v1.1

2014 年度 メディア情報学実験 2 「情報検索」テキスト

担当: 前田 亮

1. 実験の目的

本実験では, Java プログラミング環境を用いて情報検索システムを構築する手法を習得する. まずは Web 文書を対象に, 索引付け, 問合せ処理, 検索結果のランキングの各手法について習得し, その後実際にこれらを応用した検索エンジンを作成する. また, 応用として, Wikipedia 上の人物の関係を可視化するシステムを構築する.

2. 情報検索システムの概要

情報検索システムは, 利用者が入力した問合せ(検索語, キーワード)に対して適合する文書を探し, その文書を利用者に提示する. 入力した問合せに適合する文書とは, 利用者が求めている内容について記述された文書のことである. 各文書に書かれている内容を推定するには, その文書にどのような単語が出現するかを調べればよい. つまり, 問合せとして入力した単語の出現頻度が高い文書は, 入力した問合せに適合しているとみなすことができる.

上記のようなシステムを実現するためには, 各文書でどのような単語が出現するのか, 事前に集計しておく必要がある. この集計結果を索引(インデックス)として保存しておき, 実際に検索を行う際にはこの索引を用いて入力した問合せに適合する文書を探す. すなわち, 情報検索システムを構築するためには, 以下の 2 つの処理を行う必要がある.

・事前処理: 索引の作成(出現単語の集計)
・検索処理: 問合せに適合する文書を探す(問合せと各文書の類似度を計算)

本実験では, これらの処理を行うプログラムを Java 言語により実装し, 情報検索システムの構築を行う.

3. 実験内容

本実験は, 以下の手順で行う. 週の割り当てはあくまで目安であり, 必ずしもこの通りに進める必要はない. ただし, 第 7 週終了までに, 必ずすべての必須課題を終える必要がある.

第 1 週 Java の基礎の復習
第 2 週 HashMap(必須課題 1)
第 3 週 形態素解析(必須課題 2)
第 4 週 索引付け(必須課題 3)
第 5 週 簡単な検索エンジンの作成(必須課題 4), 検索結果のランキング(必須課題 5)
第 6 週 人物関係の可視化(必須課題 6, 7)
第 7 週 索引のデータベースへの格納(発展課題 1)

4. レポートと評価

レポートには以下の内容を必ず記載すること. 内容に不備がある場合は再提出を求める.

- 表紙(以下の項目を必ずすべて記載すること)
  レポートのタイトル:「メディア情報学実験 2(情報検索)」
  提出年月日(「月日」だけでなく「年」も必ず記載すること)
  所属クラス, 学生証番号, 氏名
- 必須課題 1〜6 のそれぞれについて, 以下の内容を記載すること.
  (ただし, 必須課題 2 については, 2–2 の内容だけで良い. また, 必須課題 4 については, 1. と 2. それぞれについて以下の内容を記載すること)
  1. プログラムのソースコード(サンプルからの追加・変更部分が分かるようにコメントを付けること)
  2. プログラムの追加・変更部分の説明(どのような処理を行っているかを必ず文章で説明すること)
  3. 実行結果(100 行以上になる場合は, 先頭から 100 行分のみを記載すれば良い)
- 必須課題 7 の実行結果(実行結果が表示されたウィンドウのスクリーンショット)
- (発展課題 1 を行った場合のみ)発展課題 1 の実行結果(実行結果が表示されたウィンドウのスクリーンショット), 考察
- (オプション課題を行った場合のみ)オプション課題のソースコード(サンプルからの変更部分が分かるようにコメントを付けること), プログラムの説明, 実行結果(100 行以上になる場合は, 先頭から 100 行分のみを記載すれば良い), 考察
- 本テーマ全体の考察・感想
- 参考文献, URL

上記の内容を PDF 形式の電子ファイルとして作成し, manaba+R にて提出すること.

5. 実験の準備

5.1. USB メモリ

本実験では, 索引として大容量のデータを扱うため, ホームディレクトリの容量制限(100MB)をオーバーする可能性がある. 特に発展課題(12 章)ではデータベースを扱うため, 容量制限をオーバーする可能性が高い. したがって, **発展課題(12 章)を行う場合は, 必ず各自で USB メモリを持参すること**. 容量は 1GB あれば十分である.

5.2. 実験環境

本実験は, メディア情報学科実験室 2(CC502)の Linux 環境で行う. 2 回生前期のメディアプロジェクト演習 1 の Java 演習と同様に, テキストエディタでソースプログラムを作成し, コマンドライン上でコンパイルおよび実行を行う. Java 開発環境は JDK1.6 を用いる. また, 形態素解析ソフトとして Sen, グラフ描画ソフトウェアとして Jung,

データベース管理システムとして MySQL をそれぞれ用いる. これらはいずれもオープンソースであり, 無料で入手できる. 本実験では, あらかじめ教材フォルダにインストールされているものを用いる.

5.3. 環境変数の設定

実験を開始する前に, まず以下のように環境変数の設定を行う. 内容を書き換える前に, 既存のファイルをバックアップしておく.

```
% cp ~/.cshrc ~/.cshrc.bak
```

次に, 以下のコマンドを実行することで, ~/.cshrc の末尾に環境変数の設定部分が自動的に追加される.

```
% /kyozai/amaeda/ir/scripts/irSetup.sh
```

.cshrc に書いた設定は新しくターミナルを開いた場合は自動で読み込まれるが, **はじめて変更した直後には読み込まれないので, 以下のように入力することで現在のターミナルに読み込ませる.**

```
source ~/.cshrc
```

また,
echo $SEN_HOME
で正しく設定できているか確認するとよい.
/kyozai/amaeda/ir/sen が出力として得られれば, 正しく設定できている.

6. Java 言語

6.1. HashMap クラス

2 章で述べたように, 情報検索システムの構築においては, どの文書にどのような単語が何回出現したかを集計し, 索引として保存しておく必要がある. 出現単語の集計を行うには, **HashMap クラス**を使うのが便利である. HashMap クラスは, Java 言語においてハッシュ構造およびそれに必要となる機能を提供する(注: Map とは, キーと値が対になった要素をもつ構造のこと. HashMap クラスのほか, TreeMap クラスや LinkedHashMap クラスがある). HashMap では, 格納したいデータの格納位置を, 格納したいキーをハッシュ関数により格納位置に変換することで決定する. 格納されたデータを参照する場合も, 同様にして格納位置を特定する. これにより, 格納しているデータ数の多少にかかわらず, データの格納・参照の操作の際に HashMap へのアクセスは 1 回か数回で済む. この特徴は, 格納しているデータにランダムアクセスする場合に向いている.

Java 言語において HashMap クラスを使用する際には，java.util パッケージをインポートする．HashMap クラスのインスタンス化は，以下のようにして行う．

```
HashMap<型 1,型 2>  変数名 = new HashMap<型 1,型 2> ();
```

「型 1」はキーの型，「型 2」は値の型を指定する．<String, Integer>などのように，キーと値の型を指定する記述は「ジェネリックス」と呼ばれる．このとき注意しなければならないのは，基本型（int 型，double 型など：（注）String 型はクラス型）を指定することはできないことである．int 型，double 型のかわりに，それぞれ Integer 型，Double 型を指定しなければならない．なお，HashMap のサイズは，格納データの増加に伴い適宜拡張される．

HashMap にデータを格納するには，put()メソッドを用いる．引数としてキーと値の組を与える．以下に，キー，値ともに文字列型の場合の例を示す．

```
変数名.put("key", "value");
```

追加しようとしたキーがすでに HashMap に登録されている場合，古いデータは上書きされ，put()メソッドにより追加されたデータに置き換えられる．

HashMap のデータを参照（取り出し）するには，get()メソッドを用いる．引数としてキーを与えると，それに対になっている値を戻り値として受け取る．get()メソッドで値を取得する前に，指定したキーがその HashMap に存在するかを確認する必要がある．その際には containsKey()メソッドを用いる．引数にキーを与えると，それが HashMap にある場合は true，ない場合は false（つまり戻り値は Boolean 型）が返される．containsKey()メソッドで調べた結果 true であった場合に，get()メソッドを用いる．

```
if (変数名.containsKey("key")){
  System.out.println(変数名.get("key"));
}else{
  System.out.println("指定したキーは存在しません");
}
```

次のソースプログラム（HashMapSample1.java）は，HashMap クラスを用いた日英簡易辞書システムである．辞書ファイル（dicsample.txt）を読み込み HashMap に格納しておき，訳語を知りたい日本語をキーとして HashMap を参照し，それと対になる値を出力することで，訳語を提示する．

（/kyozai/amaeda/ir/の下にも同じものがあるのでコピーして使用するとよい．また，辞書ファイルである dicsample.txt もコピーしておくとよい．）

```
//
//
// HashMap サンプル
// 簡易辞書システム
//
// args[0]: 辞書ファイル
```

```
import java.io.*;
import java.util.*; // HashMap を使う際に import

public class HashMapSample1 {

   HashMap<String, String> dic_table; //辞書を格納する HashMap
   String dic_file;

   // コンストラクタ
   HashMapSample1(String dic) {
      dic_table = new HashMap<String, String>(101); // dic_table のインスタンス化
      dic_file = dic;
   }

   void readFile(HashMap<String, String> hm, String infile) {
      try {
         BufferedReader br = new BufferedReader(
                     new InputStreamReader(
                     new FileInputStream(infile), "UTF-8"));

         String line;
         while( (line = br.readLine() ) != null ){
            String[] term = line.split("\t"); //日本語と英語に分割
            hm.put(term[0], term[1]);
         }
      }
      catch(Exception e) { System.out.println("I/O error at readFile \n" + e); }
   }


   // HashMap の内容を出力
   void outputHashMap(HashMap<String, String> hm) {
      Set keys = hm.keySet();
      for( Iterator itr = keys.iterator(); itr.hasNext(); ) {
         String jterm = (String)itr.next();
         String eterm = (String)hm.get(jterm);
         System.out.println(jterm + ": " + eterm);
      }
   }


   public static void main(String[] args) {
      HashMapSample1 hms = new HashMapSample1(args[0]);
      String jterm = "検索";  //調べたい単語
      String eterm = "";      //jterm の訳語を格納する変数

      hms.readFile(hms.dic_table, hms.dic_file);

      if ( hms.dic_table.containsKey( jterm ) ) {
         eterm = (String)hms.dic_table.get(jterm);
         System.out.println(jterm + " の訳語は " + eterm + " です。");
      }
      else
      {
         System.out.println(jterm + " の訳語は辞書にはありません。");
      }

      //hms.outputHashMap(hms.dic_table); //dic_table(HashMap)の内容を出力
   }
```

```
} //class HashMapSample1 end
```

このプログラムを実行すると，次のようになる．

```
% java HashMapSample1 dicsample.txt
検索 の訳語は retrieval です。
```

