Aula 29

MC 102 - Algoritmos e Programação de Computadores

Listas Ligadas II.

Listas Ligadas

Algumas operações em listas ligadas podem variar para permitir a utilização de outros tipos de dados.

As principais variações são nos algoritmos de Inserção e no de Remoção.

Listas Ligadas - Propriedades

- As Inserções em Listas Ligadas podem ser, basicamente, das seguintes maneiras:
  - Inserir um elemento no Final da Lista
  - Inserir um elemento no Início da Lista
  - Inserir um elemento em uma Lista Ordenada
- Já a Remoção pode ser feita das formas:
  - Buscar um elemento e remover da lista
  - Remover o Primeiro elemento
  - Remover o Último elemento

Inserção (InsereFinal)

- Um novo nó é sempre inserido no final da Lista.
- Se a Lista está vazia
  - Faz a Lista apontar para o novo nó
- Senão, procura o último elemento
  - Faz o último elemento apontar para o novo nó.

Inserção (InsereFinal)

```
void InsereFim(lista *L, pno No) {
  pno p = *L;

  if (p == NULL)
    *L = No;
  else {
    while (p->prox != NULL)
      p = p->prox;
    p->prox = No;
  }
}
```

Inserção (InsereFinal - Recursivo)

```
void InsereFim(lista *L, pno No) {
  pno p = *L;

  if (p == NULL)
    *L = No;
  else
    InsereFim(&(p->prox), No);
}
```

Inserção (InsereInício)

- Um novo nó é sempre inserido no início da Lista.
- Se a Lista está vazia
  - Faz a Lista apontar para o novo nó
- Senão
  - Faz o nó apontar para o primeiro da Lista
  - Faz a Lista apontar para o novo nó

Inserção (InsereInicio)

```
void InsereInicio(lista *L, pno No) {

 if (*L != NULL)
   No->prox = *L;

 *L = No;

 }
```

Inserção (InsereOrdenado)

- Insere um nó mantendo a lista ordenada.
- Se a Lista está vazia
  - Faz a Lista apontar para o novo nó
- Senão, encontra o primeiro nó maior ou igual ao novo nó
  - Faz o nó apontar para o nó atual
  - Faz o anterior apontar para o novo nó

Inserção (InsereOrdenado)

```
void InsereOrdenado(lista *L, pno No) {
  pno a = NULL, p = *L;
  if (p == NULL)
    *L = No;
  else {
    while (p != NULL && p->info < No->info) {
      a = p; p = p->prox;
    }
    if (a == NULL) {
      No->prox = *L;
      *L = No;
    } else {
      No->prox = p;
      a->prox = No;
    }
  }
}
```

Inserção (InsereOrd Recursivo)

```
void InsereOrdenado(lista *L, pno No) {
  pno p = *L;

  if (p == NULL)
    *L = No;
  else if (p->info >= No->info) {
    No->prox = p;
    *L = No;
  } else
    InsereOrdenado(&(p->prox), No);
}
```

Remoção (RemoveNo)

Remove o nó cuja informação é **info**.

- Se a Lista está vazia, nada a fazer.
- Senão, procura info
  - Guarda o endereço do nó a ser removido
  - Faz o nó anterior apontar para o próximo
  - Libera a memória

Remoção (RemoveNo)

```
void RemoveNo(lista *L, TipoDado Info) {
 pno a = NULL, p = *L;

 while (p != NULL && p->info != Info) {
  a = p;
  p = p->prox;
 }
 if (p != NULL) {
  if (p->info == Info) {
   if (a == NULL)
    *L = p->prox;
   else
    a->prox = p->prox;
   free(p);
  }
 }
}
```

Remoção (RemoveNo - Recursivo)

```
void RemoveNo(lista *L, TipoDado Info) {
  pno p = *L;

  if (p != NULL) {
    if (p->info == Info) {
      *L = p->prox;
      free(p);
    } else
      RemoveNo(&(p->prox), Info);
  }
}
```

Remoção (RemovePrimeiro)

Neste caso, sempre o primeiro elemento da lista será removido

- Se a Lista está vazia, nada a fazer.
- Senão
  - Guarda o endereço do nó primeiro nó
  - Faz o a lista apontar para o próximo
  - Libera a memória

RemovePrimeiro

```
void RemovePrimeiro(lista *L) {
  pno p = *L;

  if (p != NULL) {
    *L = p->prox;
    free(p);
  }
}
```

Remoção (RemoveUltimo)

Sempre o último nó da lista é removido

- Se a Lista está vazia, nada a fazer.
- Senão, procura o último nó
  - Guarda o endereço do nó último nó
  - Faz o nó anterior apontar para NULL
  - Libera a memória

RemoveUltimo

```
void RemoveUltimo(lista *L) {
 pno a = NULL, p = *L;

 while (p != NULL && p->prox != NULL) {
  a = p;
  p = p->prox;
 }
 if (p != NULL) {
  if (a == NULL)
   *L = p->prox;
  else
   a->prox = p->prox;
  free(p);
 }
}
```

