Vol.2020-AL-177 No.1
2020/3/16

データストリームに対する頻出アイテム系列発見のための省メモリアルゴリズム

鳥谷部 直弥[1,a)]　喜田 拓也[1,b)]

概要：本稿では，データストリームに対する頻出アイテム系列問題に取り組む．頻出アイテム系列問題とは，データ中にある閾値より多く出現するアイテム系列の集合を求める問題である．今回，頻出アイテム系列問題の制約を緩和した (ϵ, l) 近似 ϕ 頻出アイテム系列問題を定義する．この問題は，求めるアイテム系列の長さの最大値を l とし，入力系列長 N に対して閾値 ϕN を越えて出現するアイテム系列を出力する．ここで，$0 < \epsilon < \phi < 1$ であり，出力される集合には $(\phi - \epsilon)N$ を超えて出現するアイテム系列を偽陽性の解として含むことを許容する．この問題に対し，ϵ 近似 ϕ 頻出アイテム問題を解くアルゴリズム Space saving のアイデアに基づいたアルゴリズム StringSS を提案する．StringSS は，アイテム一つあたりの更新時間がならし $\mathcal{O}(l)$ 平均時間，作業領域が $\mathcal{O}(\frac{l}{\epsilon})$ 語空間，各アイテム系列の頻度誤差が ϵN で動作する．

1. はじめに

データストリーム [1] は，ビッグデータと呼ばれる大規模データの形態の一つである．データストリームは，オンラインかつ持続して生成され続けるデータの総称であり，その規模は時間の経過により際限なく大きくなる．例えば，GPS データやセンサデータ，インターネットのパケットログなどがその代表的な例である．データストリームを効率よく処理するには，生成されたデータを随時に受け取りつつ，古いデータを取捨選択したり要約したりする必要がある．すなわち，過去のデータを全て保持しておくことが困難であるため，静的なデータと比べて特別な取り扱いが必要となる．

データストリームを構成する個々のデータをアイテムと呼ぶことにする．本稿で対象とするデータストリームは，次の三つの性質を持つと仮定する．(i) データストリームはアイテムの系列であり，その長さは際限なく長くなりうる．(ii) アイテムは各時点に一つずつ入力される．(iii) 各時点で処理できるアイテムの個数には限りがある．したがって，データストリームに対する処理は，データストリームを構成するアイテムを保持するよりも真に小さい作業領域のみを用いて迅速に行われなければならない．

本稿では，データストリームを入力とする頻出アイテム系列問題に取り組む．頻出アイテム系列問題とは，与えられた閾値より多い頻度でデータ中に出現するアイテム系列を発見する問題である．例えば，各文字をアイテムとし，abababcabc を 1 時点目から 10 時点目までのデータストリームとすると，これに対して 3 回以上出現するアイテム系列は ab, a, b の三つである．上で述べたデータストリームの性質から，この問題を小さい作業領域で厳密に解くことは困難である．よって，今回，頻出アイテム系列問題の制約を緩和した (ϵ, l) 近似 ϕ 頻出アイテム系列問題を定義する．この問題は，求めるアイテム系列の長さの上限を l とし，入力系列長 N に対して閾値 ϕN を超えて出現するアイテム系列を見つける問題である．出力される解の集合には，$(\phi - \epsilon)N$ を超えて出現するアイテム系列を含むことを許容する．

この問題に対し，Space saving(SS) [4] のアイデアを基にしたアルゴリズムである **StringSS** を提案する．ここで，SS は，系列ではなく，単一のアイテムに関する頻出アイテム問題を解くアルゴリズムの一つである．提案する StringSS は，処理時間が受け取ったアイテム数 N に対し，ならし $\mathcal{O}(l)$ 平均時間計算量，$\mathcal{O}(\frac{l}{\epsilon})$ 語の作業領域，各アイテムの頻度誤差 ϵN 以下を達成する．

本論文の構成は以下の通りである．まず，2 章で，基本的な用語の定義と頻出アイテム問題を解くアルゴリズム，文字列処理のデータ構造について説明する．続いて，3 章で頻出アイテム系列問題を解くアルゴリズム StringSS を提案する．最後に，4 章で本論文のまとめと今後の課題について述べる．

[1] 北海道大学 大学院情報科学研究科
Sapporo, Hokkaido 060–0814, Japan
[a)] toriyabe@ist.hokudai.ac.jp
[b)] kida@ist.hokudai.ac.jp

2. 準備

本章では，基本的な用語の定義に加えて，データストリームを入力として頻出アイテム問題を解くアルゴリズムと，文字列の索引構造の説明を行う．

2.1 データストリーム

順序が定義された有限個の記号の集合 Σ を**アルファベット**，アルファベットの各要素を**アイテム**とよぶ．あるアイテム i がデータストリーム中に出現していることを $i \in \mathcal{S}$ とかく．アルファベットサイズを σ とかく．本論文では，**データストリーム** $\mathcal{S}$ を離散的な時点毎に際限なく生成されるアイテムの列であると定義する．ある時点 t に生成されたアイテムを $s_t \in \Sigma$ としたとき，時点 n におけるデータストリームは $\mathcal{S}_n = s_1 s_2 \ldots s_n$ とかける．

ここから，データストリームに対するアルゴリズムについて説明する．データストリームは終わりがないデータのため，いつアルゴリズムが終わるかが明確でない．そのため，データストリームに対するアルゴリズムは，アルゴリズムの構成によって「停止」を定める必要がある．加えて，際限なく生成された大規模なデータ全てを保持しておき，まとめて処理することは難しいため，データ全体を保持せずにデータを先頭から順に処理するオンライン処理のみから構成される．したがって，データストリームに対するアルゴリズムは，各時点での更新処理とアルゴリズムの解を求めるクエリ処理から構成される．以上から，あらかじめ保持されている静的なデータに対するアルゴリズム全てをデータストリームに適用できるわけではないことがわかる．一方，データストリームに対するアルゴリズムは，静的データを前から順に処理することで実行可能である．したがって，データストリームに対するアルゴリズムは，静的なデータに対するアルゴリズムと比べて難しく，実行可能な処理がより限られることがわかる．本研究では，十分に大きい時点 n において作業領域 $o(n)$ 語空間を用いて，更新処理，クエリ処理をそれぞれ $o(n)$ 時間で実行するアルゴリズムを，データストリームを入力とするアルゴリズムと位置づける．

