2019
春号

百萬石

Hyakumangoku Vol.52

Published by

部長挨拶

akky

akky@mma.club.uec.ac.jp

「百萬石」をお手に取っていただき、ありがとうございます。弊誌 (以下、部誌) は、電気通信大学の公認サークル MMA が新歓期 (春号) と弊学が開催する調布祭時期 (秋号) の年 2 回発行・頒布しているものです。MMA とは、”Microcomputer Making Association”の頭文字に由来しており、1975 年に創立されたサークルです。現在では計算機に関連することを中心に、ネットワークやセキュリティなどの分野でも活動しています。部誌には、MMA の部員が各々に行なっている活動を記事にしたものが掲載されています。内容は多岐に渡り、高度なプログラミングの話から旅行記など、MMA の活動の自由さが感じられるものとなっております。

新入生の皆さん、ご入学おめでとうございます。MMA はコンピュータの周辺技術に興味がある人が集まっているサークルです。部員各々興味があることを学び教え合うことで、技術力を互いに高め合っています。また MMA 部員の関心事は人それぞれで、技術のみならず様々な分野において詳しい人がいるという自由な雰囲気が最大の魅力です。

MMA では有志による定期的な勉強会も開いています。昨年度開催した勉強会は Rust[*1]の輪読会[*2]、競技プログラミング[*3]勉強会、CTF[*4]勉強会です。

さらに例年 4 月から 6 月にかけては新入生向けに講習会を行っていますので、初心者の方も基本的なスキルを一通り身に付けることができます。

最後に MMA に興味を持って頂けた方はサークル棟 2 階の部室までお気軽にお越しください。お待ちしております。

[*1] プログラミング言語

[*2] ある本を数人で分担して読み進める会のこと

[*3] 数理的な問題をプログラミングを用いて解くコンテストのこと

[*4] セキュリティに関する技能を試されるコンテストのこと

百萬石 Vol.52

Hyakumangoku Vol.52

2019 Spring

MMA

目次

第 1 章　XOR について知っている二、三の事柄　kira924age　1
1.1　XOR とは 1
1.2　ビット演算における XOR 2
1.3　XOR の基本性質 2
1.4　XOR と暗号 4
1.5　XOR swap 5
1.6　プログラミングコンテストに出題された問題 6
1.7　おわりに 9
参考文献 9

第 2 章　安全に秘密鍵を管理する　akky　10
2.1　秘密鍵をサーバー上に置くことの危険性 10
2.2　秘密鍵をサーバー上に置かない 11
2.3　終わりに 12
参考文献 12

第 1 章

XOR について知っている二、三の事柄

kira924age

概要

プログラミングコンテストや CTF に参加していると XOR への理解を問う問題にしばしば出くわします. 非常に単純な演算である XOR ですが, いろいろな性質があり様々なことに利用されています. 今回は, そんな XOR についてのお話です. いくつかの演習問題も用意したので, もしよかったらチャレンジしてみてください.

1.1 XOR とは

XOR は日本語では排他的論理和といいます. えらく長いですね. 英語では Exclisive OR といい, 一般には XOR と略されて呼ばれます. EOR という略称も存在しますがあまり使われません. ですが ARM アセンブリの排他的論理和を行う命令は EOR ですので ARM のアセンブリを読み書きするときは注意してください.

XOR は 2 つの入力が与えられたとき, その真偽値が異なるとき結果が真となり一致するとき偽となる演算です.

演算子は一般に $\bigcirc$ の中に加算記号 $+$ を書いた $\oplus$ が使われます. 本稿でも以降 XOR 演算子は $\oplus$ を使用します. 今, 1 を真 0 を偽としたとき真理値表は以下のようになります. 真理値表とは全通りの入力に対しての結果を表にしたものです.

X	Y	X $\oplus$ Y
0	0	0
0	1	1
1	0	1
1	1	0

表 1.1　XOR の真理値表

入力が同じ時は 0, 異なるときは 1 となります. これだけです. とてもシンプルですね.

1.2　ビット演算における XOR

ビット演算は 2 進表記した数値の各ビットに対して, AND, OR, XOR などの論理演算を行う演算です.

ビット演算における XOR 演算のことを単に XOR と呼ぶことも多いです. ”ビットごとの XOR” などのように論理演算の XOR とは異なることを明示することもあります.

