全文検索システム Rast の設計と実装

西田雄也，橋本将*

2005 年 5 月 10 日

概要

全文検索に対する要求は，適合率と呼出率のいずれを優先するか，検索に用いる文字エンコーディングをなににするか，あるいはインデックス化に形態素解析を用いるか，N-gram を用いるかなどさまざまである．そのため，必ずしもそれらの要求の組み合わせを満たす検索エンジンが存在するとは限らない．我々は，全文検索処理を，必要な機能を網羅した全文検索ライブラリと，個別の要求に対応するモジュールに分割することでさまざまな要求に対応できる全文検索システム Rast を開発した．本論文では Rast の特徴と基本的なシステム構成および実装を概観し，その性能も評価する．

1 はじめに

インターネット上の情報や，電子メール，業務文書，文学作品など，日々蓄積されていく膨大な情報の中から目的の情報を探し出すために，全文検索システムは広く用いられている．

よく使用される全文検索システムの実装方法には，転置インデックスを用いる方法がある．転置インデックスによる検索システムには，分かち書き方式と N-gram 方式の 2 種類がある．しかし，それぞれに利点と欠点があるため，検索対象によって求められる実装方式は一つに決まらない．

そこで，我々は分かち書き方式や N-gram 方式などの検索方式に依存する部分を外部モジュールに切り出すことで，実装方式に依存せず，文書や属性の登録や検索，分散処理などの全文検索に必要な機能を網羅した全文検索システム Rast を開発した．

本論文では，全文検索システム Rast の特徴及び設計，実装についてまとめる．また，処理性能を他の全文検索システムと比較し，その実用性を述べる．

* 株式会社ネットワーク応用通信研究所

2 検索方式の概要

全文検索には，文書内の文字を一文字ずつ照合して探し出す逐次検索と，あらかじめ検索対象文書を走査して作成したインデックス（索引）を用いる索引検索の 2 種類がある．一般に扱う文書の規模が大きく，記憶領域よりも検索時間が優先される用途には索引検索が選ばれる．

索引検索には，文書内に現われた索引語の特徴をインデックスに持つ特徴ベクトル法と，検索対象文書の半無限文字列を登録したパトリシア木を使う方法，索引語－出現文書の表構造のインデックスを持つ転置インデックス法などがある．

転置インデックス法には，索引語に N-gram を用いる N-gram 方式と分かち書きによって分割した単語を用いる分かち書き方式の二つがある．

分かち書き方式は，自然言語に応じて，単語のステミングを行ったり，品詞によってスコアの重みを変えられるといった利点がある．このため，インターネット上の情報のように膨大な文書で適合率が重視される場合によく用いられる．

それに対し，N-gram 方式は，対象となる文書の分野や言語を選ばずに利用できる．しかも，分かち書き方式に見られる辞書の整備が不要といった利点がある．このため，呼出率が優先される特許の検索

や古典文学での検索に用いられることが多い．

全文検索システム Rast では転置インデックス法を採用し，インデックスに使用する索引語切り出しルーチンを外部モジュール化することで，検索方式を選択可能にしている．

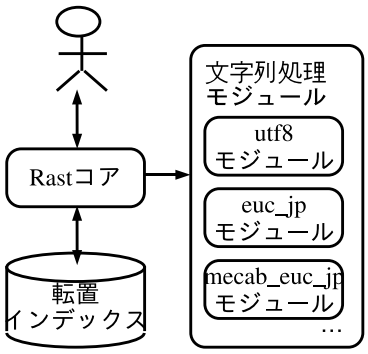

図 1　概要図

3　Rast の特徴

Rast は文字の切り出しや索引語の分割処理，文書の正規化といった文字エンコーディングに依存した文字列処理を外部モジュールにした．設定によって文字列処理モジュールを使い分けることで，分かち書き方式と N-gram 方式などの検索方式や様々な文字エンコーディングに対応する．

また，全文検索ライブラリとしてアプリケーションに組み込むことで，登録や検索，削除などといった操作をアプリケーションから呼び出して使うことができる．

Rast は個人用途や小規模，中規模 SI 向けの全文検索環境として設計している．そのため，大規模な検索対象においては，検索データベースを複数に分割して対応する多階層メタ検索を実装した．