2.2 頻出アイテム問題

アイテム i がデータ中に出現する回数を i の頻度とよび，大きさ n のデータ内の i の頻度を $f_i^{(n)}$ とかく．ここで，ある閾値 ϕ（$0 < \phi < 1$）が与えられた時，アイテム i に対して，$f_i^{(n)} > \phi n$ を満たすならば，i は時点 n で頻出であるといい，i を**頻出アイテム**とよぶ．静的なデータに対する**頻出アイテム問題**とは，データ中に存在する全て頻出アイテムを過不足なく含む集合を求める問題である．一方，データストリームに対する頻出アイテム問題とは，出力クエリを受け取った時点を N としたとき，その時点までに受け取ったアイテム列 $\mathcal{S}_N$ 内の全て頻出アイテムを過不足なく含む集合を求める問題である．今後，クエリの処理は時点 N に行うものとし，クエリ処理の結果を単に出力とかく．データストリームに対する厳密な頻出アイテム問題を定義する．

定義 1 (頻出アイテム問題)．長さ N のデータストリーム $\mathcal{S}_N$ と，閾値 ϕ（$0 < \phi < 1$）が与えられた時，集合 $\mathcal{F} = \{i \in \mathcal{A} \mid f_i^{(N)} > \phi N\}$ を出力する．

この問題を解く最もナイーブなアルゴリズムは，データストリーム中に出現した全てのアイテムを正確に数え上げることである．このアルゴリズムは，アルファベットサイズ σ に対して，$\mathcal{O}(\sigma)$ 語空間の作業領域で実行できる．そのため，σ が十分に小さい値のとき，このアルゴリズムはデータストリームに対して動作することがわかる．一方，σ が非常に大きい値のとき，このアルゴリズムはデータストリームに対して動作しない．以降，本研究では σ の値が大きく，$\mathcal{O}(\sigma)$ 語空間の作業領域を認めないという仮定をおく．この仮定と $\mathcal{O}(N)$ 語空間の作業領域を認めないというデータストリームの性質を考慮したとき，頻出アイテム問題を解くことは難しい．

先行研究では，アイテムの頻度の近似値を求めることで，緩和設定を適用した頻出アイテム問題を解いた．緩和設定のうち，出力の集合に頻出ではないアイテムを含むことを許容した設定を定義 2，定義 3 にかく．一つ目の ϕ 頻出アイテム問題では，出力される集合はいかなる頻度のアイテムも含んでも良い．一方，二つ目の ϵ 近似 ϕ 頻出アイテム問題では，出力される集合は ϕN より小さいもう一つの閾値を超えたアイテムのみ含んでも良い．

定義 2 (緩和設定 1: ϕ 頻出アイテム問題)．長さ N のデータストリーム $\mathcal{S}_N$ と，閾値 ϕ（$0 < \phi < 1$）が与えられた時，条件 $\mathcal{F} \supseteq \{i \in \mathcal{A} \mid f_i^{(N)} > \phi N\}$ を満たす集合 $\mathcal{F}$ を出力する．

定義 3 (緩和設定 2: ϵ 近似 ϕ 頻出アイテム問題)．長さ N のデータストリーム $\mathcal{S}_N$ と，閾値 ϕ, ϵ ($0 < \epsilon < \phi < 1$) が与えられた時，次の二つの条件を満たす集合 $\mathcal{F}$ を出力する．
(1) $\mathcal{F} \supseteq \{i \in \mathcal{A} \mid f_i^{(N)} > \phi N\}$
(2) $\mathcal{F} \cap \{i \in \mathcal{A} \mid f_i^{(N)} \leq (\phi - \epsilon)N\} = \varnothing$

2.3 頻出アイテム問題を解くアルゴリズム

頻出アイテム問題を解くアルゴリズムは，データストリームの性質上，アイテムの正確な頻度を求めていない．その代わりに，頻度の推定値（**推定頻度**）を用いて問題を解く．時点 n におけるアイテム i の推定頻度を $\hat{f}_i^{(n)}$ とかく．さらに，あるアイテム i に対して，頻度 $f^{(N)}$ と推定頻度 $\hat{f}_i^{(N)}$ の差を e_i としたとき，正確な頻度と推定頻度との差の最大値 $E = \max_{i \in \mathcal{S}} |e_i|$ を**頻度誤差**とよぶ．

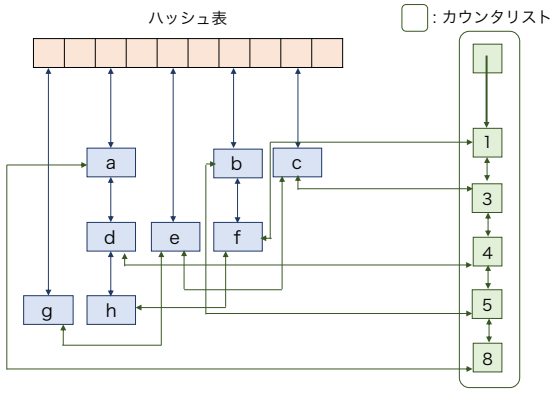

図 1 データ構造 HashList を用いたカウンタ管理の例．管理するカウンタは，$(a,8),(b,5),(c,3),(d,4),(e,3),(f,1),(g,3),(h,1)$ である．

2.4 データ構造: HashList

頻出アイテム問題を解くアルゴリズムにおいて，アイテムとその頻度の組は辞書型の構造（辞書）$\mathcal{D}$ を用いて管理される．中でも，**HashList** [2,4] は，カウンタの辞書操作を効率よく実行するデータ構造の一つである．ここで，カウンタとは，アイテムとその推定頻度を含む有限個の要素からなる組のことである．HashList は．ハッシュ表と双方向連結リストを組み合わせたデータ構造であり，作業領域 $\mathcal{O}(c)$ 語空間を用いて，カウンタの挿入，削除，探索に加えて，推定頻度が最小のカウンタを探すという操作を平均 $\mathcal{O}(1)$ 時間で実行できる．ただし，c は内部のカウンタの数である．ハッシュ表内にはカウンタを構成するアイテムのみが格納され，推定頻度は昇順になるように連結リスト（カウンタリスト）に格納する．連結リストの各セルは，保持している値に等しい推定頻度であるアイテムのリストを持つ．HashList を用いたカウンタの管理例を図 1 に示す．

