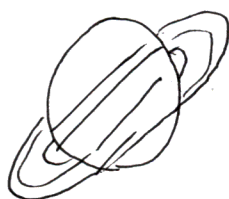
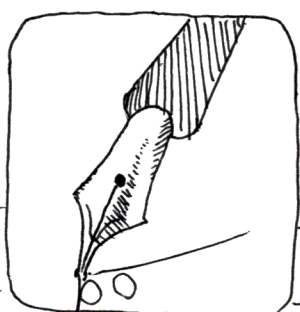


WORD

From College of Information Science



ツイート
11.4K

フォロ
514

フォロワ
810,199

@Word__writing

www.word-ac.net

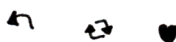
ツイート

ツイートと返信



Word writing @Word__writing · 2時間

Lorem ipsum dolor sit amet, consectetur
adipiscing elit, sed do eiusmod tempor



マイ・マイナンバーで氏名という概念を破壊する

Mランド入校ルポ～予告編～

(*>△<) では3 だが, (o・▽・o) では7. な～んだ.

Nexus 5 で市民武装～ ARMed and Dangerous ～

LuaRocks で結果にコミット

目次

| | |
|--------------------------------------|----------|
| マイ・マイナンバーで氏名という概念を破壊する | 3 |
| Mランド入校ルポ～予告編～ | 10 |
| (*>△<) では 3 だが, (o・▽・o) では 7. な～んだ. | 11 |
| Nexus 5 で市民武装～ ARMed and Dangerous ～ | 20 |
| LuaRocks で結果にコミット | 27 |
| WORD読者アンケート | 32 |
| WORD編集部への誘い | 38 |
| 編集後記 | 39 |

マイ・マイナンバーで氏名という概念を破壊する

文 編集部 Perio

1 懸賞クイズ

さっそくですが懸賞クイズです。

自然数 a, b, c, d が、

$$a > b > 0, c > d > 0, a^a b^b = c^c d^d$$

を満たすならば、 $a = c$ かつ $b = d$ か？ これを証明するか、反例を挙げよ。

解けた方がいたら編集部に送ってください。よろしくお願いします。メモリ型ステッカーやメモリ型定規などの超豪華景品を差し上げます。

郵便番号 305-0005

住所 茨城県つくば市天王台 1-1-1 筑波大学第 3 エリア C 棟 212

情報科学類 WORD 編集部

メール word@coins.tsukuba.ac.jp

つづく本文では、これが成立するという前提のもとで、新しいマイナンバーと、氏名をマイナンバーで置き換える方法を提案していきます。

2 はじめに

行政の手続きを円滑にするために、国民一人ひとりに 12 桁の番号を振る**マイナンバー制度**（法令では**個人番号**）が昨年 9 月に可決され、1 月から利用が始まりました。行政サービスを円滑化することが同法の目的ですが、マイナンバーを付与される個人からすれば、氏名・住所に加えて、新たにマイナンバーを会社やバイト先に提出する必要があり、今のところ手間が増えただけのようです。

そこでこの記事では、そのような混乱に乗じて新しいマイナンバーシステムおよび氏名の制度を提案します。新しいマイナンバーは家族間で番号に関連性があり、氏と名に対応する概念があります。そのため、氏名の代わりにマイナンバーを使うことで、氏名が不要になります。また、新しいマイナンバーは、番号そのものが家系図を表しています。

3 現在のマイナンバーと氏名

3.1 現行のマイナンバー

マイナンバーは「個人番号」の愛称で、「行政手続における特定の個人を識別するための番号の利用等に関する法律^{*1}」に基づいて国民一人ひとりに付与される番号です。マイナンバー自体は、11桁の住民票コードを変換した12桁の正の整数です。12桁目はチェックデジットで、1桁目から11桁目までの数字から、計算で求められます^{*2}。

マイナンバーは社会保障、税、災害対策などのために使われるとされています。また、パスポート、免許証、住民票などに応用できるのですが、この記事には関係ないので割愛します。

マイナンバーの性質として重要なことは(1)正の整数であること、(2)他の人と重複しないこと、(3)住民票を持つ全ての国民に付与されること^{*3}の3点です。

3.2 氏名に関する話題

この記事に関係のある、氏名についての話題を取り上げます。

夫婦別姓と子どもの姓

夫婦の片方にもう一方の姓へ改めることを強制する夫婦同姓が昨年12月に最高裁で合憲と認められ、この是非が近ごろ話題になっています。いずれは別姓にするか同姓にするかを選択できるようになるかもしれませんが、同姓にせよ別姓にせよ、子どもの姓は片方だけになってしまうという問題が依然として残ります。これは結婚すると片方の姓になってしまう現在の制度の限界ではないでしょうか。

この解決策としてポルトガルやスペイン、ブラジルの名前のように、両親の姓を両方名乗るという方法が考えられます。例えば父が野原で母が小山であれば息子は野原小山しんのすけ、というように、両方の姓を名乗ります。実際、日本でも国際結婚により、このような複合姓を名乗る人がいるようです^{*4}。しかしこの方式でも、どちらの姓を第一姓にするかという悩みが残ります。

姓の淘汰

姓は増えることがないので、1000年後には姓の種類が佐藤さんだけになることが危惧されています^[要出典]。こうなると、親戚一同がお正月に集まっても、何がなんだかわからなくなるおそれがあります。そもそも姓の種類が減ったら、姓を持つ意味が薄れるでしょう。

同姓同名

氏も名も必ずしもユニークではないため、同姓同名が存在する場合があります。これを悪用して、例えば悪意あるヲタクが推しのアイドルと同じ名前を子どもに命名して芸能界に送り込む、などのなりすまし攻撃が可能です。

^{*1} 行政手続における特定の個人を識別するための番号の利用等に関する法律: <http://law.e-gov.go.jp/htmldata/H25/H25H0027.html>

^{*2} チェックデジットの計算方法(総務省令第八十五号): http://www.soumu.go.jp/main_content/000327387.pdf

^{*3} 外国人でも90日以上在留する場合には付与される

^{*4} 家庭裁判所への申し立てが必要

また、近年取り沙汰されているキラキラネームと呼ばれる名前は、ユニークな名前を付けたいという親心からくるそうです。しかし人の子らが付ける名前ですから、同じような名前になりがちという問題があります。仮に名付けの時点で世界でオンリーワンの名前を付けたとしても、あとからパクられる可能性があります。現行制度では名付けの段階で、永遠に同姓同名が発生しないことが保証された名前を子どもに付けることはできないのです。派手な名前がかぶったらちょっと悲惨ですから、国が公共サービスとしてユニークな名前を提供したほうが便利です。

3.3 マイナンバーと氏名の問題点のまとめ

これまでに見てきたマイナンバー制度と氏名の問題点を挙げます。

- 12桁なので、再発行やベビーブームで枯渇するおそれがある。
- 国がランダムにつけるので、ありがたみがない。
- 夫婦同姓を強制するため、人権侵害である。
- 同姓同名が存在する可能性がある。

4 新マイナンバーと新しい氏名の概要

家族や価値観が多様化しているので、とにかく氏名というシステムをマイナンバーもろとも改造します。マイナンバーと氏名の一体改革です。

そこで名に代わる素数を、国が支給するのはどうでしょうか。素数は、自身と1以外では割り切ることができない数です。これは一人ひとりがプライムな存在であることに対応しています。また、新しいマイナンバーは、個人の素数と両親のマイナンバーを引数にとる関数で作ります。これは**血縁や親子の絆**の概念に相当します。

新しいマイナンバーには次のような特徴があります。

- マイナンバーに家系情報がすべて含まれる
- ふたりのマイナンバーから、ふたりの親等数^{*5}を計算できる
- 旧マイナンバーのような性質を持つ（重複しない正の整数である）
- 固定長ではなく、数に上限がない

^{*5} 親等数: 家系図の上での距離。親子なら1、きょうだいなら2、祖父・祖母と孫なら2。おば・おじとおい・めいなら3

4.1 新しいマイナンバーの割り当て規則

図1を例に、マイナンバーがどのように設定されるか説明します。ここでは、国が新生児に支給する素数を p 、両親のマイナンバーを a, b として、新生児のマイナンバーを

$$M(p, a, b) = pa^a b^b$$

という関数で割り当てます。

AさんとBさんのマイナンバーが A, B で、このふたりの間に生まれた子どもに支給された素数が p_C であるとき、新生児のマイナンバーは $C = M(p_C, A, B)$ となります。さらにCさんが大人になり、Fさん ($F = M(p_F, D, E)$) と結婚し、夫婦に新生児Gさんが生まれたとします。この時、Gさんのマイナンバーは

$$\begin{aligned} G &= M(p_G, C, F) \\ &= p_G C^C F^F \\ &= p_G (p_C A^A B^B)^{p_C A^A B^B} (p_F D^D E^E)^{p_F D^D E^E} \end{aligned}$$

となります。

ところで関数 M は、 $M(p, A, B) = M(p, B, A)$ となり、両親 A と B に関して対称です。これは整数の掛け算の交換法則に由来する性質で、子どもを作る上で、男女間の扱いが平等であることを意味しています。

