Estruturas Compostas Parte III

Listas

Leonardo Tórtoro Pereira

Slides fortemente baseados no material do professor Ricardo Farias:
http://www.cos.ufrj.br/~rfarias/cos121/

O que é Lista?

O que é Lista?

- Estrutura que armazena uma sequência de dados
- Pode ser uma lista linear sequencial
  - Pilhas, filas, deques*
- Ou uma lista linear encadeada
  - Elementos não são consecutivos na memória (necessariamente)

Lista Linear Encadeada

Lista Encadeada

- É uma representação de uma sequência de objetos do mesmo tipo na memória RAM do computador
- Cada elemento (nó) é armazenado em uma célula
  - O primeiro na primeira célula
  - O segundo na segunda célula
  - Etc.

Lista Encadeada

- Exemplo:
  - Consultório médico
    - Pessoas estão na sala de espera em lugares “aleatórios”
    - Mas sabe-se a ordem de atendimento

Lista Encadeada

- Mais exemplos:
  - Fila de banco
  - Letras em uma palavra
  - Relação de notas de alunos na turma
  - Dias da semana
  - Vagões de trem

Lista Encadeada

- ➔ Vantagem sobre listas sequenciais
  - ◆ Ganho de desempenho em velocidade em:
    - Remoção e adição de elementos
- ➔ Na contígua é necessário mover todos os elementos para uma nova lista para realizar as operações
- ➔ Na encadeada só é preciso alterar referências dos nós

Lista Encadeada

- É válido lembrar que a vantagem é realmente percebida em listas grandes
  - Mínimo de centenas de nós

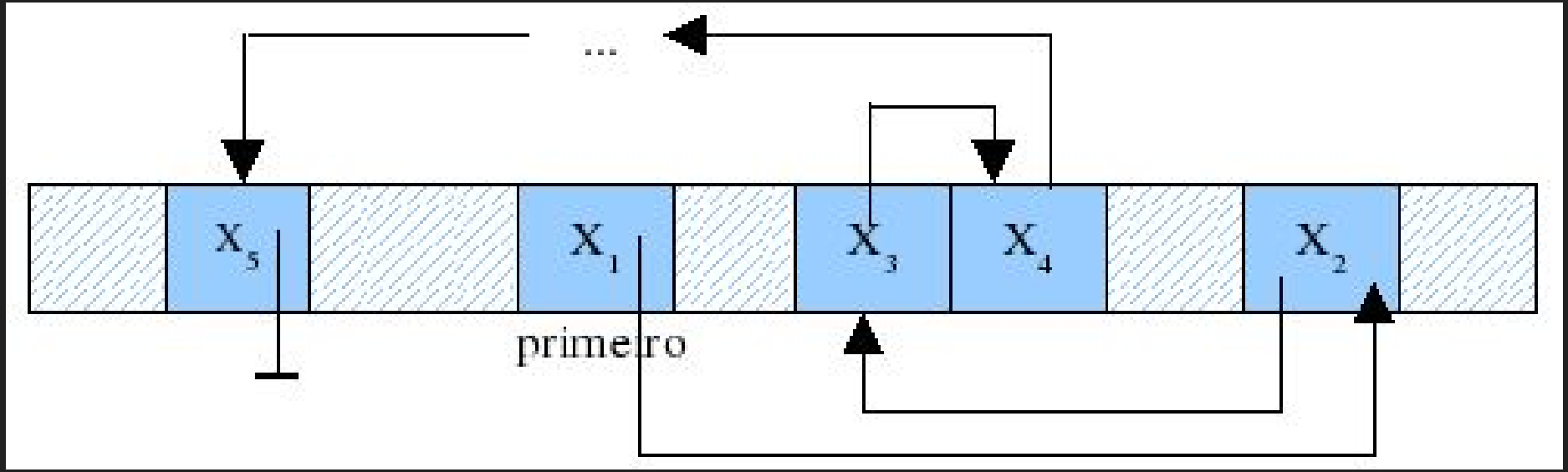
Lista Encadeada

- Existem duas implementações
  - Simplesmente encadeada
  - Duplamente encadeada

