SCC0202 – Algoritmos e Estruturas de Dados I

Listas Lineares Sequenciais

Prof.: Dr. Rudinei Goularte

(rudinei@icmc.usp.br)

Instituto de Ciências Matemáticas e de Computação - ICMC

Sala 4-229

Conteúdo

- Conceito de Lista Linear
- Operações
- Aplicação
- Casos particulares de listas
- Organização em Memória
- Ordenação
- Organização vs. alocação de memória
- TAD Listas

Conceito de Lista Linear

- Uma Lista Linear é uma sequência de componentes de um mesmo tipo
- É uma sequência de zero ou mais itens $x_1, x_2, \ldots, x_n$ na qual x_i é de um determinado tipo e n representa o tamanho da lista
- Sua principal propriedade estrutural diz respeito às posições relativas dos itens
  - Se $n \geq 1$, x_1 é o primeiro item e x_n é o último (considerando que a indexação inicia a partir de 1)
  - Em geral, x_i precede x_{i+1} para $i = 1, 2, \ldots, n - 1$ e x_i sucede x_{i-1} para $i = 2, 3, \ldots, n$

Operações

- As operações mais frequentes em listas são a busca, a inclusão e a remoção de um determinado elemento
  - Outras operações: alteração de um elemento da lista, combinação de duas ou mais listas lineares em uma única, ordenação dos nós segundo um determinado campo, determinação do primeiro ou último elemento da lista, etc.

Aplicação

- Diversos tipos de aplicações requerem uma lista
  - Informações sobre funcionários de uma empresa
  - Notas de compras
  - Itens de estoque
  - Notas de alunos
  - Lista telefônica
  - Lista de tarefas
  - Gerenciamento de memória
  - Simulações em geral
  - Compiladores, etc.

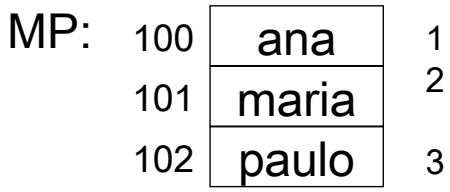
Casos particulares de listas

- **Deque (Double Ended QUEue)**: se as inserções e remoções são permitidas apenas nas extremidades da lista
- **Pilha (stack)**: se as inserções e remoções são realizadas somente em um extremo
- **Fila (queue)**: se as inserções são realizadas em um extremo e remoções em outro

Organização em Memória

- Pode ser classificado de acordo com a posição relativa na memória de dois nós consecutivos na lista
  1. Organização **sequencial** de memória: sucessor lógico de um elemento ocupa posição física consecutiva na memória (endereços consecutivos)

     L = (ana, maria, paulo)

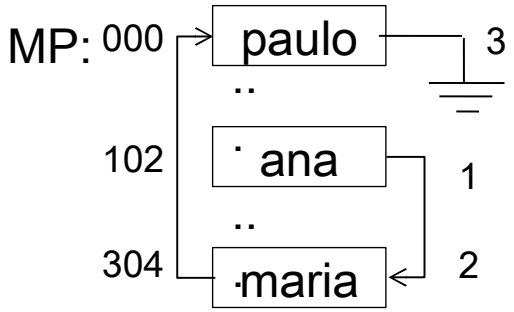


Array

("estática")

  2. Organização **encadeada**: elementos logicamente consecutivos não implicam elementos (endereços) consecutivos na memória

     L = (ana, maria, paulo)



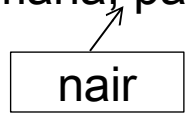
Ponteiros

(dinâmica)

Ordenação

- Qualquer que seja o tipo de organização, a lista pode diferir quanto à ordenação:
  - Ordenada: elementos ordenados segundo valores do campo chave e, eventualmente, de outros campos
    - inserção é feita no local definido pela ordenação

L=(ana, maria, paulo)

