかし、何らかの手法でこの定長を短くしたり、可変長にしたりすることで、同じ情報でもより短い符号にて記録できる。

サイコロの出目を考えよう。6 つの面が等確率で出る普通のサイコロの場合、単に定長を割り当てるなら 3bit が必要になる。しかし、00,01,100,101,110,111 と言う 6 通りの状態を出目に対応させると、**平均符号長**は

$$\frac{1}{6}(\text{の確率}) \times 2(\text{bit}) \times 2(\text{通り}) + \frac{1}{6}(\text{の確率}) \times 3(\text{bit}) \times 4(\text{通り}) = \frac{8}{3}$$

となり、3bit の定長で割り当てる場合に比べておよそ 11%短いデータで表現できている。また、ここで表した 6 つの符号は「先頭から読む際、0 を読んだら 2bit,1 を読んだら 3bit」というルールを課すことで混同することなく元の情報を取り出すことができる。

このように、データ量を節約することは保存の上でも無論便利であるが、より短時間での通信や記録が可能となるため利便性が大きい。

モールス符号の平均符号長

モールス符号は文字の出現頻度を考慮して符号の長さが決まっていたと記したが、実際の平均符号長はどのぐらいか計算してみよう。表 1.4 に各符号の符号長 (符号の文字数) と、その点長（一つの文字を表すのにかかるのが短点の何倍になるか）を記した。

ここで、点長は、長点が短点の 3 倍の長さで、点と点の間が 1 倍の長さであることから、長点の数 ×3+ 短点の数 (空白も含む)+ 文字数 −1 で計算している (実際には、これに字間・語間の長さが加わるが、これは無視している)。

表 1.4: 国際モールス符号の英字符号長

英字	出現率	語長	点長	英字	出現率	語長	点長	英字	出現率	語長	点長
A	6.42%	2	6	J	0.08%	4	13	S	5.14%	3	5
B	1.27%	4	9	K	0.49%	3	9	T	7.96%	1	3
C	2.18%	4	11	L	3.21%	4	9	U	2.28%	4	7
D	3.17%	3	7	M	1.98%	2	7	V	0.83%	7	9
E	10.31%	1	1	N	5.74%	2	5	W	1.75%	4	9
F	2.08%	4	9	O	6.32%	3	11	X	0.13%	4	12
G	1.52%	3	9	P	1.52%	4	11	Y	1.64%	4	13
H	4.67%	4	7	Q	0.08%	4	13	Z	0.05%	4	11
I	5.75%	2	3	R	4.84%	3	7	空白	18.59%	0	1

表 1.4 を元に、平均符号長を計算する。出現率 × 語長の総和を取ると、2.12 となり、空白を込めて 27 通りのデータであるにもかかわらず、二元状態 2 文字強とかなり短くなっていることがわかる。また、同様に各文字の点長の平均は (短点を 1 として)5.18 の長さである。仮に文字を全て固定長 5bit にし 1 を長点・0 を短点・字間の空白は同じと考えると、(最も効率よく当てはめたとしても) 2 倍以上の長さとなるため、長さを 2 元状態にするという制約下においてモールス符号は効率化されていることがわかるだろう[5]。

1.3.2 可逆圧縮・不可逆圧縮

圧縮には大きく分けて、「本来の情報のうち不要な部分を捨て去ることにより符号を短くする」手法と、「本来の情報の何らかの特性を利用してより短い表現を用意し符号を短くする」手法がある。前者のように、元の情報から何らかの情報が捨てられるケースでは捨て去られた部分のデータを復元して元のデータに戻すことは不可能であるので、**不可逆圧縮** (あるいは**非可逆圧縮**と呼ばれる。一方、後者の手法は元のデータに戻すことが可能なので**可逆圧縮**と呼ばれる。本来の圧縮の手法では元のデータの復元が可能であることが条件なので不可逆圧縮は圧縮とは呼べないともされるが、現実的には圧縮の一種として扱われている。

電気を使わない世界での不可逆圧縮の例を考えてみよう。不可逆圧縮としては、電車の遅延証明書が挙げられる。電車の遅延証明書は「この時間帯に最大で○分の遅れが出たことを証明する」と言う形式になっている。それぞれの電車についてどの駅で何分遅れたかは、データとしては存在するだろう。しかし、そのような細かい情報は遅延証明書を必要とする側にとっては不要な情報である。また、それぞれについて証明書を発行する手間も膨大なものになる。そこで、時間帯と最大の遅れによって遅延証明書を一律化したのである。このように、大勢に影響ない情報を捨て去るのが不可逆圧縮である。電気通信の世界では、音声データの可聴域以外の部分を捨てる (MP3) などの例が見られる。

一方、電気の世界でない可逆圧縮の例としては、歌詞カードがわかりやすい。歌詞カードには、同じ歌詞の部分に (∗ くりかえし) などと書かれる。これによって、歌詞という情報に欠損は生じない。しかし、全体としての文字数や行数は減り、狭い歌詞カードのスペース

[5]但し、モールス符号は最適な割り当て方をしているとは言えない。平均符号長を最適化すると 1.99 となってその時の点長の平均は 4.95 と 5 を切る。点長の平均を最適化した場合は 4.84 となり、平均符号長は 2.07 と、より良い方法もある点には留意されたい。

を節約したり、一覧性を向上させたりできる。歌詞カードを読む側は (∗ くりかえし) をあてはまる部分と置き換えることで (解凍) 元の歌詞を復元できる。これが可逆圧縮である。電気通信の世界では、例えばデータを zip ファイルや tgz ファイルなどが挙げられる。

本書ではデータの特性などを個別に論じないため、可逆圧縮のみを扱う。

1.3.3 エントロピー符号

モールス符号である程度考慮されていたように、元の情報の頻度に偏りがある場合、高頻度のものを短い符号に割り当てることで平均的な符号を短くすることができる。

先に、サイコロの出目を考えた時は全て等確率とした。しかし、サイコロの出目が必ずしも等確率で出るとは限らない。市販されている一般のサイコロでも、その目の彫りや塗装で厳密に等確率とは言えないのだそうである[6]。また、直方体の”サイドタ”や、おもりを入れた”グラサイ”など、明らかに等確率ではないものもある。例えば、2 つの出目が 1/3 の確率で出て、残りの目が 1/12 で出るようなサイコロ (?) のデータを考えてみる。この時、先に考えたのと同じく 00,01,100,101,110,111 を割り当てるなら、00 と 01 の 2bit の符号は、1/3 の確率で出る目に割り当てたほうが全体としては短くなるだろう。この時の平均符号長は

$$\frac{1}{3}(\text{の確率}) \times 2(\text{bit}) \times 2(\text{通り}) + \frac{1}{12}(\text{の確率}) \times 3(\text{bit}) \times 4(\text{通り}) = \frac{7}{3}$$

となり、等確率の場合に比べてより短くできている。

このように、出現確率によって長さを変えて割り当てる符号を**エントロピー符号**と呼び、次のような種類がある。

エントロピー符号の例

- アルファ符号
- ガンマ符号
- シャノン符号
- ハフマン符号
- CBT 符号 (先のサイコロの例)
- (整数の) ユニバーサル符号
- フィボナッチ符号 (KZ 符号)
- レーベンシュタイン符号
- 算術符号
- レンジ符号

以下、この内符号語が整数であり、アルゴリズムが易しいなど、比較的シンプルなものを見ていこう。

アルファ符号

アルファ符号は単進符号とも言われ、あとに示すガンマ符号の基礎となっている考えの符号である。

[6] プレシジョンダイスや”世界最速のサイコロ”といった、限りなく等確率に近づけたサイコロもあるようだ。

符号化したいデータの出現頻度が n 番目に高いとするとき、これを n 桁の符号に変換する。符号は末尾が 1 で、それ以外の桁を 0 としたものなどが代表的である。つまり、1 番目は 1,2 番目は 01,3 番目は 001,4 番目は 0001 という具合である。これ単体ではデータ数が多い時に非効率な符号となる可能性も高いので、単体で用いられることは少ない。

ガンマ符号

ガンマ符号は、アルファ符号と同様に符号化したいデータの出現頻度が n 番目に高い時に、n に応じて長くなる符号である。n を 2 進数表記し、その上に n の 2 進桁数-1 個の 0 をつける。例えば、1 番目は 1,3 番めは 011,6 番目は 00110,10 番目が 0001010 となる。例示したケースでは固定長にしたほうが効率よく見えると思われるが、データパターン数が多く偏りが強いほど、ガンマ符号の効率が良くなる。

CBT 符号

CBT 符号 (Complete Binary Tree 符号) は、先の 6 面サイコロの例として挙げたもので、固定長符号を元にしながら、不要な箇所を省いて符号を短くする方法である。

k ビットの固定長符号では $2^k - 1$ までの数値を表現できる。しかし、全データが m より小さければ、m 以上の部分は無駄になる。そこで、$2^k - m$ 未満の数を $k-1$ ビットの 2 進数に割り当て、$2^k - m$ 以上の数は、その数に $2^k - m$ を足した数の k ビット 2 新表現に割り当てる。例えば 10 通りの数を表す場合、4bit あれば十分なので、0 から 5 はそれぞれ 000 から 101 に割り当てる。6 から 9 は 1100 から 1111 に割り当てる。この時、割り当てる順番を出現頻度降順にすれば最も良い効率の割当となる。

フィボナッチ符号 (KZ 符号)

フィボナッチ符号は $F_{n+2} = F_{n+1} + F_n\ ,\ F_1 = F_2 = 1$ という式により計算されるフィボナッチ数列を用いた符号化法である。フィボナッチ数列を順に記すと 1,1,2,3,5,8,13,21... となるが、この内最初の 1 を除いた、1,2,3,5,8,13,21... を用いて数値を表現する。ゼッケンドルフの定理と呼ばれる定理では「任意の正の整数が、連続するフィボナッチ数を含まず、同じフィボナッチ数を 2 度含まない形で、相異なる 1 つ以上のフィボナッチ数の和として一意に表現できる」とされる。例えば、42 であれば、8+34($F_6 + F_9$) により表現できる。この時、前から順に $F_2, F_3, F_4, ...$ の有無を 1/0 で表して並べたものをフィボナッチ表現と呼ぶ。先の 42 であれば 00001001 であるし、33 であれば 1010101 となる。

出現頻度が n 番目に高いデータを、数 n のフィボナッチ表現にして表すものがフィボナッチ符号である。ゼッケンドルフの定理の「連続するフィボナッチ数を含まず」の件からわかる通り、フィボナッチ符号には連続した 1 が出ないため、11 などを符号の制御用に利用できる。特に、データの接頭辞として 110 を付す形のものを **KZ 符号** (Kautz-Zeckendorf 符号) と呼ぶ。これは、データを先頭から見なくともどこがデータの区切りかはっきりわかるという利点がある。

ハフマン符号

エントロピー符号の中でも最も代表的な例が**ハフマン符号**である。これは、次の手順により符号化を行う手法である。

ハフマン符号の手順

1. すべての情報源シンボルを節点 (葉) とする 2 進木 (binary tree) を書き、葉にはそのシンボルと発生確率を併記する。
2. 最も発生確率の低い葉を 2 枚選び、それらに対する新しい節点を作り 2 枚の葉と 2 本の枝で結ぶ。このとき 2 本の枝にそれぞれ 1、0 を割り当てる。さらに新しい節点には 2 枚の葉の確率の和を書き、この接点を新たな葉と見なす。
3. すべての情報 (葉) を枝で結ぶまで 2 を繰り返す。
4. 最後に作成した新しい節点を根と呼び、根から各情報 (葉) に至る枝をたどりながら、2 でその枝に割り当てた 1、0 を順に読んでいくことで符号化を行う。

具体的に、8 種類の文字をハフマン符号化した例を図 1.1 に示す。

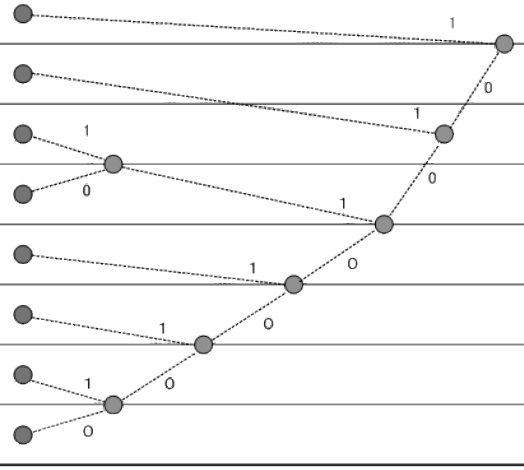
Alphabet	Probability	Tree of code	Code word
A	0.5		1
B	0.2		01
C	0.1		0011
D	0.08		0010
E	0.05		0001
F	0.04		00001
G	0.02		000001
H	0.01		000000

図 1.1: 8 種類の値のハフマン符号化 東邦大学メディアネットセンター 情報通信理論 (佐藤洋一) より引用

ハフマン符号は整数の符号語長という制約のもとでは、常に最適な符号を構成できるという特徴がある。(実数の符号語長を許す場合には、レンジ符号などより高効率なものがある。) これは**シャノンの情報源符号化定理**と呼ばれる定理により説明できる[7]。

[7]数学的な取り扱いがやや高度であるため、本書では示さない。ここでは「圧縮できる限界を示した定理」と考えれば良い

1.3.4 辞書式 (ユニバーサル) 符号

データにおいて値が無秩序に並んでいることは少なく、なんらかの関連性があるものである。データ値の並び方のパターンを抽出し、そこに元よりも短いコードを振って出力することでデータの情報量を減らすことができる。この手法を**辞書式符号**または**ユニバーサル符号**と呼び[8]、次のような種類がある。

ユニバーサル符号の例

- バイト対符号化 (BPE)
- ランレングス (RLE)
- LZ78
- Deflate

以下、この内の何種類かを見ていこう。

バイト対符号化 (BPE)

バイト対符号化 (Byte Pair Encoding) は、2 バイト (2 文字) の複数個の塊を辞書にする圧縮方法である。例えば、日本語で「ふらふらとふらつく」という文を考えてみよう。この文には「ふら」という文字列が 3 度出てくる。そこで、この 2 文字を仮に A に変えると「AA と A つく」と文字列がずいぶん短くなる。

このように、2 文字の同一の文字列があるときこれを使われていない別のバイトに置き換えていく（再帰的に適用する）ことで、文字列をより短くできる。そして、置き換えた一覧 (辞書) とともに送付することでバイト列を圧縮する。圧縮には時間がかかるが、展開はすぐにできるため、受信側が非力な機器である際に利用される手法である。

LZ78

LZ78 は読み込んだ記号列から動的に辞書を作成して、それをもとに入力記号列を置き換えていく**動的辞書法**とも呼ばれる圧縮法である。先に示した BPE と違い、辞書を別途送ることなくデータを圧縮する。

この圧縮手順は次のようになる。

1. 辞書を空にし、コード値の最大を 0 とする。
2. データ列を走査し、データパターンが辞書に存在するか調べる。辞書に存在した場合はそのコード値を出力する。なければ、コード値 0 を出力する。
3. コード化を行ったデータパターンの次のデータ 1 つをそのまま出力する。

