Struktur data MultiList

by Tedy Setiadi

Submission date: 02-Nov-2020 05:47AM (UTC+0700)
Submission ID: 1432995136
File name: Pemilihan_Struktur_data_MultiList_Representasi.docx (791.91K)
Word count: 1529
Character count: 8842

Pemilihan Strutur Data MuliList Representasi Pointer untuk Pengembagan Program Pengolahan *Sparse* Matriks

Tedy Setiadi, Eko Mursito, I Nyoman Kertajaya
Jurusan Teknik Informatika, UniversitasAhmad Dahlan
Juruan Fisika Institut Teknologi Bandung
Jurusn Matematika Universitas Malang
email: tedz68@yahoo.com

Abstrak

Pada pengembangan program pengelolaan matriks tentu kebnyakan pemrogram memilih menggunakan struktur data *matrix* atau array dua dimensi megingat secara natural sesui dengan karakteristiknya selain kemudahan dalmpengembangan alogaritmanya. Namun, bia matriks yag dikelola adalah *sparse* matriks maka pemilihan struktur data ini kurang cocok sebab akan terjadi pemborosan memori. Makalah ini menjelaskan pengembangan program pengelolan sparse matriks.

Kata Kunci: multi list, pointer, *sparse matriks*

1. Pendahuluan

1.1. Matriks

Matriks adalah susunan empat persegi panjang dari elemen-elemenya yang dinotasikan Amxn. Matriks A diatas mempunyai m baris n kolom maka A disebut metriks dengan orde m x n.

Adpun beberapa istilah yang dikenal pada matriks, antar lain

- Matrik bujursngkar yaitu matrik dngn bayaknya baris dan kolom sama
- Matriks nol yaitu mtriks dengn semu elemennya bernilai nol
- Matriks identitas yaitu matriks yang semua elemen diagonalnya bernialai 1 sedngkan elemen lainnya bernilai nol
- *Sparse* matrik (matrik jarang) yaitu matriks dimana sebagian besar elemnennya bernilai nol, dengan catatan matriks ini umumnya berukuran besr mencapai ratusan bahkan ribuan baris baris dan kolom.[1]

Matrik dapat dikenakan operasi-operasi antara lain:

Penjumlahan: Jika A dan B matriks orde m x n, maka A+B adalah matriks m x n dimana (A+B)ij= Aij+Bij (dibaca "elemen ke ij dari matriks A+B adalah berisi penjumlahan elemen ke-ij A dengan elemen ke-ij B")

Perkalian skalar : Jika A matriks m x n dan c suatu skalar, maka cA adalah matriks m x n yang berisi perkalian c dengan setiap elemen pada A; yaitu (cA)ij=cAij

dengan menggunakan struktur data multilist representasi pointer. Dengan pemilihan struktur data ini maka terjadi penghematn memori yang cukup signifikan, karena hanya elemen yang tidak kosong yang akan dikelola dan disimpan dalam memori. Konsekuensi yang muncul dari penghematan memori yang terjadi adalah harus dibayar dengan semakin rumitnya algoritma yang harus dikebangkan. Program diimplementasikan dengan bahasa C dan mampu menangani operasi penjumlahan, pengurangan, serta perkalian dua buah *sparse* matriks.

3. Perkalian : Jika A matriks m x n dan B matriks n x p, maka hasil kali A dengan B dinotasikan AB adalah matriks m x p dengan elemen ke ij

(AB)ij = Aij B Ij + Ai2 B2j +.....+ AinBnj = dengan 1<= m, 1<= p

1.2. Aplikasi *Sparse* Matriks

Sparse matriks banyak dijumpai dalam aplikasi sains dan teknik misal, aplikasi pada rekayasa sipil, model persamaan matematika pada tiang jembatan besar, kebanyakan koenfisien pada sembarang persamaan adalah nol. Pemikiran yang sama terjadi pada aplikasi model bangunan dan struktur fisik lainnya.

Pada rangkaian listrik, model dari rangkaian tersebut juga *sparse* semakin besar

rangkainnya maka semakin besar sparsitasnya. Jadi, meskipun prinsipnya dapat dihubungkan dua titik jauh pada rangkaian listrik dengan kabel, dalam kebanyakan rangkaian ini sangat kecil dilakukan.