4　Rast の構成

4.1　全文検索ライブラリ

全文検索ライブラリは全文検索システムにおいて共通な処理である隣接判定，スコアリング，要約表示などの機能を提供する．

4.1.1　検索概略

Rast での検索時の概略を以下に示す．

1. 検索質問の解析を行い，検索質問全体から各検索条件を取り出す．このとき，文字列処理モジュールを用いて検索文字列用の正規化を行い，一文字ごとに処理を行う．
2. 各検索条件ごとに該当する文書を得る．全文検索時の動作とプロパティ検索時の動作は 4.1.2 節，及び 4.1.3 節で述べる．
3. 得られた各検索条件ごとの検索結果をマージする．
4. 検索結果中から除外された文書 ID を取り除く．
5. オプションで指定されていれば，検索結果文書それぞれに対し，スコアリングを行う．また，指定されたプロパティ値を得る．
6. 検索結果文書を指定された方法でソートする．
7. オプションで指定されていれば，要約テキストを取得し，検索質問周辺の要約を得る．

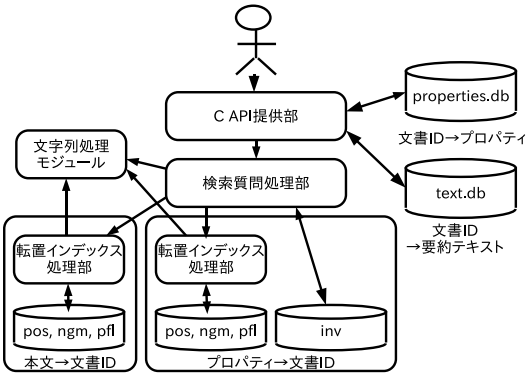

図 2　検索処理概要図

4.1.2　全文検索

Rast の全文検索時の動作を以下に示す．

1. 検索語を文字列処理モジュールを用いて索引語に分割する．
2. 各索引語ごとに格納場所情報ファイルから引き，索引語位置情報ファイルの該当箇所を読み取り，索引語が出現する文書 ID と位置情報を得る．

3. 索引語の最適化（4.1.6 節）を行う．
4. 各索引語同士の隣接判定を行う．
5. 結果を返す．

なお，索引語が格納場所情報ファイルになければ，低頻度索引語位置情報ファイルから文書 ID と位置情報を得る．Rast では，低頻度の索引語を特別に扱うことで，索引語位置情報ファイルのファイルサイズを抑えている．低頻度の索引語とは全文書中で 1 文書のみでかつ，1 箇所にしか存在しない索引語としている[*1]．

4.1.3 プロパティ検索

Rast には，本文とは別にその文書が持つメタデータをプロパティとして登録できる．プロパティは，検索時の文書の絞り込みや検索結果の追加情報として使用する．なお，プロパティのみでの検索も利用可能である．

プロパティには文字列，数値，日付といった格納形式があり，プロパティ検索には，以下の検索方法がある．

- 部分一致検索
- 大小比較検索
- 完全一致検索

部分一致検索は，全文検索フラグが立っている文字列プロパティで使用可能である．ファイルにはプロパティ名.ngm,pos,rng といったファイルを使用する．各ファイル構造は本文の全文検索時に使用するファイル構造と同じであり，本文の全文検索（4.1.2 節）と同じ方法によって文書を得る．

大小比較検索は，データベース作成時に逆引き可能フラグを立てたプロパティで使用可能である．検索には Berkeley DB[5] の B 木であるプロパティ名.inv ファイルを使用する．例えば，プロパティ値がある値 A よりも大きい文書を得る場合は以下のようになる．

1. プロパティ名.inv ファイルから A の位置にカーソルを設定する．
2. A を越えるまで次の位置にカーソルを進める．
3. カーソルを最後まで進めながら文書 ID を取得する．

また，プロパティ値がある値 A よりも小さい文書を得る場合は以下のようになる．

1. プロパティ名.inv ファイルの先頭位置にカーソルを設定する．
2. キーが A と等しいかより大きくなるまでカーソルを進めながら文書 ID を取得する．

完全一致検索は，大小比較検索と同様に逆引き可能フラグを立てたプロパティで使用可能である．検索もまた，大小比較検索と同様にプロパティ名.inv ファイルを使用する．例えば，プロパティ値がある値 A と等しい文書を得る場合は，$A \leq$ プロパティ値 $\leq A$ として大小比較検索を行う．