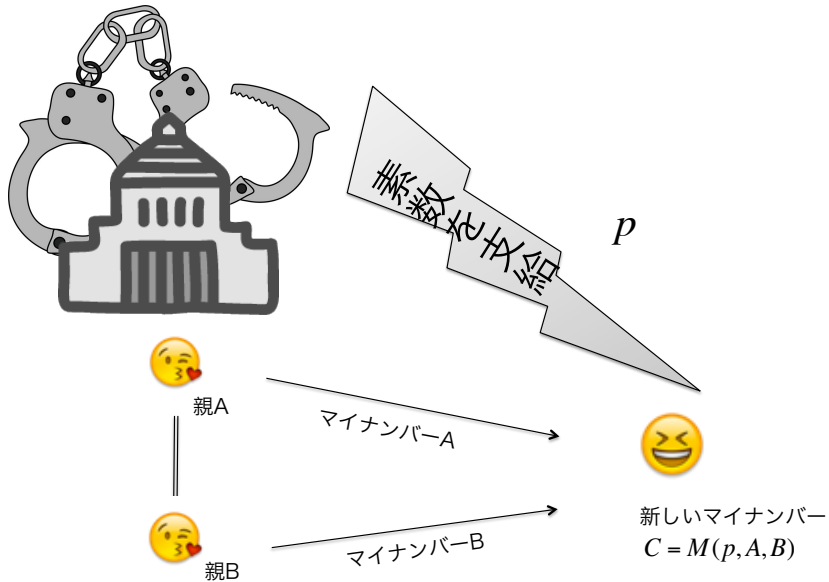


図1: 国が人に付番する図

4.2 家系図との対応

新しいマイナンバーには、自分の直接の先祖のマイナンバーがすべて含まれています。冒頭の問題が成立すれば、自分のマイナンバーから先祖の番号を得ることができるため、直系の家系図を復元することができます。例えば、自分のマイナンバー C から、 $C = p_C A^A B^B$ を満たす p_C 、 A 、 B を見つければ、両親のマイナンバーがわかり、同様の計算を A 、 B に対して行えば、祖父母のマイナンバーが得られます。

これは、アイスランドやロシアの父称^{*6}、イスラム教圏の人名^{*7}の拡張で、父や父系の祖父だけではなく、すべての先祖のマイナンバーを含んでいます。

4.3 氏名制度を置き換える

氏名には次のような悩みがありました。

- 1000 年後には、リアル鬼ごっこも^{*8} 甲斐なく、佐藤姓だけの世界になる。
- 氏名は重複するかもしれない。
- 夫婦同姓は夫婦のどちらかに改姓を迫る。
- 夫婦別姓では、子どもの姓が片方だけになる。
- 夫婦や子どもが、両方の姓を名乗る場合も、その順序に不平等がある。

旅券番号や免許証番号はユニークな ID が振られている一方で、氏名はユニークではありません。そのそも氏名がユニークであれば、マイナンバーは必要なかったのです。旅券番号などもマイナンバーと紐付いたり、一本化される見通しがあるので、悩みの種である氏名も廃止し、マイナンバーに一本化しましょう。

そこでマイナンバーが皆さんの氏名になります。

$$C = \underbrace{p_C}_{\text{C さんの名}} \times \underbrace{(A^A) \times (B^B)}_{\text{C さんの姓}}$$

こういう解釈をすれば、あえて奇抜な名付けを行わなくても、出生届を出すと同時に、世界にひとつだけの名を役所から交付され、一人ひとり、ちがうキラキラしたネームを持つようになります。

5 使用例

5.1 家族の紹介

私の家族を紹介します。両親の氏名は 2 と 3 です^{*9}。私 A に支給された素数は 5 で、私の氏名は

$$\begin{aligned} A &= M(5, 2, 3) \\ &= 540 \end{aligned}$$

^{*6} アイスランドでは姓として、父親の名前を使います。ロシアでは、姓・名に加えて、父の名前を、自分の名前として持ちます。例えばソ連の宇宙飛行士ユーリイ・アレクセーエヴィチ・ガガーリンは、ユーリイが名、ガガーリンが姓で、アレクセーエヴィチが父称です（アレクセイという人名の変形）。

^{*7} 子の名前（クンヤ） 本人の名前（イスマ） 父の名前（ナサブ） 祖父の名前など（ニスバ）

^{*8} リアル鬼ごっこ: 佐藤さんが増えすぎた世界で、数を減らすためにメロメロし合いを行わせる小説

^{*9} 実際には、これはアダムとイブのマイナンバーです。

ゲームや漫画ではよくある展開ですが、現実の日本では妹との結婚を認めていません。やや現実的なゲームや漫画では、生き別れのきょうだいが恋に落ち、それが禁断の恋だったことに後から気づくこともあります。このような悲劇も、もとはといえば氏名が悪いので、マイナンバーを氏名とすれば解決します。

$$C = M(p_C, A, B) = p_C A^A B^B$$

ふたりの人間 C と D がきょうだいである場合、少なくとも片方の親 A が共通しますから、マイナンバーは

$$C = M(p_C, A, B) = p_C A^A B^B$$

$$D = M(p_C, A, B') = p_D A^A B'^{B'}$$

6 おわりに

現在のマイナンバーは、国が国民を管理しやすくするために付与する番号ですが、新しいマイナンバーは氏名でもあり、家系図でもあるため、番号を大事にしようという気持ちがかかります。他方、夢を詰め込んだ分、マイナンバーの桁数が膨れ上がってしまいました。しかし現在の 12 桁のマイナンバーだって覚えられないので、12 桁だろうが 100 万桁だろうが、覚えられないという意味では同じですから問題ではないと思います。

9

Mランド入校ルポ～予告編～

文 編集部 カブ

合宿免許、それは車に乗ろうとする者達のための、避ける事のできる通過儀礼。大半の人間にとっては、一生に一度しか受ける事のないこの貴重な儀式を、免許の取得という即物的な目的のためだけに消費する事は、通過儀礼という概念に対する冒涇に他ならない。

通過儀礼を冒涇しなければ、免許は取得できないという絶望に救いの手を差し延べてくれたのが、島根県は益田市に位置する「M ランド益田校」だった。

M ランドは、「益田ドライビングスクール」という愛称で呼ばれている。愛称から分かるように、M ランドも所詮は全国にあまねく存在する自動車教習所の一つでしかない。一般的な教習所との違いは、とにかく宗教的で胡散臭いというところだ。

「M マネー」という独自通貨をベースに構築された独自のエコシステム、新興宗教的な空気すら感じさせる校歌「魂の友よ ～ソウルメイト～」、「M ランド同窓会」という集会の存在等、公式サイト^{*1} から得られる情報だけでも、その胡散臭さを垣間見る事ができる。

さらに、運営母体である「株式会社コガワ計画」について調べたり、M ランドの口コミを探すにつれ、その胡散臭さは実体を帯びていく。

施設内に蔓延る、「グラビトン・セラミック」や「いやしろ地」といった科学と土着信仰の融合したエセ科学の数々。教習中の校内で、「魂の友よ ～ソウルメイト～」が大音量で流れ続けているという情報。運営会社社長の「とことん故郷を、そして若い教習生たちを思うことが、国を思うことに通じている^{*2}」という、近年流行りの“日本教”を彷彿とさせるような思想。そして、M ランドを取材したメディア^{*3} による、「M ランドで教習を受けた若者は挨拶や笑顔が素晴らしい」という評価。

長老が考えた理想の世界を体現するような評判の数々が、私に、M ランドで免許を取るという決意をさせたと言っても過言ではない。

M ランドと初めて出会ったのは、真夜中、他の編集部員達と共に筑波山に向かっていった時だっただろうか。もしかすると、話を聞いたその時から、私の精神は、M ランドに囚われているのかもしれない。

精神を取り戻すために、私は M ランドへと向かう。例え、この身が小河イズムに侵食されるとしても。

^{*1} <http://www.mland-masuda.jp>

^{*2} http://kitohone.com/kitohone/kogawa_jiro/index.html より引用

^{*3} <http://www.marukoshi.jp/write/%239-8-no5376/>

($* > \triangle <$)では3だが、($o \cdot \nabla \cdot o$)では7. な～んだ.

文 編集部 OttO

1 はじめに

パソコンだいき人間の集う飲み会^{*1}において、共通の話題があると盛り上がります。とりわけ、与えられた文字列に含まれる最長の回文を求めるというトピックは、最も優れた話題の一つであるといえるでしょう。例えばアルコールを過剰に摂取し、人間と動物の境界にあったとしても、ある文字列に含まれる最長の回文を求めるという原始的な行為は、その場を熱狂の渦に巻き込みます。これは当然のことです。何しろ、我々はみな、かつては回文だったのですから……

本論は、入力文字列（長さを N とする）に含まれる最長回文を求めるアルゴリズムを三つ紹介します。まず、2 節で必要な記法を導入し、非常に簡単な $O(N^3)$ 時間アルゴリズムを述べます。このアルゴリズムは、定義から直ちに得られる（回文のことを想っていない）冷徹で残忍なアルゴリズムです。しかしあなたは、3 節と 4 節を引き続き読むことで、かつての記憶を取り戻していきます。3 節では、簡単な回文の性質のもとに、回文の温かみある $O(N^2)$ 時間アルゴリズムを得ます。4 節では、更に改良を加え、回文との親密さを取り戻した、 $O(N)$ 時間アルゴリズムを得るのです。

2 記法と $O(N^3)$ 時間アルゴリズム

文字列▷ 文字列をあらわす変数として、 w や w' 、そして α と β を用いることにします。（長さ 1 以上の文字列は） $w = \sigma_0 \sigma_1 \dots \sigma_N$ として表せます。各 σ_i は文字であり、多くのプログラミング言語では、 $w[i]$ とも書きます。文字列 w の長さを、 $|w|$ で表します。 $w = \sigma_0 \sigma_1 \dots \sigma_N$ であれば、 $|w| = N + 1$ です。

部分文字列▷ w の位置 i から j ままでを切り出して得られる（部分）文字列を $w[i; j] = \sigma_i \sigma_{i+1} \dots \sigma_j$ で表します。文字列 w' について、 $w' = w[i; j]$ を成立させる i と j が存在するとき、「 w' は w の部分文字列である」といいます。注意：長さ 0 の文字列（空の文字列）は、任意の文字列に対する部分文字列と約束しておきます。また、空でない文字列 w は、 $w = w[0; |w| - 1]$ により、自分自身の部分文字列になります。

文字列の反転▷ 文字列 w が $w = \sigma_0 \sigma_1 \dots \sigma_N$ で表されるとき、 w を反転した w^R は、 $w^R = \sigma_N \dots \sigma_1 \sigma_0$ となります。例えば、文字列 abc を反転すると cba になるという具合です。

