種々の文字列照合のための索引構造に関する研究

修士学位論文

種々の文字列照合のための索引構造に関する研究

東北大学大学院 情報科学研究科

システム情報科学専攻 篠原研究室

博士課程前期二年の課程

大友　雄平

2013年2月14日

目次

第1章

序論

1.1 背景

今日，多くの分野で文字列データが生成されている．そのようなデータの中には文字列の構造そのものに着目したいものもあれば，複数の文字列が組になっており集合的な扱いをしたいものもある．前者の例として，計算機プログラムは変数名を付け替えても同じ働きをするので，照合の際にはその差違を吸収した取り扱いができると便利である．また，多重奏の楽曲データや各種センサデータから得られた複数の時系列データは，後者の性質をもつ．

パターン照合は文字列処理における基本的な解析手法であり，その研究は昔から行われてきた．パターン照合を行うための手法は大きく分けて2種類の方法がある．1つ目は Σ を定数アルファベットとすると Σ^* の要素であるテキスト t とパターン p が与えられたときに，そのままテキスト中に出現するパターンの出現位置を求める手法である．2つ目はテキスト t は固定として，t に前処理を施して索引となるデータ構造を作成しておき，パターン p が与えられたときに高速にテキスト中に出現するパターンの出現位置を求める手法である．前者の代表例として，BM 法 [4] や KMP 法 [13] などがあり，後者の代表例として，接尾辞木 [18] や接尾辞配列 [15]，ポジションヒープ [7, 14] などがある．

パラメタ化パターン照合は，文字列の構造的なパターンの解析を行うための手法である．Σ を定数アルファベット，Π をパラメタアルファベットとし，$(\Pi \cup \Sigma)$ 上の文字列をパラメタ化文字列と呼ぶ．同じ長さの2つのパラメタ化文字列は，片方のパラメタ化文字列が全単射 $\Pi \to \Pi$ による変換でもう片方のパラメタ化文字列と一致するならば，この2

つのパラメタ化文字列はパラメタ化一致するという．例えば，$\Pi = \{x, y\}$, $\Sigma = \{a, b, c\}$, パラメタ化文字列 $\mathtt{s}_1 = axbyc$, $\mathtt{s}_2 = aybxc$ に対して x と y を入れ替えるような全単射で $\mathtt{s}_1$ を変換すると $\mathtt{s}_2$ と一致するため $\mathtt{s}_1$ と $\mathtt{s}_2$ はパラメタ化一致している．パラメタ化パターン照合問題は，パラメタ化テキスト T とパラメタ化パターン P を入力として受け取り，T 中の部分文字列で P とパラメタ化一致する位置をすべて出力する問題である．

Baker はパラメタ化パターン照合問題を定義し [1]，パラメタ化パターン照合問題を効率的に解くためのデータ構造であるパラメタ化接尾辞木を提案した [2]．パラメタ化パターン照合の応用例はソフトウェアの管理 [1, 3] や盗作検知 [9]，遺伝子の構造照合 [17] などがある．

順列パターン照合は多重奏楽曲データや各種センサデータなどのマルチトラックデータのパターン解析を行うための手法である．Σ 上の同じ長さの文字列の組をマルチトラック文字列と呼ぶ．2 つのマルチトラック文字列が，片方のマルチトラック文字列を構成する文字列の順番を順列 $\mathbf{r}$ で入れ替えて，もう片方のマルチトラック文字列と一致するならば 2 つのマルチトラック文字列は順列一致するという．例えば，$\Sigma = \{a, b\}$ 上のマルチトラック文字列 $\mathbb{X} = (aab, aba, bba)$，$\mathbb{Y} = (aba, aab, bba)$ に対して，$\mathbb{X}$ を順列 $\mathbf{r} = (2, 1, 3)$ で並べ替えると，$\mathbb{Y}$ と一致するため $\mathbb{X}$ と $\mathbb{Y}$ は順列一致している．順列パターン照合問題は，マルチトラックテキスト $\mathbb{T}$ とマルチトラックパターン $\mathbb{P}$ を入力として受け取り，$\mathbb{T}$ の中の部分文字列で $\mathbb{P}$ と順列一致する位置をすべて出力する問題である．桂らは順列パターン照合問題を定義し順列パターン照合問題を効率的に解くためのデータ構造であるマルチトラック接尾辞木を提案した [12, 19, 20]．

本論文では，Ehrenfeucht らによって近年提案された Σ 上の文字列のパターン照合問題を解くための新しいデータ構造であるポジションヒープ [7] に対して新たにハッシュ配列というデータ構造を追加して確率的にパターン照合を行う手法を提案する．またポジションヒープをパラメタ化パターン照合と順列パターン照合に適用するために新たに 2 つのデータ構造を提案する．パラメタ化パターン照合問題のためのデータ構造としてパラメタ化ポジションヒープを定義し，それを構築する線形時間のアルゴリズムを提案する．そしてパラメタ化ポジションヒープを用いたパターン照合アルゴリズムを提案する．一方，順列パターン照合問題のためのデータ構造としてマルチトラックポジションヒー

プと縮約マルチトラックポジションヒープを定義し，それらを構築する線形時間のアルゴリズムと，効率のよいパターン照合アルゴリズムを提案する．ポジションヒープは同様な文字列のパターン照合問題を解くためのデータ構造である接尾辞木と比べると省領域なデータ構造である．どちらも木構造であるが，文字列長を n とすると接尾辞木の頂点数が $2n-1$ 個以下であるのに対し，ポジションヒープの頂点数は $n+1$ 個以下となる．

1.2 本論文の構成

本論文は次のような構成となっている．第 1 章では，背景を述べた．第 2 章では，パターン照合問題を扱うための準備を行う．第 3 章では本論文における手法の拡張の基となったポジションヒープの説明を行う．第 4 章では，通常のポジションヒープに対して新たな確率的線形時間パターン照合アルゴリズムを提案する．第 5 章では，パラメタ化パターン照合問題を効率的に解くためのデータ構造としてパラメタ化ポジションヒープの定義を行い，入力サイズの線形時間構築アルゴリズムを提案する．また，パラメタ化ポジションヒープを用いたパターン照合アルゴリズムの提案を行う．第 6 章では，順列パターン照合問題を効率的に解くためのデータ構造としてマルチトラックポジションヒープの定義を行い，入力サイズの線形時間構築アルゴリズムを提案する．また，マルチトラックポジションヒープを用いたパターン照合アルゴリズムの提案を行う．第 7 章では，計算機実験によりそれぞれの手法について理論的な計算量と実際の処理時間との比較を行う．第 8 章にて結論と今後の課題をまとめる．

第2章

準備

文字の有限集合Σを**アルファベット**と呼び，Σ^*の要素をΣ上の**文字列**と呼ぶ．wの長さを$|w|$で表す．長さ0の文字列を**空文字列**と呼びεで表す．文字列$w=xyz$に対し，文字列x，y，zをそれぞれwの**接頭辞**，**部分文字列**，**接尾辞**と呼ぶ．文字列$x,y\in\Sigma^n$に対して$x[1]=y[1],x[2]=y[2],...,x[n]=y[n]$のとき，$x$と$y$は**一致している**という．$1\leq i\leq|w|$に対して，$w[i]$は$w$の$i$番目の文字を表し，$1\leq i\leq j\leq|w|$に対して，$w[i..j]$は位置$i$から位置$j$までの部分文字列を表す．また$i>j$のとき$w[i..j]=\varepsilon$とする．任意の2つの文字列$x$，$y$に対し，$x$が$y$より辞書順で小さいとき$x<y$で表す．$x<y$または$x=y$を満たすとき$x\leq y$と表す．また$\mathcal{N}$は非負整数の集合を表す．

$\Pi\cap\Sigma=\emptyset$となる**パラメタアルファベット**Πに対して，文字列$\mathfrak{t}\in(\Sigma\cup\Pi)^*$を$(\Sigma,\Pi)$上の**パラメタ化文字列**と呼ぶ．

同じ長さのΣ上の文字列の組$\mathbb{X}$を**マルチトラック文字列**，または単に**マルチトラック**と呼ぶ．$\mathbb{X}=(x_1,x_2,...,x_N)$に対し，各項を**トラック**と呼ぶ．$\mathbb{T}$のトラックの数を**トラック数**，トラックの長さを**トラック長**と呼び，それぞれ$h(\mathbb{T})$，$l(\mathbb{T})$で表す．マルチトラック$\mathbb{X}=(x_1,x_2,...,x_N)$，$\mathbb{Y}=(y_1,y_2,...,y_N)$に対して，マルチトラックの**連結**を$\mathbb{XY}=(x_1y_1,x_2y_2,...,x_Ny_N)$で定義する．マルチトラック$\mathbb{T}=\mathbb{XYZ}$に対して$\mathbb{X},\mathbb{Y},\mathbb{Z}$をそれぞれ$\mathbb{T}$の**接頭辞**，**部分文字列**，**接尾辞**と呼ぶ．$1\leq i\leq j\leq l(\mathbb{T})$に対して，位置$i$から位置$j$までの部分文字列を$\mathbb{T}[i..j]$で表す．$\mathbb{T}[i]=(t_1[i],t_2[i],...,t_N[i])$とする．$\mathbb{X}=(x_1,x_2,...,x_N)$，$\mathbb{Y}=(y_1,y_2,...,y_N)$，$1\leq i\leq N$に対して，$x_i=y_i$ならば，$\mathbb{X}=\mathbb{Y}$と定義する．トラック数$N$の**空マルチトラック**を$\mathbb{E}_N=(\varepsilon,\varepsilon,...,\varepsilon)$とする．$i>j$，$h(\mathbb{T})=N$に対して$\mathbb{T}[i..j]=\mathbb{E}_N$とする．

アルファベット Σ に対し，辺ラベルのついた根付き木 $T = (V, E)$, $E \subseteq V \times \Sigma \times V$ で次の条件を満たすものを Σ 上の**トライ木** [8] と呼ぶ：任意の $v \in V$ と $b \in \Sigma$ に対して，頂点 v を始点とするラベル b の辺 $(v, b, u) \in E$ は高々一つである．T の各頂点 $v \in V$ に対し，T の根頂点 $root$ から v に至るパス $(root, c_1, u_1)$,(u_1, c_2, u_2),...,(u_{k-1}, c_k, v) を考え，各辺のラベルを連結して得られる文字列 $c_1c_2 \ldots c_k$ を対応させる．以降，頂点 v とそれに対応した文字列を同一視して $v = c_1c_2 \ldots c_k$ と表記し T に $c_1c_2 \ldots c_k$ が含まれると呼ぶ．したがって $root = \varepsilon$ である．

第3章

ポジションヒープ

ここでは既存研究であるポジションヒープの定義と構築アルゴリズム，ポジションヒープを用いたパターン照合アルゴリズムの説明を行う．

3.1 シーケンスハッシュ木

シーケンスハッシュ木はCoffmanとEveらによって提案されたハッシュテーブルのためのトライ木である [5].

定義 1 (シーケンスハッシュ木). Σ 上の文字列の順序集合 $\mathbf{W} = (w_1, \ldots, w_n)$ に対する**シーケンスハッシュ木** $SHT(\mathbf{W}) = (V_n, E_n)$ は，以下のように再帰的に定義される Σ 上のトライ木である．

$$\begin{aligned} V_i &= V_{i-1} \cup \{p_i\} \\ E_i &= E_{i-1} \cup \{(q_i, c, p_i)\} \quad (1 \leq i \leq n) \end{aligned}$$

ただし $V_0 = \{\varepsilon\}$, $E_0 = \emptyset$ である．ここで，p_i は $p_i \notin V_{i-1}$ となる w_i の最短の接頭辞で，$q_i = w_i[1..|p_i|-1]$, $c = w_i[|p_i|]$ とする．この条件を満たす p_i が存在しない場合は $V_i = V_{i-1}$, $E_i = E_{i-1}$ とする．また，各頂点には文字列の識別番号 i を付与する．

文字列の順序集合 $\mathbf{W} = (aab, ab, bba, baa, aaba, ba)$ に対するシーケンスハッシュ木を図 3.1 に示す．Σ 上のシーケンスハッシュ木において，図 3.1 のように頂点に付与される識別番号が複数になる場合があるが， [14] にて，以下の補題が示されている．

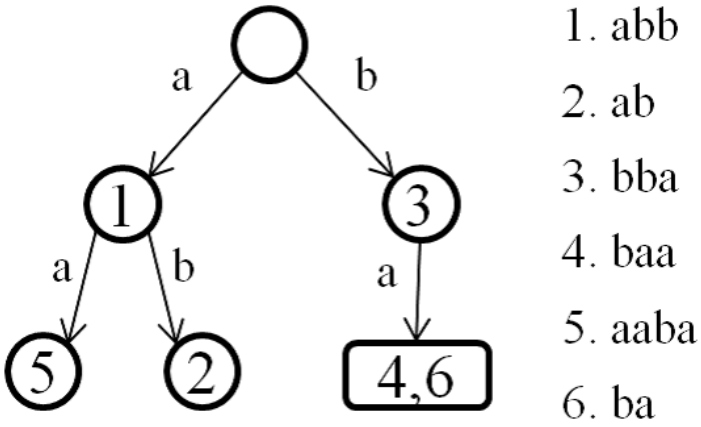


図 3.1: $(aab, ab, bba, baa, aaba, ba)$ に対するシーケンスハッシュ木.

補題 1. 互いに異なる文字列の順序集合 $\mathbf{W}$ に対して，$SHT(\mathbf{W})$ の各頂点は高々2つの $\mathbf{W}$ の要素に対応する.

補題 1 より，互いに異なる文字列の順序集合 $\mathbf{W}$ に対する $SHT(\mathbf{W})$ の各頂点は高々2つの要素が対応する．2 つの要素が対応する頂点を**二重頂点**と呼ぶ.

3.2 ポジションヒープ

定義 2 (ポジションヒープ). 文字列 t の空語を除くすべての接尾辞を長さの昇順または降順に並べて得られる順序集合に対するシーケンスハッシュ木を t に対する**ポジションヒープ**と呼ぶ,

長さ n の文字列 t に対するポジションヒープの線形時間の構築アルゴリズムが Ehrenfeucht ら [7] や Kucherov [14] によって提案されている．Nakashima ら [16] は，入力文字列の集合がトライ木として与えられたときの線形時間構築アルゴリズムを与えている．以後，本論文では接尾辞の長さの降順によるポジションヒープのみを考える.

定義 3 (接尾辞の長さ降順によるポジションヒープ). 文字列 t に対するポジションヒープ $PH(t)$ は以下のように再帰的に定義される Σ 上のトライ木である.

$$
\begin{aligned}
V_i &= V_{i-1} \cup \{p_i\} \\
E_i &= E_{i-1} \cup \{(q_i, c, p_i)\} \quad (1 \leq i \leq n)
\end{aligned}
$$

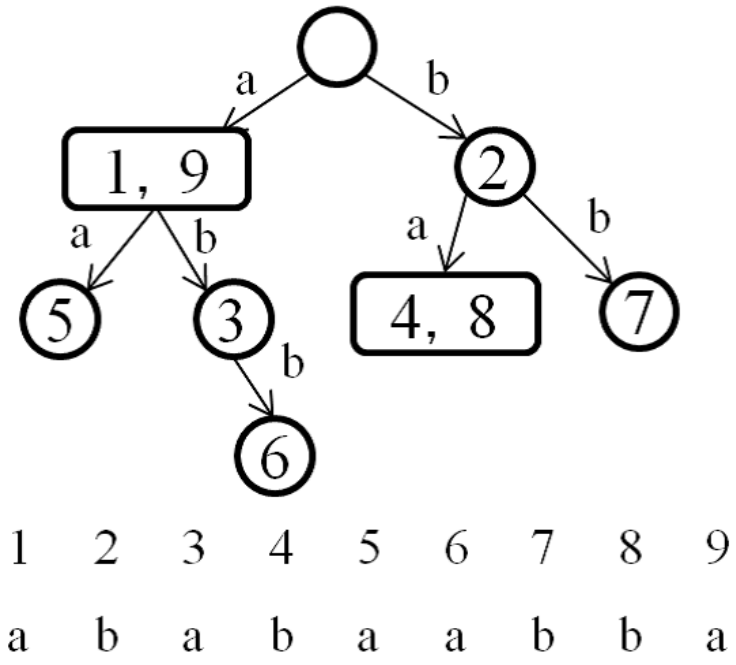


図　3.2:　文字列 $ababaabba$ に対するポジションヒープ．これは接尾辞の順序集合 $(ababaabba, babaabba, abaabba, \ldots, ba, a)$ に対するシーケンスハッシュ木である．

ただし $V_0 = \{\varepsilon\}$, $E_0 = \emptyset$ である．ここで，p_i は $p_i \notin V_{i-1}$ となる $t[i..n]$ の最短の接頭辞で，$q_i = t[i..i + |p_i| - 1]$, $c = t[i + |p_i|]$ とする．この条件を満たす p_i が存在しない場合は $V_i = V_{i-1}$, $E_i = E_{i-1}$ とする．また，各頂点には文字列の開始位置 i を付与する．図 *3.2* にポジションヒープの例を示す．

長さ n の文字列に対するポジションヒープ $PH(t)$ に対して，$1 \leq i \leq n$ 番目に $PH(t)$ に追加された部分文字列 $spref_i$ を以下のように定義する．

$$spref_i = \begin{cases} p_i & (p_i\text{が存在する場合}) \\ t[i..n] & (p_i\text{が存在しない場合}) \end{cases}$$

また，ポジションヒープを用いたパターン照合問題を解くためのアルゴリズムが Ehrenfeucht ら [7] によって提案されている．パターン長を m とすると $O(m^2 + occ)$ 時間で解くアルゴリズムが提案されており，さらに，極大到達ポインタと呼ばれる $O(n)$ 領域のデータ構造を付加することで照合をより高速に $O(m + occ)$ で行うアルゴリズムも提案されている．

Algorithm 1: ポジションヒープ構築アルゴリズム

Input: 長さ n の文字列 t
Output: ポジションヒープ $PH(t)$

create $root$ and $\perp$ nodes;
$sp(root) = \perp$;
for each character c, create edge $(\perp, c, root)$;
$currentNode = root$;
$s = 1$;
for $i = 1$ **to** n **do**
  $lastCreateNode =$ undefined;
  while not exist edge $(currentNode, t[i], nextNode)$ **do**
    create node $t[s..i]$;
    create edge $(currentNode, t[i], t[s..i])$;
    if $lastCreateNode \neq$ undefined **then**
      $sp(lastCreateNode) = t[s..i]$;
    $lastCreateNode = t[s..i]$;
    $currentNode = sp(currentNode)$;
    $s = s + 1$;
  $currentNode = nextNode$;
  if $lastCreateNode \neq$ undefined **then**
    $sp(lastCreateNode) = currentNode$;

3.3 線形時間構築アルゴリズム

Σ 上の文字列 t に対するポジションヒープは t の接尾辞の順序集合に対するシーケンスハッシュ木なので，補題 1 より $PH(t)$ の各頂点は最大 2 つの t に対する位置 p，q が付与される．$p < q$ とすると，p を**主位置**，q を**副位置**と呼ぶ．$PH(t)$ に対して， [14] にて次の補題が示されている．

補題 2. 文字列 $t \in \Sigma^n$ に対するポジションヒープ $PH(t)$ において，$q < n$ が副位置ならば $q + 1$ も副位置となる．

補題 2 より，文字列 $t \in \Sigma^n$ の各位置に対して $1, 2, ..., s - 1$ が主位置で $s, s + 1, ..., n$ が副位置となる s が存在する．s を**作業位置**と呼ぶ．

作業位置を利用して，$1 \leq i < n$ に対する $PH(t[1..i])$ から $PH(t[1..i + 1])$ を構築するアルゴリズムが [14] で提案されている．文字列 $t \in \Sigma^n$，$1 \leq i < n$ に対して $PH(t[1..i])$ が構

築できているとする．s を $t[1..i]$ の作業位置とすると，主位置 $1, 2, ..., s-1$ は $PH(t[1..i+1])$ においても主位置となるため，構築においては副位置 $s, s+1, ..., i$ のみを考慮すればよい．$s \leq j \leq i$ に対する $t[1..j]$ の接尾辞の頂点は，$PH(t[1..i])$ にすでに存在しているため，$PH(t[1..i])$ から $PH(t[1..i+1])$ を構築する際，$PH(t[1..i+1])$ に $s \leq k \leq i+1$ に対する頂点 $t[k..i+1]$ が新たに作成される場合と，作成されない場合の 2 通りが考えられる．ここで，補題 2 より $t[1..i+1]$ の作業位置を s' とすると，$s \leq k < s'$ に対する頂点 $t[k..i+1]$ が作成され，$s' \leq l \leq i+1$ に対する頂点 $t[l..i+1]$ は既に存在しており $PH(t[1..i+1])$ において二重頂点となる．構築において新たに頂点を作成する操作を効率的に行うために**接尾辞ポインタ**が [14] によって定義されている．

定義 4 (接尾辞ポインタ)．$PH(t)$ の頂点 $t[i..j]$ に対して，**接尾辞ポインタ**は $sp(t[i..j]) = t[i+1..j]$ で定義される．ただし，$sp(root) = \perp$ である．

$\perp$ は補助頂点であり，$\perp$ から根 $root$ には任意のラベル $a \in \Sigma$ で辿ることのできる辺が存在する．

$t[1..i]$ に対する作業位置 s から始まる接尾辞の頂点から接尾辞ポインタを辿ることで，$s+1, s+2, ..., i$ から始まる接尾辞の頂点を得ることができる．よって，$t[1..i]$ に対する作業位置 s のみを記憶し，そのほかの副位置を考慮しないことで $PH(t[1..i+1])$ の構築を効率的に行う．また，入力文字列 t の末尾を区切り文字 $\$ \notin \Sigma$ にすることで，最終的に二重頂点を持たない $PH(t)$ を得ることができる．

アルゴリズムの全体像を Algorithm 1 に示す．$PH(t[1..i])$ から $PH(t[1..i+1])$ の構築は，$t[1..i]$ に対する作業位置を s，$t[1..i+1]$ の作業位置を s' とすると，$s' > s$ の場合に s から始まる $t[1..i]$ の接尾辞の頂点 $t[s..i]$ からはじまる．$t[s..i]$ から伸びる辺にはラベルが $t[i+1]$ である辺は存在しないため，頂点 $t[s..i+1]$ を作成し，ラベル $t[i+1]$ の辺でつなぐ．頂点 $t[s..i+1]$ が作成されたので接尾辞ポインタを $t[s..i]$ から辿り，$t[s+1..i]$ に移動することで，頂点の作成を繰り返す．接尾辞ポインタを辿り頂点 $t[s'..i]$ に到達すると，ラベル $t[i+1]$ の辺が存在するので，構築を終了する．また，構築の際に $t[s..i+1]$,$t[s+1..i+1]$,...,$t[s'..i+1]$ の頂点を得ることができるため，$s \leq k < s'$ に対して $sp(t[k..i+1]) = t[k+1..i+1]$ とすることで，容易に $PH(t[1..i+1])$ において新たに生じる接尾辞ポインタを作成することができる．

Algorithm 2: 素朴なパターン照合アルゴリズム

Input: テキスト t，ポジションヒープ $PH(t)$，パターン p
Output: パターンの出現位置に対応した頂点リスト ans

```
1  m = |p|;
2  u = root of PH(t);
3  ok = true;
4  ans = empty list;
5  for i = 1 to m do
6      if not exist edge (u, p[i], nextNode) then
7          ok = false;
8          break;
9      u = nextNode;
10     start = position of u;
11     if t[start..start + m] = p then
12         add u to ans;
13
14 if ok=true then
15     add all descendants of v to ans;
16 return ans;
```

Algorithm 1 における while ループでは各位置 $1 \leq i \leq n$ に対する接尾辞の接頭辞を表す頂点を作成しているため，while ループはアルゴリズム全体で n 回しか回らない．for ループに関しても文字列の各文字を読み込んでいるためアルゴリズム全体で n 回しか回らない． [14] にて次の定理が示されている．

定理 1. *Algorithm 1* は長さ n の文字列 t を受け取り，$O(n)$ 時間で $PH(t)$ を構築する．

3.4 素朴なパターン照合アルゴリズム

パターン $p \in \Sigma^m$ が与えられたとき $t \in \Sigma^n$ に対するポジションヒープ $PH(t)$ を用いてパターン照合問題を解く．パターン照合問題を定義する．

定義 5 (パターン照合問題). Σ 上のテキスト t と p が与えられたとき，t の部分文字列で p と一致するすべての位置を出力する．

ポジションヒープは長さ n のテキストの各位置 $1 \leq i \leq n$ から始まる部分文字列の接尾辞に対するトライ木になっているため，パターン p でポジションヒープ $PH(t)$ を根から辿