4.1.4 登録

全文検索システム Rast での登録時の概略を以下に示す．

1. ユニークフラグが立っているプロパティが重複登録されるか否か調べる．
2. 各プロパティ値の逆引き[*2]の登録を行う．逆引き可能フラグが立っているプロパティはプロパティ名.inv ファイルに登録する．全文検索フラグが立っているプロパティは検索用の正規化を行った後で，プロパティの全文検索インデックスに登録する．
3. 本文を文字列処理モジュールに渡し，要約用の正規化を行う．
4. 要約テキストの登録を行う．
5. 要約テキストを文字列処理モジュールに渡し，検索用の正規化を行う．
6. 検索用テキストを本文の全文検索インデックスに登録する．
7. 全文検索フラグが立っているプロパティ値を本文の全文検索インデックスに登録する．

[*1] Rast 0.1.0 現在

[*2] プロパティ値から文書 ID を得ること

8. 各プロパティ値の正引き[*3]の登録を行う．

4.1.5 ファイル構造

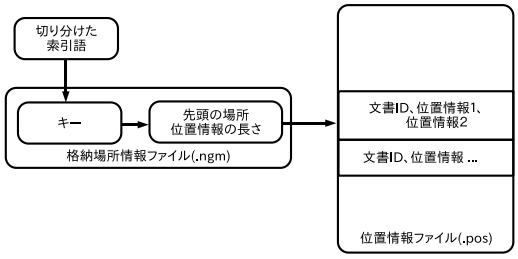

図 3　位置情報ファイル構造図

Rast が全文検索とプロパティ検索のために使用するファイルは，大まかに以下の四つに分類できる．

1. 設定ファイル
2. 本文の全文検索用インデックス
3. プロパティ値検索用インデックス
4. 要約テキスト用データベース

text.ngm（text.pos 中の格納場所情報ファイル）
位置情報の text.pos での格納場所を管理している Berkeley DB の B 木データベースである．索引語をキーに以下の情報が値として格納されている．

1. ブロック番号（ブロック番号×ブロックサイズが，text.pos 中の文書 ID と位置情報の場所となる．）
2. ブロック数（text.pos 中に書き込まれているブロックの数）
3. バイト数（対象索引語の全文書 ID と全位置情報のバイト数）
4. 文書数（対象索引語が格納されている文書数）

text.pos（索引語位置情報ファイル）　各索引語が出現した文書 ID と文書中の出現位置を格納するファイルである．以下の情報が，設定ファイルに書かれているブロックサイズごとに，繰り返し格納されている．

1. 文書 ID（1 から始まる文書ごとに一意な ID）
2. 出現位置情報のバイト数
3. 索引語の出現位置情報（文字単位）

text.pfl　text.pos ファイルの削除済み領域を管理するファイルである．新しい文書の文書 ID と位置情報を登録し続けると，text.pos ではブロックの移動が発生し，これまで格納されていた領域は空き領域となる．この空き領域を新しくブロックの確保が発生したときに使用する領域にするため，空き領域の

1. ブロック番号
2. ブロック数

が格納されている．

text.rng（低頻度索引語位置情報ファイル）　出現数が少ない索引語の文書 ID と位置情報を格納するファイルである．索引語をキーに以下の情報が値として格納されている．

1. 文書 ID
2. 出現位置情報のバイト数
3. 出現位置情報（文字単位）

4.1.6 隣接判定

全文検索システム Rast では，フレーズ検索を行うために分割した索引語の位置情報を格納し，索引語同士が並んでいるか否かを判定する．

全文検索システム Rast では，分割した索引語ごとに位置情報を格納している．そのため，検索語が複数の索引語に分かれる場合，索引語同士が並んでいるか否かを判定する．

なお，隣接判定前に以下の方法により索引語の選択を行うことで隣接判定の最適化をしている．

1. 基本的に先頭索引語から重ならない索引語を隣接判定に使用する
2. 出現文書数最少の索引語は重なっても使用する
3. 最後の索引語は重なっても使用する

4.1.7 スコアリング

検索質問によっては大量の検索結果が得られることがある．そのような場合に，検索質問に合致した

[*3] 文書 ID から各プロパティ値を得ること

文書を検索結果の上位にすることで，目的の文書をより探しやすくできる．この工夫をスコアリングと言い，全文検索システムにおいては欠かせない機能の一つである．Rast では，検索結果のスコアリングに検索条件による TF（索引語頻度）-IDF （大域的頻度）重み付けを行い，検索結果をソートしている．