以降, 特に注釈がない限り XOR という言葉をビット演算における XOR という意味で使用します.

ビット演算における XOR の例を一つ見てみましょう.

X = 4, Y = 7 であるとき, $X \oplus Y = 3$ となる.

下記のように X, Y をそれぞれ 2 進表記し, 各ビットごとに論理演算としての XOR をすれば $X \oplus Y$ の値を求めることが出来ます.

```
      X = 0100
      Y = 0111
---------------
X XOR Y = 0011
```

プログラミング言語で XOR 演算は ^ (サーカムフレックス) であることが多いです. 実際 C/C++, Java, Python, JavaScript などの多くの言語で, XOR 演算は ^ (サーカムフレックス) で表されます.

唐突ですが, 最初の演習問題を出題します.

演習問題 1

$3 \oplus 7$ を計算し, 出力するプログラムを作成せよ.

みなさんにとっては簡単すぎましたね. 以下解答例です.

- C による解答例

```c
#include <stdio.h>

int main() {
    printf("%d\n", 3^7);
    return 0;
}
```

1.3　XOR の基本性質

XOR には以下のような法則が成り立つことが知られています.

- 交換法則: $X \oplus Y = Y \oplus X$
- 結合法則: $(X \oplus Y) \oplus Z = X \oplus (Y \oplus Z)$
- 逆元が自分自身: $X \oplus Y \oplus Y = X \oplus (Y \oplus Y) = X \oplus 0 = X$

三番目の法則について詳しく見てみましょう. wikipedia の「逆元」の項目での説明が理解しやすいため引用します.

> 直観的には与えられた元に結合してその効果を「打ち消す」効果を持つ元のことである。

つまり逆元が自分自身であるとはある要素 X に対して Y を作用させたあと, それを打ち消すためには再び Y を作用させればよいということです.

これは以下の事実から容易に導けます.

- 同じ値をもつ 2 つの要素の XOR の結果は 0 となる.
  - 各ビットも同じ値なので結果は必ず 0 となることがわかる. (同じ値同士の XOR の結果は 0 と決まっていましたね.)
- 結合法則が成り立つので以下のように計算順序を入れ替えても良い.
  - $X \oplus Y \oplus Y = X \oplus (Y \oplus Y) = X \oplus 0$
- 任意の要素は 0 と XOR をとっても値が変わらない.
  - 真理値表を見ると $1 \oplus 0 = 1$ で $0 \oplus 0 = 0$ となっていることがわかります. つまり 0 を作用させても結果は変わらないのです. 各ビットごとにも同じことが言えます. 任意の値 X と 000...00 との XOR の結果は X なのです.

同じ値をもつ 2 つの要素の XOR の結果が 0 になるという性質を利用し, 多くのプロセッサではレジスタに 0 という値を代入するとき, 以下のように即値を使うのではなく

```
mov EAX, 0
```

代わりに XOR 演算を用いることが多い.

```
XOR EAX, EAX
```

こちらの方が速度や出力される機械語のサイズなどの面で有利なことがあるらしいです.

任意の値 X と 0 との XOR の値は X でした. では X と 1111...111 との XOR の値はどうなるでしょうか? X の各ビットについて 1 なら結果は 0 に, 0 なら結果は 1 となります. つまりビットが反転するのです.

ではそんなビット反転に関する問題を出題します. この問題は基本情報技術者試験の午前の過去問です.

演習問題 2

8 ビットのデータの下位 2 ビットを変化させずに, 上位 6 ビットのすべてを反転させる論理演算はどれか.

ア: 16 進数の 03 と排他的論理和をとる.

イ: 16 進数の 03 と論理和をとる.

ウ: 16 進数の FC と排他的論理和をとる.

エ: 16 進数の FC と論理和をとる.

- 解説

まず, ビットを反転させるというキーワードから施す演算は XOR だと分かるので, アかウに絞られます.

ではどんな値で XOR をとると目的を達成できるでしょうか. これまでのことを思い出せばとても簡単です.

下位 2 ビットを変化させないとあるので下位 2 ビットは 00 と決まります. また上位 6 ビットのすべてを反転とあるので上位 6 ビットも 111111 と決まります.

つまり 11111100 と XOR とればいいです. これを 16 進数に変換しましょう. 2 進数から 16 進数に変換するには 4bit ずつ見ればいいですね. 以下のように 2 進数を 4bit ごとに分割し 16 進数に変換しましょう.