1.3. Struktur Data

Struktur data yang dikenal dalam pemrograman adalah struktur data dengan tipe:

1. Dasar : integer, real, boolean, dan character, nama informasi yang didefinisikan setiap saat hanya dapat menyimpan satu nilai
2. Bentukan, yaitu tipe yang merupakan komposisi dari tipe dasar, nama informasi yang didefinisikan setiap saat hanya mengandung satu nilai sesuai dengan komposisi dari tipe yang didefinisikan.
3. Tabel (array), yaitu tipe yang mendefinisikan sekumpulan elemen tabel bertipe sama, dan nantinya akses elemen tersebut akan dialokasikan secara kontingu, dengan akses elemen yang ditentukan oleh ondeks. Nama informasi yang didefinisikan setiap saat dapat menyimpan banyak nilai, tergantung pada ukuran tabel.

Ketiga tipe tersebut sesuai dalam penyimpannanya dalam memori, dan tidak ada lagi pilihan untuk menentukan implementasinya.

Dalam penulisan program prosedural adalah menentukan strukturisasi penyimpanan informai (kamus) dan aksi yang memanipulasi struktur tersebut [3]. Jika tempat penyimpan nilai pada nama informasi dianalogikan sebagai sel maka isi dari sel dapat diacu dari letak atau alamat sel tersebut di memori. Karena dalam pemrograman, data yang diproses seringkali sangat komplek, sedangkan struktur dalam penyimpanan komputer sangat sederhana maka dibutuhkan suatu abstraksi dari penyajian data yang kompleks. Adapun struktur data abstrak yang standar di idang informatika adalah :

- List linier
- Multi List
- Stack (tumpukan)
- Queue (Antrian)
- Tree (Pohon)
- Graph (Graf)

1.4. Struktur Data Penyimpanan Sparse Matriks

Misal dipunyai matriks A berorde 10 x 10. Dari seluruh 100 elemen, hanya ada 11 elemen tidak nol. Untuk menyimpan sparse matriks seperti diatas tidak pernah dilakukan dengan menyimpan semua elemenya termasuk elemen nol, karena akan membutuhkan ruang yang besar yang tidak efisien.

Ada beberapa struktur data yang dapat digunakan untuk menyimpan elemen tidak nolnya saja dari sparse matriks, yang pada prinsipnya adalah dengan struktur data kontigu yaitu disimpan secara berurutan atau menggunakan struktur data linked-list[2].

1. Struktur Tabel (Array 2 dimensi)

Deklarasinya

```
Type Matriks = array (1..10,1..10) of real
```

Cara ini adalah cara paling mudah tetapi harus dibayar dengan pengorbanan yang sangat mahal dalam hal penyimpanannya karena semua elemen nolnya juga akan ikut tersimpan. Hal ini akan sangat memboroskan memori, dengan alasan inilah maka penyimpanan *sparse* matriks menggunakan deklasi ini tidak pernah dilakukan.

2. Struktur Array

Dengan menyimpan elemen yang tidak nol saja, menggunakan array satu dimensi (vektor). Dalam hal ini diperlukan 3 buah vektor, yang masing-masing digunakan untuk menyimpan nomor baris, nomor kolom dan nilai elemen yang tidak nol.

Pendeklarasian struktur datanya sebagai berikut:

B_Nilai : (terdefinisi, banyaknya elemen tak nol)

```
Type 1_Matriks = record
                   No Baris : integer
                   No-kolom : integer
                   Nilai : real
                   End of record
Type T_SparMat = arraty [ 1..B_Nilai] of 1_Matriks
```

indeks	Baris	Kolom	nilai
1	1	4	2
2	1	8	3
3	2	1	5
4	4	2	1
5	5	10	7
6	6	3	-2
7	6	4	-5
8	8	9	8
9	8	10	-1

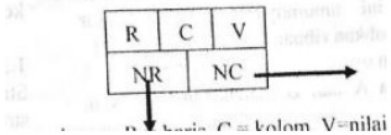
10	9	8	4
11	10	6	-17
12	10	7	9

Dengan memperhatikan gambar tersebut, maka cara penyimpanan *sparse* matriks masih boros, karena jika pada suatu baris mempunyai lebih dari satu elemen tak nol, maka nomor baris tersebut disimpan beberapa kali.