なお，<String, Integer>型の HashMap を用いたサンプル（HashMapSample2.java: 家計簿の集計プログラム）が/kyozai/amaeda/ir/ の下にあるので，このプログラムも実行してみること．

6.2. 各種メソッドの説明（split, parseInt）

サンプル HashMapSample1.java では，日本語の単語と英語の単語がタブで区切られた文字列（String クラスのインスタンス）としてファイルから読み込まれているため，タブを境目とした分割を行う必要がある．これには String クラスが持つ **split メソッド**を使用する．split メソッドに引数として「区切り文字」を与えると，戻り値として分割結果の配列（String クラス）を返す．この分割結果の配列の要素数は，自動的に「区切り文字数+1」となる．HashMapSample1.java の場合，term[0]に日本語，term[1]に英語が格納される．

また，サンプル HashMapSample2.java では，String クラスのインスタンスから int 型の値を得る必要がある．この場合，Integer クラスが持つ **parseInt メソッド**を使用する．サンプルの Integer.parseInt(data[2])という命令では，String 型の data[2]（数値を文字列で表したもの）を引数として与え，戻り値としてそれが表す int 型の数値を得ている．

【必須課題 1】 形態素解析を行った結果が記録されているファイル（morph.txt: /kyozai/amaeda/ir/ の下にある）を読み込み，単語の出現頻度を集計し，ファイル term_frequency.txt に集計結果を出力するプログラムを作成せよ．なお，/kyozai/amaeda/ir/ の下にある HashMapSample3.java に，出現頻度の集計以外の部分のプログラムが用意されているので，これを基に作成してもよい．

実行結果の例（term_frequency.txt）：

```
中村: 1
2008: 2
2009: 1
2006: 2
カップ: 5
2007: 1
2004: 1
2005: 2
年生: 3
最終: 3
さらに: 1
決め: 3
等: 1
評価: 2
エヴァートン: 1

（以下略）
```

7. 形態素解析

索引を作成するためには，どの単語がどの文書に出現するかを調べなければならない．そのためには，文書中の文章を単語に分割する必要がある．英語などでは単語ごとにスペースで区切られているが，日本語では単語の区切りが明確になっていない．日本語の単語分割を行うには，形態素解析器を用いる．

形態素解析(morphological analysis)とは，与えられた文章を単語に分割し，その単語の品詞を判別することである．形態素解析器の一つに Sen がある．Java で Sen を使いファイルに記録された文章を形態素解析するサンプル SenSample.java を以下に示す（/kyozai/amaeda/ir/の下にも同じものがある）．SenSample.java を javac でコンパイルする際,「推奨されない API を使用またはオーバーライドしています」という警告が出るが, エラーではなく警告であるので問題ない.この内容について知りたい場合は, 以下のように-Xlint:deprecation オプションを付けてコンパイルする．

```
javac -Xlint:deprecation SenSample.java
```

```
//
// 形態素解析の実行プログラム
//
// args[0]: 解析したい文章が保存されたファイル
//

import java.io.*;
import java.util.*;
import net.java.sen.StringTagger;
import net.java.sen.Token;

public class SenSample {
     String docfile; //解析したい文章を保存したファイル（読み込み）
     String outfile; //解析結果を出力するファイル（書き込み）
```

```
	//コンストラクタ
	SenSample(String inname) {
		docfile = inname;
		outfile = "sen_out_samp.txt";
	}


	void morph(String docfile, String ofile) {
		try {
			// 形態素解析を行う StringTagger を作成
			StringTagger tagger = StringTagger.getInstance(Locale.JAPANESE);

			// 入力元のファイルを指定し，BufferedReader を作成
			BufferedReader br = new BufferedReader(
					new InputStreamReader(
					   new FileInputStream(docfile), "UTF-8"));
			BufferedWriter bw = new BufferedWriter(
					new OutputStreamWriter(
					   new FileOutputStream(ofile), "UTF-8"));

			String line;
			// 一行ずつ読み込み，ループをまわす
			while ((line = br.readLine()) != null) {
				// Token 型の配列 token を取得．各要素が形態素に対応する．
				Token[] token = tagger.analyze(line);
				if (token != null) {
					// 形態素ごとに処理を行う
					for (int i = 0; i < token.length; i++) {
						// 単語の基本形（BasicString）を取得
						String term = token[i].getBasicString();
						// 品詞（part of speech）を取得
						String pos = token[i].getPos();
						// 以下で term や pos に対する処理を行う
						bw.write("term=" + term + ", pos=" + pos + "\n");
					}
				}
			}
			br.close();
			bw.close();
		} catch (Exception e) {
			e.printStackTrace();
		}
	}


	public static void main(String args[]) {

		// 引数が無い場合はヘルプを表示
		if (args.length == 0) {
			System.out.println("使用方法: java SenSample テキストファイル名");
			return;
		}

		SenSample ss = new SenSample(args[0]);
		ss.morph(ss.docfile, ss.outfile);
	}
}
```

ここでは people_sample.txt というファイルを BufferedReader クラスを使って一行ずつ読み込み，StringTagger クラスで形態素解析を行っている．

Tagger クラスが形態素解析の機能を有しており，形態素解析したい文章の文字列を引数としてそのクラスオブジェクトの analyze()メソッドを呼び出すと，形態素解析が実行される．形態素解析の結果は Token 型配列 token に入り，各要素が形態素に対応する．単語の基本形（動詞や形容詞の活

用を基本形に直したもの）は getBasicString()，品詞は getPos()というメソッドでそれぞれ取得できる．Sen のクラスやメソッドに関する詳細（API）は以下のページを参照すること．

/kyozai/amaeda/ir/sen/docs/api/index.html

kyozai ディレクトリにインストールされている Sen において，getPos()が出力する品詞は IPADIC 品詞体系に従ったものである．この体系では品詞は上位階層から下位階層に，ハイフンで区切って指定される．たとえば一般の名詞であれば「名詞-一般」，人物の姓であれば「名詞-固有名詞-人名-姓」，人物の名前であれば「名詞-固有名詞-人名-名」，地名であれば「名詞-固有名詞-地域-一般」，サ変動詞として使われる名詞であれば「名詞-サ変接続」が getPos()の戻り値となる．

以下の文書の 16～38 ページに IPADIC 品詞体系に関する詳細な説明がある．

http://chasen.naist.jp/snapshot/ipadic/ipadic/doc/ipadic-ja.pdf

なお，Java では String クラスのインスタンス（文字列）の一致を調べるには「==」ではなく equals メソッドを利用することに注意すること．たとえば String 型変数 pos の値（品詞）が「名詞」かどうかを判定するのであれば，

```
if("名詞".equals(pos)){
    System.out.println("単語: " + pos + " は名詞です");
}
```

とすればよい（実際には，全品詞情報(pos)から，最上位の階層の品詞情報を切り出す必要がある．その際は split メソッドを使うと良い．）．

これらの if 文で単語がうまく取得できない場合（if 文が真にならない場合），people.txt の文字コードと自作プログラムのソースコードの文字コードが一致していない可能性がある．この場合は nkf（Network Kanji Filter）というプログラムを利用して文字コードを一致させるとよい．

```
cp Program.java orig1.Program.java
nkf -w orig1.Program.java > Program.java
```

nkf では，-e オプションによって EUC–JP，-s オプションによってシフト JIS，-w オプションによって UTF-8 への変換をそれぞれ行うことができる．この変換結果をリダイレクションにより Program.java に書き込んでいる．people_sample.txt は UTF-8 であるので，この場合は nkf -w を使う．

【必須課題 2-1】SenSample.java を改造し，テキストファイル people_sample.txt 中の文章を形態素解析し，その結果のうち名詞だけを 1 行に 1 単語ずつファイル（ファイル名は任意）に出力するプログラムを作成せよ．（課題 2-1 はチェックを受けず，課題 2-2 ができた時点でチェックを受けること）

実行結果の例：

香川
真司
わ
年
月
日
兵庫
県
神戸
市
出身
プロ
サッカー
選手
プレミア

（以下略）

【必須課題 2-2】 必須課題 2-1 で作成したプログラムを改造し，テキストファイル people_sample.txt 中の文章に出現する各名詞の出現頻度を集計し，ファイル（ファイル名は任意）に出力するプログラムを作成せよ．出現頻度の集計方法については必須課題 1 を参考にすること．

実行結果の例：

```
中村: 1
メディア: 2
カップ: 6
名門: 1
年生: 3
最終: 3
王手: 1
等: 1
前半: 3
評価: 2
ハト: 1
スーパー: 2
回戦: 1
桁: 1
後半: 2

（以下略）
```

8. 索引付け

ある用語についての記述が書籍中のどのページ記載されているか探す際には，索引を利用することが多い．索引には，用語とその記載ページの組が，用語の 50 音順やアルファベット順に列挙されており，これを利用することで容易に目的の用語の記載ページを特定することができる．

文書の検索を行う際，検索システムは入力された検索語が含まれた文書を探す．そのためには，検索語がどの文書に含まれるかという情報が必要であり，上記の書籍の索引と同様の機能を必要とする．

検索システムにおける索引は，「**転置索引**（転置インデックス，inverted index）」と呼ばれる．転置索引は単語が与えられた時，それを含む文書の ID を取得するために使用するデータ構造である．なぜ「転置」かといえば，文書を開いて単語が出てくるのが順方向（通常の方向）とした場合，単語から文書を求めるのはその逆方向であるためである．

8.1. 転置索引の構造

転置索引には，「どの単語がどの文書に含まれるか」という情報を格納する必要がある．例えば図 1 の 3 つの文書があったとする．「情報」という単語は文書 1，2 に出現するため，「情報 1,2」という情報を転置索引に格納する．こうして転置索引を作成した結果は，図 2 のようになる．このように転置索引を作成することにより，「情報」という単語を含んだ文書を検索する際には，「情報」という単語をキーとして転置索引にアクセスすることにより，文書「1,2」に含まれることがわかる．

