可逆な文字列照合アルゴリズム

2018SE016 平工真基　　2018SE091 谷崎海良
指導教員：横山哲郎

1　はじめに

可逆計算とは任意の計算ステップで直前の状態が高々一意に定まる計算のことである．全ての可逆計算は計算後の状態が高々一意に定まり，計算前の状態も高々一意に定まる．よって，全ての可逆計算は単射部分写像で表される．Landauer の原理 [3] により，非可逆な計算を行うと必ず熱が発生すると知られている．しかし，可逆計算ならば非可逆な計算を行うことで発生する熱をなくし，エネルギーの消費を小さくすることができる可能性がある [5]．また，量子計算で用いられることも期待されている．量子力学系は時間について可逆であるため，可逆計算も量子計算の一つであると考えられるからである [5]．

可逆計算は全て単射でなければならない．しかし，既存のアルゴリズムは単射であるとは限らない．よって，既存のアルゴリズムを可逆計算で用いるためにはアルゴリズムの各ステップが全て単射であることを確かめ，非単射なステップが存在する場合は単射なステップになるようにする必要がある．今まで，線形探索等を対象に既存の非可逆なアルゴリズムを可逆化する研究がされてきた [4]．本研究では既存の文字列照合アルゴリズムを可逆化することを目的とする．

本研究では，素朴な文字列照合アルゴリズム，Rabin–Karp のアルゴリズムの 2 つの文字列照合アルゴリズムを可逆化することを目的とする．2 つのアルゴリズムの手順は文献 [2] を参考にする．次に，各文字列照合アルゴリズムを可逆化したときの時間計算量，空間計算量，ゴミ出力量を解析し，非可逆な文字列照合アルゴリズムと可逆な文字列照合アルゴリズムの解析結果を比較したり可逆な文字列照合アルゴリズム同士の解析結果を比較したりすることにより各アルゴリズムの異なる点を調べる．その後，可逆な文字列照合アルゴリズムを時間計算量，空間計算量，ゴミ出力量のいずれかの点について最適化を行う方法の提案をする．

2 つの文字列照合アルゴリズムを可逆化するために各アルゴリズムについて単射なステップと非単射なステップがどこに存在するのかを確認する．その後，非単射なステップを単射なステップにするための方法を提案することで 2 つの文字列照合アルゴリズムを可逆化する．次に，可逆プログラミング言語を用いて可逆化した文字列照合アルゴリズムを実装し，実装したプログラムをもとに時間計算量，空間計算量，ゴミ出力量を解析したりいずれかの点について最適化を行う方法を提案したりする．

2　関連研究

2.1　可逆アルゴリズム

アルゴリズムとは問題を解決するためのステップを記述したものである．アルゴリズムではある入力が与えられた後，決められたステップに従って計算を行い，計算結果を出力することで問題を解決する [2]．可逆アルゴリズムとはアルゴリズムの各ステップが全て単射なステップであるようなアルゴリズムである．全てのステップが単射なので可逆アルゴリズムでは出力から入力を一意に定めることができる．非可逆なアルゴリズムの各ステップを単射なステップのみにすることを可逆化という．可逆化するために非可逆なアルゴリズムのときには不必要だった情報を保存しなければならない場合が存在する．この情報をゴミという．非可逆なアルゴリズムを可逆化するための一般解法としては Landauer 法 [3] や Bennett 法 [1] 等がある．

Landauer 法ではアルゴリズムを実行するときに発生する情報を保存することで可逆化を行うため，出力するときに余分な情報がゴミとなってしまうので空間計算量が大きくなってしまう．

Bennett 法ではアルゴリズムが可逆になるようにゴミを生成しながら実行した後，アルゴリズムの結果のみをコピーする．その後，アルゴリズムを逆方向に実行することによってアルゴリズムを順番に実行したときに生成されたゴミを消去する．ゴミを消去することにより，最終的に入力とアルゴリズムの結果のみを残すことができる．Bennett 法ではアルゴリズムを順番に実行した後，逆方向にも実行する必要があるため非可逆なアルゴリズムと比較してアルゴリズムを実行したときの時間計算量が約 2 倍になってしまう．