回文▷ ある文字列 w を、左から読んでも右から読んでも同じものになるとき、回文と呼ぶのでした。形式的には、 $w = w^R$ となるときに、 w は「回文 (palindrome)」である、といいます。例えば、文字列 abc については、 $abc \neq cba$ であるので、これは回文ではありません。一方で、 aba は、これを反転しても aba ですから、回文ということになります。

注意：長さ 0 の文字列（空の文字列）と、長さ 1 の文字列（文字）は、定義上、常に回文です。

^{*1} 未成年者は飲酒をしないでください

最長回文問題▷ 本論で取り扱う問題を「最長回文問題」と呼び、定義を与えます。 w に含まれる最長の回文の長さは、 $P_{\max} = \max\{|w'| : w' \text{ は } w \text{ の部分文字列 かつ 回文}\}$ として定義できます。我々の目的は、 w の部分文字列 かつ 回文 w' で、 $|w'| = P_{\max}$ を満たすものを求める、ということになります。そのような文字列は複数あるかもしれませんが、1つ求まれば OK です（例： $abab$ には 2 つの最長回文 aba と bab があります）。

2.1 $O(N^3)$ 時間アルゴリズム

手始めに、速さは気にせず、とにかく問題を解くアルゴリズムを1つ作ってしまいます。まず、入力 w （長さを N とする）が、長さ n の回文を含むかどうかを、 $O(N^2)$ 時間で判定する関数 $P(w, n)$ を作ります。

```
def is_palindrome(w : string) // 入力  $w$  が回文かどうか調べる
```

```
    if  $|w| = 0$  then return TRUE ;
    for  $i \leftarrow 0$  to  $|w| - 1$  do
        if  $w[i] \neq w[|w| - 1 - i]$  then return FALSE ;
    end
    return TRUE;
```

```
def  $P(w : \text{string}, n : \text{int})$ 
```

```
    for  $i \leftarrow 0$  to  $(|w| - 1) - (n - 1)$  do
        if is_palindrome( $w[i; i + (n - 1)]$ ) then return  $i$  ;
    end
    return NoPALINDROME;
```

関数 is_palindrome は、入力の長さ N について $O(N)$ 時間必要です。関数 $P(w, n)$ は、各位置 i について、長さ n の部分文字列 $w[i; i + (n - 1)]$ に注目し、これが回文となっているかどうかを is_palindrome で調べます。従って、 $P(w, n)$ は、仮に部分文字列の切り出し $w[i; j]$ が定数時間で済むとしても、 $O(N^2)$ 時間を要します。

最長回文を求めるのに、関数 $P(w, n)$ は便利です。実際、 $P(w, |w|)$, $P(w, |w| - 1)$, $P(w, |w| - 2)$, ... として、NoPALINDROME 以外の結果が得られるまで、降順に手続きを実行することで、最長回文の長さ n と、回文の開始位置の 2 つが得られます。 $O(N^2)$ 時間必要な手続きを、 N 回動かすため、 $O(N^3)$ 時間必要になります。

以上から、目標とする問題は確かに解けます。しかし、この解き方は、「回文の良さ」を全く活かしていません。結果として、回文本来のあたたかみが失われた、 $O(N^3)$ という時間計算量になってしまいました。

耳を澄ませて！！^母 回文の^声が、あなたにも届いている筈です。

3 $O(N^2)$ 時間アルゴリズム

控えめにいっても最悪な $O(N^3)$ 時間アルゴリズムは、回文の性質を何も使っていません。使っているのは、回文の定義だけです。元来、回文とは様々な性質を備える幸せな物体です。そのことを思い出しましょう。

この節では、

「長さ $n + 2$ の回文 $\sigma w \sigma$ は、長さ n の回文 w の両脇に、同じ文字を追加して得られる」

という、回文が回文を生み出す性質を利用して、多少効率的に動く $O(N^2)$ 時間アルゴリズムを与えます。

3.1 用語の追加：中心位置 i から半径 r の回文

本節で与えるアルゴリズムの説明のために、用語を 1 つ追加します。

- 文字列 w について、中心位置を i とする、半径 r の部分文字列を、 $\theta_w(i, r) = w[i-r; i+r]$ で定義します。

— $w = \sigma_0 \dots \underbrace{\sigma_{i-r} \dots \sigma_{i-1}}_r \sigma_i \underbrace{\sigma_{i+1} \dots \sigma_{i+r}}_r \dots \sigma_N$ ならば、 $\theta_w(i, r) = \sigma_{i-r} \dots \sigma_{i-1} \sigma_i \sigma_{i+1} \dots \sigma_{i+r}$ です。

- 適切なパラメータを与えなければ、 $\theta_w(i, r)$ は定義されないことに注意してください。例えば、 i が w の中になかったり、両端となる σ_{i-r} や σ_{i+r} が w の中になような場合は、 $\theta_w(i, r)$ は未定義とします。

このとき、半径 r の部分文字列について、次の良い性質 (☆) が成立します：

$$(\star) \quad \theta_w(i, r+1) \text{ が回文} \iff \theta_w(i, r) \text{ が回文} \text{ かつ } w[i-(r+1)] = w[i+(r+1)].$$

これは、位置 i を中心に文字列を折り曲げ、重なった文字が全て同じならば回文である、という回文判定法を言い直したものに他なりません。実際には、 (\iff) が成立するためには、 $w[i-(r+1)]$ と $w[i+(r+1)]$ が、範囲外アクセスにならず、定義されている必要がありますが、細かいところは無視しています。

*2 [一応証明らしきものを。まず (\Rightarrow) の方向を示します。 $\theta_w(i, r+1) = w[i-(r+1); i+(r+1)] = \sigma_{i-(r+1)} w[i-r; i+r] \sigma_{i+(r+1)}$ が回文です。定義から、 $\sigma_{i-(r+1)} w[i-r; i+r] \sigma_{i+(r+1)} = \sigma_{i+(r+1)} w[i-r; i+r]^R \sigma_{i-(r+1)}$ が成立するので、 $\sigma_{i-(r+1)} = \sigma_{i+(r+1)}$ であることと、 $w[i-r; i+r]$ が回文であることが分かります。残った (\Leftarrow) の方向も、同じように証明できます。]

この性質 (☆) をもとに、

- 半径が 0 のところから計算をはじめて（半径 0 の文字列はすなわち文字で、いつでも回文です）、
- より半径が大きな回文に拡大できるかどうかを、順次調べていく

という方針を採用します。

前処理.

性質 (☆) をもとにする戦略には、1 つ問題があります。「中心を i とする半径 0 の文字列（文字）」の両脇に文字を追加していくやり方では、「奇数長の回文」しか得られないということです。これは事件です。例えば文字列 $abba$ の最長回文とは、自分自身 $abba$ ですが、これが求まらないことになります。

この問題を解消すべく、入力 w に含まれない文字 $\#$ を挿入し、「偶数長の回文」が含まれないようにします：

$$w = \sigma_0 \sigma_1 \dots \sigma_{N-1} \mapsto w' = \# \sigma_0 \# \sigma_1 \dots \# \sigma_{N-1} \#.$$

各文字 σ_i の前に、新しい文字 $\#$ を追加し、更に末尾に $\#$ を加えて、長さ N の文字列から、長さ $2N+1$ の文字列 w' を作っています。こうして得た文字列 w' に「偶数長の回文」がないことは、簡単に確認できます。

[仮に偶数長の回文 $v = \delta_1 \dots \delta_n \delta_{n+1} \dots \delta_{2n}$ があったとして、構成の仕方から、 δ_n か δ_{n+1} のどちらか片方が $\#$ で、もう片方が w 由来の文字 σ_i となります。このとき、 $\#$ の取り方から、 $\# \neq \sigma_i$ となり、矛盾します。]

*2 本論で、鍵括弧に入った小文字の文には、細かい説明や証明が書かれています。飛ばしても問題なく読めます。

この先では、入力文字列として、上のように詰め物がされたものを考えていると思ってください。従って、入力文字列の長さは奇数長 $(2N + 1)$ であり、偶数長の回文は含まれません。ただし、変換後の文字列における最長回文は、もとの入力における最長回文と異なることに注意してください。とはいえ、変換後の文字列に対する最長回文から、# を全て落とせば、もとの入力に対する最長回文になることは、容易に確認できます。

3.2 性質 (☆) をもとにした $O(N^2)$ 時間アルゴリズム

私たちの採用する戦略を再確認します。私たちは、前ページの性質 (☆) と $\theta_w(i, 0)$ が回文であることを利用し、 $\theta_w(i, 1)$ が回文かどうか調べ、もし回文なら引き続き $\theta_w(i, 2)$ が回文かどうかを調べて……、という繰り返しで、「中心を i とする最長回文」を求めることにします。この処理を各位置について行い、それぞれを中心とする最長回文を求めます。プログラムにして、次の通りです（以下、 $|w| = 2N + 1$ とします）：

```

result_idx ← 0;  result_radius ← 0;

for i ← 0 to 2N do
    for r ← 1 to N do
        if (w[i - r] ≠ w[i + r]) then break;
        if (result_radius < r) then (result_idx, result_radius) ← (i, r);
    end
end

return  $\theta_w(\text{result\_idx}, \text{result\_radius})$ ;

```

実際には、 $w[i - r] \neq w[i + r]$ のところで、 w の両端を突き抜けていないか（範囲外アクセスをしていないか）のチェックが必要になりますが、細かいところは省略しています。

とにかく、上のプログラムでは、2 重ループの構造が核になっており、結果として $O(N^2)$ 時間で十分となることが明らかです。2 節の $O(N^3)$ 時間アルゴリズムと比べると、回文の性質が多少は使えているので、そこまで酷くない出来です。しかし、これではまだまだ回文の良さが活かせていません。位置 i を中心とする最長回文を求めるのに、別の位置 j についての計算結果が 全く活かされていません。