[8] エントロピー符号に出てくる (整数の) ユニバーサル符号とは別の意味であるので注意。英語でも、整数の方は universal code、辞書式の方は universal source coding と名称が異なる。整数の方のユニバーサル符号とは、大きい数のほうが出現頻度が低い分布において効率的に (最適な符号化と同程度のオーダーで) 符号化可能な手法の総称である。

4. 前項・前々項で出力したデータについて、コード値を辞書の値に置き換えたデータを、新データパターンとして辞書に登録する。このときのコード値は、コード値の最大+1とする。

5. データ終端に至るまで、上記の 2〜4 項を繰り返す。

わかりづらいので、AAABBBAAABBBBAAA という文字列を圧縮することを考えてみよう。

1. 最初、辞書は空である。データ列を最初から見ていったときのパターンは A である。A というパターンは辞書にないから、コード値 0 とその次のデータ (A) を出力し、辞書にはコード値 1 として A を出力する。

   文字列：A AABBBAAABBBBAAA
   出力：0A
   辞書： 0-(空),1-A

2. 2 文字目は A であるので、1-A に合致する。このため、コード値 1 とその次のデータ (A) を出力し、辞書にはコード値 2 として AA を出力する。これにより、3 文字目の A までが表されたことになる。

   文字列：AAA BBBAAABBBBAAA
   出力：0A1A
   辞書： 0-(空),1-A,2-AA

3. 4〜6 文字目についても同様に行う。

   文字列：AAABBB AAABBBBAAA
   出力：0A1A0B3B
   辞書： 0-(空),1-A,2-AA,3-B,4-BB

4. 7 文字目〜9 文字目は、コード値 2 の AA に A をつけたものであるから、2A とする。2A に対応する AAA を辞書に登録。

   文字列：AAABBBAAA BBBBAAA
   出力：0A1A0B3B2A
   辞書： 0-(空),1-A,2-AA,3-B,4-BB,5-AAA

5. 10 文字目〜12 文字目は、コード値 4 の BB に B をつけたものであるから、2B とする。2B に対応する BBB を辞書に登録。

   文字列：AAABBBAAABBB BAAA
   出力：0A1A0B3B2A4B
   辞書： 0-(空),1-A,2-AA,3-B,4-BB,5-AAA,6-BBB

6. 以下、同様に繰り返す。

文字列：AAABBBAAABBBBAAA
出力：0A1A0B3B2A4B3A2(空)
辞書： 0-(空),1-A,2-AA,3-B,4-BB,5-AAA,6-BBB,7-BA,8-AA(空)

出力のデータはもとのデータより短くなっており、また出力内容から辞書を作成しつつ解凍することができるので、辞書を別途送る必要がない。これが動的辞書法と言われるゆえんである。なお、この例ではあまり長い例がないため圧縮効果が低いが、実際には辞書に長いデータが入って同じデータが頻出する場合などより効果的な圧縮となる。

またこの手法は、記号列の長さを増して部分列に分解していくことから、**増分分解法**とも呼ばれる。ユニバーサル符号の代表的な手法である。

ランレングス (RLE)

ランレングス (Run Length Encoding) あるいは**連長圧縮**は、同一のデータが複数続いているとき、その個数とデータで表現することにより全体の長さを短くする方法である。例えば、”AAAAAAABBBBB”というデータがあったとき、”Aが7個、Bが5個”の意味で”7A5B”とすると文字列が短くなる。これは特に、データの種類が少なく、また連続している可能性が高い場合に大きな効果を発揮する。とりわけ、白黒のファクシミリでは、白か黒かの2値であることと、よほど細かい市松のような状況でない限りは白・黒の連続性が高いと考えられることから、ランレングスがよく利用される。

1.3.5 差分 (デルタ) 圧縮

符号の圧縮の多くは、先に挙げたエントロピー符号または辞書式符号に分類される。しかし、その双方に分類されない（というより分類の前処理にあたる）のがここで扱う**差分圧縮**または**デルタ圧縮**である。この方式は、通信よりも寧ろデータのバックアップなどによく用いられる。

定期的にシステムやデータのバックアップを取る場合を考える。その期間にもよるが、毎回全てのデータを上書きする必要はないことが多い。期間が短いほどその変化は少ないだろう。そこで現行と以前のあるバックアップ時との差分を用意し、これを保存しておく。このように、ある完全なデータを一つ用意した上で、そこからの差分を用いて圧縮するのがこの差分圧縮である。性質上、前述のエントロピー符号や辞書式符号と併用することができる。特に、差分データについては”差分なし”あるいは”僅かな差分”の部分が大半を占めることが多いため、これらに短い符号を割り当てることにより圧縮効率が高まると期待できるのである。

この例ではバックアップを出したが、関連性のある複数のデータであれば同様の効果が期待できる。現実にも定点観測のデータやソフトウェアのアップデートなどでこの手法が利用されている。

演習問題

1 古いゲームにおいては、容量の制約から 16bit の整数型などを採用せざるを得なかった。そのため、例えば (0 以下の場合は 0 以下になった時の処理が必要となる) 体力などの数値は 32768 が最大となっていた (現実には、これを 3〜4 データ用意するなどして増強を図っていた)。ところが、この最大値は 2 の補数表現を採用した 16bit 符号付き整数型の最大値と 1 異なる。内部上の表現は確かに 2 の補数表現で間違いなく、処理によってこの違いが生じているのだが、どのような処理をしていると考えられるか説明せよ。

2 データの種類数と、各データの出現比率が与えられるとき、そのハフマン符号化の例と、その際の平均符号長を出力するプログラムを作成せよ。例えば、4 種のデータが表 1.5 のような比率で入力される場合、平均符号長は 1.75 となる。なお、簡単のため

表 1.5: ハフマン符号化を考えるデータ例

データ種別	A	B	C	D
出現頻度	4	2	1	1
符号例	0	10	110	111

に出現頻度は正整数値で、その合計は 50000 を超えないものとして良い。

3 浮動小数点数の配列を圧縮して送りたい場合、単純なランレングスでは効果が出づらい。そこで、各値の同一ビットのみを集めてランレングスをかけ、これを順に記すことで圧縮することを考える。すなわち、1 ビット目のランレングス情報、2 ビット目のランレングス情報…と書かれたデータを作成して圧縮するのである。この手法について以下の問いに答えよ。

(a) 浮動小数点数の配列にどのような傾向があるときにこの圧縮は効果的か。

(b) 整数や文字の配列の場合、このように同一ビットのみを集めた圧縮は効果的と言えるか。データの特性とともに考えてみよ。

第 2 章

アナログデータの記録

現実世界において対面する多くのデータは離散的ではない。音声や映像は無論のこと、道程や時間も連続的なデータ (アナログデータ) である。アナログデータをそのままに記録・伝達することは多くの場合難しい。そのため、保存にあたっては復元可能なデジタルデータへと置き換えて保存される場合が多い。

本章では、まずアナログデータをそのまま連続的な量として記録できる例を説明する。次いで、デジタルデータへの置き換えについて、その手法と制約、影響等を論ずる。

2.1 連続データの記録

デジタルデータの保存方法がなかった頃、あるいは乏しかった頃、アナログデータは変換なしにそのまま保存されていた。図画はその最も原始的なものであろう。風光明媚な風景を伝えたいと描かれた風景画、その人物の偉大さを讃えた肖像画、建物の設計図などは、人間の視覚による差異や筆致の巧拙を別にすれば、おおよそそのままに伝えている。あるいは立像なども直接保存する一例と言えよう。

時代は下り、人はアナログのデータを直接に、あるいは必要に応じて媒体を変換させつつも連続性を保ったまま保存する術を開発してきた。ここでは、光・音・運動の 3 例を紹介する。

2.1.1 光の保存例：写真

光をそのまま保存している例としては写真がある。写真はカメラを用いて図 2.1 に示すように光をフィルムに投影する。

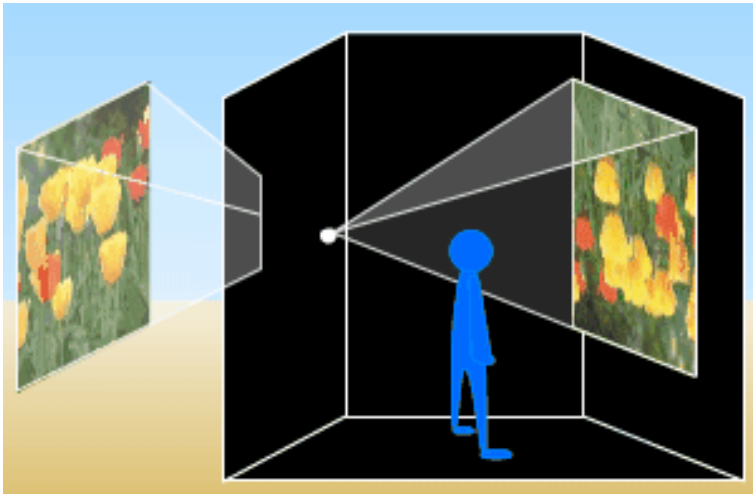


図 2.1: ピンホールカメラ。Canon サイトより引用

フィルムには感光体があり、この感光体が受けた光に応じて変化を起こす。記録自体はこれで完了である。実際に読み出すには、この変化した感光体を薬品などに浸して化学変化させて再度発色させるなどの工程が行われる。一般には現像と呼ばれる工程である。

フィルムの化学変化を光によって起こすという方法で、そのままでは消えてしまう光を物質の際によって保存しているのが写真なのである。

2.1.2 音の保存例：レコード

音声・音楽の保存というのは非常に顕著な変化を見せている。戦後の頃には、レコードが主たる媒体であった。レコードは薄い板の両面に音楽のデータを刻む事ができる媒体だった。現代でも稀に”両 A 面シングル”などという呼び名があるが、これは元々レコードに A 面と B 面という区別があったことから出来た言葉である。

その後、AB がなくなり、CD へと移った (桂文珍「心中恋電脳」)。コンパクト・ディスクは片面のみにデジタルデータで保存を行い、これを光学的に読み取って音声へと変換することで音楽を鳴らし始めた。ここがアナログとデジタルの分かれ目である。AB…つまりレコードはアナログであったが、CD はデジタルということである。アナログとデジタルの変換を A/D 変換などと書くことがあるが、なるほど、A(B)/(C)D の変換と考えると確かにこの変化は顕著である。…洒落はともかく、このデジタルとなった音楽をインターネットを介して運ぶようになったのが現代のダウンロードによる音楽配信である。ことほど左様に音声・音楽データの保存は記録や通信の変化に対応しているのである。

レコードでの音楽の保存はアナログな音を「そのままに」保存しているのであるが、これは一体どういう原理であろうか。

音声というのは空気の振動である。この空気の振動を保存すれば良い…これがレコードの最も基本的な考えである。空気の振動を適当なもの（通常は針）に集音して伝え、下の面で回転している（記録前の）レコードに当てる。これにより、レコードが振動に合わせて削られる。再生するときは、逆にレコードの削れた面をなぞるように針をあて、この振動を増幅させて音声とする。

空気の振動ということでは動きもするし減衰もする。そのため、空気の振動をそのまま別の振動へと変換して保存するのがレコードという媒体の原理である。

2.1.3 運動の保存例：地震計

レコードは空気の振動であったが、同じ原理を用いて地面の振動=地震の揺れを記録するのが地震計である。

地震計は錘（おもり）に糸をつけた振り子を用意し、錘を不動点として地面の揺れを記録する。(図 2.2)

振り子の糸の端を素早く左右に動かした場合、錘は動かない。そのため、錘の下にペンをつけ地面に紙を置くと紙だけが揺れるため地面の動きを記録することができる。これを複数

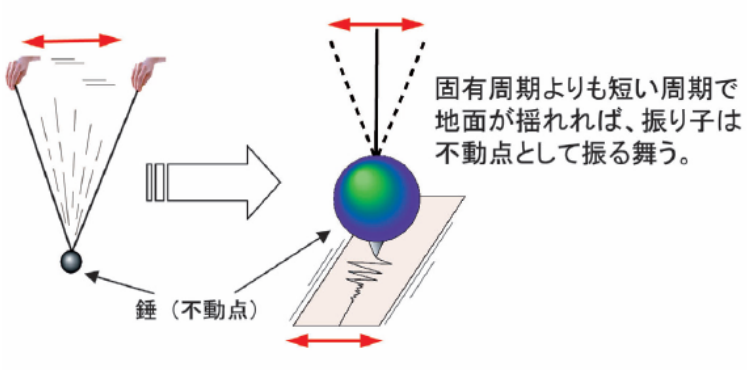


図 2.2: 地震計の原理。札幌管区気象台サイトより引用

の方向（東西・南北・上下）用意すれば、地震の 3 次元的な揺れをそのまま記録することができる。レコードと違い、これをそのままに再現するのは困難であるが（また、再現されると非常に迷惑でもあるが)、地震の状況を伝える資料として保存する、あるいは伝えるのには格好の方法と言えよう。

2.2 デジタルデータへの変換

アナログデータの保存の例を示してきたが、アナログデータは必ずしも移行や保存が楽ではなく、また通信の観点から見ても例えば地震計をそのままに伝送するには紙を運送する必要がある。また、レコードのように専用の機器が必要になる場合も多く、取扱が簡便であるとは言い難い。そこで、汎用性を持った記録・通信のためにデジタルデータへ変換して保存するということが考えられる。

アナログデータをデジタルデータに変換する場合というのは、連続性のあるものから代表値を選び出してそれを順に記録することとなる。これは、次の三つの手順からなる。

1. **標本化** (Sampling) : アナログデータから代表値（サンプリング値・標本値）を取得する。
2. **量子化** (Quantization) : サンプリング値をデジタルで表現可能な最も近い値へと丸める。
3. **符号化** (Encoding) : 量子化された値（量子化値）を記録できる形式に変換する (多くはビット列となる)。デジタルデータの符号化と同様である。

この途上において、最初に歴史に学んだ通り、取捨選択されている部分が存在する。アナログとデジタルを比較した文脈においては、しばしば「デジタルにはないアナログの良さ」が標榜されているが、その多くはこの標本化や量子化において捨象された部分を指す。捨象された部分というのは伝達に不都合であるとか影響が少ない、あるいはそもそも不必要といった部分であるため、伝達や記録と言った主目的を達するために、本質的でない部分を犠牲にしているというのがデジタルへの変換の基本的な考えである。これはそのまま、「微細な心情等は伝わりづらいが効率的」とされるデジタルデータと、「扱いは面倒だが機微が伝わる」とされるアナログデータの印象にも繋がっていると言えよう。

閑話休題。上記のようにデジタルデータへと変換し保存された後は、保存されたデータを読み取ってその値を適切につなぐことで原データの近似データを得られる。この近似データが十分良い近似であればアナログデータの保存という目的は達せられたと言える。では、この適切な近似データを得るためにはどうすればよいだろうか？以後、変換の各手順とともに良質な変換を担保する条件を見ていこう。

2.3 アナログデータの標本化

アナログデータの標本点を定め、標本値を取得するのがアナログデータの標本化である。つまり、連続的なデータの何点かを取り、その点の値を取得するという動作を標本化と呼ぶ（図 2.3）。