2.2　可逆プログラミング言語

プログラムを実行するとき，任意の状態から直前の状態を高々一意に定めることができるプログラムのことを可逆なプログラムという．可逆プログラミング言語とは可逆なプログラムしか書けないという制約が存在するプログラミング言語である．可逆なプログラムしか書けないという制約があるので変数に別の値を直接代入する等の非可逆なプログラミング言語で行えた一部の処理は行えないようになっている．非可逆なプログラムを書いた場合はエラーが起き，実行前に気づくことができる．よって，可逆なプログラムを書いたつもりが非可逆なプログラムを書いてしまいそれに気づかないまま実行してしまうということを防ぐことができる．可逆プログラミング言語の例としては

Janus や ROOPL++ 等がある．

3　文字列照合アルゴリズム

探索を行う文字列をテキスト t，テキストの中から探したい文字列をパターン p とする．このとき，t の中に p がどの位置に存在するか特定するアルゴリズムを文字列照合アルゴリズムという．

この節ではアルゴリズムの説明や単射であることの証明に写像，単射，計算量の概念を用いる．

写像　A，B を集合とする．A から B への写像とは A に属する任意の要素に対して対応する B の要素がただ 1 つに定まるような対応のことである．

単射　A から B への写像が単射とは B の任意の要素について対応する A の要素が高々 1 つに定まるような写像のことである．つまり，$b_1 \in B$ に対応する要素を $a_1 \in A$，$b_2 \in B$ に対応する要素を $a_2 \in A$ とするとき，$b_1 \neq b_2 \Rightarrow a_1 \neq a_2$ である．

計算量　計算量とはプログラムを実行する際に時間やメモリ等の資源をどれだけ消費するかを表すものである．時間を消費する量を表したものを時間計算量，メモリを消費する量を表したものを空間計算量という．

3.1　文字列

アルファベット Σ に属する文字が有限個並んだ列を文字列とし，アルファベットが並んでいる個数を文字列の長さとする．a，b を文字列とする．a に b を連結するとは a の後ろに b を続けることであり，a に b を連結した結果できた文字列を ab とする．

3.2　文字列照合問題

テキスト t を長さ n の文字列，パターン p を長さ m の文字列とする．t，p の i 文字目は t_i，p_i とする．$1 \leq i \leq n-m+1$ を満たす整数 i の内，$t_i = p_1$，$t_{i+1} = p_2$，…，$t_{i+m-1} = p_m$ となるような i があるとき，パターン p はテキスト t の位置 i に存在するという．文字列照合問題の目的は $1 \leq i \leq n-m+1$ と $t_i = p_1$，$t_{i+1} = p_2$，…，$t_{i+m-1} = p_m$ の 2 つの条件を満たす整数 i を全て求めることである．

3.3　素朴な文字列照合アルゴリズム

t，p を文字列とし t の長さを n，p の長さを m とする．$1 \leq i \leq n-m+1$ を満たす全ての整数 i について $t_i = p_1$，$t_{i+1} = p_2$，…，$t_{i+m-1} = p_m$ となるかを調べることでパターン p がテキスト t に存在する位置を全て特定することができる．素朴な文字列照合アルゴリズムの具体的な手順は以下のようになる．

1. i，j の値を 1 にする．
2. $j > m$ の場合，p は t の位置 i に存在することが分かる．i の値を 1 増やし j の値を 1 にして照合を続行する．
3. $i > n-m+1$ の場合，照合を終了する．終了までに 1 度も照合が成功しなかった場合，照合失敗とする．
4. t の $i+j-1$ 文字目と p の j 文字目が一致するかを確かめる．
5. 一致しなかった場合，i の値を 1 増やし，j の値を 1 にして 2. に戻る．
6. 一致した場合，j の値を 1 増やし 2. に戻る．

