UNIVERSIDADE DE SÃO PAULO
ESCOLA DE ENGENHARIA DE SÃO CARLOS
CENTRO DE PROCESSAMENTO DE DADOS

ESTUTURAS DINÂMICAS NA IMPLEMENTAÇÃO DE MATRIZES ESPARSAS

Edson Walmir Cazarini

EESC-USP
Biblioteca
CAMPUS DE S. CARLOS

---

outubro/88

ÍNDICE

1. Introdução 03

2. O Conceito de variável 04

3. Representação de matrizes esparsas utilizando
estruturas array 07

4. Representação de matrizes esparsas utilizando
estruturas dinâmicas 09

Tipo e aperações abstratas 12

Listagens das operações abstratas em Modula-2 14

ElementoME 14
NumeroElementosME 14
ZeraElementosME 14
ExisteElementoME 15
ColocaElementoME 15
EliminaElementoME 15
TranspostaME 16
SomaME 16
SubtracaoME 17
ProdutoPosEscalarME 17
ProdutoME 18
ProdutoEscalarVE 18
CopiaME 19
CopiaNegativaME 19

Listagens das operações auxiliares utilizadas 20

MontaLinha 20
MontaColuna 20
MontaCelula 20

5. Bibliografia 21

INTRODUÇÃO

A utilização de matrizes esparsas é muito comum nas aplicações em métodos numéricos. Muitas implementações são feitas com base no uso de matrizes na sua forma normal sem levar em conta a esparsidade da matriz, trazendo prejuizos com relação a memória utilizada e operações efetuadas, uma vez que as operações que envolvem o zero, poderiam ser dispensadas.

A forma convensional de se representar esse tipo de matriz, é através da utilização de estruturas tipo arrays equivalentes (convertendo matrizes bi-dimensionais em uni-dimensionais) com dimensionamento dinâmico.

As matrizes esparsas se caracterizam por conter um pequeno número de elementos diferentes de zero. As operações necessárias são geralmente processadas em todos os elementos da matriz. Quando um elemento for igual a zero as operações podem ser poupadas tanto as de soma e subtração, onde se repete o número que esta sendo operado, ou no produto, onde o resultado será nulo. Consideramos tambem, que o resultado de uma operação matricial (soma, subtração e produto) com matrizes esparsas tambem será uma matriz esparsa.

Variáveis dinâmicas não são declaradas e nem acessadas através de identificadores. As linguagens que suportam esse tipo de variável, em geral, colocam a disposição do usuário toda memória excedente do computador, que será gerenciada pelo programador, através de seus endereços. Uma estrutura possivel para o uso desse tipo de variável é a lista encadeada, onde se coloca e se retira elementos conforme a conveniência do programa. As variáveis liberadas pelo programa são devolvidas ao controle do processador e poderão ser re-utilizadas em qualquer outro momento pela mesma ou por outra lista gerenciada pelo programa.

O CONCEITO DE VARIÁVEL

Ao conceito de variável estão associados, as seguintes características:

2.1- **Nome da Variável** : É o rótulo que permite o acesso à memória pelo programa. O nome de uma variável é, em geral, um identificador sintaticamente válido.

2.2- **Atributos** : Os principais atributos associados a uma variável são :

2.2.1- **O Tipo** : Classe de objetos de dados adicionadas a todas as operações associadas aos componentes dessa classe que possibilitem sua criação e manipulação

2.2.2- **O escopo** : Trecho do programa em que a variável é conhecida e manipulável. Ela é visivel dentro do escopo e invisível fora dele.

2.2.3- **O Tempo de Vida** : é o intervalo de tempo durante a qual uma área de memória fica alocada à variável. Essa alocação pode ser ESTÁTICA ou DINÂMICA. Será estática se a alocação for realizada em tempo de compilação, portanto antes da execução do programa e será dinâmica, se for feito em tempo de execução do programa.

2.3- **Posição de memória** : É reservado a uma variável, uma posição na memória correspondente ao número de bytes necessários a receber os valores armazenados. Essa área depende do tipo de variável ou da quantidade declarada pelo programador que fica pronto para receber seus valores.