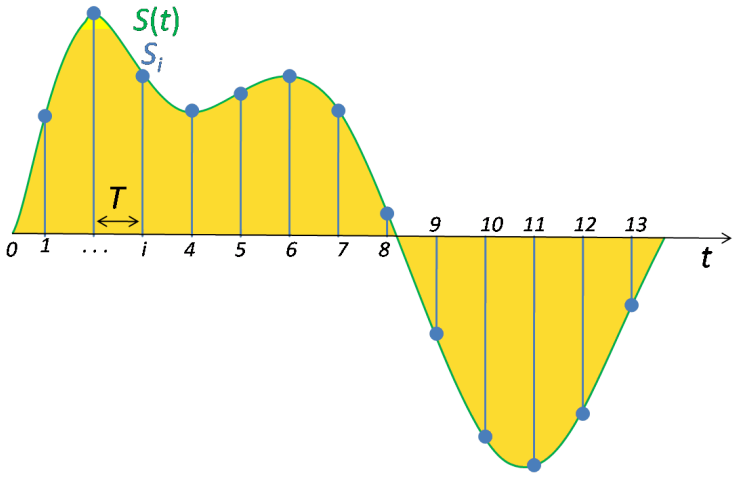


図 2.3: 標本化の例。各点における縦線が標本値を示す。Wikipedia より引用

得られた標本値を適切に結ぶことができれば原データを復元できる。しかし、標本値が十分な個数なければ原データを正確に・精確に復元することは困難になるだろう。だからといって過剰に多くの点を取ってしまえばデータの量が嵩んでしまうのは目に見えている。これらの観点からすれば、原データを復元するに必要な最小限のデータを標本値として取得することが望ましい。では、どのようなデータをとれば原データを復元できるだろうか？その答えの源流は、19 世紀初頭頃の物理学の話に遡る。

2.3.1 Fourier の夢

18 世紀から 19 世紀にかけて活躍した、フランスの数学者・物理学者 Jean Baptiste Joseph Fourier 男爵は、ある物体中で熱がどのように伝わるのかを表す熱伝導方程式を導出した。もっとも、熱伝導方程式は方程式と言っても一般にいう代数方程式—未知数を含む関係式があり、限られた値しか認められないような等式—ではなく、関数方程式—未知の関数に関する関係式があり、これを満たす関数が限られる等式—である。つまり「あてはまる数値を求める方程式」ではなく「あてはまる関数を求める方程式」である。

Fourier は著書「熱の解析的理論」において、”任意の関数は、三角関数[1]の級数[2]で表すこと

[1] ここでは $\sin, \cos$ のこと。
[2] 無限和のこと。

ができる”と主張した。Fourier 自身が与えたこの主張に対する証明は不十分であったが、後世の数学者により「ほとんどあらゆる」関数について三角関数の総和形式で表されることが示された。方程式論や方程式の解法に終生興味を持ち続けた Fourier の夢が花開いたといえる。このように、関数を三角関数の総和形式に表すことを、**Fourier 展開**あるいは **Fourier 変換**と呼ぶ[3]。

三角関数は周期的に変化する。このため、三角関数は波を表していると言える。波の周期の逆数を**周波数**と呼ぶ。Fourier 展開とは、異なる周波数の波を十分な個数集め、必要な程度に増幅あるいは減衰させて足し合わせることによって関数を表せるという主張である。各々の周波数における増幅あるいは減衰を表す係数を**周波数スペクトル**と呼ぶ。

現実に扱うアナログデータにおいて、Fourier 展開が出来ないようなデータはまずない。このため、アナログデータは三角関数の総和形式によって表すことができるといえる。そして、この三角関数の総和形式から、「どのように標本点を取ればよいのか」という答えが導かれる。そう、原データを Fourier 展開した際に含まれる周波数スペクトルを求めるに必要なだけのデータを取れば良い、ということである。

2.3.2 【補遺】Fourier 展開の数学的議論

> 本節は大筋に影響せず、やや高度な (大学程度の) 数学を扱う。
> 難解あるいは興味索然たるものと感じる折には
> 飛ばして次節を読まれたい。

先に書いた Fourier 展開について、もう少し数学的に議論しておこう。Fourier 展開の主張とは、ある関数 $f(x)$ について

$$f(x) = \frac{a_0}{2} + \sum_{k=1}^{\infty} (a_k \cos f_k x + b_k \sin f_k x) \tag{2.1}$$

が成立するということである。但し、最初の定数項は調整のために係数をつけている。各係数 a_k, b_k が周波数スペクトルを、f_k が周波数を表す。

ここで、$f(x)$ の周期が L であるとしたとき、これを表すための周波数 f_k は

$$f_k = \frac{k\pi}{L} \tag{2.2}$$

として表される。このとき、三角関数には直交性がある[4]ため、式 (2.1) の両辺に適当な周波数成分をかけて周期積分することにより、次のように係数を求めることができる。

$$a_k = \int_{-L}^{L} f(x) \cos f_k x dx \quad , \quad b_k = a_k = \int_{-L}^{L} f(x) \sin f_k x dx \tag{2.3}$$

[3] 一般には、有限区間での総和を Fourier 展開、無限区間での総和を Fourier 変換と呼び分けている。

[4] 2 つの関数同士の積を周期積分したとき、同一の関数でなければ 0 となり、同一の関数であれば 0 でない有限値となる

アナログデータの再現とは、離散化されたデータを用いて a_k, b_k を計算し、元の $f(x)$ を得ることである。つまり、アナログデータから変換された後のデジタルデータは a_k, b_k を元通り求めるのに必要なデータと言える。なお、この計算は、離散フーリエ変換 (discrete Fourier transform,DFT) あるいはそれを高速化した高速フーリエ変換 (fast Fourier transform,FFT) と呼ばれる手法による。

2.3.3 標本化定理

Fourier の主張により、原データを Fourier 展開した際に含まれる周波数スペクトルを求めるに必要なだけのデータをサンプリングすれば良いことまでは前節で見えてきた。では、「必要なデータ」とは一体どういうデータであろうか。それを示すのが**標本化定理**あるいは**サンプリング定理**である。この定理は、1928 年に Harry Nyquist によって予想され、1949 年に Claude Elwood Shannon と日本の染谷 勲によってそれぞれ独立に証明された。この予想者あるいは証明者の名前を取り、**ナイキストの定理、ナイキスト・シャノンの定理、染谷・シャノンの定理** (主に日本) と呼ばれることもある。

標本化定理は次のことを主張している。

> 標本化定理
>
> 原信号に含まれる最大周波数成分の 2 倍以上の周波数でサンプリングされたデータがあれば原信号を完全に再構成できる。逆にサンプリングデータの周波数が原信号の 2 倍を下回る場合は完全に再構成できるとは限らない。

この定理において、原信号に含まれる周波数の 2 倍という値が出てくるが、この周波数を**ナイキスト周波数**と呼ぶ。また、サンプリングの周波数 (=サンプリング間隔の逆数) を**サンプリング周波数**と呼ぶ。この言葉を用いて換言すれば、標本化定理とは「原信号の完全復元にはナイキスト周波数以上のサンプリング周波数でサンプリングされたデータが必要である」と言える。

原データの最大周波数はデータの特性等によって定まる。例えば音声の場合、人間の可聴域は概ね 20kHz と言われることから、これを少し上回る程度の最大周波数とする場合が多い。いわゆる音楽 CD の規格である CD-DA では、最大周波数を 22.05KHz、サンプリング周波数をその 2 倍の 44.1KHz として定めている。

2.3.4 【補遺】標本化定理の証明の概要

> 本節は大筋に影響しない。
> 難解あるいは興味索然たるものと感じる折には
> 飛ばして次節を読まれたい。

標本化定理は「定理」であるから当然証明されるべき事項である。但し、高等学校までの数学を基本としている本書においては、この証明は補遺としてもやや高度に過ぎる。そのため、以下で標本化定理が成立する概略を説明する。

ディラックデルタ関数 $\delta(x)$ を

$$\delta(x) = \begin{cases} 1 & x = 0 \\ 0 & x \neq 0 \end{cases} \tag{2.4}$$

と定義する。

原アナログデータを $f(t)$ とし、周期 T でサンプリングしたと考えれば、標本値は

$$p(t) = f(t) \sum_{n=-\infty}^{\infty} \delta(t - nT) \tag{2.5}$$

と与えられる。(現実的には始点・終点が定まるがここではどのような区間でもということで無限大区間を取っている。)

これを Fourier 変換して周波数スペクトルを求める（この部分の計算に畳み込みなど Fourier 変換のやや高度な知識を要するので略す)。すると、原信号のスペクトルを少しずつずらして足し合わせたものとなる（図 2.4)。

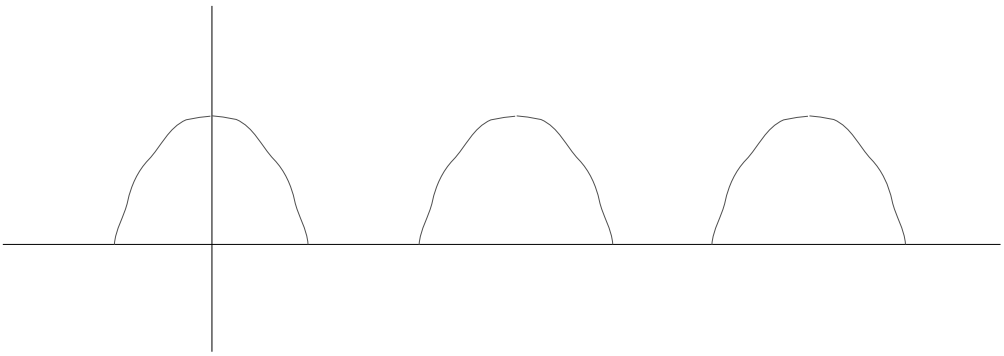


図 2.4: 標本値の Fourier 変換。y 軸のあたりに現れる原信号が、右側に等間隔で再度現れている

このとき、同一のスペクトルの「ズレ」は $\frac{2\pi}{T}$ という周波数になり、原信号のスペクトルの両裾とスペクトルの中心 (0) 間の距離は原信号の最大周波数に相当する。この裾同士が重なり合わないようにサンプリングできれば、1 つの山=原信号だけを取り出すことができる。したがって、「ズレ」の間に原信号のスペクトル 1 つ分がすっぽり入れば良い。式でいうところでは、$\frac{2\pi}{T}$ の間に原信号のスペクトル 1 つ分=最大周波数の 2 倍が入れば良いということになるので $\frac{2\pi}{T}$ は最大周波数の 2 倍より大きければいいということになる。ここで、$\frac{2\pi}{T}$ はサンプリング周期の逆数と 2π をかけたものであるから、サンプリング周波数そのものである。

2.4 標本値の量子化

標本値を取得することで、アナログデータから離散値=デジタルデータを抜き出すことが出来た。しかし、これではまだ記録するには不十分である。というのは、標本値は必ずしも表現可能な数値とは限らないためである。

端的な例として、1 辺が 1 の正方形の対角線の長さを考えよう。三平方の定理を鑑みればその長さが $\sqrt{2}$ であることは明らかである。しかし、我々はこれを数値としてそのまま完全

に保持することは出来ない。なんとならば、$\sqrt{2}$ は無限に続く上に分数としても表現できない、無理数だからである。有限の 10 進値なり 2 進値なりで保持あるいは計算するというデジタルデータの原則に沿う限りにおいて、$\sqrt{2}$ を保存するには無限の状態が必要となる。無論現実に無限の状態を用意できるわけもないため、必要な精度を以って我々はデータを丸める。つまり、切り捨てたり切り上げたりして表現可能にした近似値を $\sqrt{2}$ の値として保存する。

実際には表現可能な数の一覧を尺度として準備しておき、このうち最も近い値として保存する。保存された値のことを量子化値と呼ぶ。量子化のレベル（尺度）は線形軸とする場合が多く、その幅こそ異なれど表現可能な数値の個数は変わらない。勿論、尺度を細かくするほど必要なデータも大きくなる。同じデータであっても線形 256 段階に分けるなら 8 ビットが必要となるが、これが線形 64 段階でいいなら 6 ビットで問題ない。この尺度はデータによって必要な精度等によって変わる。n ビットに量子化する場合、そのレベル値は 2^n 個になる。例えば、標本化の折に例を挙げた CD-DA 規格は 16bit=256 レベルで、0db から 96db までの振幅を対象に量子化している。

まとめれば、量子化というのは、規格等によって用意された 2^n 個の尺度のうち最も近い値に標本値を丸める行為をいう。

2.4.1 量子化雑音

ここまで読んで明らかな通り標本値と量子化値には誤差が生じる。この誤差を**量子化雑音**と呼ぶ。アナログデータをデジタルデータに変換する際に量子化雑音は避けては通れない問題である。

量子化雑音がどれぐらいになるか見てみよう。最大値 H、最小値 L のデータを n ビットで量子化した場合、その各データの幅 D は

$$D = \frac{|H - L|}{2^n - 1} \tag{2.6}$$

で与えられる (レベル値=目盛りが全部で 2^n 個あるため、その間を等分したもの)。今、最小値を基準にデータを表すとすれば、量子化値 Q は

$$Q = L + Dk \ (k = 0, 1, \cdots, 2^n - 1) \tag{2.7}$$

と記述できる。したがって標本値 S との誤差は

$$|S - Q| = |S - (L + Dk)| \tag{2.8}$$

となる。勿論、k は誤差が最小になるように選ばれる=標本値に近い側に丸められるのが普通であるので、これを仮定すると

$$|S - Q| = |S - (L + Dk)| \leq \frac{D}{2} \tag{2.9}$$

であることがわかる。

量子化雑音を小さくする、つまり標本値の再現精度を上げるためには当然ビット数を増やすことになる。しかし、量子化レベルを 1bit 増やすということは、全ての標本値を表すのに、各々1bit ずつ増やして表現することとなるため、無闇に精度をあげるとデータが肥大化してしまう。このため、データの特性に応じて必要限度の bit を確保するのが一般的である。

2.5 量子化値の符号化

先の標本化・量子化を経てアナログデータをデジタルデータに変換できた。この後の保存は勿論先に学んだデジタルデータと同じように符号化して保存することとなる。

このとき、量子化値の性質によっては用いる符号を考慮して圧縮するなどの場合もある。ランレングスにより圧縮を書ける場合もあれば、連続である以上似たような値が続くと考えられることから差分を抽出し、これにエントロピー符号を適用するなどの手法が考えられる。

2.6 値の復元

逆に、符号化されたデータから原データを復元することを考える。基本的には、ここまでの手順を逆行するだけである。

最初に符号を読み取り、量子化値に戻す。これは実質的に（正しい手順で）符号を読むだけである。次に、量子化値を標本値として周波数スペクトルを求め（勿論、量子化値と標本値は別のものである。だが、量子化の手順で見たとおり、量子化値と標本値の差異は量子化雑音として捨てられたものであり、元の標本値をそのまま求めることは出来ない。そのため、量子化値=標本値の近似値を元の近似値として扱うのである。)、標本値以外の箇所における原データの値を計算する。その値を必要な形式で出力すれば、原データが復元できたと言えよう。

