ALGORITMOS E ESTRUTURAS DE DADOS I

Prof. Caio César de Freitas Dantas

TAD - Listas

TAD é uma ideia, um modelo que descreve as características de determinado objeto e que tipo de operações podem ser feitas sobre ele.

Quando implementamos um tipo abstrato, é importante que aqueles que utilizam nosso código não tenham que se preocupar com a forma como esse tipo foi implementado. Basta que ele saiba como utilizá-lo, ou seja, o usuário se utiliza de uma abstração.

TAD - Listas

O tipo abstrato de dados que queremos implementar é uma lista.

Uma lista é um agrupamento de valores. Diferente de um conjunto, os elementos de uma lista são ordenados e seus valores podem se repetir.

Ordenados, não quer dizer que os elementos se encontram necessariamente em ordem crescente ou decrescente, mas sim que existe um primeiro elemento, seguido por um segundo elemento e assim por diante.

Ou seja, os elementos são dispostos de forma sequencial, numa ordem fixa.

TAD - Listas

Para que essa lista tenha alguma utilidade é preciso que exista a possibilidade de realizar certas operações na mesma.

Por exemplo, adicionar itens, remover itens, contar a quantidade de itens, verificar o valor de determinado item, etc.

Contanto que a lista cumpra esses requisitos, você pode implementá-la das mais variadas formas.

Entretanto, as duas formas mais comuns de implementação de uma lista são através de vetores ou de listas encadeadas.

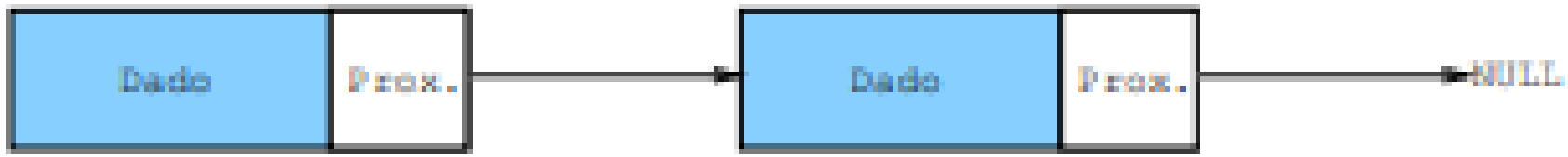
Listas Encadeadas

Uma lista encadeada (ou lista ligada) é uma representação de uma sequencia de objetos na memoria do computador.

Cada elemento é armazenado em uma célula ou nó da lista.

De maneira simplificada, um nó é composto de duas partes:

- A informação (ou o dado) de interesse;
- E uma referencia para o próximo nó.



Listas Encadeadas - Vantagens

- O principal benefício de uma lista encadeadas em relação a vetores é o fato de que os elementos de uma lista podem ser facilmente inseridos ou removidos.

- E isso pode ser feito sem necessidade de realocação ou reorganização de toda a estrutura, uma vez que os nós não precisam ser armazenados em sequencia na memoria.

- Outro ponto importante é a facilidade de inserção e remoção de nós em qualquer ponto da lista, tomados os devidos cuidados nas atualizações das referencias

Listas Encadeadas - Desvantagens

- Por outro lado, listas encadeadas por si só não permite acesso direto a um dado, ou qualquer forma eficiente de indexação. Assim, muitas operações básicas, como buscar um nó com uma determinada informação, podem significar percorrer a maioria ou todos os elementos da lista

Listas Encadeadas

Listas encadeadas são representadas em C utilizando-se estruturas (struct).

A estrutura de cada célula de uma lista ligada pode ser definida da seguinte maneira:

```
struct cel {
int dado;
struct cel *prox;
};
```

Uma outra maneira de representar, utilizando typedef, seria:

```
typedef struct cel celula;
struct cel {
int dado;
celula *prox;
};
```

Listas Encadeadas

- Uma célula c e um ponteiro p para uma célula podem ser declarados assim:

  célula c;
  célula *p;

- Se c é uma célula, então c.dado é o conteúdo da célula e c.prox é o endereço da próxima célula.

- Se p é o endereço de uma célula, então p->dado é o conteúdo da célula e p->prox é o endereço da próxima célula.

