市区町村隣接関係自動作成ソフトウェアの提案

文教大学 情報学部 経営情報学科
田中 真一
a0p21115

平成 16 年 1 月 28 日

1 研究動機

私がこの研究をしようと思ったのは、小選挙区最適区割を導出する論文 [1] を作るための実験を手伝ったことにある。論文 [1] では、実験のためにいくつかのデータを必要とした。その 1 つのデータは全国の市区町村の人口である。そしてもう 1 つのデータは隣接関係のデータである。隣接関係のデータとは、「自分に隣り合っている市区町村はどこであるか」が書かれているデータである。

人口のデータと隣接関係のデータ。この 2 つには大きな違いが存在している。それはデータそのものを求めるときの手間である。人口のデータは総務省が行う国勢調査等の結果をそのまま利用することができる。しかしながら、隣接関係のデータは論文 [1] によると、存在しないようである。すなわち、実験者が区割を求める前に、データを作るという手間が生じてしまうのである。実際に論文 [1] においては、そのデータ作りに関連する作業に手を取られた。

また隣接関係は区割だけに必要となるものではなく、他の用途でも幅広く利用できるデータであると思われる。隣接関係のデータのみで何かをするのは難しいが、例えば最短路を見つける時などに地理的な構造を捉えるデータとして活用することができると思われる。その他、隣接関係のデータがあれば、人口のデータなどを組み合わせて市町村合併の可能性を探るときに利用できると思われる。

そこで本研究においては現在行っている隣接関係のデータの作り方などを見ながら、手間のかからない隣接関係データの作成の方法を考えていきたいと思う。ここでいう手間のかからないとは作り出す手間だけでなく、作り出されたデータが正確であり、その検証の時間なども削減できることを含んでいる。また実際にプログラムすることにより、ソフトウェアを作り出したいと思う。

2 章では現在の隣接関係の作成方法を説明し、改善すべき点を考える。3 章では入力データとして利用する数値地図の説明を加える。4 章では数値地図を利用したプログラムの内容についての解説をする。5 章では作成したソフトウェアの評価及び考察を加える。そして最後の 6 章では今後の課題について考えていく。

2　隣接関係

本研究では市区町村の地理的な隣接関係を自動的に抽出するソフトウェアの提案を行いたい。ここで隣接関係という用語が出てくるが、これは何も難しいことを意味している訳ではない。「A という市区町村の隣が B という市区町村」というのが隣接関係である。

本研究では「都道府県での市区町村の隣接関係」に話題を限定する。なぜならば、一つの応用分野として選挙区の画定での利用が考えられるからである。ただしこの限定を外すことは難しくない。

隣接関係は市区町村が陸上で境界線を共有している場合のみを取り上げる。海上などに市区町村が存在する場合 (島など) には単純に隣接関係を定義することが出来ないためである。この問題に関しては、提案するソフトウェアの機能の一部としてとして追加的に搭載したいと考えている。また、どの程度境界線を共有していれば隣接なのかとの問題もあると思うが、今回はどんなに少なくとも、例え数メートルであっても境界線を共有しているならば隣接関係が存在しているとして処理をする。十字路のように点で共有する可能性も考えられるが、そのような場合は、入力データである数値地図のデータに依存するものとする。

まずは市区町村の隣接関係を作る現在の方法を紹介しながら、隣接関係の詳細な説明及び、現在の作成法の課題を探っていきたい。

2.1　現状

ここでは想定される利用場面のひとつである小選挙区の最適区割の導出をする際、論文 [1] に利用された方法を書いていきたいと思う。はじめに都道府県や市区町村の境界線が入っている地図を用意する。なお基本的に本研究において地図は PC 上で利用できる形式になっているものとする。つまり学習用地図帳ように紙に印刷されたものではなく、DVD などに保存されているデジタルデータを念頭においている。

またここで利用する地図は単純に図が書いてあるのではなく、市、郡、町、村、海岸線、他都道府県との境界線などのデータがレイヤ (キャンバスの上に重ねる透明なセル画のようなもの) ごとにまとめられている。そのことにより、単純に紙の代わりに利用するだけではなく、パソコン上で地図を見る機能に特化したソフトウェアを利用したり、画像編集に使われる Adobe Illustrator や Adobe Photoshop などを利用することにより、手間を大幅に削減できる。