2. Perancangan Struktur Data dan Algoritma

Perancangan Struktur Data

1. Untuk menyimpan sebuah elemen matriks, struktur data yang digunakan digambarkan di bawah ini:

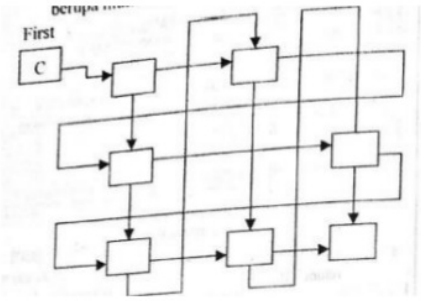


dengan R = baris, C = kolom, V = nilai yang disimpan, NC = pointer ke kolom berikutnya, NR = pointer ke baris berikutnya,

deklarasi programnya :

```
typedef stuct TCell *PCell;
typedef struct TCell {
        PCell NextRow, NextCol;   /* link ke elemen berikutnya */
        int Row, Col;             /* Baris dan Kolom */
        float Value;              /* nilai matriks   */
```

2. Untuk menyimpan sebuah matriks adalah berupa multilist




```
Typedef struct Tmatrix *PMatrix;
Typedef struct Tmatrix {
        Pcell Fist;  /*link ke cell [1.1]*/
        Int  Nrow,Ncol;  /*Dimensi Matrix*/
        Int Crow, Ccol; /* Current Row, Col*/
        Pcell  CC,PC;/*Couple Current/prev Cell*/
        Int Error; /* counter error matriks, 0 berarti oke */
}Tmatrix
```

Keterangan :

Tcell= obyek satu elemen

Tmatrix = obyek matrix, berupa multi lis TCell

First : menunjuk ke Cell

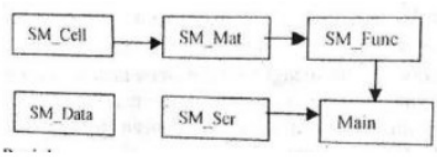
Error : menunjukkan banyaknya kesalahan isi metriks biasanya terjadi jika gagal mengnsert Cell

Ccol, Crow : current Row dan Col berisi [0,0] bila Cureret tidak valid (End)

CC : Current Cell. Menunjuk Cell M[Ccol, Crow] NIL bila Cell M[Ccol, Crow] tidak ada (=0)

PC : Previous Cell. Menyimpan CC terakhir yang tidak NIL, NIL bila CC = Cell M[1,1] (First)

Perancangan mdul program sebagai berikut:



Penjelasan masing-masing modul:

SM_Cell : untuk mndefinisikan sel (elemen) matriks serta alokasi dan dealokasi memori

SM_Mat : untuk mendefinisikan *spase* matriks serta operasi-operasi primitif yang dimilikinya

SM_Func : untuk mendefinisikan fungsi-fungsi penjumlahan, pengurngan dan perkalian matriks

SM_Data: untuk mendefiisikan data global *sparse*

SM_Scr : untuk mendefinisikn ntar muka program *sparse* matriks

SM_Main: program utama *sparse* matriks

Beberapa potongan alogaritma dari awal modul yang sah adalah sebgai berikut:

Modul SM_Cell

```
void CIAllocate(PCell *PC, int I, int J);
/* mengalokasi Cell baru dan menginisiasi baris,kolom
 IS : -        FS : PC = alokasi baru, Row(PCell) = I, Col(PCell) = J
 PC = NIL jika alokasi gagal */
{
 *PC = (PCell) malloc(sizeof(TCell));
 if (PC != NIL) {
  Row(*PC) = I;
  Col(*PC) = J;
  Value(*PC) = 0;
  NextRow(*PC) = NIL;
  NextCol(*PC) = NIL;
 }
}
void CIDeallocate(PCell *PC);
/* mendealokasi Cell
 IS : PC valid,   FS : memory PC didealokasi, PC = NIL */
{
        free(*PC);
        *PC = NIL;
}
```

```
void MtGoRow(PMatrix PM, int I, int J)
{
 MtSearchRow(PM, I, J, &PM->PC, &PM->CC);
 CRow(PM) = I;
 CCol(PM) = J;
}
void MtNextCol(PMatrix PM)
{
        CCol(PM) = CCol(PM) + 1;
        if (CCol(PM) > NCol(PM)) {
                CRow(PM) = CRow(PM) + 1;
                if (CRow(PM) > NRow(PM)) {
                        CRow(PM) = 0;
                        CCol(PM) = 0;
                        PM->CC = NIL;
                        PM->PC = NIL;
                        return;
                }
                CCol(PM) = 1;
        }
        if (PM->CC != NIL)
                PM->PC = PM->CC;
        PM->CC = NextCol(PM->PC);
        if (PM->CC != NIL) {
                if ( (Row(PM->CC) != CRow(PM)) || (Col(PM-
>CC) != CCol(PM))) PM->CC = NIL;
        }
}
```

