LABORATÓRIO DE ALGORITMOS E ESTRUTURAS DE DADOS I

Prof. Caio César de Freitas Dantas

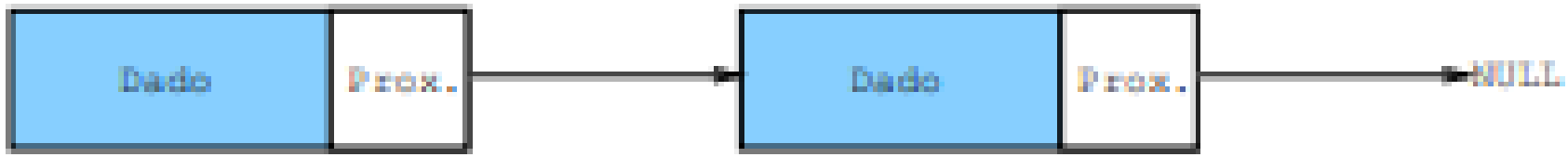
Listas Encadeadas

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

- 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 - 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 - 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

Suponha dada uma lista encadeada le, com cabeça. Queremos inserir na lista uma nova célula com conteúdo x imediatamente antes da primeira célula que contém y.

Esta função recebe uma lista encadeada le com cabeça e insere na lista uma nova célula imediatamente antes da primeira que contém y. Se nenhuma célula contém y, insere a nova célula no fim da lista. O conteúdo da nova célula é x.

Listas Encadeadas – Busca e Insere

Esta função recebe uma lista encadeada le com cabeça e insere na lista uma nova célula imediatamente antes da primeira que contém y.

```
void busca_e_insere (int x, int y, celula *le){
    celula *p, *q, *nova;
    nova = malloc (sizeof (celula));
    nova->conteudo = x;
    p = le;
    q = le->prox;
    while (q != NULL && q->conteudo != y) {
        p = q;
        q = q->prox;
    }
    nova->prox = q;
    p->prox = nova;
}
```

Listas Encadeadas – Busca e Remove

Dada uma lista encadeada le de inteiros e um inteiro y, queremos remover da lista a primeira célula que contiver y.

Se tal célula não existir, não é preciso fazer nada.

Vamos supor que a lista tem cabeça; assim, não será preciso mudar o endereço da lista, mesmo que a célula inicial contenha y.

Listas Encadeadas – Busca e Remove

Esta função recebe uma lista encadeada le com cabeça e remove da lista a primeira célula que contiver y, se tal célula existir.

```
void busca_e_remove (int y, celula *le){
    celula *p, *q;
    p = le;
    q = le->prox;
    while (q != NULL && q->conteudo != y) {
        p = q;
        q = q->prox;
    }
    if (q != NULL) {
        p->prox = q->prox;
        free (q);
    }
}
```

Lista Duplamente Encadeada

- As listas duplamente encadeadas podem ser usadas quando várias operações de inserção e remoção de elementos são necessárias.

- A conexão entre os elementos é feita através de dois ponteiros (um que aponta para o elemento anterior e o outro para o seguinte).

- Para acessar um elemento, a lista pode ser percorrida pelos dois lados. O movimento é feito em ambas as direções, do primeiro para o último elemento e vice-versa.

Lista Duplamente Encadeada

Como construir o protótipo de um elemento da lista

- Em primeiro lugar, estabeleça um elemento da lista com o tipo struct. O elemento da lista terá um campo, um ponteiro anterior e um ponteiro seguinte.

- Os ponteiros anterior e seguinte devem ser do mesmo tipo que o elemento, caso contrário eles não poderão apontar para um elemento da lista.

- O ponteiro do anterior permitirá o acesso ao elemento anterior, enquanto que o ponteiro seguinte vai permitir o acesso ao próximo elemento:

Lista Duplamente Encadeada

O ponteiro do anterior permitirá o acesso ao elemento anterior, enquanto que o ponteiro seguinte vai permitir o acesso ao próximo elemento:

```
typedef struct dl_elementoLista {
  char *dado;
  struct dl_elementoLista *anterior;
  struct dl_elementoLista *seguinte;
}dl_elemento;
```

Lista Duplamente Encadeada

Para o controle da lista, o melhor é salvar certos elementos: o primeiro elemento, o último elemento e o número de elementos.

- O ponteiro inicial terá o endereço do primeiro elemento da lista;
- O ponteiro final abrigará o endereço do último elemento da lista;
- A variável tamanho contém o número de elementos.

```
dl_elemento *início;
dl_elemento *fim;
int tamanho;
```

Lista Duplamente Encadeada

Deverá ser feita antes de qualquer outra operação da lista. A função inicia o ponteiro de lançamento e de fim, sempre através do ponteiro NULL e valor 0.

```
void inicialização (Lista *lista){
lista>início = NULL;
lista>fim = NULL;
tamanho = 0;
}
```

Lista Duplamente Encadeada

Como inserir e um elemento na lista

- Declaração dos elementos a ser inseridos;
- Alocação da memória para o novo elemento;
- Preenchimento do conteúdo do campo de dados;
- Atualização dos ponteiros para o elemento anterior e o elemento seguinte;
- A atualização dos ponteiros com vistas do primeiro e do último elemento, se necessário;
- Caso particular: em uma lista com apenas um elemento, o primeiro também é o último;
- Atualização do tamanho da lista.