例えば，$t = \texttt{nannazanannan}$，$p = \texttt{nannan}$，$n = 13$，$m = 6$ の場合，文字列照合は次のように行われる．まず，$i = 1$ のとき，$t_i = p_1$，$t_{i+1} = p_2$，…，$t_{i+m-1} = p_m$ となるかを調べる．すると，$t_6 = \texttt{z}$，$p_6 = \texttt{n}$ なので $t_6 \neq p_6$ である．よって p は t の位置 1 には存在しない．次に，$i = 2$ のときを調べる．$t_2 = \texttt{a}$，$p_1 = \texttt{n}$ なので $t_2 \neq p_1$ である．よって p は t の位置 2 には存在しない．以下同様の操作を繰り返す．すると，$i = 8$ のときに $t_i = p_1$，$t_{i+1} = p_2$，…，$t_{i+m-1} = p_m$ となる．したがって，p は t の位置 8 に存在することを特定できた．この素朴な文字列照合アルゴリズムの例を図で表すと図 1 のようになる．

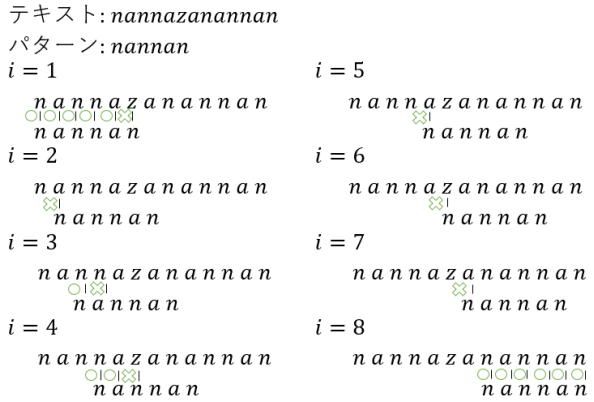

図 1　素朴な文字列照合アルゴリズムを用いた文字列照合

素朴な文字列照合アルゴリズムでは $1 \leq i \leq n-m+1$ を満たす全ての整数 i について最大で m 回の比較が行われる．つまり，m 回の比較が $n-m+1$ 回行われる．よって計算量は $\mathrm{O}((n-m+1)m)$ となる．

3.4　Rabin–Karp のアルゴリズム

Rabin–Karp のアルゴリズムでは p を整数に変換するハッシュ関数 h を用いて文字列照合を行う．p にハッシュ関数を適用することを $h(p)$ とする．b はアルファベットの種類の個数，q は整数とし

$$h(p) = (p_1 \times b^{m-1} + p_2 \times b^{m-2} + \cdots + p_m) \bmod q \quad (1)$$

とする．t から連続する m 文字を抜き出し，抜き出した文字列も同じように整数に変換する．i 文字目から連続する m 文字を抜き出した文字列を整数に変換するとき，

$$h(t_i t_{i+1} \ldots t_{i+m-1}) = (t_i \times b^{m-1} + t_{i+1} \times b^{m-2} + \cdots + t_{i+m-1}) \bmod q \quad (2)$$

とする．なぜ，q で割った余りで考えるのかというと q で割らない場合，$h(p)$ や $h(t_it_{i+1}\dots t_{i+m-1})$ の値が大きくなりすぎてしまう可能性があるからである．値が大きくなりすぎてしまった場合，1 度四則演算を行うために必要な計算回数が多くなるため計算効率が落ちてしまう．さらに，コンピュータを用いてハッシュ値を計算する場合，ハッシュ値がコンピュータの 1 ワードを超える値になるとハッシュ値同士を比較するために複数ステップが必要になってしまう．よって，q で割った余りで考える．q で割った余りで考えるので 2 つの文字列 p, $t_it_{i+1}\dots t_{i+m-1}$ が一致していないにもかかわらず偶然ハッシュ値が同じ値になってしまう可能性がある．しかし，2 つの文字列 p, $t_it_{i+1}\dots t_{i+m-1}$ が一致しているにもかかわらず偶然ハッシュ値が違う値になってしまうという可能性はない．したがって，2 つのハッシュ値が等しくなった場合のみ 2 つの文字列が一致しているかを素朴な文字列照合アルゴリズムを用いて確かめることで p が t のどこに存在しているかを調べることができる．

