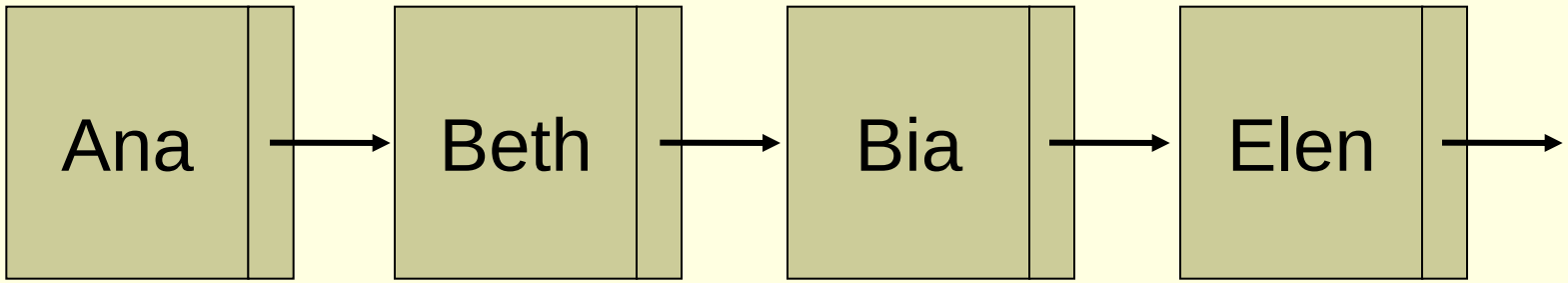
Listas Ordenadas

SCC-202 – Algoritmos e Estruturas de Dados I

Lucas Antiqueira

Listas ordenadas

- Definição
  - *Listas em que os elementos estão ordenados por algum critério*
    - Em geral, por ordem alfabética



Listas ordenadas

- Situações
  - Cadastro de funcionários por ordem alfabética
  - Lista de passageiros em um vôo
  - Lista de deputados presentes no congresso
  - Telefones da cidade

Listas ordenadas

- Nessas três situações os acessos (inserção, eliminação e consulta) são direcionados a um elemento específico, e não mais ao “primeiro” (ou último) a entrar na lista (como na fila ou pilha).
  - “Carla” foi despedida (retire seu nome do cadastro)
  - “Edmundo” é funcionário? Verifique se seu nome consta do cadastro
  - Qual o salário do funcionário “Pedro”?
  - “Sandra” foi contratada; inclua-a no cadastro

Listas ordenadas

Chave de busca → nome

- "Carla" foi despedida (retire seu nome do cadastro)
- "Edmundo" é funcionário? Verifique se seu nome consta do cadastro
- Qual o salário do funcionário "Pedro"?
- "Sandra" foi contratada; inclua-a no cadastro

Listas ordenadas

- **Operações**
  - cria(lista)
  - IsEmpty(lista)
  - IsFull(lista)
  - esta_na_lista(lista,x)
  - inserir(lista,x)
  - remover(lista,x)
  - imprimir_todos_da_lista(lista)
  - Etc.

Listas ordenadas

- Implementações
  - Seqüencial
  - Encadeada

Ordenação implica em mudanças na forma de manipulação da lista

Lista ordenada seqüencial

- O que acontece se quero incluir na lista a funcionária “Alice”?