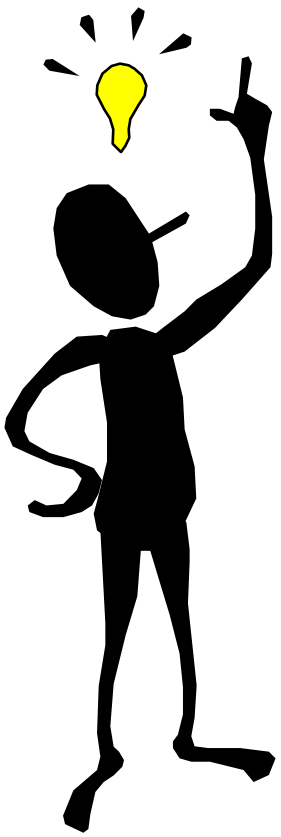


Se Sequencial→deslocamento de registros na MP → > tempo

  - Não ordenada:
    - inserção ocorre nas extremidades (mais barato)

L=(paulo, ana, maria)

Como escolher o tipo de lista?



- *Escolher* entre uma ou outra organização de memória (sequencial ou encadeada) vai depender do comportamento da lista na aplicação (tamanho, operações mais frequentes, etc.).
- A *eficiência* das operações depende da organização usada e de outros fatores: se a lista está ordenada, se é grande ou pequena, etc.

Organização vs. alocação de memória

- Alocação Estática: reserva de memória em tempo de compilação
- Alocação Dinâmica: em tempo de execução

Alocação da memória \ Organização da memória:	Sequencial	Encadeada
Estática		
Dinâmica		

  - Sequencial e estática: Uso de arrays
  - Sequencial e dinâmica: Alocação dinâmica de Array
  - Encadeada e dinâmica: Uso de ponteiros
  - Encadeada e estática: Array simulando encadeamento

TAD Lista

- Vamos definir um TAD com as principais operações sobre uma lista
  - Para simplificar vamos definir apenas as operações principais. Posteriormente, outras operações podem ser definidas
    - Criar lista
    - Inserir item (última posição)
    - Remover item (última posição)
    - Recuperar item (dada uma chave)
    - Contar número de itens
    - Verificar se a lista está vazia
    - Verificar se a lista está cheia
    - Imprimir lista

TAD Lista

- Criar lista
  - Pré-condição: existir espaço na memória
  - Pós-condição: inicia a estrutura de dados

TAD Lista

- Inserir item (última posição)
  - Pré-condição: a lista não está cheia
  - Pós-condição: insere um item na última posição, retorna **true** se a operação foi executada com sucesso, **false** caso contrário

TAD Lista

- Remover item (última posição)
  - Pré-condição: a lista não pode estar vazia
  - Pós-condição: o item na última posição é removido da lista, retorna **true** se a operação foi executada com sucesso, **false** caso contrário

TAD Lista

- Recuperar item (Buscar item)
  - Pré-condição: uma chave x válida é informada
  - Pós-condição: caso obtenha sucesso, retorna o item na lista cuja chave x foi fornecida. Retorna erro caso não encontre a chave x. Se x ocorre mais de uma vez, retorna a primeira ocorrência

TAD Lista

- Verificar se a lista está vazia
  - Pré-condição: nenhuma
  - Pós-condição: retorna **true** se a lista estiver vazia e **false** caso-contrário
- Verificar se a lista está cheia
  - Pré-condição: nenhuma
  - Pós-condição: retorna **true** se a lista estiver cheia e **false** caso-contrário

TAD Lista

- Contar número de itens
  - Pré-condição: nenhuma
  - Pós-condição: retorna o número de itens na lista

- Imprime lista
  - Pré-condição: nenhuma
  - Pós-condição: imprime na tela os itens da lista

TAD Lista - Implementação Sequencial

- Características
  - Os itens da lista são armazenados em posições contíguas de memória
  - A lista pode ser percorrida em qualquer direção
  - A inserção de um novo item pode ser realizada após o último item com custo constante

TAD Lista - Implementação Sequencial

- As escolhas dos tipos de dados a seguir valem tanto para lista ordenada como não ordenada. O que vai diferir no caso de não ordenada são as funções de inserção, busca e remoção.