Lista Simplesmente Encadeada

Lista Simplesmente Encadeada

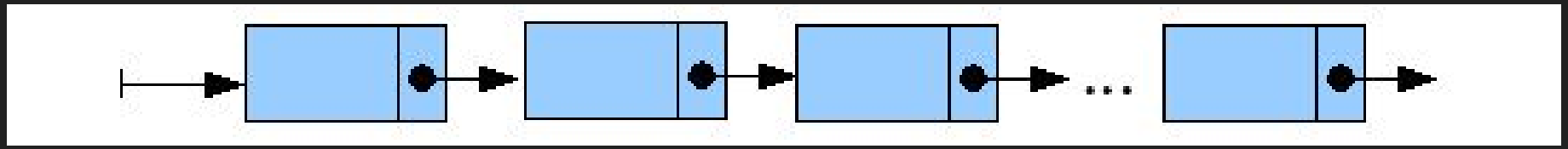
- Cada elemento possui
  - Espaço para armazenar informação
  - Espaço para armazenar referência do local na memória do próximo elemento (ou do anterior)



http://www.cos.ufrj.br/~rfarias/cos121/aula_11.html

Lista Simplesmente Encadeada

➔ Podemos fazer uma representação simbólica da lista simplesmente encadeada da seguinte maneira:

