2007－AL－111 (5)
2007／3／9

2次元線形リスト構造のポインタ誤りを検出・特定・修正するアルゴリズムの提案

上村 尚史† 新森 修一‡

†鹿児島純心女子短期大学
‡鹿児島大学理学部

概要: 本稿では, 2次元線形リストにおける単一のポインタ誤りを検出・位置の特定・修正するアルゴリズムを提案する. 2次元線形リストは広く用いられているデータ構造であるが, そのポインタに誤りが発生した場合には, データ構造そのものの信頼性に多大な影響を及ぼしてしまう. 本研究では, その構造的特徴に注目し, データ構造に関する単純な値を利用することで単一のポインタ誤りを 100 %の確率で修正することに成功した. 行数を n, 列数を m とすると, その時間計算量は $O(\max\{ n^2m, nm^2 \})$ であった.

Proposal of the algorithm that detect, identify and correct a single pointer error in 2-dimensional linear list

Naofumi KAMIMURA† Shuichi SHINMORI‡

† Kagoshima Immacurate Heart College
‡ Faculty of Science,Kagoshima University

Abstract. We propose the algorithm that detect, identify and correct a single pointer error in 2-dimensional linear list. The occurence of the pointer error exerts a large influence on the reliability of the data structure. We succeeded in the complete correction of a single pointer error by paying attention to the structural feature, and using a simple value concerning the data structure. When the number of rows and columns are assumed to be n and m respectively, the computational complexity of the algorithm is $O(\max\{ n^2m, nm^2 \})$.

1 はじめに

情報化が広く浸透している現在,「データ」は企業等の業務活動から日常生活に至るまで必要不可欠なものとなっている. そして, データによっては, その誤りが非常に大きな影響を及ぼすことも考えられ, これに対処する研究がなされている. 本研究は, データ構造とそれに関連したアルゴリズムに関する研究であるが, 特に2次元線形リストを研究対象としている. 本研究では, まず, 2次元線形リストの信頼性を向上させるよう, その構造に工夫を加えている. そして, そのデータ構造内のポインタに誤りが発生した場合に, その検出・特定・修正を行うアルゴリズムを提案している.

2 データ構造とポインタ誤り

ここでは, まず, 本研究が対象としているデータ構造について述べる.

2次元線形リストは, 行方向と列方向の両方に対して, ある要素と次の要素をポインタによって繋ぎ合わせた線形リストである.

行 (列) の先頭には行 (列) のノードの情報 (科目名や学生名等) を持つレコードがあり, レコード同士は線形リストをなすものとする. リストの先頭には特別なレコードを置き, これをヘッダレコードと呼ぶことにする.

さらに, レコードには, それに属するノードがあり, それらのノードも線形リストを形成する. ノードの線形リストもレコードと同様に循環型であるので, 先頭には特別なノード (ヘッダノードと呼ぶことにする) を置く. ヘッダノード以外のノード (データノードと呼ぶことにする) にデータが保持されている.

通常の2次元線形リストのままでは, その操作性に難があるため, 各ノードに, それが属する行や列の情報を付加することが多い. 本研究では, さらにいくつかの変更を施し, それを2次元線形多重リストと呼ぶことにする. その構成例を Fig.1 に示す.

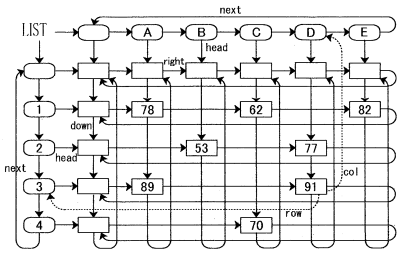

Fig. 1: New 2-dimensional linear list.

本研究で提案した2次元線形多重リストはヘッダノード同士もポインタで繋がれ NULL となっているポインタが一切ないことが特徴である．

さらに，本研究で提案する2次元線形多重リストでは，以下のような各種の情報を保持する．

- 行 (列) ヘッダレコードには，行 (列) レコード数を保持する．
- 行 (列) ヘッダノードには，その行 (列) のノード数を保持する．
- 共通ヘッダノードには，データノード数を保持する．

これらの値を2次元線形多重リストにおける構造に関する値と呼ぶことにする．

本研究では，データ構造や操作に関連するアルゴリズムは，全て，C言語によりプログラミング化している．C言語によるプログラムでは，2次元線形多重リスト，レコード，ノードをそれぞれ以下のように構造体を用いて定義する．