ることで，t の中で p の接頭辞と $1, 2, ..., k \leq m$ 文字一致する部分文字列の開始位置が得られる．$k = m$ のときに限り辿りついた頂点に付与されている位置とその子孫にあたる位置からはじまる m 文字の部分文字列はすべて p と一致する．残りの $1, 2, ..., k-1 < m$ 文字一致した各位置について $O(m)$ 時間で通常の文字列比較を行いパターンと一致しているかの検証を行う．よって全体で $O(m^2 + occ)$ 時間でパターン照合問題を解くことができる．ただし，occ はパターンの出現数を表す．$k < m$ の場合には $k = m$ のときの検証のみを行えばよいので同様に $O(m^2 + occ)$ 時間でパターン照合問題を解くことができる．アルゴリズムを Algorithm 2 に示す． [7] にて次の補題が示されている．

補題 3. *Algorithm 2* は $O(m^2 + occ)$ 時間でパターン照合問題を解く．

3.5 拡張ポジションヒープを用いたパターン照合アルゴリズム

拡張ポジションヒープは，通常のポジションヒープに対して極大到達ポインタと呼ばれるデータ構造と，ポジションヒープの 2 頂点が与えられた際に子孫判定を行うためのデータ構造を付与したポジションヒープである。拡張ポジションヒープを用いることでパターン照合問題をパターン長の線形時間で解くことができる．

[7] にて提案されている長さ n の文字列 t に対するポジションヒープ $PH(t)$ に対する極大到達ポインタ mrp を次のように定義する．

定義 6 (極大到達ポインタ)**.** 各位置 $1 \leq i \leq n$ に対応する頂点 $spref_i$ に対して，**極大到達ポインタ** mrp は以下の式で表される．

$$mrp(spref_i) = arg\ max_{s \in S_i} |s|$$

$$S_i = \{spref_j \mid 1 \leq j \leq n \text{ かつ } t[i..i + |spref_j|] = spref_j\}$$

極大到達ポインタに加えてトライ木の 2 頂点 u, v が与えられたときに，u が v の子孫であるかの判定を定数時間で行うためのデータ構造は，トライ木を深さ優先探索し，行きがけ順と帰りがけ順の順番を $O(n)$ 時間，領域で記憶し u の行きがけ順 $\leq v$ の行きがけ順かつ u の帰りがけ順 $\geq v$ の帰りがけ順ならば v が u の子孫であることを利用することで実現できる [6]．極大到達ポインタと子孫判定を用いることで，ポジションヒープの頂点

v に対する極大到達ポインタ $mrp(v)$ が頂点 u またはその子孫であれば，v の位置から始まる部分文字列の接尾辞が u になることがわかる．子孫判定を行うデータ構造と極大到達ポインタを付与したポジションヒープを**拡張ポジションヒープ**と呼ぶ．拡張ポジションヒープを用いたパターン照合アルゴリズムを Algorithm 3 に示す．

長さ m のパターン p が与えられたときに，最大 k 文字拡張ポジションヒープを辿れたとすると，辿る際に通った頂点を記憶しておき，再び根から k 文字辿る過程で通った頂点に対して，極大到達ポインタを用いて 15 行目の if 文で子孫判定を行うことで，各頂点が持つ位置から始まる部分文字列が，k 文字以上パターンと一致しているかを判定している．判定に通過した候補の位置に k を加えることで，新たに読み込むパターンの接尾辞の開始位置に対応したテキスト中の位置が得られる． 21 行目の while ループ内で残りのパターン接尾辞が一致しているかの判定を行う．辿れなかったパターンの接尾辞で根から新たに k' 文字辿り，新たに読み込むパターンの接尾辞の開始位置に対応したテキスト中の位置に対して， 30 行目の if 文で子孫判定を行うことで，新たに辿れた部分文字列が，k' 文字以上パターンと一致しているかを判定している．この判定をパターンをすべて読み込むまで繰り返す．while ループの中で辿れた深さの数しかパターンの候補がないので， 30 行目の子孫判定の回数は全体で $2m$ 回以下となる．$k = m$ の場合は通常のポジションヒープを用いたパターン照合アルゴリズムと同様に辿った先の頂点とその子孫にあたる頂点に対応する位置を解に含めることで $O(m + occ)$ 時間でパターン照合問題を解くことができる． [7] にて以下の補題が示されている．

補題 4. *Algorithm 3* は，テキスト t に対するポジションヒープ $PH(t)$ を拡張した拡張ポジションヒープ，パターン p が与えられたとき，パターン照合問題を $O(m + occ)$ 時間で解く．

Algorithm 3: 拡張ポジションヒープを用いたパターン照合アルゴリズム

Input: テキスト t，子孫判定が可能なポジションヒープ $PH(t)$，$PH(t)$ に対する極大到達ポインタ mrp，パターン p

Output: パターンの出現位置に対応した頂点リスト ans

```
1  m = |p|;
2  u = root;
3  for i = 1 to m do
4      if not exist edge (u, p[i], nextNode) then
5          break;
6      u = nextNode;
7  ans = empty list;
8  v = root;
9  ok = true;
10 for i = 1 to m do
11     if not exist edge (v, p[i], nextNode) then
12         ok = false;
13         break;
14     v = nextNode;
15     if mrp(v) is descendant of u then
16         add v to ans;
17
18 if ok=true then
19     add positions of all descendants of v to ans;
20 else
21     while i ≠ m do
22         q = root;
23         for i to m do
24             if not exist edge (q, p[i], nextNode) then
25                 break;
26             q = nextNode;
27         if q=root then
28             return empty list;
29         foreach w ∈ ans do
30             if mrp(w) is not descendant of q then
31                 erase w from ans;
32
33
34
35 return ans;
```

第4章

ポジションヒープを用いた確率的パターン照合アルゴリズム

[7] では文字列 $t \in \Sigma^n$ に対する子孫判定が可能なポジションヒープ $PH(t)$ と $PH(t)$ に対する極大到達ポインタ，パターン $p \in \Sigma^m$ が与えられたとき，パターン照合問題を $O(m+occ)$ 時間で解くことのできるアルゴリズムが提案されている．ここでは，極大到達ポインタの代わりに $O(n)$ 領域のハッシュ配列を用いることで確率的に $O(m+occ)$ 時間でパターン照合問題を解くアルゴリズムを提案する．このアルゴリズムの利点は補助構造を作成する際に構築後のトライ木を辿らないため補助構造の構築が高速に行える点と，実装が極大到達ポインタを用いた手法に比べて容易になる点である．

4.1 ハッシュ配列

Karp と Rabin は文字列をハッシュ化して，ハッシュ値の比較で文字列の比較を行う手法を考案した [11]．この手法を拡張してパターンの候補に対する検証を確率的に $O(1)$ 時間で行う．文字列 $t \in \Sigma^n$ に対するハッシュ関数 $H_{\alpha,\beta}$ を以下のように定義する．

$$H_{\alpha,\beta}(t) = \sum_{i=1}^{n} \phi(t[i]) \cdot \alpha^{i-1} \bmod \beta$$

ここで，α, β は互いに素な正整数であり，ϕ は $a, b \in \Sigma$ に対して $a \neq b$ ならば $\phi(a) \neq \phi(b)$ かつ $a = b$ ならば $\phi(a) = \phi(b)$ となる Σ から $\mathcal{N}$ への写像である．ハッシュ値は β で割った余りをとっているため，異なる文字列でも同じハッシュ値になる偽陽性の可能性があり，α と β の選び方で異なる文字列が同じハッシュ値になる確率が変わる．ここでは，Gonnet

らが [10] で採用した α と β の値を用いて確率の評価を行う．また Gonnet らは効率的に実装を行うためのパラメータも提案しており，実験では効率的に実装を行うためのパラメータを採用する．Gonnet らは確率評価のために β を大きな素数，α を β を法としたときのアルファベットサイズ以上の原始根とすることで次の補題を示している．

補題 5. α を β を法としたときのアルファベットサイズ以上の原始根，β を大きな素数とすると，同じ長さの異なる文字列 t_1, t_2 のハッシュ値 $H_{\alpha,\beta}(t_1)$ と $H_{\alpha,\beta}(t_2)$ が同じ値になる確率は $\frac{1}{\beta}$ 以下となる．

文字列 $t \in \Sigma^n$ の任意の位置 $1 \leq i \leq n-m$ からはじまる長さ m の部分文字列に対するハッシュ値を $O(1)$ 時間で求めるために t に対する**ハッシュ配列** $HA_{\alpha,\beta}$ を構築する．

定義 7 (ハッシュ配列)**.**

$$\begin{aligned} HA_{\alpha,\beta}[i] &= HA_{\alpha,\beta}[i-1] + \phi(t[i]) \cdot \alpha^{i-1} \bmod \beta \quad (1 \leq i \leq n) \\ HA_{\alpha,\beta}[0] &= 0 \end{aligned}$$

$HA_{\alpha,\beta}$ を用いることで，位置 i からはじまる長さ m の部分文字列のハッシュ値 $H_{\alpha,\beta}(t[i..i+m-1])$ を $H_{\alpha,\beta}(t[i..i+m-1]) = (HA_{\alpha,\beta}[i+m-1] - HA_{\alpha,\beta}[i-1]) \cdot \alpha^{-(i-1)} \bmod \beta$ で求めることができる．α^{-i} は α の逆元の i 乗である．α, β は互いに素なので α の逆元は常に存在する．α の $-i$ 乗 $(0 \leq i \leq n)$ を β で割った余りを $O(1)$ 時間で参照するために**逆元配列** $INV_{\alpha,\beta}$ を構築する．

定義 8 (逆元配列)**.**

$$\begin{aligned} INV_{\alpha,\beta}[i] &= INV_{\alpha,\beta}[i-1] \cdot \alpha^{-1} \bmod \beta \quad (1 \leq i \leq n) \\ INV_{\alpha,\beta}[0] &= 1 \end{aligned}$$

4.2 確率的パターン照合アルゴリズム

長さ n のテキスト t とポジションヒープ $PH(t)$，t に対するハッシュ配列 $HA_{\alpha,\beta}$，逆元配列 $INV_{\alpha,\beta}$，長さ m のパターン p が与えられたとき，t の中に現れる p と一致する部分

Algorithm 4: ハッシュ配列を用いたパターン照合アルゴリズム

Input: テキスト t，ポジションヒープ $PH(t)$，ハッシュ配列 $HA_{\alpha,\beta}$，パターン p，逆元配列 $INV_{\alpha,\beta}$

Output: パターンの出現位置に対応する頂点リスト ans

```
1  ans = empty list;
2  m = |p|;
3  pHash = Σ_{i=1}^{m} φ(p[i]) · α^{i-1} mod β;
4  u = root;
5  ok = true;
6  for i = 1 to m do
7      if not exist edge (u, p[i], nextNode) then
8          ok = false;
9          break;
10     u = nextNode;
11     start = position of u;
12     if ((HA_{α,β}[start + m − 1] − HA_{α,β}[start − 1]) · INV[start − 1] mod β) = pHash
       then
13         add u to ans;
14
15 if ok=true then
16     add all descendants of u to ans;
17 return ans;
```

文字列の位置をすべて求めることを考える．はじめに p に対するハッシュ値を $O(m)$ 時間で求める．次に，$PH(t)$ を p で辿ることで最大 $m-1$ 個のパターンと照合している可能性のある位置が得られる．m 文字辿れない場合は，パターンと接頭辞が一致している部分文字列は辿る際に得られた頂点に限られるため，各候補についてのみ，ハッシュ値の比較を行えばよい．照合位置の候補からハッシュ配列を用いて m 文字の部分文字列に対してハッシュ値を $O(1)$ 時間で求め，p のハッシュ値との比較を行い一致していれば位置の候補を解に含める．加えて，p が $PH(t)$ に含まれている場合に限り到達した頂点とその子孫に対応する位置がパターンの照合位置となる．よって，合計で $O(m+occ)$ 時間で確率的にパターン照合を行うことができる．ここで，occ はパターンの出現数を表す．アルゴリズムを Algorithm 4 に示す．

ハッシュ値による比較は最大 $m-1$ 回行われるため，このアルゴリズムが正しく動作する確率は $(1-\frac{1}{\beta})^{m-1}$ となる．

定理 2. *Algorithm 4* は長さ n のテキスト t とポジションヒープ $PH(t)$, t に対するハッシュ配列 $HA_{\alpha,\beta}$, 逆元配列 $INV_{\alpha,\beta}$, 長さ m のパターン p が与えられたとき, 確率 $\left(1-\frac{1}{\beta}\right)^{m-1}$ でパターン照合問題を $O(m+occ)$ 時間で解く.

第5章

パラメタ化パターン照合問題に対するポジションヒープ

ここでは，パラメタ化パターン照合問題に対するポジションヒープであるパラメタ化ポジションヒープの定義と構築アルゴリズムの提案，パラメタ化ポジションヒープを用いたパターン照合アルゴリズムの提案を行う．

5.1 パラメタ化一致

定義 9 (Σ, Π) 上のパラメタ化文字列 $\mathtt{s}_1$ と $\mathtt{s}_2$ に対して，$\mathtt{s}_1$ のパラメタを変換したときに $\mathtt{s}_2$ と一致するような Π から Π への全単射が存在するならば，$\mathtt{s}_1$ と $\mathtt{s}_2$ は**パラメタ化一致する**という．

パラメタ化一致の判定を効率的に行うために prev 符号 [2] を用いる．

定義 10 (prev 符号)．(Σ, Π) 上のパラメタ化文字列 $\mathtt{t}$ に対して，次式で定義される $\mathcal{N} \cup \Sigma$ 上の文字列 $prev(\mathtt{t}) = w$ を **prev 符号**と呼ぶ．

$$w[i] = \begin{cases} \mathtt{t}[i] & (\mathtt{t}[i] \in \Sigma \text{のとき}) \\ 0 & \left(\begin{array}{l} \mathtt{t}[i] \in \Pi \text{で，すべての } 1 \leq j < i \\ \text{に対して } \mathtt{t}[i] \neq \mathtt{t}[j] \text{ のとき} \end{array} \right) \\ i - \max\{j \mid \mathtt{t}[j] = \mathtt{t}[i] \text{ かつ } 1 \leq j < i\} & (\text{その他}) \end{cases}$$

例えば，$(\{a, b\}, \{x, y\})$ 上のパラメタ化文字列 $xyxyaxxyb$ に対する prev 符号は $0022a314b$ である．

パラメタ化文字列 t と s は，$prev(\mathtt{t}) = prev(\mathtt{s})$ のとき，かつそのときに限りパラメタ化一致することが容易に確かめられる．パラメタ化パターン照合問題を以下のように定義する．

定義 11 (Σ, Π) 上のパラメタ化テキスト t とパラメタ化パターン p が与えられたとき，t の部分文字列で p とパラメタ化一致するすべての位置を出力する．

パラメタ化パターン照合問題を解くためのデータ構造として，パラメタ化ポジションヒープを提案する．

5.2 パラメタ化ポジションヒープ

定義 12 (Σ, Π) 上の長さ n のパラメタ化文字列 t に対する**パラメタ化ポジションヒープ** $PPH(\mathtt{t}) = (V_n, E_n)$ は以下のように再帰的に定義される $(\mathcal{N} \cup \Sigma)$ 上のトライ木である．

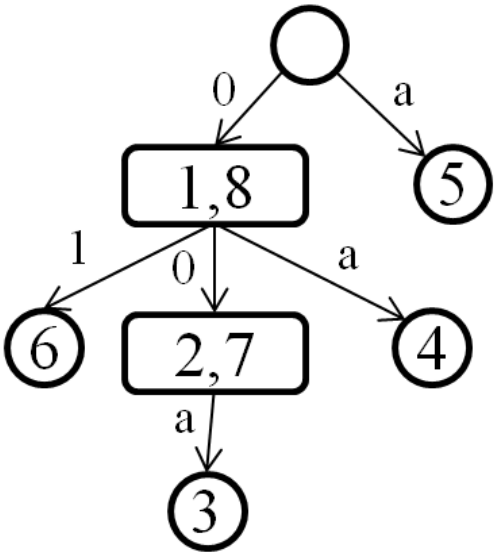
$$
\begin{aligned}
V_i &= V_{i-1} \cup \{prev(p_i)\} \\
E_i &= E_{i-1} \cup \{(prev(q_i), c, prev(p_i))\} \quad (1 \leq i \leq n)
\end{aligned}
$$

ただし $V_0 = \{\varepsilon\}$, $E_0 = \emptyset$ である．ここで，p_i は $prev(p_i) \notin V_{i-1}$ となる $\mathtt{t}[i..n]$ の最短の接頭辞で，$q_i = \mathtt{t}[i..i + |p_i| - 1]$, $c = prev(\mathtt{t}[i..n])[|p_i|]$ である．この条件を満たす p_i が存在する場合の i を**主位置**と呼ぶ．p_i が存在しない場合の i を**副位置**と呼び，$V_i = V_{i-1}$，$E_i = E_{i-1}$ とする．また，各頂点には文字列の始点となる位置 i を付与し，i 番目に $PPH(\mathtt{t})$ に追加された $prev$ 符号 $pspref_i$ を以下のように定義する．

$$
pspref_i = \begin{cases} prev(p_i) & (p_i\text{が存在する場合}) \\ prev(\mathtt{t}[i..n]) & (p_i\text{が存在しない場合}) \end{cases}
$$

例としてパラメタ化文字列 $xyxyaxxy$ に対するパラメタ化ポジションヒープを図 5.1 に示す．ここで 1,2,3,4,5,6 は主位置であり，7 と 8 は副位置である．

$PPH(\mathtt{t})$ の構築を効率よく行うために 3 つの補題を示す．



1	2	3	4	5	6	7	8
x	y	x	y	a	x	x	y
0	0	2	2	a	3	1	4
	0	0	2	a	3	1	4
		0	0	a	3	1	4
			0	a	0	1	4
				a	0	1	0
					0	1	0
						0	0
							0

図 5.1: $(\{a\},\{x,y\})$ 上のパラメタ化文字列 $\mathtt{t} = xyxyaxxy$ に対するパラメタ化ポジションヒープ．これは順序集合 $(0022a314, 002a314, 00a314, \ldots, 00, 0)$ に対するシーケンスハッシュ木であるが，$prev(t) = 0022a314$ に対するポジションヒープではないことに注意されたい．

補題 6. 任意のパラメタ化文字列 $\mathtt{x}$，$\mathtt{y}$ に対して $prev(\mathtt{x}) = prev(\mathtt{y})$ ならば $prev(\mathtt{x}[2..|\mathtt{x}|]) = prev(\mathtt{y}[2..|\mathtt{y}|])$ となる．

証明 先頭文字が Σ の要素である場合は自明であるため，先頭文字が Π の要素である場合を考える．$prev(\mathtt{x}) = prev(\mathtt{y})$ のとき $prev(\mathtt{x}[2..|\mathtt{x}|]) \neq prev(\mathtt{y}[2..|\mathtt{y}|])$ と仮定すると，先頭一文字削った際に新たに prev 符号において 0 になる位置は異なるが，その位置は $prev(\mathtt{x})$，$prev(\mathtt{y})$ において，先頭位置との差分となっており，$prev(\mathtt{x}) = prev(\mathtt{y})$ に矛盾する．よって、$prev(\mathtt{x}) = prev(\mathtt{y})$ ならば $prev(\mathtt{x}[2..|\mathtt{x}|]) = prev(\mathtt{y}[2..|\mathtt{y}|])$ となる． □

補題 7. パラメタ化文字列 $\mathtt{t}$ に対するパラメタ化ポジションヒープ $PPH(\mathtt{t})$ に $prev(\mathtt{w})$ が含まれるならば，$prev(\mathtt{w}[2..|\mathtt{w}|])$ も含まれる．

証明 $PPH(\mathtt{t})$ に $prev(\mathtt{w})$ が含まれるならば，トライ木の性質より $prev(\mathtt{w}[1..1])$, $prev(\mathtt{w}[1..2])$, ..., $prev(\mathtt{w}[1..|\mathtt{w}|-1])$ が含まれる．よって，$\mathtt{t}$ には $1 \leq i \leq |\mathtt{w}|$ に対して $prev(\mathtt{z}_i) = prev(\mathtt{w}[1..i])$ となる部分文字列 $\mathtt{z}_1$，$\mathtt{z}_2$，...，$\mathtt{z}_{|\mathtt{w}|}$ がこの順序で現れる．このことから，$\mathtt{t}$ には $\mathtt{z}_2[2..|\mathtt{z}_2|]$，$\mathtt{z}_3[2..|\mathtt{z}_3|]$，...，$\mathtt{z}_{|\mathtt{w}|}[2..|\mathtt{z}_{|\mathtt{w}|}|]$ がこの順序で現れる．よって，$PPH(\mathtt{t})$ には $prev(\mathtt{z}_2[2..|\mathtt{z}_2|])$，$prev(\mathtt{z}_3[2..|\mathtt{z}_3|])$，...，$prev(\mathtt{z}_{|\mathtt{w}|}[2..|\mathtt{z}_{|\mathtt{w}|}|])$ に対応する文字列がこの順番で

挿入されており，補題 6 より $PPH(\mathtt{t})$ には $prev(\mathtt{w}[2..2])$，$prev(\mathtt{w}[2..3])$，...，$prev(\mathtt{w}[2..|w|])$ が含まれることがわかる．□

補題 8. 長さ n のパラメタ化文字列 t に対するパラメタ化ポジションヒープ $PPH(\mathtt{t})$ において，$q < n$ が副位置ならば $q+1$ も副位置である．

証明 q は副位置なので，$pspref_p = pspref_q$ となる $p < q$ が存在する．補題 7 より，$k = |pspref_p|$ とすると $PPH(\mathtt{t})$ には $prev(\mathtt{t}[p+1..p+k])$，$prev(\mathtt{t}[q+1..q+k])$ が含まれる．ここで副位置 q に対する $pspref_q$ の定義より $pspref_q = prev(\mathtt{t}[q..n])$ なので，$prev(\mathtt{t}[q+1..q+k]) = pspref_{q+1}$ となる．よって任意の $q < n$ が副位置ならば $q+1$ も副位置となる．□

補題 8 より，主位置と副位置は連続した 2 つの領域を構成する．即ち，1 から $s-1$ が主位置で s から n が副位置であるような s が存在する．このような s を**作業位置**と呼び，$PPH(\mathtt{t})$ を構築する際に利用する．

5.3 線形時間構築アルゴリズム

パラメタ化ポジションヒープの構築アルゴリズムを Algorithm 5 に示す．このアルゴリズムは初期化と n 回のループからなる．初期化では $root$ と $\perp$ を作成し，$\perp$ から $root$ に任意の文字で辿ることのできる辺を作成する．第 i 番目のループにおいてパラメタ化文字 $\mathtt{t}[i]$ を読み込み $PPH(\mathtt{t}[1..i-1])$ を次のように定義されるパラメタ化接尾辞ポインタ psp を用いて更新し，$PPH(\mathtt{t}[1..i])$ を構築する．

定義 13 (パラメタ化接尾辞ポインタ)．$PPH(\mathtt{t})$ の**パラメタ化接尾辞ポインタ** psp は，$root$ 以外の頂点 $v = prev(\mathtt{t}[i..j])$ に対して $psp(v) = prev(\mathtt{t}[i+1..j])$ と定義する．また $root$ に対し $psp(root) = \perp$ とする．

更新作業によって主位置が副位置に変化することはないため，副位置の更新のみを考えればよい．また，補題 7 より，$PPH(\mathtt{t})$ の任意の頂点 v に対して，パラメタ化接尾辞ポインタを辿った先の頂点 $psp(v)$ が常に存在することから，$PPH(\mathtt{t}[1..i-1])$ の作業位置 s に対して $pspref_s$ の接尾辞ポインタは $pspref_{s+1}$ を指す．よって，s 以外の副位置に対する文字列を表す頂点は，$pspref_s$ を表す頂点から接尾辞ポインタを辿ることで得ること

ができる．このことから，更新作業は s の更新とパラメタ化接尾辞ポインタの構築を行うだけで，s 以外の副位置の更新は考慮しなくてもよい．

$\mathtt{t}[i]$ を読み込んだとき，$PPH(\mathtt{t}[1..i-1])$ に $prev(\mathtt{t}[s..i])$ を表す頂点が既に存在する場合，s は副位置になるが，しない場合は以下の更新作業を行うことで主位置になる．

$PPH(\mathtt{t}[1..i-1])$ において新たに $p = prev(\mathtt{t}[s..i])$ を表す頂点 u を作成し，$pspref_s$ を表す頂点 v からラベル $p[|p|]$ で u につながる辺を張ることで $PPH(\mathtt{t}[1..i])$ を構築する．作業位置は u の接尾辞ポインタ $psp(u)$ を辿ることで新たに $s+1$ に変わる．$s+1$ が主位置になるか副位置になるかによって繰り返し更新を行う．

パラメタ化文字を読み込んだ時と作業位置 s の値が増えたとき位置 s からはじまる部分文字列をエンコードしなおす必要がある．この操作を定数時間で行うために next 符号 [2] を用いる．