2.5 Space saving (SS)

2005 年に Metwally ら [4] が提案した Space saving 法（SS）は，ϵ 近似 ϕ 頻出アイテム問題を解くアルゴリズムの一つである．このアルゴリズムは，入力として二つのパラメータ ϕ,ϵ が与えられたとき，ϕ 頻出アイテム問題を解く方法を応用して，ϵ 近似 ϕ 頻出アイテム問題を解く．すなわち，SS は ϕ のみを受け取り，ϕ 頻出アイテム問題を解くこともできる．閾値パラメータ ϕ を受け取り，条件 $\mathcal{F}' \supseteq \{i \in \Sigma \mid f_i^{(N)} > \phi N\}$ を満たす集合 $\mathcal{F}'$ を求める操作を $\mathrm{FIP}_{\mathrm{SS}}(\phi)$ とかく．SS は，操作 $\mathrm{FIP}_{\mathrm{SS}}(\epsilon)$ を用いて，頻度が ϵN より大きいアイテムを全て含む集合を求める問題 (ϵ 頻出アイテム問題) を解くように更新処理を行う．クエリ処理は，更新処理による集合から，推定頻度が閾値 ϕN より大きいアイテムのみを絞り込む．

SS の更新処理では，エラーパラメータ ϵ に対して，$k = \lceil \frac{1}{\epsilon} \rceil$ とする．この処理は以下の三つの操作から構成される．(i) 辞書に入っていなければ新たなアイテムとして挿入する（挿入操作），(ii) そのアイテムが辞書に入っているなら頻度を 1 増やす（加算操作），(iii) 辞書内のアイテムを入れ替える（交換操作）

辞書内で保持するカウンタは，アイテム i とその推定頻度 $\hat{f}_i^{(n)}$ から構成される．加算操作は，その推定頻度を 1 だけ増やす．アイテムの挿入操作は，新たなアイテムとその推定頻度 1 から構成されるカウンタを辞書に挿入する．挿入時に，辞書内のカウンタの個数，すなわち辞書内で管理されているアイテムの個数が k 個の場合は，推定頻度が一番小さいアイテムと新たなアイテムを交換し，元の推定頻度に 1 を加えた値を新しい推定頻度する．今後，辞書内の最も小さい頻度をデルタ値とよび，時点 n に対して Δ^n とかく．

Metwally らの論文 [4] から SS の更新処理の性能に関する補題 4，SS の更新操作におけるアイテムの頻度誤差を示した補題 5 とクエリ操作における頻度誤差と正当性に関する補題 6 が成り立つ．

補題 4 ([4] の 3.1 節)．Space saving は，データ構造として HashList を用いると，$\mathcal{O}(1)$ 平均時間，$\mathcal{O}(\frac{1}{\epsilon})$ 語空間で ϵ 頻出アイテム問題を解く．

補題 5 ([4] の 4.1 節)．SS の更新操作のある時点 n における，アイテム i の頻度を $f_i^{(n)}$，推定頻度を $\hat{f}_i^{(n)}$，そのアイテムが辞書に挿入された時点 n' におけるデルタ値を $\Delta^{(n')}$ とする．このとき，任意の時点，アイテムに対して，$f_i^{(n)} \leq \hat{f}_i^{(n)} \leq f^{(n)} + \Delta^{(n')} \leq f^{(n)} + \epsilon n$ が成り立つ．

補題 6 ([4] の 4.1 節)．アイテム i の頻度を $f_i^{(N)}$，推定頻度を $\hat{f}_i^{(N)}$ とする．ここで，パラメータ ϕ $(0 < \phi < 1)$ と，集合 $\mathcal{F}' \supseteq \{i \in \mathcal{A} \mid f_i^{(N)} > \epsilon N\}$ を与えたとき，集合 $\mathcal{F} = \{i \mid \hat{f}_i^{(N)} > \phi N\}$ は ϵ 近似 ϕ 頻出アイテム問題の解になる．

したがって，補題 4 と補題 5，補題 6 により，Space saving の正当性と性能に関する以下の定理 7 が成り立つ．

定理 7 ([4] の定理 2)．Space saving は，データ構造として HashList を用いると，更新処理が $\mathcal{O}(1)$ 平均時間，クエリ処理が $\mathcal{O}(\frac{1}{\epsilon})$ 平均時間，作業領域が $\mathcal{O}(\frac{1}{\epsilon})$ 語空間，頻度誤差が $\Delta^{(N)}$ で，ϵ 近似 ϕ 頻出アイテム問題を解く．

2.6 文字列とその索引構造

データストリームを構成するアイテムが文字であるとき，データストリームは文字の列であり，条件付きの文字列として扱うことができる．そのため，文字列に対する用語を，データストリームに対しても用いることができる．加えて，文字列に対するデータ構造やアルゴリズムが，データストリーム上で動作するとき，その手法はデータストリームに対するアルゴリズム，データ構造とみなすことができる．

以降では，アイテム系列に対する基本的な定義を行う．

Algorithm 1 Space saving の更新処理

```
procedure FIP_SS(ε)
    n ← 0, D ← ∅, k = ⌈1/ε⌉.
    loop
        i ← s_n.
        if i ∈ D then
            f̂_i^(n) ← f̂_i^(n) + 1.
        else if |D| ≤ k then
            D ← D ∪ {i}.
            f̂_i^(n) ← 1.
        else
            j ← 推定頻度が一番小さいアイテム.
            f̂_i^(n) ← f̂_j^(n) + 1.
            D ← (D \ {j}) ∪ {i}.
        end if
        n ← n + 1.
    end loop
end procedure
```

Σ^* の要素をアイテム系列とよび，あるアイテム系列 T に対してその長さを $|T|$ とかく．今後，長さが 1 のアイテム系列とアイテムを同一視する．集合 Σ^* に対して，連結演算と繰り返し演算を定義する．二つのアイテム系列 s と t に対して，s の末尾に t を連結させる演算を演算子 $\cdot$ を用いて，$s \cdot t$ とかく．特に問題のない限り，$s \cdot t$ を st と演算子を省略して用いることにする．あるアイテム a の b 回の繰り返しを a^b とかく．