思い出そう。回文は相互扶助の共同体である。

次節では、回文の性質をキチンと使い、 $O(N)$ 時間アルゴリズムまで推し進めます。

4 $O(N)$ 時間アルゴリズム

前節のアルゴリズムは、中心を i とする回文を徐々に拡大し、中心 i の最長回文を求めています。一方で、本節のアルゴリズムは、大雑把に言えば、この部分の計算を大幅に端折ることによって、 $O(N)$ 時間アルゴリズムを得ます。その「端折り」を実現するために、回文の持つ性質をもう少しだけ引き出すことになります。

先に手続きの全体像を見て、それから、細部についての説明をします。手続きの全体像は次で与えられます。

```

Max[0] ← 0; C ← 0; result_idx ← 0;
for i ← 1 to 2N do
    k ← i - C; j ← C - k;
    if (i < (C + Max[C])) then
        if ((C - Max[C]) < (j - Max[j])) then パターン I の処理;
        else パターン II の処理;
    else
        パターン III の処理;
    if (C + Max[C] < i + Max[i]) then C ← i;
    if (Max[result_idx] < Max[i]) then result_idx ← i;
end
return  $\theta_w(\text{result\_idx}, \text{Max}[\text{result\_idx}])$ ;

```

この手続きにおいて、 $\text{Max}[i] = r$ は、中心を i にする最長回文の半径は r となる、ということを意味します。

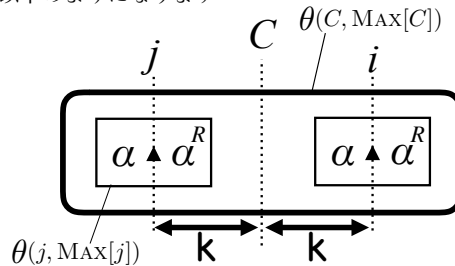
この手続きでも、前節のアルゴリズムと同様に、全ての位置について、それを中心とする最長回文の半径を求めます。ただし、 $O(N)$ 時間で済むようにしたいので、 $\text{Max}[i]$ の計算において、すでに計算済みの $j < i$ における $\text{Max}[j]$ の値を積極的に用います（とは言え、現段階では具体的にどうするかはまだ見えませんが）。

上のプログラムの核心は、3つのパターンへの場合分けです。プログラムの形では少し分かり辛いので、各パターンがどのような状況を表しているか、図を使ってみていきます。各パターンが、どのような状況を表しているかが分かれば、それぞれのパターンで、何をすれば良いか自然と明らかになります。

パターン I. このパターンは、

- 注目している位置 i が、位置 C を中心にする確定済み回文に含まれていて $i < C + \text{Max}[C]$; かつ
- C に対する i の鏡像位置 j について、 j を中心とする最長回文が、 C を中心にする最長回文に含まれる

場合です。これを図にすると以下ようになります：



図から殆ど明らかなのですが、次のことが成立します：

- (★) 位置 j を中心とする最長回文が $\alpha w[j] \alpha^R$ の形であるとき、 i の周辺状況も $\alpha w[i] \alpha^R$ となる。

このことは、 C を中心にこの頁を折り曲げる様子を想像すれば、十分に分かると思います。

[$\theta_w(j, \text{Max}[j])$ が回文なので, $\theta_w(j, \text{Max}[j]) = \alpha w[j] \alpha^R$ の形をとることは明らかです. 次に, i の周辺が $\alpha w[i] \alpha^R$ となることをみます. i と j の取り方から, この2つの文字は, 折り曲げた結果重なることになります. 「 C を中心にする回文」が一番外側の枠なので, 折り曲げると, 左右反転した形で, j の周辺状況が i に移らなければなりません. 結果として, j より左側 α が, i より右側に α^R として移り, j より右側 α^R が, i より左側に $(\alpha^R)^R = \alpha$ として移ります.]

性質(★)から, i を中心とする最長回文の半径は, $\text{Max}[j]$ 以上であることは間違いありません. 実際には, i を中心とする最長回文の半径は **ちょうど $\text{Max}[j]$ になる** と言えます.

[背理法で示しましょう. もし $\text{Max}[j]$ よりも, i を中心とする最長回文の半径が真に大きいのであれば, i の周辺の状況は $\dots \sigma \alpha w[i] \alpha^R \sigma \dots$ と書けます. そうだとすると, i と j は丁度 C を中心に折り返した関係になるので, j の周辺の状況も $\dots \sigma \alpha w[j] \alpha^R \sigma \dots$ とならなければなりません. すると, j の周辺には, 半径 $\text{Max}[j] + 1$ 以上の回文が存在するということになります. ですが, $\text{Max}[j]$ は, j を中心とする「最長回文の半径」を表しています. j を中心とする半径が $\text{Max}[j]$ を超える回文があつてはならないので, これは矛盾しています.]

上の議論から, パターン I では, $\text{Max}[i]$ として $\text{Max}[j]$ の値を取れば良いことが分かります.

パターン II. このパターンは,

- 注目している位置 i が, 位置 C を中心にする確定済み回文に含まれていて $i < C + \text{Max}[C]$; かつ
- C に対する i の逆位置 j について, j を中心とする最長回文が, C を中心にする最長回文に含まれ「ない」

場合です. 注目すべき場合は, 以下の2つです:

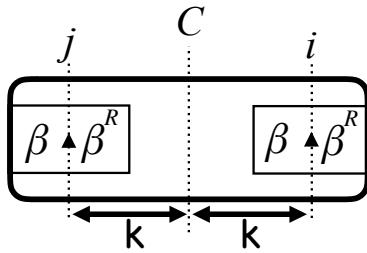


図 1: パターン II-a

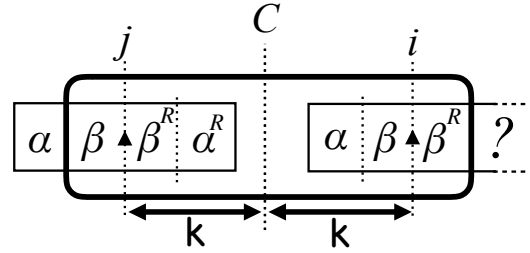


図 2: パターン II-b

これらの場合には, パターン I とは違い, $\text{Max}[i] = \text{Max}[j]$ という等式は (一般には) 成立しません. パターン II-a とパターン II-b に分けて, 詳しくみてみましょう.

パターン II-a (正確には $j - \text{Max}[j] = C - \text{Max}[C]$ の場合) について. 図から, $\text{Max}[i] \geq |\beta|$ ($= \text{Max}[j]$) が明らかです. しかし, $\text{Max}[i] = |\beta|$ となるかどうかは分かりません. なぜなら, $w[i + (|\beta| + 1)] = w[i - (|\beta| + 1)]$ が成立するときには, 現時点で判明している i を中心に持つ回文 $\beta w[i] \beta^R$ を, 拡大できるからです. 従って, $\beta w[i] \beta^R$ をどこまで拡大できるのか, 実際に w にアクセスして確認する必要があります.

パターン II-b (正確には $j - \text{Max}[j] < C - \text{Max}[C]$ の場合) について. 図から, $\text{Max}[i] \geq |\beta|$ は明らかです. よって, パターン II-a と同様にして, $w[i + (|\beta| + 1)] = w[i - (|\beta| + 1)]$ ならば拡大する……という処理を続

け、どこまで拡大できるか確認します。ただし、この場合には、 $w[i + (|\beta| + 1)] \neq w[i - (|\beta| + 1)]$ が必ず成立するので、拡大処理は直ちに失敗し、 $\text{Max}[i] = |\beta|$ となることが示せます。

[仮に、 $\text{Max}[i] > |\beta|$ であるとする、 i の周辺状況を $\dots\sigma\beta w[i]\beta^R\sigma\dots$ と書くことができます。図と照らし合わせると、 α の右端と、“?” の左端が、共に σ になります。ここで、 α が空でないことが、パターン II-b において、重要なところです。すると、 C を中心とする半径 $\text{Max}[C]$ の回文を、更に 1 つ分拡大することができてしまいます。しかし、 $\text{Max}[C]$ が、 C を中心とする「最長」回文の半径ですので、これらは矛盾します。]

パターン I とパターン II の決定的な違いは、文字列 w へのアクセスが必要になるかどうか、です。パターン I では、すでに計算済みの $\text{Max}[j]$ の値をそのまま採用できたので、 w へのアクセスは不要でした。一方、パターン II では、不明な領域があるので、 w へのアクセスが避けられません。とはいえ、パターン II でも、 i の周辺状況 $\dots\beta w[i]\beta^R\dots$ で確定している β と β^R の一致を検査する部分はスキップしているので、 w へのアクセスを減らす工夫がなされています。

パターン III. このパターンは、

- 注目している位置 i が、位置 C を中心にする確定済み回文に含まれて「いない」 $i \geq C + \text{Max}[C]$
(細かい議論はしませんが、位置 i が、どの確定済み回文にも含まれていないことと等しいです)

場合です。パターン III は、パターン II よりも一層酷く、 i の周辺事情についての知識が何もないです。この場合には、 i からスタートして、 $w[i - 1]$ と $w[i + 1]$ を比較、一致するなら $w[i - 2]$ と $w[i + 2]$ を比較……として、どこまで回文として拡大できるかを調べる必要があります。

以上の説明をもとに得られるプログラムは、次の通りです。

```

Max[0] ← 0; C ← 0;

for i ← 1 to 2N do
    k ← i - C; j ← C - k;
    if (i < (C + Max[C])) then
        if ((C - Max[C]) < (j - Max[j])) then Max[i] ← Max[j];
        else
            βℓ ← (C + Max[C]) - i; // 図における β の長さ
            for l ← 1 to 2N do
                if (w[i + (βℓ + l)] ≠ w[i - (βℓ + l)]) { Max[i] ← (βℓ + l) - 1; printY(i + (βℓ + l)); break; }
                printX(i + (βℓ + l));
            else
                for l ← 1 to 2N do
                    if (w[i + l] ≠ w[i - l]) { Max[i] ← (l - 1); printY(i + l); break; }
                    printX(i + l);
            if (C + Max[C] < i + Max[i]) then C ← i;