＝ 2次元線形多重リスト ＝

```
1.  typedef struct _LIST{
2.    RCRD  *col_rcd;          /* 列レコードリストの先頭を指すポインタ */
3.    RCRD  *row_rcd;          /* 行レコードリストの先頭を指すポインタ */
4.  } LIST;
```

＝ レコード ＝

```
1.  typedef struct _NODE{
2.      DATA    data;          /* データ */
3.      struct _NODE *down;    /* 次のノード (列方向) を指すポインタ */
4.      struct _NODE *right;   /* 次のノード (行方向) を指すポインタ */
5.      struct _RCRD *col;     /* 属する列のレコードを指すポインタ */
6.      struct _RCRD *row;     /* 属する行のレコードを指すポインタ */
7.  } NODE;
```

＝ ノード ＝

```
1.  typedef struct _RCRD{
2.      short   type;          /* 列、行どちらのレコードであるかを示す値 (c/r) */
3.      char   *id;            /* 識別子 (文字列) */
4.      char   *label;         /* ラベル (文字列) */
5.      NODE   *head;          /* ノードリストの先頭のノードを指すポインタ */
6.      struct _RCRD *next;    /* 次のレコードを指すポインタ */
7.  } RCRD;
```

以降，ある順にポインタを辿る場合には，便宜上「. (ピリオド)」を用いて表記することとする．

本研究では，2次元線形多重リストにおける単一の誤りを考察の対象としており，次の仮定を設けている．

仮定 1 対象とするデータ構造には，最大1つのポインタ誤りが存在する．

すなわち，誤りはポインタのみに発生し，保持している「値」は常に正しいことを仮定して，その誤りの検出・特定・修正を行うアルゴリズムを提案する．

ここで，ポインタの誤りを大別すると，以下の 4 つが考えられる．

1 ポインタが NULL (空値) となる (これをポインタの断絶と呼ぶ).

2 異なる行 (列) のノードを指す (これを行 (列) のズレと呼ぶ).

3 同じ行 (列) の，本来よりも前のノードを指す (これを飛び越しと呼ぶ).

4 同じ行 (列) の，本来よりも後のノードを指す (これをループと呼ぶ).

3 レコードにおけるポインタの誤り

本章と次章では，2 次元線形多重リストに含まれるポインタのそれぞれに対して，その誤りを検出・特定・修正するアルゴリズムを提案する．時間計算量に関する記述においては，行レコードの総数を n，列レコードの総数を m，データの総数を N とする．

Fig.2 に示すように，col_rcd ポインタはリストにおける列ヘッダレコードを指すポインタ，row_rcd ポインタはリストにおける行ヘッダレコードを指すポインタである．また，以降では Fig.2 と同様に，対象とするポインタを太い矢印で示すこととする．

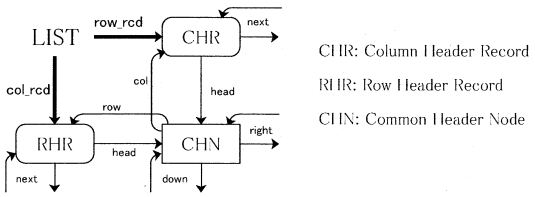

Fig. 2: col_rcd and row_rcd pointers.

以下に，col_rcd，row_rcd ポインタの誤りの検出・特定・修正の手順を示す．

手順 3.1 (ヘッダレコードを指すポインタの誤りの検出・特定・修正)

1. **if** (row_rcd = NULL) **or** { (row_rcd.id = "") **and** (row_rcd.type = "r") } **then**
2. row_rcd ← col_rcd.head.row;
3. **else if** (col_rcd = NULL) **or** { (col_rcd.id = "") **and** (col_rcd.type = "c") } **then**
4. col_rcd ← row_rcd.head.col;
5. **end if;**

まず，検査対象となっているポインタが NULL となっているかどうかチェックする．さらに，ヘッダレコードはその ID が空となっている点が他のレコードと異なるため，その特徴を利用する．修正には，共通ヘッダノードの row (col) ポインタを用いればよい．

next ポインタのチェックは，col_rcd，row_rcd ポインタのチェックを行った後に行う．next ポインタは Fig.3 に示すように，レコードリストにおける次のレコードを指すポインタである．

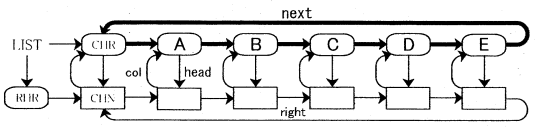

Fig. 3: next pointers of column records.