加えて，$T = xyz$（$x, y, z \in \Sigma^*$）であるとき，アイテム系列 x, y, z をそれぞれ，アイテム系列 T の接頭辞，部分アイテム系列，接尾辞とよぶ．アイテム系列 T の u 番目から v 番目までの部分アイテム系列を $T[u, v]$ とかく．アイテム系列 T の全ての部分アイテム系列からなる集合を $\mathrm{Sub}(T)$，集合 $\mathrm{Sub}(T)$ の要素のうち，長さが l 以下のアイテム系列のみを格納した集合を，$\mathrm{Sub}_l(T)$ で表す．あるアイテム系列 p がアイテム系列 T 中に出現する回数を p の頻度とよび，データストリームにおいて，時点 n における p の頻度を $f_p^{(n)}$ とかく．ある閾値 ϕ（$0 < \phi < 1$）が与えられた時，あるアイテム系列 p に対して，$f_p^{(n)} > \phi n$ を満たすとき，p は**頻出アイテム系列**である．

最後に，文字列に対する索引構造である接尾辞トライ (Suffix Trie; STrie) や接尾辞木 (Suffix Tree; STree) [5, 6] を，アイテム系列 T に対する索引構造として説明する．これらの構造はアイテム系列の長さ n に対して線形時間でオンライン構築可能であり，長さ m のパターン照合を $\mathcal{O}(m \log |\Sigma|)$ 時間で実行可能である．

これらの構造は，アイテム系列の全ての接尾辞を保持する木構造であり，保持している全ての接尾辞に対して，その全ての接頭辞にアクセスすることができる．そのため，アイテム系列の全ての部分アイテム系列を保持している構造といえる．各構造は，ノードとノード間の遷移枝から構成される．各ノードは文字列中のある一つの部分アイテム系列と対応しており，そのノードが存在するならばかつそのときに限り，対応する部分アイテム系列は入力中に出現する．各ノードは子ノードへの遷移枝に加えて，接尾辞リンクとよばれる先頭から一文字削ったアイテム系列に対応するノードへの遷移枝を持つ．

STrie は，文字列中の各部分文字列に対応するノードを保持する構造であるため，作業領域は $\mathcal{O}(n^2)$ 語空間である．STrie は複数のアイテム系列に対する索引構造に応用することができ，複数アイテム系列のための STrie を一般化接尾辞トライ（Generalized Suffix Trie）とよび，GSTrie とかく．

2.7 頻出アイテム系列問題

データストリームを入力とする頻出アイテム系列問題を以下に定義する．

定義 8 (データストリームに対する頻出アイテム系列問題)**.** 長さ N のデータストリーム $\mathcal{S}_N$ と，閾値 ϕ（$0 < \phi < 1$）が与えられた時，集合 $\mathcal{F} = \{p \in \mathrm{Sub}(\mathcal{S}_N) \mid f_p^{(N)} > \phi N\}$ を出力する．

頻出アイテム系列問題を厳密に解くアルゴリズムとして，前節で説明したアイテム系列に対する索引構造をオンライン構築する手法が考えられる．しかし，索引構造のオンライン構築は，アイテム系列列自体もしくは元のアイテム系列を復元できる構造を保持することを認めており，時点 n における索引構造の保持のために少なくとも $\mathcal{O}(n)$ 語空間が必要とする．そのため，データストリーム全体への索引構造としては不適切である．加えて，データストリーム中に存在するアイテム系列の個数は，$\mathcal{O}(\min(n^2, |A|^{n+1}))$ であるため，出現するアイテム系列全てを厳密に数え上げることもできない．以上より，データストリームに対して，頻出アイテム系列問題を解くことは難しい．したがって，問題設定を以下の二点で緩和した上で，頻出アイテム問題を解くアルゴリズムを応用し，頻出アイテム系列問題を解く手法について考察していく．

- 長さが l 以下の部分アイテム系列のみを考慮する，
- 出力される集合に頻度が $(\phi - \epsilon)N$ 以上，ϕN 以下のアイテムを含むことを許容する

この問題を (ϵ, l) 近似 ϕ 頻出アイテム系列問題とし，以下に定義する．

定義 9 (緩和設定: (ϵ, l) 近似 ϕ 頻出アイテム系列問題)**.** 長さ N のデータストリーム $\mathcal{S}_N$ と，閾値 ϕ, ϵ $(0 < \epsilon < \phi < 1)$，$l \in N^+$ が与えられた時，次の二つの条件を満たす集合 $\mathcal{F}$ を出力する．

(1) $\mathcal{F} \supseteq \{p \in \mathrm{Sub}(\mathcal{S}_N) \mid f_p^{(N)} > \phi N \ \text{かつ} \ |p| \leq l\}$

(2) $\mathcal{F} \cap \{p \in \mathrm{Sub}(\mathcal{S}_N) \mid f_p^{(N)} < (\phi - \epsilon)N \ \text{または} \ |p| > l\} = \varnothing$

2.8 ナイーブアルゴリズム

本節では，(ϵ, l) 近似 ϕ 頻出アイテム系列問題を解くナイーブアルゴリズムについて説明する．データストリーム $\mathcal{S}$ が与えられたとき，各時点に対する長さ j のアイテム系列の列を $\mathcal{S}^j$ とする．すなわち，時点 n における列は $\mathcal{S}^j_n = s^j_1 s^j_2 ... s^j_n$ とかける．ここで，時点 1 から $j-1$ までは長さ j の部分文字列が存在しないため，データストリーム中に出現しない特殊文字 $\$ \notin \mathcal{A}$ を用いて補うこととする．すなわち，時点 t における長さ j の文字列 t^j_t は以下のように定義される．

- $s^j_t = \$^{j-t}\mathcal{S}[1,t]$，$(t < j)$
- $s^j_t = \mathcal{S}[t-j+1,t]$，$(t \geq j)$

例えば，$\mathcal{S} = abracadabra\cdots$ のとき，$\mathcal{S}^3 = \$\$a\ \$ab\ abr\ bra\ rac\ aca\ cad\cdots$ となる．

本論文では，l 個のデータストリーム $\mathcal{S}^1, ..., \mathcal{S}^l$ のそれぞれに，ϵ 近似 ϕ 頻出アイテム問題を解くアルゴリズムを適用することで，(ϵ, l) 近似 ϕ 頻出アイテム系列問題を解くアルゴリズムをナイーブアルゴリズムと位置付ける．まず，頻出アイテム問題の解を用いて，頻出アイテム系列問題を解くための補題 10 を示す．

補題 10. データストリーム $\mathcal{S}^1, ..., \mathcal{S}^l$ が与えられたとき，各データストリームに，ϵ 近似 ϕ 頻出アイテム問題を解くアルゴリズムを適用することで求められた解をそれぞれ $\mathcal{F}^1, ..., \mathcal{F}^l$ とする．この時，$\mathcal{F}' = \mathcal{F}^1 \cup \mathcal{F}^2 \cup \cdots \cup \mathcal{F}^l$ は，(ϵ, l) 近似 ϕ 頻出部分文字列問題の解になる．