Rabin–Karp のアルゴリズムの具体的な手順は以下のようになる．

1. b^{m-1} の値を求める．
2. $h(p)$ の値を求める．
3. $h(t_1t_2\dots t_m)$ の値を求める．
4. $i=1$ とする．
5. $h(p)$ と $h(t_it_{i+1}\dots t_{i+m-1})$ の値を比較する．
6. $h(p)=h(t_it_{i+1}\dots t_{i+m-1})$ の場合，素朴な文字列照合アルゴリズムを用いて文字列を比較する．
7. $i<n-m+1$ の場合，$h(t_{i+1}t_{i+2}\dots t_{i+m})$ の値を求める．
8. $i\geq n-m+1$ の場合，照合を終了する
9. $i<n-m+1$ の場合，i の値を 1 増やして 5 に戻る．

例えばテキスト $t=$ `nanzan`，パターン $p=$ `an`，$n=6$, $m=2$ の場合，文字列照合は以下のように行われる．
$\Sigma=\{\mathtt{a},\mathtt{b},\mathtt{c},\dots,\mathtt{z}\}$，$b=26$，$q=13$ とする．文字列から整数への変換は a から z のアルファベットをそれぞれ 1 から 26 までの整数に対応させて行う．最初に，b^{m-1} の値を求める．$m=2$ であるため $b^{m-1}=26$ である．次に，$h(p)$ の値を求める．$p=$ `an` であるため $h(p)=14+26\times 1 \bmod 13=1$ である．その後，$h(t_1t_2)$ の値を求めると t の 1 文字目から連続する 2 文字を抜き出した文字列は `na` であるため $h(t_1t_2)=1+26\times 14 \bmod 13=1$ である．$h(p)$ と $h(t_1t_2)$ の値を比較すると $h(p)=1$，$h(t_1t_2)=1$ であるため，p と `na` は一致している可能性がある．よって，2 つの文字列を素朴な文字列照合アルゴリズムを用いて比較する．すると，`na` と p は一致していないことが分かる．次に，$h(t_2t_3)$ の値を求める．t の 2 文字目から連続する 2 文字を抜き出した文字列は `an` であるため $h(t_2t_3)=26\times(1-26\times 14)+14 \bmod 13=1$ である．$h(p)$ と $h(t_2t_3)$ の値を比較すると $h(p)=1$，$h(t_2t_3)=1$ であるため，p と `an` は一致している可能性がある．よって，2 つの文字列を素朴な文字列照合アルゴリズムを用いて比較する．すると，`an` と p は一致していることが分かる．以下同様の操作を繰り返す．すると，p は t の位置 2 と位置 5 に存在することを特定できる．この Rabin–Karp のアルゴリズムの例を図で表すと図 2 のようになる．

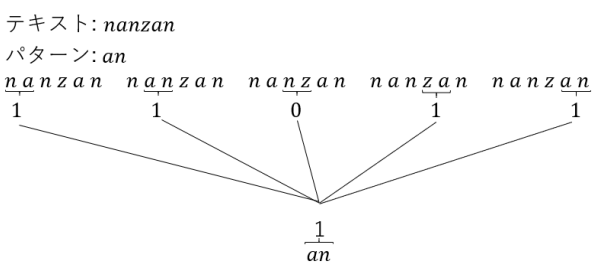

図 2 Rabin–Karp のアルゴリズムを用いた文字列照合

Rabin–Karp のアルゴリズムを用いて文字列照合を行うために必要な前処理は b^{m-1}，$h(p)$，$h(t_1t_2\dots t_m)$ を求めることである．b^{m-1} を求めるために $\mathrm{O}(\log m)$，$h(p)$ と $h(t_1t_2\dots t_m)$ を求めるために $\mathrm{O}(m)$ かかる．よって，前処理に必要な時間計算量は $\mathrm{O}(m)$ である．