まず，列ヘッダレコードには，列レコードの総数が保持されているので，その数分だけ next ポインタを辿る．ただし，途中でヘッダレコードに戻ってきた場合には打ち切るものとする．

手順 3.2 (あるレコードの next ポインタが本来指すべきレコードの探索)

1. R:対象となるレコード
2. **if** R.type = 'c' **then**
3. return R.head.right.col;
4. **else if** R.type = 'r' **then**
5. return R.head.down.row;
6. **else**
7. return NULL;
8. **end if**;

さて，レコードの数だけ next ポインタを辿ったとき，途中で打ち切られることなく，列ヘッダレコードにちょうど戻って来れば，仮定 1 より，next ポインタに誤りはないと判断してよい．そうでなければ，いずれかの next ポインタに誤りが発生していることになる．

誤りが検出された場合には，next と R.head.right.col (R.head.down.row) の比較によって誤り位置の特定と修正を行う．

4 ノードにおけるポインタの誤り

レコードのポインタに誤りがなければ，ノードのポインタのチェックを行う．

まず，共通ヘッダノードについて誤りがないか確認する．Fig.4 に対象とするポインタを示す．

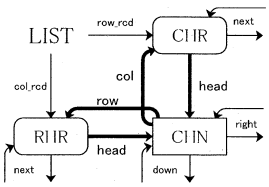

Fig. 4: head, row and col pointers of common header node.

共通ヘッダノードのポインタの誤りの検出・特定・修正は手順 4.1 のように行う．

手順 4.1 (共通ヘッダノードのポインタの誤りの検出・特定・修正)

1. L:リスト
2. **if** L.row_rcd.head = NULL **then**
3. L.row_rcd.head ← L.col_rcd.head;
4. **else if** L.col_rcd.head = NULL **then**
5. L.col_rcd.head ← L.row_rcd.head;
6. **end if**;
7. **if** L.row_rcd.head ≠ L.col_rcd.head **then**
8. **if** L.row_rcd = L.row_rcd.head.row **and** L.col_rcd = L.row_rcd.head.col **then**
9. L.col_rcd.head ← L.row_rcd.head;
10. **else**
11. L.row_rcd.head ← L.col_rcd.head;
12. **end if**;
13. **else**
14. **if** L.row_rcd.head.row ≠ L.row_rcd **then**
15. L.row_rcd.head.row ← L.row_rcd
16. **else if** L.row_rcd.head.col ≠ L.col_rcd **then**
17. L.row_rcd.head.col ← L.col_rcd
18. **end if**;
19. **end if**;

6. では，行・列ヘッダレコードの head ポインタが同じノードを指しているかを確認している．もし，異なるノードを指していれば，仮定 1 より，いずれかの head ポインタに誤りがあることになる．

レコードと共通ヘッダノードのポインタに誤りがなければ，列レコードのヘッダノードにおけるポインタ誤りの検出・特定・修正を行う．Fig.5 に対象とするポインタを示す．

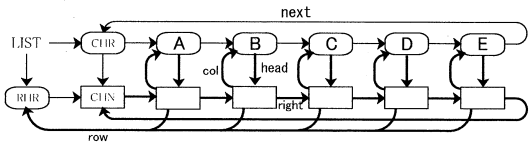

Fig. 5: head, row, col and right pointers of header node.

まず，right ポインタの確認を行う．列レコードの数とそのヘッダノードの数は等しいため，その数だけ right ポインタを辿る．このとき，本来指すべきノードを A.col.next.head として得ること以外は，next ポインタとほぼ同様である．

right ポインタに誤りがなければ，次は，row，col，head ポインタの確認を行う．レコードリストとヘッダノードリストを並行して交互に辿ることで，これらのポインタの誤りを検出・特定・修正することができる．

