情報組織化論 3（データベース利用論）

98513 高久雅生

1999 年 3 月 11 日

4.3 Inverted File Structure

DBMS と情報検索システムの両方に共通でよく用いられているデータ構造は、転置ファイル (Inverted File) である。転置ファイルは 3 つの基本ファイルから構成されている。文書ファイル、転置リスト（時にポスティングファイルとも呼ばれる）、及び辞書ファイルである。「転置ファイル (Inverted File)」という名前は、文書を転換したものを蓄積するというその基本的な手法に由来するものである。つまり、各単語に対して単語の出現した文書のリストを蓄積すると言う意味では、文書の転換（転置）したもの（その単語に対する転置リスト）が蓄積されるともいえる。システム中における各文書は一意の番号で識別される。転置リストに登録されているのは、その識別子である。ある単語に対して転置リストを探索するには辞書ファイルを通じて行なう。辞書ファイルはシステム中の全ての一意の単語（処理トークン）とその転置リストの場所へのポインターをソートしたリストである（図 4.5 参照）。辞書ファイルは検索質問の最適化で用いられる転置リストの長さのような別の情報をも蓄積できる。

適合率を増やし、転置リストの構造をより最適化するために、資料からは他の情報も追加されることもある。例えば、ゾーニングが使われていれば、辞書ファイルはゾーン毎に区分されているかもしれない。資料中の「Abstract」部分のための辞書ファイルと転置リストと、「本文」部分のための辞書ファイルと転置リストが別になっていることもあるだろう。特定のゾーンに検索を限った場合とは対照的に、このことにより、ユーザが全資料を検索したい時には、オーバーヘッドは増加するだろう。また、別の一般的な最適化としては、転置リストが 1 つか 2 つの項目しか含まない時に行なわれるものもある。これらの項目は辞書ファイルの一部として蓄積されうる。転置ファイルは単語が見つかった各文書の文書番号を含んでいる。近接演算や連続した単語句 (contiguous word phrase) や用語の重みづけアルゴリズムを用いるためには、単語の全ての出現はその位置と一緒に転置リストに蓄積される。したがって、もし "bit" という単語が文書 #1 中で 10,12,18 番に現れたら、転置リストは次のようになるだろう。

bit - 1(10), 1(12), 1(18)

重みも転置リストに蓄積されうる。特殊な文字を含む単語はしばしば最善の内部表現と内部操作を行なうために、それ自身の辞書に蓄積される（例えば、範囲を要する日付や数字）。

検索が行なわれる際には、検索質問中の用語に対して転置リストが探索され、転置リスト間でそれにふさわしい論理操作が適用される。検索結果は質問を満たす資料の最終的な

ヒットリストである。ランクづけを行なうシステムに対しては、ランク順にリストは再編成される。文書番号は文書ファイルから文書をとってくるために用いられる。図 4.5 に示す転置リストを使うと、まず、検索質問 (bit AND computer) は “bit” と “computer” に対する転置リストを探索するために、辞書ファイルを用いる。これらの 2 つのリストは論理的に AND が取られ、つまり、 (1,3) AND (1,3,4) の結果は (1,3) のヒットリストとなる。

転置リストへのポインターを持つ辞書を使う代わりに、 B ツリー (B-Tree) も使われる。転置リストは葉 (leaf) か、より高いレベルで参照される。図 4.6 に図 4.5 中の単語がどのようになるかを示す。 m のオーダーの B ツリーは次のように定義される。

根ノード (root node) は 2 ~ $2m$ のキーを持つ。
他の全内部ノードは m ~ $2m$ のキーを持つ。
全てのキーは昇順に整列されている。
全ての葉ノードは同一のレベルにあるか、もしくは、高々 1 レベルしか違わない。

Cutting と Pedersen は大量の更新を行なうデータに対する有効な転置ファイルの蓄積手法として、 B ツリーを用いることについて述べている。 (Cutting-90)

情報システムの性質として、資料は一旦作成されると滅多に変更されないというものがある。ほとんどの商用システムはこの事実を利用して、増大する文書ファイルと対応する転置リストがある上限のサイズになるとそれらを凍結して、新しい構造を構築し始めるのである。文書ファイル、辞書ファイル、転置リストのこれらのデータベースの各々は、保管され、ユーザの検索質問に提供されている。このことは、最新の情報にのみ関心のある質問に対しては、最新のデータベースを検索すればよいという利点を持つのである。より古い資料は滅多に削除されたり、修正されたりしないので、保管されたデータベースは永続的にバックアップされ、したがって、運用オーバーヘッドを省くものとなる。新しい転置データベースを始める際には、頻出する単語が辞書ファイルと転置リストに追加されるまでは、辞書ファイルと転置リストに新しい単語の追加という顕著なオーバーヘッドが存在する。保管されたデータベースの従来の知識は、新しいデータベースを開始する際に、存在している辞書ファイルと転置構造体を使うことで利用可能となる。このようにして、新しい文書の初期追加時に起きるオーバーヘッドはだいぶ減らせる。[2]