Modul SM_Mat

```
void MtSearchRow(PMatrix PM, int I, int J, PCell *RPrev, PCell *PC)
/* Mencari Cell PM(I,J) dengan traversal row
 * Jika ketemu : PC = Cell PM(I,J), RPrev = cell baris sebelum PC
 * Jika tidak  : PC = NIL, RPrev = cell baris terakhir sebelum posisi I,J
*/
{
 boolean found;
        *RPrev = NIL;
 *PC = First(PM);
 found = FALSE;
 /* tidak ada kasus kosong */
 do {
  if ((Row(*PC) == I) && (Col(*PC)==J)) {
   found = TRUE;
  }
  else if ((Col(*PC) > J) || ((Col(*PC)==J) && Row(*PC) > I)) {
   *PC = NIL;
  }
  else {
   *RPrev = *PC;
   *PC = NextRow(*PC);
  }
 } while ( (*PC != NIL) && (!found));
}

/*******TRAVERSAL Current Cell **********/
void MtGoFirst(PMatrix PM)
{
 CRow(PM) = 1;
 CCol(PM) = 1;
 PM->CC = First(PM);
 PM->PC = NIL;
}
```

Modul SM_Func

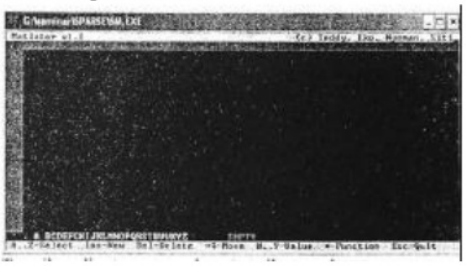
```
PMatrix MtAdd(PMatrix PM1, PMatrix PM2)
{PMatrix PM;

 if ((NRow(PM1)==NRow(PM2)) && (NCol(PM1) == NCol(PM2)))
 {
  MtAllocate(&PM,NRow(PM2),NCol(PM1));
  if( MtIsEmpty(PM)) {
   MtGoFirst(PM);
   MtGoFirst(PM1);
   MtGoFirst(PM2);
   while (! MtIsEnd(PM1)) {
    MtSet(PM,MtGet(PM1)+MtGet(PM2));
    MtNextRow(PM);
    MtNextRow(PM1);
    MtNextRow(PM2);
   }
   return PM;
  }
 }
 return NIL;
}

PMatrix MtMult(PMatrix PM1, PMatrix PM2)
*Menghitung PM1*PM2
*IS : PM1, PM2 valid
*FS : return PM1*PM2
*      return NIL jika :
*      matrix tidak bisa dikalikan (dimensi beda)
*      hasil tidak berhasil dicreate (memory tidak cukup)
*/
{PMatrix PM;
    float SumMt;
    int i, j, k;
    if (NCol(PM1)==NRow(PM2)) {
        MtAllocate(&PM,NRow(PM1),NCol(PM2));
```

```
        if (! MtIsEmpty(PM)) {
                MtGoFirst(PM);
                for (i = 1; i <= NRow(PM); i++) {
                    MtGoFirst(PM2);
                    for (j = 1; j <= NCol(PM); j++) {
                        SumMt = 0.0;
                        MtGoCol(PM1, i, 1);
                        for (k = 1; k <= NRow(PM2); k++)    {
                    SumMt=SumMt+MtGet(PM1)*MtGet(PM2);
            MtNextRow(PM2);
            MtNextCol(PM1);
            MtSet(PM, SumMt);
            MtNextCol(PM);
                                }
                            }
                            return PM;
                    }
            }
            return NIL;
}
```

3. Hasil Aplikasi Program

Berikut diberikan beberapa contoh *snapshot* program yang telah diuji kebenarannya.

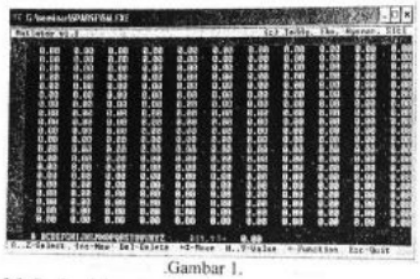
3.1. Tampilan awal



Tampilan di atas merupkan tampilan awal program *motivator v1.1*. berupa fitur *Select* untuk memilih matriks yang akan diakses (A sampai Z). *New* untuk menentukan orde matriks yang baru. *Delete* untuk menghapus matriks, *Move* untuk memilih baris atau kolom yang akan diakses. *Value* untuk mengisi nilai elemen matriks serta *Function* untuk memilih operasi pada matriks. Pada tampilan awal di atas diinisialisasi dengan mengakses matriks A dengan status *empty* artinya listnya masih kosong (nil).