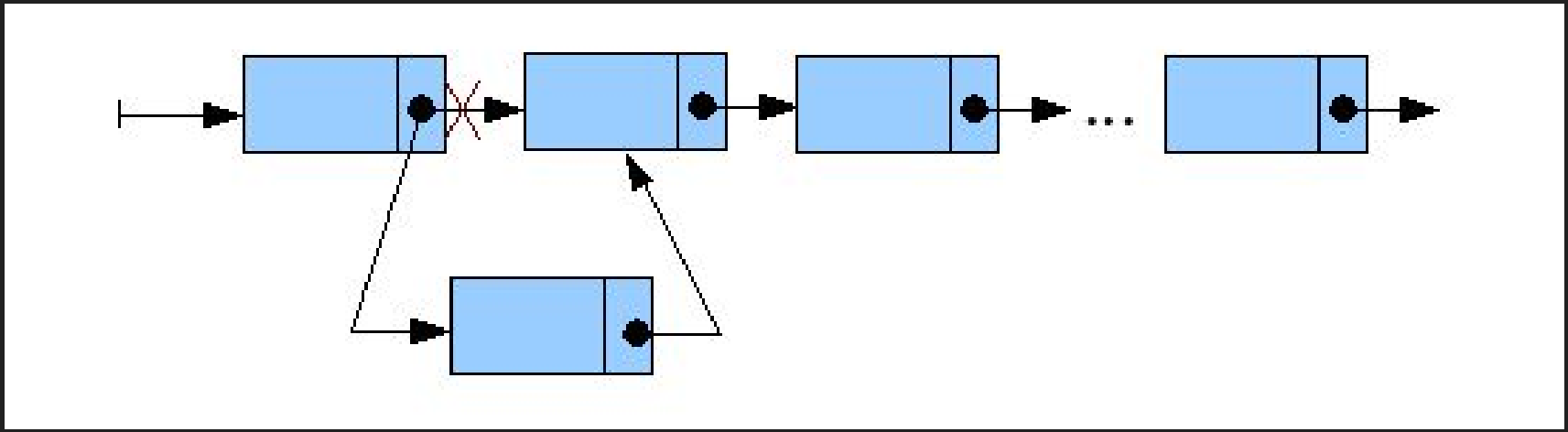


http://www.cos.ufrj.br/~rfarias/cos121/aula_11.html

Lista Simplesmente Encadeada: Manipulação

Manipulação

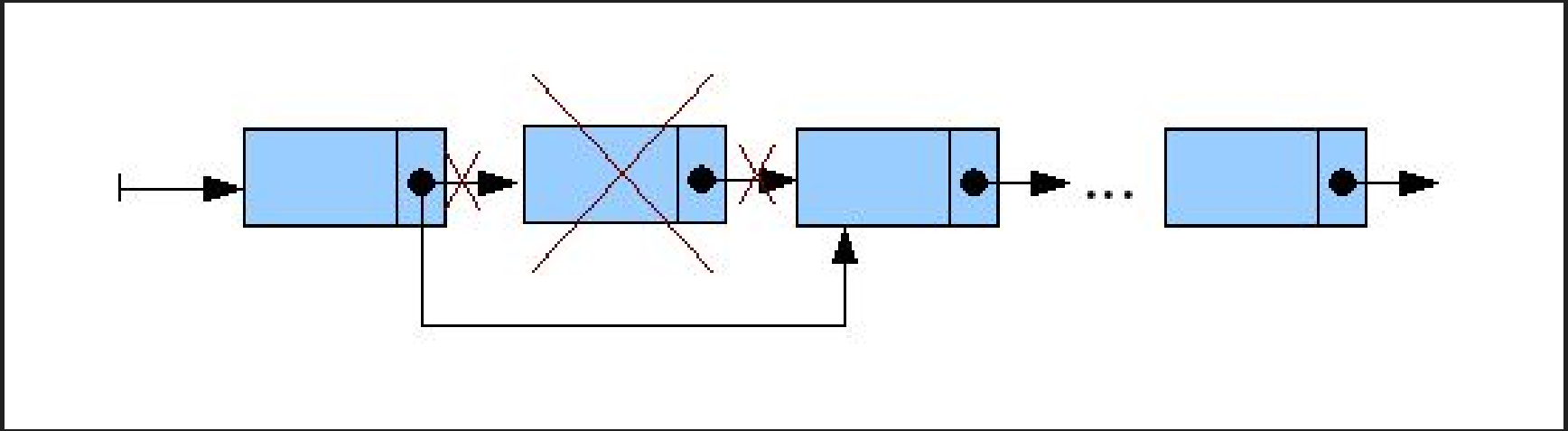
- Inserção
  - Manipula-se a referência ao próximo nó da lista



http://www.cos.ufrj.br/~rfarias/cos121/aula_11.html

Manipulação

- Remoção
  - Manipula-se a referência ao próximo nó da lista



http://www.cos.ufrj.br/~rfarias/cos121/aula_11.html

Estrutura

```
struct Nodo {
   int info;
   struct Nodo *prox;
};

struct ListaSimplesEnc {
   struct Nodo *prim;
};
```

Criar e Mostrar Lista

```
void criarLista (struct ListaSimplesEnc *pList) {
   pList -> prim = NULL;
}

void mostrarLista (struct ListaSimplesEnc *pList){
   struct Nodo *p;
   for (p = pList -> prim; p != NULL; p = p->prox) {
      printf("%d\t", p->info);
   }
   printf("\n");
}
```

Inserir item no Início

```
void inserirIni (struct ListaSimplesEnc *pList, int v){
    struct Nodo *novo;
    novo = (struct Nodo*) malloc (sizeof (struct Nodo));
    novo -> info = v;
    novo -> prox = pList -> prim;
    pList -> prim = novo;
}
```

Remover item no Início

```
void removerIni (struct ListaSimplesEnc *pList){

   struct Nodo *pAux = pList -> prim;
   pList -> prim = pList -> prim -> prox;
   free(pAux);

}
```

Inserir item em ordem

```
void inserirOrd (struct ListaSimplesEnc *pList, int v){
    struct Nodo *novo;
    novo = (struct Nodo*) malloc (sizeof (struct Nodo));
    novo -> info = v;
    struct Nodo *pAtu, *pAnt;
    pAnt = NULL;
    pAtu = pList -> prim;
    while ( pAtu != NULL && pAtu->info < v){
        pAnt = pAtu;
        pAtu = pAtu -> prox;
    }
    novo -> prox = pAtu -> prox;
    pAnt -> prox = novo;
}
```

Verifica se Está Vazia

```
int estaVazia(struct ListaSimplesEnc *pList)
{

   return (pList->prim == NULL);

}
```

Testando!

```
void main () {
   struct ListaSimplesEnc minhaLista;
   int valor, op;

   criarLista(&minhaLista);

   while( 1 ){
```

Testando!

```
printf( "1 - Inserir elemento no inicio\n" );
printf( "2 - Inserir elemento em ordem (so se a lista
                                estiver ordenada)\n" );
printf( "3 - Remover elemento no inicio\n" );
printf( "4 - Remover elemento\n" );
printf( "5 - Mostrar lista\n" );
printf( "6 - Sair\n" );
printf( "Opcao? " );
scanf( "%d", &op );

switch( op ){
```