- Se p é o endereço da ultima célula da lista, então p->prox vale NULL.

Listas Encadeadas

- O endereço de uma lista encadeada é o endereço de sua primeira célula. Se p é o endereço de uma lista, pode-se dizer simplesmente “p é uma lista”.

Operações:

- Inserção
- Remoção
- Busca
- Cabeça da Lista

Listas Encadeadas - Inserção

- Considere o problema de inserir uma nova célula em uma lista encadeada. Suponha que quero inserir a nova célula entre a posição apontada por p e a posição seguinte. (É claro que isso só faz sentido se p é diferente de NULL).

```
void insere (int x, celula *p)
{
    celula *nova;
    nova = malloc (sizeof (celula));
    nova->conteudo = x;
    nova->prox = p->prox;
    p->prox = nova;
}
```

Listas Encadeadas - Inserção

Não é preciso movimentar células para abrir espaço para um nova célula, como fizemos para inserir um novo elemento em um vetor. Basta mudar os valores de alguns ponteiros.

Se a lista tem cabeça, a função pode ser usada para inserir no início da lista: basta que p aponte para a célula-cabeça.

Mas no caso de lista sem cabeça a função não é capaz de inserir antes da primeira célula.

```
void insere (int x, celula *p)
{
   celula *nova;
   nova = malloc (sizeof (celula));
   nova->conteudo = x;
   nova->prox = p->prox;
   p->prox = nova;
}
```

Listas Encadeadas - Remoção

- Considere o problema de remover uma certa célula de uma lista encadeada. Como especificar a célula em questão? A ideia mais óbvia é apontar para a célula que quero remover.

- Mas é fácil perceber que essa ideia não é boa; é melhor apontar para a célula anterior à que quero remover. (Infelizmente, não é possível remover a primeira célula usando essa convenção).

```
void remove (celula *p)
{
   celula *lixo;
   lixo = p->prox;
   p->prox = lixo->prox;
   free (lixo);
}
```

Listas Encadeadas - Remoção

- Não é preciso copiar informações de um lugar para outro, como fizemos para remover um elemento de um vetor: basta mudar o valor de um ponteiro.

- A função consome sempre o mesmo tempo, quer a célula a ser removida esteja perto do início da lista, quer esteja perto do fim.

```
void remove (celula *p)
{
   celula *lixo;
   lixo = p->prox;
   p->prox = lixo->prox;
   free (lixo);
}
```

Listas Encadeadas - Busca

é fácil verificar se um objeto x pertence a uma lista encadeada, ou seja, se é igual ao conteúdo de alguma célula da lista:

Esta função recebe um inteiro x e uma lista encadeada le de inteiros e devolve o endereço de uma célula que contém x. Se tal célula não existe, devolve NULL.

```
celula *busca (int x, celula *le)
{
    celula *p;
    p = le;
    while (p != NULL && p->conteudo != x)
        p = p->prox;
    return p;
}
```

Listas Encadeadas - Impressão

é fácil verificar se um objeto x pertence a uma lista encadeada, ou seja, se é igual ao conteúdo de alguma célula da lista:

Esta função recebe um inteiro x e uma lista encadeada le de inteiros e devolve o endereço de uma célula que contém x. Se tal célula não existe, devolve NULL.

```
void imprime (celula *le) {
    celula *p;
    for (p = le; p != NULL; p = p->prox)
        printf ("%d\n", p->conteudo);
}
```

Listas Encadeadas – Cabeça da Lista

Às vezes convém tratar a primeira célula de uma lista encadeada como um mero marcador de início e ignorar o conteúdo da célula. Dizemos que a primeira célula é a cabeça (= head cell = dummy cell) da lista encadeada.

Uma lista encadeada le com cabeça está vazia se e somente se le->prox == NULL. Para criar uma lista encadeada vazia com cabeça, basta dizer

```
celula *le;
   le = malloc (sizeof (celula));
   le->prox = NULL;
```

```
void imprima (celula *le) {
   celula *p;
   for (p = le->prox; p != NULL; p = p->prox)
      printf ("%d\n", p->conteudo);
}
```

Listas Encadeadas

- Busca e Insere
- Busca e Remove
- Outros tipos de Listas

FIM!