ここでは便宜的に図 1 のような市区町村を前提として説明をしていく。

まず図 1 の市区町村に 1 から連番を振っていく。結果、図 2 のようなものができあがる。

次に、ある市区町村に隣り合っている、言い換えるならば、境界線を共有している市区町村に線を引いていく。この線が存在するということは隣接関係があるということを表すこととなる。そのような処理を施したものを図 3 とする。

そして図 3 のようなものを参考にしながら、隣接している（線で繋がれている）市区町村のペアは 1 として、隣接していない（線で繋がれていない）市町村のペアを 0 として結

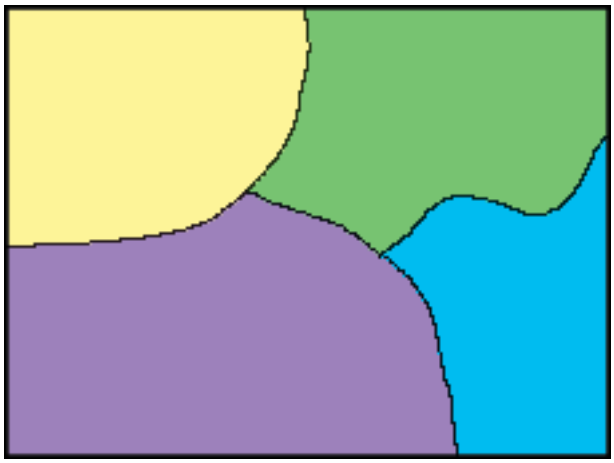

図 1: 前提となる市区町村

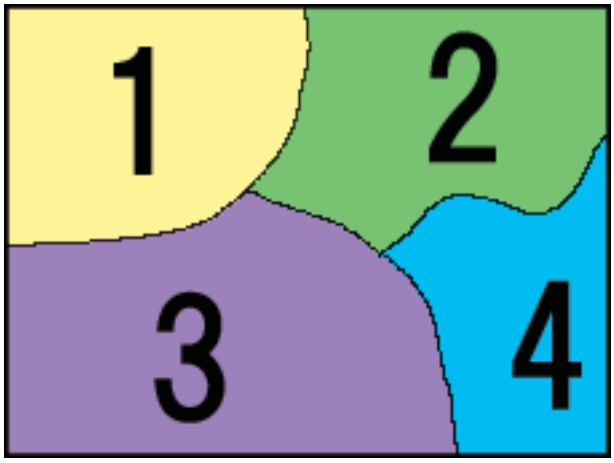

図 2: 連番を振った市区町村

果を出力する。また自分自身は 1 として出力するものとする。

　また出力の形式としては (1) のようなものを利用している。これはグラフ理論で隣接行列と呼ばれる形式であり、論文 [1] の解析プログラムの入力形式である。

$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{pmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{pmatrix} \end{array} \tag{1}$$

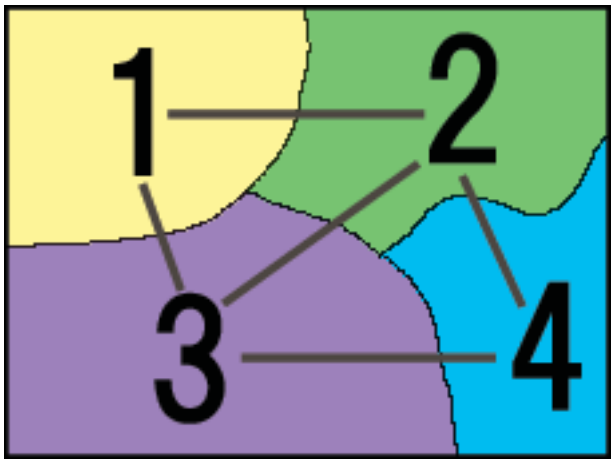

図 3: 隣接関係

2.2 作成上の問題点

現状では隣接関係を定める際に、地図を見て書くという手間のかかる作業を行っている。これは実際に利用する際においては前提条件のように島を含まないなど、排除している条件を全て含めるために、仕方のない部分もある。(島の場合には定期航路が存在しているかなどを確かめて隣接行列を作成する。) しかしながら現状のやり方では大変な部分も存在し、それは決して小さなものではない。

まずは時間がかかる点である。地図上で 1 つ 1 つの市区町村がどこと隣り合っているかを探さなくてはならない。ここでの見落としは後の全ての作業に影響を与えてしまうために、見落としが発生するのは大変に困る。