定義 14 (next 符号)．(Σ, Π) 上のパラメタ化文字列 $\mathtt{t}$ に対して，次式で定義される $\mathcal{N} \cup \Sigma$ 上の文字列 $next(\mathtt{t}) = u$ を **next 符号**と呼ぶ．

$$u[i] = \begin{cases} \mathtt{t}[i] & (\mathtt{t}[i] \in \Sigma \text{のとき}) \\ 0 & \left(\begin{array}{l} \mathtt{t}[i] \in \Pi \text{で，すべての } i < j \leq n \\ \text{に対して } \mathtt{t}[i] \neq \mathtt{t}[j] \text{ のとき} \end{array} \right) \\ -i + \min\{j | \mathtt{t}[j] = \mathtt{t}[i] \text{ かつ } i < j \leq n\} & (\text{その他}) \end{cases}$$

例えば，$next(xyxyaxxyb) = 2234a100b$ である．

prev 符号, next 符号と Π 上の各パラメタ文字の最後の出現位置を記憶しておくことで，s から始まるパラメタ化文字列に対する prev 符号と next 符号を求めることができる．パラメタ文字を読み込んだ際は Algorithm 5 の 7 行目から 14 行目のように読み込んだ文字の最後の出現位置から prev 符号が，prev 符号から next 符号が定数時間で更新できる．作業位置 s が増えた際は Algorithm 5 の 25 行目から 30 行目のように next 符号から prev 符号の値を定数時間で更新することができる．$PPH(\mathtt{t}[s..i])$ における prev 符号と next 符号はそれぞれ $prev[s..i], next[s..i]$ に格納される．

上記の更新作業で新たに頂点 $u_1, u_2, \ldots, u_k$ がこの順序で作成され，作業位置 $s+k$ に対する $pspref_{s+k}$ を表す頂点が v であるとき，パラメタ化接尾辞ポインタ psp は $psp(u_i) = u_{i+1}$

$(1 \leq i \leq k-1)$, $psp(u_k) = v$ となる．なぜなら，作成された頂点 $u_1, u_2, \ldots, u_k$ が表すパラメタ化文字列の prev 符号はそれぞれ，先頭一文字を削り末尾に文字 $\mathtt{t}[i]$ を追加した文字列の prev 符号 $prev(\mathtt{t}[s..i]), prev(\mathtt{t}[s+1..i]), \ldots, prev(\mathtt{t}[s+k-1..i])$ であり，更新作業後の作業位置からの prev 符号は $prev(\mathtt{t}[s+k..i])$ を表すからである．このように作業位置の更新とパラメタ化接尾辞ポインタの構築を繰り返しながら $PPH(\mathtt{t})$ を構築する．

1 回の更新で作業位置 s に対する $pspref_s$ の長さを縮める操作が何度か繰り返されることがあるが，全体 n 回のループでは読み込んだ n 文字しか $pspref_s$ の長さを縮めることができないので，Algorithm 5 は $O(n)$ 時間で終了する．

定理 3. *Algorithm 5*は，長さ n のパラメタ化文字列 $\mathtt{t}$ に対するパラメタ化ポジションヒープ $PPH(\mathtt{t})$ を $O(n)$ 時間で構築する．

5.4 素朴なパターン照合アルゴリズム

長さ n のパラメタ化テキスト $\mathtt{t}$ とパラメタ化ポジションヒープ $PPH(\mathtt{t})$，長さ m のパラメタ化パターン $\mathtt{p}$ が与えられたとき，$\mathtt{t}$ の中に表れる $\mathtt{p}$ とパラメタ化一致する部分文字列の位置をすべて求めることを考える．はじめに $prev(\mathtt{p})$ を $O(m)$ 時間で求める．次に，$PPH(\mathtt{t})$ を $prev(\mathtt{p})$ で辿ることで最大 $m-1$ 個のパターンと照合している可能性のある位置が得られる．照合位置の候補から m 文字の部分文字列に対して prev 符号を $O(m)$ 時間で求め，$prev(\mathtt{p})$ との照合を行う．加えて，$prev(\mathtt{p})$ が $PPH(\mathtt{t})$ に含まれている場合に限り到達した頂点とその子孫に対応する位置がパターンの照合位置となる．よって，合計で $O(m^2 + occ)$ 時間でパターン照合を行うことができる．ここで，occ はパターンの出現数を表す．

定理 4. 素朴なパターン照合アルゴリズム *5.4* はパラメタ化テキスト $\mathtt{t} \in (\Sigma \cup \Pi)^n$ と $\mathtt{t}$ に対するパラメタ化ポジションヒープ $PPH(\mathtt{t})$，パラメタ化パターン $\mathtt{p} \in (\Sigma \cup \Pi)^m$ が与えられたとき，パラメタ化パターン照合問題を $O(m^2 + occ)$ 時間で解く．

5.5 確率的パターン照合アルゴリズム

素朴なパターン照合アルゴリズムにおいて問題となっている点は，パターンの各候補位置に対して，検証を行うために$O(m)$時間必要になっているところである．ここでは，前処理として$O(n)$時間で$O(n)$領域のハッシュ配列を新たに構築し，確率的に$O(|\Pi|)$時間で候補の検証を行う．このアルゴリズムの利点は後述する極大到達ポインタを用いたアルゴリズムに比べて実装が容易であり，高速に動作する点である．

長さnのパラメタ化文字列$\mathtt{t}$のprev符号に対するハッシュ関数$pH_{\alpha,\beta}$とハッシュ配列$PHA_{\alpha,\beta}$を以下のように定義する．

$$
\begin{aligned}
pH_{\alpha,\beta}(\mathtt{t}) &= \sum_{i=1}^{n} \phi_p(prev(\mathtt{t})[i]) \cdot \alpha^{i-1} \bmod \beta \\
PHA_{\alpha,\beta}[i] &= PHA_{\alpha,\beta}[i-1] + \phi_p(prev(\mathtt{t})[i]) \cdot \alpha^{i-1} \bmod \beta \ (1 \le i \le n) \\
PHA_{\alpha,\beta}[0] &= 0
\end{aligned}
$$

ϕ_pは$a, b \in \Sigma$に対して$a \neq b$ならば$\phi_p(a) \neq \phi_p(b)$かつ$a = b$ならば$\phi_p(a) = \phi_p(b)$となり，$c \in \mathcal{N}$に対して$\phi_p(c) = c$となる$(\Sigma \cup \mathcal{N})$から$\mathcal{N}$への写像である．prev符号におけるアルファベットサイズは高々nなので，βを大きな素数，αをβを法としたときのn以上の原始根とすることで，[10]より，以下の補題が示せる．

補題 9. αをβを法としたときの文字列長以上の原始根，βを大きな素数とすると，同じ長さの異なるパラメタ化文字列$\mathtt{t_1}, \mathtt{t_2}$のハッシュ値$pH_{\alpha,\beta}(\mathtt{t_1})$と$pH_{\alpha,\beta}(\mathtt{t_2})$が同じ値になる確率は$\frac{1}{\beta}$以下となる．

Σ上の文字列に対する確率的パターン照合アルゴリズムでは，ハッシュ配列から$O(1)$時間で比較を行うことができたが，(Σ, Π)上のパラメタ化文字列$\mathtt{t} \in (\Sigma \cup \Pi)^n$の場合は，各位置$1 \le i \le n$から始まる部分文字列で最も左側に出現する$\Pi$の要素はprev符号において0となるため，単純に比較を行うことができない．prev符号において0となる位置を考慮したハッシュ値の比較アルゴリズムをAlgorithm 6に示す．入力はパラメタ化テキスト$\mathtt{t}$，ハッシュ配列$PHA_{\alpha,\beta}$，候補となる部分文字列の始点$start$，パターン中の

Π の各要素の最左出現位置の集合 $focc$，パターンのハッシュ値 $pHash$，パターン長 m，逆元配列 $INV_{\alpha,\beta}$ である．$focc$ はパターンを走査することで $O(m)$ 時間で求めることができる．Algorithm 6 は，はじめにテキストの位置 $start$ からはじまる不完全なハッシュ値を求める．次にパターン中の各 Π の要素の最左出現位置を用いてテキストの部分文字列の prev 符号中で 0 になる位置 $zeroPos$ を求めハッシュ値を修正していく． 4 行目の if 文で $zeroPos$ の要素が確かに Π の要素であることを確かめている． 6 行目の if 文では位置 $zeroPos$ の prev 符号の値が 0 かどうか確かめており，0 でない場合は， 8 行目の if 文で，$zeroPos$ が $start$ からはじまる部分文字列で本当に最左出現の Π の要素かを確かめている．アルゴリズム中で用いる $prev(\mathtt{t})$ は β がテキスト長より大きい場合はハッシュ値から偽陽性なしで $O(1)$ 時間で復元でき，それ以外の場合は新たな記憶領域に $prev(\mathtt{t})$ を記憶しておく．最後のハッシュ値の比較文において，パラメタ化一致していない場合に限りテキストの部分文字列に対するハッシュ値が正しく復元できない場合がある．例えば $(\{a,b\},\{x,y,z\})$ 上のパラメタ化テキスト $zxzyb$ とパターン $xayb$ のときがあげられる．しかし，その場合のハッシュ値にする前の prev 符号のアルファベットサイズも高々テキスト長なためハッシュ値の評価には問題がないと考えられる．

ハッシュ値による比較は最大 $m-1$ 回行われるため，このアルゴリズムが正しく動作する確率は $\left(1-\frac{1}{\beta}\right)^{m-1}$ となる．よって，素朴なパターン照合アルゴリズムの検証部分をハッシュ値の比較に変えることで確率 $\left(1-\frac{1}{\beta}\right)^{m-1}$ でパターン照合問題を $O(m|\Pi|+occ)$ 時間で解くことができる．

定理 5. *パラメタ化ポジションヒープを用いた確率的パターン照合アルゴリズム 5.5 は，長さ n のパラメタ化テキスト $\mathtt{t}$ とパラメタ化ポジションヒープ $PPH(\mathtt{t})$，$prev(\mathtt{t})$ に対するハッシュ配列 $PHA_{\alpha,\beta}$，逆元配列 $INV_{\alpha,\beta}$，長さ m のパラメタ化パターン $\mathtt{p}$ が与えられたとき，確率 $\left(1-\frac{1}{\beta}\right)^{m-1}$ でパターン照合問題を $O(m|\Pi|+occ)$ 時間で解く*

5.6 拡張パラメタ化ポジションヒープを用いたパターン照合

ここでは，パラメタ化ポジションヒープに対してパラメタ化極大到達ポインタと子孫判定を定数時間で行うためのデータ構造を付与した拡張パラメタ化ポジションヒープを用いて，パターン $\mathtt{p}\in(\Sigma\cup\Pi)^m$ が与えられたとき，$O(m|\Pi|+occ)$ 時間でパターン照合

問題を解くアルゴリズムを提案する．

パラメタ化テキスト $\mathtt{t}$ に対するパラメタ化ポジションヒープ $PPH(\mathtt{t})$ に対する**パラメタ化極大到達ポインタ** $pmrp$ を以下のように定義する．

定義 15 (パラメタ化極大到達ポインタ)．各位置 $1 \leq i \leq n$ に対応する頂点 $pspref_i$ に対して，**パラメタ化極大到達ポインタ** $pmrp$ は以下の式で表される．

$$pmrp(pspref_i) = arg\ max_{s \in S_i}|s|$$

$$S_i = \{pspref_j \mid 1 \leq j \leq n \text{ かつ } prev(\mathtt{t}[i..i + |pspref_j|]) = pspref_j\}$$

パラメタ化極大到達ポインタに加えてトライ木の 2 頂点 u, v が与えられたときに，u が v の子孫であるかの判定を定数時間で行うためのデータ構造は，通常の拡張ポジションヒープに付与したデータ構造を用いる．パラメタ化極大到達ポインタと子孫判定を用いることで，ポジションヒープの頂点 v に対する極大到達ポインタ $pmrp(v)$ が頂点 u またはその子孫であれば，v の位置から始まる部分文字列の接尾辞に対する prev 符号が u になることがわかる．子孫判定を行うデータ構造とパラメタ化極大到達ポインタを付与したパラメタ化ポジションヒープを**拡張パラメタ化ポジションヒープ**と呼ぶ．

5.6.1 パラメタ化極大到達ポインタ構築アルゴリズム

ここでは，パラメタ化接尾辞ポインタを用いてパラメタ化テキスト $\mathtt{t} \in (\Sigma \cup \Pi)^n$ に対するパラメタ化ポジションヒープ $PPH(\mathtt{t})$ に対するパラメタ化極大到達ポインタを $O(n)$ 時間で構築するアルゴリズムを提案する．

アルゴリズムを Algorithm 7 に示す．図中の $prevTable[left..right]$, $nextTable[left..right]$ には着目している部分文字列 $\mathtt{t}[left..right]$ の prev 符号，next 符号がそれぞれ格納される． 19 行から 29 行の操作と 31 行から 38 行の操作で部分文字列 $\mathtt{t}[left..right]$ の prev 符号，next 符号を更新する． 12 行目の while ループで位置 $left$ に対する頂点から辿れるだけ辿り，$pmrp(pspref_{left})$ を求めている． 31 行目にて接尾辞ポインタを辿るが，補題 7 より，$prev(\mathtt{t}[left..right])$ が存在するならば，$prev(\mathtt{t}[left + 1..right])$ が存在するため，この操作は正しい．各位置 $1 \leq left \leq n$ に対して，$pmrp(pspref_{left}) = prev(\mathtt{t}[left..right])$

を求めているが，$left$，$right$ はともに単調増加であるため，このアルゴリズムは $O(n)$ 時間で動作する．

定理 6. *Algorithm 7*は，パラメタ化テキスト $\mathtt{t} \in (\Sigma \cup \Pi)^n$ に対するパラメタ化ポジションヒープ $PPH(\mathtt{t})$ に対するパラメタ化極大到達ポインタを $O(n)$ 時間で構築する．

5.6.2 パターン照合アルゴリズム

拡張パラメタ化ポジションヒープを用いたパターン照合アルゴリズムを提案する．アルゴリズムを Algorithm 8 に示す． 長さ m のパターン $\mathtt{p}$ が与えられたときに，最大 k 文字パラメタ化拡張ポジションヒープを辿れたとすると，辿る際に通った頂点を記憶しておき，再び根から k 文字辿る過程で通った頂点に対して，極大到達ポインタを用いて 16 行目の if 文で子孫判定を行うことで，各頂点が持つ位置から始まる部分文字列が，k 文字以上パターンとパラメタ化一致しているかを判定する．判定に通過した候補の位置に k を加えることで，新たに読み込むパターンの接尾辞の開始位置に対応したテキスト中の位置が得られる． 22 行目の while ループ内で残りのパターンの接尾辞がパラメタ化一致しているかの判定を行う．辿れなかったパターンの接尾辞に対する prev 符号で根から新たに k' 文字辿り，新たに読み込むパターンの接尾辞の開始位置に対応したテキスト中の位置に対して， 33 行目の if 文で子孫判定と Π の要素が正しく接続しているかの判定を行うことで，新たに辿れた部分文字列が k' 文字以上パターンとパラメタ化一致しているかを判定している．この判定をパターンをすべて読み込むまで繰り返す．while ループの中で辿れた深さの数しかパターンの候補がないので， 33 行目の判定の回数は全体で $2m$ 回以下となる．判定を通過した候補に対して， 36 行目のパターンにおける Π の要素の最左出現の位置のテキストでの要素が互いに異なるかの判定を行う． 33， 36 行目の操作の正しさを述べるために以下の補題を示す．

補題 10. パラメタ化テキスト $\mathtt{t}$，$\mathtt{s} \in (\Sigma \cup \Pi)^n$，$1 \leq k < n$ に対して $prev(\mathtt{t}[1..k]) = prev(\mathtt{s}[1..k])$，$prev(\mathtt{t}[k+1..n]) = prev(\mathtt{s}[k+1..n])$ を満たし，$prev(\mathtt{s}[k+1..n])$ において新たに 0 となる Π の要素の $\mathtt{s}[1..\mathtt{k}]$ における最右と $\mathtt{s}[\mathtt{k}+1..\mathtt{n}]$ における再左の $\mathtt{s}$ に対する位置の集合 A，$\mathtt{s}$ に含まれる Π の要素の最左の位置の集合 B が与えられたとき，$prev(\mathtt{t}) = prev(\mathtt{s})$ の判定を $O(|\Pi|)$ 時間で行うことができる．

証明 集合 A，B はそれぞれ以下の式で表すことができる．

$$A = \{(i - prev(s)[i], i) \mid 1 \leq i \leq n, prev(\mathtt{s}[k+1..n])[i-k] = 0 \text{ かつ } prev(s)[i] \neq 0\}$$

$$B = \{1 \leq i \leq n \mid s[i] \in \Pi \text{かつ任意の } 1 \leq j < i \text{ に対して } s[j] \neq s[i]\}$$

$prev(\mathtt{t}[1..k]) = prev(\mathtt{s}[1..k])$，$prev(\mathtt{t}[k+1..n]) = prev(\mathtt{s}[k+1..n])$ を満たすので，$prev(\mathtt{t}[k+1..n])$ と $prev(\mathtt{s}[k+1..n])$ において 0 である位置の $\mathtt{t}[k+1..n]$ と $\mathtt{s}[k+1..n]$ における要素がそれぞれ $\mathtt{t}[1..k]$ と $\mathtt{s}[1..k]$ において表れていて正しく接続できているか，または初めて出現しており既出の要素と異なっているかの判定が行えればよい．前者については A の各要素 a に対して $\mathtt{t}[a] = \mathtt{s}[a]$ を確かめればよく，後者については B の各要素 b に対して $\mathtt{t}[b]$ が互いに異なればよい．集合 A，B の大きさは高々$|\Pi|$ であるため，$prev(\mathtt{t}) = prev(\mathtt{s})$ の判定を $O(|\Pi|)$ 時間で行うことができる． □

補題 10 より 33，36 行目の操作それぞれは$O(|\Pi|)$時間で行える．よって全体で$O(m|\Pi| + occ)$ 時間でパラメタ化パターン照合問題を解くことができる．

$k = m$ の場合は通常のパラメタ化ポジションヒープを用いたパラメタ化パターン照合アルゴリズムと同様に辿った先の頂点とその子孫にあたる頂点に対応する位置を解に含めることで $O(m + occ)$ 時間でパラメタ化パターン照合問題を解くことができる．

定理 7. *Algorithm 8* はパラメタ化テキスト $\mathtt{t}$ に対する拡張パラメタ化ポジションヒープとパラメタ化パターン $\mathtt{p} \in (\Sigma \cup \Pi)^m$ が与えられたとき $O(m|\Pi| + occ)$ 時間でパラメタ化パターン照合問題を解く．

Algorithm 5: パラメタ化ポジションヒープ構築アルゴリズム

Input: 長さ n のパラメタ化文字列 $\mathtt{t}$

Output: パラメタ化ポジションヒープ $PPH(\mathtt{t})$

create $root$ and $\perp$ nodes;
$psp(root) = \perp$;
for each character c, create edge $(\perp, c, root)$;
$currentNode = root$;
$s = 1$;
for $i = 1$ **to** n **do**
  if $\mathtt{t}[i] \in \Pi$ **then**
    $next[i] = 0$;
    if $lastOccur[\mathtt{t}[i]]$ = undefined **then**
      $prev[i] = 0$;
    else
      $prev[i] = i - lastOccur[\mathtt{t}[i]]$;
      $next[i - prev[i]] = prev[i]$;
    $lastOccur[\mathtt{t}[i]] = i$;
  else
    $prev[i] = \mathtt{t}[i], next[i] = \mathtt{t}[i]$;
  $lastCreateNode$ = undefined;
  while not exist edge $(currentNode, prev[i], nextNode)$ **do**
    create node $pspref_s$;
    create edge $(currentNode, prev[i], pspref_s)$;
    if $lastCreateNode \neq$ undefined **then**
      $psp(lastCreateNode) = pspref_s$;
    $lastCreateNode = pspref_s$;
    $currentNode = psp(currentNode)$;
    if $\mathtt{t}[s] \in \Pi$ **then**
      if $next[s] > 0$ **then**
        $prev[s + next[s]] = 0$;
      else
        $lastOccur[\mathtt{t}[s]]$ = undefined;
    $s = s + 1$;
  $currentNode = nextNode$;
  if $lastCreateNode \neq$ undefined **then**
    $psp(lastCreateNode) = currentNode$;

Algorithm 6: ハッシュ配列を用いたハッシュ値の比較

Input: パラメタ化テキスト t，ハッシュ配列 $PHA_{\alpha,\beta}$，候補となる部分文字列の始点 $start$，パターン中の Π の各要素の最左出現位置の集合 $focc$，パターンのハッシュ値 $pHash$，パターン長 m，逆元配列 $INV_{\alpha,\beta}$

Output: 候補となる部分文字列の始点から m 文字の部分文字列に対するハッシュ値 $tHash$ が $pHash$ と一致しているか否か

```
1  tHash = (PHA_{α,β}[start + m − 1] − PHA_{α,β}[start − 1]) mod β;
2  foreach pos ∈ focc do
3      zeroPos = start+pos;
4      if t[zeroPos] ∈ Σ then
5          return false;
6      if prev(t)[zeroPos] = 0 then
7          continue;
8      if zeroPos − prev(t)[zeroPos] ≥ start then
9          return false;
10     tHash = tHash − (PHA_{α,β}[zeroPos] − PHA_{α,β}[zeroPos − 1]);
11 return (tHash · INV_{α,β}[start − 1] mod β)=pHash;
```

Algorithm 7: パラメタ化極大到達ポインタ構築アルゴリズム

Input: パラメタ化テキスト $\mathtt{t}$，パラメタ化ポジションヒープ $PPH(\mathtt{t})$ とパラメタ化接尾辞ポインタ $psp()$

Output: パラメタ化極大到達ポインタ $pmrp$

```
1  n = |t|;
2  currentNode = root;
3  right = 1;
4  nextTable = prevTable = array of length n;
5  lastOccur = array of length |Π| initialized undefined;
6  if t[1] ∈ Π then
7  |  nextTable[1] = prevTable[1] = 0;
8  |  lastOccur[t[1]] = 1;
9  else
10 |  nextTable[1] = prevTable[1] = t[1];
11 for left = 1 to n do
12 |  while right ≤ n do
13 |  |  if not exist edge(currentNode, prevTable[right], nextNode) then
14 |  |  |  break;
15 |  |  currentNode = nextNode;
16 |  |  right += 1;
17 |  |  if right > n then
18 |  |  |  break;
19 |  |  if t[right] ∈ Π then
20 |  |  |  nextTable[right] = 0;
21 |  |  |  if lastOccur[t[right]] = undefined then
22 |  |  |  |  prevTable[right] = 0;
23 |  |  |  else
24 |  |  |  |  prevTable[right] = right-lastOccur[t[right]];
25 |  |  |  |  nextTable[right − prevTable[right]] = prevTable[right];
26 |  |  |  lastOccur[t[right]] = right;
27 |  |  else
28 |  |  |  nextTable[right] = prevTable[right] = t[right];
29 |  |
30 |  pmrp(left) = currentNode;
31 |  currentNode = psp(currentNode);
32 |  if t[left] ∈ Π then
33 |  |  if nextTable[left] > 0 then
34 |  |  |  prevTable[left + nextTable[left]] = 0;
35 |  |  else
36 |  |  |  lastOccur[t[left]] = undefined;
37 |  |
38 |
39 return pmrp;
```

Algorithm 8: 拡張パラメタ化ポジションヒープを用いたパターン照合アルゴリズム

Input: パラメタ化テキスト t に対する拡張パラメタ化ポジションヒープ，パラメタ化パターン p

Output: パターンの出現位置に対応した頂点リスト ans

```
1  m = |p|;
2  build prev,next encoding of p into prevTable, nextTable;
3  u = root;
4  for i = 1 to m do
5      if not exist edge (u, prevTable[i], nextNode) then
6          break;
7      u = nextNode;
8  ans = empty list;
9  v = root;
10 ok = true;
11 for i = 1 to m do
12     if not exist edge (v, prevTable[i], nextNode) then
13         ok = false;
14         break;
15     v = nextNode;
16     if pmrp(v) is descendant of u then
17         add v to ans;
18
19 if ok=true then
20     add positions of all descendants of v to ans;
21 else
22     while i ≠ m do
23         build prev encoding of p[i..m] into prevTable[i..m] using nextTable;
24         zp = places of new zero in prevTable;
25         q = root;
26         for i to m do
27             if not exist edge (q, prevTable[i], nextNode) then
28                 break;
29             q = nextNode;
30         if q=root then
31             return empty list;
32         foreach w ∈ ans do
33             if w goes on a check using pmrp and zp, leave w as candidate;
34
35     foreach w ∈ ans do
36         if w goes on a check using first-occ of elements of Π, leave w as candidate;
37
38 return ans;
```

第6章

順列パターン照合問題に対するポジションヒープ

ここでは，順列パターン照合問題に対するポジションヒープであるマルチトラックポジションヒープ/縮約マルチトラックポジションヒープの定義と構築アルゴリズムの提案，マルチトラックポジションヒープ/縮約マルチトラックポジションヒープを用いたパターン照合アルゴリズムの提案を行う．