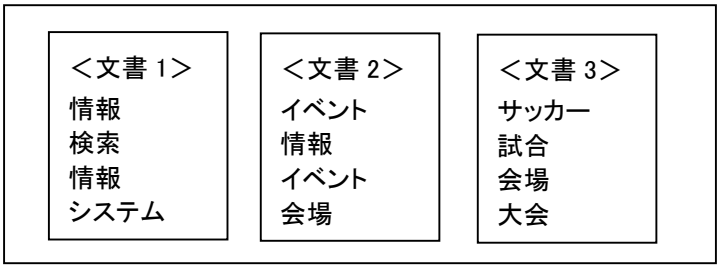

図 1. 索引付けの対象文書の例

図 2 では各単語が出現する文書の ID（番号）のみを格納しているが，それに加えて各文書でのその単語の出現頻度もあわせて格納することもある．一つの文書中によく出現する単語はその文書において重要な単語であると考えられるため，単語の出現頻度情報を利用することもあるからである．たとえば図 1 の文書 2 では，「イベント」という単語が 2 回出現しているが，このことにより文書 2 は「イベント」に関する内容が書かれていることが予想される．転置索引に単語の出現頻度をあわせて格納するには，たとえば「文書 ID:出現頻度」のような形式とすればよい．この形式で格納された索引を「**ポスティングリスト**」と呼ぶ．文書 ID と出現頻度の間をコロン（:）で区切っているのは，文書ごとの区切りをコンマ（,）としているため，別の区切り記号にする必要があるからである．出現頻度情報もあわせて格納した転置索引の例を図 3 に示す（本実験では出現頻度情報もあわせて格納した転置索引を作成する）．たとえば図 3 の 1 行目の「情報 1:2,2:1」は，「情報」という単語が，文書 1 に 2 回，文書 2 に 1 回出現していることを表している．

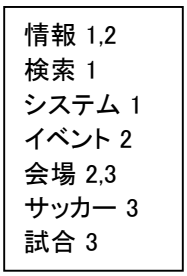

図 2. 転置索引の例
（文書 ID のみ格納）

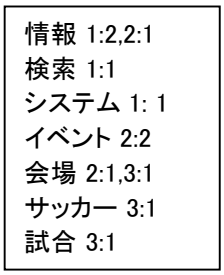

図 3. 転置索引の例
（出現頻度情報も格納）

8.2. 転置索引の作成プログラム

転置索引の作成は，以下の手順で行う．

① 文書ごとに単語の出現頻度の集計を行い，集計用 HashMap に格納

② 集計用 HashMap の内容を転置索引用 HashMap に格納

③ 文書の数だけ①，②を繰り返す

※集計用 HashMap は，同じインスタンスを次の文書で再利用可能

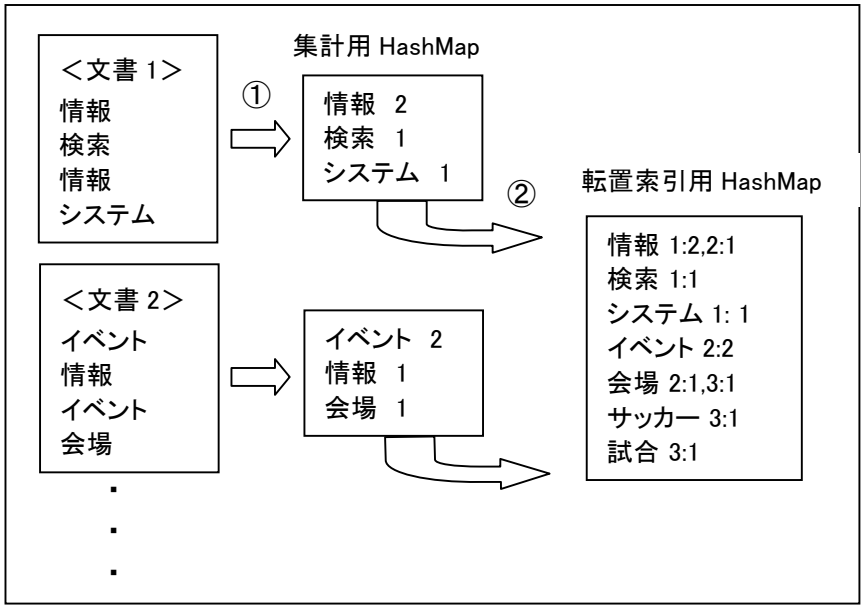

図 4. 転置索引の作成手順

転置索引用として，HashMap<String, String>を用意する．①の集計は，必須課題 2-2 で作成したプログラムで実行できる．ただし，集計結果をファイルに出力するのではなく，転置索引用 HashMap に put する（上記処理②）．このとき，すでにその単語が転置索引用 HashMap に格納されていた場合は，一旦その単語に対応する値を転置索引用 HashMap から取得し，その文字列に索引情報（「文書 ID:出現頻度」）を付け加えたうえで，再度，転置索引用 HashMap に格納する（たとえば，図 4 の例では，文書 1 の処理が完了した後に，転置索引用 HashMap のキー「情報」に値「1:2」が格納されるが，次に文書 2 の処理が完了した後に，同じキー「情報」に値「2:1」を追加する必要がある．このため，まず「情報」の値「1:2」を取り出し，その文字列の末尾に「,」で区切って「2:1」を追記し，再度，転置索引用 HashMap に格納することになる）．

【必須課題 3】テキストファイル people.txt（/kyozai/amaeda/ir/ の下にある）に保存されている文書群を読込み，転置索引を作成するプログラムを作成せよ．ただし，索引に用いる単語は名詞のみとする．なお，people.txt は複数の文書が 1 ファイルにまとめられており，

文書名＜tab＞本文文章・・・・

という形式で，1 文書が 1 行（改行コード「\n」まで）に保存されている（<tab>はタブ文字「\t」を表す）．テキストエディタ等で表示すると，本文が長いため複数行に渡っているように見えるが，実際には改行コード「\n」までが，ここで扱う「1 文書」であることに注意．また，転置索引はテキストファイル「inv_index.txt」に出力せよ．出力形式は以下の通りにせよ．

単語 1＜tab＞文書 ID:出現頻度,文書 ID:出現頻度, . . .

単語 2＜tab＞文書 ID:出現頻度,文書 ID:出現頻度, . . .

例:

```
情報    1:2, 2:1
検索    1:1
…
```

実行結果の例（inv_index.txt）：

```
要      10:1
小学校  1:1,3:2,4:1,8:1,9:3,10:2,12:1,14:1,15:1,16:1,17:3,18:1,19:1
和彦    6:1
最終    0:1,1:1,2:4,3:3,4:3,5:3,6:2,8:3,9:3,10:10,12:1,13:6,14:3,16:1,17:1,18:1
しのぶ  14:1
つる    10:1
神宮    8:1,9:1
梶山    1:2
視      1:1,5:1
代行    8:1
幕張    3:1
佐々木  6:1
公用    3:1
巧み    2:1,3:1
交通    10:1

（以下略）
```

9. 簡単な検索エンジンの作成

本章では，前章で作成した転置索引を利用して文書の検索を行うシステムを実装する．

9.1. 文書検索プログラム

サンプル Retrieval1.java は検索語を含む文書を取得するプログラムである（/kyozai/amaeda/ir/の下にも同じものがある）．

検索語を半角空白で区切って入力すれば，複数の単語で検索することもできる．ただし AND 条件（両方を含む文書を取得）ではなく OR 条件（どちらかを含む文書を取得）である．

人物のフルネームのように二つの単語（姓と名）から構成される検索語の場合，転置索引にはそれぞれの単語が別々に入っているために，検索してもヒットしないので注意すること．

このプログラムで得られるのは単に検索語を含む文書のリストであり，関連が深いものが上位に来るとは限らない．そのため次章で検索結果のランキングの機能を追加する．

```
//
// 文書検索
//
// args[0]: 転置索引ファイル

import java.io.*;
import java.util.*; // HashMap を使う際に import
import java.util.Scanner;

public class Retrieval1 {

    HashMap<String, String> inv_table; //転置索引を格納する HashMap
    String inv_file;
```

```
final int doc_max = 20; //最大文書数
Document[] doc; // 文書情報を格納するクラスの配列

// コンストラクタ
retrieval1(String inv) {
    inv_table = new HashMap<String, String>(1001); // dic_table のインスタンス化
    inv_file = inv;
    doc = new Document[this.doc_max];
}


void docIntialize(Document[] doc) {
    for(int i = 0; i < doc.length; i++) {
        //文書情報の設定 (本来は, 文書情報ファイルを読み込む)
        int docid = i;
        String docname = "doc_" + i; //仮の文書名
        double length = 0.0; //仮の文書ベクトルの大きさ

        doc[i] = new Document(docid, docname, length);
    }
}


void readFile(HashMap<String, String> hm, String infile) {
    try {
        BufferedReader br = new BufferedReader(
                                new InputStreamReader(
                                    new FileInputStream(infile), "UTF-8"));

        String line;
        while( (line = br.readLine() ) != null ){
            String[] term = line.split("\t"); //単語とポスティングリストに分割
            hm.put(term[0], term[1]);
        }
        br.close();
    }
    catch(Exception e) { System.out.println("I/O error at readFile \n" + e); }
}


void retrieve(String query, HashMap<String, String> invt, Document[] doc) {
    String[] qterm = query.split("\\s"); // 検索語の分割(半角スペースで)
    //検索語ごとに類似度を計算
    for(int i = 0; i < qterm.length; i++) {
        if ( invt.containsKey( qterm[i] ) ) {
            String postings = (String)invt.get(qterm[i]);
            String[] pos_doc = postings.split(","); //ポスティングリストの分割
            for(int j = 0; j < pos_doc.length; j++) {
                String[] appear = pos_doc[j].split(":"); //文書 id と出現頻度の分割
                int docid = Integer.parseInt(appear[0]); // 文書 id の取得
                //本来, スコア(適合度)の計算は必要ない (ベクトルモデルでは必要)
                double newscore = doc[docid].getScore() + 1.0 / qterm.length;
                doc[docid].setScore(newscore);
            }
        } // end if
    } //end outer for loop
} //end method retrieve


//検索結果の出力
void outResult(Document[] doc) {
    //Arrays.sort(doc); // 検索結果の並び替え

    for(int i = 0; i < doc.length; i++) {
        if ( doc[i].getScore() > 0.0 ) {
            System.out.println( doc[i].getDocname() + ": " + doc[i].getScore() );
        }
```

```
        }
    }

    public static void main(String[] args) {
        retrieval1 ret = new retrieval1(args[0]);
        ret.readFile(ret.inv_table, ret.inv_file);
        ret.docIntialize(ret.doc);

        String query = "";
       System.out.print("検索語を入力して下さい(単語ごとに「半角 space」で区切る): ");
        try{
           BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
           query = br.readLine();
           br.close();
        }
        catch(Exception e) { System.out.println("I/O error at query input \n" + e); }

        ret.retrieve(query, ret.inv_table, ret.doc);
        ret.outResult(ret.doc);
    }

} //class Retrieval1 end


// 文書を表すクラス
class Document implements Comparable<Document> {
    private int docid;       // 文書 ID
    private String docname;    // 文書名
    private double length;    // 文書ベクトルの大きさ
    private double score;       // 文書のスコア(検索時に決まる)

    // コンストラクタ
    public Document(int docid, String docname, double length) {
        this.docid = docid;
        this.docname = docname;
        this.length = length;
    }

    // 文書 ID を返すメソッド
    public int getDocid() {
        return this.docid;
    }

    // 文書名を返すメソッド
        public String getDocname() {
        return this.docname;
    }

    // 文書ベクトルの大きさを返すメソッド
    public double getLength() {
        return this.length;
    }

    // 文書のスコアを返すメソッド
    public double getScore() {
        return this.score;
    }

    // 文書のスコアを設定するメソッド
    public void setScore(double score) {
        this.score = score;
    }

    // ソート用メソッド(スコアの降順でソート)
    public int compareTo(Document d) {
        double s1 = this.score;
        double s2 = d.score;
```

```
        if (s1 == s2) {
            return 0;
        }
        else if (s1 < s2) {
            return 1;
        }
        else {
            return -1;
        }
    }
} //end class doc
```