Testando!

```
case 1: // inserir elemento no inicio

            printf( "Valor? " );
            scanf( "%d", &valor );
            inserirIni(&minhaLista, valor);
            break;
```

Testando!

```
case 2: // inserir elemento ordenado
            printf( "Valor? " );
            scanf( "%d", &valor );
            inserirOrd(&minhaLista, valor);
            break;
```

Testando!

```
case 3: // remover o primeiro
          removerIni(&minhaLista)
          break;
case 4: // remover determinado elemento
          //Implementem :)
          break;
```

Testando!

```
            case 5: //  mostrar lista
            if (estaVazia(&minhaLista)) {
               printf("Lista vazia");
            }
            else {
               mostrarLista(&minhaLista);
            }
            break;
            case 6: // abandonar o programa
               exit(0);
        }
    }
}
```

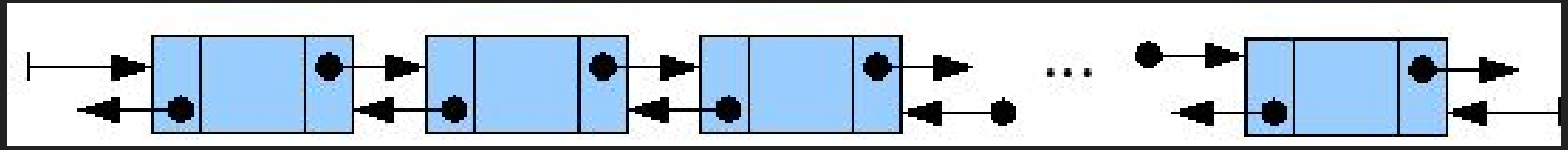
Lista Duplamente Encadeada

Lista Simplesmente Encadeada

- Cada elemento possui
  - Espaço para armazenar informação
  - Espaço para armazenar referência do local na memória do próximo elemento
  - Espaço para armazenar referência do local na memória do elemento anterior

Lista Simplesmente Encadeada

- Representação simbólica da lista duplamente encadeada:



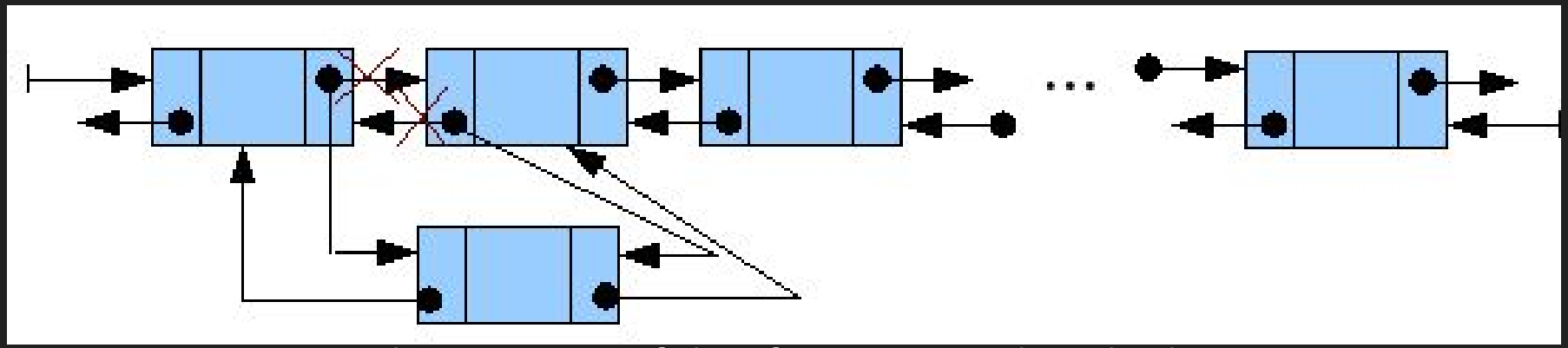
- O campo *próximo* faz referência ao próximo nó da lista
- O anterior faz referência ao anterior
- Facilita a navegação (agora nos dois sentidos)