演習問題

1 光・音・運動の 3 データ以外の何らかのデータについて、デジタルへの変換なしに記録あるいは保存する方法を例示せよ。

2 文中に出てきた CD-DA においては、44.1kHz のサンプリング周波数で 16bit 量子化している。また、ステレオ再生とするため、原データは 2 つに分かれており、その各々が先の通りサンプリングされている。

(a) 1 秒あたり何 bit の容量が必要となるか (ビットレート) 計算せよ。

(b) 長さの異なる CD-DA 音源の Wave ファイルを 4〜5 個用意し、その再生時間（秒）と容量（ビット）のグラフを作成してみよ（圧縮されている Wave ファイルもあるので、長さに対してあまりにサイズが小さい場合は除外すること)。また、そのグラフを直線で結んだときの傾き (ビットレート) と切片 (Wave ファイルのヘッダサイズ) を求めよ。(Wave ファイルを作成する手っ取り早い方法として、適当な CD アルバム 1 枚を Wave ファイルで取り込む方法を挙げておく。)

3 量子化雑音による誤差が最大 0.05 ℃に収まるように、気温のデータをサンプリングしたい。期待されるデータの最高気温が 60 ℃、最低気温が-90 ℃であるとき、量子化値の表現には何 bit が必要となるか。

第II部

電気伝送の方法

第 3 章

電気伝送の方式

データを符号化して保存できたら、これを相手に送ったり、あるいは受け取ったりする。これがまさしく通信そのものである。最も単純な方法は物理的に媒体をやり取りする—**スニーカーネット**—だが、空間や時間の制限故に常にこれが最適とは限らない[1]。電気的にデータをやり取りできれば、ごく短時間で遠く離れた場所への通信も可能となる。電話・ラジオ・テレビなどがごく身近な例といえる。

電気的なデータのやり取りには電気的な接続が必要となる。先の例でも、電話やテレビはそれぞれ電話線・アンテナケーブルなどで電気的に繋がっており、ラジオは電波で電気的に繋がっている。本章ではこの「接続」について学んでいこう。

3.1 伝送路に求められる性質

電気通信においては、アナログデータをそのまま電圧変化などの電気的信号に変えて伝送するケースも考えられるが、多くの場合符号化したデータを伝送するデジタルデータ伝送となる。ここで用いられる符号は一般に 2 進符号である。これを送るために、どのような性質があれば望ましいだろうか。

電気的な信号をやり取りする場合、電圧変化が信号に対応する。このとき、元々流れている何らかの電気 (直流のときもあれば交流のときもある) からの変化が信号に対応する。この変化が見やすいこと、つまり「どこから信号が始まるか」がわかりやすいということが通信には都合が良い。また、安全のために電圧変化によって遮断されてしまう場合 (遮断特性) があるが、これに影響されがちな通信路・方式は不都合である。他にも、伝送時にいつも雑音が入るような伝送路では伝送が正しく行われないし、雑音や遮断の状況が見えないようでは運用しづらい。

これらをまとめると、次のような性質がある伝送路が望ましいと言えるだろう。

[1] ここでは「最適とは限らない」と書いたが、最適な場合も勿論ある。データとしてやり取りできないもの：例えば物理的なプレゼントなどは言うまでもないが、データにおいてもスニーカーネットが最善となる例は少なくない。用意されている通信路に対して送るデータがあまりに膨大な場合は宅配便などで記憶媒体を直接送付するほうが良い場合も多い。「新幹線に目一杯 HDD 積んで走らせるのが最も伝送速度が速い方法だ」などというジョークもあるが、電気的に記録されたデータのやり取りとしてスニーカーネットは決して過去の遺物ではないのである。

伝送路に求められる性質

- 情報の始点終点の検出などが容易である
- 遮断特性の影響を受けにくい
- 雑音に強い
- 回線が容易に監視できる

3.1.1 雑音

先に**雑音**という言葉を用いた。雑音というと例えば録音したものに余計な話し声が入っている場合などを想像するだろうが、通信においても同様である。混入することで通信の妨げとなるような伝送したい信号以外の (=希望しない) 不規則性信号を雑音というのである。雑音に強い特性というのは**信号対雑音比** (S/N 比 Signal-to-noise ratio) が小さいということに対応する。S/N 比とは読んで字のごとく、信号に対しどの割合で雑音が入るかという雑音の指標である。勿論、これが 0 に近いほうがより良い回線である。

以下、雑音の例を見てみよう。

白色雑音・有色雑音

電気的な理由などで全ての周波数帯に等しく雑音が出る場合、これを**白色雑音**という。対して、周波数帯に偏りがある場合は**有色雑音**と呼ばれる。

インパルス性雑音

自動車のエンジンの点火系統などから発生することのある、瞬間的かつ大きな雑音のことを**インパルス性雑音**と呼ぶ。電気の突然の遮断などによって発生することもある。先に書いた「遮断特性の影響を受けにくい」というのは、インパルス性雑音が発生しにくいという条件の一つである。

干渉信号・漏話

通信路というのは、普通複数人で共用していたり、隣あっていたりするものである。このとき、別の通信で使用している信号が漏れてきて混入してしまうことがある。これを**干渉信号**と呼ぶ。無線通信で近隣から同一周波数の電波があった場合などの干渉が代表的な例である。また、複数の電気線が束ねられている場合などは信号が別ケーブルから漏れ聞こえてくることもある (漏話・クロストーク)。

受信信号の歪み

通信時の中継器などでは、電気信号を増幅させるものが含まれることがある。このとき、非線形な入力特性をもつ増幅回路、つまり増幅後の電圧と増幅前の電圧の比が一定値にならないような増幅回路が存在すると、受信信号を歪めてしまうことがある。特に (あとで説明

する) 帯域伝送において、入力信号の周波数の和・差になるような別周波数の信号が生成されてしまう場合などが挙げられる。

3.1.2 通信の品質

通信路を実際に用意する場合、先に書いたような特性が水準以上であることが求められる。これらを評価する技術的な指標として、接続基準・安定基準・伝送基準の 3 つの基準が定められている。

接続基準

接続基準とは、サービス要求がどの程度迅速・確実に相手に接続されるまでの時間の基準である。例えば、電話であればダイヤルしてから最大どの程度の時間で相手に接続されるべきかなどを示した基準である。問い合わせなどでの「何日以内で返信」などというのも広い意味では接続基準と言える。通常は時間の単位で評価する。ネットワークではタイムアウト値 (=接続がこの値以上ないと通信が遮断されているとみなす) がこの基準に関係した値である。

接続基準は短いほど品質が良いが、短いということは通信相手側の反応を上げるということでもあるので当然システム側へ負荷がかかる。このため、システム負荷として許容可能な範囲で短く設定するのが基本となる。

安定基準

安定基準とは、サービスそのものを提供できるかどうかの信頼性に関する基準である。通常、稼働率または不稼働率が使われる。稼働率とは「稼働する平均 (見込み) 時間」を「稼働が希望される期間」で割った比率で、これが高いほど希望されているうちでのサービスそのものの提供時間が長いということである。回線の場合、サービスとは通信であり、その回線を用いて通信できる時間の割合を示していると言って良い。

注警報の伝達や緊急通報など、「止めることが許されない」サービスにおいては、稼働率を高めるために予備設備を準備するなどの冗長化を行わなければならない。一方、人命などに関わるわけではないネットゲームなどでは、随時のメンテナンスなどでサービスを停止することも許され、稼働率はプレイヤーが飽きない程度であれば問題ない。このように、安定基準はシステムの特性に応じて特に水準が変わりやすい基準と言える。

伝送基準

伝送基準とは、サービス提供中の情報伝達の正確性に関する基準である。アナログ通信であれば雑音の項で扱った S/N 比が、デジタル通信であれば送った bit のうち何 bit がどの程度の確率で誤るかという符号誤り率が使われる。インターネットなど世界的なシステムにおいては、通信システムの電気的な特性を勘案した国際的な勧告値が定められている。

3.2 伝送路の接続と通信方式

先に書いたような伝送路を用意し、2つの端末 (等) を接続すれば、電気的には伝送が可能になる（勿論、その後で設定等は必要になるが)。だが、伝送には「方向」がある。つまり、送る側と受け取る側がいて初めて伝送が成立するのである。そのため、伝送路というのはその通信の状況によって「どちら向きに流れるか」を考える必要がある。伝送路自体に向きがあるとは限らないが、通信のときには必ず向きがある。

現実世界の例として、エスカレーターを考えよう。エスカレータの中には「上りの分1本しか設置していない」ものもあれば「上り・下りが定まった1セットが設置されている」ものもあるし、「時間によって上りか下りが変わるものが1本ある」というようなケースもある。通信でも同様で、上り専用1本しか設置していないものもあれば、上り下り1セットのものもあるし、上り下り兼用1本という場合もある。

3.2.1 単方向通信

通信路が定められており、その両端の一方が送信・他方が受信で固定されたままの通信方式を**単方向通信**と呼ぶ。一つの糸電話で一方がずっと話している状況がこれに当たる。例えば、パソコンのディスプレイはパソコンに何かデータを送ることはなく、受信したものを表示するだけであるが、この通信が単方向通信と呼ばれる。より一般の例では、建物内等での放送やラジオ、電波時計の電波受信なども単方向通信と言える。歴史的な例では狼煙なども単方向通信であろう。

単方向通信では、通信路1本を敷設し、送信機・受信機を接続すれば通信が行われ、送信側・受信側の交代などを考える必要がない。このため準備が容易であるという利点がある。

3.2.2 半二重通信

単方向通信と同じように一本の通信路を定めた場合でも、その両端の送信・受信を交互に入れ替えて利用すれば両方向への通信を行うことができる。これを**半二重通信** (half duplex) と呼ぶ。一つの糸電話で喋り終えるたびに交互に耳と口を入れ替えるような運用である。この例からもわかるとおり、一方の送信中は他方が受信するしかなく、同時の送信や同時の受信を行うことが出来ないのが半二重通信の特徴である (図 3.1)。

電波や赤外線など、無線双方向通信は半二重通信によるものが多い（後に説明する FDD を用いた全二重通信のものもある[2])。これは、同じ周波数の電波で信号を送る場合や同じ場所で赤外線通信を行う場合、双方が同時に送信したら混信・干渉してしまうためで、結果として通信路を1組しか用意できないということになる。先の糸電話とあまり変わらない例であるが、トランシーバーは一方が話していると来た方は話すことが出来ないため、半二重通信である。

半二重通信であっても、双方向の通信に十分な太さの回線で十分に短い時間で送受信を切り替えると、見かけ上は同時に送受信が行えるようになる。この方法を**時分割複信** (Time Division Duplex, **TDD**) と呼ぶ。WiMAX などがこの方式でサービスを提供している。

[2]実用化されてはいないが、無線での（後に説明する FDD を用いない）全二重通信実現も研究されている。

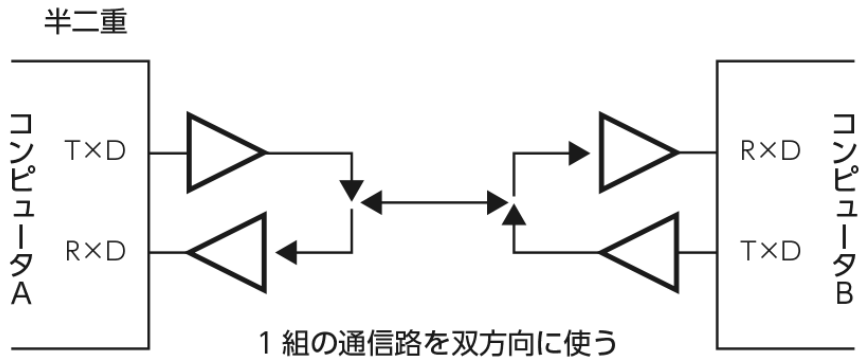


図 3.1: 半二重通信の例：ThinkIT 新人エンジニアが知っておきたいネットワーク基礎知識より引用

3.2.3 全二重通信

半二重通信とは異なり、そもそも線を 2 本用意するなどして双方が同時に送受信できるように通信を行えるようにした通信方式を**全二重通信** (full duplex) と呼ぶ。先からあげている糸電話の例で言えば、糸電話を 2 組用意して、耳・口両方に対応するように当てることで同時会話を可能にしたものである。単方向 2 組であるため (図 3.2) 用意するべき設備自体が多くなるが、通信速度などの面でもっとも良い効果が得られる。

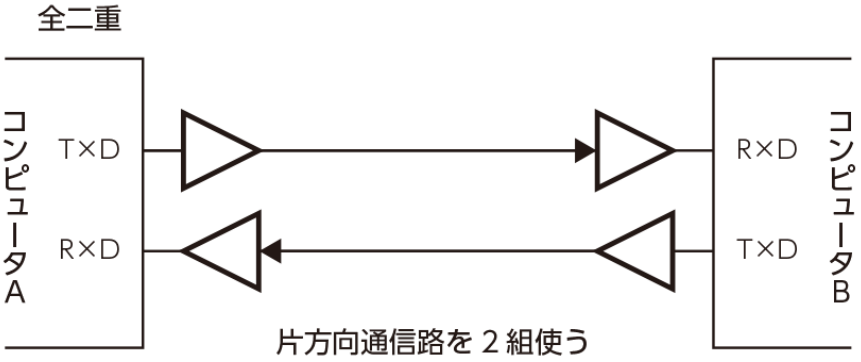


図 3.2: 全二重通信の例：ThinkIT 新人エンジニアが知っておきたいネットワーク基礎知識より引用

現実世界の例では、固定電話が全二重通信を行っている。また、半二重通信の後に書いたが、LTE による通信などは一つの伝送路で全二重通信を実現している。この LTE の例のように、一つの伝送路で全二重通信を実現する方法がある。先に挙げた TDD も見かけ上一つの伝送路で全二重通信を実現しているという意味ではその方法の一つである (厳密には半二重通信であるが)。

周波数分割複信

伝送を行うときにはある周波数を持った搬送波というものを用いることが多い (これについては後の章で詳しく見ていく)。このとき、異なる周波数の搬送波を用い、一方を上り方向・他方を下り方向と使い分けることで一つの伝送路で全二重通信を実現できる。この方法を**周**

波数分割複信 (Frequency Division Duplex, **FDD**) と呼ぶ。先に挙げた LTE に限らず、無線通信で全二重通信を実現する場合は、TDD で見かけ上の全二重通信にするか、この FDD で全二重通信を実現する。有線では、ADSL が代表的な例と言える。ADSL では多くを下りの周波数に割り当てて、通常ダウンロードの方が多い回線利用者に適合したサービスを提供している。

エコーキャンセラ方式

電話などの音声通信で同一の回線を使った場合、エコーがかかってしまうことがある。そこで、信号からエコーを除去して通信することで雑音を減らす手法が開発された。これを通常のデータ通信などに応用したものが**エコーキャンセラ方式**である。信号を分けるハイブリッド回路などの仕組みを設け、送信信号と受信信号を分離する。そして送信信号のエコー分を除去して受信すれば、受信信号が得られる。欧米の ISDN 加入者線などで使われている方法である。また、複数周波数帯を用いた通信において、周波数の僅かな重なり (スペクトルオーバーラップ) を除去する用途などにも用いられている。