6.1 順列一致

$(1, 2, \ldots, N)$ の順列を $\mathbf{r} = (r_1, r_2, \ldots, r_N)$ とする．$\mathbf{r}$ で決まるマルチトラック $\mathbb{X} = (x_1, x_2, \ldots, x_N)$ の順列マルチトラック文字列とは $\mathbb{Y} = (x_{r_1}, x_{r_2}, \ldots, x_{r_N})$ であり，$\mathbb{Y} = \mathbb{X}\langle\mathbf{r}\rangle$ で表す．

定義 16 (順列一致)．トラック数 N，トラック長 n のマルチトラック $\mathbb{X}, \mathbb{Y}$ に対して，$\mathbb{X}$ のトラックを $(1, 2, \ldots, N)$ の順列 $\mathbf{r} = (r_1, r_2, \ldots, r_N)$ で並べ替えたときに，$\mathbb{Y}$ と一致するような $\mathbf{r}$ が存在するならば，$\mathbb{X}$ と $\mathbb{Y}$ は**順列一致する**という．

順列一致を効率的に行うためにマルチトラックのソート順列と連結文字列を定義する．

定義 17 (ソート順列)．マルチトラック $\mathbb{X} = (x_1, x_2, \ldots, x_N)$ において $x_{r_i} \leq x_{r_j}$ $(1 \leq i \leq j \leq N)$ を満たす順列を**ソート順列**といい，$SI(\mathbb{X}) = (r_1, r_2, \ldots, r_N)$ で表す．マルチトラック $\mathbb{X} = (x_1, x_2, \ldots, x_N)$ の *sort* 演算子を $sort(\mathbb{X}) = \mathbb{X}\langle SI(\mathbb{X})\rangle$ で定義する．マルチトラック $\mathbb{T}$，$n = l(\mathbb{T})$，$1 \leq i \leq j \leq n$ に対して，$sort(\mathbb{T}[i..j]) = \mathbb{T}[i..j]\langle SI(\mathbb{T}[i..n])\rangle$ が成り立つ．

定義 18 (連結文字列)．長さ n のマルチトラック $\mathbb{X} = (x_1, x_2, \ldots, x_N)$ の**連結文字列**を $cs(\mathbb{X}) = x_1[1]x_2[1] \ldots x_1[2]x_2[2] \ldots x_1[n]x_2[n] \ldots x_N[n]$ とする．

マルチトラック $\mathbb{X}$, $\mathbb{Y}$ は $cs(sort(\mathbb{X})) = cs(sort(\mathbb{Y}))$ のとき，かつその場合に限り順列一致する．順列パターン照合問題を定義する．

定義 19 (順列パターン照合問題). マルチトラックテキスト $\mathbb{T}$ とマルチトラックパターン $\mathbb{P}$ を受け取り，$\mathbb{T}$ の部分文字列で $\mathbb{P}$ と順列一致するすべての位置を出力する．

順列パターン照合問題を解くためのデータ構造として，マルチトラックポジションヒープと縮約マルチトラックポジションヒープを提案する．

6.2 マルチトラックポジションヒープ

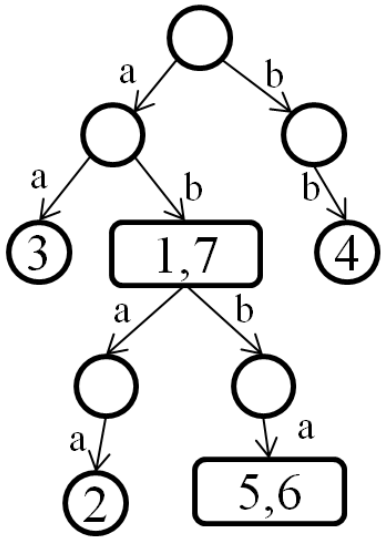
定義 20 (マルチトラックポジションヒープ). 長さ n のマルチトラック $\mathbb{T}$ に対する**マルチトラックポジションヒープ** $MTPH(\mathbb{T}) = (V_n, E_n)$ は以下のように再帰的に定義される Σ 上のトライ木である．

$$
\begin{aligned}
V_i &= V_{i-1} \cup \left(\bigcup_{j=1}^{N} \{ cs(p_i\langle \mathbf{r_i} \rangle)[1..L+j] \} \right) \\
E_i &= E_{i-1} \cup \left(\bigcup_{j=1}^{N} \{ (v_j, cs(c\langle \mathbf{r_i} \rangle)[j], u_j \} \right) \\
v_j &= cs(p_i\langle \mathbf{r_i} \rangle)[1..L+j-1], \quad u_j = cs(p_i\langle \mathbf{r_i} \rangle)[1..L+j], \\
\mathbf{r_i} &= SI(\mathbb{T}[i..n]), \quad L = l(q_i)N \quad (1 \le i \le n)
\end{aligned}
$$

ただし $V_0 = \{\varepsilon\}$，$E_0 = \emptyset$ である．ここで，p_i は $cs(p_i\langle \mathbf{r_i} \rangle) \notin V_{i-1}$ を満たす $\mathbb{T}[i..n]$ の最短接頭辞，$q_i = p_i[1..l(p_i)-1]$，$c = \mathbb{T}[i+l(p_i)]$ である．この条件を満たす p_i が存在する場合の i を**主位置**と呼ぶ．p_i が存在しない場合の i を**副位置**と呼び，$V_i = V_{i-1}$，$E_i = E_{i-1}$ とする．また，深さ正で N の倍数である頂点には文字列の始点となる位置 i を付与し，i 番目の再帰において $MTPH(\mathtt{t})$ に追加された連結文字列 $mtspref_i$ を以下のように定義する．

$$
mtspref_i = \begin{cases} cs(p_i\langle \mathbf{r_i} \rangle) & (p_i\text{が存在する場合}) \\ cs(\mathbb{T}[i..n]\langle \mathbf{r_i} \rangle) & (p_i\text{が存在しない場合}) \end{cases}
$$

マルチトラック $(abababa, baabbab)$ に対するマルチトラックポジションヒープの例を図 6.1 に示す．



$$\begin{array}{ccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ \left(\begin{array}{ccccccc} a & b & a & b & a & b & a \\ b & a & a & b & b & a & b \end{array}\right) \end{array}$$

1. a b b a a a b b a b b a a b
2. a b a a b b b a a b b a
3. a a b b a b b a a b
4. b b a b b a a b
5. a b b a a b
6. a b b a
7. a b

図 6.1: マルチトラック $(abababa, baabbab)$ に対するマルチトラックポジションヒープ.

$MTPH(\mathbb{T})$ の構築を効率よく行うために 3 つの補題を示す.

補題 11. 任意のマルチトラック $\mathbb{X}$, $\mathbb{Y}$ に対して, $sort(\mathbb{X}) = sort(\mathbb{Y})$ ならば $sort(\mathbb{X}[2..l(\mathbb{X})]) = sort(\mathbb{Y}[2..l(\mathbb{Y})])$ である.

証明 $sort(\mathbb{X}) = sort(\mathbb{Y})$ ならば, $\mathbb{X}[2..l(\mathbb{X})]$ と $\mathbb{Y}[2..l(\mathbb{Y})]$ を構成するトラックの集合は等しい. よって, $sort(\mathbb{X}) = sort(\mathbb{Y})$ ならば $sort(\mathbb{X}[2..l(\mathbb{X})]) = sort(\mathbb{Y}[2..l(\mathbb{Y})])$ である. □

補題 12. マルチトラック $\mathbb{T}$ に対するマルチトラックポジションヒープ $MTPH(\mathbb{T})$ に $cs(sort(\mathbb{W}))$ が含まれるならば, $cs(sort(\mathbb{W}[2..l(\mathbb{W})]))$ が含まれる.

証明 $MTPH(\mathbb{T})$ に $cs(sort(\mathbb{W}))$ が含まれるならば, トライ木の性質より $cs(sort(\mathbb{W}[1..1]))$, $cs(sort(\mathbb{W}[1..2]))$, ..., $cs(sort(\mathbb{W}[1..l(\mathbb{W})-1]))$ が含まれる. よって, $\mathbb{T}$ には $1 \leq i \leq l(\mathbb{W})$ に対して $sort(\mathbb{Z}_i) = sort(\mathbb{W}[1..i])$ となる部分文字列 $\mathbb{Z}_1, \mathbb{Z}_2, \ldots, \mathbb{Z}_{l(\mathbb{W})}$ がこの順序で現れる. このことから, $\mathbb{T}$ には $\mathbb{Z}_2[2..l(\mathbb{Z}_2)]$, $\mathbb{Z}_3[2..l(\mathbb{Z}_3)]$,...,$\mathbb{Z}_{l(\mathbb{W})}[2..l(\mathbb{Z}_{l(\mathbb{W})})]$ がこの順序で現れる. よって, $cs(sort(\mathbb{Z}_2[2..l(\mathbb{Z}_2)]))$, $cs(sort(\mathbb{Z}_3[2..l(\mathbb{Z}_3)]))$,...,$cs(sort(\mathbb{Z}_{l(\mathbb{W})}[2..l(\mathbb{Z}_{l(\mathbb{W})})]))$ に対応する文字列がこの順番で $MTPH(\mathbb{T})$ に挿入されおり, 補題 11 より, $MTPH(\mathbb{T})$ には $cs(sort(\mathbb{W}[2..2]))$, $cs(sort(\mathbb{W}[2..3]))$, ...,$cs(sort(\mathbb{W})[2..l(\mathbb{W})])$ が含まれることがわかる. □

補題 12 を用いて $MTPH(\mathbb{T})$ において任意の $q < n$ が副位置ならば $q+1$ も副位置であることを示す.

補題 13. 長さ n のマルチトラックに対するマルチトラックポジションヒープ $MTPH(\mathbb{T})$ において任意の $q < n$ が副位置ならば $q+1$ も副位置である．

証明 q は副位置なので，$mtspref_p = mtspref_q$ となる $p < q$ が存在する．補題 12 より，$k = |mtspref_p|/N$ とすると $MTPH(\mathbb{T})$ には $cs(sort(\mathbb{T}[p+1..p+k]))$，$cs(sort(\mathbb{T}[q+1..q+k]))$ が含まれる．ここで $mtspref_q = cs(sort(\mathbb{T}[q..n]))$ なので，$cs(sort(\mathbb{T}[q+1..q+k])) = mtspref_{q+1}$ となる．よって任意の $q < n$ が副位置ならば $q+1$ も副位置となる． □

このことから 1 から $s-1$ が主位置で s から n が副位置であるような s が存在する．s を**作業位置**と呼び，$MTPH(\mathbb{T})$ の構築時に利用する．

6.3 マルチトラックポジションヒープの線形時間構築アルゴリズム

マルチトラックポジションヒープの構築アルゴリズムを Algorithm 9 に示す．Algorithm 9 は初期化と n 回のループからなる．初期化では $SI(\mathbb{T}[i..n])(1 \le i \le n)$ を求め，根 $root$ と補助頂点 $\perp_j$ $(1 \le j \le N)$ を作成し，$\perp_j$ から $\perp_{j+1}$ $(1 \le j \le N-1)$ と $\perp_N$ から $root$ に任意の文字で辿ることのできる辺を作成する．第 i 番目のループにおいてマルチトラック文字列の末尾一列 $\mathbb{T}[i]$ を読み込み $MTPH(\mathbb{T}[1..i-1])$ をマルチトラック接尾辞ポインタと呼ばれる関数 $mtsp$ を用いて更新し，$MTPH(\mathbb{T}[1..i])$ を構築する．更新のために用いるマルチトラック接尾辞ポインタを以下のように定義する．

定義 21 (マルチトラック接尾辞ポインタ)．$MTPH(\mathbb{T})$ 中の $root$ 以外の任意の頂点 $v = cs(sort(\mathbb{T}[i..j]))$ に対して，**マルチトラック接尾辞ポインタ**を $mtsp(v) = cs(sort(\mathbb{T}[i+1..j]))$ とする．$root$ に対し $mtsp(root) = \perp_1$ とする．

補題 12 より，$MTPH(\mathbb{T})$ 中の任意の頂点に対し，マルチトラック接尾辞ポインタを辿った先の頂点が存在する．また，$MTPH(\mathbb{T}[1..i-1])$ の作業位置 s に対して $mtspref_s$ を表す頂点 v に対する接尾辞ポインタは $mtspref_{s+1}$ を指す．

$SI(\mathbb{T}[i..n])(1 \le i \le n)$ は，文字列 $t_1\$t_2\$\ldots\$t_N$ に対して接尾辞木 [18] を構築して木を辿り，辿った葉の順番を書き出すことで $O(nN)$ 時間で求めることができることが [12] で示されている．

$\mathbb{T}[i]$ を読み込んだときに $MTPH(\mathbb{T}[1..i-1])$ において $x{=}cs(sort(\mathbb{T}[s..i]))$ を表す頂点 u が存在する場合，s は副位置になり，それ以外の場合は主位置になる．主位置になる場合は更新作業を以下のように行う．

$MTPH(\mathbb{T}[1..i-1])$ において新たに x を表す頂点 u を作成し，$mtspref_s$ を表す頂点 v から文字列 $x[|x|-N..|x|]$ で u につながるように辺と頂点を作成することで $MTPH(\mathbb{T}[1..i])$ を構築する．作業位置は u の接尾辞ポインタ $mtsp(u)$ を辿ることで新たに $s+1$ に変わる．$s+1$ が主位置になるか副位置になるかによって繰り返し更新を行う．

上記の更新作業で新たに深さが N の倍数の頂点 $u_1, u_2, ..., u_k$ がこの順序で作成され，作業位置 $s+k$ に対する $mtspref_{s+k}$ を表す頂点が v であるとき，接尾辞ポインタ $mtsp$ は $mtsp(u_i) = u_{i+1}(1 \leq i \leq k-1), mtsp(u_k) = v$ となる．このように作業位置の更新と接尾辞ポインタの構築を繰り返しながら $MTPH(\mathbb{T})$ を構築する．

1 回の更新で作業位置 s に対する $mtspref_s$ の長さを N 文字分短くする操作が何度か繰り返されているが，全体 n 回のループでは読み込んだ nN 文字しか $mtspref_s$ の長さを短くすることができないので，このアルゴリズムは $O(nN)$ 時間で終了する．

定理 8. *Algorithm 9* は，長さ n，トラック数 N のマルチトラック $\mathbb{T}$ に対するマルチトラックポジションヒープ $MTPH(\mathbb{T})$ を $O(nN)$ 時間で構築する．

6.4 縮約マルチトラックポジションヒープ

定義 22 (縮約マルチトラックポジションヒープ). マルチトラック $\mathbb{T}$ に対する**縮約マルチトラックポジションヒープ** $CMTPH(\mathbb{T})$ は，$\mathbb{T}$ の接尾辞のソート順列によって決定されるマルチトラックの連結文字列の順序集合に対するシーケンスハッシュ木である．

順序は構築アルゴリズムによって決定される．マルチトラックテキスト $(abababa, baabbab)$ に対する縮約マルチトラックポジションヒープの例を図 6.2 に示す．

6.5 縮約マルチトラックポジションヒープの線形時間構築アルゴリズム

長さ n，トラック数 N のマルチトラック $\mathbb{T}$ に対するマルチトラックポジションヒープ $MTPH(\mathbb{T})$ を入力として受け取り，$MTPH(\mathbb{T})$ を深さ優先探索で辿りながら構築する．辿

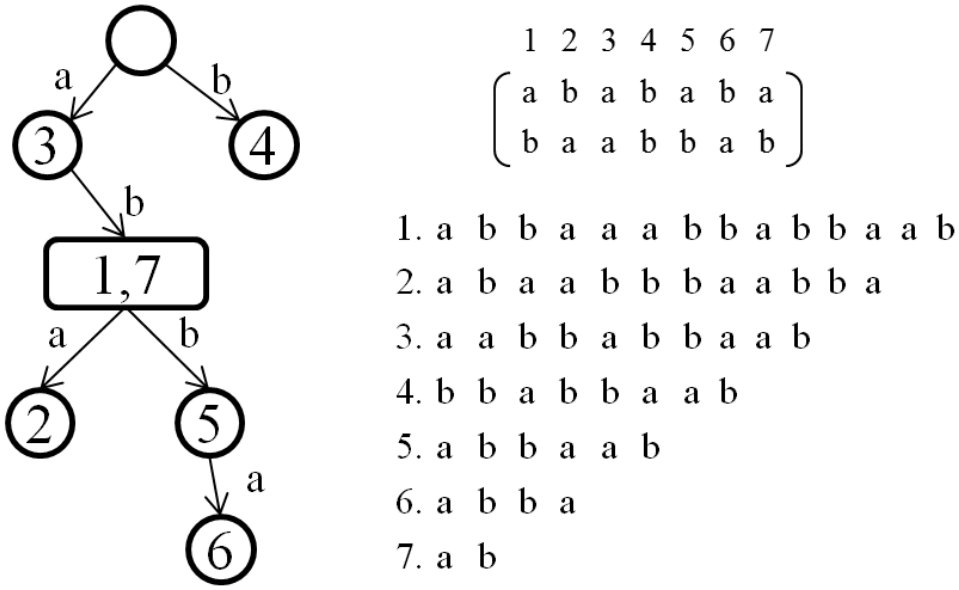


図 6.2: マルチトラックテキスト $(abababa, baabbab)$ に対する縮約マルチトラックポジションヒープ．この例はマルチトラックテキスト $(abababa, baabbab)$ に対するマルチトラックポジションヒープを，Σ の要素の辞書式順序で辿った際に構築できる縮約マルチトラックポジションヒープである．

る際にデータ構造としてデックを利用する．頂点 u からラベル c を辿り頂点 v に入る際にデックの後ろに辺情報 (u, c, v) を追加し，入った頂点の深さが正で N の倍数であれば $MTPH(\mathbb{T})$ の $mtspref_s$ $(1 \leq s \leq n)$ を表す頂点なので，デックの先頭から辺情報を取り出し，$MTPH(\mathbb{T})$ の $mtspref_s$ を表す頂点を $MTPH(\mathbb{T})$ の $mtspref_s$ の接頭辞を表す頂点として新たに $CMTPH(T)$ に追加する．次に頂点 v の子を再帰的に辿り，子に対しで上記の操作を繰り返す．再帰が終わり頂点から出る際にデックの後ろの辺情報 (u, c, v) がまだ取り出されていなければデックの後ろから辺情報を取り出す．アルゴリズムを Algorithm 11, 12 に示す．変数 map はマルチトラックポジションヒープの頂点と縮約マルチトラックポジションヒープの頂点の対応をとるための一時情報を格納するための配列である．

計算量は $MTPH(\mathbb{T})$ を深さ優先探索するので $MTPH(\mathbb{T})$ の頂点数と辺数の和である $O(nN)$ である．また，$CMTPH(\mathbb{T})$ は頂点の追加回数が $\mathbb{T}$ の接尾辞の個数である $O(n)$ であるため，領域は $MTPH(\mathbb{T})$ の $O(nN)$ よりも小さい $O(n)$ となる．

定理 9. 縮約マルチトラックポジションヒープ構築アルゴリズムは，トラック長 n，トラック数 N のマルチトラック $\mathbb{T}$ に対するマルチトラックポジションヒープ $MTPH(\mathbb{T})$ を

入力として受け取り，縮約マルチトラックポジションヒープ $CMTPH(\mathbb{T})$ を $O(nN)$ 時間で構築する．

6.6 マルチトラックポジションヒープを用いた素朴なパターン照合

トラック長 n，トラック数 N のテキスト $\mathbb{T}$ とマルチトラックポジションヒープ $MTPH(\mathbb{T})$，トラック長 m，トラック数 N のパターン $\mathbb{P}$ が与えられたとき，$\mathbb{T}$ の中に表れる $\mathbb{P}$ と順列一致する部分文字列の位置をすべて求めることを考える．はじめに $cs(sort(\mathbb{P}))$ を $O(mN)$ 時間で求める．次に，$MTPH(\mathbb{T})$ を $cs(sort(\mathbb{P}))$ で辿ることで最大 $m-1$ 個のパターンと順列一致している可能性のある位置が得られる．順列一致している可能性のある位置 i から m 文字の部分文字列 $\mathbb{T}[i..i+m]$ に対して $cs(sort(\mathbb{T}[i..i+m]))$ を $O(mN)$ 時間で求め，$cs(sort(\mathbb{P}))$ との照合を $O(mN)$ 時間で行う．加えて，$cs(sort(\mathbb{P}))$ が $MTPH(\mathbb{T})$ に含まれている場合に限り，到達した頂点とその子孫がパターンと順列一致している位置となる．合計で $MTPH(\mathbb{T})$ を用いて $O(m^2N + occ)$ 時間で順列パターン照合を行うことができる．

定理 10. マルチトラックテキスト $\mathbb{T}$ とマルチトラックポジションヒープ $MTPH(\mathbb{T})$，マルチトラックパターン $\mathbb{P}$ が与えられたとき，順列パターン照合問題を $O(m^2N + occ)$ 時間で解く．

6.7 縮約マルチトラックポジションヒープを用いた素朴なパターン照合

トラック長 n，トラック数 N のテキスト $\mathbb{T}$ と縮約マルチトラックポジションヒープ $CMTPH(\mathbb{T})$，トラック長 m，トラック数 N のパターン $\mathbb{P}$ が与えられたとき，$\mathbb{T}$ の中に表れる $\mathbb{P}$ と順列一致する部分文字列の位置をすべて求めることを考える．はじめに $cs(sort(\mathbb{P}))$ を $O(mN)$ 時間で求める．次に，$CMTPH(\mathbb{T})$ を $cs(sort(\mathbb{P}))$ で辿ることで最大 $mN-1$ 個のパターンと順列一致している可能性のある位置が得られる．順列一致している可能性のある位置 i から m 文字の部分文字列 $\mathbb{T}[i..i+m]$ に対して $cs(sort(\mathbb{T}[i..i+m]))$ を $O(mN)$ 時間で求め，$cs(sort(\mathbb{P}))$ との照合を $O(mN)$ 時間で行う．加えて，$cs(sort(\mathbb{P}))$ が $CMTPH(\mathbb{T})$ に含まれている場合に限り，到達した頂点とその子孫がパターンと順列一致している位置となる．合計で $CMTPH(\mathbb{T})$ を用いて $O(m^2N^2 + occ)$ 時間で順列パターン照合問題を

解くことができる．

定理 11. マルチトラックテキスト $\mathbb{T}$ と縮約マルチトラックポジションヒープ $CMTPH(\mathbb{T})$，マルチトラックパターン $\mathbb{P}$ が与えられたとき，順列パターン照合問題を $O(m^2N^2 + occ)$ 時間で解く．

6.8 確率的パターン照合アルゴリズム

素朴なパターン照合アルゴリズムにおいて問題となっている点は，パターンの各候補位置に対して，検証を行うために $O(mN)$ 時間必要になっているところである．通常のポジションヒープでは，極大到達ポインタを付与することで，全体の照合を高速化していたが，マルチトラックポジションヒープ/縮約マルチトラックポジションヒープでは，パターンのソート順列によってきまるマルチトラックの接尾辞の連結文字列が，マルチトラックポジションヒープ・縮約マルチトラックポジションヒープに含まれているとは限らないため，極大到達ポインタを用いても照合を高速化することはできない．ここでは，前処理として $O(nN)$ 時間で $O(nN)$ 領域のハッシュ配列を新たに構築し，各パターンの候補に対して確率的に $O(N)$ 時間で候補の検証を行う．

マルチトラックテキスト $\mathbb{T} = (t_1, t_2, ..., t_N)$ の各 t_i に対して，通常のポジションヒープの確率的パターン照合アルゴリズムで用いたハッシュ配列 $HA^i_{\alpha,\beta}$ $(1 \leq i \leq N)$ を構築する．パターン $\mathbb{P} = (p_1, p_2, ..., p_N)$ が与えられた際にパターンのハッシュ値を求める．$O(mN)$ 時間で $cs(sort(\mathbb{P}))$ を求めた後に，マルチトラックポジションヒープ/縮約マルチトラックポジションヒープを辿りパターンの候補を求める．ここでパターンの候補に対して，ハッシュ値の比較を行うが，比較のために単純に昇順に並べ替えると $O(NlogN)$ 時間かかってしまう．そこで，構築時に利用した $\mathbb{T}$ に対するソート順列を覚えておくことで，ハッシュ値の比較におけるハッシュ値の昇順並べ替えを省略する．よって，$O(N)$ 時間で一つのパターンの候補に対してパターンの検証が確率的に行える．全体では，マルチトラックポジションヒープを用いたパターン照合では $O(mN + occ)$ 時間，縮約マルチトラックポジションヒープを用いたパターン照合では $O(mN^2 + occ)$ 時間でパターン照合問題を解くことができる．また，ソート順列とハッシュ配列を記憶するために，縮約マルチトラックポジションヒープを用いた索引の領域も $O(nN)$ となる．ハッシュ値の比較アルゴ

リズムを Algorithm 13 に示す.