$h(t_{i+1}t_{i+2}\dots t_{i+m})(1\leq i\leq n-m)$ は

$$\begin{aligned}&h(t_{i+1}t_{i+2}\dots t_{i+m})=(t_{i+1}\times b^{m-1}+t_{i+2}\times b^{m-2}+\\&\cdots+t_{i+m}) \bmod q\\&=(b(h(t_it_{i+1}\dots t_{i+m-1})-b^{m-1}\times t_i)+t_{i+m}) \bmod q\end{aligned}$$

より $h(t_it_{i+1}\dots t_{i+m-1})$ の値を利用することによって $\mathrm{O}(1)$ で求めることができる．したがって前処理で求めるのは $h(t_1t_2\dots t_m)$ のみで良い．

b と q が互いに素である場合，i と $h(t_it_{i+1}\dots t_{i+m-1})$ から $h(t_{i+1}t_{i+2}\dots t_{i+m})$ への関数は単射であるため Rabin–Karp のアルゴリズムを可逆化する際も用いることができる．i と $h(t_it_{i+1}\dots t_{i+m-1})$ から $h(t_{i+1}t_{i+2}\dots t_{i+m})$ への関数 $h(t_{i+1}t_{i+2}\dots t_{i+m})=(b(h(t_it_{i+1}\dots t_{i+m-1})-b^{m-1}\times t_i)+t_{i+m}) \bmod q$ が単射であることを証明する．

$h(t_it_{i+1}\dots t_{i+m-1})=h$ とする．
i と h から $h(t_{i+1}t_{i+2}\dots t_{i+m})$ への関数

$$f(h,i)=(b(h-b^{m-1}\times t_i)+t_{i+m}) \bmod q \qquad (3)$$

を考える．$f(h,i)=f(h',i)\Rightarrow h=h'$ を証明することで $f(h,i)$ が単射であることを証明する．b と q が互いに素でない場合，h と h' の値が一致していなくても $f(h,i)=f(h',i)$ となってしまう可能性がある．そのため，b と q は互いに素とする．まず，f の定義より，

$$\begin{aligned}&f(h,i)=f(h',i)\\\Leftrightarrow\ &(b(h-b^{m-1}\times t_i)+t_{i+m}) \bmod q\\&=(b(h'-b^{m-1}\times t_i)+t_{i+m}) \bmod q\end{aligned}$$

である．次に，両辺を展開すると

$$\begin{aligned}\Leftrightarrow\ & b \times h - b^m \times t_i + t_{i+m} \bmod q \\ & = b \times h' - b^m \times t_i + t_{i+m} \bmod q\end{aligned}$$

となる．次に，両辺から $-b^m \times t_i + t_{i+m}$ を引くと

$$\Leftrightarrow\ b \times h \bmod q = b \times h' \bmod q$$

となる．次に，両辺を b で割ると

$$\Leftrightarrow\ h \bmod q = h' \bmod q$$

となる．両辺を b で割るとき，b と q が互いに素でないと

$$b \times h \bmod q = b \times h' \bmod q \Leftrightarrow h \bmod q = h' \bmod q$$

が成り立たない場合がある．例えば $b = 10$，$q = 35$，$h = 8$，$h' = 25$ の場合，

$$10 \times 8 \bmod 35 = 10 \times 25 \bmod 35$$

である．しかし，

$$8 \bmod 35 \neq 25 \bmod 35$$

である．したがって，b, q は互いに素でなければならない．h，h' は q で割った余りの値であるため $0 \leq h, h' \leq q-1$ である．よって，

$$h \bmod q = h' \bmod q \Leftrightarrow h = h'$$

である．したがって，$f(h,i) = f(h',i) \Rightarrow h = h'$ である．以上のことから，$f(h,i)$ は単射である．

Rabin–Karp のアルゴリズムの時間計算量が最悪となる場合は $1 \leq i \leq n-m+1$ を満たす任意の整数 i で $h(p)$ の値と $h(t_i t_{i+1} \ldots t_{i+m-1})$ の値が等しくなるときであり，その時の計算量は素朴な文字列照合アルゴリズムと同じ $\mathrm{O}((n-m+1)m)$ となる．しかし，q の値を大きい素数にすることによって $\mathrm{O}(n+m)$ で動くことが期待できる．