また、全てを手作業で行っている都合上、隣接行列の作成も手で入力している。この作業の段階でミスを犯すと、せっかく図 3 のようなものを準備しても、隣接行列を間違って作ってしまうことになる。

実際に実験をする時には、人口のデータと隣接関係を合わせるソフトウェアを利用する時に問題が発生することになる。このような時には、人口のデータが間違っていることはほぼなく、隣接関係のデータが間違っていることがほとんどである。このようなデータのズレによってかなりの時間が費やされることもあった。なぜならば隣接関係の構造上、一つミスがある場合には修正は複数になることが多いためである。

2.3 解決方法

今まで見てきたように人間の手に頼る方式には限界があり、利点も数多く存在しているが、それを越えてしまう弱点も存在をしている。この隣接関係を作り出す問題においては、入力すべきデータと出力すべきデータがはっきりしている。入力すべきデータとしてはこの後詳しく説明を加えるが、数値地図と呼ばれるものが適していると思われる。またその

数値地図を処理することにより、出力すべきデータである隣接関係を得ることができる。そのことから今回の問題に関してははじめにも書いたようにソフトウェアを利用するのが良いと考えられる。そこで、次の章では入力データに利用をする数値地図の解説をし、その次にソフトウェアについての説明をする。

3　入力データ

この章においては入力データとして利用をする数値地図に関しての詳細な説明をする。その後、数値地図を利用する上で生じる問題点を考えていきたい。

3.1　普通の地図との違い

一般的に地図というと以下のようなものを想像されるのではないかと思う。

図 4: 一般的に想像される地図

ここまでは正確に述べてこなかったが、数値地図という名前の地図にはいくつかのカテゴリが存在している。

先に表示した図 4 は「地図画像」タイプの地図である。このタイプの地図は地図データをラスタデータとして持っている。もっと分かりやすく例えるならば、普段目にしている紙に描かれた地図をスキャニングしただけの地図である。そのためパッと見ただけで分かるのがよい点なのだが、今回するような処理には向いていない。今回する処理とは隣接関係の抽出であり、地形や周りにある施設を知りたい訳ではない。隣接関係を抽出するため

だけに画像処理のアルゴリズムを導入するのは大げさすぎる。また、データのサイズが画像のデータであるので大きくなる、紙の代わりにディスプレイを利用しているだけになりかねない、といった問題もある。

一方で今回使用する地図データは正確には「数値地図 25000（行政界・海岸線）」と呼ばれるものである。これは先ほど見て頂いた一般的に地図と認識されやすいものではない。先ほどはラスタデータであったが、今回利用する地図はベクトルデータとしてデータを保持している。以下はその一例である。

```
M 533903 藤沢                    1   22   33   12    0  205
H2 1   22   33   12    0  204 9410 9902
N  1 2    1 9864 2594 0 3   -5    2    1    0    0    0    0    0    0
N  1 1    210000 2468 1 3   31  -30   -1    0    0    0    0    0    0
N  1 1    310000 2839 1 3   30  -29   -2    0    0    0    0    0    0
```

一見すると分かりにくそうに見えるのであるが、ラスタデータに比べると桁違いにデータのサイズが小さい。またデータ収録時に数値化、構造化された形でデータが保持されているので、今回の目的のような隣接関係のデータの抽出に使用するにはは都合がよい形式の地図であるといえる。その代償として、データを見るための専用のソフトウェアであるビューワー等を利用しないと人間が直感的に分かるような地図とはならないといったことが挙げられる。

3.2 数値地図の構造

数値地図 25000（行政界・海岸線）の構造は一般に公開されているが、どのような場合であっても対応できるように、若干構造が複雑になっている。そこで今回利用する部分に限って説明を加えたいと思う。

全国をいくつかのエリアに区切ってデータを収録しているのであるが、その基本となるのは小・中学校でよく使われる「2 万 5 千分 1 地形図」の 1 面である。この「2 万 5 千分 1 地形図の 1 面」を 2 次メッシュと呼ぶことにする。

しかしながら全国のデータを 2 次メッシュのみで収録すると、ファイル数が非常に多くなり利用しにくくなる。そこでいくつかの 2 次メッシュを集めて 1 次メッシュを作り、この 1 次メッシュファイルを元にしてファイルを収録している。ただし、この 1 次メッシュは単純に 2 次メッシュを集めただけであり、都道府県毎などにはなっていない。