```

プログラムを掲載する目的は、上の説明がどう反映されるかを示すことにあります。ですから、パターン II や III での拡大処理中に w の左端や右端を超えた場合の処理など、実際に考えなければならない細々としたことは省いています。

さて、私たちは、このプログラムが入力 w (長さ $2N+1$ とする) に対して、 $O(N)$ 時間で動くことを示す必要があります。このプログラムは、入力 w へのアクセスがその計算の殆どを占めるので、 w に N の定数倍の回数しかアクセスしないことを示せば十分です。では、このことはどう示せば良いでしょうか。この間に答えるために、上のプログラムに、`printX` や、`printY` という関数を埋め込みました。

関数 `printX(i)` は、整数 i を引数にとり、 i^X という X 印付き出力を行います。同様に、`printY(i)` は、 i^Y という Y 印付き出力を行うものです。

上のプログラムが停止した段階に得られているログの形は、どうなっているでしょうか。プログラムの for ループの形に注目すると、

- 内側の for ループでは、 $a^X b^X c^X \dots p^X q^Y$ という複数の X 印出力の後に単一の Y 印出力（これを“ブロック”と呼ぶ）が得られ；
- 外側の for ループにより、ブロックが連なったものが出力として得られる

ということが分かります。すなわち、全体としては、以下のような形をとります：

$$i_1^X i_2^X i_3^X i_4^Y i_5^X i_6^Y i_7^Y i_8^Y i_9^X i_{10}^X i_{11}^Y \dots i_n^X i_{n+1}^Y.$$

どんなログに対しても、次の性質が成り立ちます：

- ブロックの個数は、高々 $2N$ である。すなわち、ログ全体で i^Y の形をした記号は、全部で高々 $2N$ 個。
 - 外側のループが $2N$ 回で打ち切られるので、これは明らか。
- 一度 i^X として出力された i は、ログの他の箇所では絶対に現れない。すなわち、ログ全体で i^X の形をした記号は、全部で高々 $2N$ 個。
 - これを正確に示すのは少し大変ですので、ここでは直感的なことだけ。出力 i^X が現れるのは、ある位置 i についての計算がパターン II またはパターン III に該当し、未知の位置 i' の内容を確認する必要があるからです。同時に、このことは、 i' が位置 i についての最長回文に含まれる（既知となる）ことを意味します。従って、2 度以上出力されることはあり得ません。

この性質から、どんなログについても、そのサイズは高々 $4N$ であり、このことから、 w へのアクセスが $O(N)$ 回程度と分かります。従って（少しギャップがありますが）上のプログラムが $O(N)$ 時間で動くことが納得してもらえらるかと思います。

5 おわりに

わたしは、そしてあなたは、回文として生を受けたにも関わらず、その記憶を失ってしまっていた。

その証拠が、はじめに与えた $O(N^3)$ 時間アルゴリズムである。あのやり方を前にして、あなたは恐怖を感じることができただろうか。回文文明を灰塵に帰すといっても過言ではない、おぞましい $O(N^3)$ 時間アルゴリズムに平然と対峙できていたあなたは、あまりにも人間^{非回文的}であった。

しかし、今やどうだ。あなたには回文としての誇りが取り戻されつつある。

見よ！！あなたは、今や回文たちが生き物であり、お互いを助け合うことで一つの社会を形成しているという事実を、遂に思い出すことができたはずである。残忍な $O(N^3)$ 時間アルゴリズムの前で平然としていたあなたは、いまや、回文にならんとしている。

さて、くだらないことはここまでにしておいて、本論を書くにあたって参考にした情報を記載したいとします。まず $O(N^3)$ 時間アルゴリズムと、 $O(N^2)$ 時間アルゴリズムは、誰でもすぐに思いつくもので、これといって何を参考したというものでもありません。

一方で、 $O(N)$ 時間アルゴリズムについては、ウィキペディアの記事 [1] を参考に書きました。ただ、[1] とは異なり、本記事ではパターン II についての説明を 2 つの場合に分けてあります。これは、そうした方が、より明瞭な説明を行えると考えたからです。ウィキペディア [1] によると、 $O(N)$ 時間アルゴリズムは、Manacher による 1975 年の論文 [4] がベースになっているということから、しばしば Manacher's Algorithm と呼ばれるようです。確かに、[4] を読むと、これが元になっていることが分かります。

ウキペイディアにも書いていることなのですが、Manacher's Algorithm と異なる $O(N)$ 時間アルゴリズムとして、接尾辞木 (suffix tree) を用いるものがあります。接尾辞木はある種のデータ構造なのですが、これは、現代を生きる上での最重要項目に属すといっても過言ではありません。接尾辞木の言葉を使うと、最長回文を求める問題は、驚くほど簡単に解くことができます。興味があれば、是非とも、Gusfield の書籍 [3] を手にとり、9.2 節をのぞいてみてください。図書館にもありますし、余裕があれば、購入しても損はしない書籍です。

実装した最長回文計算プログラムの力試しをしたいのであれば、オンラインジャッジシステム POJ の問題 3974 番 [2] をオススメします。POJ の使い方の説明をする余裕はないのですが、問題 3974 番に設けられた実行制限時間を考えると、 $O(N^2)$ 時間アルゴリズムではなく、 $O(N)$ 時間アルゴリズムが必要になるはずで

終わりだよ～

参考文献

- [1] Longest palindromic substring. https://en.wikipedia.org/wiki/Longest_palindromic_substring.
- [2] PKU JudgeOnline Problem 3974 (Palindrome). <http://poj.org/problem?id=3974>.
- [3] Dan Gusfield. *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology*. Cambridge University Press, 1997.
- [4] Glenn Manacher. A new linear-time “on-line” algorithm for finding the smallest initial palindrome of a string. *J. ACM*, 22(3):346–351, July 1975.

Nexus 5 で市民武装 ～ARMed and Dangerous～

文 編集部 くりす

1 はじめに

みなさん、おはこんばんちわ。Nexus 6 を買ったり、卒論を無事に提出したりして大盛り上がりのくりすです。というわけで今回の話題は畑……ではなく、Nexus 6……でもなく、Nexus 6 を新しく買ったことによって production 環境からめでたく解放された、我が先代 Nexus 5 をとことん魔改造して核武装させるおはなしです。

突然ですが、右の図の不審者`H`H`H 人物は誰でしょうか？

いや、まあ答え書いてあるんですけど……Ubisoft 製のゲーム「WATCH_DOGS」にておなじみの Aiden くんです。「アイデン」じゃないですよ！ スマートフォンひとつで街中を滅茶苦茶にハッキングしては神秘体験を引き起こす危険人物です。

そんなことが本当に可能なのでしょうか？

できます。たぶん。

Kali Linux NetHunter^{*1} というシロモノを使って、極めて異常性の高い武装スマートフォンを手に入れることができます。

合衆国民は銃火器を所持する権利が憲法によって認められています。修正第二条の示すとおり、規律ある民兵は自由な国家の安全にとって必要であるから、我々も武器を取らない手はないですね？



図 1: Aiden Pearce

2 NetHunter ってなんだ

ひとことと言えば、NetHunter は Android に Kali Linux^{*2} をくっつけたものです。既存の Android OS の追加拡張、要するに MOD です。カスタム ROM とは異なり、普通の（root 化された）Android 上で動きます。電話やタブレットとしての機能を保ったまま、そこに Kali Linux の本体がほぼ丸ごと詰め込まれるというわけです。

NetHunter を導入するとできるようになること

- USB Wi-Fi ドングルを使用し、Wi-Fi 通信を攻撃したり偽のアクセスポイントを立てたりする
- BadUSB^{*3} 機能

^{*1} <http://www.nethunter.com/> および <http://www.kali.org/kali-linux-nethunter> - 以下「NetHunter」と記す。

^{*2} 言わずと知れたハッカーの神器。様々な“その手の”ツールを詰め込んだ Debian ベースの OS

^{*3} 任意の USB デバイスを模倣し、USB 接続されたマシンに対しキーボード入力を行ったりネットワーク通信を盗聴したりする攻撃手法

- Android 端末内に Kali Linux フルセットを持ち歩く (VNC を使えばデスクトップまで使える！)
- Ingress をやってるような無害な見た目をしながらハッキングできる

この、いい感じに厨二心をくすぐってくる品揃え！ 夢がありますね！

ところで、先述のゲーム「WATCH_DOGS」の冒頭では、Aiden くんが武装フォンを携えて高級ホテルへ潜入し、リモートから Damien くんがせつせとハックし宿泊客からカネをかすめ取る描写があります。「んなわけがあるかです…… (CV:わたしちゃん)」という意見が多いこのシーンですが、実は NetHunter を使うとあれができる環境が整うんです。

まず Aiden くんがホテルへスマートフォンを持って潜入し、ホテル内の Wi-Fi に接続します。ホテルですからそもそも PSK が要らないかもしれません。NetHunter には OpenVPN がプリインストールされていますから、それを使って Damien くんのマシンにリバース接続すれば、革命的で魔法のような人間バックドアの完成です！

ーとまあ、それは NetHunter の得意とする曲芸のほんの一例にすぎません。

3 市民、武器を取れ！

NetHunter が公式にサポートされている端末は、Nexus {4,5,6,7,9,10} と OnePlus One です。Nexus シリーズはすべて技適を通過しているので、日本での使用もバッチリ合法です。

root 化・カスタムリカバリ導入済みの端末なら、ほかの zip イメージ同様にリカバリ経由でワンタッチでインストールできます。端末の root 化手順に関する説明はどうせ Qii●a などにあるはずなので本稿では省略します。なお最低限知っておかなくてはならないこととして、**OEM Unlock** の過程では全データが消去されるということは念を押しておきます。このような破壊的操作が伴うので、大事なデータを失いたくない場合は必ずバックアップをとっておきましょう。

本稿執筆時点 (2016-01-07) 時点では、NetHunter の最新版は 3.0 です。

Android 4.4. で* (KitKat), 5.* (Lollipop), 6.* (Marshmallow) 搭載の前述の端末では、公式で配布されている zip イメージをそのまま使用できます。そのほか、公式 GitHub リポジトリから pull して自前でビルドすることも可能で、こちらの方法は最新の開発版をテストしたり、ビルド時に自前で用意したアプリなどを同梱したり、サポート外の端末向けのインストーラを作成したりするのに適しています。