具体的には，以下の計算式にて算出した数値を検索条件に対する文書のスコアとする．次に検索条件ごとのスコアの合計を百万倍し[*4]，検索質問全体に対する文書のスコアとして使用する．

$$TF = \frac{\text{検索条件の文書中の出現数}}{\text{文字数}}$$
$$IDF = log_{10}\frac{\text{全文書数}}{\text{検索条件が出現する文書数}} + 1$$
$$TF - IDF = TF * IDF$$

4.1.8 要約表示

全文検索システムの評価として，検索結果の見やすさも挙げられる．Rast では，登録時に要約を登録するオプションを付与し，検索時にも要約を取得するオプションを付与すれば，検索結果に検索質問に合致した周辺テキストも得ることができる．

Rast 0.1.0 において，周辺テキストとは最初の検索条件の最後の文字の前後から，指定した文字数分のことを意味する．周辺テキストの取得には，検索時に索引語位置情報ファイルから得た文字単位の位置情報を使用している．

なお，プロパティ中の全文検索で得られた場合は要約テキストの最初の位置を周辺テキストの開始位置とする．

4.2 文字列処理モジュール

Rast では，索引作成時や検索時に入力テキストを処理する際に，文字列処理モジュールと呼ばれる外部モジュールによって，文字や索引語の切り出しを行う．このため，ユーザは，文字列処理モジュールを差し替えることによって，UTF-8・EUC-JP といったさまざまな文字エンコーディングや，N-gram 方式・分かち書き方式といった複数の方式から，用途にあったものを選択することができる．

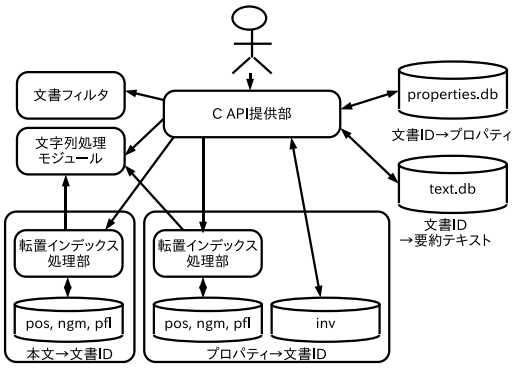

図 4　登録処理概要図

また，文字列処理モジュールは，Rast 本体に静的にリンクされるのではなく，実行時に dlopen(3) などを用いて動的にリンクされるプラグイン構造を取っている．このため，サードパーティによる文字列処理モジュールの提供が容易になっている．

文字列処理モジュールは，モジュール名と同名のライブラリ（`utf8` モジュールの場合，`utf8.so`）として用意され，「rast_encoding_モジュール名」という名前の rast_encoding_module_t 構造体型の変数を持つ．この構造体には，文字列処理モジュールが提供する関数へのポインタが格納されており，必要に応じて Rast 本体から呼び出される仕組みになっている．

具体的には，以下のような関数が文字列処理モジュールによって提供される．

1. 現在の位置の文字のバイト数を取得する関数
   rast_error_t *get_char_len(rast_tokenizer_t *tokenizer, rast_size_t *len)
2. 現在の位置の索引語を取得する関数
   rast_error_t *get_token(rast_tokenizer_t *tokenizer, rast_token_t *token)—
3. 次の索引語のバイト数のオフセットと文字数のオフセットを取得する関数
   rast_error_t *get_next_offset(rast_tokenizer_t *tokenizer, rast_size_t *byte_offset, rast_size_t *char_offset)

[*4] 計算機で扱い易い整数値にするため

4. 要約表示のための文字数が変化する正規化を行う関数（全角→半角変換処理などを行う）
   void normalize_text(apr_pool_t *pool, const char *src, rast_size_t src_len, char **dst, rast_size_t *dst_len)
5. 検索のための文字数が変化しない正規化を行う関数（大文字→小文字変換処理などを行う）
   void normalize_chars(apr_pool_t *pool, const char *src, rast_size_t src_len, char **dst, rast_size_t *dst_len)
6. 指定した文字が空白文字を表すか否かを判定する関数
   int is_space(rast_char_t *ch)