3.2.1 2 次メッシュの中身

2 次メッシュファイルの中身は以下のような形で収録されている。

1. メッシュヘッダレコード

2. レイヤヘッダレコード

3. ノードレコード

4. ラインレコード

5. 座標値レコード

6. エリアレコード

7. エリア構成ライン番号レコード

1. のメッシュヘッダレコードは、メッシュコード、図名、各図形要素の総数を記録している。

```
M 533903 藤沢                    1    22    33    12     0   205
```

この場合、2 次メッシュコードは 533903 であり、図名は藤沢、などの情報が読み取れる。

2. のレイヤヘッダレコードは、レイヤコード、各図形要素の総数、作成年月を記録している。

```
H2 1    22    33    12     0   204 9410 9902
```

この場合、最初の作成年月は 94 年 10 月であり、最終更新年月は 99 年 02 月であることが読み取れる。

3. ノードレコードは、それぞれのノードに関する情報が書かれている。
ここでノードとは、「行政界を近似した折れ線の始点、終点。中間の点は含まない。」と定義されている。

```
N   1 2     1 9864 2594 0 3    -5     2     1     0     0     0     0     0     0
```

この場合、ノード番号は 1 であり、その X 座標値は 9864、Y 座標値は 2594 であり、521 の 3 本のラインと接続することがわかる。
5 の前に-がついているが、以下で説明する 4. のライレコードで定義してある順番を逆にたどることを示している。

4. のラインレコードは、各行政界線に関する情報を記録している。ここでラインとは、「行政界を近似した折れ線、ノードから中間の点を経てノードまでの一連の折れ線。図郭線も含む。」と定義されている。
また、5. の座標値レコードは 4. のラインレコードに従属する。そのために複数行に渡って記録されていることもある。

```
L  1 3    1     0    10    2114204   1114205    1     8 0  0
 9864 2594 9868 2556 9885 2518 9905 2486 9911 2480 9943 2467 9975 2473
10000 2468    0     0     0     0     0     0     0     0     0     0     0     0
```

この場合、1 行目はラインレコード、2、3 行目は座標値レコードである。

ラインレコードより、このラインは「郡市及び特別区界」であり、レイヤ内ではレイヤコード 1 であり、8 つの座標点から構成されていることがわかる。またこの線を中心線と見立てると、左側にある市区町村は、14204 であり、右側にある市区郡は 14205 である。ここで座標点とは、「ラインを構成する点。ノード及び中間の点を含んでいる。」と定義される。

また、2、3 行目は、8 つの座標点の正規化 X 座標、正規化 Y 座標を表している。ここで正規化座標とは、「地図上の各点の位置は、計測基図の図郭を基準にした座標軸系に変換 (正規化) されている。各座標値は図郭の左下を (0,0)、右上を (10000,10000) とした相対位置で示される。X は東西方向の正規化座標、Y は南北方向の正規化座標を表す。」とされている。

6. エリアレコードは、行政界線で囲まれた領域に関する情報を記録している。

ここでエリアとは、「1 つ以上のループで囲まれている連続した領域」と定義されている。

また、7. のエリア構成ライン番号レコードは 6. のエリアコードに従属する。ただし、5.6. の座標軸、ラインレコードとは違い、基本的にそれぞれが 1 行となる。特殊な場合においてのみ、7. エリア構成ライン番号レコードが複数行となる。

```
A  114205    1 2630 5000   1   9 神奈川県藤沢市
    1   9   27    4    5    1   31  -11  -17  -15   -3    0    0    0
```

この場合、行政コード 14205 の神奈川県藤沢市のエリアが 9 本のラインから作られていることがわかる。また、7. のエリア構成ライン番号レコードから、9 本のラインが、27 4 5 1 31 11 17 15 3 であることがわかる。

3.3 問題点

一見すると数値地図から隣接関係を作り出すソフトウェアを作成することは非常に簡単に思われる。なぜならば隣接関係がすでに数値地図に記述されているからである。はじめにも書いたように、同じ行政界を共有しているならば、それはすなわち隣接関係にあるということである。たとえば、4. ラインレコードを参照することにより、隣接関係が存在することが分かる。

しかしながら数値地図をただ単純に処理しただけでは、必要としている隣接関係を作り出すことはできない。それは大きく分けて 2 つの理由がある。1 つ目は数値地図を利用するために生じる構造的な問題、もう 1 つは小選挙区向けに作っているために存在する特有の問題である。