転置リスト構造は、大規模データベースを検索する際に最善の実行が行なえるという理由から利用されている。このような最善性は検索質問の解決を行なう際のデータフローの最小化に由来するものである。質問に直接関連するデータのみが 2 次的な蓄積から返されるのである。辞書ファイル中にある情報に基づいて、検索質問の解決を最適化するのに利用される手法にも多くのものが存在する。

転置リストファイルの構造は概念とそれらの関係を蓄積するのによく適応したものである。個々の転置リストはある特定の概念の表現としても考えられる。転置リストはその概念を含む全資料の用語索引である。概念のより良い表現は、資料中の場所や転置リスト中の資料の重みを蓄積することにより、加えられる。この情報から概念関係は検索アルゴリズムの一部として決定される（第 7 章参照）。概念の探索は辞書ファイルと転置リスト中

[2] INQUIRE DBMS はこの機能を提供している。

の一覧によって、容易に行なえる。自然言語処理アルゴリズムでは、別の構造がより適切であるか、もしくは必要な意味的・文法的情報を保持するために転置リストに加えて、他の構造を加える必要があるかも知れない。

4.4 N-Gram Data Stractures

N- グラム (N-Grams) は合成のための特殊な手法、情報システムにおける独特のデータ構造とみなされている。 N- グラムはある決まった長さ “n” 文字からなる文字列である。一般的に語の意味を表す語幹を決定しようとするステミングとは違い、 N- グラムは意味を考慮しない。その代わりに、ある一定の数の文字列に基づいたアルゴリズムを持っている。検索可能なデータ構造は、 N- グラムを重ねたものに変換され、それがデータベースを構築するのに使われる。図 4.7 に 2-、 3-、 5- グラムによる「sea colony」という語句に対する例を示す。 N が 2 以上の N- グラムでは、 N- グラムの一部として普通は語間記号の 1 文字を除いているが、語間記号を採用するシステムもある。記号 # は何らかの記号である語間記号(空白、ピリオド、セミコロン、コロンなど)を表現している。作成された N- グラムは独立した処理トークンとなり、検索可能となる。一つの単語から同一の N- グラムが複数回作成されることもあり得る。

4.4.1 History

N- グラムの最初の利用は、第 2 次世界大戦で暗号に使われたものに遡る。 Fltcher Pratt は次のように述べている。「2- グラムや 3- グラム表を利用すれば、誰でも単純な暗号を解読できる。」(Pratt-42) また、用語の合成法として 2- グラムを用いることが Adamson に述べられていた。 (Adamson-74) しかし、それはステミングの一般的な定義を導いてはいない。なぜなら、 N- グラムの生成するものは、意味のある語幹とは対照的に、単語の断片であるからである。データ構造の処理としてステミングよりもより適切に分類されているように思われるのは、この長い単語をより短い単語の断片にマッピングするという特徴によるものである。

N- グラム(特に 3- グラム)のもう一つの主要な利用は、スペルエラーの検出と修正の領域において行なわれている。 (Angell-83, McIllroy-82, Morris-75, Peterson-80, Thorelli-62, Wang-77, Zamora-81) ほとんどの研究は、英語の語彙における N- グラム(大抵は 3- グラムだが)の出現の可能性の統計的な値に注目し、 N- グラムに全く含まれないか、滅多に現れない単語を誤りを含む単語の可能性があるとしている。 Damerau はスペルエラーを図 4.8 に示すような 4 つの範疇に分類した。 (Damerau-64) Zamora は分類表を用いて、 3- グラムによる解析がミススペルや転換した文字列を認識できる有効なデータ構造を提供することを示した。このことは、情報システムにとっては正規化処理(第 1 章参照)中での訂正のために、入力誤りを判別するための基礎となった。 N- グラムパターンの頻出度は資料の言語を判別するのにも使われる。 (Damashek-95, Cohen-95)

情報検索においては、 3- グラムはテキスト圧縮 (Wisn-87) と索引用語の長さを操作する (Will-79, Sheck-78) ために用いられている。 D’Amore と Mah は実装した転置ファイルシステムの索引要素としてさまざまな異なる N- グラムを用いた。 (D’Amore-85) それらは、 SDI で利用される Logicon LMDS システム (Yochum-95) のためのプロファイルを符