N トラック中 K トラックが異なると仮定すると，Algorithm 13 が偽陽性の出力を行う確率は，K 回文字列が異なるときにハッシュ値が同じになる必要があるので，K 回連続で異なる文字列が同じハッシュ値になる確率

$$\left(\frac{1}{\beta}\right)^K$$

となる．一様に $1 \leq K \leq N$ トラックが異なると仮定すると，Algorithm 13 が間違える確率はすべての組み合わせをとることで,

$$\frac{\sum_{K=1}^{N} \binom{N}{K} \left(\frac{1}{\beta}\right)^K}{2^N - 1} = \frac{\left(1 + \frac{1}{\beta}\right)^N - 1}{2^N - 1}$$

となる．マルチトラックポジションヒープを用いた場合は高々$m-1$ 回，縮約マルチトラックポジションヒープを用いた場合は高々$mN-1$ 回 Algorithm 13 でハッシュ値の比較を行うので，それぞれ確率 $\left(1 - \frac{\left(1+\frac{1}{\beta}\right)^N - 1}{2^N - 1}\right)^{m-1}$，$\left(1 - \frac{\left(1+\frac{1}{\beta}\right)^N - 1}{2^N - 1}\right)^{mN-1}$ で順列パターン照合問題を $O(mN + occ)$ 時間，$O(mN^2 + occ)$ 時間で解くことができる.

定理 12. マルチトラックポジションヒープを用いた確率的パターン照合アルゴリズムは，トラック長 n，トラック数 N のマルチトラックテキスト $\mathbb{T} = (t_1, t_2, ..., t_N)$ とマルチトラックポジションヒープ $MTPH(\mathbb{T})$，各トラック t_i に対するハッシュ配列 $HA^i_{\alpha,\beta}$ $(1 \leq i \leq N)$，逆元行列 $INV_{\alpha,\beta}$，テキストに対するソート順列 $SI(\mathbb{T})$，トラック長 m，トラック数 N のマルチトラックパターン $\mathbb{P}$ が与えられたとき，確率 $\left(1 - \frac{\left(1+\frac{1}{\beta}\right)^N - 1}{2^N - 1}\right)^{m-1}$ で順列パターン照合問題を $O(mN + occ)$ 時間で解く.

定理 13. 縮約マルチトラックポジションヒープを用いた確率的パターン照合アルゴリズムは，トラック長 n，トラック数 N のマルチトラックテキスト $\mathbb{T} = (t_1, t_2, ..., t_N)$ と縮約マルチトラックポジションヒープ $CMTPH(\mathbb{T})$，各トラック t_i に対するハッシュ配列 $HA^i_{\alpha,\beta}$ $(1 \leq i \leq N)$，逆元行列 $INV_{\alpha,\beta}$，テキストに対するソート順列 $SI(\mathbb{T})$，トラック長 m，トラック数 N のマルチトラックパターン $\mathbb{P}$ が与えられたとき，確率 $\left(1 - \frac{\left(1+\frac{1}{\beta}\right)^N - 1}{2^N - 1}\right)^{mN-1}$ で順列パターン照合問題を $O(mN^2 + occ)$ 時間で解く.

Algorithm 9: マルチトラックポジションヒープ構築アルゴリズム

Input: 長さ n, トラック数 N のマルチトラック $\mathbb{T}$
Output: マルチトラックポジションヒープ $MTPH(\mathbb{T})$

compute $SI(\mathbb{T}[i..n])$ for all $1 \leq i \leq n$;
create $root$ and $\perp_N$ nodes;
for each character c, create edge $(\perp_N, c, root)$;
for $N-1$ **to** 1 **do**
    create $\perp_j$ node;
    for each character c, create edge $(\perp_j, c, \perp_{j+1})$;
$mtsp(root) = \perp_1$, $currentNode = root$, $s = 1$;
for $i = 1$ **to** n **do**
    $lastCreateNode =$ undefined;
    while true **do**
        $str = cs(\mathbb{T}\langle SI(\mathbb{T}[s..n])\rangle[i])$;
        if $mtnext(currentNode, str) \neq$ undefined **then**
            break;
        for $j = 1$ **to** N **do**
            if not exist edge $(currentNode, str[j], v)$ **then**
                if $j = N$ **then**
                    create node $mtspref_s$;
                    create edge $(currentNode, str[j], mtspref_s)$;
                    $currentNode = mtspref_s$;
                else
                    create node u;
                    create edge $(currentNode, str[j], u)$;
                    $currentNode = u$;
            else
                $currentNode = v$;
        $currentNode = mtnext(currentNode, str)$;
        if $lastCreateNode \neq$ undefined **then**
            $mtsp(lastCreateNode) = mtspref_s$;
        $lastCreateNode = mtspref_s$;
        $currentNode = mtsp(currentNode)$;
        $s = s + 1$;
    $currentNode = mtnext(currentNode, cs(\mathbb{T}\langle SI(\mathbb{T}[s..n])\rangle[i]))$;
    if $lastCreateNode \neq$ undefined **then**
        $mtsp(lastCreateNode) = currentNode$;

Algorithm 10: $mtnext(node, str)$ 関数

```
for i = 1 to N do
    if exist edge (node, str[i], v) then node = v;
    else return undefined;
return node;
```

Algorithm 11: 縮約マルチトラックポジションヒープ構築アルゴリズム

Input: $MTPH(\mathbb{T}), N = h(\mathbb{T})$

```
create root of CMTPH(T);
map[root of MTPH(T)] = root of CMTPH(T);
contract(root of MTPH(T), 0, empty deque);
```

Algorithm 12: *contract* 関数

Input: 着目している $MTPH(\mathbb{T})$ の頂点 $node$, 深さ $depth$, 辺情報を格納するためのデック $deque$

```
if depth > 0 and (depth mod N) = 0 then
    pop element (from, label, to) from front of deque;
    create new node v of CMTPH(T);
    create edge (map[from], label, v);
    map[to] = v;
foreach outgoing edge (node, label, child) from node do
    push element (node, label, child) to back of deque;
    contract(child, depth + 1, deque);
    if element of back of deque is (node,label,child) then
        pop element (node, label, child) from back of deque;
```

Algorithm 13: ハッシュ配列とソート順列を用いたハッシュ値の比較

Input: テキスト $\mathbb{T} = (t_1, t_2, ..., t_N)$ の各トラック t_i に対するハッシュ配列 $HA^i_{\alpha,\beta}$ $(1 \leq i \leq N)$，候補となる部分文字列の始点 st，パターン $\mathbb{P} = (p_1, p_2, ..., p_N)$ の各トラック p_i のハッシュ値 $pHash^i$ $(1 \leq i \leq N)$，パターンのトラック長 m，テキストの接尾辞 $\mathbb{T}[st..n]$ に対するソート順列 tSI，パターンに対するソート順列 pSI，逆元配列 $INV_{\alpha,\beta}$

Output: テキストの始点から m 文字の部分文字列に対するハッシュ値がパターンのハッシュ値と一致しているか否か

for $i = 1$ **to** N **do**
  $tHash = (HA^{tSI[i]}_{\alpha,\beta}[st + m - 1] - HA^{tSI[i]}_{\alpha,\beta}[st - 1]) \cdot INV_{\alpha,\beta}[st - 1] \mod \beta$;
  if $tHash \neq pHash^{pSI[i]}$ **then**
    return false;
return true;

第7章

実験

ここでは，通常のパターン照合問題とパラメタ化パターン照合問題，順列パターン照合問題に対して，それぞれポジションヒープ，パラメタ化ポジションヒープ，マルチトラックポジションヒープ/縮約マルチトラックポジションヒープを構築した際の実行時間と，それぞれのポジションヒープを用いたパターン照合を行った際の実行時間を計測した．ハッシュ値を求める際の α と β の値は Gonnet らが [10] にて効果的な実装として提案している値である $\alpha =$ 素数 (101)，$\beta = 2^{64}$ として実験を行った．このパラメータを用いた実験中に偽陽性の出力は起こらなかった．計測環境は CPU:Intel(R) Core(TM) i5 CPU M430(2.27Ghz)，メモリ:2.92GB，OS:Windows7 32bit である．また，パターン照合問題におけるパターンの出現数はパターン長 (マルチトラックの場合はトラック長) の 0.5 から 1 倍になるように設定した．

7.1 ポジションヒープ

7.1.1 構築時間

ポジションヒープの構築時間を比較した．各 10 回の実行時間の平均をとった．実験結果を図 7.1, 7.2, 7.3 に示す．図中の PH がポジションヒープの構築時間，PH+MRP が極大到達ポインタを用いた照合を行うためのデータ構造を構築する時間を含めたポジションヒープの構築時間，PH+ハッシュがハッシュ配列を用いた照合を行うためのデータ構造を構築する時間を含めたポジションヒープの構築時間である．構築時間は入力サイズに対して線形に増加していることがわかる．また，ハッシュ配列を構築する時間はポジ

ションヒープを構築する時間に比べて非常に小さくなっていることが分かる．また，極大到達ポインタと子孫判定を行うデータ構造を構築する時間よりもハッシュ配列を構築する時間のほうが短くなっている．

7.1.2 照合時間

ポジションヒープを用いた照合時間を比較した．各 400 回の実行時間の平均をとった．実験結果を図 7.4, 7.5, 7.6 に示す．図中の MRP と素朴が既存研究である極大到達ポインタを用いた線形時間パターン照合アルゴリズムと二乗時間パターン照合アルゴリズムであり，図中のハッシュが提案手法であるハッシュ配列を用いた確率的線形時間パターン照合アルゴリズムの照合時間である．ハッシュ配列を用いた手法は極大到達ポインタを用いた手法と素朴な手法に比べて高速に動作する．

7.2 パラメタ化ポジションヒープ

7.2.1 構築時間

パラメタ化ポジションヒープの線形時間構築アルゴリズムの実行時間を計測した．各 15 回の実行時間の平均をとった．実験結果を図 7.7, 7.8, 7.9, 7.10, 7.11 に示す．図中の PPH がパラメタ化ポジションヒープの構築時間を，PPH+MRP がパラメタ化ポジションヒープに加えてパラメタ化極大到達ポインタと子孫判定のためのデータ構造の構築時間を加えた際の実行時間を，PPH+ハッシュがパラメタ化ポジションヒープに加えてハッシュ配列の構築時間を加えた際の実行時間を表す．テキスト長に対して実行時間が線形で増加していることがわかる．

7.2.2 照合時間

パラメタ化ポジションヒープを用いたパターン照合アルゴリズムの実行時間を計測した．各 400 回の実行時間の平均をとった．実験結果を図 7.12, 7.13, 7.14, 7.15, 7.16 に示す．図中の素朴がパラメタ化ポジションヒープのみを用いたパターン長の二乗時間アルゴリズムの実行時間を，MRP がパラメタ化極大到達ポインタを用いたパターン照合アル

ゴリズムの実行時間を，ハッシュがパラメタ化ポジションヒープに加えハッシュ配列を用いた確率的パターン照合アルゴリズムの実行時間を表す．パラメタ化極大到達ポインタを用いた照合アルゴリズムと確率的パターン照合アルゴリズムは理論的なオーダの通り $|\Pi|$ が増加するにしたがって実行時間が増加していることがわかる．

7.3 マルチトラックポジションヒープ・縮約マルチトラックポジションヒープ

7.3.1 構築時間

マルチトラックポジションヒープの構築アルゴリズムの実行時間を計測した．各 10 回の実行時間の平均をとった．Σ 上のマルチトラックに対してトラック長 n とトラック数 N，Σ の大きさを変化させた際の実行時間を図 7.17, 7.18, 7.19 に示す．トラック長とトラック数の線形で実行時間が増加していることがわかる．

ハッシュ配列を構築する時間を計測した．各 10 回の実行時間の平均をとった．構築した時の実行時間をトラック長 n とトラック数 N，Σ の大きさを変化させ計測した結果を図 7.20, 7.21, 7.22 に示す．トラック長 n とトラック数 N の線形で実行時間が増加していることがわかる．

マルチトラックポジションヒープから縮約マルチトラックポジションヒープを構築するアルゴリズムの実行時間を計測した．各 10 回の実行時間の平均をとった．$|\Sigma|$ 上のマルチトラック $\mathbb{T}$ に対して構築した $MTPH(\mathbb{T})$ から $CMTPH(\mathbb{T})$ を構築した時の実行時間をトラック長 n とトラック数 N，Σ の大きさを変化させ計測した結果を図 7.23, 7.24, 7.25 に示す．トラック長 n とトラック数 N の線形で実行時間が増加していることがわかる．

7.3.2 照合時間

マルチトラックポジションヒープ・縮約マルチトラックポジションヒープを用いた照合時間を比較した．各 300 回の実行時間の平均をとった．パターン長 m を変化させた場合の実験結果を図 7.26, 7.27, 7.28, 7.29 に，トラック数 N を変化させた場合の実験結果を図 7.30, 7.31 に示す．図中の MTPH がマルチトラックポジションヒープを用いた素朴な

パターン照合アルゴリズム，MTPH+ハッシュがマルチトラックポジションヒープとハッシュ配列を用いたパターン照合アルゴリズム，CMTPH が縮約マルチトラックポジションヒープを用いた素朴なパターン照合アルゴリズム，CMTPH+ハッシュが縮約マルチトラックポジションヒープとハッシュ配列を用いたパターン照合アルゴリズムの照合時間である．ハッシュ配列を用いた手法は，用いない手法に比べて高速に動作する．図 7.29 において縮約マルチトラックポジションヒープとハッシュ配列を用いた手法が最も高速に動作しているが，これは索引のサイズが小さいためキャッシュのヒット率がほかの手法に比べて良好なためであると考えられる．トラック数を増やして索引のサイズが大きくなった場合は図 7.31 の通り理論的なオーダに従う時間でアルゴリズムが動作していることがわかる．