証明. 定義 9 より，$\mathcal{F}'$ が (ϵ, l) 近似 ϕ 頻出部分文字列問題の解となるための条件は以下の二つである．

条件 1: $\mathcal{F}' \supseteq \{p \in \mathrm{Sub}(\mathcal{S}_N) \mid f_p^{(N)} > \phi N \text{ かつ } |p| \leq l\}$

条件 2: $\mathcal{F}' \cap \{p \in \mathrm{Sub}(\mathcal{S}_N) \mid f_p^{(N)} < (\phi - \epsilon)N \text{ または } |p| > l\} = \varnothing$

はじめに，条件 1 が成り立つことを示す．$A_j = \{p \in \mathrm{Sub}(\mathcal{S}_N) \mid f_p^{(N)} > \phi N \text{ かつ } |p| = j\}$ とすると，$A_1 \cup \cdots \cup A_l = \{p \in \mathrm{Sub}(\mathcal{S}_N) \mid f_p^{(N)} > \phi N \text{ かつ } |p| \leq l\}$ が成り立つ．ここで，頻出アイテム問題の解の性質より，全ての $j \in \{1, \cdots, l\}$ に対して $\mathcal{F}_j \supseteq A_j$ が成り立つため，$F_1 \cup \cdots \cup F_l \supseteq A_1 \cup ... \cup A_l$ である．以上より，条件 1 が成り立つ．次に，条件 2 が成り立つことを示す．$B_j = \{p \in \mathrm{Sub}(\mathcal{S}_N) \mid f_p^{(N)} < (\phi - \epsilon)N \text{ かつ } |p| = l\}$ とする．ここで，頻出アイテム問題の解の性質より，$i = j$ のとき $B_i \cap F_j = \varnothing$ である．加えて，長さの異なる文字列の集合であるため，$i \neq j$ のとき $B_i \cap F_j = \varnothing$ である．したがって，$(F_1 \cup F_2 \cdots \cup F_l) \cap (B_1 \cup B_2 \cdots \cup B_l) = \varnothing$ であり，$\mathcal{F}' \cap \{p \in \mathrm{Sub}(\mathcal{S}_N) \mid f_p^{(N)} < (\phi - \epsilon)N \text{ かつ } |p| \leq l\} = \varnothing$ が成り立つ．さらに，$\mathcal{F}'$ は長さは l 以下の文字列の集合であるため，$\mathcal{F}' \cap \{|p| > l\} = \varnothing$ が成り立つ．以上より，条件 2 が成り立つ． □

各長さごとに頻出アイテム系列問題を解くとき，各長さごとに求められた解をそれぞれ $\mathcal{F}^1, ..., \mathcal{F}^l$ を保持するためのデータ構造が必要である．そのため，HashList を用いて，各長さ毎にアイテム系列を管理する．長さ j のアイテム系列を管理する辞書を $\mathcal{D}^j$ とし，$\mathcal{D}$ を $\mathcal{D}^1$ から $\mathcal{D}^l$ までの l 個のデータ構造が格納する全てのアイテム系列を同時に扱うデータ構造とし，StringHashList とよぶ．

ここから，ナイーブアルゴリズムの性能について考察する．前節までのアルゴリズムは，アイテムに対してハッシュ値の計算時間とハッシュ表の操作を定数時間で実行できた．しかし，アイテム系列を扱う場合，アイテム系列を辞書内で管理するためにはその長さに線形な作業領域が必要となるため，同じアルゴリズムを使用した場合でも計算量が変化する．はじめに，長さ j のアイテム系列のハッシュ値を計算する時間は $\mathcal{O}(j)$ 時間である．そのため，各時間で長さが 1 から l までのアイテム系列に対してハッシュ値を求めるための時間計算量は，$\mathcal{O}(1)$ 最悪時間である．しかし，扱う文字列が部分的に一致しているため，ハッシュ関数としてカープラビンハッシュを用いることにより，全体を $\mathcal{O}(l)$ 最悪時間で求めることが可能になる．すなわち，各長さのアイテム系列のハッシュ値を定数時間で求められる．一方，ハッシュ表の操作は，ハッシュ値の衝突を考慮したとき文字列の一致判定を行う必要があるため，長さ j のアイテム系列に対する操作を $\mathcal{O}(j)$ 平均時間必要となる．以上から，データ構造として StringHashList を用いたナイーブアルゴリズムに関して，以下の補題 11 が成り立つ．

補題 11. ϵ 近似 ϕ 頻出アイテム問題を解くアルゴリズム ALG による辞書の更新処理に必要な時間計算量を $TIME$，作業領域を $SPACE$ とする．このとき，データストリーム S を受け取り (ϵ, l) 近似 ϕ 頻出アイテム系列問題を解くために必要な時間計算量は $\mathcal{O}(l^2 \cdot TIME)$ 平均時間であり，作業領域は $\mathcal{O}(l^2 \cdot SPACE)$ 語空間である．

証明. ハッシュ表への操作がボトルネックとなるため，長さ j のアイテム系列の列 $\mathcal{S}_j$ から頻出なアイテム系列を求めるために必要な時間計算量は $\mathcal{O}(j \cdot TIME)$ 平均時間であり，作業領域は $\mathcal{O}(j \cdot SPACE)$ 語空間である．したがって，全体では時間計算量が $\mathcal{O}(l^2 \cdot TIME)$ 平均時間であり，作業領域が $\mathcal{O}(l^2 \cdot SPACE)$ 語空間である． □

2.9 既存手法: StringLC

本節では，(ϵ, l) 近似 ϕ 頻出アイテム系列問題を解く既存アルゴリズムである鳥谷部ら [7] のアルゴリズム $\mathrm{StringALG}_{\mathrm{LC,GSTrieList}}$ を説明する．本論文では，この既存アルゴリズムを StringLC とよぶ．StringLC は，ϵ 近似 ϕ 頻出アイテム問題を解くアルゴリズムである Lossy counting (LC) [3] を応用し，データ構造として，一般化接尾辞トライと HashList を組み合わせた構造である GSTrieList を用いることにより，頻出なアイテム系列を発見するアル