1111 1100(2) = F C(16)

したがって答えはウです.

1.4　XOR と暗号

XOR による暗号といえば, one-time pad を思い浮かべる人は多いでしょう. one-time pad は平文と同一の長さの乱数列を鍵として用いて平文と乱数列を XOR したものを暗号文とします.

とても単純ですが, one-time pad は鍵が流出しない限り, 完全な暗号として知られています. one-time pad によって暗号化された暗号文は鍵がない限り, 絶対に復号出来ません.

なぜなら正しい平文に複合できたかどうかの判定のしようがないからです.

そんな一見, 完全に見える one-time pad が使われることは, 実はほとんどありません. なぜでしょうか?

理由はいくつかありますが, 最も大きな理由は鍵の配送問題にあります.

one-time pad では, 平文と同一の長さの鍵を安全に送受信する必要があります. そんなことができるのであれば, 平文をその手法を使って, 直接送ることが可能なはずです. わざわざ one-time pad を使うメリットはほとんどないと言えるでしょう.

今回の演習問題は XOR を使った暗号問題です.

one-time pad を出題したら絶対に解けないので, one-time pad の問題ではありません.

安心してください.

演習問題 3

平文の各文字ごとにある同一の値で XOR をとった暗号文が与えられる.
平文には ”mma” という文字列が含まれるそうだ.
暗号を解読せよ.
暗号文: `cqxw{yq4‘{4yyu54rxusom{aKc}zi`

鍵の候補は 0x01〜0xff が繰り返される 0xff 通りしかないので全通り試せます.

鍵の候補: 0x0101010101...0101, 0x020202020...0202, ..., 0xffffffff..ff

- C による解答例

```c
#include <stdio.h>
#include <string.h>

int main() {
    char cipher[100] = "cqxw{yq4`{4yyu54rxusom{aKc}zi";
    char plain[100];
    int key, j;
    for (key = 1 ; key <= 0xff; key++) {
        int cipher_len = strlen(cipher);
        for (j = 0; j < cipher_len; j++) {
            plain[j] = cipher[j] ^ key;
        }
        plain[cipher_len] = 0;
        if (strstr(plain, "mma") != NULL) {
            printf("%s\n", plain);
        }
    }
    return 0;
}
```

- 実行結果

```
welcome to mma! flag{you_win}
```

1.5 XOR swap

今回は, いきなり演習問題です.

演習問題 4

2 つの変数の値の swap (交換) を XOR 演算により実現せよ.
ただし, 一時変数を使ってはならない.

2 つの変数の値の交換を実現したいとき, おそらく, あなたは以下のように一時変数を利用することをまず考えたでしょう.

ソースコード 1.1 swap.c

```c
void swap(int *x, int *y) {
    int tmp;  // 値を保持するための一時変数
    tmp = *x; // x の値を一時保存
    *x = *y;  // x の値を y の値に変更
    *y = tmp; // y の値を x の値に変更
}
```

しかし, 今回, 一時変数を使うことを禁じられています.
しかも XOR を使うことを強要されています! 困りましたね.
実は XOR を利用すると, 以下のように一時変数を使わず swap を実現することが出来ます. これは XOR swap algorithm と呼ばれる有名なアルゴリズムです.

ソースコード 1.2　xor_swap.c

```
void xor_swap(int *x, int *y) {
    if (x != y) {
        *x ^= *y;
        *y ^= *x;
        *x ^= *y;
    }
}
```

一見, なぜこれがうまく動作するのか不思議ですが, 各演算を行ったあとの変数の値を追って行けば確かに swap が実現できていることが分かります.

以下の命令で, 変数 x に $x \oplus y$ の値が代入される.

```
*x ^= *y;
```

次の命令で, 変数 y に $y \oplus (x \oplus y) = x$ の値が代入される.

```
*y ^= *x;
```

最後の命令で, 変数 x に $x \oplus y \oplus x = y$ の値が代入される.

```
*x ^= *y;
```

確かに, swap されていますね.

XOR swap algorithm は cool ですが, 現代のプロセッサにおいては必ずしも良いやり方ではないそうです.

興味のある方は wikipedia の XOR 交換アルゴリズム の項を参照してみてください.

1.6　プログラミングコンテストに出題された問題

プログラミングコンテストというプログラミングの能力を競う競技があります. その競技では XOR に関する問題が頻出です.