3.3.1 構造的な問題点

次の例を見ていただきたい。

```
A  114203    8   79 5752   1   3 神奈川県平塚市
    1   3   22  -14   -9    0    0    0    0    0    0    0    0    0
A  114203    9  137  223   1   2 神奈川県平塚市
    1   2   33   -8    0    0    0    0    0    0    0    0    0    0
A  114212   10   59 7313   1   3 神奈川県厚木市
    1   3   14   23  -12    0    0    0    0    0    0    0    0    0
A  114204   11 9961 2653   1   3 神奈川県鎌倉市
    1   3    2   30   -1    0    0    0    0    0    0    0    0    0
A  114203   12    5 2611   1   2 神奈川県平塚市
    1   2   20   -7    0    0    0    0    0    0    0    0    0    0
```

図 5: 複数回登場する市区群

2 次メッシュの基図は 2 万 5 千分 1 地形図である。そのため、ひとつの地図上に同じ市区群が複数回現れることがある。このような場合、単純に処理をしてしまうと、同じ市区群にも関らず別々の市区群として処理してしまう。数値地図を利用する中ではそれぞれのデータを保持するために、同じ市区郡を別のコードで呼ぶ方が便利なこともあるが、今回の利用に関しては複数回現れることは避けなくてはならない。

ここでは比較的見やすい構造をしている部分を例として取り上げたが、実際の使用の中では 4. ラインレコードの使用の部分で複数回同じ数字が出てきてしまうことが多い。また複数回現れるのは 2 次メッシュに限ってではない。さらに広範囲のデータである 1 次メッ

シュの中でも複数回現れてしまうことがある。そのためにこの問題を解決しなくては、隣接関係のデータを作り出すことはできない。

3.3.2　特有の問題点

日本の小選挙区制度では都道府県をひとつの固まりとして考えている。そのために他の都道府県の市区町村が隣り合っていたとしても、その市区町村間においては隣接関係が存在しないという処理を行う。

しかしながら、始めにも書いたように 1 次メッシュファイルは単純に 2 次メッシュを集めただけのものであり、都道府県毎のデータではない。そのために、2 つ以上の 1 次メッシュファイルを統合する機能と、他都道府県間においては隣接関係を認めないようにしなくてはならない。この問題点は隣接関係を作り出すだけであれば、さほど重要な点ではない。しかしながら、今回は都道府県ごとのデータを抽出する必要があるために解決しなくてはならない。

もう一点忘れてはならないのは、島などの関係である。前提としては島などの隣接関係は自動的に作り出せないかもしれないが、それは作り出すことができないだけであって、隣接関係が存在しないわけではない。そのために隣接関係を自動的には作れないが、簡単にそのことを知らせる機能も搭載しなくてはならない。また人間の手を煩わせないという目的からするならば、できる限り島等の隣接関係の入力もソフトウェア側で処理すべきであると考えられる。

4　ソフトウェア

1 つ前の章で入力データとして数値地図が利用できることを説明した。しかしながら数値地図をそのまま利用することができず、その点を解消しないと十分に利用することが出来ないことを確認した。そこでこの章では前章で発見した問題の解決法を解説し、さらにどのようなソフトウェアを提案したかを説明したいと思う。

4.1　解決法

前の章では数値地図の有用性を生かすには問題点があり、その問題点を解決しなくてはならないことを発見した。ここではそれぞれの問題に対しての解決法を提案していく。

4.1.1　構造的な問題点

構造的な問題点はデータの保存形式に注目をして解決をしてみたいと思う。まず入力するときには 4. ラインレコードを参照する。

```
L  1 3    1     0    10    2114204   1114205    1     8 0  0
```

ここで注目するべきデータは、14204 と 14205 である。つまり、1 つのラインにつき 2 つのデータのみ存在する。そこで、はじめにその 2 つのデータを始めに読み込んでしまう。(ラインデータは単独で存在をしないために、単純にデータを読み込むことが難しいため) 後に、左側 (14204) を検索のキーとして右側 (14205) を検索し、それを新たな行列として保存する。次に右側と左側を入れ替え、同様の処理を行う。このような処理を行うことにより手数はかかるが、重複したデータが出ないようにすることができる。

4.1.2　特有の問題点

特有の問題点とは都道府県ごとのデータを出力するというものであった。またこの部分は完全にオプションの部分であり、この機能をスキップすることにより地図に描かれているままの隣接関係を抽出できる。