http://www.cos.ufrj.br/~rfarias/cos121/aula_11.html

Lista Duplamente Encadeada: Manipulação

Manipulação

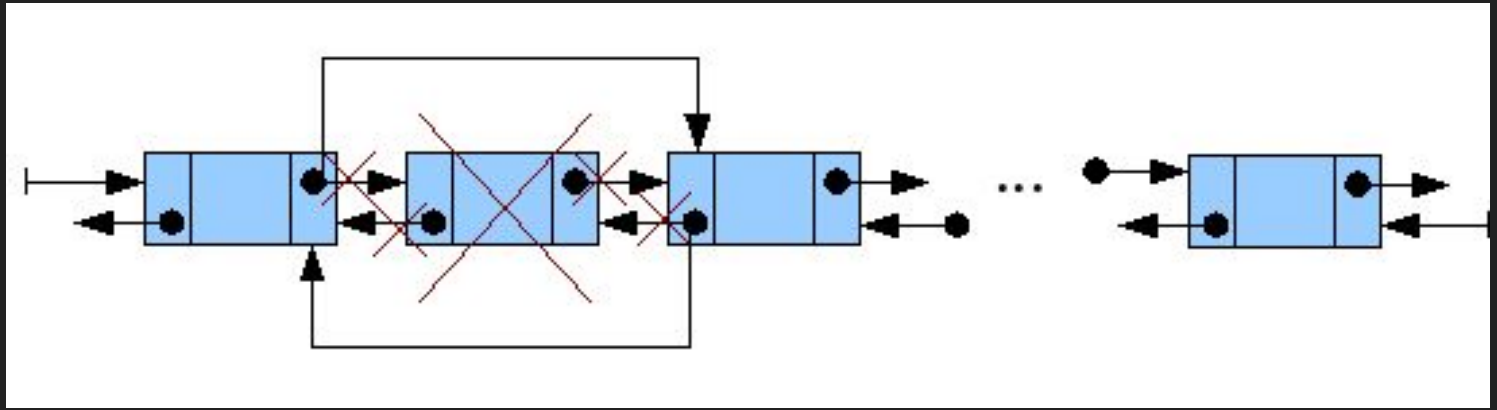
- Inserção
  - Manipula-se a referência ao próximo nó da lista e ao anterior



http://www.cos.ufrj.br/~rfarias/cos121/aula_11.html

Manipulação

- Remoção
  - Manipula-se a referência ao próximo nó da lista e ao anterior



http://www.cos.ufrj.br/~rfarias/cos121/aula_11.html

Estrutura

```
struct Nodo {

   int info;
   struct Nodo *ant;
   struct Nodo *prox;

}no;
```

Ler valores até entrada 0

```
void main () {
   struct Nodo *inicio = NULL, *tmp, *p;
   int v;

   while( 1 ){
      printf( "\nValor? " );
      scanf( "%d", &v );

      if ( v == 0 ) break;
```

Cria nó novo e adiciona no início se vazia

```
// cria um novo nodo para ser inserido na lista
tmp = (struct Nodo* ) malloc ( sizeof( struct
                                        Nodo ));
tmp -> info = v;
tmp -> prox = NULL;

if (inicio == NULL) { // lista vazia
   inicio = tmp;
   tmp->ant = tmp->prox = NULL;
}
```

Se tem elemento, adicionar em ordem

```
else {
   // p irá percorrer a lista
   p = inicio;
   while ( p->prox != NULL && p->info != v) {
      p = p->prox;
   }
   if ( p->info != v ){
      p->prox = tmp;
      tmp->ant = p;
   }
}
```

Mostrar a lista

```
// mostrando a lista
   p = inicio;
   while ( p != NULL ) {
      printf( "%d\t", p->info);
      p = p->prox;
   }
   printf( "\n");
}
```

Referências

Referências

- http://www.cos.ufrj.br/~rfarias/cos121/aula_10.html
- http://www.cos.ufrj.br/~rfarias/cos121/aula_11.html
- https://www.ime.usp.br/~pf/algoritmos/aulas/lista
- https://www.cs.usfca.edu/~galles/visualization/java/visualization.html