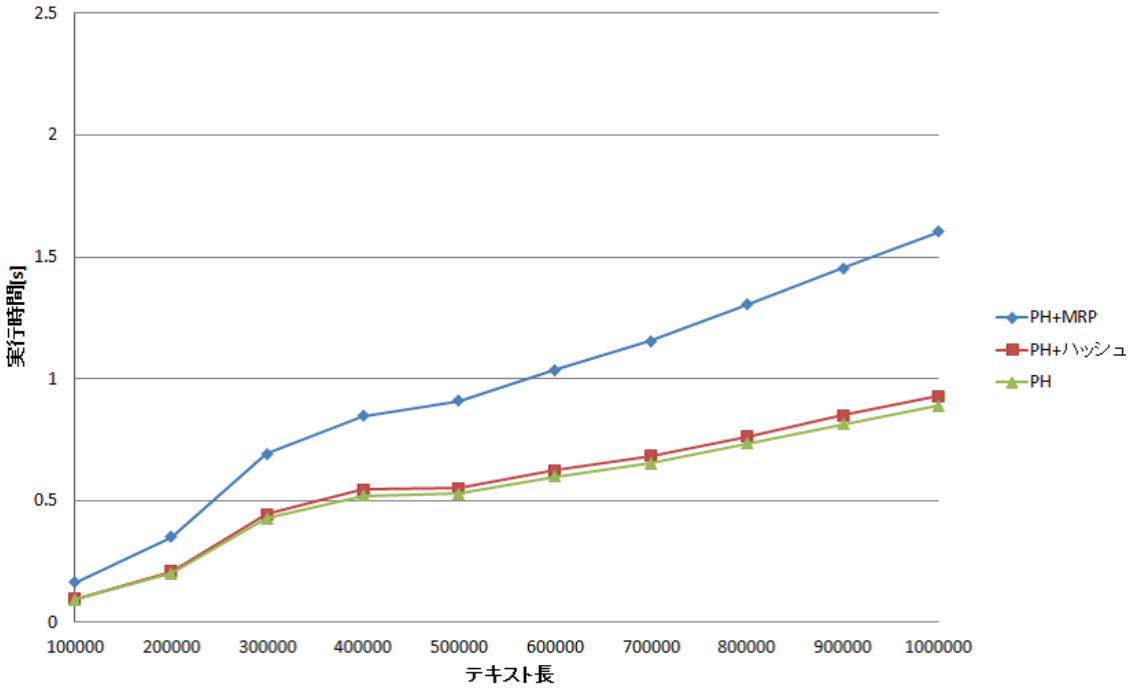


図 7.1: PH の構築時間 $|\Sigma| = 2$

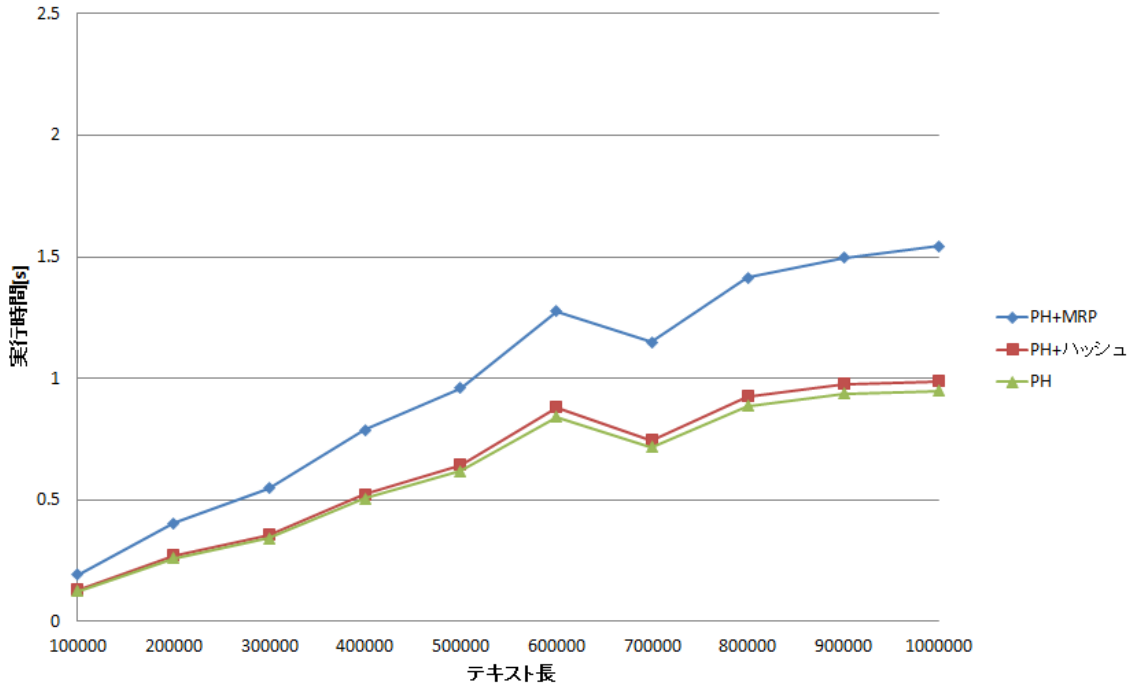


図 7.2: PH の構築時間 $|\Sigma| = 4$

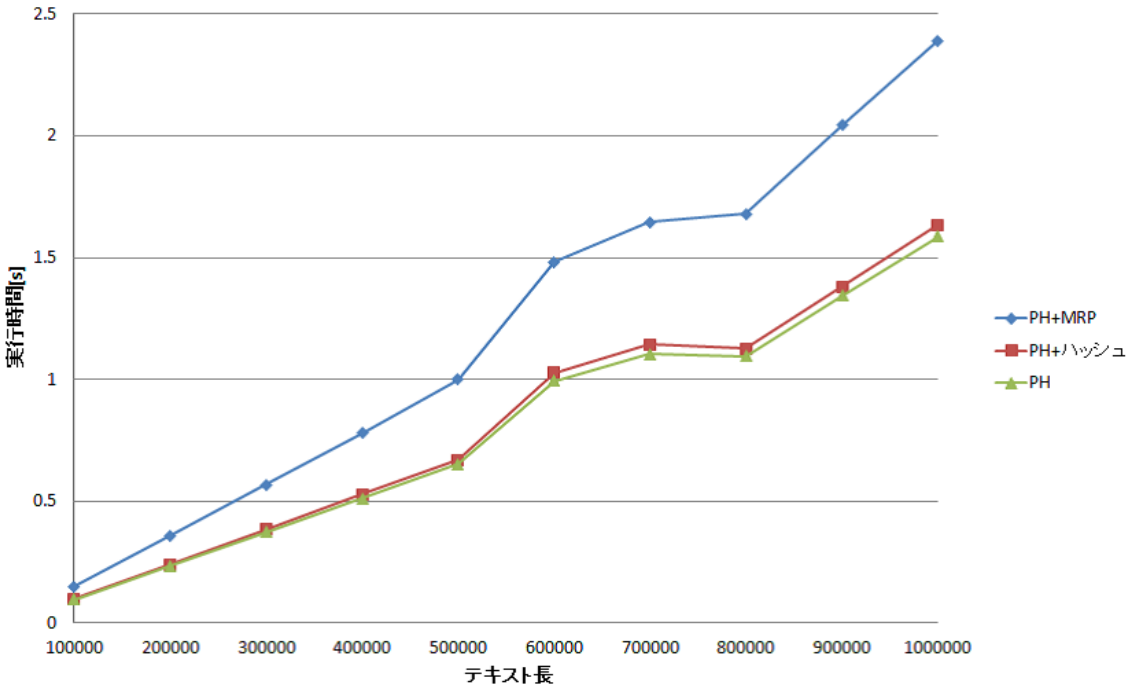


図 7.3: PH の構築時間 $|\Sigma| = 10$

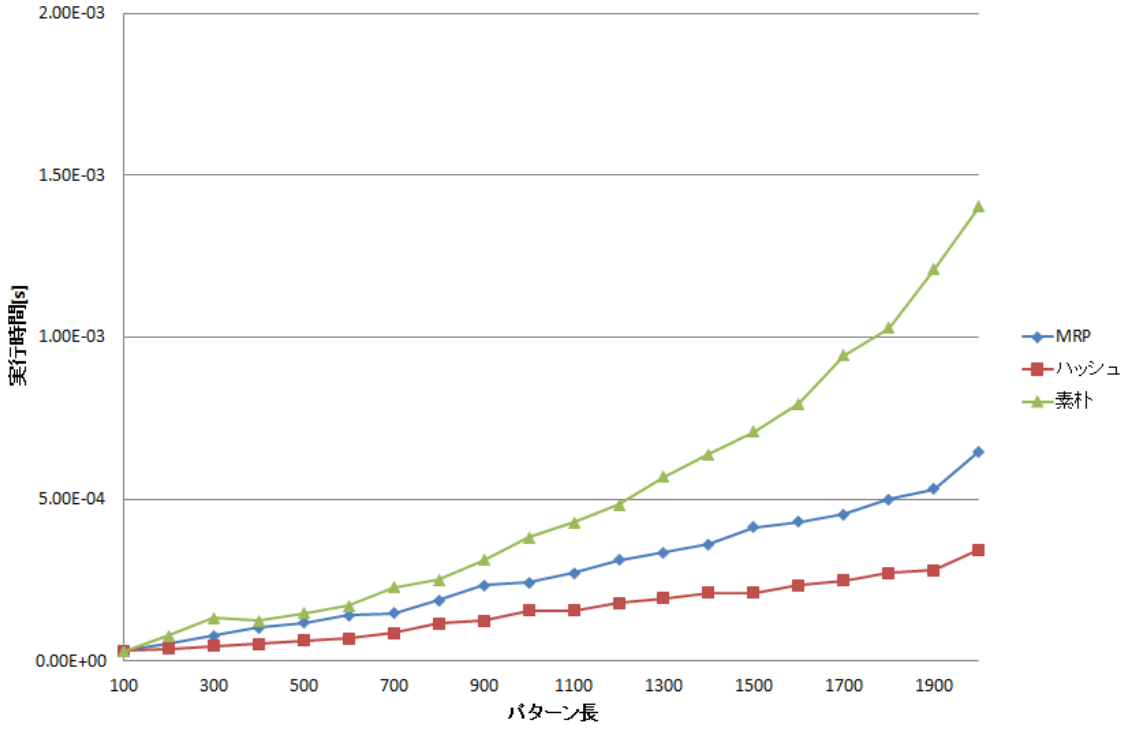


図 7.4: PH を用いたパターン照合時間 $|\Sigma| = 3$

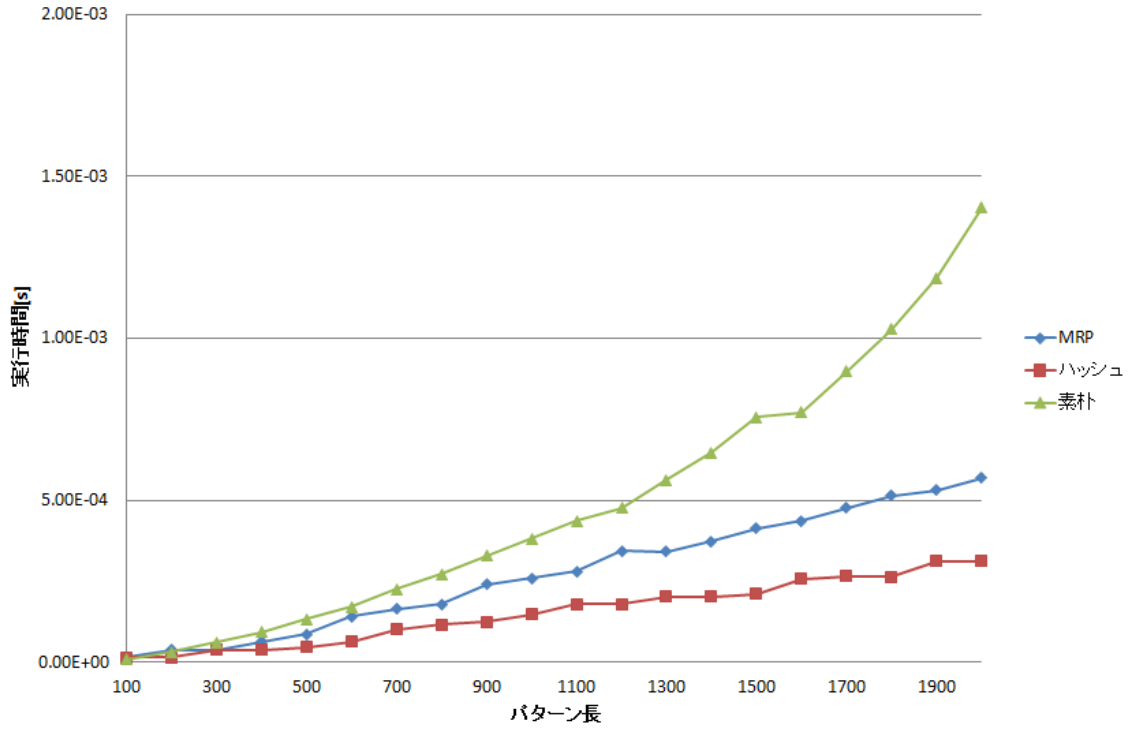


図 7.5: PH を用いたパターン照合時間 $|\Sigma| = 5$

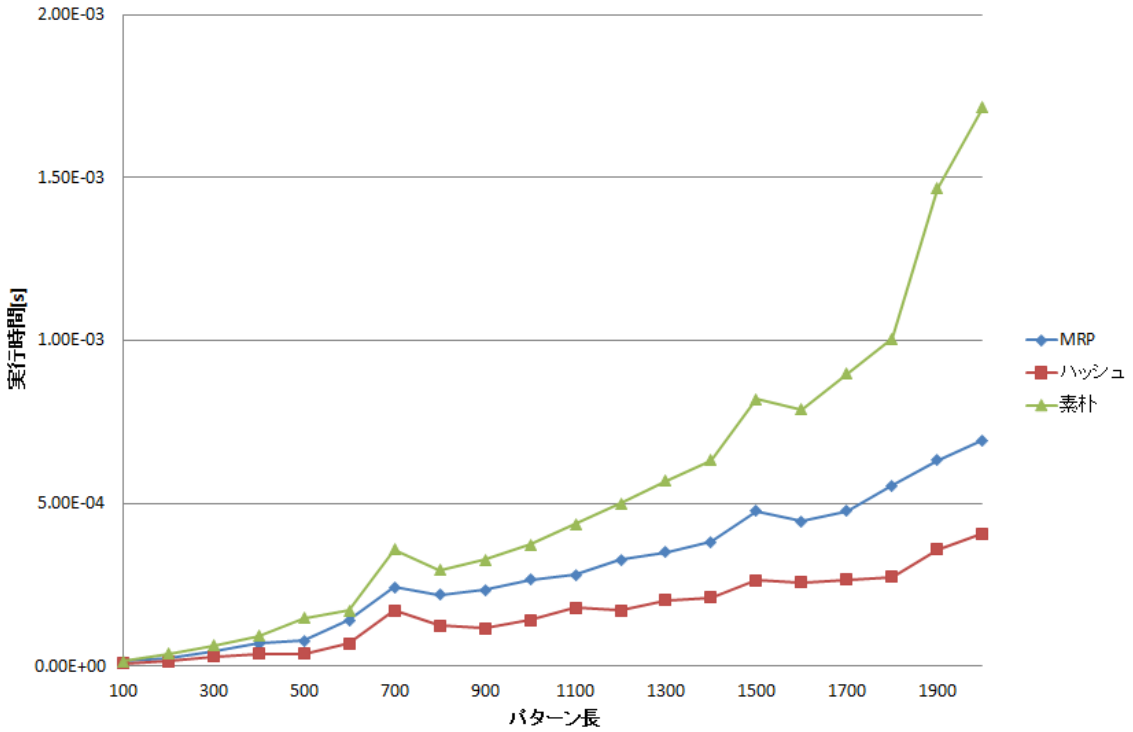


図 7.6: PH を用いたパターン照合時間 $|\Sigma| = 11$

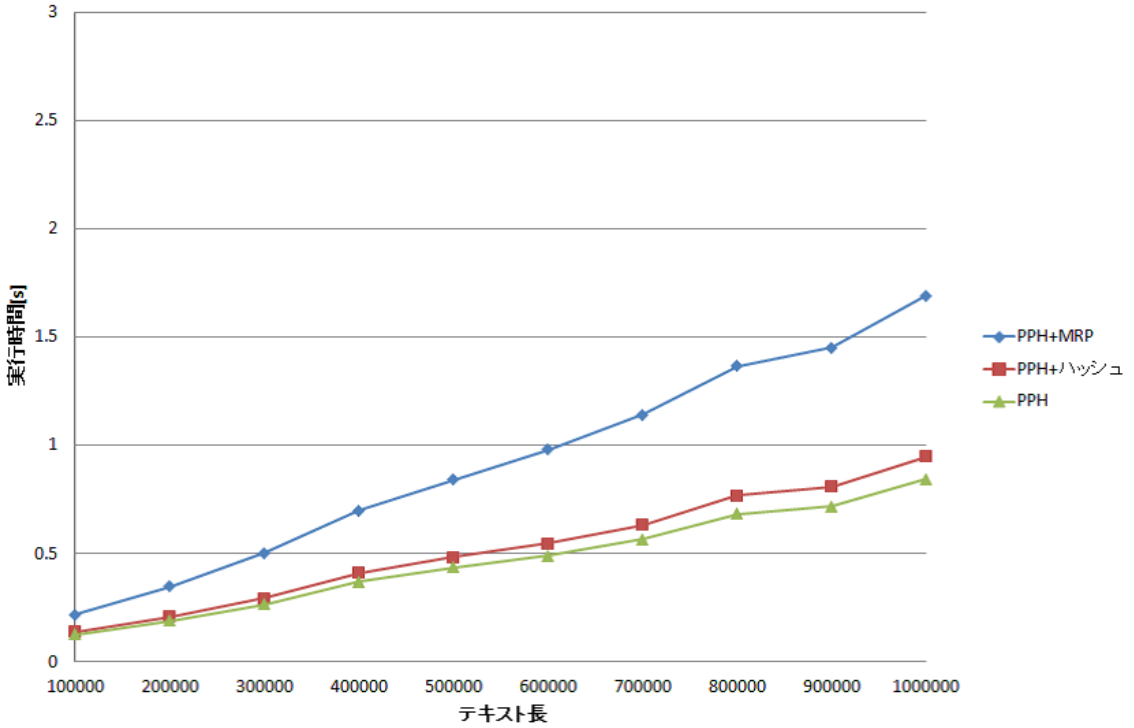


図 7.7: PPH の構築時間 $|\Sigma| = 2, |\Pi| = 2$

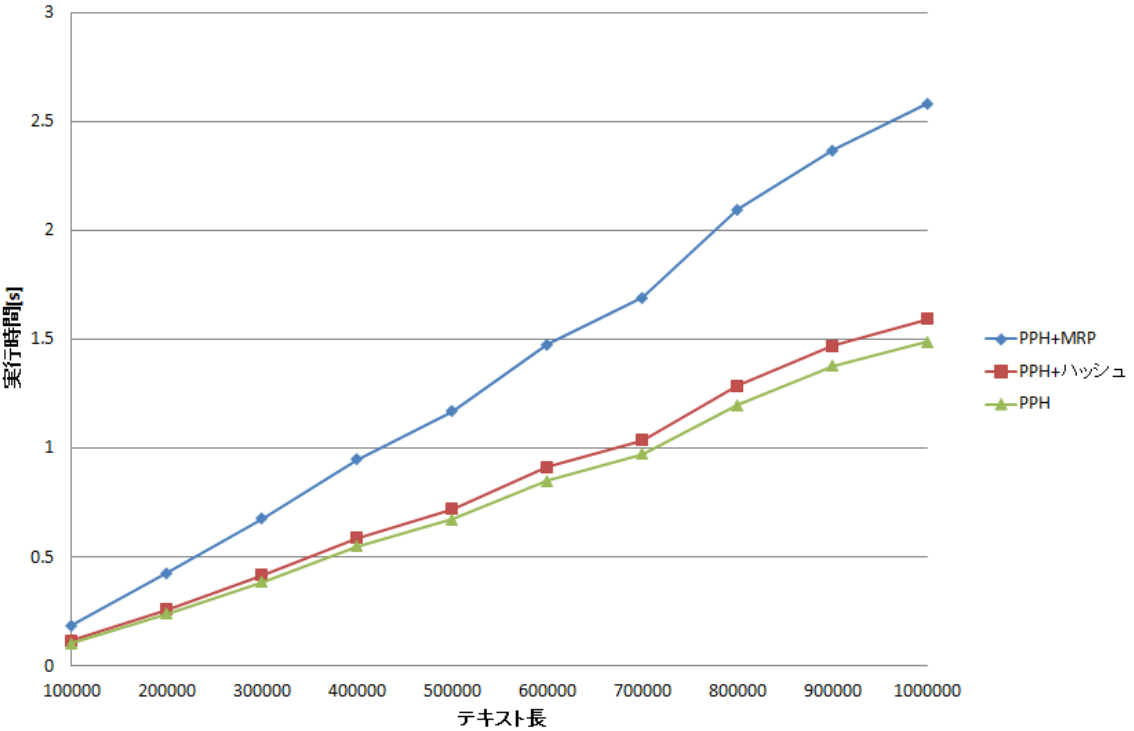


図 7.8: PPH の構築時間 $|\Sigma| = 50, |\Pi| = 2$

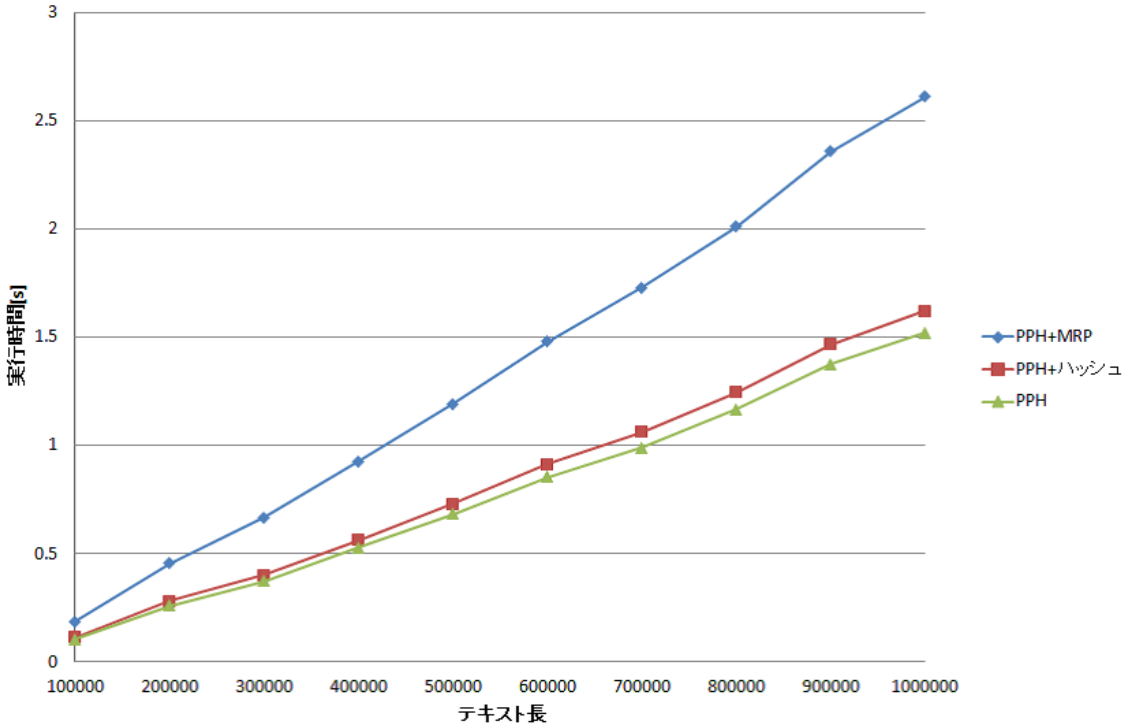


図 7.9: PPH の構築時間 $|\Sigma| = 100, |\Pi| = 2$

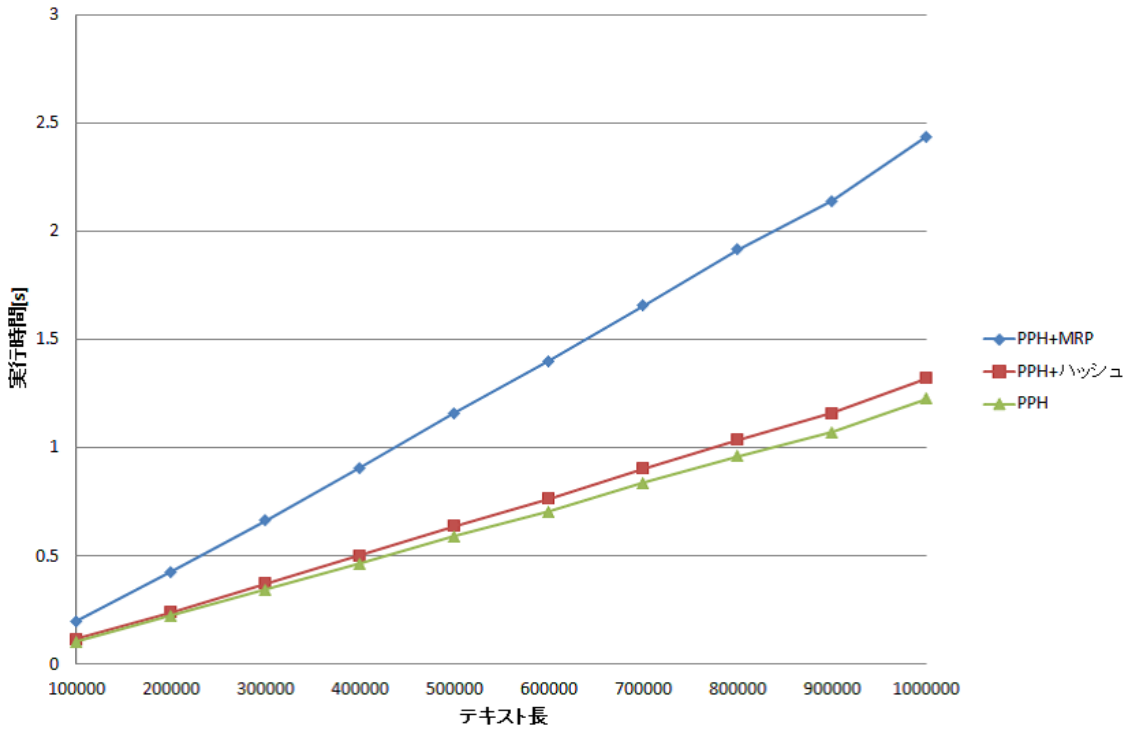


図 7.10: PPH の構築時間 $|\Sigma| = 2, |\Pi| = 50$

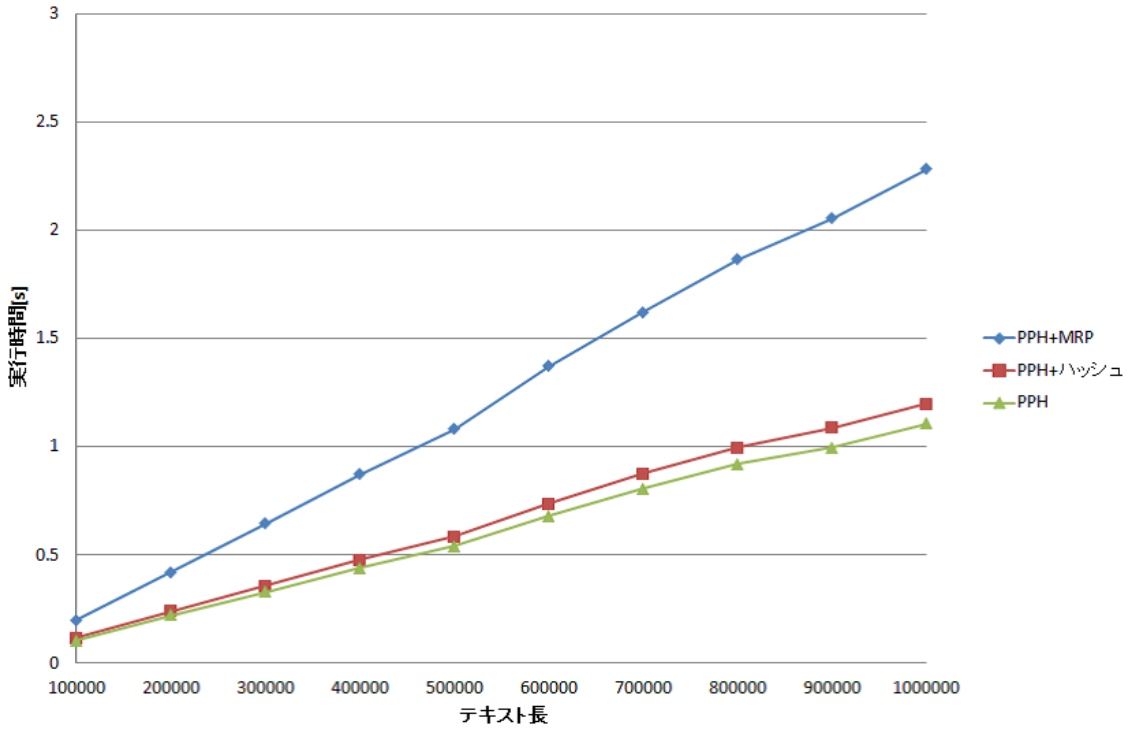


図 7.11: PPH の構築時間 $|\Sigma| = 2, |\Pi| = 100$

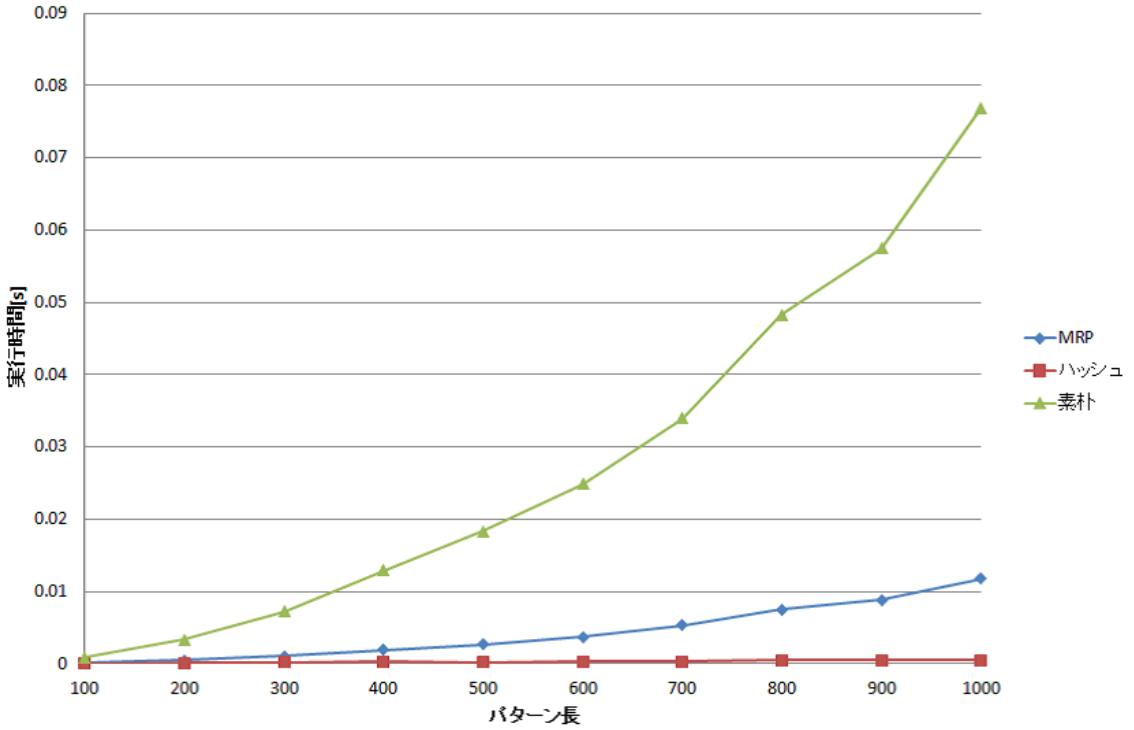


図 7.12: PPH を用いた照合時間 $|\Sigma| = 2, |\Pi| = 2$

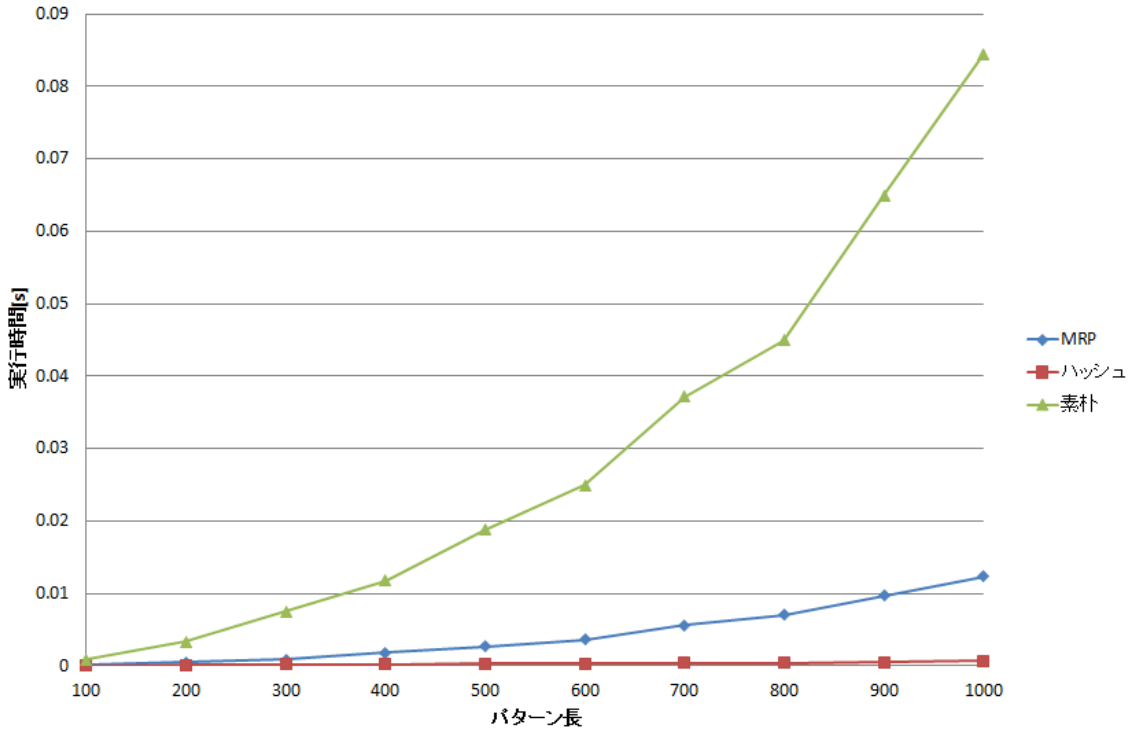


図 7.13: PPH を用いた照合時間 $|\Sigma| = 50, |\Pi| = 2$

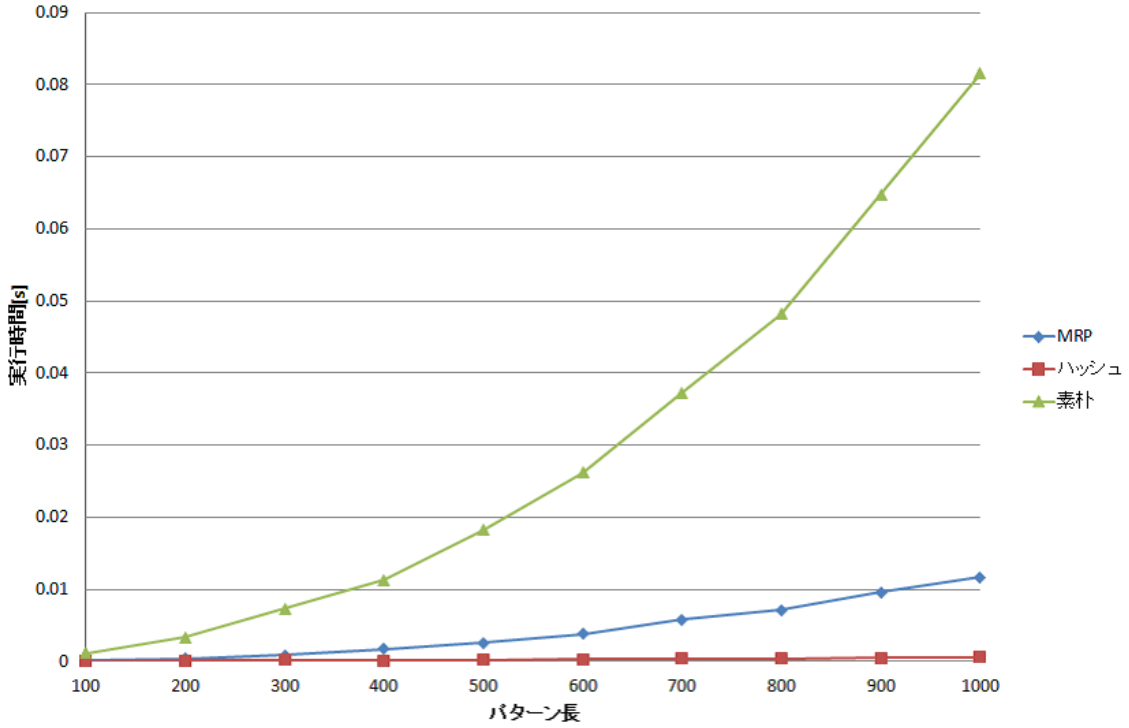


図 7.14: PPH を用いた照合時間 $|\Sigma| = 100, |\Pi| = 2$

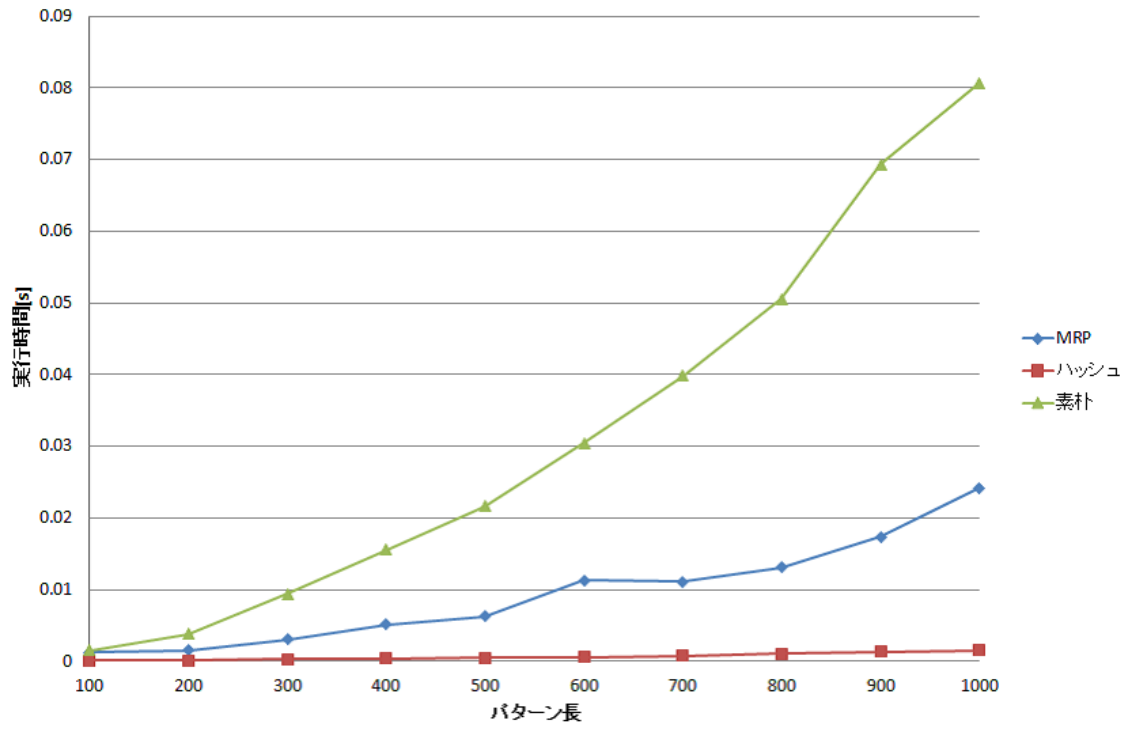


図 7.15: PPH を用いた照合時間 $|\Sigma| = 2, |\Pi| = 50$

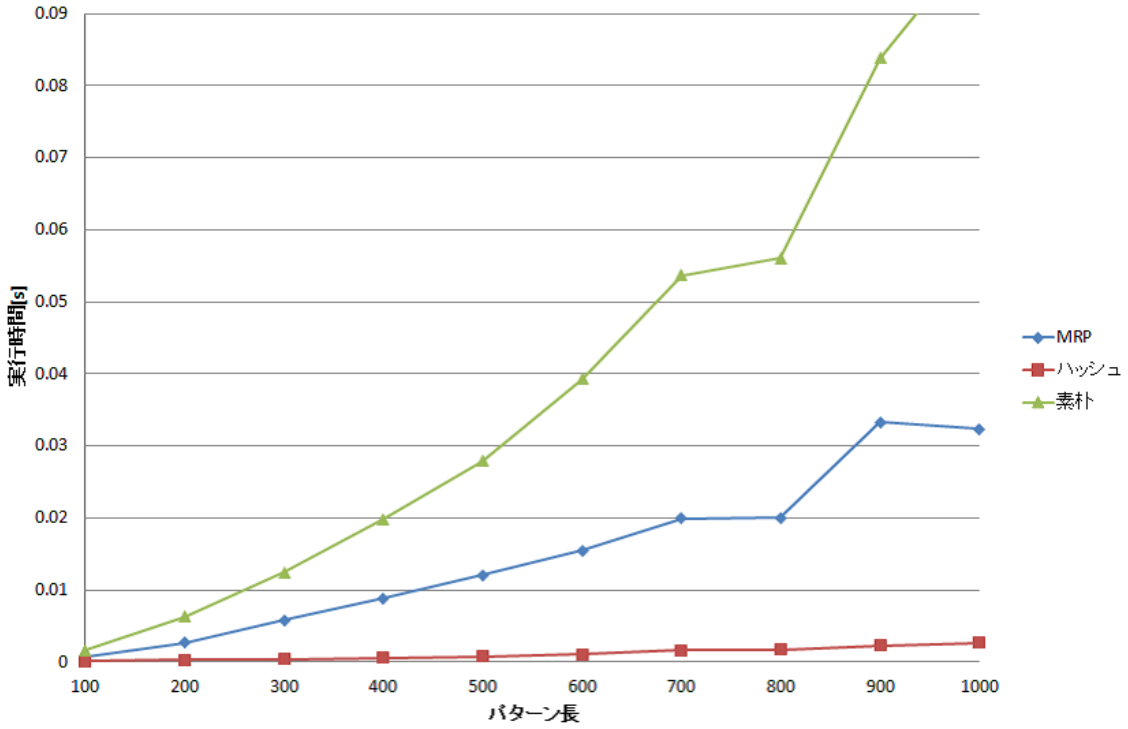


図 7.16: PPH を用いた照合時間 $|\Sigma| = 2, |\Pi| = 100$

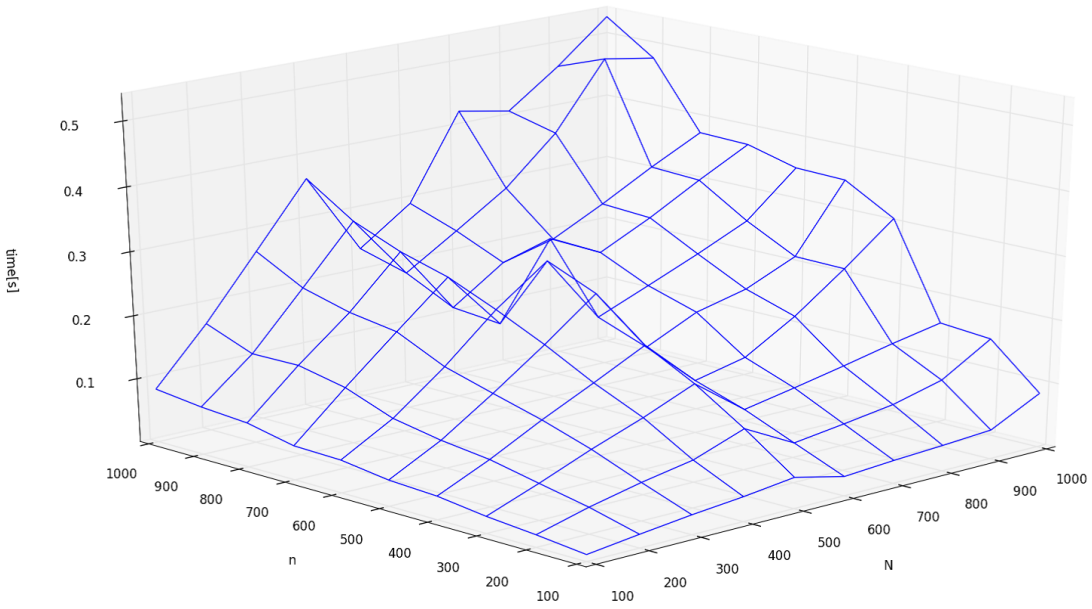


図 7.17: MTPH の構築時間 $|\Sigma| = 2$

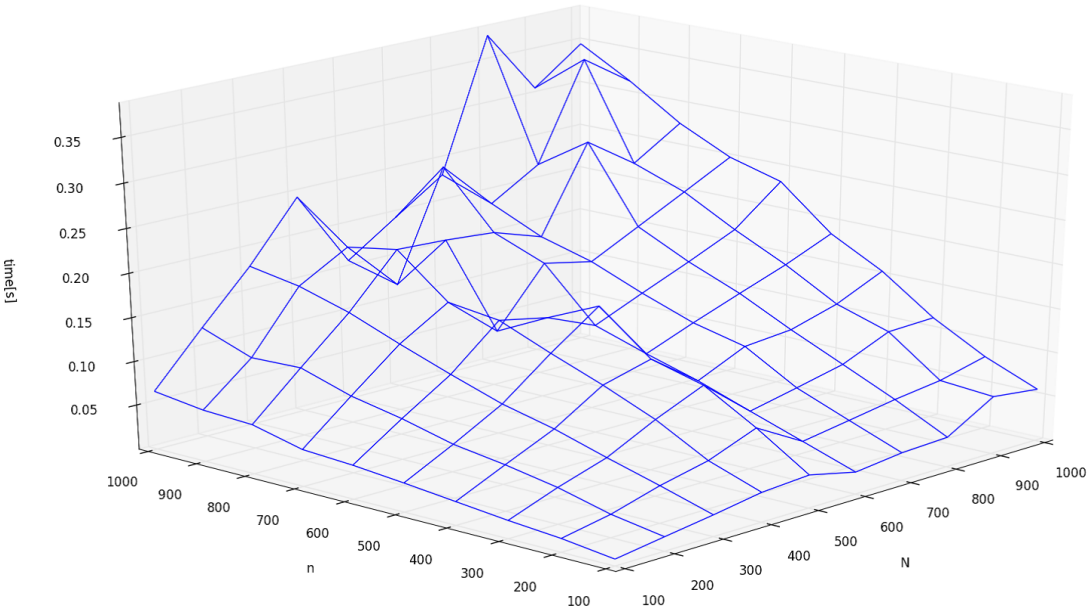


図 7.18: MTPH の構築時間 $|\Sigma| = 4$

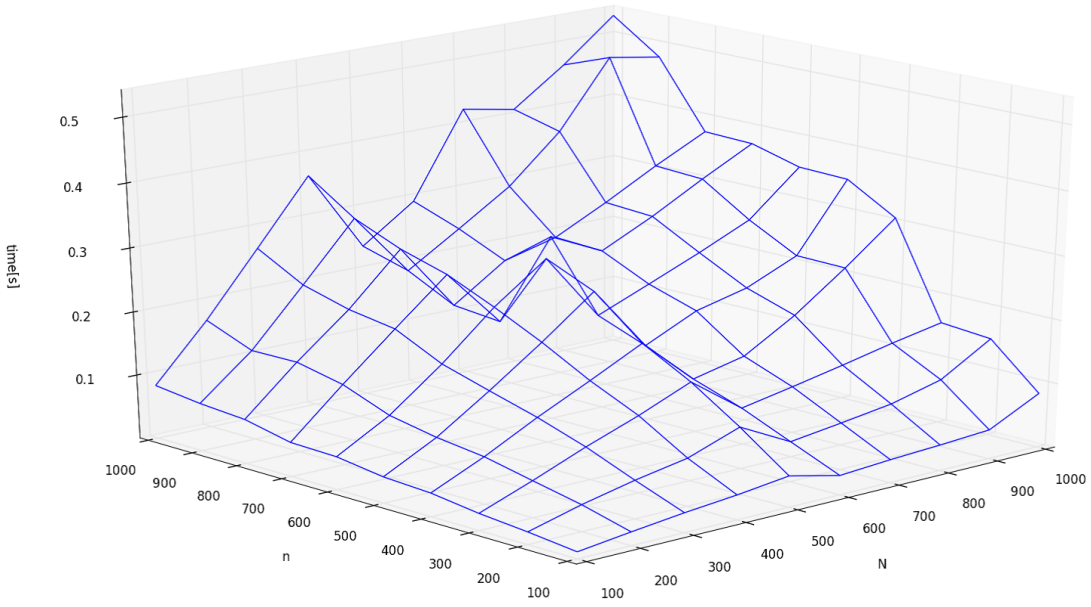


図 7.19: MTPH の構築時間 $|\Sigma| = 10$

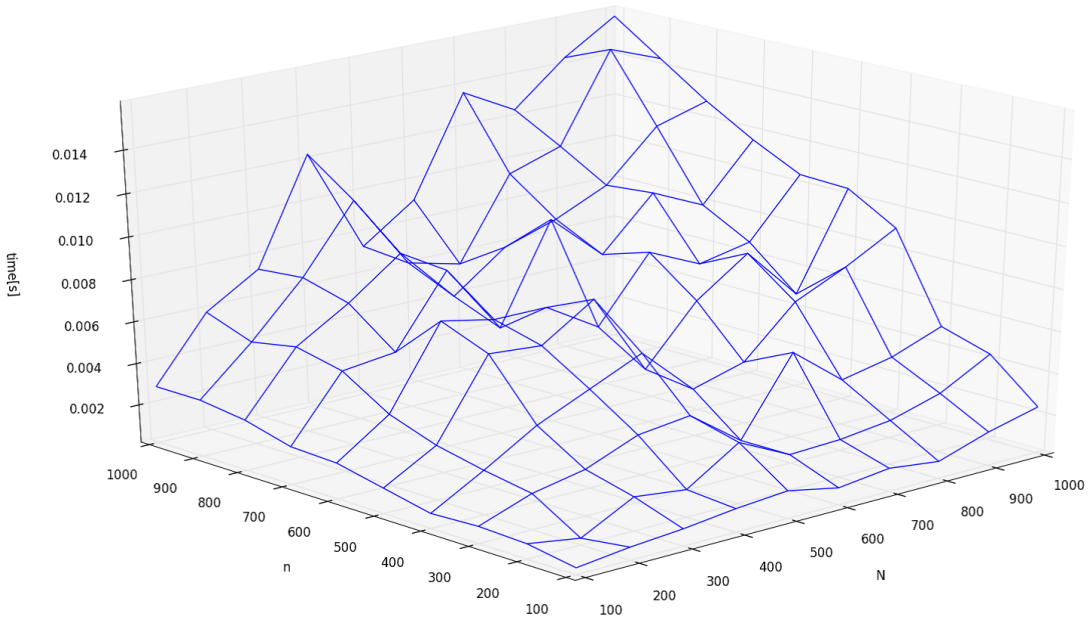


図 7.20: ハッシュ配列の構築時間 $|\Sigma| = 2$

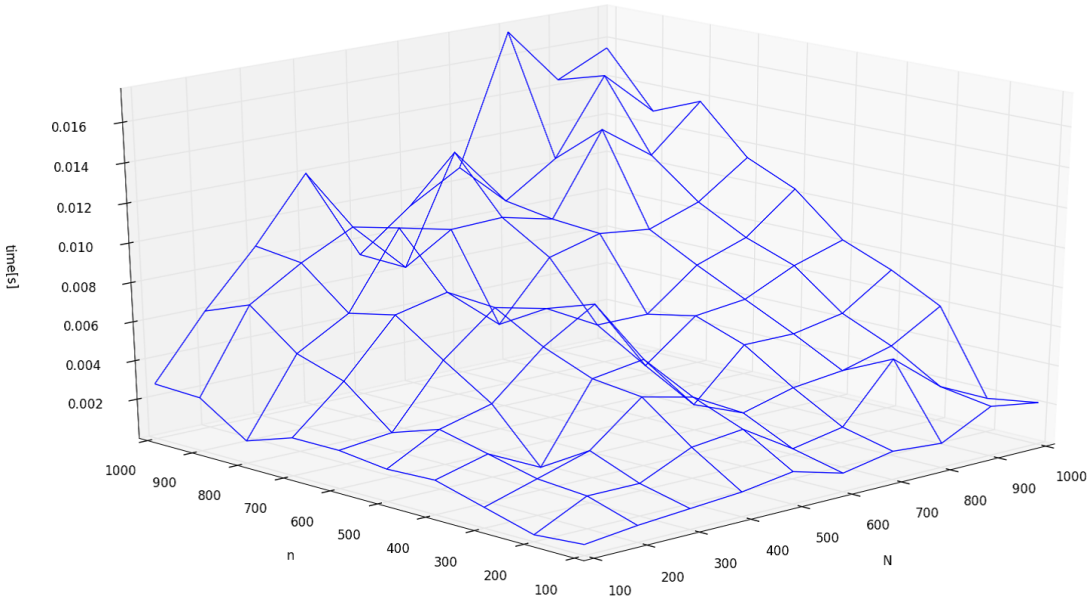


図 7.21: ハッシュ配列の構築時間 $|\Sigma| = 4$

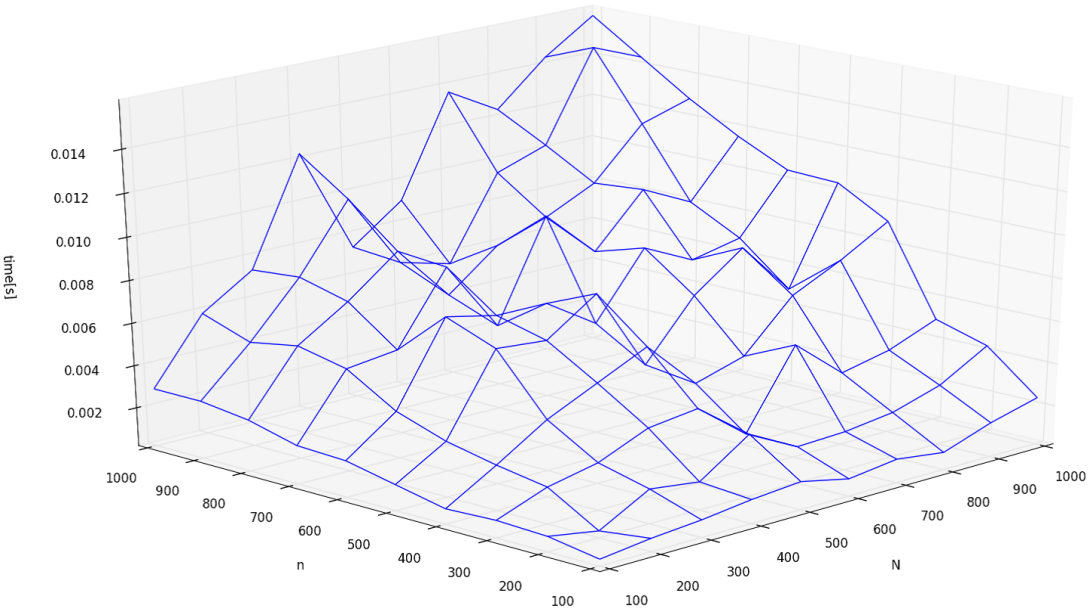


図 7.22: ハッシュ配列の構築時間 $|\Sigma| = 10$

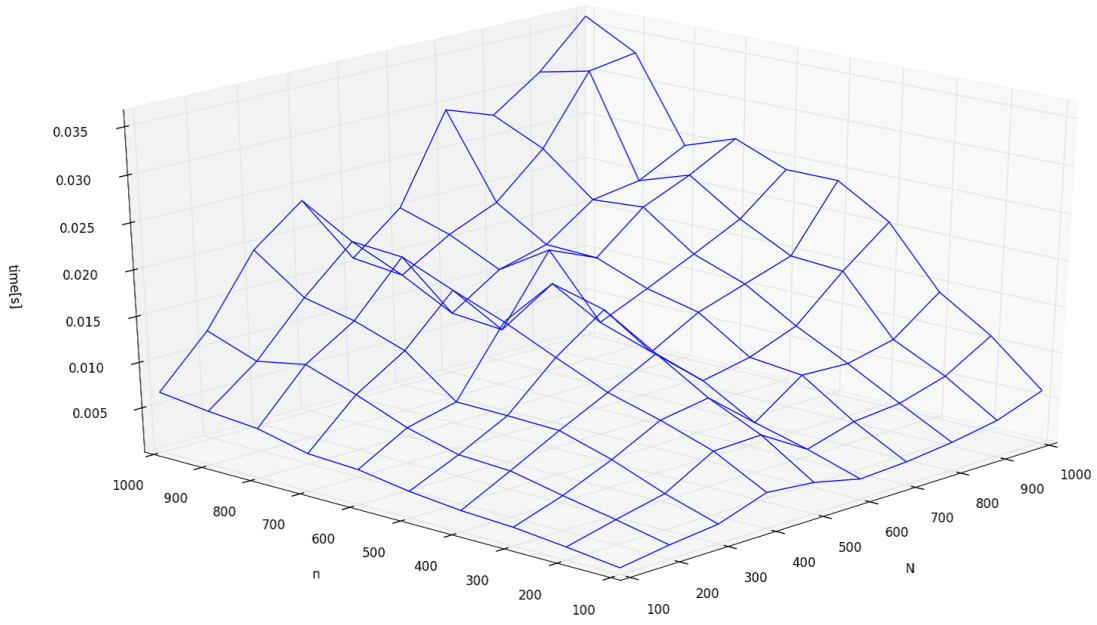


図 7.23: CMTPH の構築時間 $|\Sigma| = 2$

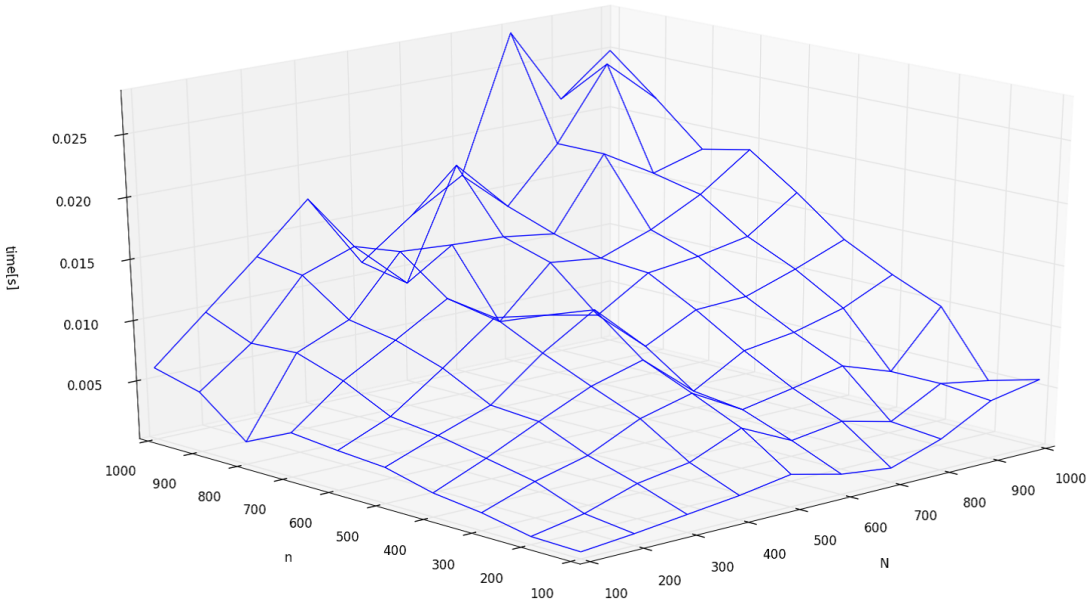


図 7.24: CMTPH の構築時間 $|\Sigma| = 4$

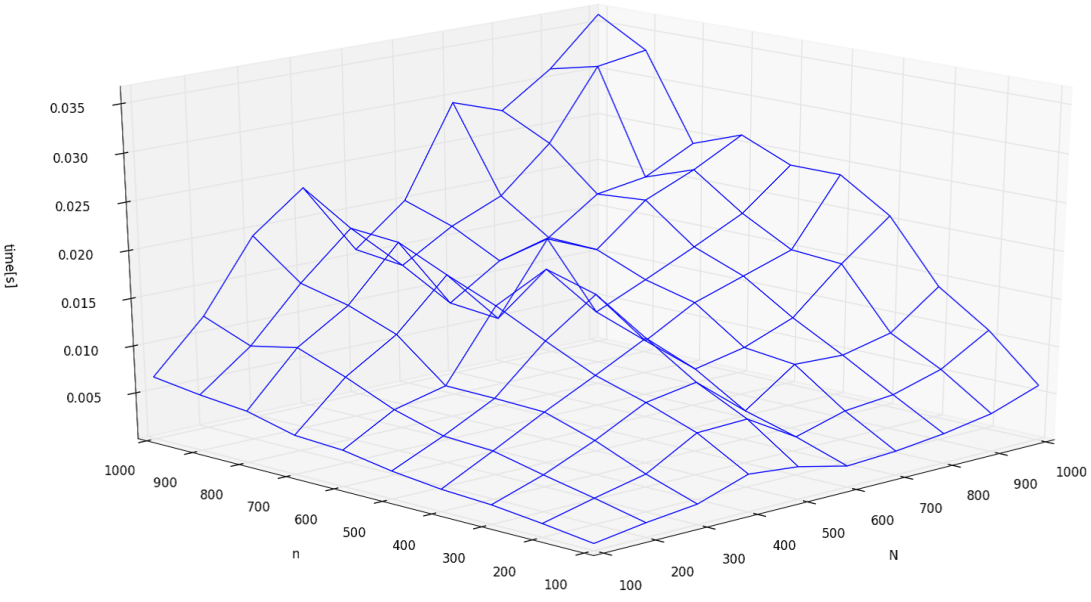


図 7.25: CMTPH の構築時間 $|\Sigma| = 10$

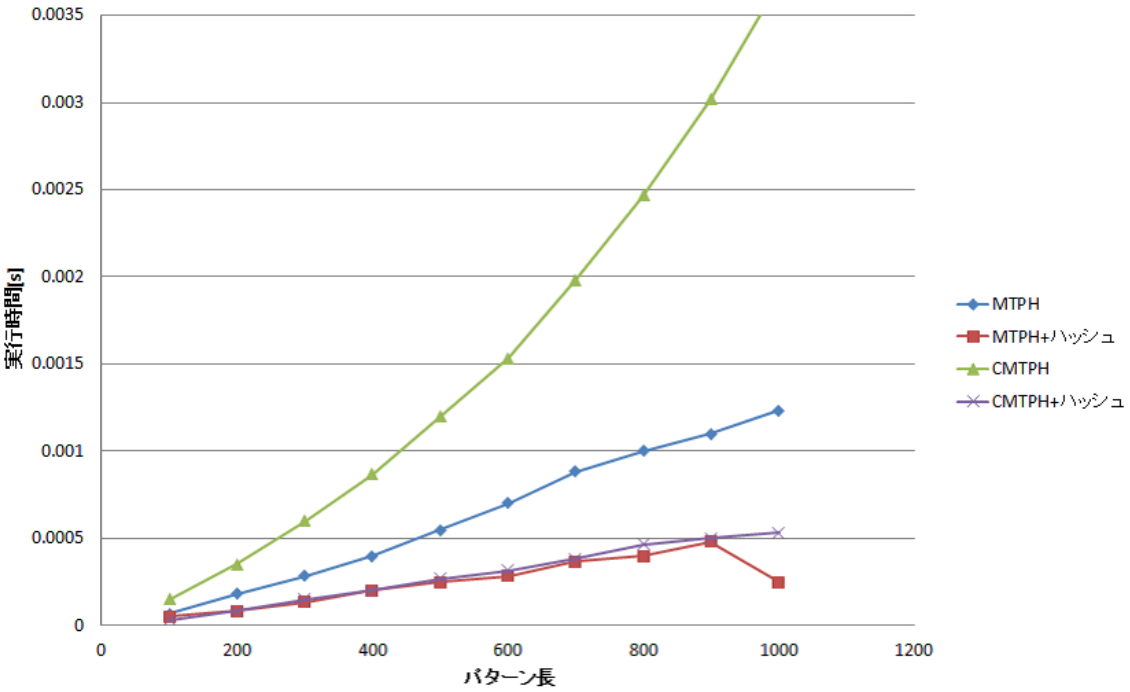


図 7.26: MTPH/CMTPH を用いた照合時間 $N = 2, |\Sigma| = 2$

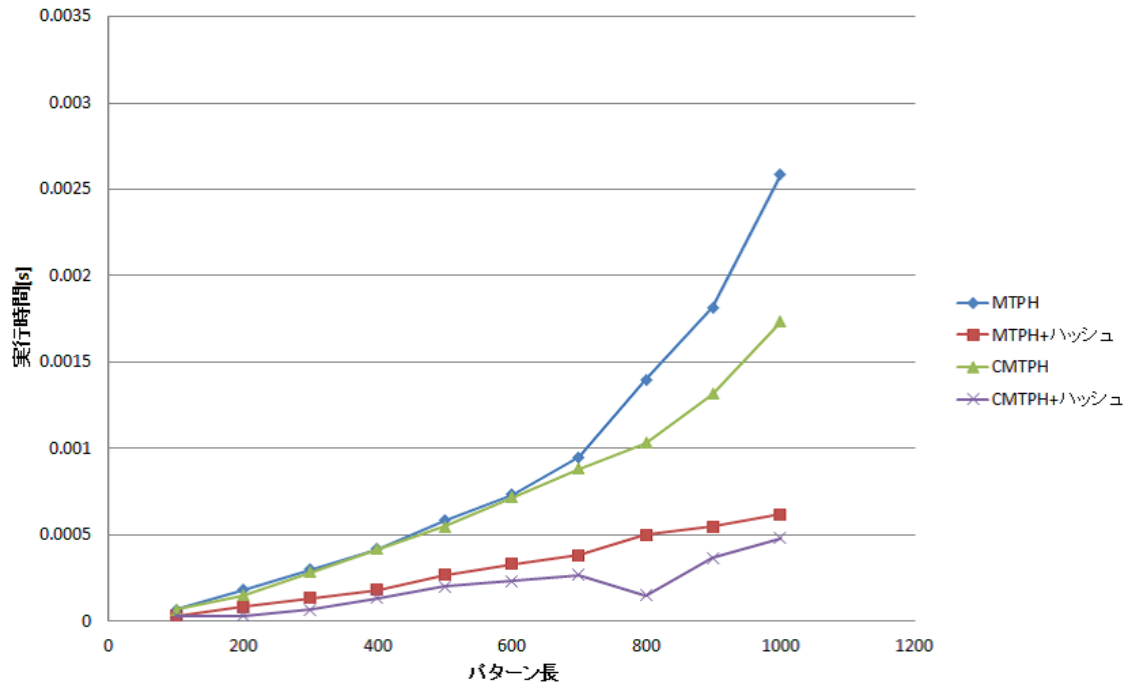


図 7.27: MTPH/CMTPH を用いた照合時間 $N = 2, |\Sigma| = 10$

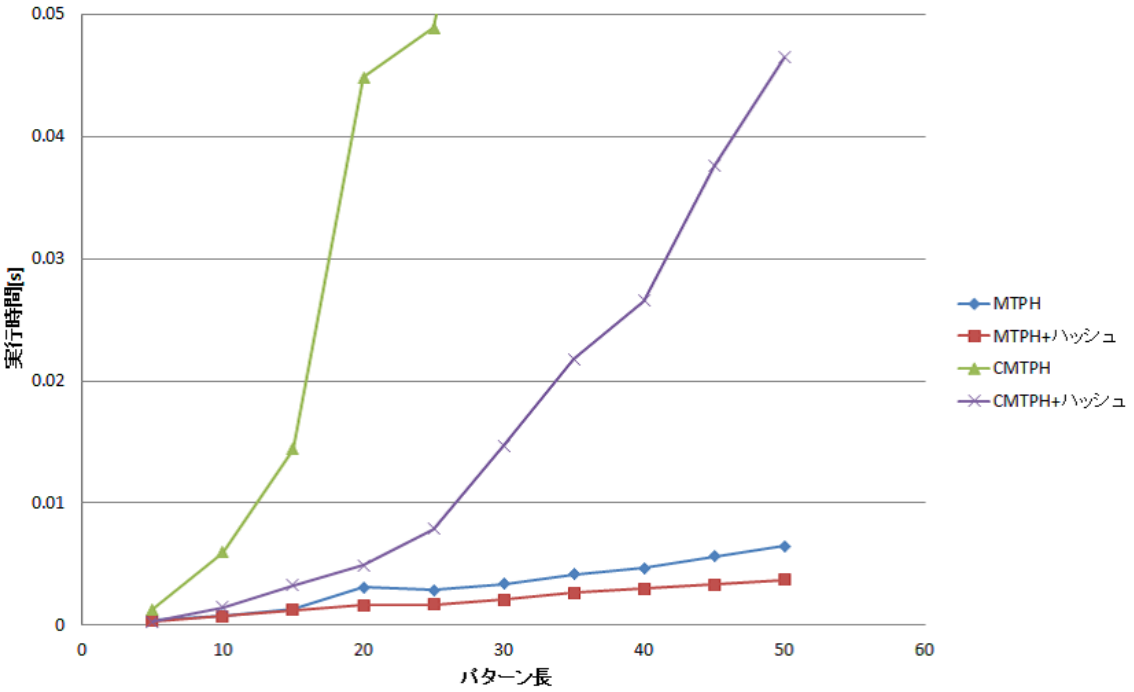


図 7.28: MTPH/CMTPH を用いた照合時間 $N = 500, |\Sigma| = 2$

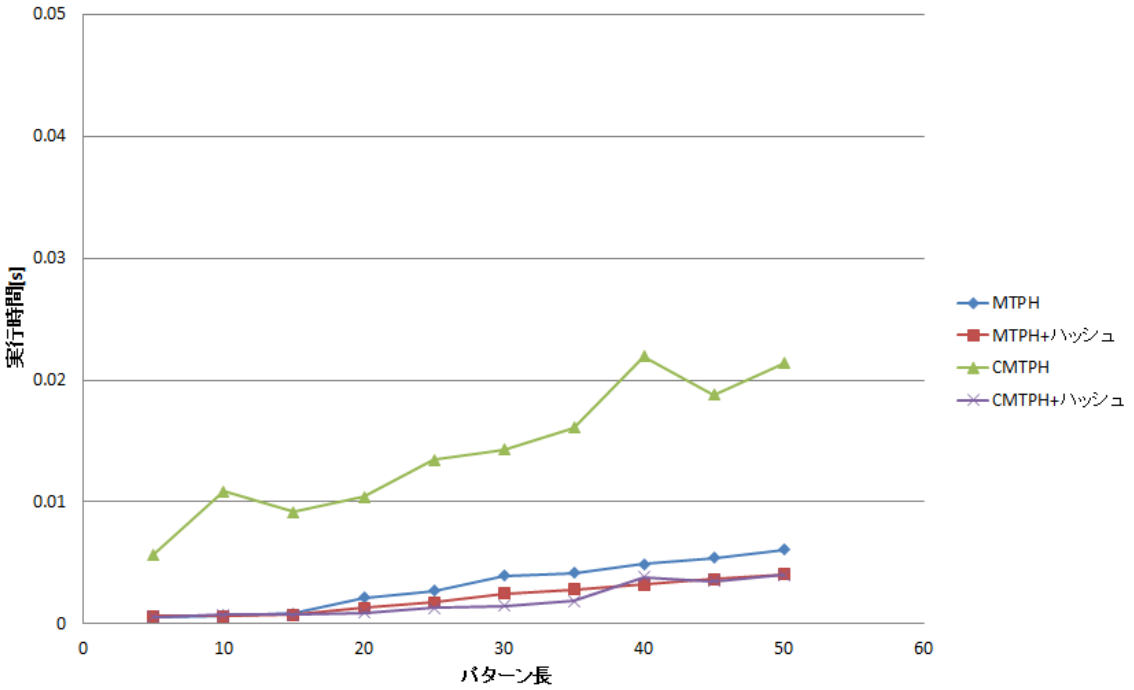


図 7.29: MTPH/CMTPH を用いた照合時間 $N = 500, |\Sigma| = 10$

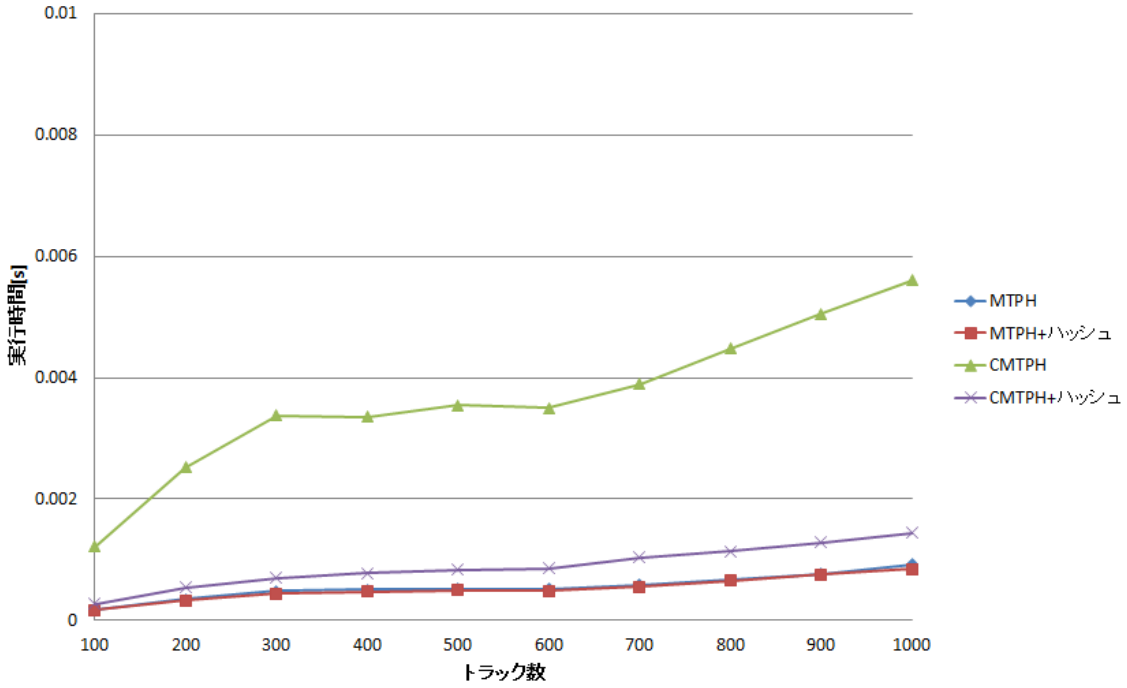