3.5　可逆化する際の問題点と解決方法

素朴な文字列照合アルゴリズムを可逆化する際の問題点は素朴な文字列照合アルゴリズムの手順 5 の文字列照合が失敗したとき，j の値を 1 に初期化する方法である．j を 1 に初期化する方法は 2 つ考えられる．1 つ目は照合が失敗した場合は $j = 1$ になるまで j の値を 1 ずつ減らす方法である．2 つ目は照合が失敗した場合も照合を $t_{i+m-1} = p_m$ まで続けることである．最後まで照合を続けた場合，j の値は $m+1$ で確定する．よって，j と $m+1$ の排他的論理和を計算し，その後 j の値に 1 を足すことで j の値を 1 にすることができる．パターンの最初の方で文字列照合が失敗した場合は 1 つ目の方法の方が少ない時間計算量で j を初期化することができ，パターンの最後の方で文字列照合が失敗した場合は 2 つ目の方法の方が時間計算量が少なくなる．各文字の出現する確率が等しいとすると文字列の最初の方で照合が失敗する確率が高い．よって，1 つ目の方法の方が時間計算量が少なくなる場合が多いと考える．

Rabin–Karp のアルゴリズムを可逆化する際の問題点は $h(p)$ の値と $h(t_i t_{i+1} \ldots t_{i+m-1})$ $(1 \leq i \leq n-m+1)$ の値が等しくなり，2 つの文字列が一致しているかを確かめた結果不一致であることが判明したときである．なぜなら，素朴な文字列照合アルゴリズムと同様に j をどのように初期化するかという問題が発生するからである．j の初期化は素朴な文字列照合アルゴリズムと同様の方法で行うことができる．

4　結果

素朴な文字列照合アルゴリズムと Rabin–Karp のアルゴリズムについて非単射なステップを確認し，単射なステップにする方法を提案した．Rabin–Karp のアルゴリズムで利用する i と $h(t_i t_{i+1} \ldots t_{i+m-1})$ から $h(t_{i+1} t_{i+2} \ldots t_{i+m})$ への関数が単射であることを証明した．これら 2 つのことをすることにより，素朴な文字列照合アルゴリズムと Rabin–Karp のアルゴリズムを可逆化した．

5　今後の課題

可逆化した文字列照合アルゴリズムを可逆プログラミング言語を用いて実装する．実装したプログラムをもとに可逆化した各アルゴリズムの時間計算量，空間計算量，ゴミ出力量を解析する．非可逆な文字列照合アルゴリズムと可逆化した文字列照合アルゴリズムを比較したり可逆化した文字列照合アルゴリズム同士を比較したりすることにより各アルゴリズムの異なる点を調べる．時間計算量，空間計算量，ゴミ出力量のいずれかの点について最適化を行う方法を提案する．

参考文献

[1] Bennett, C.H.: Logical Reversibility of Computation,*IBM Journal of Research and Development*, Vol.17, No.6, pp.525–532 (1973).

[2] Cormen. T, Leiserson. C, Rivest. R, et al.(著), 浅野 哲夫, 岩野 和生, 梅尾 博司ほか (訳), アルゴリズムイントロダクション, 近代科学社, 第 3 版総合版 (2013).

[3] Landauer, R.: Irreversibility and Heat Generation in the Computing Process, *IBM Journal of Research and Development*, Vol.5, No.3, pp.183–191(1961).

[4] 家崎 雄太, 水野 竣太郎: 可逆線形探索, 南山大学 2017 年度卒業論文 (2018).

[5] 森田健一: 可逆計算, 近代科学社 (2012).