2.4- **O Valor de uma variável** : é a interpretação do conteúdo gravado na área alocada. A interpretação desse conteúdo é necessária porque depende do tipo de variável. O tipo de uma variável é caracterizado pela especificação de uma classe de valores que poderão ser associados, juntamente com seus operadores, estática ou dinâmicamente. Na maioria das linguagens (FORTRAN, COBOL, Pascal, Ada) essa associação é estática, e através de um comando de declaração. Dependendo do tipo, o conteúdo corresponde a um código do valor representado. Normalmente, a asso-

ciação entre **o valor e sua posição na memória é dinâmica** uma vez que alterações no conteúdo gravado é feito em tempo de execução, através de um **comando de atribuição.** A associação estática, em tempo de compilação, caracteriza uma constante que terá seu valor congelado durante toda a execução do programa.

Variáveis Estáticas : As variáveis estáticas, são definidas no programa através de comandos de declaração. Na declaração são fixados o seu nome, tipo, escopo, conteúdo inicial (em algumas linguagens), e área de memória alocada. O tempo de vida, em geral, será o tempo de execução do programa. Determinadas linguagens, fazem uso compartilhado de memória em função de seu escopo.

Variáveis Dinâmicas : As variáveis dinâmicas, não são declaradas. São anônimas, recebendo apenas a declaração de tipos. Em geral, toda a memória excedente é considerada como disponível e podera ser solicitada através de comando próprio, característico de cada linguagem (no Pascal, e em Modula-2, será o procedimento NEW). O sistema informa o endereço da área em que a variável deve ser colocada, associando a área necessária no momento da solicitação. A comunicação entre o programa e a área alocada. será feita através de seu endereço que poderá ser armazenado em variáveis do tipo POINTER.

O **Tempo de vida** de uma **variável dinâmica** será administrado pelo programador que solicitará a alocação da área quando necessário e devolverá ao sistema quando seu armazenamento não se fizer mais necessário. A devolução ao sistema será feito através do comando apropriado (no Pascal e em Modula-2, será o procedimento DISPOSE), ficando livre para outras solicitações.

Estruturas dinâmicas : São estruturas de dados em que os elementos de dados são criado e cancelados dinamicamente, ou seja, ela aumenta ou diminue de tamanho durante a execução do programa. A declaração dessas estruturas são feitas através de **estruturas de dados recursivas.**

Estruturas Recursivas : Entende-se por estruturas recursivas, àquelas em que na sua definição são utilizados componentes que são definidos na própria estrutura que está sendo definida.

Exemplo :

```
TYPE
     Ponteiro = POINTER TO Elemento;
     Elemento = RECORD
        dados : INTEGER;
        valor : REAL;
      Proximo : Ponteiro;
                END;
```

- O tipo *Ponteiro* é definido como POINTER de um tipo **Elemento** que ainda é desconhecido.

- O tipo **Elemento** é definido como um RECORD, composto dos ítens : **Dados, Valor e Proximo. Dados** e **Valor**, são de tipos conhecidos, (respectivamente **INTEGER** e **REAL**). O tipo declarado para **Proximo** é *Ponteiro*, definido anteriormente em função do tipo **Elemento**.

Esta estrutura cresce em função dos seus elementos, dinamicamente, durante a execução do programa.

As aplicações mais correntes, são estruturas tipo **listas encadeadas** e **árvores binárias** cujo encadeamento é feito através de ponteiros declarados em cada célula de dados.

REPRESENTAÇÃO DE MATRIZES ESPARSAS UTILIZANDO ESTRUTURAS ARRAY

Cada elemento da matriz será representado por um registro que conterá o valor (diferente de zero) do elemento propriamente dito, e a sua identificação correspondente à sua posição [i,j] na matriz. Usa-se o cálculo de equivalência do elemento [i,j] através da expressão :

$$k = (j - 1) * N + i ,$$

onde: i = número da linha do elemento,
j = número da coluna,
N = número de linhas da Matriz

O elemento [i,j] da matriz bidimensional, será representado por [k] no conjunto unidimensional e identificará o elemento [i,j] da matriz esparsa quando este for diferente de zero. Os elementos iguais a zero não serão armazenados.

A representação de uma matriz esparsa, utilizando a estrutura ***array*** :

```
TYPE

    ElementoMatriz = RECORD
                       k : CARDINAL;
                   Valor : REAL;
                           END;

     MatrizEsparsa = ARRAY [1..1000] OF ElementoMatriz;
```

k = o identificador do elemento
Valor = valor do elemento, diferente de zero.

O tipo ***MatrizEsparsa***, será um agregado de dados composto de elementos do tipo **RECORD**, contendo as informações necessárias a sua localização.