手順 4.2 (列レコードのヘッダノードの row，col，head ポインタの検出・特定・修正)

```
1.  R:対象となるレコード
2.  rp ← R.col_rcd;
3.  np ← rp.head;
4.  while np.right ≠ R.col_rcd.head do
5.    begin
6.      if np.right.row ≠ R.row_rcd then
7.        np.right.row ← R.row_rcd;
8.      else if np.right.col ≠ rp.next then
9.        np.right.col ← rp.next;
10.     else if rp.next.head ≠ np.right then
11.       rp.next.head ← np.right;
12.     else;
13.       np ← np.right;
14.       rp ← rp.next;
15.     end if;
16.   end;
17. end while;
```

right ポインタに誤りがない場合に，それと next ポインタを利用して残りのポインタに誤りがないか確認し，誤りがあれば修正を行っている．

レコードと共通ヘッダノードのポインタに誤りがなければ，列レコードのヘッダノードにおけるポインタ誤りの検出・特定・修正を行う．データノードには right，down，row，col の 4 つのポインタがある．まず，行を固定してその行に属するデータノードの right 及び row ポインタのチェックを行う．そのチェックをすべての行に対して行い誤りがなければ，すべての列に対して，それに属するデータノードの down 及び col) ポインタのチェックを行う．

ここでは，ある行 (ここではそのレコードを rcd とする) に属するデータノードの right 及び row ポインタのチェックについて述べる．

ヘッダノードには，それに属するノードの総数が保持してあるので，そのノードの数分だけ right ポインタを辿る．ただし，途中でヘッダノードに戻ってきた場合には打ち切るものとする．

また，途中で right ポインタが NULL となっていれば，手順 4.3 により本来指すべきノードを探索し，right ポインタを修正して処理を終了する．

手順 4.3 (あるノードの right ポインタが本来指すべきノードの探索)

```
1.  A:対象となるノード
2.  rp ← A.col.next;
3.  while rp ≠ col_rcd do
4.    begin
5.      np ← rp.head.down;
6.      n ← rp.head.data;
7.      while n > 0 do
8.        begin
9.          if np.row = A.row then return np;
10.         np = np.down;
11.         n ← n − 1;
12.         if np = rp.head then break;
13.       end;
14.     end while;
15.     rp ← rp.next;
16.   end;
17. end while;
17. return A.row.head;
```

right ポインタに誤りがなければ，ちょうどヘッダノードに戻ってくるが，行のズレやループが発生している場合には，ヘッダノードに戻って来ない．また，飛び越しが発生している場合には，途中でヘッダノードに戻ってくることになる．

ヘッダノードに戻って来ない場合，right ポインタに誤りがあることになり，仮定 1 より他のポインタには誤りがないものとしてよい．したがって，再度，right ポインタを辿りながら row ポインタが正しいレコードを指しているか確認することで行のズレを検出できる．

手順 4.4 (ある行のズレの原因である right ポインタの特定)

```
1.  R:対象となる行のレコード
2.  n ← R.head.data + 1;
3.  np ← R.head;
4.  while n > 0 do
5.    begin
6.      if np.right.row ≠ R then return np;
7.      np ← np.right;
8.      n ← n − 1;
9.    end;
10. end while;
11. return NULL;
```

ヘッダノードに戻って来ない場合に行のズレが検出されなければループが発生していることになる．そこで手順 4.5 により，ループの発生原因となっている right ポインタを探索する．

手順 4.5 (ある行のループの原因である right ポインタの特定)

```
1.  R:対象となる行のレコード
2.  n ← R.head.data + 1;
3.  np ← R.head;
4.  while n > 0 do
5.    begin
6.      if np = np.right then return np;
```

```
7.        while rp ≠ np.right.col do
8.          begin
9.            if rp = list.col_rcd then return np;
10.           rp ← rp.next;
11.         end;
12.       end while;
13.       np ← np.right;
14.       n ← n − 1;
15.     end;
16.     if np.right = R.head then return NULL;
17.   end while;
18.   return NULL;
```

問題となっている行のノードリストのノードを1つずつ辿りながら，同時に，その次のノードの列まで列レコードを辿っている．このとき，ループの原因となっている `right` ポインタに到達すると，変数 rp はその `right` ポインタが指す次のノードの列に到達することなく列ヘッダレコードに到達する．こうして，行のズレやループの原因となっている `right` ポインタが特定できれば，手順4.3によって本来指すべきノードを探索し，修正して，終了する．