9.2. 検索プログラムの説明

サンプル Retrieval1.java では，クラス変数として HashMap(inv_table)を持っており，ここにファイルから転置索引の情報を格納する．文書の検索を行う際は，この inv_table にアクセスすることで，検索語がどの文書に含まれているかを取得する．

どの文書に検索語が含まれていたかを記録するために，Document クラス型のオブジェクト配列を用意している．検索語が含まれていた場合，その文書 ID に対応する添字の Document オブジェクト中の変数 score に値（1/検索語数）が加算される（今回はスコアを計算する必要はないが，次章でランキングする際に使用する）．全ての検索語に対して調べ終わった段階で，Document オブジェクト中の変数 score が 0 を超えている文書は検索語を含んでいることを意味しているため，その文書 ID を出力する．

【必須課題 4】 Retrieval1.java では，検索結果として表示されるのは文書 ID（と検索語とのマッチ率）のみであるが，検索結果が文書 ID ではなく文書名（人物名）で表示されるように書き換えたい．そのために必要となる以下の 2 つのプログラムを作成せよ．

1. 文書情報を取得し，ファイル（docdata.txt)に出力するプログラム．必須課題 3 で作成した索引付けプログラムを変更して作成すること．

出力形式は，

文書 ID<tab>文書名：文書ベクトル長

とすること．

また，「文書ベクトル長」は，各単語の出現頻度の 2 乗の合計の平方根とせよ．

実行結果の例（docdata.txt）：

```
0       香川真司:62.313722405261586
1       本田圭佑:100.93562304756433
2       遠藤保仁:104.0
3       田中マルクス闘莉王:69.45502141674135
4       長友佑都:93.96807968666806
5       阿部慎之助:86.31338250816034
6       藤川球児:111.77656283854859
7       中村剛也:76.10519036176179
8       青木宣親:78.07688518377255
9       田中将大:101.72511980823616
10      石川遼:140.48843368761715
11      片山晋呉:39.67366884975475
12      丸山茂樹:96.16652224137046
13      宮里藍:71.95832127002409
14      横峯さくら:64.28063471995279
15      栗原恵:39.824615503479755
16      大山加奈:33.83784863137726
17      木村沙織:56.894639466297704
18      竹下佳江:32.357379374726875
19      荒木絵里香:27.477263328068172
```

2. 上記 1.のプログラムで出力した文書情報ファイルと転置索引ファイルを読み込み，検索結果が文書名で表示されるように Retrieval1.java を変更せよ.

実行例：

```
% java Kadai4_2 inv_index.txt docdata.txt
検索語を入力して下さい（単語ごとに「半角 space」で区切る）： 春
本田圭佑: 1.0
阿部慎之助: 1.0
青木宣親: 1.0
横峯さくら: 1.0
栗原恵: 1.0
大山加奈: 1.0
荒木絵里香: 1.0
```

※下線部は入力した文字を示す

10. 検索結果のランキング

10.1. 検索結果のランキングとは

前章で作成した検索エンジンでは，単に検索語が含まれる文書の集合を検索結果として返すだけであり，各文書がどれだけ問合せにふさわしいか（適合しているか）は考慮されていない．検索結果が数件であればこれでも問題ないが，検索結果の文書数が多い場合，その中からユーザが必要な文書を探すのに非常に手間がかかってしまう．そこで，検索結果を，問合せにふさわしい順（適合度順）に並べることを考える．これを検索結果のランキングと呼ぶ．

10.2. ベクトル空間モデル

ベクトル空間モデルとは，情報検索のモデルの一つであり，文書を索引語の重みベクトルで表現するモデルである．問合せも同様に索引語の重みベクトルで表現することで，各文書が問合せにどれだけ適合しているかをベクトルの類似度で測ることができる．検索結果の文書集合を類似度の降順でソートすることで，検索結果のランキングが可能となる．

検索対象となる文書を $D_1, D_2, \ldots, D_n$ とし，これらの文書の集合全体を通して全部で m 個の索引語 $w_1, w_2, \ldots, w_n$ があるとする．このとき，ある文書 D_j は，次のようなベクトルで表現できる．これを**文書ベクトル**と呼ぶ．

$$\mathbf{d}_j = \begin{bmatrix} d_{1j} \\ d_{2j} \\ \vdots \\ d_{mj} \end{bmatrix}$$

ここで，d_{ij} は索引語 w_i の文書 D_j における重み（たとえば出現頻度）である．すなわち文書ベクトルのひとつの次元は索引語に対応し，成分は文書におけるその索引語の重み（たとえば出現頻度）を表している．重みとして出現頻度を使用する場合，d_{ij} は索引語 w_i が文書 D_j に出現する回数を表す．

文書ベクトルの大きさ（長さ, length）は以下のように通常の 2-ノルムの計算によって求めることができる．（ピタゴラスの定理の多次元版）．

$$\left\|\mathbf{d}_j\right\| = \sqrt{\sum_{i=1}^{m} d_{ij}^2} = \sqrt{d_{1j}^2 + d_{2j}^2 + \cdots + d_{mj}^2}$$

文書集合全体は，次のような $m \times n$ 行列 D によって表現することができる．この行列 D を索引語・文書行列と呼ぶ．行列の各列は文書ベクトルであり，各行は索引語に関する情報を表しており，**索引語ベクトル**と呼ぶ．

$$D = [\mathbf{d}_1 \quad \mathbf{d}_2 \quad \cdots \quad \mathbf{d}_n] = \begin{bmatrix} d_{11} & d_{12} & \cdots & d_{1n} \\ d_{21} & d_{22} & \cdots & d_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ d_{m1} & d_{m2} & \cdots & d_{mn} \end{bmatrix} \qquad \mathbf{q} = \begin{bmatrix} q_1 \\ q_2 \\ \vdots \\ q_m \end{bmatrix}$$

問合せも文書と同様に索引語の重みベクトルで表現できる．問合せに含まれる索引語 w_i の重み（出現頻度を使う場合，通常は 0 か 1）を q_i とすると，問合せベクトル $\mathbf{q}$ は次のように表わされる．

たとえば問合せが「免許　二輪」であるならば，「免許」と「二輪」に対応する次元の成分のみが 1，それ以外の成分はすべて 0 というベクトルになる．

問合せベクトルの大きさ（長さ）も 2-ノルムであり，以下のように求められる．

$$\|\mathbf{q}\| = \sqrt{\sum_{i=1}^{m} q_i^2} = \sqrt{q_1^2 + \ldots + q_m^2}$$

ただし問合せベクトルの成分は 0 か 1 しかとらないため，実質的にこれは問合せに含まれる語（検索語）の数の平方根と等しい．

なお，文書ベクトルおよび問合せベクトルの次元数 m は，文書集合全体を通して互いに異なる索引語の数であり，索引語の選び方や文書集合の規模にもよるが，一般に数千～数万次元という高次元のベクトルになる．

10.3. コサイン類似度

前節で説明した通り，ベクトル空間モデルでは，問合せベクトルと各文書ベクトルとの類似度を計算することで，問合せに適合する文書を検索する．ベクトル間の類似度の計算は様々な尺度が考えられるが，情報検索で良く用いられるのがコサイン類似度（コサイン尺度，コサイン距離などとも呼ばれる）である．

問合せベクトル $\mathbf{q}$ と各文書ベクトル $\mathbf{d}_j$ とのコサイン類似度は以下の式で計算できる．

$$\cos(\mathbf{d}_j, \mathbf{q}) = \frac{\mathbf{d_j} \cdot \mathbf{q}}{\|\mathbf{d_j}\|\|\mathbf{q}\|} = \frac{\sum_{i=1}^{m} d_{ij} q_i}{\sqrt{\sum_{i=1}^{m} d_{ij}^2} \sqrt{\sum_{i=1}^{m} q_i^2}}$$

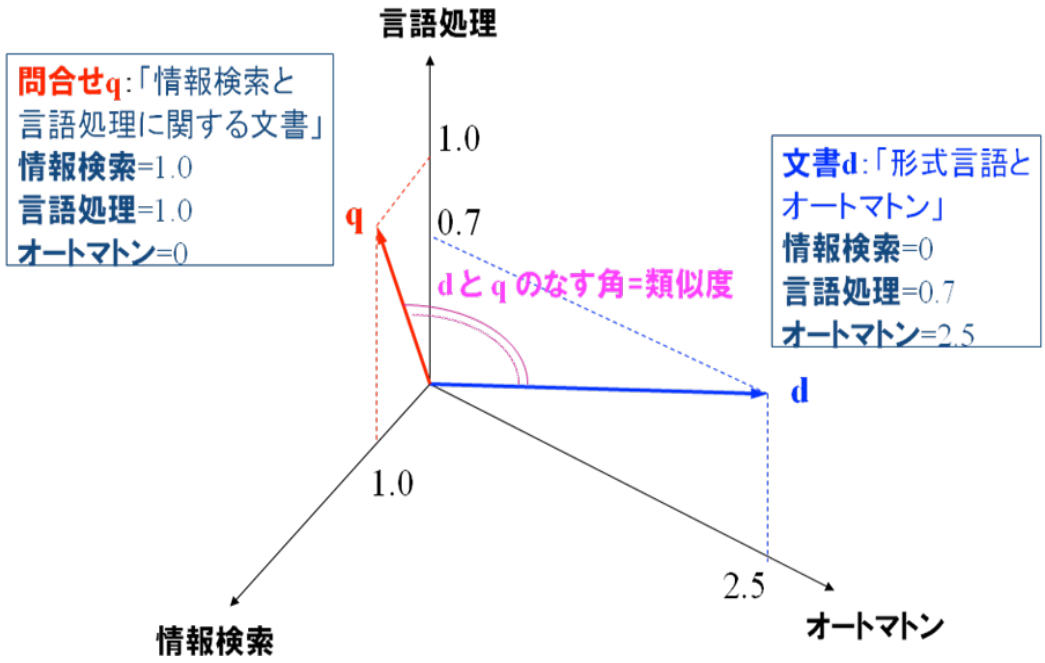

図 5. コサイン類似度による文書検索の例

コサイン類似度が表すのは問合せベクトルと文書ベクトルの間の角度 θ の余弦(cos)である. cos は 0 から $\pi/2$ の範囲では単調減少であるため, θ が小さいほど $\cos\theta$ は大きい. ゆえにコサイン類似度は問合せと文書の 2 つのベクトルのなす角が小さいほど大きな値を取り, 類似性の指標として使用できる. 類似度を求めるのに使われるのは問合せベクトルと文書ベクトルの間の角度だけであり, 問合せの長さ(検索語の数)や文書の長さは影響を与えない.