図 7.30: MTPH/CMTPH を用いた照合時間 $m = 10, |\Sigma| = 2$

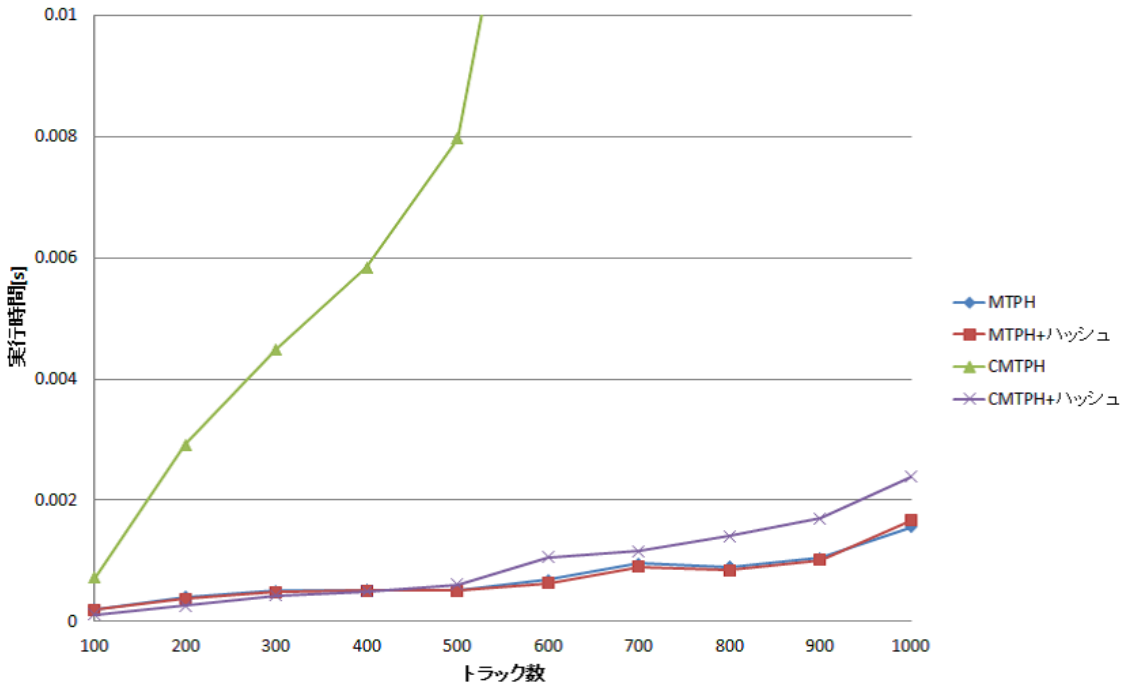


図 7.31: MTPH/CMTPH を用いた照合時間 $m = 10, |\Sigma| = 10$

第8章

まとめ

文字列に対する索引構造として近年導入されたポジションヒープを，パラメタ化文字列とマルチトラック文字列の照合問題であるパラメタ化パターン照合問題と順列パターン照合問題に適用できるように拡張を行った．通常のポジションヒープでパターン長の線形時間でパターン照合を行うために用いられている極大到達ポインタをパラメタ化照合のために拡張し，パラメタ化ポジションヒープを用いた高速なパターン照合アルゴリズムを提案した．また，極大到達ポインタの代わりにハッシュ配列を構築することで，パターンの線形時間でパターン照合を行う確率的パターン照合アルゴリズムを提案した．マルチトラックポジションヒープでは単純に極大到達ポインタを用いただけではパターン照合が行えないが，通常のポジションヒープでハッシュ配列を用いてパターン照合を行う方法を適用し，パターン照合に要する時間を高速化した．通常のパターン照合，パラメタ化パターン照合に関しても，ハッシュ配列を用いた手法は高速に動作することを実験的に確かめた．各々の計算量を表 8.1 にまとめる．いずれの索引構造の構築時間と領域も，入力となるテキストのサイズに関して線形となる．パラメタ化パターン照合に関しては厳密，確率的な手法についてパターン長 m とパラメタサイズ $|\Pi|$ の積に対して線形になっている．順列パターン照合に関しては確率的なパターン照合についてのみパターンの入力サイズに対して線形になっており，厳密に照合位置を出力するアルゴリズムはパターンのサイズの線形ではない．

今後の課題としては，順列パターン照合問題において厳密に照合位置を出力するパターンの線形時間照合アルゴリズムの開発や，縮約マルチトラックポジションヒープを入力サイズの線形時間のままマルチトラックポジションヒープを経由せずに構築する方法の

	構築時間	照合時間	データ構造の領域
PH [7, 14]	$O(n)$	$O(m^2 + occ)$	$O(n)$
PH+MRP [7, 14]	$O(n)$	$O(m + occ)$	$O(n)$
PH+HA	$O(n)$	$O(m + occ)$	$O(n)$
PPH	$O(n)$	$O(m^2 + occ)$	$O(n)$
PPH+MRP	$O(n)$	$O(m\|\Pi\| + occ)$	$O(n)$
PPH+HA	$O(n)$	$O(m\|\Pi\| + occ)$	$O(n)$
MTPH	$O(nN)$	$O(m^2N + occ)$	$O(nN)$
CMTPH	$O(nN)$	$O(m^2N^2 + occ)$	$O(n)$
MTPH+HA+SI	$O(nN)$	$O(mN + occ)$	$O(nN)$
CMTPH+HA+SI	$O(nN)$	$O(mN^2 + occ)$	$O(nN)$

表 8.1: 各データ構造の計算量．パラメタ化ポジションヒープはパラメタ化テキスト $\mathtt{t}$, $n = |\mathtt{t}|$，パラメタ化パターン $\mathtt{p}$, $m = |\mathtt{t}|$ に対する計算量であり，マルチトラックポジションヒープ，縮約マルチトラックポジションヒープはマルチトラックテキスト $\mathbb{T}$, $n = l(\mathbb{T})$, $N = h(\mathbb{T})$，マルチトラックパターン $\mathbb{P}$, $m = l(\mathbb{P})$, $N = h(\mathbb{P})$ に対する計算量である．MRP は極大到達ポインタ，HA はハッシュ配列と逆元配列，SI はソート順列をそれぞれ用いた手法である．PH と PH+MRP に関してのみ既存研究である．

開発，パラメタ化パターン照合で要する $|\Pi|$ の部分を減らすアルゴリズムの開発が挙げられる．

参考文献

[1] Brenda S. Baker. A program for identifying duplicated code. *Computing Science and Statistics*, Vol. 24, pp. 49–57, 1992.

[2] Brenda S. Baker. A theory of parameterized pattern matching: algorithms and applications. In *Proc. 25th annual ACM symposium on Theory of computing*, pp. 71–80, 1993.

[3] Brenda S. Baker. Parameterized pattern matching: Algorithms and applications. *Journal of Computer and System Sciences*, Vol. 52, No. 1, 1996.

[4] Robert S. Boyer and J. Strother Moore. A fast string searching algorithm. *Communications of the ACM*, Vol. 20, pp. 762–772, 1977.

[5] E. G. Coffman and J. Eve. File structures using hash functions. *Communications of the ACM*, Vol. 13, No. 7, pp. 427–432, 1970.