TAD Lista - Implementação Sequencial

ITEM *lista[n];

`struct`

ITEM =

Pode ser um registro com um número qualquer de campos

TAD Lista - Implementação Sequencial

```
1  #ifndef LISTA_H
2     #define LISTA_H
3
4     #define TAM_MAX 100 /*estimativa do tamanho máximo da lista*/
5     #define inicial 0
6     #define ERRO -32000
7     #define ORDENADA 0  /*0 = lista não ordenada; 1 = lista ordenada*/
8     #include “item.h”
9
10    typedef struct lista_ LISTA;
11
   ...
```

TAD Lista - Implementação Sequencial

```
...
11
12    LISTA *lista_criar(void);
13    bool lista_inserir(LISTA *lista, ITEM *item);
14    bool lista_apagar(LISTA **lista);
15    ITEM *lista_remover(LISTA *lista, int chave);
16    ITEM *lista_busca(LISTA *lista, int chave);
17    int lista_tamanho(LISTA *lista);
18    bool lista_vazia(LISTA *lista);
19    bool lista_cheia(LISTA *lista);
20    void lista_imprimir(LISTA *lista);
21
22 #endif
```

TAD Lista - Implementação Sequencial

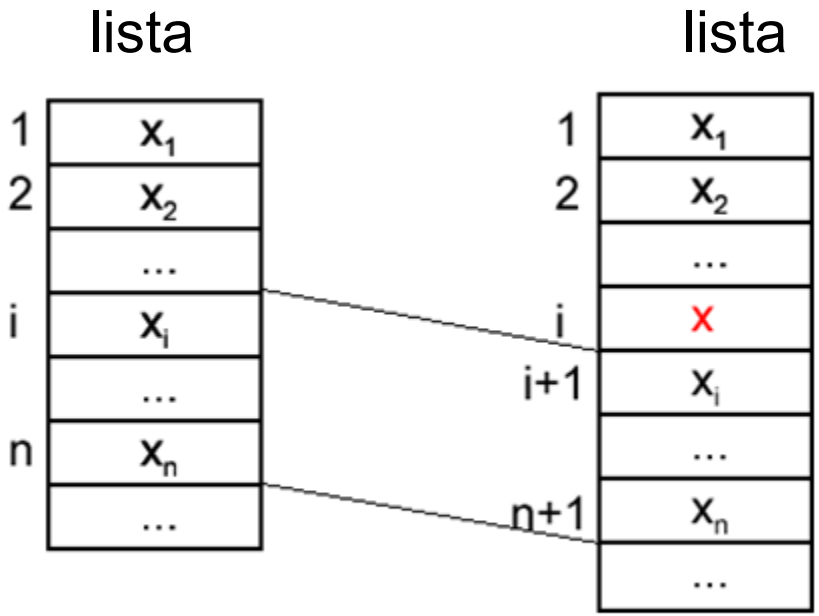
```
1 #include "lista.h"
2 ...
8 struct lista_ {
9         ITEM *lista[TAM_MAX];
10        int inicio;  //inicio da lista
11        int fim;   //fim da lista -1a posicao livre para insercao
12 };
14
15 /*Cria logicamente uma lista, inicialmente vazia*/
16 LISTA *lista_criar(void){
     /*pré-condição: existir espaço na memória*/
     LISTA *lista = (LISTA *) malloc(sizeof(LISTA));
     if (lista != NULL){
        lista->inicio = inicial;
        lista->fim = lista->inicio; /*lista vazia*/
     }
23   return (lista);
24 }
25
```

TAD Lista - Implementação Sequencial

```
26 /*Insere um elemento no fim da fila. LISTA NÃO
ORDENADA.*/
27 bool lista_inserir(LISTA *l, ITEM *item){
28
29  if ((l != NULL) && !lista_cheia(l)){
30     (l->lista[l->fim]) = item;
31     l->fim++;
32     return(true);
33  }
34  return(false);
35 }
38
39 ...
```