さてここでの処理の仕方であるが、ここでは構造的な問題点で作り出した行列を利用する。この行列に入力されているデータは行政コードである。そのためはじめの 2 桁を比較することにより同一の都道府県であるかどうかを判断することができる。ただし、行政コード以外の数字が入ってしまうことがあるために、そのような特別な場合は、警告を出すようにしなくてはならない。

4.2　ソフトウェアの実際

ここでは、実際にどのようなソフトウェアを作成したかを説明したいと思う。また実装等に関しては、付録として付したソースコードを参照して頂きたい。

今回のソフトウェアではさまざまな処理を行うことを目的としたが、まず最低限装備しなければならない機能は、数値地図から隣接関係を取り出し、それを実験者が利用できる形式にすることである。その中で必要な処理はもちろんいくつもあるのであるが、このソフトウェアに特有のものは主に今まで説明を加えてきた問題点である。

しかしながら実際のその機能を実現しようとすると、多くの壁にぶつかることとなった。一番大きな壁となったのは、今までこのようなソフトウェアの開発を行ったことがなかったことである。今回は論文 [1] との連帯を念頭に置いていたので C 言語を利用したのであるが、自分の思ったような機能が実装することができずに非常に悔しい思いをした。

不馴れな点が多く、実装の段階で単純なアルゴリズムを多用し、C 言語らしいプログラムを書くことが困難であった。処理が単純であったためにあまり問題とはならなかったのであるが、複雑な処理を行うように変更を加える場合には、配列の使い方などに注意しないと、大幅に時間がかかるようになってしまう危険性が存在する。

そのような点もあり、今回の論文中ではもっとも適したソフトウェアを作り出し、多くの機能を搭載することが困難であり、非常に単純な機能のみを搭載したソフトウェアとなってしまった。

5　ソフトウェアの評価及び考察

ソフトウェアの評価についてはソフトウェアの実際の欄でも書いたように、満足がいくものを作り出すことが出来なかった。しかしながら、最低限持つべき機能を搭載することは出来たのではないかと思われる。それは数値地図から隣接行列を作り出すということである。確かにまだまだ実用の段階までは到達できていないと思うが、入力データとして数値地図を利用し、利用する部分をつきとめることが出来たこと自体は、大きな進展であると思われる。

6　今後の課題

もともと処理がそれほど多いものではないので、新たな機能を追加することができればと思う。例えば、現在の方式では入力のデータを利用者が自分で選ばなければならないのであるが、ソフトウェアが自動的に必要なファイルを検索するようになれば便利であると思う。また、今回は実現できなかった隣接関係をただ境界線を共有するだけではなく、他の方式で探し出すことができるようになれば、単にデータを提供するだけではなく、新たな発見につながるようなデータを提供できるようになると考えられる。

謝辞

この論文を書くにあたり、本当に多くの方の力をお借りしました。なかなか進まないにもかかわらず、いつも暖かく見守っていただいた根本先生。アイディアが浮かばないときに意見を出してくれたゼミ生の皆さん。そして、時には厳しくも暖かいアドバイスを頂いた OB/OG の皆さん。本当に多方面にわたり協力して頂きましてありがとうございました。

参考文献

[1] 根本 俊男, 堀田 敬介、「区割画定問題のモデル化と最適区割の導出」、オペレーション・リサーチ vol.47(4)(2003)p300-p306

[2] 山地 秀美、「C 言語文字列操作+ファイル入出力完全制覇」、技術評論社 (2002)

[3] 疋田輝雄、「C で書くアルゴリズム」、サイエンス社 (1995)