...	...
Zorro	i+1
Zoroastro	i
Zico	i-1
...	...
Antônio	3
André	2
Ana	1

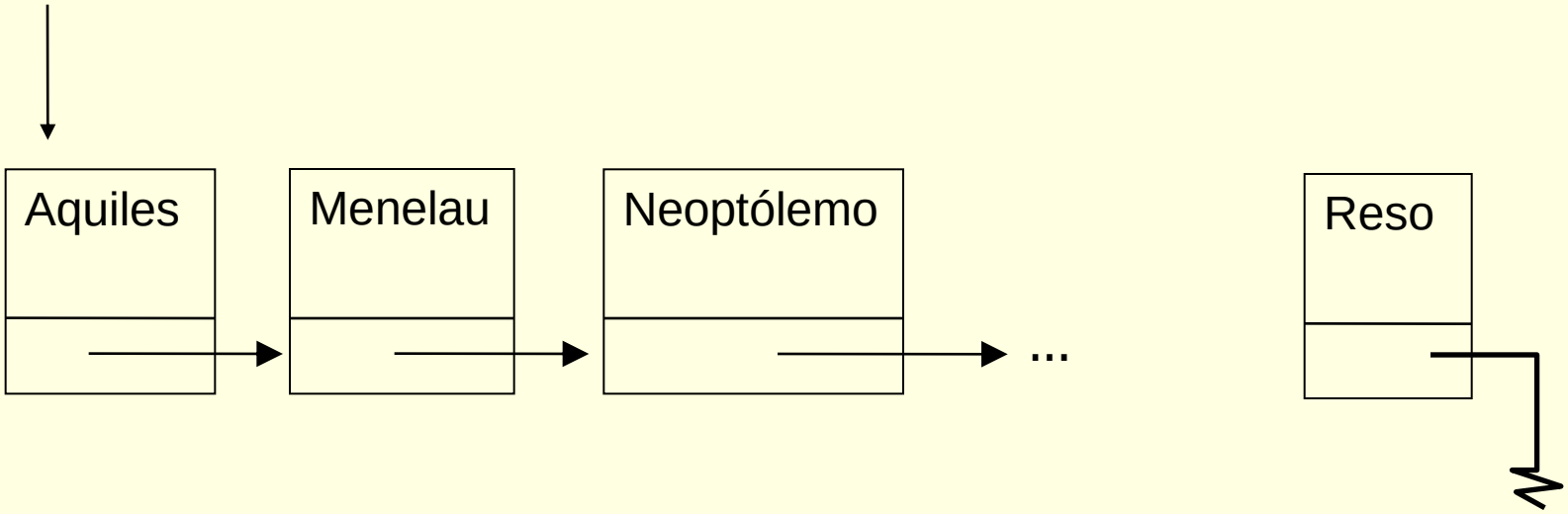
Lista ordenada seqüencial

- O que acontece se quero incluir na lista a funcionária “Alice”?

Tem que deslocar todos os elementos da lista para inserir o novo elemento na posição correta

Lista ordenada encadeada

- Lista de Gregos, Troianos e Tantálidas



- O que deve acontecer na lista quando
  - Nasce Filoctetes?
  - Nasce Agamêmnon?
  - Morre Menelau?

Lista ordenada encadeada

- Vantagem sobre a alocação seqüencial
  - Os nós podem ser inseridos e eliminados na posição correta, sem realocação dos demais elementos
- Complexidade de algoritmos
  - O que implica?
  - A diferença é significativa?

Lista ordenada estática e encadeada

- **Declaração da estrutura**

```
#define TAM 500
typedef int elem;

typedef struct bloco {
  elem info;
  int prox;
} no;

typedef struct {
  int ini, primeiro_vazio;
  no v[TAM];
} ListaOrd;
```

Lista ordenada dinâmica e encadeada

- **Declaração da estrutura**

```
typedef int elem;

typedef struct bloco {
  elem info;
  struct bloco *prox;
} no;

typedef struct {
  no *ini;
} ListaOrd;
```

Lista ordenada dinâmica e encadeada

```
void cria(ListaOrd *L) {
   L->ini = NULL;
}

void finaliza(ListaOrd *L) {
   no *p = L->ini;

   while (p != NULL) {
        L->ini = L->ini->prox;
        free(p);
        p = L->ini;
   }
}
```

```
void imprimir(ListaOrd *L) {
   no *p = L->ini;

   while (p != NULL) {
        printf("%d ", p->info);
        p = p->prox;
   }
}
```

Operações já vistas em outras implementações, e que não mudam nesse caso

Lista ordenada dinâmica e encadeada

- Exercício: Como verificar se um elemento existe na lista?

```
int esta_na_lista(ListaOrd *L, elem x) {
   no *p = L->ini;

   while ((p != NULL) && (p->info < x))
       p = p->prox;

   if ((p != NULL) && (p->info == x))
       return 1;
   else
       return 0;
}
```

Versão recursiva

```
int esta_na_lista_rec(no *p, elem x) {
   if ((p == NULL) || (p->info > x))
       return 0;
   else {
       if (p->info == x)
             return 1;
       else
             return esta_na_lista_rec(p->prox, x);
   }
}

Chamada inicial: esta_na_lista_rec(L->ini, x);
```

Lista ordenada dinâmica e encadeada

- Exercício: Identifique as possíveis situações que podem ocorrer durante a inserção

```
void inserir(ListaOrd *L, elem x, int *erro) {
   no *p, *ant, *aux;

   aux = (no*) malloc(sizeof(no));
   if (aux == NULL)
        *erro = 1;
   else {
        *erro = 0;
        aux->info = x;
        if ((L->ini == NULL) || (x < L->ini->info)) {
                aux->prox = L->ini;
                L->ini = aux;
        } else {
                ant = NULL;
                p = L->ini;
                while ((p != NULL) && (p->info < x)) {
                        ant = p;
                        p = p->prox;
                }
                aux->prox = p;
                ant->prox = aux;
        }
   }
}
```

```
void inserir(ListaOrd *L, elem x, int *erro) {
    no *p, *ant, *aux;

    aux = (no*) malloc(sizeof(no));
    if (aux == NULL)
        *erro = 1;
    else {
        *erro = 0;
        aux->info = x;
        if ((L->ini == NULL) || (x < L->ini->info)) {
                aux->prox = L->ini;
                L->ini = aux;
        } else {
                ant = NULL;
                p = L->ini;
                while ((p != NULL) && (p->info < x)) {
                        ant = p;
                        p = p->prox;
                }
                aux->prox = p;
                ant->prox = aux;
        }
    }
}
```