最近解いた問題から 2 題を紹介します. どちらも AtCoder というコンテストサイトの問題を改題したものです. プログラミング技術を磨く上で, とても良いサービスなので新入生のみなさんも是非やってみましょう.

なんと利用料金は完全無料です.

1.6.1　連続する数列の XOR

演習問題 5

初項が 0, 項数が n+1 の連続する数列 a = {0, 1, 2, 3, ..., n } がある.

$a_0 \oplus a_1 \oplus a_2 \oplus ... \oplus a_n$ の値を求めよ.

ただし, $0 \leqq n \leqq 10^{18}$

この問題は AtCoder Beginner Contest 121 D 問題 XOR World の改題です.

X を偶数としたとき X と X+1 をビットごとに比較したとき異なっているのは最下位の 1 ビットだけです. したがって X が偶数のとき, X $\oplus$ (X+1) = 1 が成り立ちます.

今, n を 4 で割った余りが 0 のとき, 以下のように 1 の組が, 偶数個できて, n だけが孤立するので, 結果は n となる.

(0, 1), (2, 3), 4

n を 4 で割った余りが 1 のとき, 以下のように 1 の組が, ちょうど奇数個できるため, 結果は 1 となる.

(0, 1), (2, 3), (4, 5)

n を 4 で割った余りが 2 のとき, 以下のように 1 の組が, 奇数個出来て n が孤立し, n が偶数なので, 結果は n+1 となる.

(0, 1), (2, 3), (4, 5), 6

n を 4 で割った余りが 3 のとき, 以下のように 1 の組が, ちょうど偶数個できるため, 結果は 0 となる.

(0, 1), (2, 3), (4, 5), (6, 7)

以上よりこの問題を O(1) で解くことが出来ました.

- C による実装例

```c
#include <stdio.h>

typedef long long int ll;

ll solve(ll n) {
    switch(n % 4) {
        case 0: return n;
        case 1: return 1;
        case 2: return n+1;
        case 3: return 0;
    }
}

int main() {
    ll n;
    scanf("%lld", &n);
    printf("%lld\n", solve(n));
    return 0;
}
```

1.6.2　和と XOR が一致する区間

演習問題 6

長さ N の整数列 A が与えられる.

ある長さが 0 でない区間のうち, その区間の要素の総和と, その区間の全ての要素を XOR した値が一致するようなもののうち, 最長となる区間の長さを答えよ.

$1 \leqq N \leqq 2 \times 10^5$, $0 \leqq A_i \leqq 2^{20}$

この問題は AtCoder Beginner Contest 098 D 問題 XOR Sum 2 の改題です.

和と XOR は似ています. XOR の真理値表のうち 3 つは結果が和と一致しています.

$0 + 0 = 0 \oplus 0$

$1 + 0 = 1 \oplus 0$

$0 + 1 = 0 \oplus 1$

しかし, 1 と 1 の演算を行うと, $1 + 1 = 2$ であるのに対し, $1 \oplus 1 = 0$ となり, 一致しません.

基本的には足し算の結果のほうが XOR の結果よりも大きくなりそうです.

では, どのようなときに足し算と XOR の結果は一致するのでしょうか?

それは 1 と 1 の演算が一度も行われないときです.

例えば, $X = 4$, $Y = 9$ のとき,

```
      A = 0100(2)
      B = 1001(2)
A XOR B = 1101(2)
```

$A + B = 13$, $A \oplus B = 1101(2) = 13$ となり条件を満たします.

これはビットごとで見たとき 1 と 1 の演算が行われないからです.

したがって, 数列の部分列の各要素を 2 進数表記したときの各桁について 1 の出現回数の合計が 1 以下のとき区間の和と区間の要素をすべて XOR した結果は一致します.

2 進数同士の足し算を考えて, 繰り上がりが起こらないときに条件を満たすと言い換えることもできます.

そのような区間の部分区間も条件を満たすことは直感的にわかります.

したがって, 尺取法によりこの問題を O(N) で解くことができます.