Lista Duplamente Encadeada

Para adicionar um elemento na lista, existem várias soluções:

- Inserção em uma lista vazia;
- Inserção no início da lista;
- Inserção no fim da lista;
- Inserção antes de um elemento;
- Inserção depois de um elemento.

Lista Duplamente Encadeada

Adicionar em uma lista vazia

Passos:

- Alocação da memória para o novo elemento;
- Preenchimento do campo de dados do novo elemento;
- O ponteiro anterior do novo elemento indicará o NULL;
- O ponteiro seguinte do novo elemento dirigirá para NULL;
- Os ponteiros de início e de fim indicarão para o novo elemento;
- O tamanho é atualizado.

Lista Duplamente Encadeada

Adicionar em uma lista vazia

```
int inserção_na_lista_vazia (dl_Lista * lista, char *dado){
  dl_elemento *novo_elemento;
  if ((novo_elemento = aloc (novo_elemento)) == NULL)
    return -1;
  strcpy (novo_elemento>dado, dado);

  novo_elemento->anterior = lista>início;
  novo_elemento->seguinte = lista>fim;
  lista->início = novo_elemento;
  lista->fim = novo_elemento;
  lista->tamanho++;
  return 0;
}
```

Lista Duplamente Encadeada

Adicionar no início da lista

Passos:

- Alocação da memória para o novo elemento;
- Preenchimento do campo de dados do novo elemento;
- O ponteiro anterior ao novo elemento aponta para NULL;
- O ponteiro seguinte aponta para o 1° elemento;
- O ponteiro anterior ao 1° elemento indica o novo elemento;
- O ponteiro de início direciona para o novo elemento;
- O ponteiro de fim não muda;
- O tamanho é incrementado.

Lista Duplamente Encadeada

Adicionar no início da lista

```
int ins_início_lista (dl_Lista * lista, char *dado){
  dl_elemento *novo_elemento;
  if ((novo_elemento = aloc (novo_elemento)) == NULL)
    return -1;
  strcpy (novo_elemento->dado, dado);
  novo_elemento->anterior = NULL;
  novo_elemento->seguinte = lista>início;
  lista->início->anterior = novo_elemento;
  lista->início = novo_elemento;
  lista->tamanho++;
  return 0;
}
```

Lista Duplamente Encadeada

Adicionar no fim da lista

Passos:

- Alocação da memória para o novo elemento;
- Preenchimento do campo de dados do novo elemento;
- O ponteiro seguinte ao novo elemento aponta para NULL;
- O ponteiro anterior ao novo elemento aponta para o último elemento (é o ponteiro de fim que contém, por enquanto, o seu endereço);
- O ponteiro seguinte em relação ao último elemento indicará o novo elemento;
- O ponteiro de fim aponta para o novo elemento;
- O ponteiro de início não muda;
- O tamanho é incrementado.

Lista Duplamente Encadeada

Adicionar no fim da lista

```
int ins_fim_lista (dl_Lista * lista, char *dado){
  dl_elemento *novo_elemento;
  if ((novo_elemento = aloc (novo_elemento)) == NULL)
    return -1;
  strcpy (novo_elemento->dado, dado);
  novo_elemento->seguinte = NULL;
  novo_elemento->anterior = lista>fim;
  lista->fim->seguinte = novo_elemento;
  lista->fim = novo_elemento;
  lista->tamanho++;
  return 0;
}
```

Lista Duplamente Encadeada

Remover Elemento

A posição escolhida é 1 (o caso da remoção do primeiro elemento da lista).

O ponteiro remove elemento e conterá o endereço do primeiro elemento e o ponteiro de início integrará o endereço mantido pelo ponteiro seguinte ao 1° elemento que queremos remover

Se este ponteiro equivale a NULL, então atualizaremos o ponteiro de fim já que é o caso de uma lista com apenas um elemento. Caso contrário, apontaríamos o ponteiro anterior ao segundo elemento para NULL).

Lista Duplamente Encadeada

Remover Elemento

```
int remov(dl_Lista *lista, int pos){
  int i;
  dl_elemento *remov_elemento,*em andamento;

  if(lista>tamanho == 0)
    return -1;

  if(pos == 1){ /* remoção do 1º elemento */
    remov_elemento = lista>início;
    lista>início = lista>início>seguinte;
    if(lista>início == NULL)
      lista>fim = NULL;
    else
      lista>início>anterior == NULL;
```

Lista Duplamente Encadeada

Exibir Elemento

```
void exibe(dl_Lista *lista){ /* mostrar avançando */
  dl_elemento *andamento;
  andamento = lista>início; /* ponto de partida do 1º elemento */
  printf("[ ");
  while(andamento != NULL){
    printf("%s ",andamento>dado);
    andamento = andamento>seguinte;
  }
```

Lista Circular

```c
typedef struct no{
    int valor;
    struct no *proximo;
}No;

typedef struct{
    No *inicio;
    No *fim;
    int tam;
}Lista;

void criar_lista(Lista *lista){
    lista->inicio = NULL;
    lista->fim = NULL;
    lista->tam = 0;
}
```

Lista Duplamente Encadeada

Exercício

Implemente um programa em C que utiliza a estrutura apresentada para implementar uma lista. O programa deve mostrar ao usuário as seguintes opções:

- Sair;
- Inserir no Inicio;
- Inserir no Final;
- Remover Elemento;
- Buscar Elemento;
- Zerar Lista;
- Exibir Lista;

FIM!