演習問題

1 マイクには、ある方向・角度からでないと声が入りづらくなっている指向性マイクと、全方向からの声が等しく入る無指向性マイクがある。これを適切に使い分けることで雑音を減らすことができるが、それぞれどのような場面で使うのが適切か例示せよ。

2 レンタルサーバにおいては、稼働率 99.9%などと喧伝している業者もあり、これが達成されない場合は一部返金対応などが行われるなど、宣伝の材料としても用いられている。稼働率を基準にサービスを考える場合は、どの程度の時間に対応するか計算し、許容される停止時間を検討して決める必要がある。

(a) 具体的に稼働率 99%,99.9%,99.99%の各レンタルサーバの年間停止時間を計算してみよ。

(b) 年間の停止可能秒数が入力されるとき、期待される稼働率を出力するプログラムを作成せよ。

3 次の各ケーブルによる伝送について、半二重通信か全二重通信かを調べてみよ。

(a) USB2.0 ケーブル

(b) USB3.0 ケーブル

(c) UTP ケーブル (現在一般に使われる LAN ケーブル)

(d) 同軸ケーブル

第 4 章

電気信号の同期

電気的に接続しても信号を送るだけではどこからが意味のあるデータか、信号か見分けがつかない。例えば電話は基本的に常に電話線や基地局と接続できる状態に有り、その呼び出しがあったとき初めて音を鳴らすわけだが、どこからどの電波が呼び出しなのかということがわからなければ適切に受信できない。また、受信できたときにはビットレベルで適切に区切れる必要がある。つまり、「データ内部のビットを合わせる」「データの開始や終了を合わせる」という行為が必要になる。前者を**ビット同期**、後者を**ブロック同期**と呼ぶ。

本章では、これら 2 種類の同期について学んでいこう。

4.1 ビット同期

冒頭に示したとおり、ビット同期とは、ビットごとにタイミングを合わせる同期方法である。

ビット同期には、大きく分けて**連続同期方式**と**非同期方式**の 2 種類がある。

4.1.1 連続同期方式

連続同期方式とは、伝送データと別に同期パルス信号を送り続けることでビット位置を知らせる手法である。同期パルス信号の 1 周期が 1 ビットを表すとして情報を読み取る。図 4.1 は、連続同期方式の伝送信号と同期パルスを分割して示したものである。横軸より上側に伝送信号を、下側に同期パルスを示したものである。同期パルスの 1 周期毎に区切り、この 1 周期の間の電圧を見て、電圧が高い状態にある時間の方が長ければ 1、低い状態にある時間の方が長ければ 0 とする。図 4.1 の例では 10100101 というビット列が取り出される。無論、このように 8 ビットだけでなくより長いデータも同様にして取り出すことができる。

この方式を実現する単純な方法としては 2 本の伝送路を用意するということがあるが無駄が多い。このため、現在は 1 本の通信回線上にデータと同期パルスを重ねて送出する方法が広く用いられている。また、重ね合わせて送出するだけであり、元のデータにビット等を付加する必要がないため、長いデータを送出する際にも伝送効率がよい。

4.1.2 非同期方式 (調歩同期方式)

非同期方式は**調歩同期方式**とも呼ばれる。これはデータの始まりを示すビットを付けてビット位置を知らせる手法である。

非同期方式の場合、通信が行われていないときには常に一定の電圧がかかっている。ここで、スタートビットと呼ばれるビットが電圧 0 として流れてくる。これがデータの始まりで

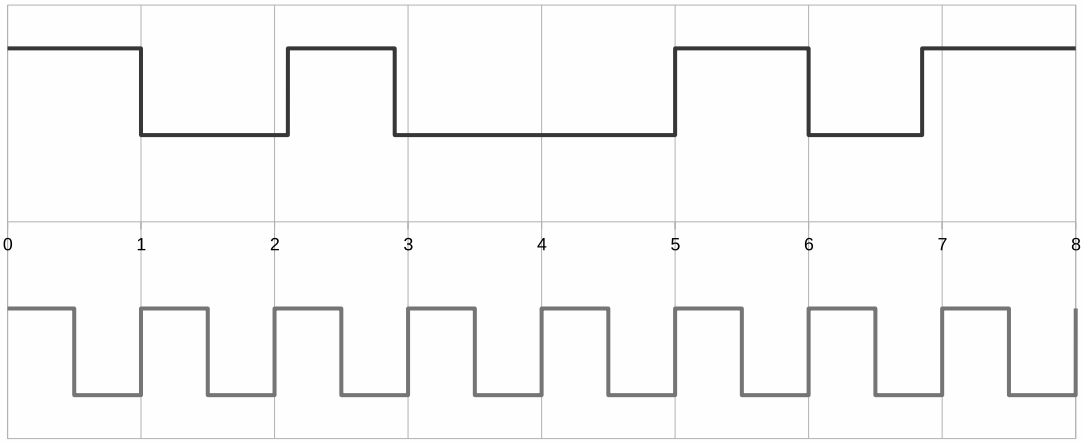


図 4.1: 連続同期方式：伝送信号 (上) と同期パルス (下)

ある。この後、予め定まった長さ毎に電圧を解釈し 0 または 1 のビットに割り当てていく。この部分をデータビットと呼ぶ。データビットを一定数 (通常 7bit か 8bit) 読んだら、次の同期のために電圧を 1 に戻す。この 1 に戻すビットのことをストップビットと呼ぶ。このように、予め長さを決めた上でデータの始端・終端で同期を取るのが非同期方式である。非同期方式とは同期式と異なる（同期式ではない）という意味であり、「同期処理自体が行われない」という意味ではない点に注意されたい。

図 4.2 に、8bit の調歩同期方式の例を示す。これは図 4.1 と同様、10100101 というビット列を送った例である。平常時は 1 に相当する電圧がかかっている (-1 から 0)。次いでスタートビット (0 から 1) が来て、8bit のデータビット (1 から 9) が届く。ここで、2 から 3 のように途中で値が変わっている場合、長い側の値を採用するのは連続同期方式と同様である。その後、ストップビットが届き (9 から 10)、平常の 1 の状態に戻る (10 から 11)。次のデータが開始したらまたスタートビットが届く (11 から 12)。なお、連続したデータを送る場合、平常に戻る箇所 (10 から 11) はないこともある。

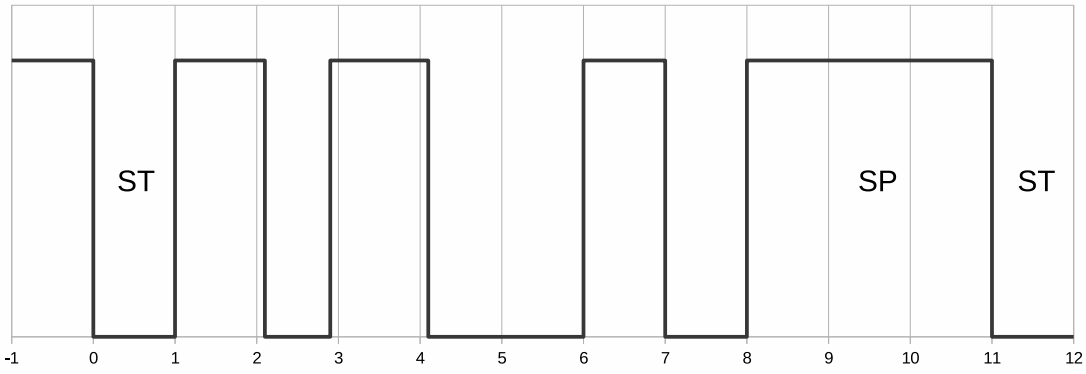


図 4.2: 8bit の調歩同期方式:ST はスタートビット・SP はストップビット

調歩同期方式では一定のデータビット毎にスタートビットとストップビットが付加されるため通信の効率が落ちてしまう。一方で、同期パルス形式に比べより自由なタイミングで、場合によっては安価に環境を用意できるのが利点である。

4.2 ブロック同期

ビット同期によって各ビットが正しく識別できたとしても、データの開始や終了、区切りがわからなければ正しく解釈できない。そこで、データの区切り位置を送受信双方で合わせるのがブロック同期である。先の調歩同期方式ではビット列の解釈を合わせるとともにスタートビット・ストップビットによってデータの始点・終点を定められたため、調歩同期方式はブロック同期も行えるが、連続同期方式ではそうは行かない。

ブロック同期方式として**キャラクタ同期方式**と**フラグ同期方式** (**フレーム同期方式**) を紹介する。

4.2.1 キャラクタ同期方式

キャラクタ同期方式は、その名前の示すとおり、文字の情報を送るときに多く使われる手法である。このため、キャラクタ同期方式は 1 オクテット毎にデータを送付する。この方式では、SYN(シン) キャラクタと呼ばれる制御コードを送ることでデータの先頭位置を示す。ここで、一般に SYN キャラクタは 00010110 で表される文字である (通例、通信時には低位のビットから送るため、受信順序は逆となる)。同期の確実性のため SYN キャラクタは 2 度以上続けて送られるのが普通である。

送信側ではデータを送信していないとき、常に SYN キャラクタを送り続ける。一方受信側では、この SYN キャラクタを監視することによって同期をとる。つまり、bit 単位で見ていた調歩同期方式を octet 単位に変えたような動作である。順に送られ続けるビットから SYN キャラクタを見出し、ここをデータの区切りとすることで同期が確立する。一旦同期が確立すると、octet 単位でデータを読み込み続け、SYN キャラクタ以外のパターンを受け取ったときにこれをデータとして解釈していく。

尚、キャラクタ同期方式は必ずしも文字だけに使われるわけではない。文字以外のデータも octet 単位に区切って送信することができる。このとき、SYN キャラクタと同じパターンが混在してしまうことがあるが、これはエスケープ処理として DLE(デル) キャラクタを前に付加して送信する。受信側では SYN キャラクタの前に DLE キャラクタを受信した場合、DLE キャラクタを破棄し、SYN キャラクタとして解釈するのである。

キャラクタ同期方式の場合、調歩同期方式と違って文字ごとに bit を付加する必要がなく、連続してデータを送信できるのが利点である (これはフレーム同期方式にも言えることだが)。

4.2.2 フラグ同期方式（フレーム同期方式）

フラグ同期方式は、あらかじめ決められた同期用のビットパターン（フラグパターンと呼ばれ、01111110 が使われる）を送信データの始めと終わりに付けて送り出す方式である。受信側では、このデータを監視し、フラグを検出すると次のフラグまでの間を一連のデータとして解釈する。キャラクタ同期との違いは、データの区切りを 8 ビットと定めていないため 8 ビットの倍数とは限らない長さのデータを送ることができるという点である。この点を除

けば、SYN がフラグパターンに変わったキャラクタ同期方式と考えても（多少の差異はあれど）間違いではない。

キャラクタ同期方式との差異として挙げられるのはエスケープの方式である。フラグ同期方式ではフラグパターンに 1 が 6 連続しているという点に着目し、送信時に 1 が 5 ビット連続したら強制的に 0 を挿入する。逆に、受信側では 1 が 5 ビット連続した後の 0 を取り除く。この決まりによりデータを間違いなく受信することができる。

フラグ同期方式はキャラクタ同期方式に比べエスケープのビット数が少ないことや可変長で送信できることなどから、伝送効率が良い手法である。

演習問題

1 図 4.3 は bit 同期の単位時間を 1 とした場合の入力電圧を示したものである。入力電圧 1(=0.5 以上) が bit の 1 に、0(=0.5 未満) が bit の 0 に対応するとき、この図から抽出されるべき 8 ビットのビット列を答えよ。尚、入力電圧データにはノイズがある点に注意せよ。

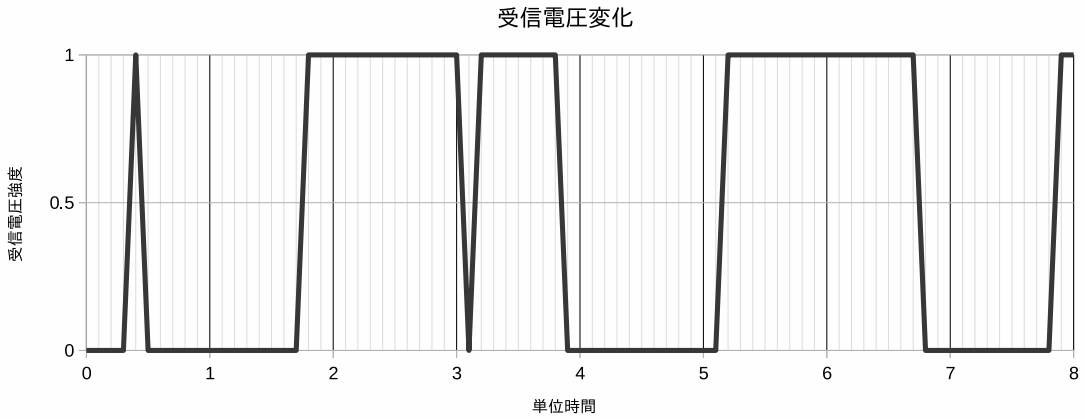


図 4.3: 入力電圧データ

2 7bit 単位でデータが区切られる調歩同期方式で 70kB のデータを送信したい。このとき、調歩同期方式で付加されるビットを含めた実通信量は何バイトになるか。

3 ある通信においてブロック同期が適切に行われなかった結果、最初の 4bit が読まれず、データ列が 16 進表記で”1A 2B 4D F0”となった。最初の読まれなかった 4bit が 16 進表記で C であり、最後の 0 が不要なデータであった場合、この正しいデータ列はどのようになるか。

第 5 章

ベースバンド伝送

前章では、電気信号が届いた際に問題となる「同期」について記した。では、実際に情報を電気信号に乗せ、送るにはどういう方法を取ればよいだろうか。本章および次章ではまさしく通信の本丸と言える信号の伝送について説明する。本章では伝送方式を概観し、その基本となる伝送方式のベースバンド伝送方式について解説を行う。

5.1 信号伝送方式

コンピュータに記録されたデータは bit により構成されている。この構成する 0 と 1 を意味するように電圧の高低を二元状態に当ててやれば、電圧が情報を表していると言える。この電圧の変化はパルス[1]であり、そのまま電圧変化として送信することもできる。このように、信号を変換した波をそのままに伝送する方式を**ベースバンド伝送**または**基底帯域伝送**と呼ぶ。

一方、波をそのまま送るのではなく、他の**搬送波** (**キャリア**とも) に載せて信号を送る手法もある。この手法を**ブロードバンド伝送**または**帯域伝送**と呼ぶ。ベースバンドよりも装置などが複雑になるが、搬送波に載せて信号を送ることによる様々なメリットが必要となることも多いためこちらの方法が取られるものも多い。

本章では、ベースバンド伝送方式の仕組みと利点について見ていく。帯域伝送方式については次章で取り扱う。

5.2 ベースバンド伝送方式

ベースバンド伝送は、古くから有線通信に使われてきており、現代でも LAN の通信や固定電話の一部などで利用されている。古くから使われる理由は、単純な方式で実現が容易であるからと言える。