[6] T. Cormen, C. Leiserson, R. Rivest, and C. Stein. *Introduction to Algorithms*. McGraw–Hill, 2001.

[7] A. Ehrenfeucht, R. M. McConnell, N. Osheim, and S. W. Woo. Position heaps: A simple and dynamic text indexing data structure. *Journal of Discrete Algorithms*, Vol. 9, No. 1, pp. 100–121, 2011.

[8] Edward Fredkin. Trie memory. *Communications of the ACM*, Vol. 3, pp. 490–499, 1960.

[9] K. Fredriksson and M. Mozgovoy. Efficient parameterized string matching. *Information Processing Letters*, Vol. 100, No. 3, pp. 91–96, 2006.

[10] Gaston H. Gonnet and Ricardo A. Baeza-Yates. An analysis of the karp-rabin string matching algorithm. *Information Processing Letters*, pp. 271–274, 1990.

[11] Richard M. Karp and Michael O. Rabin. Efficient randomized pattern-matching algorithms. *IBM J. of Research and Development*, Vol. 31, pp. 249–260, 1987.

[12] Takashi Katsura, Kazuyuki Narisawa, Ayumi Shinohara, Hideo Bannai, and Shunsuke Inenaga. Permuted pattern matching on multi-track strings. In *SOFSEM2013*, to appear.

[13] Donald E. Knuth, Jr. James H. Morris, and Vaughan R. Pratt. Fast string searching in strings. *SIAM Journal on Computing*, Vol. 6, No. 2, pp. 323–350, 1977.

[14] G. Kucherov. On-line construction of position heaps. In *String Processing and Information Retrieval*, Vol. 7024 of *LNCS*, pp. 326–337, 2011.

[15] Udi Manber and Gene Myers. Suffix arrays: a new method for on-line string searches. *ACM-SIAM symposium on Discrete algorithms '90*, pp. 319–327, 1990.

[16] Yuto Nakashima, Tomohiro I, Shunsuke Inenaga, Hideo Bannai, and Masayuki Takeda. The position heap of a trie. In *String Processing and Information Retrieval*, Vol. 7608 of *LNCS*, pp. 360–371, 2012.

[17] T. Shibuya. Generalization of a suffix tree for RNA structural pattern matching. *Algorithmica*, Vol. 39, No. 1, pp. 1–19, 2004.

[18] P. Weiner. Linear pattern matching algorithm. In *Proc. 14th IEEE Symposium on Switching and Automata Theory*, pp. 1–11, 1973.

[19] 桂敬史, 成澤和志, 篠原歩. マルチトラック文字列に対するパターン発見について. 夏の LA シンポジウム 2011, 2011.

[20] 桂敬史, 成澤和志, 篠原歩, 坂内英夫, 稲永俊介. マルチトラック文字列の順列パターン照合と索引構造. コンピュテーション研究会, No. 119, pp. 1–8, 2012.

謝辞

学士，博士前期課程の 3 年間にわたり，このような研究の場を与えて頂くとともに，本研究の直接的な指導教員として多くのご教示，ご鞭撻を賜りました東北大学大学院情報科学研究科，篠原歩教授，ならびに成澤和志助教に厚く御礼申し上げます.

また，本論文審査の副審査員を務めて頂きました，東北大学大学院情報科学研究科，徳山豪教授，ならびに東北大学大学院工学研究科，伊藤彰則教授には，御専門の立場から的確なご助言や貴重なご意見を賜りましたことを，心から御礼申し上げます.

そして，日頃の研究室での活動全般にわたりご支援頂いた東北大学大学院情報科学研究科篠原研究室の皆様，学生生活を影から支えていただいた両親に心から感謝申し上げます.