合化する重要なデータ構造でもあった。遡及検索では、 Acquaintance システムが大規模文書データベース用の検索文書ファイルを蓄積するのに N- グラムを使用している。 (Damashek-95, Huffman-95)

4.4.2 N-Gram Data Structure

図 4.7 に示すように、 N- グラムは語を無視し、（語間記号によってその処理は任意に制限できるが）入力を連続したデータとして扱うデータ構造である。データ構造は固定長の重複した記号の断片からなり、それが検索可能な処理トークンを定義している。これらのトークンは、トークンが見つかった全ての資料への論理的なリンクを保持している。転置リストや（第 5 章で述べる）文書ベクトルやその他の保持しているデータ構造は、リンクデータ構造を蓄積するのに使われ、検索処理過程において利用される。中には頻出する N-グラムの最小限のものが、検索過程の第 1 段階として保持されているものもある。 (Yochum-85) このような実装の例は第 5 章で述べる。

固定長の単語片の大きさを決めることは、多くの文脈において研究されてきている。 Yochum と D'Amore は N の異なる値での影響を研究した。 Fatah Comlekoglu は転置ファイルを使った N- グラムデータ構造を、 N=2 から N=26 まで調査した。 3- グラム（長さ 3 の N-グラム）は、データ構造の大きさと対比して情報のトレードオフをとると、最適の長さであると結論付けられた。 Aquaintance システムは単語の境目を無視する、より長い N- グラムを使用している。 N- グラムの利点は、検索可能トークンにおける有限の限界を決定するというものである。

$$MaxSeg_n = (\lambda)^n$$

生成された N- グラムの一意のものの最大数 $MaxSeg$ は、 N- グラムの長さ n とアルファベットによる処理可能記号（語間記号ではない）の数 λ の関数として計算可能である。

一意の処理トークン数を減らすことや、限られた最小限の計算機上で高速の処理を行なう実装手法があるが、何らかの環境の下では誤った検索結果が起き得る。例えば、 3- グラムを使用し、語間記号や資料中の N- グラムの文字位置情報を含まないシステムでは、 “retail” の検索に対し、 “retain detail” を含む資料を見つけるだろう。（つまり、 “retail” に関連する 3- グラムの全ては “retain detail” の処理中に生成される）語間記号の導入もこの例は防げないだろう。 N- グラムの文字位置の導入によって、 “retail” を定義する “ret”, “eta”, “tai”, “ail” という N- グラムが全てそれぞれ 1 文字以内に連続的に存在していないということがわかるだろう。 N- グラムが長ければ長いほど、単語片における情報は多くなるので、この種の誤りは少なくなるようである。しかし、 N- グラムが長くなればなるほど、ほとんどの単語が一つの N- グラムに含まれるので、全単語のデータ構造と同じ結果を出すようになる。 N- グラムのもう一つの欠点は、リンクデータ構造を蓄積している転置リスト（または別のデータ構造）のサイズの増大である。実際に、 3- グラムの使用によって、もし語間記号を含まなければ、（図 4.7 によれば）処理トークン数は 5 の要素まで増大する。したがって転置リストは 5 つの要素まで増大する。

N- グラムデータ構造による処理トークンの限界から、資料を N- グラムの検索可能構造にマッピングすることや、検索質問の処理において、最適化された実行手法も適用することができる。ある特定の N- グラムには意味を持っていない。なぜなら、それは処理トーク

ンの断片であり、概念を表現したものではないからである。したがって、 N- グラムは概念とその関係をほとんど表現しない。しかし、 N- グラムの併用は標準的な単語索引と同等に使いうるものであり、実行時に顕著な改良をもって 85% 以内の適合率と同等の再現率を達成している。 (Adams-92) 資料の N- グラムによるベクター表現も資料間の類似度を計算するのに利用することが可能である。

4.5 PAT Data Structure

処理トークン間に語間記号を置く N- グラムを使うことは、テキスト入力を連続した N 文字のトークンで索引づけするデータ構造に等しいものとなる。断続的なテキスト入力のデータ構造を配置することへの異なる観点は PAT 木や PAT 配列に由来するものである。入力ストリームは部分文字列からなる検索可能なデータ構造に変換される。 PAT 木データ構造の元々の概念はパトリシア木として記述され (Flajolet-86, Frakes-92, Gonnet-83, Knuth-73, Morrison-69)、テキストや画像の検索 (Gonnet-88) や遺伝子データベースへの適用 (Manber-90) と、応用可能な構造として新たな力を得ている。 PAT という名前は PAtricia Trees の省略形である。（PATRICIA は Practical Algorithm To Retrieve Information Coded In Alphanumerics を意味する）

