Notas de Aula - AED1 – Matrizes Esparsas (parte 2)
Prof. Jefferson T. Oliva

Na aula anterior vimos definições de matriz esparsa, que é uma alternativa em relação à matriz "convencional" (e.g. int mat[5][5]).

Por mais que uma matriz "convencional" tenha a vantagem de simplicidade, a mesma pode acarretar em desperdício de recursos computacionais, pois é necessário um espaço contíguo de n (linhas) x m (colunas) vezes o tamanho do tipo de dado. Em uma imagem em preto e branco, por exemplo, o interesse pode ser a representação de apenas de contornos de objetos. Neste caso, matrizes esparsas podem possibilitar a economia de recursos computacionais por apenas representar o conteúdo "relevante" de uma matriz.

Matrizes esparsas possuem diversas aplicações. Um dos exemplos apresentados na aula anterior foi a planilha eletrônica, na qual a maior parte do espaço e é referente às células preenchidas.

Conversão de Matriz Numérica para Esparsa

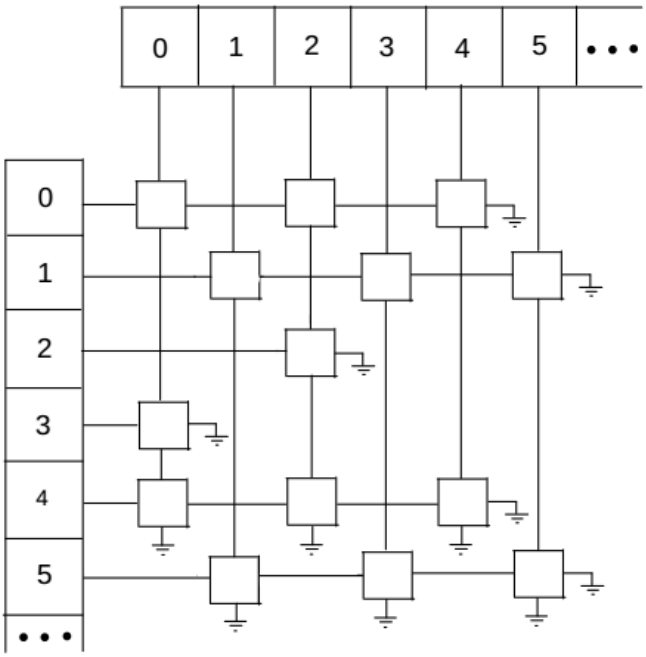
Implementação de uma função para inserção de elementos no início de uma lista encadeada.

```
static void inserir_primeiro(ListaE *l, int col, int item){
        Cell *nova = criar_celula(item);
        nova->col = col;
        nova->next = l->head;
        l->head = nova;
}
```

Para cada linha da matriz numérica, percorrer do último elemento ao primeiro, utilizando a função sugerida no item anterior.

```
Spa_Mat* converter(int **mat, int l, int c){
        int i, j;
        Spa_Mat* me = criar(l, c);
        for (i = 0; i < l; i++)
                for (j = c - 1; j >= 0; j--)
                        if (mat[i][j] > 0)
                                inserir_primeiro(me->lin[i], c, mat[i][j]);
        return me;
}
```

Outro Exemplo de TAD para Matrizes Esparsas



No exemplo de matriz acima, podemos determinar que cada elemento (Cell) possui novos campos em comparação com o que foi visto na aula anterior. No exemplo apresentado a seguir, cada célula possui também um número inteiro para representar a linha em que o item está localizado na matriz. Outro novo campo na estrutura da célula é uma referência para a célula da próxima linha (desde que esteja localizada na mesma coluna):

```
typedef struct Cell Cell;

struct Cell{
      int item, lin, col;
      Cell *next;
      Cell *down;
};
```

A lista encadeada permanece a mesma em relação à aula anterior.

Por fim, na estrutura da matriz esparsa é adicionado um outro vetor de listas encadeadas, sendo esta para representar as colunas da matriz:

```
typedef struct Spa_Mat Spa_Mat;

struct Spa_Mat{
      int n_lin;
      int n_col;
      ListaE **lin;
      ListaE **col;
};
```

Arquivo .h

```
typedef struct Cell Cell;

typedef struct ListaE ListaE;

typedef struct Spa_Mat Spa_Mat;

Spa_Mat* criar(int l, int c);

Cell* criar_celula(int item, int l, int c);

int buscar(int item, Spa_Mat* mat);

void alterar(int item, int l, int c, Spa_Mat* mat);
```