現在，Rast では，以下の三つの文字列処理モジュールが提供されている．

4.2.1 utf8 文字列処理モジュール

utf8 文字列処理モジュールは，文字エンコーディングに UTF-8 を，索引語の分割に N-gram 方式を採用した文字列処理モジュールである．

原田氏らの研究 [1] に基づき，N は UNICODE[6] の文字ブロックに応じて可変とし，文字ブロックの切り替えが発生する箇所は bi-gram で切り出す方式を採用している．具体的には，ラテン言語，ギリシア文字，キリル文字，平仮名，片仮名は tri-gram に，それ以外は bi-gram にした．また，フランス語などでは，一つの単語に異なる文字ブロックの文字が含まれるため，文字ブロックの切り替えが発生する箇所が bi-gram となってしまう．このため，Basic Latin と Latin-1 Supplement などの一部の文字ブロックは同一視して N を決定するようにした．

4.2.2 euc_jp 文字列処理モジュール

euc_jp 文字列処理モジュールは，文字エンコーディングに EUC-JP を，索引語の分割に N-gram 方式を採用した文字列処理モジュールである．

utf8 文字列処理モジュールと同様に，文字種に応じて N の値を可変とし，異なる文字種にまたがる部分は bi-gram として切り出しを行う．

モジュール名	utf8	mecab_euc_jp
関数の数	13	11
行数	433	416

表 1　モジュール作成コスト評価（utf8 モジュールと mecab_euc_jp モジュール）

4.2.3 mecab_euc_jp 文字列処理モジュール

mecab_euc_jp 文字列処理モジュールは，文字エンコーディングに EUC-JP を，索引語の分割に MeCab による分かち書き方式を採用した文字列処理モジュールである．

隣接判定のために，すべての索引語が入力テキスト全体を被覆する必要があるため，空白や記号などもすべて索引語としての切り出しを行っている．

5 評価

5.1 モジュール作成コスト

実際に N-gram 方式を実現する文字列処理モジュールのソースコード utf8.c と，MeCab[*5]による分かち書き方式を実現する文字列処理モジュールのソースコード mecab_euc_jp.c の行数と定義された関数の数を計測した．

これより，C 言語としては比較的少ないコストで複数の検索方式が実現できることがわかる．

5.2 性能評価

5.2.1 計測方法と結果

Rast の utf8 モジュールと mecab_euc_jp モジュールを用いた検索が実用的な速度で登録や検索を行える事を検証するため，Rast と既存の完成された検索システムである Estraier[8] と Namazu[7] との比較を行なった．登録と検索対象にはメーリングリスト ruby-list@ruby-lang.org 及び ruby-dev@ruby-lang.org に投稿された 63126 件のメール本文を使用した．

登録時間の計測では，63126 件のメーリングリスト全件を登録したデータベースの構築時間を計測

[*5] 奈良先端科学技術大学院大学自然言語処理学講座の開発する形態素解析器 ChaSen を基に開発された高速な形態素解析器 http://chasen.org/ taku/software/mecab/

CPU	Pentium 4 3GHz
メモリ	1GB
ハードディスク	Ultra ATA 100

表 2　計測に使用したマシン

した．また，63126 件の登録後に新たに一件追加登録し，その登録時間を計測した．検索時間の計測では，「Ruby」，「tk」などの単語で 5 回検索し，その平均を検索時間とした．各検索システムのデータベース構築時間と追加登録時間を表 3 に，検索時間を表 4 に示す．また，計測に使用したマシンを表 2 に示す．

表中において，cold start とは索引ファイルがメモリ上に全く読み込まれていない状態で検索を行うことであり，hot start は索引ファイルのデータがキャッシュに載っている状態で検索を行うことである．また，「N/A」と表記されている部分は使用した検索語において適合した文書が多すぎる事から．検索に使用した検索システムでは検索できなかったことを表している．なお，Estraier はバージョン 1.2.28 を、Namazu はバージョン 2.0.14 を使用した．

5.2.2　考察

63126 件の登録時間は Rast よりも Estraier が 3.5 倍程度速いという結果になった．また，その後の追加登録時間は Rast と Estraier で大差ないことが確認できた．検索時間は検索語が「a」の場合を除き，問題ない時間で検索できることがわかった．

大量の文書登録は，実際にはシステムが導入される最初しか行わない．追加登録時間が既存の全文検索システムと大差ないため，運用上は実用的であると考える．なお，Rast は N 未満の検索語については，索引語位置情報ファイルから前方一致検索を行い，複数の索引語を使用する．そのため，検索語が「a」の場合は文書 ID の比較候補が非常に多くなるため、検索に時間がかかっている。多くの文書に含まれる検索語は無視しても問題ないことが多い，既存の全文検索システムのように多すぎる旨のメッセージを出すなどして無視することが考えられる．