A introdução de um elemento, utilizando esta nomenclatura será feita, achando-se o local onde será colocado, abre-se um espaço transladando todos os elementos seguintes de uma casa. Os elementos são colocados em ordem crescente para facilitar sua localização. A busca de um elemento provocará um procedimento inverso, o elemento é localizado através de um algorítmo que leva em conta a ordem dos elementos. Se for eliminado da matriz, todos os elementos à direira sofrerão uma translação para a esquerda.

REPRESENTAÇÃO DE MATRIZES ESPARSAS UTILIZANDO ESTRUTURAS DINÂMICAS

O objetivo está em representar a matriz através de uma estrutura de lista encadeada, onde seus elementos estão colocados em células (o elemento juntamente com sua identificação) e uma célula será ligada à célula seguinte por meio de um ponteiro.

A introdução de uma célula será feita, obtendo-se o local onde o elemento deverá ser colocado, respeitando-se a ordem dos identificadores, e a ligação será feita através de acerto nos ponteiros das células envolvidas, não sendo necessário movimentar nenhum elemento. A eliminação de uma célula será feita da mesma forma.

A definição desse tipo em Modula-2 será :

```
TYPE
   MatrizEsparsa = POINTER TO CelulaMatrizEsparsa;
   CelulaMatrizEsparsa =
            RECORD
                  k : CARDINAL; (ou INTEGER em Pascal)
              Valor : REAL;
            Proximo : MatrizEsparsa;
            END;
```

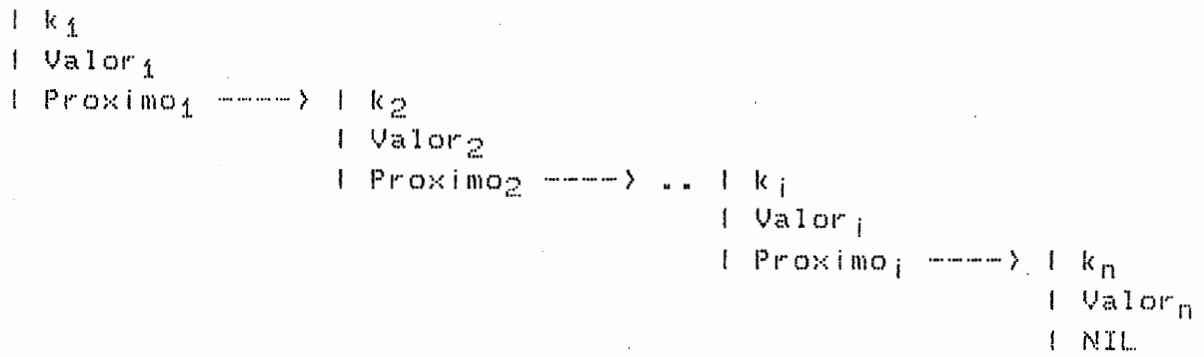
onde :

k : corresponde ao identificador do elemento representando a posição do elemento em um vetor unidimensional equivalente;
Valor : conterá o número real diferente de zero;
Proximo : conterá o endereço da próxima célula.

Será um **Tipo Abstrato de Dado**, se associado ao conjunto de operações que o envolve, como a colocação de um elemento na matriz assim representada, a busca de um determinado elemento, a eliminação de um elemento ou de uma matriz toda com a devolução da área ocupada ao sistema. Essa caracterização trará vantagens à legibilidade do programa e consequente ganho de qualidade e produtividade na programação.

Estas células serão organizadas em forma de lista encadeada, onde a ligação das células de dados, será feito através do elemento **Proximo**. O valor de **Proximo** será o endereço da próxima célula composta da identificação e o valor do elemento. O último elemento de uma matriz (ou da lista) conterá o valor de **Proximo** igual a NIL (constante que define o elemento vazio)

Uma matriz esparsa pode ser vista da seguinte maneira :



Cada vez que se necessita de um elemento, este é localizado dentro da lista, através de seu identificador ij. Para otimizar a busca de um elemento, o encadeamento é feito de forma ordenada (ordem crescente), diminuindo sensivelmente o tempo de busca.

Para operacionalizar esse processamento, será necessário dispor de procedimentos que irão manipular esses elementos, formando um tipo abstrato de dado :

ColocaElementoME : Procedimento que tem por objetivo, incluir um elemento em uma matriz esparsa.