5.2.1 ベースバンド伝送方式の要件

ベースバンド伝送に必要とされる要件は大きく次の 3 点である。

[1] 短時間に急峻な変化をする信号のこと。また、通信等の分野においては矩形的な変化をする波もパルスと呼ぶ。

ベースバンド伝送方式の要件

- 連続性の抑圧：0or1 が連続するとタイミング抽出がしにくく (=同期が難しく) なるため、連続しすぎないこと。
- パワースペクトルの集中性：電圧の変化があまり頻繁 (高周波) であると、伝送路によって伝えられないことが起こるため、電圧の周波数がある一定の幅に集中しているのが望ましい。
- 直流平衡：直流遮断特性 (機械の保全等のため直流電圧を通さない特性) による遮断の影響が少ないこと。

これらの要件のため、後に上げるような様々なベースバンド伝送方式が有り、回路や機械の特性に応じて利用方式が変わる。

5.2.2 ベースバンド伝送方式の特徴

ベースバンド伝送は単なる電気の伝送であるため、長距離でなければ比較的単純に装置を準備できる。このように、回路規模が小さいのがベースバンド伝送方式の特徴である。

一方で、一度に一つの信号しか送れない、雑音に弱い、無線通信では実現が難しいなどの難点もある。特に、最初の「1 つの信号しか送れない」点は実際の通信において大きな欠点となる。帯域伝送方式を利用することが多いのはこの難点の解消のためであるが、逆に小規模な伝送では不要な（気にしなくてよい）問題であるためにベースバンド伝送も現役で使われていると言える。

5.2.3 ベースバンド伝送方式の分類

ベースバンド伝送方式は、どのような電圧状態を 1/0 に対応させるかによって分類が分かれる。以下、その各方式について記述していく。尚、通常は電圧 0 の状態と電圧 E ないし $-E$ の状態を用いることとなるが、以下は便宜のため 0 と 1/-1 を使っているとして記述する。

RZ と NRZ

ある bit を表す単位時間の電圧のうち半分を電圧 0 に当てる方式を **RZ 方式** (Return to Zero 方式) と呼ぶ。対して特段そのような措置を取らない手法もあり、この場合は **NRZ 方式** (Non Return to Zero 方式) と呼ぶ。RZ 方式はビットパルスの半分が 0 となるのでビット同期に都合が良い。 対して NRZ 方式は電圧の変化が少なくなるため、電圧の変化が高周波でなくなるという利点がある。

両極方式と単極方式

先の RZ 方式/NRZ 方式に加え、電圧変化にどの幅を使うかにより、**両極方式**と**単極方式**の 2 種類の方式がある。前者は正極 (電圧+1) と負極 (電圧-1) を各々1/0 に割り当てて利用する。後者の単極方式では正極または負極の一方と電圧 0 の状態を 1/0 に割り当てる。

図 5.1 に、単極・両極の RZ/NRZ 方式の例を示す。

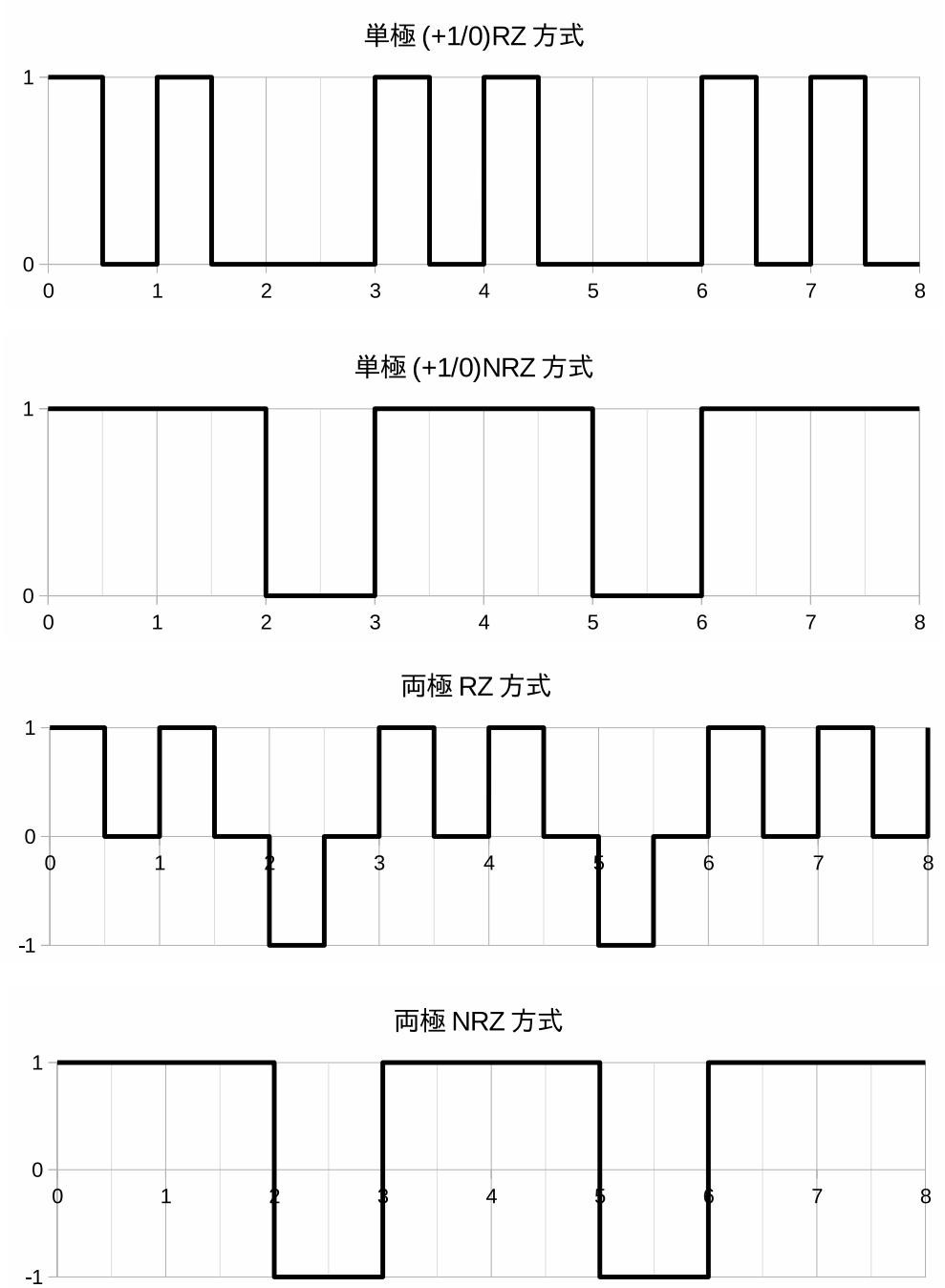


図 5.1: 単極・両極の RZ・NRZ 方式における符号語 11011011 の電圧変化。横軸の数値はビットの単位時間を示す。

バイポーラ方式

バイポーラ方式 (bipolar system) では、正極/負極をともにビット 1 に当て、電圧 0 を 0 に割り当てる。つまり、電圧の絶対値が 1/0 に対応する。また、1 を送り出す場合は「前回 1 を送ったときと異なる側の極」を使う。つまり、正極を使って 1 を送った後は（その後にいくつ 0 があろうとも）次の 1 は負極を用いて送る。逆もしかりである。この手法では、直流

遮断特性の影響を取り除くことができるが、0 が連続した場合にビット同期が取り難いという難点もある。

図 5.2 に、バイポーラ方式の RZ/NRZ 方式の例を示す。

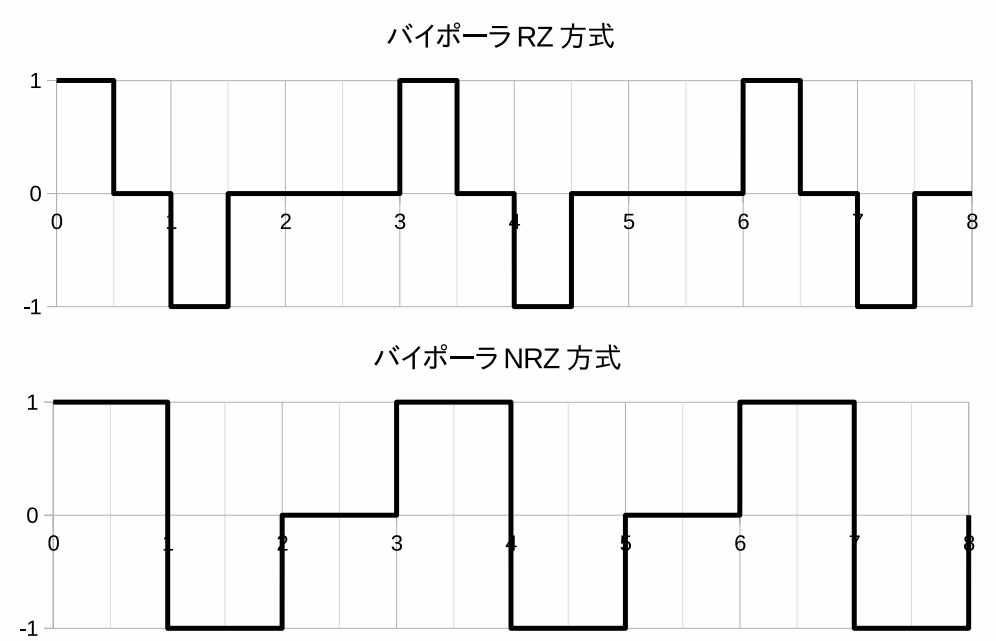


図 5.2: バイポーラ RZ/NRZ 方式における符号語 11011011 の電圧変化。横軸の数値はビットの単位時間を示す。

差分方式

差分方式 (differential system) は前の単位時間の電圧から電圧に変化があった場合は 1 を、なかった場合には 0 を示す方式である。例えば、ある初期状態の電圧が-1 であり、次の単位時間が 1、その次の単位時間でも 1 だったとすると、これは 10 という符号語になる。

図 5.3 に、差分 NRZ 方式の例を示す (RZ 方式では各単位時間後半を 0 にした図となる)。

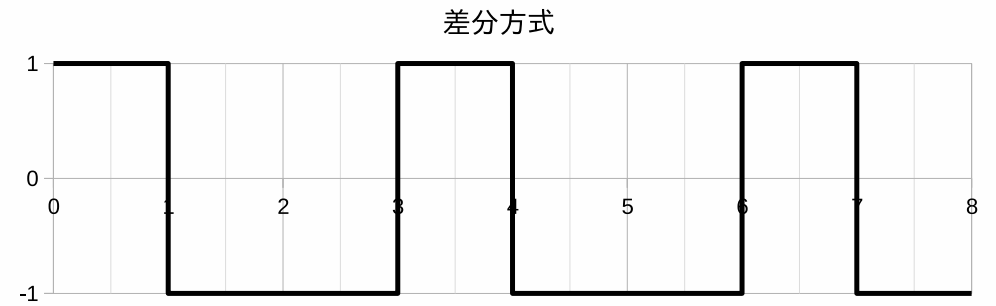


図 5.3: 差分 NRZ 方式における符号語 11011011 の電圧変化。グラフに出ていないが初期状態は-1 としている。横軸の数値はビットの単位時間を示す。

ダイコード方式

ダイコード方式 (dicode system) では現在のビットが 1 から 0 へ変化する場合-1,0 から 1 へ変化する場合 1、変化しない場合 0 の電圧によって表す。初期状態のビットが 0 であるとして、1 の電圧,0 の電圧,-1 の電圧と続いた場合の符号語は 110 となる。

図 5.4 に、ダイコード NRZ 方式の例を示す (RZ 方式では各単位時間後半を 0 にした図となる)。

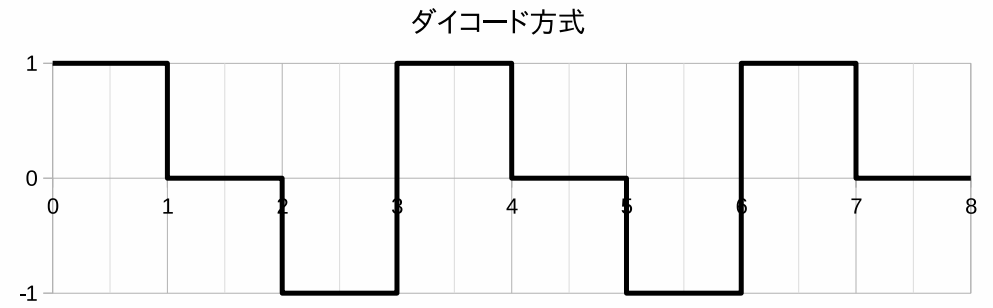


図 5.4: ダイコード NRZ 方式における符号語 11011011 の電圧変化。グラフに出ていないが初期状態は 0 としている。横軸の数値はビットの単位時間を示す。

マンチェスター/ダイパルス方式

マンチェスター方式 (Manchester system) あるいは**ダイパルス方式** (dipulse system) は 0・1 で 180 度位相の違うパルスを送出するものである。通常、ビットの中心で位相を反転させる（このことから、split-phase 符号などとも呼ばれる)。例えば最初-1 でスタートしてビットの中心で電圧を 1 に変えた場合 1、その逆に最初 1 でスタートして後半を-1 に変えた場合は 0 とするのである。位相の反転のおかげでビット同期を取りやすく、後述する Ethernet LAN で用いられている。

図 5.5 に、マンチェスター方式の RZ/NRZ 方式の例を示す。

5.3 伝送速度

ベースバンド伝送の場合、**伝送速度**は約 50kbps から 16Mbps である。この速度でよく使われる単位”bps”についてここで付章的ながら説明しておく。

bps は bit/sec と書くこともあるが、bit per second(bit 毎秒) という単位である。例えば、100bps は 1 秒間に 100bit=2 進数 100 桁を伝送する速度ということである。似た単位に Bps というのもある。B を大文字にした場合は一般に Byte per second(Byte 毎秒) として 1 秒間に何バイトのデータを伝送するかという単位になる。現在 1Byte=8bit であるので、1Bps=8bps である。

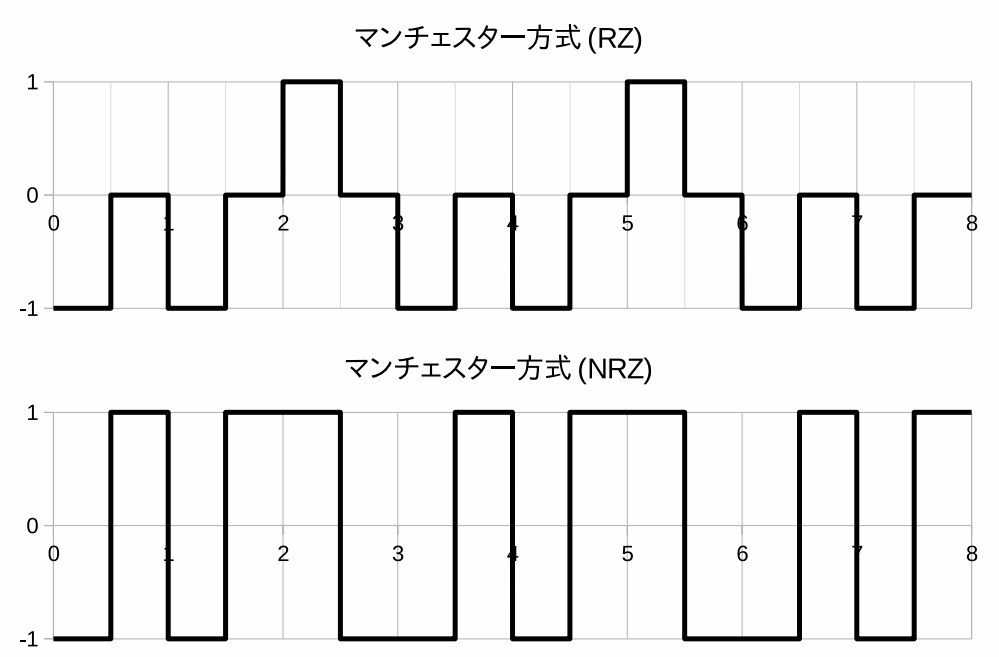


図 5.5: マンチェスター RZ/NRZ 方式における符号語 11011011 の電圧変化。横軸の数値はビットの単位時間を示す。

回線速度測定サービスというのがしばしばあるが、これは大きなファイルを用意して、これを送信するのにかかった時間により速度を計算している。300MB のファイルを 5 分でダウンロードできたとすれば、下りの実効速度[2]は 1MBps=8Mbps となる。