6　今後の展望

現状の Rast では文字エンコーディングごとの処理の違いと検索方式の違いを一つの文字列処理モジュールで記述するため，`mecab_euc_jp` 文字列処理モジュールと `euc_jp` 文字列処理モジュールは同一の処理が存在する．

そのため，将来は文字エンコーディングの処理を別モジュールにして，検索方式の違いを吸収するモジュールが，関数テーブルを継承することで共通化することが考えられる．

また，今後は類似文書検索，正規表現検索など検索システムとしての機能をより高めるとともに，様々なファイル形式の文書からテキストを抽出する文書フィルタの整備やインデックスサイズの最適化，トランザクション対応，さらなるモジュールの分割などでより実用性を高めるよう改善したい．

7　おわりに

本論文では，全文検索システム Rast について，基本的な枠組みを提供するフレームワークと，個別の要求に対応するモジュールが分割していることで，全文検索のさまざまな要求に容易に対応できることを述べた．また，Rast の設計と実装についてまとめ，モジュールが分割されていても実用的な性能で検索や登録ができることを実証した．

なお，Rast は http://www.netlab.jp/rast/で公開している．Rast をアプリケーションに組み込めば，アプリケーションに全文検索機能を提供することもできる．

諸氏が開発するシステムで全文検索が必要となったとき，Rast がその手助けとなれば幸いである．

謝辞

全文検索システム Rast は独立行政法人 情報処理推進機構（IPA）の平成 16 年度オープンソフトウェア活用基盤整備事業の委託を受けて開発を行っています．

検索システム	Estraier	Namazu	Rast(N-gram utf8)
6 万件登録時間 (sec)	156.19	626.09	531.33
1 件追加登録時間 (sec)	8.699	23.808	7.938018
データベースサイズ	139Mb	116Mb	324Mb

表 3　性能評価 文書登録時間 (登録文書数 63126 件)

検索システム	Estraier	Namazu	Rast(N-gram utf8)
検索語	Ruby		
適合文書数	29547	N/A	50066
cold start (sec)	0.4318912	0.0347416	0.665318
hot start (sec)	0.076197	0.0010586	0.5553598
検索語	tk		
適合文書数	2232	612	4739
cold start (sec)	0.3576128	0.3562174	0.51231968
hot start (sec)	0.0183802	0.0346768	0.1419772
検索語	tcl/tk		
適合文書数	338	109	522
cold start (sec)	0.3760914	0.2879038	0.557817832
hot start (sec)	0.016116	0.0080904	0.019303
検索語	emacs		
適合文書数	653	94	863
cold start (sec)	0.330301	0.2961386	0.2069632
hot start (sec)	0.0110054	0.0076692	0.0129308
検索語	a		
適合文書数	N/A	3045	61918
cold start (sec)	0.2709114	0.6318622	15.1325726
hot start (sec)	0.009056	0.180159	10.3322752
検索語	問		
適合文書数	N/A	86	12991
cold start (sec)	0.260278	0.4412032	0.9149052
hot start (sec)	0.008311	0.0063548	0.2162194

表 4　性能評価 文書検索時間 (登録文書数 63126 件)

参考文献

[1] 原田昌紀, 風間一洋, 佐藤進也: “Unicode を用いた N-gram 索引の一実現方式とその評価”, 情報処理学会研究会報告, 2000-NL-136-17, 2000.

[2] 小川泰嗣, 山本研策, 真野博子, 伊東秀夫: “全文検索システムのための複数転置ファイルを用いた登録高速化とランキング検索”, DEWS2002, 1999.

[3] 徳永健伸: “情報検索と言語処理”, 東京大学出版会, 1999.

[4] 北 研二，津田 和彦，獅々堀正幹: “情報検索アルゴリズム”, 共立出版, 2002.

[5] “Sleepycat Software: Products: Berkeley DB”, http://www.sleepycat.com/products/db.shtml

[6] “Unicode Home Page”, http://www.unicode.org/

[7] “全文検索システム Namazu”, http://www.namazu.org/

[8] “Estraier: a personal full-text search system”, http://estraier.sourceforge.net/