ElementoME : é uma **função do tipo REAL** com a finalidade de localizar um elemento de uma matriz esparsa colocada em uma lista encadeada. A resposta desse procedimento será seu **Valor** caso seja encontrado, e **zero** caso não seja encontrado na lista.

ExisteElementoME : é uma **função do tipo BOOLEAN**, que informa ao programa se uma determinada matriz está vazia (No ou False) ou se contém pelo menos um elemento (Yes ou True).

NumeroElementosME: é uma **função do tipo CARDINAL**, destinada a contar o número de elementos da matriz esparsa.

ZeraElementosME : Procedimento que libera todos os elementos da Matriz esparsa, devolvendo a área de dados ao controle do sistema.

SomaME : Procedimento para somar duas matrizes esparsas, obtendo como resultado uma matriz esparsa.

ProdutoME : Procedimento para produto de duas matrizes esparsas obtendo como resultado uma matriz esparsa.

SubtracaoME : Procedimento para subtração de duas matrizes esparsas, obtendo como resultado uma matriz esparsa.

ProdutoEscalarVE : é uma **função do tipo REAL**, para o produto escalar entre dois vetores esparsos.

EliminaElementoME: Procedimento destinado a eliminar um elemento de uma matriz esparsa.

TranspostaME : Procedimento destinado determinar a transposta de uma matriz esparsa.

ProdutoPorEscalarME: Procedimento destinado a Multiplicação de uma matriz esparsa por um escalar.

CopiaME : Procedimento destinado a copiar uma matriz esparsa, para outro endereço.

CopiaNegativaME : Procedimento destinado a copiar uma matriz esparsa, para outro endereço, trocando o sinal seus elementos.

MatrizEsparsa : Tipo abstrato que define um elemento da matriz esparsa, através do tipo estruturado RECORD.

Definição do tipo e operações abstratas, em Modula-2 :

```
DEFINITION MODULE MetNum02;

  EXPORT  QUALIFIED MatrizEsparsa, ColocaElementoME, TranspostaME,
     EliminaElementoME,   ZeraElementosME,  SomaME,   SubtracaoME,
     ProdutoME,  NumeroElementosME,  ProdutoPorEscalarME, CopiaME,
     ExisteElementoME, ElementoME, CopiaNegativaME,
     ProdutoEscalarVE;


  TYPE
    MatrizEsparsa = POINTER TO CelulaMatrizEsparsa;
    CelulaMatrizEsparsa =
      RECORD
      ij : CARDINAL;
      Valor : REAL;
      Proximo : MatrizEsparsa;
      END;


  PROCEDURE ColocaElementoME(ij : CARDINAL; Valor : REAL; VAR
        Matriz : MatrizEsparsa);


  PROCEDURE ProdutoME(NumeroLinhasA, NumeroColunasA,
     NumeroColunasB  : CARDINAL; MatrizA, MatrizB : MatrizEsparsa;
     VAR MatrizC : MatrizEsparsa);


  PROCEDURE  SomaME(MatrizA, MatrizB : MatrizEsparsa; VAR  MatrizC
     : MatrizEsparsa);


  PROCEDURE SubtracaoME(MatrizA, MatrizB : MatrizEsparsa; VAR
     MatrizC : MatrizEsparsa);


  PROCEDURE ProdutoPorEscalarME(Escalar : REAL; VAR Matriz :
     MatrizEsparsa);


  PROCEDURE ProdutoEscalarVE(VetorA, VetorB : MatrizEsparsa) :
     REAL;


  PROCEDURE ZeraElementosME(VAR Matriz : MatrizEsparsa);


  PROCEDURE ExisteElementoME(Matriz : MatrizEsparsa) : BOOLEAN;
```

```
PROCEDURE NumeroElementosME(Matriz : MatrizEsparsa) : CARDINAL;

PROCEDURE EliminaElementoME(ij : CARDINAL; VAR Matriz :
   MatrizEsparsa);

PROCEDURE ElementoME(ij : CARDINAL; Matriz : MatrizEsparsa) :
   REAL;

PROCEDURE CopiaME(MatrizA : MatrizEsparsa; VAR MatrizB :
   MatrizEsparsa);

PROCEDURE CopiaNegativaME(MatrizA : MatrizEsparsa; VAR MatrizB:
   MatrizEsparsa);

PROCEDURE TranspostaME(NumeroLinhasA, NumeroColunasA :
   CARDINAL; MatrizA : MatrizEsparsa; VAR MatrizB :
   MatrizEsparsa);

END MetNum02.
```