RemoveUltimo - Recursivo

```
void RemoveUltimo(lista *L) {
  pno p = *L;

  if (p != NULL) {
    if (p->prox == NULL) {
      *L = p->prox;
      free(p);
    } else {
      RemoveUltimo(&(p->prox));
    }
  }
}
```

Listas Ligadas - Propriedades

- Usano Lista Ligada, podemos construir uma Fila (FIFO – First In, First Out) se as operações forem:
  - Inserir um elemento no Final da Lista
  - Remover o Primeiro elemento
- Isso significa que:
  - O elemento mais antigo da lista se encontra na primeira posição
  - O elemento mais novo da lista se encontra na última posição
- Age da mesma forma que uma fila de espera

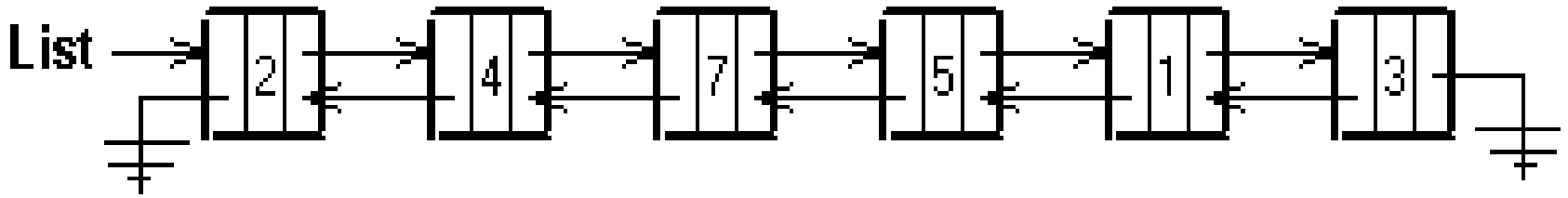
Listas Ligadas - Propriedades

- Para o tipo Pilha (FILO – First In, Last Out) as operações em Lista Ligada devem ser:
  - Inserir um elemento no Início da Lista
  - Remover o Primeiro elemento
- Isso significa que:
  - O elemento mais antigo da lista será o último a ser removido, e se encontra na última posição
  - O elemento mais novo da lista se encontra na primeira posição
- Se comporta como uma pilha de pratos

Listas Duplamente Ligadas

Listas Duplamente Ligadas

- A partir de qualquer nó é possível chegar a qualquer outro nó da lista.



Listas Duplamente Ligadas

```
/* Defnicao do Tipo de Dado que sera utilizado */
typedef unsigned int TipoDado;

/* Definicao da estrutura de um no e do tipo no */
typedef struct no {
  TipoDado info;
  struct no *prox;
  struct no *ant;
} no;

/* Definicoes de Tipos para auxiliar nos cabecalhos das funcoes */
typedef no* pno;
typedef no* lista;
```

Listas Duplamente Ligadas - Propriedades

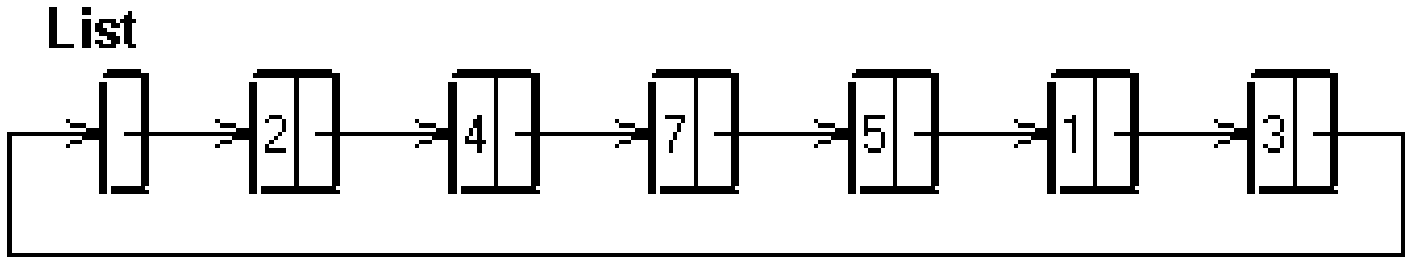
Propriedades de Listas Duplamente Ligadas

- Uma lista vazia tem seus apontadores igual a NULL
- O último nó tem seu ponteiro prox igual a NULL
- O primeiro nó tem seu ponteiro ant igual a NULL
- Qualquer nó da lista sempre aponta para o nó seguinte (usando o ponteiro prox), e para se anterior (usando o ponteiro ant).
- Todo nó criado deve ter seus ponteiros igual a NULL, até que sua posição na lista seja definida

Listas Ligadas Circulares

Listas Ligadas Circulares

- A partir de qualquer nó é possível chegar a qualquer outro nó da lista, fazendo-se um loop.
- Em geral, Listas Ligadas Circulares são implementadas utilizando-se nó cabeça



Listas Ligadas Circulares - Propriedades

Propriedades de Listas Ligadas Circulares

- Uma lista vazia aponta para o nó cabeça
- O último nó ponta para o nó cabeça
- O primeiro nó tem seu ponteiro ant igual a NULL
- Qualquer nó da lista sempre aponta para o nó seguinte.