5.3.1 【補足】単位の接頭辞

本節は前提知識の補足である。
既知の読者におかれては飛ばして次節を読まれたい。

bps あるいは Bps(Byte も含む) 単位の接頭辞としては、k(キロ),M(メガ),G(ギガ),T(テラ),P(ペタ) あたりが用いられる。k は 10^3 を示し、以下 $10^6, 10^9, 10^{12}, 10^{15}$ となる。

一方、コンピュータ技術を論ずる場合には、2 の累乗が都合が良いため、$10^3 = 1000$ を $2^{10} = 1024$ に置き換えて用いる場合がある。便宜的に k,M,G,T,P をそのまま用いる場合もあるが、これを明示する場合には ki(キビ),Mi(メビ),Gi(ギビ),Ti(テビ),Pi(ペビ) という単位 (各々$2^{10}, 2^{20}, 2^{30}, 2^{40}, 2^{50}$) を用いる。

時折、外付け HDD をコンピュータに接続した場合、パッケージ記載の値と容量が異なるように見える場合がある。この原因として、先の 10^3 と 2^{10} の違いが出てくることがある。例えば 1TB の HDD は、10^{12} で計算した場合と 2^{40} で計算すると約 100GB,10%もの違いが出る。

[2]標榜されている速度ではなく、実際の速度のこと。

演習問題

1 本章内で説明しなかった方式に差動マンチェスタ方式がある。これは、次のようにして定まる。

- ビットの変わり目で電圧を反対側の極にする (+1 なら-1、-1 なら+1)。
- ビットの値が 0 であれば、1 ビット極電圧を維持する。
- ビットの値が 1 であれば、ビット中間にてその電圧を反対側の極にする。

この差動マンチェスタ方式について、符号語 11011011 を送る場合、電圧変化はどのようになるか図示せよ。

2 本章内で出てきた全方式 (前問の差動マンチェスタ方式を含む) について、符号語 10010011 を送る場合の電圧変化を図示せよ。

3 2000 年台前半のソフトウェア公開サイトでは、代表的な速度 56kbps(モデム), 64kbps(ISDN), 1Mbps(ADSL), 100Mbps(FTTH) について、ダウンロードにかかる時間が表示されていることがしばしばあった。

ダウンロード対象ファイルのサイズが入力されるとき、先の 4 回線と自身の好きな速度の 1 回線でのダウンロード時間を出力するプログラムを作成せよ。ファイルのサイズは小数第 2 位までの数値と接頭辞 (G,M,k,Gi,Mi,ki) 及び B(バイト) で入力されるものとする (10.25MB など。)。なお、入力されるファイルサイズは必ず接頭辞がついてあるものとして良い。

第 6 章

ブロードバンド伝送

第 7 章

誤り制御

ここまでに紹介したような方法で十分な水準の通信を確立できる状況となった。しかしながら、どの方法を取るにせよ何らかの事情で通信に誤りが起こることは避けきれない。そこで、通信において正確性を担保するために、誤りを検知して対処するための**誤り制御** (error control) の手法がある。今回はこの誤り制御について紹介していく。

7.1 誤り制御

人間が直接声でやり取りをする際に、不明瞭な発音等により意図が正確に伝わらないことは往々にして起こりうる。例えば、数字の 1 と 7 はそれぞれ「いち」「しち」と読むわけだが、文脈からどちらかの判断がつかないようなケースでは聞き間違えは致命的となりうる。「1 時からの番組の録画予約をお願いします」と伝えたときに、7 時から録画されてしまうと見たい番組が見られない。同じように、アルファベットを伝えるシーンでも b,d,p などは読みが似ているため、メールアドレスを口頭で連絡するなどの必要があるときには、誤りに注意するであろう。

このように、人間同士が声でやり取りする際に誤りが懸念される場合には、予め対処を取るだろう。簡単な例を挙げると

- わかりやすいように言い換える :「しち」でなく「なな」、「でぃー」でなく「でー」
- 通話表を使う : いろはの「い」、Bravo の「b」
- 復唱して確かめる
- 文脈を付け加えて確認する :「1 時からの○○という番組？」

などは、誤りなく伝えるための手法と言える。そして、これにより誤りが見つかった場合、相手から正しい情報を再度聞き出すなり、明らかに別とわかるなら自分で考えて修正するなりして誤りをなくす。

通信においても、例えば電圧の都合などでビットが不明瞭である[1]、雑音が入った、機器の特性など、様々な事情で誤りが起こりうる。これらの対処を行うのが誤り制御である。とりわけ、致命的な過誤を防ぐためにも、誤り制御は重要と言える。とはいえ、誤り制御にあまり大きなリソースを割くのは難しい。先の人間の例であっても、「1 時からの番組の録画予約をお願いします」の、1 と 7 以外の場所の誤りを逐一チェックする (「からの」じゃなくて

[1] 例えば、CD であればレーザー刻印が十分深くない、電圧がそもそも境界値周辺で振動しているなど。逆に、前者のような刻印をあえて作ってコピーガードとしている例もある。

「までの」ではないか、「録画」でなくて「録音」ではないか、「お願いします」ではなく「お願いしません」ではないか…など) ようなことは普通やらない。つまり、誤り制御には次のような性質が求められる。

誤り制御に求められる性質

- 誤りを十分に高い確率 (100%とまでは言えなくてもそれに限りなく近い確率) で検知できること。
- 大幅なリソースを要しないこと。
- 誤り制御に関する情報自体に誤りが含まれる可能性を考慮すること。また、その誤りが少ない程度に簡潔な情報であること。

この性質を満たすような手法を用意し、誤りを検出すれば、後はこれを訂正するだけである。人間の例でわかるとおり、訂正方法は「相手に訂正してもらう (再送してもらう)」か「自分で訂正する (訂正できるだけの情報をもらっておく)」のいずれかである。前者のように、誤りを検出するだけして、誤った場合に再送してもらう方法を**帰還方式** (**ARQ**,Automatic Repeat reQuest) といい、これに使う符号を**誤り検出符号**と呼ぶ。一方、後者のように、誤りを検出すると共にその情報から訂正まで行う方式を**無帰還方式** (**FEC**,Forward Error Control) といい、これに使う符号を**誤り訂正符号**と呼ぶ。

7.1.1 誤り制御の簡単な例

極めて単純な誤り制御の例を 3 つほどあげてみよう。ファイルをダウンロードする場合を考える。この時、正確なファイルがダウンロードされたかどうかを確認したい。

最も単純な方法として、そのファイルを複数個ダウンロードし、ビット別に比較するという方法が考えられる。十分な数のファイルをダウンロードし、多数決で各ビットを定めていけば (エラー率が十分低い=伝送基準が十分高い前提の下) 正しいファイルが出来上がるだろう。しかし、この方法は本来一つで良いはずのファイルを何度もダウンロードしなければならず、例えば OS の iso ファイルなどサイズの大きいファイルを対象とした場合にリソースを大幅に食ってしまう。

そこで、ファイル自体に関するデータを確認するという手法がある。例えば、ファイルサイズが本来のサイズとバイト単位で一致するかどうか確認するのである。バイト単位で一致しなければ過剰あるいは不足があり、明らかに正しいデータではないから、再ダウンロードを行う。この手法はやり取りするデータがごく小さいことからリソースが守られるが、同一サイズでビットが変わっているという（起こりえそうな）ケースには無力である。

他には、ファイル・フォーマットを確かめるという手法がある。マニアックな域ならバイナリを見るような人もいるかもしれないが、最も簡単な方法ならそのファイル自体を開ける状態にして開いてみる。ソフトウェアなら動作させればいいし、iso ならマウントするなりディスクに焼くなりする。動画や音声なら再生してみれば良い。このとき、正しく開ければおそらく間違ってはいないだろう。この手法は、通信に関するリソースを要求しないが、開

く側でのリソースは決して少なくない。また、サーバに搭載するソフトなど、気楽に試せるものばかりでもない。

ここで挙げた例はいずれも誤り制御であるが、それぞれにリソースや検出できる誤りに違いがある。多くの場合、誤り制御はリソースと対処率のトレードオフと言ってよい。このトレードオフの中で、様々な誤り制御手法が生まれてきた。この後は、順次それを紹介していく。

7.2 誤り検出符号

誤り検出符号は誤り訂正符号よりも通常小回りが利く (サイズが小さい・アルゴリズムが速いなど) ものが多いため、現実的な通信もこちらの方式を採用する場合が多い。また、誤り訂正符号と違い、誤っているかどうかがわかれば対処できるため (後述するが、誤り訂正符号は 1 ビットまでの誤りなら対応できるがそれ以上は対応できないなどの限界がある)、誤りの状況によらない (通信の輻輳等により発生する集中的な誤り (バースト誤り) にも、散発的に発生する単独の誤り (ランダム誤り) にも対応する) という利点もある。

以下、誤り検出符号の何種類かの方式について記す。

7.2.1 パリティビット

データを送る際に、区切りごとの偶奇 (パリティ) を定めてビットを付加してデータを送信する方法ならびにその際に付加するビットを**パリティビット**と呼ぶ。これによる誤り検出は**パリティチェック**という。

最もよくあるケースでは、実データ 7bit 毎にパリティビット 1bit を加え、1byte 毎のパリティを一定にするようにする。1byte 毎のパリティとは、単純にデータ中の 1 の個数の偶奇である。例えば、1byte 毎に偶数になるようなパリティビットを付加する場合、1110001 や 1110111 のように 1 が偶数個ある実データには 0 を付加するし、1100100 や 1111111 のように 1 が奇数個ある実データには 1 を付加する。受信側は、1byte 毎に 1 の個数を数え、これが奇数である場合は再送を要求する。

もちろん、偶奇が同じ場合には誤りと判断されないので、2bit・4bit 等、偶数個の bit に誤りがあった場合には検出できない。また、7bit に 1bit をつける場合は元のデータの 7 分の 8 のデータとなり、15%弱のデータ肥大化となる。

【補遺】パリティチェックの実送信サイズ

先にデータ肥大化を 15%弱と書いたが、これはデータの再送分などを考慮しておらず、単純に送信すべきデータの量を見た場合である。この補遺では、パリティチェックを行う場合、再送分なども含めて実送信サイズがどれぐらいになるのか (再送要求は十分無視できるサイズとして) 考えてみる。

bit 毎の誤り率を p とし、$n-1$bit 毎にパリティビットを付加して $n$bit 毎にデータが区切られるものとする。今、検出できる誤りである、奇数個の誤りが起きる確率 P_{err}[2]は

$$P_{\mathrm{err}} = \sum_{i=1}^{n/2} p^{2i-1}(1-p)^{n-2i+1}{}_n\mathrm{C}_{(2i-1)} \tag{7.1}$$

により与えられる。ここで、検出できる誤りの起きる確率とは再送確率であることに注意する。再送が k 回必要な確率 P_k は

$$P_k = P_{\mathrm{err}}^k(1-P_{\mathrm{err}}) \tag{7.2}$$

で与えられる。したがって 1 データ区切り毎の送信回数 (=1+送信回数) の期待値 E は

$$E = \sum_{k=0}^{\infty}(k+1)P_k \tag{7.3}$$

となる。ここで、P_k は P_{err} を公比とする等比数列であり、当然 $|P_{\mathrm{err}}| < 1$ であるのでこの無限等比級数は収束する。これをもとに計算すると

$$E = \frac{1}{(1-P_{\mathrm{err}})} \tag{7.4}$$

となる。したがって、再送を平均して $\frac{n}{(1-P_{\mathrm{err}})}$bit の送信を行うこととなる。元のデータのサイズは区切り毎に $n-1$bit であるから、倍率でいえば $\frac{n}{(n-1)(1-P_{\mathrm{err}})}$ 倍のデータサイズとなる。

7.2.2 チェックディジット

符号だけでなく、実データにもよく使われる方法として、**チェックディジット**と呼ばれる方法がある。パスワードや ISBN などにしばしば使われる。

チェックディジットは、データの区切り毎に（通常、そのデータを整数値とみて）何らかの計算を行い、その値を区切り毎に置くという手法である。(適当な計算が 1 の個数の偶奇であり、それに一致する値を置くという場合はパリティビットそのものになる。) ただし、チェックディジットは通常ビットレベルではなく、もう少し大きな単位で行われることが多い。

チェックディジットは「適当な計算」によりさまざまな種類がある（ゲームのデータ等の場合、独自形式のものも存在する)。ここでは、代表的なものを紹介しておく。

DR 型・DSR 型

DR(Division Remainder) 型のチェックディジットは、数値をある値で割った剰余を付加する方法である。値としては、7 や 9 がよく使われ、7DR や 9DR と表記する。また、各々後述の 7DSR や 9DSR と合わせて、セブンチェックやナインチェックと呼ぶ。

元データが 9876543 だったとする。このときこれを 7 で割った余りは 5、9 で割った余りは 6 である。したがって、7DR の場合 98765435 というデータを、9DR の場合 98765436 というデータを送ることになる (チェックディジットを末尾においた)。

[2]偶数個のエラーが出る場合は検出されないため無視している。

また、余りそのものではなく、割る値に対する基数の補数を付す場合もあり、これをDSR(Divide Subtract Remainder) 型と呼ぶ。先の例 9876543 について、7DSR でチェックディジットを付加すると (7-5=2 であるため)98765432 となり、9DSR だと 98765433 となる。

運送会社の伝票番号には 7DR が使われている。

モジュラス・ウェイト型

モジュラス・ウェイト型は、元の 10 進整数の各桁に適当な重み (ウェイト) をかけた総和をとり（このときウェイトをかけた時点で 2 桁以上の数が出た場合に、総和をそのままとる場合は一括方式、桁和を用いる場合分割方式と呼ぶ)、別の値での剰余の補数をとってチェックディジットとする方式である。チェックディジットが 2 桁になる場合は 0 にする場合が多い。

具体例として、モジュラス 11 ウェイト 2~7 を用いる、マイナンバーや住民票コード、運転免許証番号で用いられる手法を見てみよう。この方式は、次の手法でチェックディジットを決定する。