Listagens das operações abstratas em Modula-2 :

```
IMPLEMENTATION MODULE MetNum02;
  FROM MetNum01 IMPORT IndKC;
  FROM Storage IMPORT ALLOCATE, DEALLOCATE, Available;

-------------------------------------------------------------------------------

  PROCEDURE ElementoME(ij: CARDINAL; Matriz: MatrizEsparsa):
      REAL;
    BEGIN
      IF Matriz <> NIL THEN
        IF Matriz^.ij = ij THEN
          RETURN Matriz^.Valor;
        ELSE
          IF Matriz^.ij < ij THEN
            RETURN ElementoME(ij,Matriz^.Proximo);
          ELSE
            RETURN 0.;
          END;
        END;
      ELSE
        RETURN 0.;
      END;
    END ElementoME;

-------------------------------------------------------------------------------

  PROCEDURE NumeroElementosME(Matriz : MatrizEsparsa) : CARDINAL;
    BEGIN
      IF Matriz <> NIL THEN
        RETURN 1 + NumeroElementosME(Matriz^.Proximo);
      ELSE
        RETURN 0;
      END;
    END NumeroElementosME;

-------------------------------------------------------------------------------

  PROCEDURE ZeraElementosME(VAR Matriz : MatrizEsparsa);
    BEGIN
      IF Matriz <> NIL THEN
        IF Matriz^.Proximo = NIL THEN
          DISPOSE(Matriz);
        ELSE
          ZeraElementosME(Matriz^.Proximo);
        END;
        Matriz := NIL;
      END;
    END ZeraElementosME;
```

```
PROCEDURE ExisteElementoME(Matriz : MatrizEsparsa) : BOOLEAN;
  BEGIN
    IF Matriz = NIL THEN
      RETURN FALSE;
    ELSE
      RETURN TRUE;
    END;
  END ExisteElementoME;
```

```
PROCEDURE ColocaElementoME(ij : CARDINAL; Valor : REAL; VAR
    Matriz : MatrizEsparsa);
  VAR
    Posicao : MatrizEsparsa;
  BEGIN
    IF Matriz = NIL THEN
      MontaCelula(ij, Valor, Matriz);
    ELSE
      IF Matriz^.ij = ij THEN
        Matriz^.Valor := Valor;
      ELSE
        IF Matriz^.ij > ij THEN
          MontaCelula(ij, Valor, Posicao);
          IF Posicao <> NIL THEN
            Posicao^.Proximo := Matriz;
            Matriz := Posicao;
          END;
        ELSE
          ColocaElementoME(ij, Valor, Matriz^.Proximo);
        END;
      END;
    END;
  END ColocaElementoME;
```

```
PROCEDURE EliminaElementoME(ij : CARDINAL ; VAR Matriz :
    MatrizEsparsa);
  VAR
    Auxiliar : MatrizEsparsa;
  BEGIN
    IF Matriz <> NIL THEN
      IF Matriz^.ij = ij THEN
        Auxiliar := Matriz;
        Matriz := Matriz^.Proximo;
        DISPOSE(Auxiliar);
      ELSE
        IF Matriz^.ij < ij THEN
          EliminaElementoME(ij, Matriz^.Proximo);
        END;
      END;
    END;
  END EliminaElementoME;
```

```
PROCEDURE TranspostaME(NumeroLinhasA, NumeroColunasA :
    CARDINAL;
    MatrizA : MatrizEsparsa; VAR MatrizB : MatrizEsparsa);
  VAR
    ij, i, j : CARDINAL;
  BEGIN
    ZeraElementosME(MatrizB);
    FOR j := 1 TO NumeroColunasA DO
      FOR i := 1 TO NumeroLinhasA DO
        ij := IndKC(i,j,NumeroLinhasA);
        ColocaElementoME(IndKC(j,i,NumeroColunasA),ElementoME
                         (ij, MatrizA), MatrizB);
      END;
    END;
  END TranspostaME;
```

```
PROCEDURE  SomaME(MatrizA, MatrizB : MatrizEsparsa; VAR MatrizC:
    MatrizEsparsa);
  VAR
    AuxiliarC : MatrizEsparsa;
  BEGIN
    ZeraElementosME(MatrizC);
    IF MatrizA <> NIL THEN
      IF MatrizB <> NIL THEN
        IF MatrizA^.ij = MatrizB^.ij THEN
          MontaCelula(MatrizA^.ij,  ( MatrizA^.Valor + MatrizB^.
                      Valor ), AuxiliarC);
          IF AuxiliarC <> NIL THEN
            MatrizC := AuxiliarC;
            SomaME(MatrizA^.Proximo, MatrizB^.Proximo,
                   MatrizC^.Proximo);
          ELSE
            SomaME(MatrizA^.Proximo, MatrizB^.Proximo,
                   MatrizC);
          END;
        ELSE
          IF MatrizA^.ij < MatrizB^.ij THEN
            MontaCelula(MatrizA^.ij, MatrizA^.Valor, MatrizC);
            SomaME(MatrizA^.Proximo,MatrizB, MatrizC^.Proximo);
          ELSE