ビルドやインストールの手順の説明は本家 Wiki に譲ります。

<https://github.com/offensive-security/kali-nethunter/wiki>

インストーラには多少のバグがあり、よく (見かけ上) フリーズします。固まってから約 20 分ほど経っても改善がみられなければ、強制再起動して OK です。インストールの最中に強制再起動するなんて狂気の沙汰っぽいですがとくに問題はありませんでした。わたしの環境では強制再起動をもってインストールが完了しました。

自前でビルドしたインストーラを使用した場合は、基幹となる Kali chroot は付属ではなくインターネットを通じてインストールするかたちになります。Wi-Fi に接続し、NetHunter メニューの「Kali Chroot Manager」

» 「Install Chroot」を選択し、画面の指示に従うと、30 分ほどで端末に Kali Linux の chroot 環境が爆誕します。

インストールが終わると、ホーム画面に「NetHunter」というアプリが生まれます。祭の開幕です。

以降は、3.0 をインストールしたことを想定した内容となります。

4 射撃訓練

せっかく武器を手に入れたのですから、試し撃ちしたいですね？ スプ●トゥーンで新しいブキを手に入れたら、まずは試し撃ちしますよね？ もちろんやりますよ。はじめにポートスキャンを試してみましょう。

ポートスキャナの王者「Nmap」が NetHunter に同梱されています。これを使って、Nmap の公式テスト用サーバにポートスキャンを実行してみます。

1. NetHunter アプリを開く
2. Nmap Scan をメニューから選択する
3. scanme.nmap.org と入力し、SCAN をタップ (図 2)

そうするとターミナルが開き、ほどなくしてポートスキャンの結果が表示されます。便利ですね！

もちろん、NetHunter のパワーはポートスキャンなんていう生易しいものとどまりません。もっと火力を感じられる射撃訓練を行いたい場合は、「Metasploitable 2^{*4}」などを叩き台にして、NetHunter 付属の「cSploit」というなかなか危険なアプリを活用してみてください。Metasploitable 2 は各種ツールの試し撃ちの的として作られた、意図的に脆弱性だらけにされている VM イメージです。こちらも、インストールや起動の方法は公式ドキュメントに譲ります。

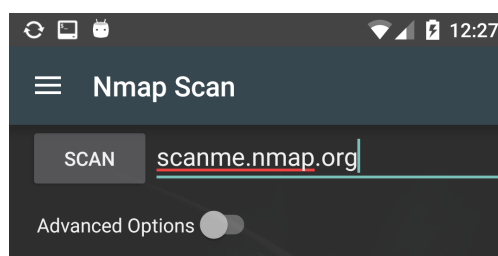


図 2: Nmap Scan

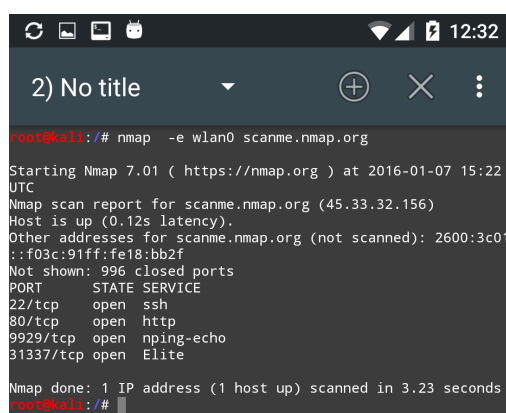


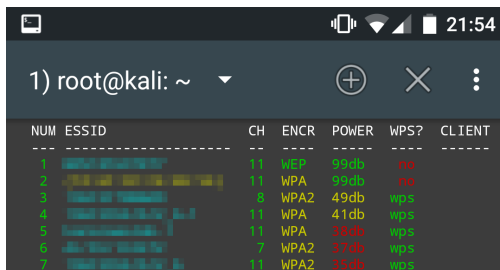
図 3: スキャン結果

^{*4}<http://sourceforge.net/projects/metasploitable/files/Metasploitable2/>

※重要 Metasploitable 2 は脆弱性が満載されているので、自身でコントロールできないネットワークに設置することは避けてください。

とくに、このユースケースでは VM 間のみのネットワークにどうしても参加できないホスト（スマートフォン）が関係し、推奨されるように NAT や Host-only ネットワークを使うことができないので、やむを得ず通常の無線 LAN を使用することになります。

5 たのしいたのしい、フィールドワーク



| NUM | ESSID | CH | ENCR | POWER | WPS? | CLIENT |
|-----|------------|----|------|-------|------|--------|
| 1 | XXXXXXXXXX | 11 | WPA2 | 99db | Yes | |
| 2 | XXXXXXXXXX | 11 | WPA | 99db | Yes | |
| 3 | XXXXXXXXXX | 9 | WPA2 | 49db | WPS | |
| 4 | XXXXXXXXXX | 11 | WPA | 41db | WPS | |
| 5 | XXXXXXXXXX | 11 | WPA | 33db | WPS | |
| 6 | XXXXXXXXXX | 7 | WPA2 | 37db | WPS | |
| 7 | XXXXXXXXXX | 11 | WPA2 | 37db | WPS | |

図 4: Wifite

こうして楽しく遊んでいる Nexus 5 は、携帯電話です。携帯することに意義があります。外へ持って行かなくてはなりませんね！

21 世紀になって早くも 15 年、街中は電波であふれ返っています。カフェ、ホテル、空港はホットスポットが装備されていて当たり前ですし、どこぞのナントカバンクなどは、「Wi-Fi つかえます」という文言とともに電波を発信する犬が記載されたステッカーを貼っては超どうでもいい所にもホットスポットをばらまいてきました。もはや 2.4GHz 帯は乱戦状態です。と

いうことは、その中にひとつ変なアクセスポイントが紛れ込んでいても、それは日常風景であり大して怪しまれないということです [要出典]。

NetHunter には相当数の Wi-Fi 系のツールが仕込まれています。由緒正しき「Aircrack-ng」はもちろんのこと、「Mana Wireless Toolkit」や、Aircrack-ng の凶悪性を一層引き立たせる「Wifite」というシロモノが公式に導入されています。

なお、Nexus 端末などでこれらの Wi-Fi 系ツールを使用するには、USB-OTG^{*5}を通じて USB Wi-Fi ドングルを使用する必要があります。こちらも技適を通過しているものを使用してください。さもないと電波法第 57 条違反おじさんになってしまいます。

Mana Wireless Toolkit

NetHunter には「Mana Wireless Toolkit（略称：Mana）」という悪意の塊のようなポータブルアクセスポイント機能が搭載されています。ポータブルアクセスポイントですから、検証中に周囲の Wi-Fi ユーザが「おっ、あいてんじゃ〜ん」などとぬかしながらこの極悪アクセスポイントに押しかけてくるかもしれません。

実験の際は関係ないユーザを巻き込まないように端末にお札を貼り、ESSID を「WhiteHouse」みたいな一発で CIA が飛んできそうなものにして抑止力を確保しておく安全かもしれません（大嘘）。

^{*5} USB On-The-Go. 普通の USB 機器を端末の microUSB ポートを通じて使用する技術

Wifite

「Wifite」もまた悪意の塊のようなツールです。やっていることは Aircrack-ng と一緒なのですが、それが 2 ステップくらいポチポチやると全自動でクラックまで完了するという、「サルでもできる Wi-Fi クラック」をそのまま体現してしまうものです。携帯端末を使って 3 分で WEP キーがクラックできる時代です。そりゃクレジットカード業界で使用禁止になる^{*6}わけですね。WEP がウンコだという話はむかしむかし 2011 年頃の WORD に掲載した記憶がうっすらあるのですが、何号の何という記事が完全に忘却してしまいました……。ですので、みなさん WORD のバックナンバーをぜひお読みください！！！！

<http://www.word-ac.net/>

WiGLE Wifi

NetHunter の一部ではないのですが、個人的に気に入っているアプリ「WiGLE Wifi」を紹介します。WiGLE Wifi は近辺の無線アクセスポイントの情報を記録するアプリです。記録される情報には ESSID や BSSID、セキュリティレベル、WPS の可否、観測点の GPS 位置情報があります。観測されたアクセスポイント情報はアプリ内で、もしくはエクスポートすることで、図 5 のように Google Maps/Earth 上にプロットできます。

アクセスポイント情報は SQLite で保存されますので、SQLite を直接エクスポートして独自に統計情報を取ることも可能です。たとえば、わたしが WiGLE Wifi で観測したことのある、7 万件弱のアクセスポイントの 14.4%が WEP を使用していることなどが導き出せます。

Ing●ess のお供にどうでしょう？

6 BadUSB

いわゆる病原菌を持っているのは人間だけではありません。NetHunter のような恐ろしいデバイスを USB ケーブルで PC に接続しても、やっぱり悪いことは起きます。その例が BadUSB^{*7}です。BadUSB は、USB 機器のファームウェアに変更を加えることで USB 機器の認識のされ方を変え、ただの USB メモリにキー入力を行わせたり、マルウェアをマウスの中に潜ませたりする手法の総称です。

以下に、NetHunter でサポートされている BadUSB の例を示します。

^{*6} PCI-DSS 要件 4.1 <https://www.pcisecuritystandards.org/>

^{*7} <https://srlabs.de/badusb/>

HID Attack



図 5: WiGLE Wifi

ちなみに UAC 突破もお手の物。スマートフォンの形状をした“入力デバイス”だからキーボードやマウスを模倣して UAC ダイアログの「許可」を光の速さで選択できるわけですね。

USB ケーブルで NetHunter を PC につなぐと入力デバイス（つまりキーボードやマウス）として振る舞い、光の速さでコマンドやシェルコードやらを“入力”して攻撃を実行するということです。見かけ上はエンドユーザが自分の意思で入力を行っているので、アンチウイルスは作動しないし、ソフトウェアの脆弱性も不要であるところがポイントです。Exploit なんて要らなかったんや……。