3.2. Tampilan Sparse Matriks

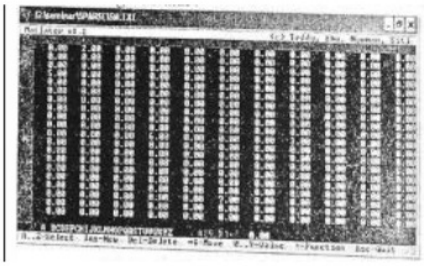
Sewktu orde sudah didefinisikan maka tampilan berupa *sparse* matriks dengan semua elemen diset nol, dan posisi kursor (*bold*) menunjukkan elemen yang sedang di akses. Terlihat pada gambar 1.



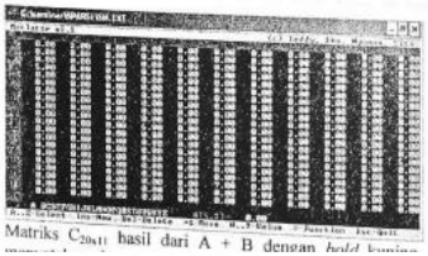
Gambar 1.

3.3. Penjumlahan matriks C= A+B

Matriks A_{20x11} , dengn *bold* kuning menyatkan elemen tidak nol sebgai berikut:



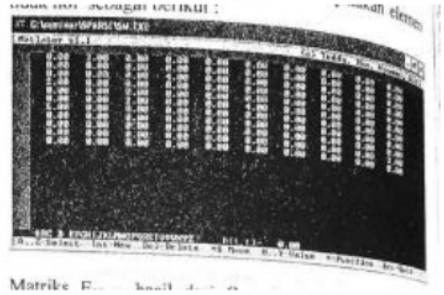
Matriks B_{20x11} dengan *bold* kuning menyatakan elemen tidak nol, sebagai berikut:



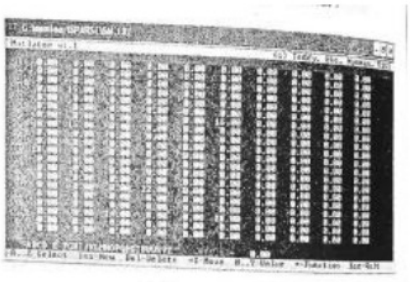
Matriks C_{20x11} hasil A+B dengan *bold* kuning menyatakan elemen tidak nol sebagai berikut:

3.4. Perkalian matriks E= CxD

Matriks D_{11x10} dengan *bold* Kuning menyatakan ekemen tidak nol sebagai berikut:



Matriks E_{20x10} hasil dari C x D dengn *bold* kuning menyatkan elemen tidak nol sebagai berikut:



4. Kesimpulan

Sturktur data yang paling ccocok untuk mengelola *sparse* matriks adalah dengan multilist. Dalam penelitian ini telah berhasil dikembangkan program operasi dasar penambahan, pengurangan, serta perkalian pada *sparse* matriks dengan menggunakan struktur data multilist representasi pointer. Efisiensi memori yang diperoleh harus dibayar mahal dengan sulitnya algoritma yang harus dikembangkan

Referensi

[1] Doerr, Alan. *Applied Discrete Structures for Computer Science,* Sciennce Reasearch Associates, 1985

[2] Horowitz, Eilis, *Fundamentals of data Structures, Galgotia,* 1999

[3] Inggriani Liem, Diktat Kuliah Struktur Data dan Algoritma Pemrograman, Jurusan Teknik Informatika, ITB, 1996

[4] Kernighan, Brian W, *The C Programing Language,* Hall, 1998

Struktur data MultiList

ORIGINALITY REPORT

7%	7%	0%	0%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	skripsigratislengkap99.blogspot.com Internet Source	2%
2	www.coursehero.com Internet Source	2%
3	www.cermat.upm.edu.my Internet Source	1%
4	fitrimarwaningsih.wordpress.com Internet Source	1%
5	pt.scribd.com Internet Source	1%
6	adegunaone.blogspot.com Internet Source	1%

Exclude quotes	Off	Exclude matches	Off
Exclude bibliography	Off		