ゴリズムである．このアルゴリズムは，辞書内の全てのアイテム系列に対してその部分アイテム系列もまた辞書内に含むという性質（**部分アイテム系列性**）をもつ．

定義 12 (部分アイテム系列性)．ϵ 近似 ϕ 頻出アイテム問題を解くアルゴリズム ALG と，辞書型の構造 DS を用いて頻出アイテム系列問題を解くアルゴリズムを $\mathrm{StringALG}_{\mathrm{ALG,DS}}$ とする．はじめに，ある時点 t で辞書が保持するあるアイテム系列に対して，その部分アイテム系列全てもまた辞書内に保持されているとき，時点 t でそのアイテム系列は辞書内で閉じているという．任意の時点において，辞書が保持する全てのアイテム系列が閉じているとき，アルゴリズム $\mathrm{StringALG}_{\mathrm{ALG,DS}}$ は**部分アイテム系列性**を満たすという．

補題 13. StringLC は部分アイテム系列性を満たす．

証明. 初期状態は，部分アイテム系列性を満たす．LC における新たなアイテムの挿入操作，既存のアイテムの頻度を増やす操作，アイテムの削除操作によって，部分アイテム系列性が満たされることを示す．はじめに，時点 t において，長さ j のアイテム系列 $\mathcal{T}^j[t-j+1...t]$ が辞書に存在するとき，辞書内の文字列に変化がないため，部分アイテム系列性を満たす．次に，時点 t において，長さ j のアイテム系列 $\mathcal{T}^j[t-j+1...t]$ が辞書に挿入されるとき，$\mathcal{T}^j[t-j+1...t-1]$，$\mathcal{T}^j[t-j+2...t]$ は辞書内にすでに存在する．加えて，$\mathcal{T}^j[t-j+1...t-1]$，$\mathcal{T}^j[t-j+2...t]$ に対しても再帰的に成り立つため，部分アイテム系列性を満たす．ここで，$\mathrm{Naive}_{\mathrm{LC}}$ において，$\hat{f}^{(t)}_{\mathcal{T}^j[t-j+1...t]} \leq \hat{f}^{(t)}_{\mathcal{T}^j[t-j+1...t-1]}$ かつ $\hat{f}^{(t)}_{\mathcal{T}^j[t-j+1...t]} \leq \hat{f}^{(t)}_{\mathcal{T}^j[t-j+2...t]}$ が成り立つ．そのため，アイテム系列を削除する操作において，あるアイテム系列 $\mathcal{T}^j[i,j]$ が削除されるとき，そのアイテム系列を部分アイテム系列とするアイテム系列もまた削除される．したがって，StringLC は部分アイテム系列性を満たす． □

さらに，GSTrie の作業領域に関して以下の補題 14 が成り立つ．

補題 14 ([7] の 3.3 節)．ϵ 近似 ϕ 頻出アイテム問題を解くアルゴリズム ALG による辞書の更新処理に必要な時間計算量を $TIME$，作業領域を $SPACE$ とする．ALG と GSTrieList を用いて，アイテム系列問題を解くアルゴリズム $\mathrm{StringALG}_{\mathrm{ALG,GSTrieList}}$ に対して以下が成り立つ．このアルゴリズムが部分アイテム系列性を満たすならば，時間計算量は $\mathcal{O}(l \cdot TIME)$ 平均時間であり，作業領域は $\mathcal{O}(l \cdot SPACE)$ 語空間である．

以上より，StringLC の正当性と性能に関する定理 15 が成り立つ．

定理 15. StringLC は，更新処理が $\mathcal{O}(l)$ 平均時間，クエリ処理が $\mathcal{O}(\frac{l}{\epsilon}\log \epsilon N)$ 平均時間，作業領域が $\mathcal{O}(\frac{1}{\epsilon}\log \epsilon N)$ 語空間，頻度誤差が $\Delta^{(N)}$ で，(ϵ, l) 近似 ϕ 頻出アイテム系列問題を解く．

3. 提案手法

前節では，Lossy counting を用いて (ϵ, l) 近似 ϕ 頻出アイテム系列問題を解くアルゴリズム StringLC を説明した．Space saving は，Lossy counting と比べて作業領域が小さいことがわかっている．そこで，本節では，Space saving を用いて (ϵ, l) 近似 ϕ 頻出アイテム系列問題を解くアルゴリズム StringSS を提案する．はじめに，Space saving を用いて各長さ毎に頻出なアイテム系列を求めるアルゴリズム $\mathrm{StringALG}_{\mathrm{SS,\ DS}}$ が部分アイテム系列性を満たさないことを説明する．

補題 16 ($\mathrm{StringALG}_{\mathrm{SS,\ DS}}$ の部分アイテム系列性)．$\mathrm{StringALG}_{\mathrm{SS,\ DS}}$ は部分アイテム系列性を満たさない．

証明. 初期状態は，部分アイテム系列性を満たす．SS における新たなアイテムの挿入操作，既存のアイテムの頻度を増やす操作，アイテムの削除操作によって，部分アイテム系列性が満たされるかを示す．はじめに，時点 t において，長さ j のアイテム系列 $\mathcal{T}^j[t-j+1...t]$ が辞書に存在するとき，辞書内の文字列に変化がないため，部分アイテム系列性を満たす．次に，時点 t において，長さ j のアイテム系列 $\mathcal{T}^j[t-j+1...t]$ が辞書に挿入されるとき，$\mathcal{T}^j[t-j+1...t-1]$，$\mathcal{T}^j[t-j+2...t]$ は辞書内にすでに存在する．加えて，$\mathcal{T}^j[t-j+1...t-1]$，$\mathcal{T}^j[t-j+2...t]$ に対しても再帰的に成り立つため，部分アイテム系列性を満たす．ここで，$\mathrm{Naive}_{\mathrm{SS}}$ において，$\hat{f}^{(t)}_{\mathcal{T}^j[t-j+1...t]} \leq \hat{f}^{(t)}_{\mathcal{T}^j[t-j+1...t-1]}$ かつ $\hat{f}^{(t)}_{\mathcal{T}^j[t-j+1...t]} \leq \hat{f}^{(t)}_{\mathcal{T}^j[t-j+2...t]}$ が成り立つとは限らない．これは，削除操作を行う際に，削除するアイテム系列を最も頻度の低い中から任意の一つになるためである．したがって，$\mathrm{StringALG}_{\mathrm{SS,\ DS}}$ は部分アイテム系列性を満たさない． □