NetHunter の HID Attack 機能では 3 種類のアタックが用意されています。

PowerSploit

Windows PowerShell に対し自動入力をすることで、Meterpreter を起動し攻撃者にリバースシェル接続を行う

Windows CMD

通常の Windows コマンド。アンチウイルスに（おそらく）全く引っかからない手法。しかし画面を注意深く見ている人なら、一瞬コマンドプロンプトのようなウィンドウが出現するのを見逃さないかもしれない。

DuckHunter HID

USB Rubber Ducky Project^{*8} で用いられる独自のスクリプト言語を用いた手法

^{*8} <https://github.com/hak5darren/USB-Rubber-Ducky/wiki>

BadUSB MITM

USB のユニバーサルつぶりはなにも入力デバイスだけにとどまりません。ネットワークインタフェースも USB で難なく動作します。これを悪用すると、USB ネットワークインタフェースのふりをしてネットワーク通信を USB にねじ曲げ、それを横取りするという頭のおかしいネットワークインタフェースが作れるわけです。通信はブリッジすれば途絶えないので被害者にも気づかれませんし、Android 端末には USB テザリングという標準機能があるので、Android 端末がネットワークインタフェースを兼ねるということは全然おかしいことではないのです。

7 おわりに

武器を手に入れたからといって乱射すると、まあ、捕まります。ここに書かれた情報の目的はあくまで、

- とりあえず武装してみたい・特殊能力を手に入れたい
- Aiden くんみたいなマホーの電話を手に入れて神秘体験してみたい
- コンピュータセキュリティに関する造詣を深めたい

といった至極まともな厨二病患者の夢をある程度まで叶えることであり、「あの娘の家の Webcam を覗きたい」だとか「TW●NS に侵入したい」だとかそういった反社会的な行動は一切サポートされていません。やらかした人間に関してはその責任について全く関知しませんし、昨今のクールジャパンはいつぞやの†ダガーナイフ†に始まり悪用されたものは片っ端から規制されていく世の中ですので、誰かの無責任な行動によってこのような美しい厨二文化がまたひとつ違法となってしまうことはあってほしくありません。

逮捕されるなよっ ★

LuaRocks で結果にコミット

文 編集部 びしょ〜じょ

1 はじめに

こんにちは、びしょ〜じょです。みなさんは結果にコミットしてますか。ということで今回は Lua のパッケージマネージャー *LuaRocks* の使い方を学びましょう。

2 LuaRocks とは

LuaRocks とは Lua のパッケージマネージャーです (2 回目)。<https://luarocks.org/> にパッケージを登録したり、登録されているパッケージを使ったりします。lua ファイルのみならず、C で書かれたモジュールもインストール時にビルドしてくれるなどします。

主に Lua で書かれており、またモジュールとしても提供されているので、ビルドツールなどを自前で作るにも助かりますね。

今回はコマンドラインで用いる `luarocks` コマンドについての説明です。

3 レッツゴー LuaRocks

以降は ArchLinux 環境下という前提のもとで進めます。Lua はマルチプラットフォームであり、LuaRocks も Windows や何やをサポートしているので、だいたいレギュラーな環境でなければ多分なんとかなるでしょう。

最新は v2.3 で、GitHub で開発されています^{*1}。

3.1 INSTALL

`pacman -S luarocks`で簡単インストールです。ディストロ別のパッケージマネージャーに任せましょう。やる気が出たら自力でビルドしてください。 `luarocks --version`でなんか出たら OK です。

3.2 構造

LuaRocks でインストールされるモジュールはどこにどう展開されるのか、パッケージングして出荷するときやインストールして使うときのためにも、ここでぜひ知っておきましょう (図 1)。

^{*1} <https://github.com/keplerproject/luarocks/>

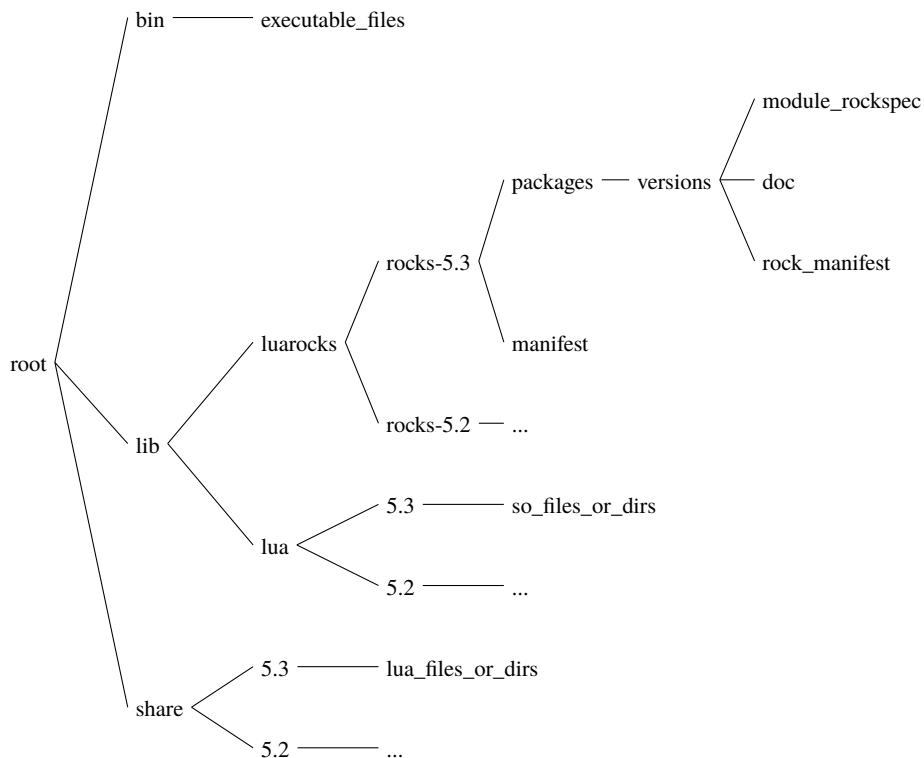


図 1 directory structure

5.2 とか 5.3 は Lua のバージョン。root は /usr/lib/luarocks/rocks-LUA_VERSION または \$HOME/.luarocks です。

4 使う

luarocks コマンドは 19 個のサブコマンドからなっています。とにかく時間がないのでここで説明するのは `help`、`search`、`install`、`remove`、`path`だけにします。

4.1 GENERAL OPTIONS

その前にどのサブコマンドでも使えるオプションを紹介します。

`--verbose`

挙動を逐一出力します。主にデバッグ用。

`--local`

これが一番重要です。例えば `install` にこのオプションを渡さないと、`/usr/local/luarocks/rocks-LUA_VERSION` にパッケージをインストールしようとします。つまりこのオプションを渡すことで、`root` 権限が必要ない領域の中ですべてやるということになります。

`--server=<server>`

`search`、`install`などのサーバーは `https://luarocks.org/` ですが、このオプションでパッケージのサーバーを追加できます。

`--only-server=<server>`

逆にこれは唯一となるサーバーを指定します。

`--tree=<tree>`

インストールするディレクトリを指定します。`install --local example`は `install --tree=$HOME/.luarocks example`と同義です。詳細はこちら^{*2}^{*3}。

他にもいくつかありますが、とりあえず使うのはこの辺でしょう。もう少しテクニカルなことがしたくなったらぜひ調べてください。ではサブコマンドの説明に入ります。

4.2 COMMANDS

`help`

まずはヘルプ。`help SUB_COMMAND`を実行すると、各サブコマンドの詳細がわかります。

```
$ luarocks help SUB_COMMAND
```

`search`

サーバーからパッケージを検索します。

```
$ luarocks search [--source] [--binary] {<name> [<version>]|--all}
```

時間がないので詳細は省略しますが、パッケージ名に加えてバージョンでも調べられますはい次。

^{*2} <http://leafo.net/guides/customizing-the-luarocks-tree.html>

^{*3} これはWORD36号で紹介したMoonScriptというプログラミング言語の作者 leaf氏による解説であり、Twitterで空リブをしたら^{*4}「記事を書いたらリンクを張るぜ」と言われたので、こちらもリンクを張らせてもらうことにしました。

^{*4} <https://twitter.com/moonscript/status/692798297363214336>

```
luarocks --local install word_articles
```

install

文字通りパッケージをインストールします。

```
$ luarocks install {<rock>|<name> [<version>]}
```

手元にある `rockspec`^{*5} またはパッケージ名、オプションでバージョンを指定します。他パッケージとの依存関係もなんとかしてくれるので、ユーザーは何も考えなくて OK。

上記でも説明した通り、`--local` をつけると `$HOME/.luarocks` に、そうでなければ `/usr/local/luarocks/rocks-LUA_VERSION` にインストールされます。

--only-deps

依存するパッケージのみインストールします。

remove

`install` に対する `remove` です。

```
$ luarocks remove [--force=[fast]] <name> [<version>]
```

パッケージ名、オプションでバージョンが指定可能。これもまた、`--local` をつけることで `$HOME/.luarocks` 以下からパッケージを削除します。

--force

依存関係を無視して無理やりパッケージを削除します。

path

Lua の関数 `require` で検索するファイル名は、環境変数 `$LUA_PATH`、または `$LUA_CPATH`^{*6} で定義されます。この 2 つの環境変数をそれっぽく出してくれるのがこのサブコマンドです。

```
$ luarocks path
```

`export LUA_PATH=.....` のように出力されます。例えば `zshrc` などに `eval $(luarocks path)` などと書いておくといい感じですね。

もう厳しい^{*7}。残りは次回。

^{*5} パッケージの設定ファイル。Lua のサブセットで記述されています。

^{*6} Lua は `so` ファイルも読み込むことができ、`lua` ファイルは `$LUA_PATH`、`so` ファイルは `$LUA_CPATH` から読み込まれます。