            MontaCelula(MatrizB^.ij, MatrizB^.Valor, MatrizC);
            SomaME(MatrizA,MatrizB^.Proximo, MatrizC^.Proximo);
          END;
        END;
      ELSE
        CopiaME(MatrizA, MatrizC);
      END;
    ELSE
      CopiaME(MatrizB, MatrizC);
    END;
  END SomaME;
```

```
PROCEDURE SubtracaoME(MatrizA, MatrizB : MatrizEsparsa; VAR
    MatrizC : MatrizEsparsa);
  VAR
    AuxiliarC : MatrizEsparsa;
  BEGIN
    ZeraElementosME(MatrizC);
    IF MatrizA <> NIL THEN
      IF MatrizB <> NIL THEN
        IF MatrizA^.ij = MatrizB^.ij THEN
          MontaCelula(MatrizA^.ij,  ( MatrizA^.Valor - MatrizB^.
                      Valor ), AuxiliarC);
          IF AuxiliarC <> NIL THEN
            MatrizC := AuxiliarC;
            SubtracaoME(MatrizA^.Proximo, MatrizB^.Proximo,
                        MatrizC^.Proximo);
          ELSE
            SubtracaoME(MatrizA^.Proximo, MatrizB^.Proximo,
                        MatrizC);
          END;
        ELSE
          IF MatrizA^.ij < MatrizB^.ij THEN
            MontaCelula(MatrizA^.ij, MatrizA^.Valor, MatrizC);
            SubtracaoME(MatrizA^.Proximo,  MatrizB, MatrizC^.
                        Proximo);
          ELSE
            MontaCelula(MatrizB^.ij, ( -MatrizB^.Valor ),
                        MatrizC);
            SubtracaoME(MatrizA,  MatrizB^.Proximo, MatrizC^.
                        Proximo);
          END;
        END;
      ELSE
        CopiaME(MatrizA, MatrizC);
      END;
    ELSE
      CopiaNegativaME(MatrizB, MatrizC);
    END;
  END SubtracaoME;
```

```
PROCEDURE ProdutoPorEscalarME(Escalar : REAL; VAR Matriz :
    MatrizEsparsa);
  BEGIN
    IF Matriz <> NIL THEN
      IF Escalar <> 1. THEN
        IF ABS(Escalar) > 1.e-11 THEN
          Matriz^.Valor := Escalar * Matriz^.Valor;
          ProdutoPorEscalarME(Escalar, Matriz^.Proximo);
        ELSE
          ZeraElementosME(Matriz);
        END;
      END;
    END;
  END ProdutoPorEscalarME;
```

```
PROCEDURE  ProdutoME(NumeroLinhasA,NumeroColunasA,NumeroColunasB
    : CARDINAL; MatrizA, MatrizB : MatrizEsparsa; VAR MatrizC :
    MatrizEsparsa);
  VAR
    i, j : CARDINAL;
    LinhaA, ColunaB : MatrizEsparsa;
  BEGIN
    LinhaA := NIL;
    ColunaB := NIL;
    ZeraElementosME(MatrizC);
    FOR i := 1 TO NumeroLinhasA DO
      MontaLinha(i, NumeroLinhasA, NumeroColunasA, MatrizA,
                 LinhaA);
      IF LinhaA <> NIL THEN
        FOR j := 1 TO NumeroColunasB DO
          MontaColuna(j, NumeroColunasA, NumeroColunasB,
                      MatrizB, ColunaB);
          IF ColunaB <> NIL THEN
            ColocaElementoME( IndKC( i, j, NumeroLinhasA ),
                  ProdutoEscalarVE(LinhaA,ColunaB), MatrizC);
          END;
        END;
      END;
    END;
  END ProdutoME;
```

```
PROCEDURE ProdutoEscalarVE(VetorA, VetorB : MatrizEsparsa) :
    REAL;
  BEGIN
    IF (VetorA <> NIL) AND (VetorB <> NIL) THEN
      IF VetorA^.ij = VetorB^.ij THEN
        RETURN VetorA^.Valor * VetorB^.Valor + ProdutoEscalarVE
                  (VetorA^.Proximo, VetorB^.Proximo);
      ELSE
        IF VetorA^.ij < VetorB^.ij THEN
          RETURN ProdutoEscalarVE(VetorA^.Proximo,VetorB);
        ELSE
          RETURN ProdutoEscalarVE(VetorA,VetorB^.Proximo);
        END;
      END;
    ELSE
      RETURN 0.;
    END;
  END ProdutoEscalarVE;
```

```
PROCEDURE CopiaME(MatrizA : MatrizEsparsa; VAR MatrizB :
    MatrizEsparsa);
  BEGIN
    IF MatrizA <> NIL THEN
      MontaCelula(MatrizA^.ij, MatrizA^.Valor, MatrizB);
      CopiaME(MatrizA^.Proximo, MatrizB^.Proximo);
    ELSE
      MatrizB := NIL;
    END;
  END CopiaME;
```

```
PROCEDURE CopiaNegativaME(MatrizA: MatrizEsparsa; VAR MatrizB :
    MatrizEsparsa);
  BEGIN
    IF MatrizA <> NIL THEN
      MontaCelula(MatrizA^.ij, ( -MatrizA^.Valor ), MatrizB);
      CopiaNegativaME(MatrizA^.Proximo, MatrizB^.Proximo);
    ELSE
      MatrizB := NIL;
    END;
  END CopiaNegativaME;
```