図 5 に, コサイン類似度を用いた類似度計算の例を示す. この例では, 単純のために 3 次元(索引語が 3 種類しかない)としている. この例では, 問合せと文書のコサイン類似度は以下のように計算できる.

$$\cos(\mathbf{d},\mathbf{q}) = \frac{1.0\times 0 + 1.0\times 0.7 + 0\times 2.5}{\sqrt{0^2+0.7^2+2.5^2}\sqrt{1.0^2+1.0^2+0^2}} = \frac{0.7}{\sqrt{6.74}\sqrt{2}} = 0.1907$$

問合せに「情報検索」と「言語処理」が含まれているため, 問合せベクトル **q** は第 1 成分と第 2 成分が 1, 第 3 成分が 0 のベクトルである. ゆえに内積を求める計算では文書ベクトル **d** の第 1 成分(0.7)と第 2 成分(2.5)だけが足されることになる.

また, 分母は文書ベクトル **d** の大きさ(ノルム)と問合せベクトル **q** の大きさ(ノルム)が掛け合わされた値である. 文書集合中のすべての文書に対してコサイン類似度を計算し, 降順にソートすることで, 検索結果の適合度順によるランキングが可能になる.

10.4. コサイン類似度によるランキングの実装

9 章で作成した検索エンジンに, コサイン類似度によって検索結果をランキングして表示する機能を実装する. 10.2～10.3 節で説明した通り, 本来はコサイン類似度の計算には問合せと各文書をベクトルで表す必要があるが, 実際には各ベクトルのほとんどの要素は 0 となるため, そのまま計算するのは効率的ではない. そこで, 以下のようなアルゴリズムで計算を行うと良い.

```
すべての文書 d について以下の処理(2～11)を繰り返す {
    変数 dotProduct を 0 に初期化
    すべての検索語 t について以下の処理(4～8)を繰り返す {
        検索語 t が索引にある場合, それに対応するポスティングリストを取り出す
        変数 tf を 0 に初期化
        ポスティングリストの先頭から順番に, 文書 IDd'と頻度のペアを取り出す {
            文書 IDd'が d と等しければ, tf に頻度を格納
        }
        dotProduct に tf を加算
    }
    問合せ q と文書 d のコサイン類似度を以下の式で求める(Math.sqrt()は平方根を求めるメソッド)
        cos(d, q) = dotProduct / (文書 d のベクトルの大きさ * Math.sqrt(検索語数))
}
```

上のアルゴリズムでは, コサイン類似度の式の分子の部分は, 問合せに含まれない索引語の次元はすべて 0 になることを利用し, 問合せに含まれる索引語の次元についてのみ計算を行っている.

また，上のアルゴリズムの Step 9 はコサイン類似度の式の分子の部分の計算であり，問合せと各文書における索引語 *t* の重みを掛け合わせる．本実験では，検索語の重みはすべて 1 とするので，単に頻度の値を加算すれば良いことになる．

また，Step 11 で「文書 *d* のベクトルの大きさ」を用いるが，これは必須課題 4 で作成した文書ベクトル長の計算結果を用いれば良い．

【必須課題　5】 ベクトル検索モデルによる検索システムのプログラムを作成せよ．なお，検索結果は検索語との類似度順にソートして出力すること．

実行例：

```
% java Kadai5 inv_index.txt docdata.txt
検索語を入力して下さい（単語ごとに「半角 space」で区切る）： 春
荒木絵里香: 0.03639372626234195
大山加奈: 0.029552706228277083
栗原恵: 0.02511009804759127
横峯さくら: 0.015556784782176378
青木宣親: 0.012807887989463999
阿部慎之助: 0.011585688927269844
本田圭佑: 0.009907304971296067
```

※下線部は入力した文字を示す

11. 人物相関の視覚化

11.1. グラフ描画ソフトウェア Jung

Wikipedia に登録されている人物の相関関係を視覚化するために，グラフ描画ソフトウェア Jung を用いる．ここでいう「グラフ」とは，棒グラフや円グラフのことではなく，グラフ理論に基づくノード（頂点）とエッジ（辺）で構成されるグラフのことである．Jung は，Java でグラフを表示するためのオープンソースの API ライブラリである．参考文献[16]に詳しい説明があるので，適宜参照すること．

11.2. 人物間のコサイン類似度の計算

10.3 節で説明したコサイン類似度は，検索結果のランキングのため，問合せベクトルと各文書ベクトルの類似度の計算に用いたが，問合せベクトルの代わりに文書ベクトルを用いれば，2 つの文書ベクトル間の類似度を計算することができる．前章で用いたコサイン類似度計算のプログラムを，すべての文書（人物）の組み合わせについて計算することで，人物間の類似度の行列を作ることができる．

なお，10.4 節のコサイン類似度の計算では，問合せの単語があらかじめ分かっているため，それに対応する索引語の次元についてのみ計算することができたが，文書間のコサイン類似度の計算では，ある文書に含まれる索引語は転置索引をすべて調べなければ分からない．そこで，以下のアルゴリズムで計算を行うと良い．

```
1.  すべての文書の組み合わせ a,b について以下の処理(2～11)を繰り返す {
2.      変数 dotProduct を 0 に初期化
3.      すべての索引語 t について以下の処理(4～10)を繰り返す(転置索引 HashMap の内容すべて) {
4.          変数 tf_a, tf_b を 0 に初期化
5.          索引語 t のポスティングリストを取り出す
6.          ポスティングリストの先頭から順番に，文書 IDd'と頻度(TF)のペアを取り出す {
7.              文書 IDd'が a であれば，tf_a に頻度を格納
8.              文書 IDd'が b であれば，tf_b に頻度を格納
9.          }
10.         索引語 t のポスティングリストに文書 a,b 両方が含まれていれば，dotProduct に，
            tf_a * tf_b を加算
11.     文書 a と b のコサイン類似度は，dotProduct / (length_a * length_b)で求められる
        (ただし，length_a , length_b は必須課題 4 で取得した各文書の文書ベクトル長)
    }
```

【必須課題 6】 上で説明した計算手順を参考に，人物間のコサイン類似度の行列を求めて表示するプログラムを作成せよ． なお，出力結果はファイル(personal_relation.txt)に保存せよ．

実行結果の例（personal_relation.txt）：

```
香川真司,本田圭佑,遠藤保仁,田中マルクス闘莉王,長友佑都,阿部慎之助,藤川球児,中村剛也,青木宣
親,田中将大,石川遼,片山晋呉,丸山茂樹,宮里藍,横峯さくら,栗原恵,大山加奈,木村沙織,竹下佳江,荒
木絵里香 ⏎
1.0,0.6962204598716486,0.763043437107556,0.6240756523152371,0.6884125056548204
,0.6120656330001543,0.5457118990563504,0.5330636856767503,0.5265904164029858,0
.6344978589731431,0.4982661629480239,0.4170350969665854,0.5088031818531109,0.4
890732542161935,0.38696157895291056,0.5951757234961899,0.5287963256347953,0.56
04576766695519,0.5827480520701381,0.5577585038893755 ⏎
0.6962204598716486,1.0,0.7529551778185011,0.663434759382152,0.6367078578295722
,0.507225871538729,0.4776535002908645,0.4476859151699315,0.4398064670444757,0.
5521203807688222,0.4466058929999178,0.399052414438575,0.3986966497829898,0.416
4855017352265,0.36943956946998413,0.5380968625055764,0.49188329119551183,0.566
4586923117259,0.5394958336168106,0.5127216455647476 ⏎
0.763043437107556,0.7529551778185011,1.0,0.693171347257351,0.6769893013404868,
0.5958831737689076,0.5441831590830201,0.5293786316932206,0.492857382056105,0.6
185598703810262,0.5071591883387911,0.48060359036912115,0.5365292686435258,0.52
58118559986494,0.3996897013266854,0.6311326566961882,0.5842342692820931,0.5771
4643716109,0.629686346475702,0.6295414764033957 ⏎

（以下略）
```

※上の例は実行結果の最初の 4 行を表示している(実際の 1 行が表示上では複数行にまたがっていることに注意．「⏎」が実際の改行を表している)

11.3. Jung による人物相関のグラフ作成

前節で作成した人物間のコサイン類似度の行列を基に, Jung を用いて人物相関のグラフを作成する. ノードは人物とし, 2 人の人物間のコサイン類似度がある閾値を超えた場合にそれらの人物間にエッジを作成するものとする. 入力は, 前節で作成した人物間のコサイン類似度の行列を CSV 形式(Comma Separated Values, コンマで値を区切った形式)でファイルに保存したものとする. 入力ファイルの形式は以下の通りである.

人名 1,人名 2,…,人名 n
cos(p_1,p_1),cos(p_1,p_2),...,cos(p_1,p_n)
cos(p_2,p_1),cos(p_2,p_2),...,cos(p_2,p_n)
...
cos(p_n,p_1),cos(p_n,p_2), ..,cos(p_n,p_n)

1 行目は人名のコンマ区切りのリストであり, 2 行目以降は, 人物間のコサイン類似度である. cos(p_i,p_j)は人物 p_i と p_j のコサイン類似度を表す.