Se a lista está vazia, ou se a inserção deve ser feita no início da lista

```
void inserir(ListaOrd *L, elem x, int *erro) {
    no *p, *ant, *aux;

    aux = (no*) malloc(sizeof(no));
    if (aux == NULL)
         *erro = 1;
    else {
         *erro = 0;
         aux->info = x;
         if ((L->ini == NULL) || (x < L->ini->info)) {
                  aux->prox = L->ini;
                  L->ini = aux;
         } else {
                  ant = NULL;
                  p = L->ini;
                  while ((p != NULL) && (p->info < x)) {
                           ant = p;
                           p = p->prox;
                  }
                  aux->prox = p;
                  ant->prox = aux;
         }
    }
}
```

Senão, procurar a posição correta para inserir

Lista ordenada dinâmica e encadeada

- Exercício: Identifique as possíveis situações que podem ocorrer durante a remoção

```
void remover(ListaOrd *L, elem x, int *erro) {
   no *p, *ant;
   if ((L->ini == NULL) || (x < L->ini->info))
        *erro = 1;
   else if (x == L->ini->info) {
        *erro = 0;
        p = L->ini;
        L->ini = L->ini->prox;
        free(p);
   } else {
        ant = NULL;
        p = L->ini;
        while ((p != NULL) && (p->info < x)) {
                ant = p;
                p = p->prox;
        }
        if ((p != NULL) && (p->info == x)) {
                *erro = 0;
                ant->prox = p->prox;
                free(p);
        } else
                *erro = 1;
   }
}
```

```
void remover(ListaOrd *L, elem x, int *erro) {
    no *p, *ant;
    if ((L->ini == NULL) || (x < L->ini->info))
          *erro = 1;
    else if (x == L->ini->info) {
          *erro = 0;
          p = L->ini;
          L->ini = L->ini->prox;
          free(p);
    } else {
          ant = NULL;
          p = L->ini;
          while ((p != NULL) && (p->info < x)) {
                    ant = p;
                    p = p->prox;
          }
          if ((p != NULL) && (p->info == x)) {
                    *erro = 0;
                    ant->prox = p->prox;
                    free(p);
          } else
                    *erro = 1;
    }
}
```

Se a lista estiver vazia, ou se o elemento for menor que o primeiro da lista, indica erro

```
void remover(ListaOrd *L, elem x, int *erro) {
    no *p, *ant;
    if ((L->ini == NULL) || (x < L->ini->info))
          *erro = 1;
    else if (x == L->ini->info) {
          *erro = 0;
          p = L->ini;
          L->ini = L->ini->prox;
          free(p);
    } else {
          ant = NULL;
          p = L->ini;
          while ((p != NULL) && (p->info < x)) {
                    ant = p;
                    p = p->prox;
          }
          if ((p != NULL) && (p->info == x)) {
                    *erro = 0;
                    ant->prox = p->prox;
                    free(p);
          } else
                    *erro = 1;
    }
}
```

Se o elemento for o primeiro da lista, remove-o

```
void remover(ListaOrd *L, elem x, int *erro) {
    no *p, *ant;
    if ((L->ini == NULL) || (x < L->ini->info))
          *erro = 1;
    else if (x == L->ini->info) {
          *erro = 0;
          p = L->ini;
          L->ini = L->ini->prox;
          free(p);
    } else {
          ant = NULL;
          p = L->ini;
          while ((p != NULL) && (p->info < x)) {
                    ant = p;
                    p = p->prox;
          }
          if ((p != NULL) && (p->info == x)) {
                    *erro = 0;
                    ant->prox = p->prox;
                    free(p);
          } else
                    *erro = 1;
    }
}
```

Senão, procura o elemento no restante da lista

```
void remover(ListaOrd *L, elem x, int *erro) {
    no *p, *ant;
    if ((L->ini == NULL) || (x < L->ini->info))
          *erro = 1;
    else if (x == L->ini->info) {
          *erro = 0;
          p = L->ini;
          L->ini = L->ini->prox;
          free(p);
    } else {
          ant = NULL;
          p = L->ini;
          while ((p != NULL) && (p->info < x)) {
                    ant = p;
                    p = p->prox;
          }
          if ((p != NULL) && (p->info == x)) {
                    *erro = 0;
                    ant->prox = p->prox;
                    free(p);
          } else
                    *erro = 1;
    }
}
```

Se encontrá-lo, remove-o; senão, indica erro

Produzindo uma lista ordenada

- Exercício: Implemente uma função que ordene os elementos de uma lista, trocando o conteúdo entre os blocos

Produzindo uma lista ordenada

- Exercício: Implemente uma função que ordene os elementos de uma lista, trocando somente os ponteiros dos blocos

Créditos

- *Material gentilmente cedido pelo Prof. Thiago A. S. Pardo*