1. 数値の各桁に、下の桁から順に 2,3,4,*cdots*,7 の係数をかける。(おさまらない場合はまた 2 に戻る。つまり、7 桁目には 2 を、8 桁目には 3 を $\cdots$ という風にかける)
2. 各桁の結果の総和を求め、これを 11 で割った剰余を求める。
3. 求めた剰余を 11 から引き (=補数を求め)、これをチェックディジットとする。ただし、10 以上になる場合はチェックディジットを 0 とする。

試しに、この方式で 123456789 のチェックディジットを求めてみよう。各桁の値と重み、その積は次のようになる。

桁	1	2	3	4	5	6	7	8	9
重み	4	3	2	7	6	5	4	3	2
桁 × 重み	4	6	6	28	30	30	28	24	18

一番下の桁 × 重みの総和を求めると 174 となる。これを 11 で割った剰余は 9 であるので、チェックディジットは 11-9=2 となる。このチェックディジットを最終桁につけると 1234567899 となる。

マイナンバーの場合は 11 桁の数にこれを適用したものを末尾につけて 12 桁に、免許証番号の場合は最初の 10 桁にこれを適用したものをつけて 11 桁、最後に再発行の回数をつけて 12 桁としている。

さて、先の例は一括方式であったが、分割方式のものもある。クレジットカードなどで使われるモジュラス 10 ウェイト 2・1 分割方式は、次のような手順でチェックディジットを決定する。

1. 数値の各桁に、下の桁から順に 2・1 の係数をかける。(おさまらない場合はまた 3 に戻る。つまり、3 桁目には 2 を、4 桁目には 1 を $\cdots$ という風にかける)
2. 掛けた値が 2 桁になった場合は、その十の位の数と一の位の数を足したものを用いる。(分割)
3. 各桁の結果の総和を求め、これを 10 で割った剰余を求める。
4. 求めた剰余を 10 から引き (=補数を求め)、これをチェックディジットとする。ただし、10 になる場合はチェックディジットを 0 とする。

この方式で今度は 31415926 のチェックディジットを求めてみよう。各桁の値・重み・積と分割結果は次のようになる。

桁	3	1	4	1	5	9	2	6
重み	1	2	1	2	1	2	1	2
桁 × 重み	3	2	4	2	5	18	2	12
分割	3	2	4	2	5	9	2	3

総和を求めると 30 となり、剰余は 0 である。したがって、補数は 10 になるが、この場合チェックディジットを 0 とすることとなっているので、最終的に 314159260 というデータになる。

ここまで見てきてわかる通り、モジュラスの後の数が剰余をとる数、ウェイトの後の数が下の桁からの重みを示す。ISBN13 で用いられているモジュラス 10 ウェイト 3・1 方式の場合は、下の桁からの重みが 3,1 の順となり、総和の剰余ならびに補数の基数となるのは 10 である。

7.2.3 チェックサム

チェックサムはファイルダウンロード等の場合に特によく使われる手法で、単純にデータ区切り中の一部の数値の合計を別途作成し、この合計値が一致するか否かで誤り検知を行う方法である。例えば、4 バイト毎の最後の 1 バイトの合計などである。

チェックサムは計算が単純ながら検出率がかなり高いため、しばしば使われる。また、この手法がハッシュ関数等後に示す手法の元となっている場合もある。一方、計算が単純であるゆえの欠点として、比較的簡単に数値を合わせることができるため意図的な改竄などには無力である。

7.2.4 CRC(巡回冗長検査)

CRC(Cyclic Redundancy Check, **巡回冗長検査**) は多項式による除算の剰余を用いて誤り検出を行う手法である。チェックサム以上に偶発的な誤りについての検出精度が高く、また実装や数学的な分析も比較的容易である一方、チェックサムと同じく意図的な改竄には無力である。

CRC では、元の情報ビット列を除数を示すビット列 (通常、各ビットを立てて多項式と見立てて**生成多項式**と呼ぶ) で割り、この剰余をとる。ただし、2 進数であることと繰り下がりの計算を無視することから、引き算ではなく XOR [3]を行う。とった剰余を元の情報ビット列に付加することで一つの情報となる。

例として、USB の通信に出てくる USB トークンパケット[4]で用いられている CRC-5-USB を見てみよう。

今、10110001 という情報が送られるとする。CRC-5-USB の生成多項式は x^5+x^2+1、bit 列で表すと 100101 である。除算の筆算の形式（ただし、商と除数は省略) で書けば、

元の数	1	0	1	1	0	0	0	1
最上位から XOR	1	0	0	1	0	1		
XOR 結果			1	0	0	1	0	1
第 3 位から XOR			1	0	0	1	0	1
剰余								0

となり、剰余は 0 である (通常の除算の筆算と異なり、引き算ではなく XOR である点に注意すること。)。CRC-5-USB の生成多項式は 5 次 (=6bit) であるので、その剰余は 4 次 (=5bit) である。したがって、この剰余を 00000 として、元のデータに付すことで CRC-5-USB による誤り検出符号付きデータが作成される。

7.3 誤り訂正符号

誤り訂正符号は、誤り検出符号に比べると複雑な手法が多く、その分データ量も増えるなどの難点がある。一方で、再送要求が生じないため、再送要求による負荷が大きいと考えられる場合や特定時間しか使えない回線などでは重宝される手法でもある。

ここでは誤り訂正符号の例として、ハミング距離による訂正符号と、ハミング符号を紹介する。

7.3.1 ハミング距離による訂正符号

情報のうち bit が異なれば 1, 同じであれば 0 を加算し、これを全 bit について行った総和を**ハミング距離**と呼ぶ。n ビットの情報 a, b の i ビット目を a_i, b_i と記せばハミング距離 H は

$$H = \sum_{i=1}^{n} (1 - \delta_{a_i b_i}) \tag{7.5}$$

と書ける。ただし、δ_{ij} はクロネッカーデルタ[5]である。

[3]排他的論理和 (exclusive or)。二つの集合の一方のみに属するような部分のこと。bit の場合、同じ値の XOR は 0、異なる値 (1 と 0) の XOR は 1 となる。bit 毎の XOR は繰り上がりのない二進加算 (=繰り下がりのない二進減算) と同じである。

[4]続くパケットが IN か OUT かを示すなど、通信制御の役割を持つ。

[5]i と j が同じ場合は 1、異なる場合は 0 を返すという関数。

今、3bit の符号のうち、000,111 の 2 つの符号 (ハミング距離が 3 であるような符号) のみを正しい情報とし、それ以外は利用しない符号であるとする。このとき、3bit に対して最大 1bit しか誤らないという仮定をおけば、ある 3bit が 010 になるなど、本来あり得ない値が来た場合にハミング距離が短い側の符号にすることで訂正できる。もちろん、2bit 以上の誤りの場合は誤った側に解釈してしまうし、3bit 使っていながら 2 値であるので元のデータよりもかなり大きなデータとなってしまうが、先の仮定が成り立つ状況かつ元データのサイズが小さい場合などに確実で容量も一定の伝送ができるともいえる。

同様に、ハミング距離がある一定値の符号の組を用意してそれ以外は使わず、誤ったデータが来た場合近い側に合わせるという手法で比較的単純な訂正符号となる。

7.3.2 ハミング符号

ハミング符号は、パリティビットをさらに拡張した方式で、元データのうち選択的に選ばれた何ビットかのパリティビットを複数種用意し、これをデータ末尾に付加することによって誤りの検出のみならず例えば 1 ビットの訂正を可能にする方法ならびにその符号である。

ハミング符号では、最初、元データの 1 ブロック (データブロック) とパリティビットを次の手順に従って並べる。

1. 最初、ビット列が空の状態であると考え、この 1bit 目から埋めていくとする。
2. ビット列を先頭から順に埋める。以下、現在考えている箇所をハミング符号の第 i 番目とあらわす。
   - i が 2 の累乗数であるとき、このビットはパリティビットとする。
   - i が 2 の累乗数でないとき、データブロックのまだ取っていないビットのうち、先頭のものをハミング符号に含める。
3. 前項をデータブロックが尽きるまで行う。

この手順に従って、例えば bit 列の 15bit 目までを記述すると表 7.1(d は元データのデータブロック、p はパリティビットを示し、各々添え字は何番目かを示す) になる。

表 7.1: ハミング符号のパリティ決定用 bit 列 15bit 目までのビット内訳

ビット列の番号	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
内容	p_1	p_2	d_1	p_3	d_2	d_3	d_4	p_4	d_5	d_6	d_7	d_8	d_9	d_{10}	d_{11}

ここで、i 番目のパリティビット p_i は、ビット番号を 2 進表記したときに 2^{i-1} の位のビットが立っている箇所のパリティビットの役割を果たす。例えば、p_2 の場合は、2 の位のビットが立っている番号ということで、ビット列をベースとして 2,3,6,7,10,11,14,15bit でのパリティを指す。このとき、一般には even となるような値がパリティとなる (換言すれば、パリティビット以外の対象ビットの 1 の個数が奇数なら 1、偶数なら 0 である)。元データで言え

ば、1,3,4,6,7,10,11bit 目の XOR 総和が p_2 となる。この計算によって得られたパリティビットを、先の表 7.1 に加えた符号全体をハミング符号と呼ぶ。

ハミング符号のある 1bit に誤りがあった場合、その bit の関係する箇所でパリティが合わなくなる。例えば、元データの 5bit 目 (表 7.1 でわかる通り、ビット列で言うと 9 番目) に誤りがあった場合、$9_{(10)} = 1001_{(2)}$ であることから、p_1 と p_4 が合わなくなる。1bit の誤りである仮定の下この誤り方は一意に定まるため、何 bit 目が誤っているか特定・訂正できる。

ハミング符号を付す場合は、パリティビットを 2^m の単位でつけることから、ブロック長を $2^m - 1$ とする。この内訳はパリティビットが m 個、元データが $2^m - 1 - m$ 個である。このときのハミング符号を Hamming(2^m,$2^m - 1 - m$) と記す。例えば、Hamming(7,4) は 1 ブロック当たり 7bit, うちパリティビットが 3bit で元データが 4bit であるようなハミング符号である。以下、この Hamming(7,4) での例を見てみよう。

Hamming(7,4) のパリティビット数は 3bit である。表 7.1 をみてパリティを決定すると

$$\begin{aligned} p_1 &= d_1 \oplus d_2 \oplus d_4 \\ p_2 &= d_1 \oplus d_3 \oplus d_4 \\ p_3 &= d_2 \oplus d_3 \oplus d_4 \end{aligned}$$

である。(ただし、$\oplus$ は XOR 演算を示す。) 例えば元データが 1101 であった場合、$p_1 = 1, p_2 = 0, p_3 = 0$ となり、ハミング符号は 1010101 となる。

では、これが誤って送られて、例えば 1010111 と来たとしよう。このとき、先のパリティを計算すると $p_1 = 1, p_2 = 1, p_3 = 1$ となり、p_2, p_3 が異なる。この 2 つに含まれていて p_1 に含まれないのは d_3 であるから、誤っているのは d_3 と分かり、これを 0 に訂正することができるのである。

演習問題

1 Linux ディストリビューションの iso ファイルなど、大きなファイルのダウンロードには、md5 ファイルという誤り検出用ファイルがしばしば用意されている。これは md5sum という手法による誤り検出である。この手法について調べ、md5sum のどのような性質がダウンロードの確認に適しているのか説明しなさい。

2 ISBN-10(旧 ISBN 形式) では、9 桁の番号にモジュラス 11 ウェイト 2 ～ 10 方式とも呼ばれるモジュラス・ウェイト型のチェックディジットを付加している。このチェックディジットは次のように計算されている。

1. 数値の各桁に、下から 2,3,4,$\cdots$,10 の重みをかけ、各桁の結果の合計を一括方式で求める。
2. 合計を 11 で割った剰余を求め、その補数を求める (11 からその剰余を引く)。

3. 先に求めた補数が 11 の場合はチェックディジットは 0、10 の場合は X とする。それ以外の場合は補数そのものがチェックディジットとなる。チェックディジットを末尾につけて一連の番号とする。

10 桁の ISBN コードと思われる数字と X からなる列が入力されるとき、これが正しいかどうかを判断するプログラムを作成しなさい。ただし、X は末尾に限り入力されるものとする。

3 1 バイト毎の後半 4 ビットを足し合わせるチェックサムを考える。例えば、16 進数で (112233) というデータであれば、$1+2+3=6$ がチェックサムである。今、データ `FE DC BA 98 76 54 32 10` があるとき、これを改ざんして、記した方式のチェックサムで検出できないようなデータを作ってみなさい。また、そのようなデータで、元のデータと一切合致しないようなデータも一例作りなさい。

4 生成多項式 $x+1$(除数 11) による CRC はパリティビットであることを証明しなさい。

5 CRC-4-ITU は生成多項式 x^4+x+1 による CRC である。このとき、剰余が同じとなるような、同じ長さの別情報を 2 つ見つけなさい。これが似た情報であるほど誤りやすいといえるが、2 つの情報は何ビット異なるか？

6 4bit でハミング距離が 3 となるような符号の組を一例あげなさい。この符号ではデータ量がどう増え、またどのような誤りなら訂正できるか？

7 元データファイル及び加えるパリティビットの数 m が与えられるとき、元データファイルを Hamming(2^m,2^m-1-m) でハミング符号化 (Encode) するプログラム (Encoder) ならびに、エンコードデータが与えられるときに、元データを訂正して取り出す (Decode) プログラム (Decoder) を作成しなさい。なお、エンコード時に最終ブロックについて bit の数に端数が出た場合、これには 0 を付してビット長をそろえなさい。また、Encode ファイルのヘッダとして、最初の 1byte をパリティビット数、次の 1byte を最終ブロックの元データのビット長 (端数のビット数) を付すこととし、デコード用の情報としなさい。

8 （発展問題）bit 毎の誤り率を p とし、$n-1$bit 毎にパリティビットを付加して $n$bit 毎にデータが区切られ、このセット N 個によって送信できる一連のデータがあったとする。このとき、このデータにパリティチェックで検出できない誤りが含まれる確率を p, n, N を用いて求めなさい。

第III部

コンピュータ・ネットワークの基礎

第 8 章

通信プロトコル

第 9 章

ネットワークの構成

第 10 章

ネットワークアーキテクチャ

第 11 章

ネットワーク機器

第 12 章

Layer1：物理層

第 13 章

Layer2：データリンク層

第 14 章

Layer3：ネットワーク層 (1)IP とアドレス

第 15 章

Layer3：ネットワーク層 (2) ルーティング

第 16 章

Layer4：トランスポート層

第 17 章

Layer5〜7：上位3層

第IV部

ネットワーク・プロトコル

第 18 章

ネットワークモデル・再論

第 19 章

telnet

第 20 章

DNS

第 21 章

DHCP

第 22 章

FTP

第 23 章

HTTP

第 24 章

メールに纏わるプロトコル

第 25 章

認証に纏わるプロトコル

第Ⅴ部

安定通信網の構築と運用

第 26 章

冗長化

第 27 章

セキュリティ

第 28 章

トラブルシューティング手法の基礎

第 A 章

解答例・解説

演習問題の解答例・解説を掲載する。

第 0 章

1 解答例 1

2 解答例 2

第 1 章

1 解答例 1

2 解答例 2

3 解答例 3

索 引