上の形式のファイル(ファイル名 personal_relation.txt)を読み込んで人物相関のグラフを作成するサンプルプログラム PersonalRelationGraph.java を以下に示す. (/kyozai/amaeda/ir/の下にも同じものがある).

```
/* PersonalRelationGraph.java */

import java.io.*;
import javax.swing.JFrame;
import edu.uci.ics.jung.graph.*;
import edu.uci.ics.jung.graph.decorators.*;
import edu.uci.ics.jung.graph.impl.*;
import edu.uci.ics.jung.visualization.*;
import edu.uci.ics.jung.utils.*;

public class PersonalRelationGraph {
  public static void main(String[] args) {
    try {

    // フレーム(ウィンドウの外枠)を作成
    JFrame window = new JFrame("PersonalRelationGraph");

    // (疎な)無向グラフの作成
    final Graph graph = new UndirectedSparseGraph();

    // エッジを作成するコサイン類似度の閾値
    double threshold = 0.3;     ← 閾値(0~1 の間)で調節すると表示結果も変わる

    // 人名の配列
    String[] names = null;

    // 相関行列(コンマ区切り)
    // ファイルの 1 行目は人名リスト, 2 行目以降は人物間のコサイン類似度の行列
    double[][] correlation = null;
    LineNumberReader in = new LineNumberReader(new InputStreamReader(
     new FileInputStream("personal_relation.txt")));

    // 1 行目は人名のリスト
    String line = in.readLine();
```

```
    // 人数分の行列の 2 次元配列を作成
    names = line.split(",");
    correlation = new double[names.length][names.length];

    // ファイルから 2 行目以降を 1 行ずつ読み込み行列に格納
    while ((line = in.readLine()) != null) {
      String[] t = line.split(",");
      double[] d = new double[names.length];
      for(int i = in.getLineNumber() - 1; i < t.length; i++) {
        d[i] = Double.parseDouble(t[i]);
      }
      correlation[in.getLineNumber() - 2] = d;
    }

    // Vertex(頂点)の配列に, 作成した Vertex オブジェクトを代入
    Vertex[] vertices = new Vertex[names.length];
    for (int i = 0; i < vertices.length; i++) {
      vertices[i] = graph.addVertex(new UndirectedSparseVertex());
      // 頂点のラベルの設定
      vertices[i].addUserDatum(NameVertexStringer.VERTEX_NAME_KEY,
     names[i], UserData.SHARED);
    }

    // 閾値以上の相関係数を持っている Vertex(頂点)同士を Edge(辺)で結ぶ
    for (int i = 0; i < correlation.length - 1; i++) {
      for (int j = i + 1; j < correlation[i].length; j++) {
        double current = correlation[i][j];
        if (Math.abs(current) >= threshold) {
          graph.addEdge(
            new UndirectedSparseEdge(vertices[i], vertices[j]));
        }
      }
    }

    // FR (Fruchterman-Reingold) アルゴリズムによるレイアウトを作成
    Layout layout = new FRLayout(graph);

    // グラフの描画を行うレンダラを作成
    PluggableRenderer renderer = new PluggableRenderer();

    // レンダラに頂点ラベルを設定
    renderer.setVertexStringer(new NameVertexStringer());

    // グラフを表示するビューア(パネル)を作成
    VisualizationViewer viewer =
            new VisualizationViewer(layout, renderer);

    window.add(viewer);                 // フレームにビューアのパネルを追加
    window.setSize(900, 900);           // フレームのサイズを 900x900 ドットに
    window.setLocationRelativeTo(null);
    window.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    window.setVisible(true);

    } catch (Exception e) {
      e.printStackTrace();
    }
  }
}
```

```
// 関連付けられた名前を返す VertexStringer
class NameVertexStringer implements VertexStringer {
    // 頂点に名前を関連付けるときのキー
    public static final Object VERTEX_NAME_KEY = new Object();

    public String getLabel(ArchetypeVertex vertex) {
        return (String) vertex.getUserDatum(VERTEX_NAME_KEY);
    }
}
```

上のプログラムを用いて，あるサンプルデータを表示した例を図 6 に示す．プログラムの詳細の説明は省くが，詳しくは参考文献[16][17]を参照すること．

【必須課題 7】 必須課題 6 で作成した人物間のコサイン類似度の行列を CSV 形式で出力し，上のプログラム `PersonalRelationGraph.java` で表示させてみよ．

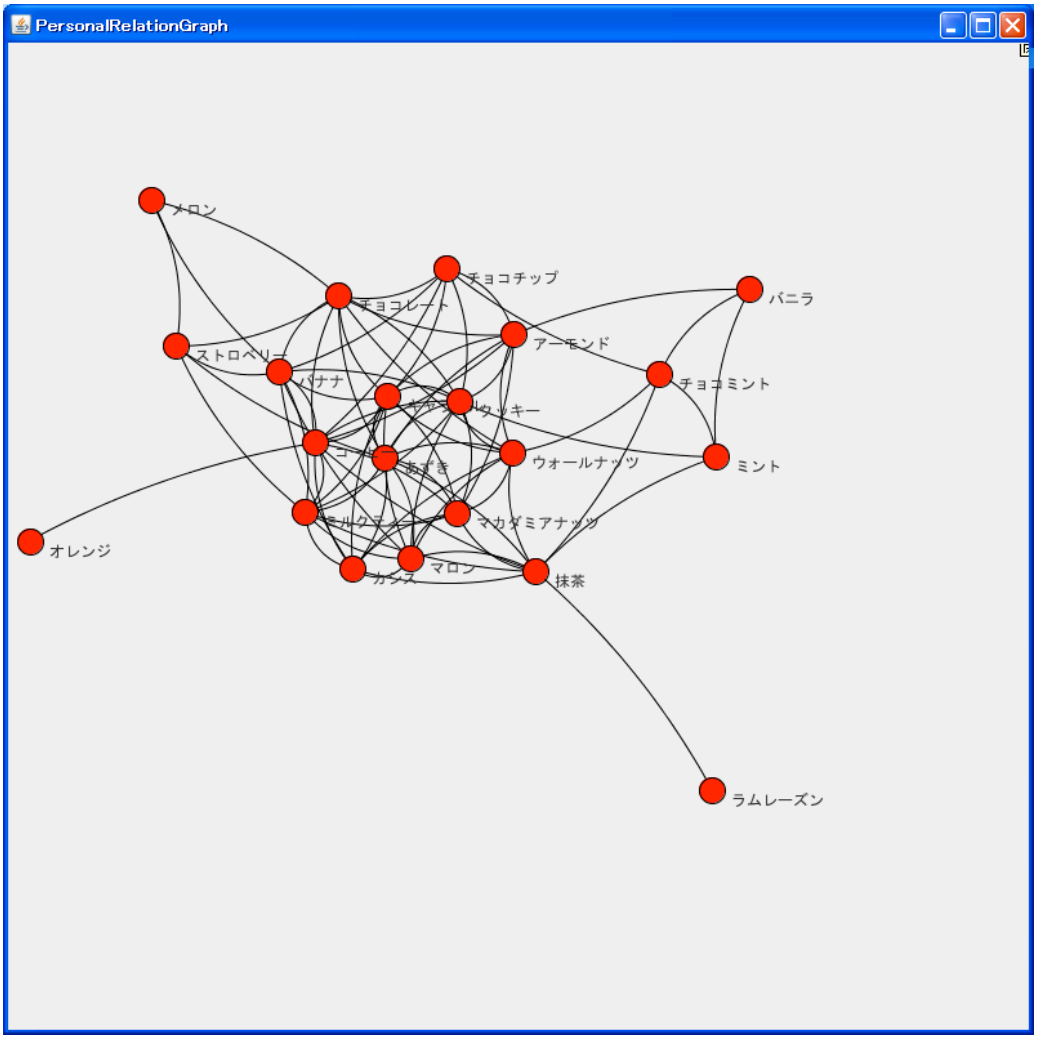

図 6. Jung によるグラフ表示の例

12. 索引のデータベースへの格納 ＜発展＞

8 章で作成した転置索引は，Java のデータ構造としてメモリ上に格納されているだけであり，プログラムを終了すると消えてしまう．実際の検索エンジンで用いるためには，索引をディスク上に保存しておく必要がある．これには様々な方法が考えられるが，本実験では，オープンソースのリレーショナルデータベース管理システム（RDBMS: Relational Database Management System）である MySQL（http://www-jp.mysql.com/）を用いる．**なお，以前の節で作成した `PostingList` はデータベースを使用することで不要になる．**

12.1. スキーマ設計

リレーショナルデータベースでは，すべてのデータはテーブル（表）に格納される．テーブルに含まれる個々の属性をフィールドまたはカラム（列），1 件分の情報をレコード（行）と呼ぶ．カラムにどのような型のデータを格納できるかは，テーブルを作成する際にあらかじめ決めておく必要がある．これをスキーマ設計と呼ぶ．

データベースで扱えるデータ型は，プログラミング言語と同様に様々なものが用意されている．MySQL で使用できる主なデータ型を表 1 に示す．

表 1． MySQL の主なデータ型

分類	データ型	バイト数	概要
数値	TINYINT	1	-128～127
	SMALLINT	2	-32,768～32,767
	MEDIUMINT	3	-8,388,608～8,388,607
	INT	4	-2,147,483,648～2,147,483,647
	BIGINT	8	-9,223,372,036,854,775,808～9,223,372,036,854,775,807
	FLOAT	4	単精度浮動小数点数
	DOUBLE	8	倍精度浮動小数点数
	DECIMAL		精密計算に使用する数値型
	BIT | BOOL | BOOLEAN	1	TINYINT の別名
文字列	CHAR(*文字数*)		固定長文字列
	VARCHAR(*文字数*)		可変長文字列
	TINYTEXT	可変	2^8-1 バイトの可変長文字列
	TEXT	可変	2^{16}-1 バイトの可変長文字列
	MEDIUMTEXT	可変	2^{24}-1 バイトの可変長文字列
	LONGTEXT	可変	2^{32}-1 バイトの可変長文字列
日付	DATETIME	8	日付時刻型．1000-01-01 00:00:00～9999-12-31 23:59:59
	DATE	3	日付型．1000-01-01～9999-12-31
	TIME	8	時刻型．-838:59:59～838:59:59 00:00:00
	TIMESTAMP	4	タイムスタンプ型．1970-01-01 00:00:00～2037-12-31 00:00:00
	YEAR	1	年．1901～2155
バイナリ	TINYBLOB	可変	2^8-1 バイト
	BLOB	可変	2^{16}-1 バイト
	MEDIUMBLOB	可変	2^{24}-1 バイト
	LONGBLOG	可変	2^{32}-1 バイト
列挙	ENUM(*ver1, ver2, ...*)		単一選択が可能な列挙型
	SET(*ver1, ver2, ...*)		複数選択が可能な列挙型

12.2. MySQL の使用準備

MySQL を使用する前に，まず設定ファイルを各自のホームディレクトリにコピーする必要がある．以下のコマンドを実行する．

```
% cp /kyozai/amaeda/ir/.my.cnf ~
```

次に，データベースを保管するディレクトリ（/tmp/jikken1_data/）および権限テーブル（MySQL が内部的に使用するテーブル）を作成するために，以下のコマンドを実行する．

```
% mysql_install_db
Installing MySQL system tables...
090911  2:24:38 [Warning] Forcing shutdown of 2 plugins
OK
Filling help tables...
090911  2:24:39 [Warning] Forcing shutdown of 2 plugins
OK

（中略）

Please report any problems with the /kyozai/amaeda/mysql/scripts/mysqlbug
script!

The latest information about MySQL is available at http://www.mysql.com/
Support MySQL by buying support/licenses from http://shop.mysql.com/
```