3.1 提案データ構造: GWSTrieList

GSTrieList 内の接尾辞リンクにその逆向きのリンクとしてワイナーリンクを加えて構造を GWSTrieList とよぶ．このデータ構造は，基本的に GSTrieList と同様の動きをする．しかし，親ノードから子ノードへの遷移とワイナーリンクを用いることにより，あるアイテム系列を部分アイテム系列として含む辞書内に存在するアイテム系列全てを，その個数に線形な時間で探索することが可能になる．

3.2 提案アルゴリズム: StringSS

補題 16 で示したように，$\mathrm{StringALG}_{\mathrm{SS,\ DS}}$ は部分アイテム系列性を満たさない．本節では，SS の考え方を用いた別のアルゴリズム SS' をベースとする部分アイテム系列性を満たすアルゴリズム $\mathrm{StringALG}_{\mathrm{SS'},\ \mathrm{GWSTrieList}}$ を提案する．このアルゴリズムを StringSS とよぶ．

はじめに，各長さごとに**空列**を定義し，空列を管理する

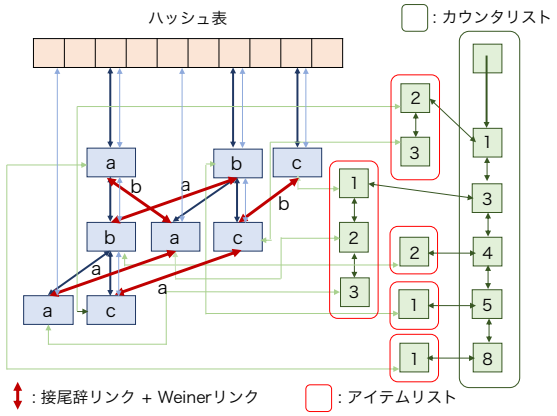

図 2　データ構造: GWSTrieList

辞書のカウンタとノードをそれぞれ空列カウンタ，空列ノードとよぶ．

定義 17 (空列)**.**
長さ l の空列 φ_l とは，以下の性質を満たす列のことである．
(1) $\varphi_l \notin \mathcal{A}^*$.
(2) $|\varphi_l| = l$.
(3) $s + \varphi_l = \varphi_l + s = s$.
ただし，$s \in \mathcal{A}^*$ とする．

Space saving の更新処理は，アイテムの挿入操作，加算操作，交換操作で構成された．StringSS では交換操作を新たなアイテムを挿入した後に古いアイテムを削除する操作とみなし，全体を挿入操作，削除操作の二つの操作として扱う．さらに，Space saving を用いたナイーブアルゴリズムでは，各長さのアイテム系列を順番に受け取り処理を行ったが，StringSS は先に全てのアイテム系列を挿入した後に，削除操作を行う．

はじめに，アイテム系列の削除操作を説明する．辞書内の各長さのアイテム系列の個数 $c_1, \cdots, c_l$ と，各長さのアイテム系列のうち最小の推定頻度 $d_1, \cdots, d_l$ を管理しているものとする．ある時点 n における辞書内のアイテム系列の総数 $C^{(n)} = c_1 + \cdots c_l$ とし，辞書内のアイテム系列の最小頻度 $\Delta^{(n)} = \min(d_1, \cdots, d_l)$ とする．時点 n における削除操作は，長さが l から 1 の順に $c_j = \epsilon n$ を満たすならば，その長さの推定頻度が最も小さいアイテム系列を削除する．加えて，そのアイテム系列の部分木内のアイテム系列を削除し，同長さの空列を挿入する．ここで，アイテム系列の部分木とは，そのアイテム系列を部分アイテム系列として持つ全てのアイテム系列にアクセスするための構造であり，GWSTrieList においては親ノードから子ノードへ遷移するポインタとワイナーリンクを合わせることで実現している．空列は辞書内に空列を各長さごとに保管するリストを作成し，そこに連結リストからのリンクをつなぐものとし，ハッシュ表内の個数に含まれるものとする．そのため，削除操作の対象となったアイテム系列と，そのアイテム系列を部分アイテム系列としてアイテム系列とでは，

Algorithm 2 StringSS

```
1: procedure StringSSUpdate(ϵ, l)
2:     n ← 0, D ← ∅, k = ⌈1/ϵ⌉.
3:     l 次元のベクトル Δ_l^(0) = {d_1, ..., d_l} を全て 0 で初期化する.
4:     loop
5:         INSERT(n, D, k, Δ_l^(n)).
6:         DELETE(n, D, k, Δ_l^(n)).
7:         n ← n + 1.
8:     end loop
9: end procedure
```

Algorithm 3 StringSS の削除操作

```
1: procedure DELETE(n, D, k, Δ^(n))
2:     for j = l to 1 do
3:         i ← S[n − j + 1...n].
4:         if 長さが j のアイテム系列が k + 1 個辞書に保持されている then
5:             頻度が最小のアイテム系列を削除する.
6:             for each 削除したアイテム系列の部分木構造となるノード do
7:                 対象のノードを同じ長さの空列ノードにする.
8:             end for
9:         end if
10:    end for
11: end procedure
```

扱いが異なることに注意したい．StringSS の更新処理における削除操作に関する系 18 が成り立つ．

系 18. StringSS の更新処理における削除操作に関して以下が成り立つ．時点 t のアイテム系列の挿入操作後に辞書内に存在する全てのアイテム系列が辞書内で閉じているものとする．このとき，アイテム系列の削除操作後に辞書内に存在する全てのアイテム系列は辞書内で閉じている．

つぎに，StringSS の更新処理における挿入操作について説明する．時点 n における挿入操作は，長さが 1 から l の l 個のアイテム系列 $\mathcal{S}[n,n], \mathcal{S}[n-1,n], \cdots \mathcal{S}[n-l+1,n]$ を扱う．長さ j のアイテム系列 $\mathcal{S}[n-j+1,n]$ を辞書内に挿入する操作は，以下の通りである．そのアイテム系列がすでに辞書内に存在するとき，そのアイテム系列の推定頻度を 1 だけ増やす．これは，Space saving の加算操作に対応する．辞書内に存在しない場合は，対象のアイテム系列が部分アイテム系列性を満たすかを調べる．$\mathcal{S}[n-j+1,n-1]$ が辞書内に存在しないとき，対象となるアイテム系列は部分アイテム系列性を満たさないため，同じ長さの空列ノードを代わりに挿入する．$\mathcal{S}[n-j+1,n-1]$ が辞書内に存在するとき，空列が存在しないならば，対象のアイテム系列を推定頻度が $d_j + 1$ として辞書内に挿入する．StringSS の更新処理における挿入操作をアルゴリズム 4 にかく．StringSS の更新処理における挿入操作に関する系 19 が成り立つ．