`right` ポインタを辿っている途中で，ノードの数分だけ辿る前にヘッダノードに戻って来た場合は飛び越しが発生していることになる．飛び越しの原因となっている `right` ポインタは，ループのときのように，その位置のみを簡単に特定することはできない．したがって，手順4.3によって本来指すべきノードを探索し，順次確認していく．そして，誤り位置が特定されれば，修正して，終了する．

`right` ポインタに誤りが検出されない場合には，`row` ポインタのチェックを行う．ある行に属しているすべてのノードの `row` ポインタは，その行のレコードを指しているはずである．そこで，ノードリストの先頭から末尾まで `right` ポインタを辿りながら，それが指すノードの `row` ポインタを確認する．もし，一致していなければ修正し，終了する．

5 適用順序と時間計算量

本研究で提案した，構造に関する値によるポインタ誤りの検出・特定・修正のアルゴリズムの適用順序は，以下のようになる．

1. 行・列ヘッダレコードを指す `row_rcd`・`col_rcd` ポインタの誤りを手順3.1によって検出・特定・修正を行う． この時間計算量は $O(1)$ である．
2. 行・列レコードの `next` ポインタの誤りを手順3.2によって検出・特定・修正を行う． この時間計算量は $O(\max\{ n, m \})$ である．
3. 共通ヘッダノードを指す2つの `head`，共通ヘッダノードの `row`・`col` ポインタの誤りを手順4.1によって検出・特定・修正を行う． この時間計算量は $O(1)$ である．
4. 行・列ヘッダノードを指す `head`・`down` (行ヘッダノードの場合)・`right` (列ヘッダノードの場合)，行・列ヘッダノードの `row`・`col` ポインタの誤りを手順4.2によって検出・特定・修正を行う． この時間計算量は $O(\max\{ n, m \})$ である．
5. すべての行・列レコードにおいて，それに属するデータノードそれぞれの `right`・`down`・`row`・`col` ポインタの誤りの検出・特定・修正を行う． この時間計算量は $O(\max\{ n^2m, nm^2 \})$ である．

6 シミュレーション

本研究で提案するアルゴリズムをC言語を用いて実装し，「行数6，列数5，データ数19，ポインタ総数152本」のサンプルデータを用いたシミュレーションの結果を Table 1 に示す．Table 1 より，検出率，修正率ともに100％であることが確認できた．

Table 1: Numelical result for single pointer error.

number of cases :	3,326	
dettection times :	3,326	100 %
correction times :	3,326	100 %

本研究は単一のポインタ誤りを対象としているが，複数のポインタ誤りに対してもシミュレーションを行っている．誤りの個数に関わらずほぼ100 %の確率で検出が可能であり，誤りが3個程度までであれば90 %以上の確率で修正が可能であるという結果が得られており，複数のポインタ誤りに対しても十分有効なアルゴリズムであるといえる．

7 まとめ

本研究では，2次元線形多重リスト内にポインタ誤りは最大で1つであるという仮定のもとで，そのポインタ誤りの検出・特定・修正を試を行うアルゴリズムを提案した．

その準備として，まず，既知の2次元線形リストに構造的な工夫を加え，その信頼性を高めている．そして，本来は同一のアドレスを指すはずのポインタ同士を比較する，あるいは，レコードやノードの数だけポインタを辿った結果を利用するなどの手法を用いている．

単一のポインタ誤りに対しては，本研究で提案した独自のアルゴリズムによって100 %の確率で修正が可能であり，シミュレーションを行ってそのことを確認している．行数を n，列数を m とすると，その時間計算量は $O(\max\{\, n^2m, nm^2 \,\})$ であった．

また，複数のポインタ誤りに対しても十分有効なアルゴリズムであることがシミュレーションによって分かった．いくつかの対応できない場合を確認しているが，今後の研究課題としたい．

参考文献

[1] 茨木俊秀, アルゴリズムとデータ構造, 昭晃堂, 1989

[2] 茨木俊秀, Cによるアルゴリズムとデータ構造, 昭晃堂, 1999

[3] 近藤 嘉雪, 定本 Cプログラマのためのアルゴリズムとデータ構造, SOFTBANK BOOKS, 1998

[4] Niklaus Wirth, アルゴリズムとデータ構造, 近代科学社, 1990，浦昭二, 国府方久史 共訳

[5] 電子情報通信学会編, 斎藤信男, 西原清一, データ構造とアルゴリズム, コロナ社, 1998