12.3. USB メモリの使用方法

本実験では，作成されるデータベースのサイズが大きくなるため，実験中は/tmp/jikken1_data/ディレクトリ以下にデータベースを作成し，毎週の実験終了後に各自の USB メモリ等にコピーすることとする．今回以降の実験終了後に 12.3.1 節の内容を，次回以降の実験開始前に 12.3.2 節の内容を実行すること．

12.3.1. USB メモリのマウントとデータのコピー（実験終了後）

メディア情報学科実験室 1,2 の Linux では，USB メモリを PC 本体の USB ポートに差し込むと，自動的にマウントされる．マウント先のディレクトリは，通常/media/disk-1 となる（自動的にファイルブラウザが立ち上がるので，マウント先を確認すること）．なお，RAINBOW の Linux 環境では，マウント・アンマウントの方法が異なる場合があるため，RAINBOW GUIDE などで確認すること．

正常にマウントされたら，以下のコマンドを実行することでデータベースをコピーすることができる．

```
% cp -rf /tmp/jikken1_data/ /media/disk-1
```
（disk-1 は，マウント時にファイルブラウザに表示されたディレクトリ名）

コピーが完了したら，必ずアンマウントを行う．これは，デスクトップ上にあるUSBメモリのアイコンを右クリックし，「アンマウント(U)」をクリックすることにより行う．アンマウントが正常に終了すると，アイコンがデスクトップから消える．この状態で，USBメモリを抜いて良い．

12.3.2. USBメモリ上のデータの復元（実験開始前）

前節でUSBメモリにコピーしたデータを/tmp/jikken1_data/ディレクトリに復元するには，前節と同様にマウントした後に以下のコマンドを用いる．

```
% cp -rf /media/disk-1/jikken1_data/ /tmp
(disk-1はマウント時に表示されたディレクトリ名)
```

12.4. MySQLの起動と終了

MySQLを使用するには，まずサーバ(mysqld)を起動させる必要がある．サーバの起動は以下のコマンドで行う．Linuxをシャットダウンするとサーバも終了するため，実験開始前に毎回起動する必要があることに注意．**また，mysqldを複数起動するとエラーが生じるので注意すること．** psコマンドを使用してすでに起動しているかどうかを確かめても良い．

```
% mysqld
```

mysqlサーバ(mysqld)がすでに起動している場合，このコマンドはエラーを生じさせるが，以下の演習を進める上では問題は生じない．強制終了させる場合は，psコマンドとgrepコマンドを使ってプロセス番号を調べ，killを行えばよい．

サーバの終了は以下のコマンド(シェルスクリプト)で行うこともできる．

```
% kill_mysqld.sh
```

mysqldコマンドによって起動されるのはサーバ(デーモン)であり，このサーバにmysqlコマンド(クライアントプログラム)を使って接続することでデータの入出力を行う．これについては次節で述べる．

なお，mysqldの挙動がおかしくなった場合，mysqlのデータが置かれたディレクトリを削除した上で再設定を行うと解決することがある．そのためのコマンドは以下である．

```
% rm -rf /tmp/jikken1_data
% mysql_install_db
```

/tmp/jikken1_dataを削除すると、それまでに作成したデータベースやテーブルは消えてしまう．しかしUSBメモリに保存したものをコピーしてくれば復元は可能である．

また，mysqldの状態に問題が生じている場合，psコマンドでmysqldのプロセス番号を調べ，killコマンドを使って停止させることで解決することもある．その場合に使用するコマンドは以下である．

```
% ps aux | grep mysqld
(↑これを使ってmysqldのプロセス番号を調べる．左から二列目がプロセス番号である)

% kill -KILL《プロセス番号をここに入れる》
```

12.5. MySQL のコマンドラインからの使用

MySQL は，コマンドラインから対話的に操作することも，後述のようにプログラムから操作することもできる．本節では，コマンドラインから使用するための mysql クライアントの使用方法について説明する．mysql クライアントは以下のコマンドで起動する．

```
% mysql -uroot

Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 17
Server version: 5.1.36 MySQL Community Server (GPL)

Type 'help;' or '\h' for help. Type '\c' to clear the current input
statement.

mysql>
```

「mysql>」は mysql クライアントのプロンプトであり，ここから MySQL に対して様々なコマンドを実行することができる．mysql クライアントを終了する場合は，プロンプト上で「exit」と入力する．

なお，mysql クライアント上で日本語を正しく扱うためには，文字コードの設定を行う必要がある．GNOME 端末の文字コードである UNICODE（UTF-8）に設定するには，以下のコマンドを用いる．この設定は保存されないので，mysql クライアントを起動するたびに毎回このコマンドを実行する必要がある．

```
mysql> SET NAMES utf8;           （ターミナルの文字コードを UTF-8（UNICODE）に設定）
Query OK, 0 rows affected (0.00 sec)
```

12.5.1. データベースの作成

テーブルを作成する前には「データベース」を作成する必要がある．ここでいうデータベースとは，ある目的（プログラム）のために使用するテーブル群をまとめて格納するフォルダのようなものである．ここでは，本実験用のデータベースとして「exp」という名前のデータベースを作成するために，以下のコマンドを実行する．

```
mysql> CREATE DATABASE exp;
Query OK, 1 row affected (0.00 sec)
```

CREATE DATABASE は SQL のコマンドの一つであり，データベースを作成するためのものである．なお，SQL では，コマンドの大文字・小文字は区別されないが，本テキストでは慣習に従って大文字で記述する．また，SQL ではコマンドの最後に必ずセミコロン（「;」）を付加することに注意．

データベースに対して操作するには，まず操作する対象のデータベースを選択する必要がある．このために次のように USE コマンドを用いる．

```
mysql> USE exp;
Database changed
```

あるいは，以下のように mysql クライアントの実行時の引数で指定することもできる．

```
% mysql -uroot exp
```

12.5.2. テーブルの作成

テーブルを作成するコマンドは CREATE TABLE である. 例えば, 表 2 のようなテーブルは, 以下のコマンドで作成することができる.

表 2. テーブルの例（テーブル名 student）

カラム	学生証番号※	氏名	入学年度
データ型	固定長文字列(12 文字)	可変長文字列(最大 20 字)	年

```
mysql> CREATE TABLE student(id CHAR(12) PRIMARY KEY, name VARCHAR(20),
year YEAR);
Query OK, 0 rows affected (0.06 sec)
```

表 2 の「※」が付いているカラム「学生証番号」は「主キー(primary key)」と呼ばれ, すべてのテーブルに必ず 1 つ必要である. 主キーとは, テーブル内のレコードを一意に特定するためのカラムであり, 主キーのカラムが同じ値のレコードを重複して格納することはできない.

構文の詳細については, 参考文献・URL の[14]などを参照のこと.

データベース内にあるテーブルを確認するには, 以下のコマンドを用いる.

```
mysql> SHOW TABLES;
+---------------+
| Tables_in_exp |
+---------------+
| student       |
+---------------+
1 row in set (0.00 sec)
```

テーブル内のフィールド(カラム)構成を確認するには, 以下のコマンドを用いる.

```
mysql> SHOW FIELDS FROM student;
+-------+-------------+------+-----+---------+-------+
| Field | Type        | Null | Key | Default | Extra |
+-------+-------------+------+-----+---------+-------+
| id    | char(12)    | NO   | PRI | NULL    |       |
| name  | varchar(20) | YES  |     | NULL    |       |
| year  | year(4)     | YES  |     | NULL    |       |
+-------+-------------+------+-----+---------+-------+
3 rows in set (0.00 sec)
```

テーブルを削除するには, 以下のコマンドを用いる.

```
mysql> DROP TABLE student;
```

12.5.3. データの挿入

前節で作成したテーブル student にデータを挿入する. データの挿入には以下のように INSERT コマンドを用いる. 構文の詳細については, 参考文献・URL の[14]などを参照のこと.

```
mysql> INSERT INTO student(id, name, year) VALUES('2600080000-0', '立命太
郎', '2008');
Query OK, 1 row affected (0.01 sec)
```

12.5.4. テーブルの表示

テーブルに格納されている内容を表示するには, SELECT コマンドを用いて以下のように行う.

```
mysql> SELECT * FROM student;
+--------------+----------+------+
| id           | name     | year |
+--------------+----------+------+
| 2600080000-0 | 立命太郎      | 2008 |
+--------------+----------+------+
1 row in set (0.01 sec)
```

12.6. Java プログラムからのデータベース操作

Java プログラムからデータベースに対して操作を行うために, JDBC と呼ばれる API が用意されている. JDBC 自体は Java 開発環境である JDK に含まれているが, 各種 DBMS と接続するための JDBC ドライバを別途用意する必要がある. MySQL 用の JDBC ドライバとして Connector/J を利用する.

以下に, 上で作成した student テーブルにデータを挿入するサンプルプログラム JDBCSample.java を示す. (/kyozai/amaeda/ir/の下にも同じものがある).

```
/* JDBCSample.java */
import java.sql.*;

public class JDBCSample {
  public static void main(String[] args) {
    try {
      // JDBC ドライバをロード
      Class.forName("org.gjt.mm.mysql.Driver");

      // データベースへ接続
      String url =
        "jdbc:mysql:///exp?useUnicode=true&characterEncoding=utf-8";
      String user = "root";
      String password = "";
      Connection db = DriverManager.getConnection(url, user, password);

      // ステートメントオブジェクトを生成
      Statement stmt = db.createStatement();

      // テーブル student へのデータの挿入
      String sql = "INSERT INTO student(id, name, year) " +
                   "VALUES('2600079999-9', '立命花子', '2007');";
      int num = stmt.executeUpdate(sql);     // INSERT 文の実行
      System.out.println(num + "件が挿入されました. \n");

      // テーブル student の表示
      sql = "SELECT * FROM student;";
      ResultSet rs = stmt.executeQuery(sql); // SELECT 文の実行
```

```
        // 検索された行数分ループ
        while (rs.next()) {
          String id = rs.getString("id");       // id カラムの内容を取得
          String name = rs.getString("name");   // name カラムの内容を取得
          String year = rs.getString("year");   // year カラムの内容を取得
          // 表示
          System.out.println(id + " " + name + " " + year);
        }

        // データベースから切断
        stmt.close();
        db.close();
      } catch (Exception e) {
        e.printStackTrace();
      }
    }
}
```

データベースアクセスに関わる主なクラスやメソッドは以下の通りである.

`Connection` クラス: データベースとの接続を表す.

`createStatement` メソッド: データベース操作に使用する `Statement` クラスのインスタンスを得る.

`Statement` クラス: データベースにおいて SQL 文を実行する際に使われるクラス.

`executeQuery` メソッド: 引数で指定された SQL 文を実行する.

`ResultSet` クラス: SQL 文の実行結果を保持しておくためのクラス.