付録

```
#include <stdio.h>
#include <string.h>
```

```
#include <stdlib.h>

int main(void)
{
// 入出力ファイルに利用
char inputf[255];
char outf[255];
FILE *fpi;
FILE *fpo;

/*
* データを保存する配列、構造体の宣言
* 詳細はそれぞれのコメント文を参照の事
* なお、それぞれの項目は、FORMMBY.TXT の用語を使用している。
*/

struct mesh_head { // メッシュヘッダレコード用
int mesh; // 2 次メッシュコード
int node; // ノード数
int line; // ライン数
int area; // エリア数
int nokori; // レコード数
};

struct area_record { // エリアレコード用
int gyousei; // データ項目コード
char todouhuken[10]; // 都道府県名
char gunshi[20]; // 郡市・北海道の支庁名
char kutyou[20]; // 政令指定都市・区町村名
};

struct mesh_head syousai[100]; // メッシュヘッダレコード用の構造体
struct area_record naiyo[1000]; // エリアレコード用の構造体

int rinsetsu[5000][2]; // ラインレコード用の配列

char str[80];  // 読み込み用
char buf[80];  // 処理用
```

```
int i, j, k;  // カウント用

int countMESH; // 表示位置の指定
int countLINE; // 表示位置の指定
int countAREA; // 表示位置の指定
int nagasa; // kuwari の整列用

int min, t, max; //ソート時に利用

int kuwari[5000][30];
int a[100][30];

/* 配列 kuwari[][] の初期化 */

for(i=0; i<5000; i++){
for(j=0; j<30; j++){
kuwari[i][j] = 0;
}
}

for(i=0; i<100; i++){
for(j=0; j<30; j++){
a[i][j]=0;
}
}

/* ファイルの処理 */
printf("入力ファイル名を入力してください\n");
scanf("%s",inputf);
printf("出力ファイル名を入力してください\n");
scanf("%s",outf);

fpi = fopen(inputf, "r");
if ( fpi == NULL) {
printf("ファイルをオープンできませんでした。\n");
return 0;
}
```

```
fpo = fopen(outf, "w");
if ( fpo == NULL) {
printf("ファイルをオープンできませんでした。\n");
return 0;
}

i = 0;
j = 0;
k = 0;

/* データの読み込み */
while ( fgets (str, 80, fpi) != NULL) {
strncpy(buf, str, 2);
buf[2] = '\0';

if (strcmp(buf,"M ") == 0){  //メッシュヘッドレコードの場合
strncpy(buf, str+2, 6);
buf[6] = '\0';
syousai[i].mesh = atoi(buf);

strncpy(buf, str+2+6+20+3, 5);
buf[5] = '\0';
syousai[i].node = atoi(buf);

strncpy(buf, str+2+6+20+3+5, 5);
buf[5] = '\0';
syousai[i].line = atoi(buf);

strncpy(buf, str+2+6+20+3+5+5, 5);
buf[5] = '\0';
syousai[i].area = atoi(buf);

strncpy(buf, str+2+6+20+3+5+5+5+5, 5);
buf[5] = '\0';
syousai[i].nokori = atoi(buf);

i++;
```

```
}
else if (strcmp(buf,"L ") == 0){  //ラインレコードの場合
strncpy(buf, str+2+2+2+5+6+5+1+5+1, 5);
buf[5] = '\0';
rinsetsu[j][0] = atoi(buf);
rinsetsu[j+1][1] = atoi(buf);

strncpy(buf, str+2+2+2+5+6+5+1+5+1+5+5, 5);
buf[5] = '\0';
rinsetsu[j][1] = atoi(buf);
rinsetsu[j+1][0] = atoi(buf);

j=j+2;
}
else if (strcmp(buf,"A ") == 0){  //エリアレコードの場合
strncpy(buf, str+2+2, 5);
buf[5] = '\0';
naiyo[k].gyousei = atoi(buf);

strncpy(buf, str+2+2+5+5+5+5+4+4, 8);
buf[8] = '\0';
strcpy(naiyo[k].todouhuken, buf);

strncpy(buf, str+2+2+5+5+5+5+4+4+8, 16);
buf[16] = '\0';
strcpy(naiyo[k].gunshi, buf);

strncpy(buf, str+2+2+5+5+5+5+4+4+8+16, 16);
buf[16] = '\0';
strcpy(naiyo[k].kutyou, buf);

k++;
}
}

/* 画面上への出力 */
countMESH = i;
countLINE = j/2;
```

```
countAREA = k;

printf("出力情報\n");
printf("メッシュヘッダレコード数は%d です。\n",countMESH);
printf("ラインレコード数は%d です。\n",countLINE);
printf("エリアレコード数は%d です。\n\n",countAREA);
printf("-----\n");

countLINE=0;
countAREA=0;

for(i=0; i<countMESH; i++){