^{*7} この記事は赤入れの締め切り 1 時間前に書き始めました。

5 おわりに

『りりくる Rainbow Stage!!!』^{*8} というものが3月に発売されるそうです。この記事がみなさんの手元に来るところには発売されているはずだ。詳細はボクにもわかりませんが、おそらく素晴らしいものであることは間違いない。買うんだ。

^{*8} http://www.particle.jp.net/lilycle_rs/

情報科学類誌 WORD 読者アンケート

題字 元編集部 ふあい

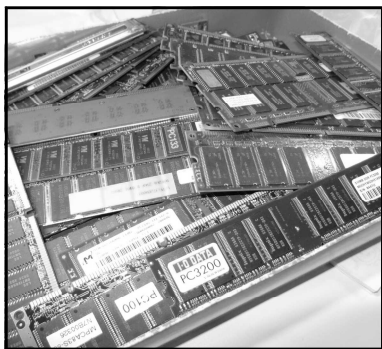
文 編集部 リザウド

1. あいさつ

ドーモ、アンケートを引き継いだリザウド(仮名)です。 <http://www.word-ac.net/>にあるような良いアンケートの伝統ってやつを引き継いでいきたいと思います。^{*1}

2. 今回の粗品

アンケートを提出する際、各置き場の回収箱ではなく 3C ラウンジ横の WORD 編集部 (3C212) まで直接持ってきてくださった方には、先着でゴミ粗品を差し上げています。



メモリ型定規

最新型はまず入っていない。マザボに挿す際は自己責任で。もしかしたらレトロなメモリ型があるかもしれない。

^{*1} 偉大な先人たちの記事は <http://www.word-ac.net/> で読めます



WORD ステッカー

貼れるや！ステッカーだからね！

3. アンケート集計

Q1：所属を教えてください。

情報科学類：1人

国際総合学類：1人

生物資源学類：1人

Q2：性別を教えてください。

男：2人

絶食系男子：1人

Q3：年齢を教えてください。

20歳：1人

24歳：1人

プロデューサー歴2ヶ月：1人

Q4：良かったと思う記事があれば教えてください。

先号の一覧はこちら。

-
- | | | |
|----------------------------|---|-------------|
| 1. 表紙 | 8. 国民休暇の果てに | 15. アンケート用紙 |
| 2. 号名 | 9. WORD-Lua Official Reference Document | 16. 配布場所 |
| 3. 目次 | 10. 野獣出現!子課長(体調が)壊れる | 17. 配布時期 |
| 4. 「openfab創房」前編:室内外観・機材紹介 | 11. WrongWord～敗北者たち～後編 | 18. 配布媒体 |
| 5. Schemeプログラマの雑文 | 12. WORD 読者アンケート | 19. 冊子の厚さ |
| 6. 書評:読書の秋、ファシズムの秋 | 13. 編集後記 | |
| 7. 基本無料!学内で引けるキノコガチャ! | 14. 裏表紙 | |
-

集計結果：

4：1票

11：1票

7：2票

10：1票

良い:7 デレ~~マ~~ステの課金は怖いよね

【新米 P(仮)】

部内有識者に聞いてみました。

「推しが来たら全力で」 【とある P】

「課金したくなったら運の尽き」【ベテラン P】

ということだそうです、課金すればするほど中の人が飯を食べるのでジャブジャブ課金したくなるような推しを見つけていきましょう。

7,10,11

かわいい、キノコ、ネコ、OB

【生物学類 alohaeta (?) さん】

かわいいキノコがネコの OB ? (推測)^{*2}

Q4：良くなかったと思う記事があれば教えてください。

2：1 票

11：1 票

5：1 票

15：1 票

6：1 票

号名に入ってるのはさもありなんって感じですね。

悪い: 15 回答欄がちいさい、Q7 がない

【情報科学類 新米 P(仮)さん】

アンケート用紙も資源節約のために色々変更を行っているところなので、ご了承ください。Q7 がなかったのは申し訳ないです。^{*3}

Openfab について他学類だが是非利用しようと思った

【国際総合学類 絶食系男子 さん】

Openfab については今号でも記事があるのでチェックするしかないですね?^{*4}

*2 意思の汲み取りの失敗(バットコミュニケーション)

*3 ヒューマンエラーが起こりまくるのがアンケート

*4 落ちました(終了)

悪い: 2 何とかならなかったのか

【情報科学類 新米 P(仮)さん】

何とかならなかった。

Q5：過去の記事に関する感想を教えてください。

入試の時の号(36)の中の現在地を見つけるゲームについての記事
ああいうの好きです。

【国際総合学類 絶食系男子 さん】

お、GeoGuessr(<https://www.geoguessr.com/>)の記事ですね。編集部ではかなり下火になってしまい、職人の技というのも消失の危機というやつです。

彼らの雄姿を見たい人は<http://www.word-ac.net/> ^{*5} へどうぞ！

らふにんさんは異常だった

【情報科学類 新米 P(仮)さん】

らふにんさん、衰えたとか言っておきながらたまに編集部に来てはおやつに ZEY ◎とかいってるんですよ。

Q6：今までで一番後悔した買い物を教えてください

覚えてないからガチャ回してくる

【情報科学類 新米 P(仮)さん】

覚えてないのとガチャを回してくるという関連性が分かりませんが、逝ってらっしゃいませ。

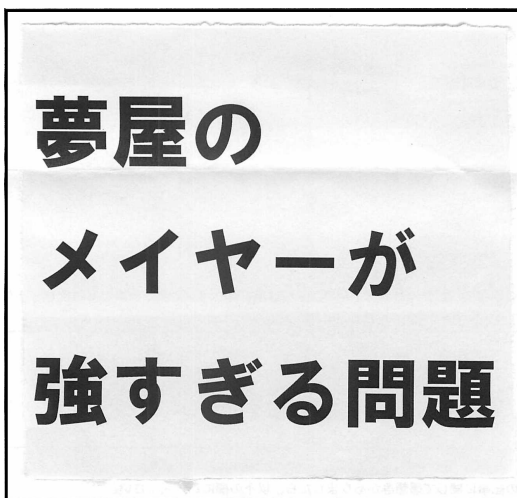
*5 本日のノルマ n 回目

男性用ブルマ

【生物学類 alohaeta (?) さん】

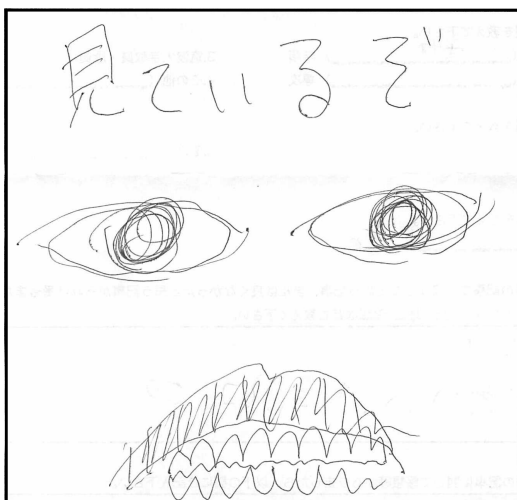
良いですね。装備した写真を WORD に投稿して永遠の黒歴史にもしていきましょう。

Q7：自由記述欄。

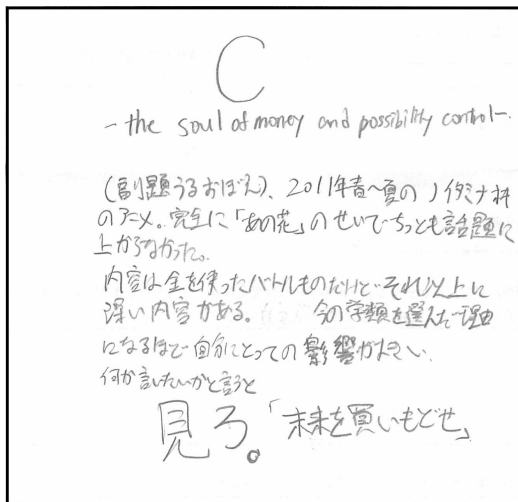


メイヤーの方からお言葉を頂いています。

「夢屋は手段ではなく、夢屋こそ目的そのもの」



イエーイ見てるー?????



Q ちゃんにバクバクされたい。

4. 終わりに

新体制よろしくです。

WORD 編集部への誘い

文 編集部 カブ

我々 WORD 編集部は情報科学類の学類誌「WORD」を発行している情報科学類公認の団体です。WORD は「情報科学」から「カレーの作り方」までを手広くカバーする総合的な学類誌です。年に数回発行しており、主に第三エリア 3A、3C 棟や図書館前で配布しています。

編集部の拘束時間には週一回の編集会議と、年に数回の赤入れや製本作業等発行に伴う作業があります。日常的に活動する必要はありませんので、他サークルの掛け持ちの障壁にはなりません。実際、多くの編集部員が他サークルと掛け持ちで在籍しています。

WORD 編集部には、情報科学類生や情報メディア創成学類生などが在籍しています。例えば、以下のよう
な人達が在籍しています。

小林秀和 @KOBA789

リザウド Lisp 大好きクラブ会員

びしょ〜じょ 百合仙人

もねと 生きるジャーゴン辞書

ioriveur つくばの革命家

下記に当てはまる方や、WORD に興味を持った方は是非、情報科学類学生ラウンジ隣の編集部室 (3C212) へいつでも見学に来てください。時間を問わず常に開いています。

- AC 入試で入学した方
- それ以前に AC な方
- 印刷、組版や製版に興味がある方
- ネットワークの管理を経験したことがある方
- APL や Lisp が書ける方

その他質問がある方は、word@coins.tsukuba.ac.jp か、Twitter アカウントをお持ちの方は@word_tsukuba までお気軽にお問い合わせください。

情報科学類誌

WORD

From College of Information Science

@word__writing号

発行者

編集長

制作・編集

情報科学類長

井上 陽介

筑波大学情報学群

情報科学類WORD編集部

(第三エリアC棟212号室)

2016年2月17日 初版第一刷発行

(256部)