PAT 木の生成時には、入力文字列中の各ポジションはその位置から始まる部分文字列 (substring) のくさびとなるもので、入力の終端までの全ての新しいテキストを含んでいる。全ての部分文字列は unique なものである。テキストをこのように見ると、多くの異なる検索処理構造に適用できる。それは文書検索マシンや並列処理演算装置のハードウェアの一般的な構造によく適合するものである。（9 章参照）部分文字列はテキスト中のどこからでも開始でき、開始した位置とその長さによって一意に索引づけできる。もし、全ての文字列が入力の終端までのものであるなら、その長さは位置とアイテム全体の長さによって差が出るので、開始位置の情報だけが必要とされる情報となる。部分文字列に null 文字を付け加えて、入力文字列の長さを越えさせることもできる。これらの部分文字列は sistring(ssemi-infinite string) と呼ばれる。図 4.9 で入力テキストに対する sistring の例をいくつか示す。

PAT 木は sistring で定義される不均衡な、数字からなる 2 分木である。 sistring の個々のビットはゼロなら左へ、他は右へという分岐を決定している。 PAT 木は親ノードからツリー中の各ノードにビット位置やビット数で分岐を決定する際に、どちらのビットを使うかを指定することもできる。これは分岐のない階層を越えて移る際に有効である。

キーは PAT 木中の葉ノード（bottom ノード）に蓄積される。大きさ n のテキスト入力に対しては、 n 個の葉ノードと最大 $n-1$ の上位のノードが存在する。葉ノードにあたる sistring に制限を加えることもできる。 word の範囲に検索を限ることに関心があれば、例えば語間記号のすぐ後にある sistring に限るということもできるだろう。図 4.10 に PAT 木の生成に用いられる sistrings の一例を示す。 h が 100、 o が 110、 m が 001、 e が 101 という 2 進表現であるとき、 "home" という語は 100110001101 ...となるだろう。これらの sistring を用いた PAT 木を図 4.11 に示す。さらに中間ノードにスキップする値を入れて簡略化したツリーを図 4.12 に示す。ここでの（長方形で表されている）中間ノード中の値は、似た語と相違点があり比較を行なう次のビットまでのスキップするべきビット数である。この最終バージョンによって、スペースは節約できるが、スキップしたビットが検

索語にマッチしていることを保証するために検索値を（円の中の）葉ノードの中身と比較する必要がある。（つまりスキップしたビットは比較されない。）

検索語も 2 進表現で表され、 sistring を持つ PAT 木に対して適合するものを探す検索語を比較する。

2 章で触れたように、もっとも一般的な検索の種類の一つとして前方一致検索がある。PAT 木はこの目的のための理想的なものとして構成されている。なぜなら、各サブツリーは木構造中のそのノードまでで規定されている prefix に対応する全ての sistring を含んでいるからである。したがって、 prefix ノード以降の全ての葉ノードは前方一致検索の基準を満たす sistring を定義しているのである。このような PAT 木の論理的な並び方は、範囲値によって限られたサブツリーを決定することが容易なので、範囲検索をも助けるものとなる。もし、全入力文字列が PAT 木の生成に使われれば、後方一致、中間一致、固定長によるワイルドカードを用いた検索（2.1.5 参照）は全て容易である。なぜなら、与えられた文字列によって根ノードから適切な sistring の存在するノードへの道筋が一意に決定するからである。候補となるサブツリーの多数のものが検索語にマッチするので、曖昧検索は非常に困難となる。

PAT 木の検索および配列としてのその表現に関する詳細な議論は、 Gonnet、 Baeza-Yates、Snider によって行なわれている。 (Gonnet-92) Signature ファイルや転置ファイルに対する比較の結果、 Gonnet らは PAT 配列は Signature ファイルより正確性があり、転置ファイルにおいては効率の悪い文字列探索機能（後方一致検索、近接文字列検索、最長繰り返し検索）を有すると結論付けている。

文字列検索を目標とするならば、 PAT 木（と PAT 配列）は候補となりうる構造である。PAT 木（配列）はテキストを文字列操作を行なえる代わりとなる構造に蓄積する。その構造はより抽象的な概念やアイテムと結び付くそれらの関係を蓄積できない。その構造は興味深い応用への可能性を持ってはいるが、現時点では主要な商用の製品で使われていない。