/* メッシュヘッドレコードの出力 */
printf("Mesh:%d\n",syousai[i].mesh);
printf("Node:%d\n",syousai[i].node);
printf("Line:%d\n",syousai[i].line);
printf("Area:%d\n",syousai[i].area);

/* ラインコードの出力 */
printf("\n");
for (j=0; j<syousai[i].line; j++){
printf("左:%d\t",rinsetsu[countLINE][0]);
printf("右:%d\n",rinsetsu[countLINE][1]);
countLINE = countLINE+2;
}
printf("\n");

/* エリアレコードの出力 */
for(k=0; k<syousai[i].area; k++){
printf("データ項目コード:%d\n",naiyo[countAREA].gyousei);
printf("都道府県名:%s\n",naiyo[countAREA].todouhuken);
printf("群市:%s\n",naiyo[countAREA].gunshi);
printf("区町村:%s\n",naiyo[countAREA].kutyou);
countAREA++;
}
printf("\n");
}
/* 画面上への出力　終了 */
```

```
/*
隣接関係の読み取り
rinsetsu から kuwari にデータを移す。
その際に、不必要なデータ及び、重複が起こらないようにする。
*/
nagasa = -1;

for ( i=0; i<5000; i++){
k = 0;
if((rinsetsu[i][0] != 0) && (rinsetsu[i][0] != 88888) && (rinsetsu[i][0] != 99999)){
nagasa++;
kuwari[nagasa][k] = rinsetsu[i][0];
if((rinsetsu[i][1] != 88888) && (rinsetsu[i][1] != 99999)){
k++;
kuwari[nagasa][k] = rinsetsu[i][1];
}
}

j=i+1;
for ( j; j<5000; j++){
if(kuwari[nagasa][0] == rinsetsu[j][0]){
if((rinsetsu[j][1] != 88888) && (rinsetsu[j][1] != 99999)){
k++;
kuwari[nagasa][k] = rinsetsu[j][1];
}
rinsetsu[j][0] = 0;
}
}
}
/* 隣接関係の読み取り 終了 */

/* 並び替えの処理を以下で行う。
並び替えた結果は a[][] に記憶される。
はじめに kuwari[][0](列) の項目で並び替え、
その後に kuwair[x][項目数](行) の項目で並び替える。
*/
```

```
// 事前準備
for(i=0; i< nagasa; i++){
a[i][1]=kuwari[i][0];
}

// 行のソート
for (i=0; i < nagasa-1; i++){
min = i;
for (j = i+1; j <= nagasa-1; j++){
if (a[j][1] < a[min][1]){
min =j;
}
}
t = a[min][1]; a[min][1] = a[i][1]; a[i][1] = t;
}

// 列の項目数の計算、及び a[][] へのデータコピー
for(i=0; i<nagasa; i++){
for(j=0; j<nagasa; j++){
if(a[i][1] == kuwari[j][0]){
for(k=0; k<30; k++){
if(kuwari[j][k] == 0){
a[i][0] = k;
break;
}
else{
a[i][k+1] = kuwari[j][k];
}
}
}
}
}

// 列のソート
for(k=0; k<nagasa; k++){
for (i=2; i < a[k][0]; i++){
min = i;
for (j = i+1; j <= a[k][0]; j++){
```

```
if (a[k][j] < a[k][min]){
min =j;
}
}
t = a[k][min]; a[k][min] = a[k][i]; a[k][i] = t;
}
}

//同一列内での同一している数字の削除
for(i=0; i<nagasa; i++){
for(j=2; j<a[i][0]; j++){
if(a[i][j] == a[i][j+1]){
for(k=j+1;k<a[i][0];k++){
a[i][k]=a[i][k+1];
}
a[i][0]--;
}
}
}

// 最大表示幅
max = a[i][0];
for (i=1; i <= nagasa; i++){
if(max<a[i][0]){
max = a[i][0];
}
}
printf("max:%d\n",max);
/* 隣接関係の解析 終了 */

/* 隣接関係の解析の結果*/
for(i=0; i<nagasa; i++){
for(j=0; j<30; j++){
printf("%d,",kuwari[i][j]);
}
printf("\n");
}
```

```
printf("-----\n");

for(i=0; i<nagasa; i++){
printf("%d,==,",a[i][1]);
for(j=2; j<=a[i][0]; j++){
printf("%d,",a[i][j]);
}
if(max-a[i][0] > 0){
for(k=0; k<max-a[i][0]; k++){
printf("0,");
}
}
printf("\n");
}
/* 隣関係の解析の結果 終了 */

for(i=0; i<nagasa; i++){
fprintf(fpo,"%d,==,",a[i][1]);
for(j=2; j<=a[i][0]; j++){
fprintf(fpo,"%d,",a[i][j]);
}
if(max-a[i][0] > 0){
for(k=0; k<max-a[i][0]; k++){
fprintf(fpo,"0,");
}
}
fprintf(fpo,"\n");
}

/* 終了処理 */
fclose(fpi);
fclose(fpo);
return 0;
}
```