系 19. StringSS の更新処理における挿入操作に関して以下が成り立つ．時点 t の開始時に辞書内に存在する全てのアイテム系列が辞書内で閉じているものとする．このと

Algorithm 4 StringSS の挿入操作

```
1: procedure INSERT(n, D, k, Δ_l^(n))
2:     for j = 1 to l do
3:         i ← S[n − j + 1...n].
4:         if i ∈ D then
5:             アイテム系列 i の頻度を 1 増やす
6:         else if S[n − j + 1...n − 1] ∈ D then
7:             辞書内にアイテム系列 i を管理するノードを頻度 d_j+1
   で挿入する.
8:         else
9:             長さが j の空列ノードを頻度 d_j + 1 で挿入する
10:        end if
11:    end for
12: end procedure
```

き，アイテム系列の挿入操作後に辞書内に存在する全てのアイテム系列は辞書内で閉じている．

補題 20 (StringSS の部分アイテム系列性)**.** StringSS は部分アイテム系列性を満たす．

証明. 初期状態は部分アイテム系列性を満たす．系 18 と系 19 により，この補題が成り立つ． □

ここから，StringSS の性能と正当性を説明する．はじめに，StringSS のアルゴリズムから，以下の 3 つの系が成り立つ．

系 21. StringSS に関して以下が成り立つ．ある時点 n において，$\Delta^{(n)} \leq \epsilon n$ が成り立つ．

系 22. StringSS に関して以下が成り立つ．時点 n においてあるアイテム系列 i が辞書内に挿入されるとき，$f_i^{(n)} \leq \hat{f}_i^{(n)} = \Delta^{(n)} + 1$ を満たす．

系 23. StringSS に関して以下が成り立つ．時点 n においてあるアイテム系列 i が辞書内から削除されるとき，$f_i^{(n)} \leq \hat{f}_i^{(n)} \leq \Delta^{(n)}$ を満たす．

加えて，系 21，系 22，系 23 より，系 24 と系 25 もまた成り立つ．

系 24. StringSS に関して以下が成り立つ．ある時点 n において，$f_i^{(n)} > \Delta^{(n)}$ を満たす全てのアイテム系列 i は辞書内に保持される．

系 25. ある時点 n において，StringSS の頻度誤差は $\Delta^{(n)}$ で抑えられる．

以上より，StringSS の更新処理の性能に関する補題 26 を示す．

補題 26. StringSS の更新処理は，$\mathcal{O}(l)$ 平均時間，$\mathcal{O}(\frac{l}{\epsilon})$ 語空間で実行可能である．

証明. 系 24 より，更新処理の正当性が示される．更新処理の挿入操作は，GWSTrieList への辞書操作のみから構成されるため，$\mathcal{O}(l)$ 平均時間で実行可能である．各時点における削除操作は，GWSTrieList の親から子ノードへの遷移とワイナーリンクを用いた全探索を用いることで，削除するノード数に線形な計算時間で行うことができる．したがって，アイテム系列一つあたりの削除に必要な時間は，$\mathcal{O}(1)$ 最悪時間である．さらに，更新処理全体で削除する個数が入力されるアイテム系列の個数で抑えられるため，削除操作にかかる時間はならし $\mathcal{O}(l)$ 最悪時間である．したがって，更新処理にかかる時間は $\mathcal{O}(l)$ 平均時間である．補題 14 と補題 20 により，作業領域は $\mathcal{O}(\frac{l}{\epsilon})$ 語空間である． □

最後に，補題 6 と系 25 と補題 26 から，以下の定理 27 が成り立つ．

定理 27. StringSS は，更新処理が $\mathcal{O}(l)$ 平均時間，クエリ処理が $\mathcal{O}(\frac{l}{\epsilon})$ 平均時間，作業領域が $\mathcal{O}(\frac{l}{\epsilon})$ 語空間，頻度誤差が $\Delta^{(N)}$ で，(ϵ, l) 近似 ϕ 頻出アイテム系列問題を解く．

4. おわりに

本論文は，頻出アイテム系列問題の緩和設定である (ϵ, l) 近似 ϕ 頻出アイテム系列問題を解くアルゴリズム StringSS を提案した．StringSS は，アイテム数 N に対してならし $\mathcal{O}(l)$ 平均時間，作業領域が $\mathcal{O}(\frac{l}{\epsilon})$ 語空間，頻度誤差が ϵN で動作する．このアルゴリズムは，頻出アイテム問題を解くアルゴリズム Space saving を応用した部分アイテム系列性を満たすアルゴリズムであり，GSTrieList にワイナーリンクを加えた構造であるデータ構造 GWSTrieList と組み合わせることで，(ϵ, l) 近似 ϕ 頻出アイテム系列問題を解く．

今後の課題として，頻出アイテム系列問題を解く提案アルゴリズムの比較実験と，他の文字列アルゴリズムに対する応用の検討が挙げられる．

参考文献

[1] Babcock, B., Babu, S., Datar, M., Motwani, R. and Widom, J.: Models and issues in data stream systems, *Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems(PODS)*, ACM, pp. pp.1–16 (2002).

[2] Karp, R. M., Shenker, S. and Papadimitriou, C. H.: A simple algorithm for finding frequent elements in streams and bags, *ACM Transactions on Database Systems (TODS)*, Vol. 8, No. 1, pp. 51–55 (2003).

[3] Manku, G. S. and Motwani, R.: Approximate frequency counts over data streams, *Proceedings of the 28th international conference on Very Large Data Bases*, pp. 346–357 (2002).

[4] Metwally, A., Agrawal, D. and El Abbadi, A.: Efficient computation of frequent and top-k elements in data streams, *International Conference on Database Theory*, Springer, pp. 398–412 (2005).

[5] Ukkonen, E.: On-line construction of suffix trees, *Algorithmica*, Vol. 14, No. 3, pp. 249–260 (1995).

[6] Weiner, P.: Linear pattern matching algorithms, *14th Annual Symposium on Switching and Automata Theory (swat 1973)*, IEEE, pp. 1–11 (1973).

[7] 鳥谷部直弥，古谷　勇，喜田拓也：データストリームのための頻出部分文字列発見アルゴリズム，*2019-AL-173(6)*，情報処理学会，pp. 1–8 (2019).