`next` メソッド: SQL 文の結果として得られたタプル集合において, 次のタプルを参照するためのメソッド. もし次のタプルが存在しなければ `false` を返す.

`getString` メソッド: 現在参照しているタプルに対し, 引数で指定された属性の値を `String` 型で得る.

`getInt` メソッド: 現在参照しているタプルに対し, 引数で指定された属性の値を `int` 型で得る.

上のプログラムを実行すると, 以下の実行結果が得られる(12.5.3 節で挿入した 1 レコードのみが登録された状態で実行した場合).

```
% java JDBCSample
1 件が挿入されました.

2600080000-0 立命太郎 2008-01-01
2600079999-9 立命花子 2007-01-01
```

12.7. 転置索引テーブルの作成

8 章で作成した転置索引は, **表 3**, **表 4** の 2 つのテーブルで構成することができる.

表 3. 転置索引テーブル(テーブル名 `inv`)

カラム	索引語※ term	ポスティングリスト(文書 ID のリスト) postings	総出現頻度 totalfreq
データ型	可変長文字列(最大 32 文字)	可変長文字列(最大 65536 文字)	非負の整数

表 4. 文書テーブル(テーブル名 doc)

カラム	文書 ID※ docid	文書名 docname	文書ベクトルの大きさ(ノルム) length
データ型	非負の整数	可変長文字列(最大 256 文字)	浮動小数点数

12.5.1 節で作成したデータベース exp に, 上の 2 つのテーブルを作成する. これらは以下の CREATE TABLE コマンドで作成することができる.

```
mysql> CREATE TABLE inv(term CHAR(32) primary key, postings VARCHAR(65536),
 totalfreq INT UNSIGNED);
Query OK, 0 rows affected (0.08 sec)

mysql> CREATE TABLE doc(docid INT UNSIGNED PRIMARY KEY, docname CHAR(255),
length DOUBLE);
Query OK, 0 rows affected (0.10 sec)
```

12.8. 転置索引のデータベースへの格納

テキストファイル people.txt から転置索引を作成し, 前節で作成した 2 つのテーブル inv と doc に格納するプログラムを作成する. 以下はテーブル inv への格納を行うプログラム InsertInvIndex.java のうち, データベースへの格納に関わる後半の部分である. (/kyozai/amaeda/ir/の下にプログラム全体がある).

```
/* InsertInvIndex.java */

        （中略）

            // 転置索引をデータベースに格納
            Connection db = null;

            // JDBC ドライバをロード
            Class.forName("org.gjt.mm.mysql.Driver");

            // データベースへ接続
            String url =
               "jdbc:mysql:///exp?useUnicode=true&characterEncoding=utf-8";
               // データベース名=exp, Unicode=使用, 符号化方法=UTF-8 を指定
            String user = "root";  // MySQL のユーザ名
            String password = "";  // MySQL のパスワード

            // データベースに接続し Connection オブジェクトを取得
            db = DriverManager.getConnection(url, user, password);

            // 転置索引テーブルを空にする
            db.createStatement().executeQuery("TRUNCATE inv;");

            // データを挿入する SQL 文を準備
            PreparedStatement stmt = db.prepareStatement(
               "INSERT INTO inv(term, postings, totalfreq) VALUES(?, ?, ?);");

            // すべての索引語について繰り返す
            Set entries = invIndex.entrySet();
            Iterator iter = entries.iterator();
            while (iter.hasNext()) {
```

```
        // 転置索引から, 索引語とポスティングリストの組を取り出す
        Map.Entry entry = (Map.Entry)iter.next();
        String term = (String)entry.getKey();
        String postingStr = ((PostingList)entry.getValue()).toString();
        int totalFreq = totalFreqHash.get(term).intValue();
        // SQL 文に, 挿入するデータを設定
        stmt.setString(1, term);
        stmt.setString(2, postingStr);
        stmt.setInt(3, totalFreq);
        stmt.executeUpdate();
      }
      stmt.close();
```

（後略）

なお, InsertInvIndex.java では単語の各文書における出現頻度だけでなく, 全文書にわたって足し合わせた総出現頻度(totalfreq)も求め, データベースの inv テーブルに登録するようにしている. 各単語の総出現頻度 totalFreq を保存しておくための HashMap が totalFreqHash である. HashMap にはクラスのインスタンスしか格納できないが, int 型の変数はプリミティブ型と呼ばれ, クラスのインスタンスではないため, totalFreq をいったん Integer 型に変換した上で格納が行われている. また, Integer 型から int 型を得るには intValue()メソッドを使用する. 以下に該当する部分を抜粋して載せる.

```
/* InsertInvIndex.java */
```

（中略）

```
// 単語の総出現頻度に格納
if(totalFreqHash.containsKey(term)){
  // Integer クラスのインスタンスから int 型の値を求め, 1 増やして格納
  int totalFreq = totalFreqHash.get(term).intValue();
  totalFreq++;
  totalFreqHash.put(term, new Integer(totalFreq));
}else{
  // 初めて現れたのならば頻度 1 で HashMap に格納
  totalFreqHash.put(term, new Integer(1));
}
```

（後略）

データベースに登録された総出現頻度 totalfreq は単語のソーティングに使うことができる. SQL 文で以下のように書けば, 総出現頻度が高い 5000 件の単語を取り出せる. DESC は降順で並べるという意味を表す.

```
SELECT * FROM inv ORDER BY totalfreq DESC LIMIT 5000;
```

逆に頻度の高すぎる単語を無視したい場合は, 以下のように書けば上位 100 件の単語を飛ばし, 101 番目から 5,000 件の単語を取得することができる. OFFSET は先頭から数えていくつ飛ばすかを表す. なお, OFFSET の指定は必ず LIMIT の指定よりも後に行わなくてはならないので注意すること.

```
SELECT * FROM inv ORDER BY totalfreq DESC LIMIT 5000 OFFSET 100;
```

また，以下のように書けば，総出現頻度が 10 以上の単語だけを取り出せる．

```
SELECT * FROM inv WHERE totalfreq >= 10;
```

【発展課題　1】 MySQL 上で，データベースに格納されたインデックスに格納された単語のうち，出現頻度の上位 50 件を出力せよ．

13. オプション課題

上記の課題をすべて完了して時間に余裕がある場合は，以下の課題を自由に選んで行うこと．これらの課題を行うことはプラス評価の対象となる．また，以下に挙げられていない独自の拡張を自分で考えて実装しても良い．その場合は特にプラス評価する．

【オプション課題 1】索引語の重み付け

10 章では，索引語の重みとして単語の出現頻度 TF（term frequency）を使用したが，他にもさまざまな重み付けの手法が考えられる．単語の重み付けの手法として最も一般的なものは TF・IDF（term frequency・inverse document frequency）である．これらの重み付け手法を実装し，TF を用いた場合と検索結果を比較し考察せよ．

【オプション課題 2】スニペットの表示

検索結果を表示する際に，各文書について 2〜3 行程度の抜粋（スニペットと呼ばれる）を表示することで，ユーザが目的の文書を見つけやすくなる．スニペットには，問合せに用いられた単語ができるだけ含まれていることが望ましい．このような機能を実装せよ．

【オプション課題 3】複合語への対応

Sen による形態素解析では，たとえば「東京都」は「東京」と「都」に分割されるため，検索語に「東京都」を指定しても検索されないという問題がある．逆に「東京都」をひとつの語とみなしてしまうと，「東京」では検索されなくなる．これらの問題を解決する機能を実装せよ．

ヒント：転置索引のポスティングリストに，索引語の出現位置の情報を付加することで実現できる．

【オプション課題 4】ブーリアン検索

10 章で作成した検索システムはベクトル空間モデルに基づくものであるが，実際の検索エンジンはブーリアンモデルに基づいているものが多い．9 章で作成した検索システムを，問合せにブーリアン演算子（AND, OR, NOT）を使用できるように拡張せよ．この場合，ランキングの機能は実装しなくても良い．

【オプション課題 5】転置索引の更新

7〜8 章で作成した索引付けのプログラムに，新たな文書を追加した際に索引を更新する機能を追加せよ．

【オプション課題 6】Web インタフェース

サーブレットなどを用いて，Web ブラウザから検索できるインタフェースを実装せよ．

参考文献・URL

情報検索に関する参考書:

[1] 情報検索アルゴリズム, 北研二他著, 共立出版, 2002.
[2] 情報検索と言語処理, 徳永健伸著, 東京大学出版会, 1999.
[3] 情報検索の基礎, Christopher D. Manning 他著, 岩野和生他訳, 共立出版, 2012.

Java に関する参考書:

[4] スッキリわかる Java 入門 第 2 版, 中山清喬他著, インプレス, 2014. ※
[5] 独習 Java 第 4 版, O'Neil, J.著, トップスタジオ訳, 翔泳社, 2008. ※
[6] プログラミング言語 Java 第 4 版, Arnold, K.他著, 柴田芳樹訳, ピアソンエデュケーション, 2007.
[7] Java 言語仕様 第 3 版, Gosling, J.他著, 村上雅章訳,ピアソンエデュケーション, 2006.
※がついているものが初級者向けの入門書であり, 学習に適している. それ以外はリファレンス的な書籍である.

Java に関する参考 URL:

[8] JavaTM 2 Platform Standard Edition 5.0 API 仕様, 2004. http://docs.oracle.com/javase/jp/1.5.0/api/
[9] 小田原大, @IT:Java TIPS -- オブジェクトを手軽にソートする, 2005.
http://www.atmarkit.co.jp/fjava/javatips/140java030.html

Sen に関する参考 URL:

[10] Sen － Java.net, http://java.net/projects/sen（Sen の公式サイト, 英語）
[11] 山田 剛一:Sen - 日本語形態素解析システム,
http://www.mlab.im.dendai.ac.jp/~yamada/ir/MorphologicalAnalyzer/Sen.html

JDBC に関する参考 URL:

[12] Synergy Marketing, Inc.:JDBC-TECHSCORE-. http://www.techscore.com/tech/J2EE/JDBC/index.html

MySQL に関する参考 URL:

[13] MySQL :: 世界でもっとも普及している, オープン ソース データベース. http://www-jp.mysql.com/
（MySQL の日本語公式サイト）
[14] MySQL :: MySQL 5.1 リファレンスマニュアル. http://dev.mysql.com/doc/refman/5.1/ja/index.html
[15] Synergy Marketing, Inc.:SQL -TECHSCORE-. http://www.techscore.com/tech/sql/

Jung に関する参考 URL:

[16] グラフを扱う Java ライブラリ「Jung」の紹介 - Twitter のグラフ構造を視覚化 - public static void main, 2008.
http://d.hatena.ne.jp/Kishi/20081125/1227610660
[17] 陽坂智佐:Jung で相関行列のグラフ化, 2008. http://txqz.net/blog/2008/10/25/1155