AULA 04
ESTRUTURA DE DADOS

Lista linear sequencial (continuação)

Norton T. Roman & Luciano A. Digiampietri

Lista linear sequencial

Na última aula aprendemos listas lineares sequenciais:

- Utilizamos um arranjo para armazenar nossos registros;
- A inserção de registros era feita na posição indicada pelo usuário.

Lista linear sequencial

Na aula de hoje abordaremos dois aspectos:

- Otimização da busca por elementos;
- Mudança na ordem de inserção dos elementos;

Busca por elementos

O usuário diz qual elemento é buscado e a função retorna a posição desse elemento:

- As chaves dos elementos não estão em ordem crescente;
- Se o elemento não existir a função retorna -1;

Busca por elementos (versão inicial)

```
int buscaSequencial(LISTA* l, TIPOCHAVE ch) {
  int i = 0;
  while (i < l->nroElem) {
    if(ch == l->A[i].chave) return i;
    else i++;
  }
  return -1;
}
```

Busca por elementos

Ideia: Ao invés de fazer duas comparações por iteração, seria possível fazer só uma?

- Precisamos sempre comparar a chave do elemento atual com a chave do elemento buscado;
- Mas como garantir que não iremos passar do último elemento?
- Garantindo que a chave buscada será encontrada!

Busca por elementos

Criação de um elemento sentinela:

- Elemento extra (um registro) adicionado à lista para auxiliar alguma operação;
- Será inserido no final da lista (após o último elemento válido) durante as buscas;
- Conterá a chave do elemento buscado.

Busca por elementos (sentinela)

```
int buscaSentinela(LISTA* l, TIPOCHAVE ch) {
  int i = 0;
  l->A[l->nroElem].chave = ch;
  while(l->A[i].chave != ch) i++;
  if (i == l->nroElem) return -1;
  else return i;
}
```

Busca por elementos (sentinela)

Há apenas um probleminha:

- Se a lista já estiver cheia, não haverá espaço para criar o sentinela;
- O que fazer?
- Criamos a lista com uma posição extra (um registro a mais) para garantir que haverá espaço para o sentinela.
- Essa posição extra nunca terá um registro válido.

Modelagem

```
#include <stdio.h>
#define MAX 50
#define ERRO -1
#define true 1
#define false 0

typedef int bool;

typedef int TIPOCHAVE;
```

```
typedef struct{
  TIPOCHAVE chave;
  // outros campos...
} REGISTRO;

typedef struct {
  REGISTRO A[MAX+1];
  int nroElem;
} LISTA;
```

Busca por elementos (sentinela)

```
int buscaSequencial(LISTA* l,
                    TIPOCHAVE ch) {
  int i = 0;
  while (i < l->nroElem) {
    if(ch == l->A[i].chave) return i;
    else i++;
  }
  return -1;
}
```

```
int buscaSentinela(LISTA* l,
                   TIPOCHAVE ch) {
  int i = 0;
  l->A[l->nroElem].chave = ch;
  while(l->A[i].chave != ch) i++;
  if (i == l->nroElem) return -1;
  else return i;
}
```

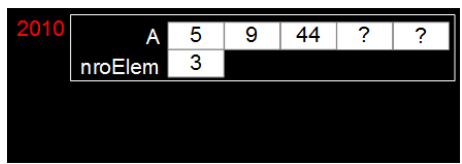
Busca por elementos

Mas a busca binária não é mais eficiente?

- Sim, porém ela necessita que as chaves dos elementos estejam ordenadas;
- Para isso, precisaremos mudar nossa função de inserção de elementos.
- A função de inserção seguirá a lógica do *insertion sort*.

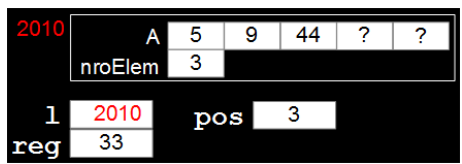
Inserção de elementos - ordenada

```
bool inserirElemListaOrd(LISTA* l, REGISTRO reg) {
  if(l->nroElem >= MAX) return false;
  int pos = l->nroElem;
  while(pos > 0 && l->A[pos-1].chave > reg.chave) {
    l->A[pos] = l->A[pos-1];
    pos--;
  }
  l->A[pos] = reg;
  l->nroElem++;
  return true;
}
```



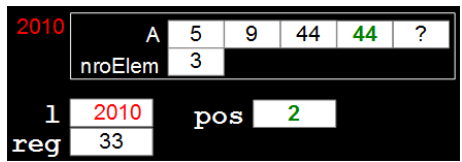
Inserção de elementos - ordenada

```
bool inserirElemListaOrd(LISTA* l, REGISTRO reg) {
  if(l->nroElem >= MAX) return false;
  int pos = l->nroElem;
  while(pos > 0 && l->A[pos-1].chave > reg.chave) {
    l->A[pos] = l->A[pos-1];
    pos--;
  }
  l->A[pos] = reg;
  l->nroElem++;
  return true;
}
```



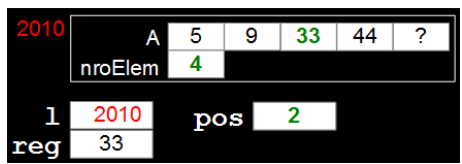
Inserção de elementos - ordenada

```
bool inserirElemListaOrd(LISTA* l, REGISTRO reg) {
  if(l->nroElem >= MAX) return false;
  int pos = l->nroElem;
  while(pos > 0 && l->A[pos-1].chave > reg.chave) {
    l->A[pos] = l->A[pos-1];
    pos--;
  }
  l->A[pos] = reg;
  l->nroElem++;
  return true;
}
```



Inserção de elementos - ordenada

```
bool inserirElemListaOrd(LISTA* l, REGISTRO reg) {
  if(l->nroElem >= MAX) return false;
  int pos = l->nroElem;
  while(pos > 0 && l->A[pos-1].chave > reg.chave) {
    l->A[pos] = l->A[pos-1];
    pos--;
  }
  l->A[pos] = reg;
  l->nroElem++;
  return true;
}
```



Busca binária

```
int buscaBinaria(LISTA* l, TIPOCHAVE ch) {
  int esq, dir, meio;
  esq = 0;
  dir = l->nroElem-1;
  while(esq <= dir) {
    meio = ((esq + dir) / 2);
    if(l->A[meio].chave == ch) return meio;
    else {
      if(l->A[meio].chave < ch) esq = meio + 1;
      else dir = meio - 1;
    }
  }
  return -1;
}
```

Elementos ordenados pelas chaves

Com a ordenação dos elementos pela chave:

- A busca ficou mais eficiente (busca binária);
- Não precisamos do sentinela;
- O que acontece com a exclusão?

Exclusão de elementos

```
bool excluirElemLista(LISTA* l, TIPOCHAVE ch) {
  int pos, j;
  pos = buscaBinaria(l,ch);
  if(pos == -1) return false;
  for(j=pos; j < l->nroElem-1; j++) l->A[j] = l->A[j+1];
  l->nroElem--;
  return true;
}
```

AULA 04
ESTRUTURA DE DADOS

Lista linear sequencial (continuação)

Norton T. Roman & Luciano A. Digiampietri