Arquivo .c

```
struct Cell{
	int item, lin, col;
	Cell *next;
	Cell *down;
};


struct ListaE{
	Cell *head;
};


struct Spa_Mat{
	int n_lin;
	int n_col;
	ListaE **lin;
	ListaE **col;
};


Spa_Mat* criar(int l, int c){
	Spa_Mat* mat = (Spa_Mat*) malloc(sizeof(Spa_Mat));
	int i;

	mat->n_col = c;
	mat->n_lin = l;
	mat->lin = (ListaE**) malloc(sizeof(ListaE*) * l);
	mat->col = (ListaE**) malloc(sizeof(ListaE*) * c);

	for (i = 0; i < l; i++){
		mat->lin[i] = (ListaE*) malloc(sizeof(ListaE));
		mat->lin[i]->head = NULL;
```

```
        }

        for (i = 0; i < c; i++){
                mat->col[i] = (ListaE*) malloc(sizeof(ListaE));
                mat->col[i]->head = NULL;
        }

        return mat;
}


Cell* criar_celula(int item, int l, int c){
   Cell *novo = NULL;

   if (item > 0){
      novo = (Cell*) malloc(sizeof(Cell));
      novo->item = item;
      novo->lin = l;
      novo->col = c;
      novo->next = NULL;
      novo->down = NULL;
   }

   return novo;
}


static int validar_ME(int l, int c, Spa_Mat* mat){
   return (Spa_Mat != NULL) && (l >= 0) && (l < mat->lin) && (c >= 0) && (l < mat->col);
}


int buscar(int l, int c, Spa_Mat* mat){
        int i;
        int aux = 0;
        Cell *aux;

        if (validar_ME(l, c, mat)){
           aux = mat->lin[l]->head;

           while ((aux != NULL) && (aux->col < c))
              aux = aux->next;

           if ((aux != NULL) && (aux->col == c))
              return aux->item;
        }

        return 0;
}
```

```
// Obtém a última célula antes de l
static Cell* obter_celula_antes_linha(int l, ListaE* col){
   Cell *auxA  = NULL;
   Cell *auxP = col->head;

   while ((auxP != NULL) && (auxP->lin < l)){
      auxA = auxP;
      auxP = auxP->down;
   }

   return auxA;
}


void alterar(unsigned int item, int l, int c, Spa_Mat* mat){
   Cell *auxLA, *auxLP, *auxCA, *novo;

   if (validar_pos_matriz(l, c, mat)){
      auxLA = NULL;
      auxLP = mat->lin[l]->head;

      while ((auxLP != NULL) && (auxLP->col < c)){
         auxLA = auxLP;
         auxLP = auxLP->next;
      }

      // Inserção ou alteração de elemento na posição (l, c)
      if (item > 0){
         // Significa que existe um elemento na posição (l, c).
         // Nesse caso, basta alterar o valor da célula.
         if ((auxLP != NULL) && (auxLP->col == c))
            auxLP->item = item
         else{ // Caso contrário, uma nova célula deve ser alocada
            novo = criar_celula(item, l, c);
            auxCA = obter_celula_antes_linha(l, mat->col[c]);

            // auxLA é nulo apenas quando mat->lin[l]->head for nulo ou
            // l é menor que mat->lin[l]->head->lin
            if (auxLA != NULL){
               novo->next = auxLA->next;
               auxLA->next = novo;
            }else{
               novo->next = mat->lin[l]->head;
               mat->lin[l]->head = novo;
            }

            // auxCA é nulo apenas quando mat->col[c]->head for nulo ou
            // c é menor que mat->col[c]->head->col
            if (auxCA != NULL){
               novo->down = auxCA->down;
               auxCA->down = novo;
            }else{
               novo->next = mat->col[c]->head;
```

```
                mat->col[c]->head = novo;
            }
        }
    // Remoção de um elemento na posição (l, c)
    }else if ((auxLP != NULL) && (auxLP->col == c)){
        auxCA = obter_celula_antes_linha(l, mat->col[c]);
        auxCA->down = auxLP->down;
        auxLA->next = auxLP->next;

        free(auxLP);
    }
  }
}
```

Referências

Oliva, J. T. Matrizes Esparsas. AE22CP - Algoritmos e Estrutura de Dados I. Notas de Aula. Engenharia de Computação. Dainf/UTFPR/Pato Branco, 2020.

Roman, N. T.; Digiampietri, L. A. Matriz Esparsa. ACH2023 - Algoritmos e Estrutura de Dados I. Notas de Aula. Sistemas de Informação. EACH/USP/São Paulo, 2018