Listagens das operações auxiliares em Modula-2 :

```
PROCEDURE MontaLinha(Linha, NumeroLinhas, NumeroColunas :
 CARDINAL;Matriz: MatrizEsparsa;VAR VetorLinha: MatrizEsparsa);
  VAR
    ij, j : CARDINAL;
  BEGIN
    ZeraElementosME(VetorLinha);
    IF Matriz <> NIL THEN
      FOR j := 1 TO NumeroColunas DO
        ij := IndKC(Linha,j,NumeroLinhas);
        ColocaElementoME(j, ElementoME(ij,Matriz), VetorLinha);
      END;
    END;
  END MontaLinha;
```

```
PROCEDURE MontaColuna(Coluna, NumeroLinhas, NumeroColunas :
  CARDINAL;Matriz:MatrizEsparsa;VAR VetorColuna:MatrizEsparsa);
  VAR
    i, ij : CARDINAL;
  BEGIN
    ZeraElementosME(VetorColuna);
    IF Matriz <> NIL THEN
      FOR i := 1 TO NumeroLinhas DO
        ij := IndKC(i,Coluna,NumeroLinhas);
        ColocaElementoME(i, ElementoME(ij,Matriz),VetorColuna);
      END;
    END;
  END MontaColuna;
```

```
PROCEDURE MontaCelula(ij : CARDINAL; Valor: REAL; VAR Endereco:
   MatrizEsparsa);
  BEGIN
    IF ABS(Valor) > 1.e-11 THEN
      NEW(Endereco);
      Endereco^.ij := ij;
      Endereco^.Valor := Valor;
      Endereco^.Proximo := NIL;
    ELSE
      Endereco := NIL;
    END;
  END MontaCelula;


BEGIN
END MetNum02.
```

BIBLIOGRAFIA

HOROWITZ, E & SHAMI, S - Fundamentals of Data Structures., Computer Science Press, c1976.

GHEZZI,Carlo & JAZAYERI,Mehdi - Programming Language Concepts., John Wiley & Sons, New York, c1982.

SILVA,José Carlos & ASSIS,Fidelis Sigmaringa - Linguagens de Programação. Conceitos e Avaliação., McGraw-Hill, São Paulo, c1988.

LOGITECH MODULA-2/86 Software Development System. User's Manual., LOGITECH,Inc, 1986

CAZARINI, Edson Walmir - Linguagem de Programação Modula-2., CPD-EESC-USP, São Carlos, c1988