- C による実装例

```c
#include <stdio.h>

typedef long long ll;

ll max(ll a, ll b) { return (a > b) ? a : b; }

int main() {
    int n, i; scanf("%d", &n);
    int a[200005] = {0};

    for (i = 0; i < n; i++) {
        scanf("%d", &a[i]);
    }

    ll ans = 0;
    ll s = 0, x = 0, r = 0, l;

    for (l = 0; l < n; l++) {
        while ( r < n && ( (s+a[r]) == (x^a[r]) ) ) {
            s += a[r];
            x ^= a[r];
            r++;
        }
        ans = max(ans, (r-l));
```

```
        if (l == r) { r++; }
        else {
            s -= a[l];
            x ^= a[l];
        }
    }

    printf("%lld\n", ans);

    return 0;
}
```

1.7 おわりに

XOR を題材に, いくつかのトピックを紹介してみました.

簡単すぎたと思ったあなたは CTF やプログラミングコンテストの問題を解くことをおすすめします. もっと難しい問題が無限にあります. 是非チャレンジしてみてください.

参考文献

[1]『暗号技術入門 第 3 版 秘密の国のアリス』 結城浩 SB クリエイティブ

[2]『クラウドを支えるこれからの暗号技術』 光成滋生 秀和システム

[3] https://ja.wikipedia.org/wiki/排他的論理和

[4] https://ja.wikipedia.org/wiki/XOR 交換アルゴリズム

[5] https://img.atcoder.jp/abc121/editorial.pdf

第 2 章

安全に秘密鍵を管理する

akky

概要
先日友人から ssh で使用する秘密鍵をサーバー上に置くのは危険だと教わったので，そのことについて調べてまとめました．ローカルマシンの環境は macOS です．

2.1　秘密鍵をサーバー上に置くことの危険性

秘密鍵をサーバー上に置いておくと，次のような危険にさらされます．

- 権限の設定を間違えて，他のユーザーに秘密鍵を見られてしまう．
- 権限の強いユーザーに秘密鍵を見られてしまう．
- サーバーが攻撃を受け，秘密鍵が漏洩してしまう．

このことから，特に共用で使用しているサーバー上には秘密鍵を置かない方が良いでしょう．

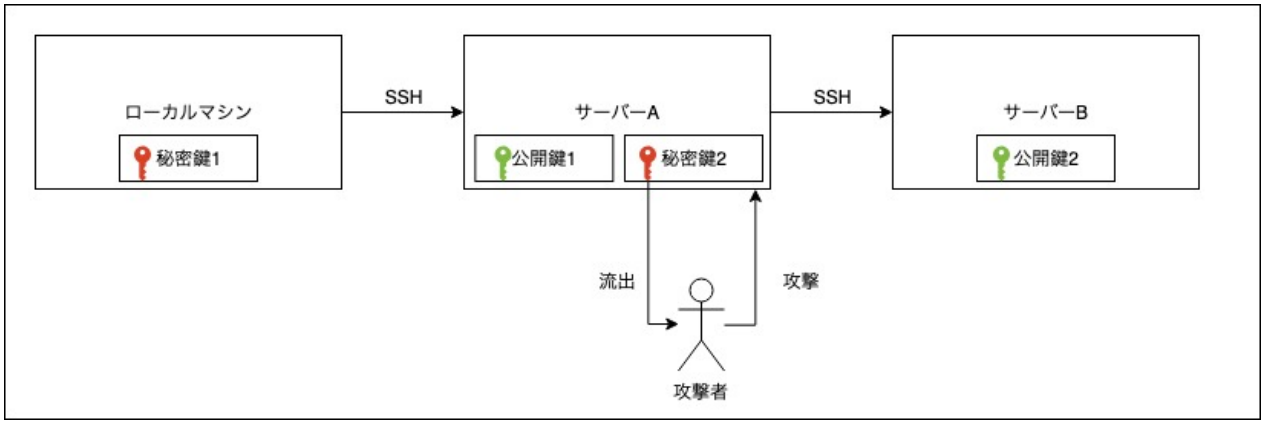

図 2.1　サーバー上に秘密鍵を置くのは危険!

2.2　秘密鍵をサーバー上に置かない

しかし開発サーバーから git を使ってリモートリポジトリを操作する場合や，サーバーを踏み台にして他のサーバーにつなぐ場合など，サーバー間で公開鍵認証を行いたい場合もあります．この際有効なのがサーバーに秘密鍵を置かずに，ローカルマシンの秘密鍵を SSH する際に引き継いでサーバー上で使う方法です．

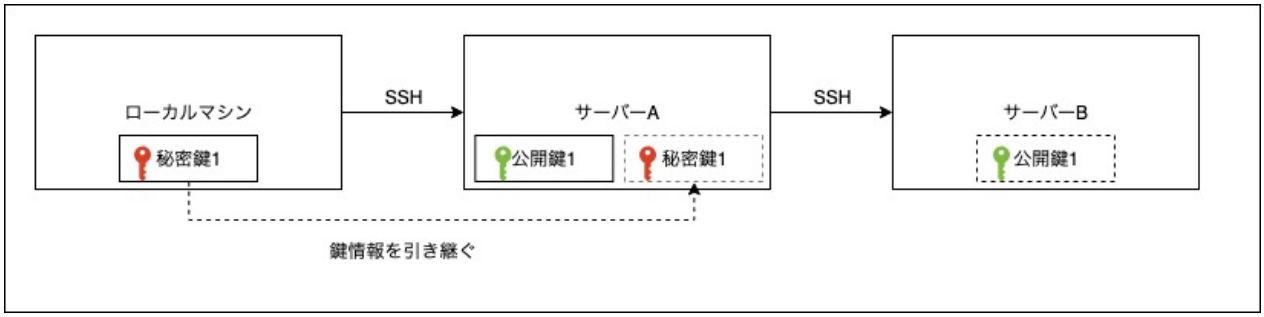

図 2.2　鍵をローカルマシンから引き継ぐ

ローカルマシン ↔ サーバー $A(hostA)$ ↔ サーバー $B(hostB)$ のようにつなぐとしたとき，具体的な手順は以下の通りです．

1. ローカルマシンで鍵を生成する

```
$ ssh-keygen
```

2. ローカルマシンの~/.ssh/config を編集する

```
$ vim ~/.ssh/config
```

以下のように書きます

ソースコード 2.1　ローカルの~/.ssh/config

```
Host hostA
    HostName <hostAのipアドレスまたはドメイン名>
    User <hostAのログインユーザー名>
    ForwardAgent yes
```

3. 秘密鍵を ssh-agent に登録する

ssh-agent は鍵の管理ツールです．

```
$ ssh-add -K ~/.ssh/id_rsa
$ ssh-add -l # 登録されている秘密鍵のfingerprint一覧
```

4. 公開鍵を hostA に登録する

ssh-copy-id を使うのが楽です．

```
$ ssh-copy-id hostA
```

5. hostA に接続し，~/.ssh/config を編集する．

```
$ ssh hostA
$ vim ~/.ssh/config
```

例として hostB を GitHub とすると以下のように編集します．

ソースコード 2.2　hostA の~/.ssh/config

```
Host github.com
    Hostname ssh.github.com
    User git
    ForwardAgent yes
```

また，hostA にローカルマシンの秘密鍵が引き継がれているかは

```
$ ssh-add -l
```

で確認できます．ローカルマシンで実行した時と fingerprint が同じならば正しく引き継がれています．

6. hostB に公開鍵を登録し，hostB に接続する．
   hostB を GitHub とすると Web から公開鍵を登録し

```
$ ssh github.com
PTY allocation request failed on channel 0
Hi <GitHub account>! You've successfully authenticated, but
    GitHub does not provide shell access.
Connection to ssh.github.com closed.
```

とすることで，正しく認証されているか分かります．

重要なのは~/.ssh/config に **ForwardAgent yes** を書くことです．この設定によって ssh の接続先でも秘密鍵を引き継いで使うことができます．

2.3　終わりに

ssh を使い始めてから今までサーバー上に秘密鍵を作りまくってきたので，秘密鍵を引き継げることには感動しました．秘密鍵は極力ローカルマシンに置こうと思います．

参考文献

[1] https://qiita.com/isaoshimizu/items/84ac5a0b1d42b9d355cf

百萬石 2019 -春- © 電気通信大学 MMA

2019 年 4 月 3 日　初版第一刷発行　【本書の無断転載を禁ず】
2019 年 4 月 4 日　初版第二刷発行

著　者　kira924age, akky
表　紙　kyo1
編集者　kyo1
発行者　電気通信大学 MMA
発行所　電気通信大学 MMA 部室
〒182−8585 東京都調布市調布ヶ丘 1−5−1 サークル会館 208 号室
http://www.mma.club.uec.ac.jp/
印刷所　電気通信大学 MMA 部室
製本所　電気通信大学 MMA 部室

電 気 通 信 大 学