TAD Lista - Implementação Sequencial

- Inserção no meio da lista
  - Inserir um item na posição i
  - Problema do deslocamento de itens



Implementação da Inserção no Meio da Lista

```
1 bool lista_inserir_posicao(LISTA *l, int pos, ITEM *item) {
2         int i;
3
4  //verifica se existe espaço e se a posição está na lista
5  if (!lista_cheia(l) && (pos <= l->fim - 1)) {
6    for (i = (l->fim-1); i >= pos; i--) { //move os itens
7       l->lista[i + 1] = l->lista[i];
8    }
9
10   l->lista[pos] = item; //insere novo item
11   l->fim++; //incrementa tamanho
12
13    return (true);
14  } else {
15     return (false);
16  }
17 }
```

(fim – pos)+1 deslocamentos
(cópias de registros!)

Lista Ordenada

- Características
  - Uma lista pode ser mantida em ordem crescente/decrescente segundo o valor de alguma chave
  - Essa ordem facilita a pesquisa de itens
  - Por outro lado, a inserção e remoção são mais complexas pois deve manter os itens ordenados

TAD Lista Ordenada

- Tipo Abstrato de Dados
  - O TAD lista ordenada é o mesmo do TAD lista, apenas difere na implementação
  - As operações diferentes serão a inserção, remoção e busca (recuperação) de itens

TAD Lista Ordenada

- Inserir item
  - Pré-condição: a lista não está cheia
  - Pós-condição: insere um item em uma posição tal que a lista é mantida ordenada

TAD Lista Ordenada

- Inserir item
  - Como manter a ordenação?
    - Buscar por posição na qual se deve inserir
      - Requer uma busca que retorne tal posição
      - A interface do TAD deve ser mantida!
- Utilizar o algoritmo `lista_inserir_posicao`

TAD Lista Ordenada

- Buscar item (ordenadamente)
  - Pré-condição: uma chave x válida é informada
  - Pós-condição: caso obtenha sucesso, retorna o item da lista cuja chave x foi fornecida. Retorna erro caso não encontre a chave x. Se x ocorre mais de uma vez, retorna a primeira ocorrência

Busca em Listas Sequenciais

- Introdução
  - Uma tarefa comum a ser executada sobre listas é a busca de itens dada uma chave
  - No caso da lista não ordenada, a busca será sequencial (consultar todos os elementos)
  - Porém, com uma lista ordenada, diferentes estratégias podem ser aplicadas que aceleram essa busca
    - Busca sequencial “otimizada”
    - Busca binária

Busca Sequencial

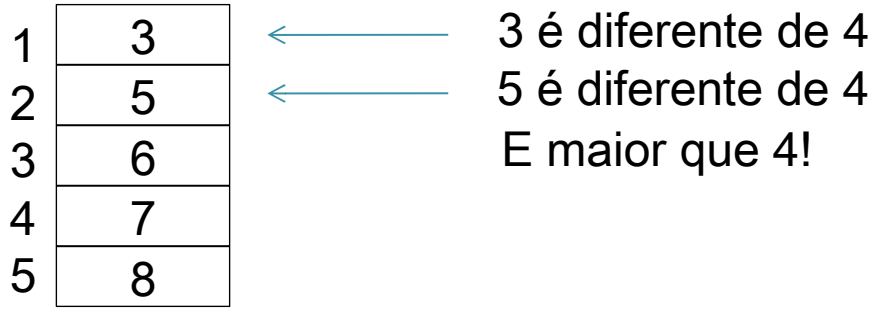
- Na busca sequencial, a ideia é procurar um elemento que tenha uma determinada chave, começando do início da lista, e parar quando a lista terminar ou quando o elemento for encontrado

```
1 int lista_busca_sequencial(LISTA *l, int x) {
2   int i;
3
4   for (i = 0; i <= l->fim; i++) /*não usar 'i < TAM_MAX' !!!*/
5     if (item_get_chave(l->lista[i]) == x)
6         return (i);
7   return (ERRO);
8 }
```

Qual o custo?

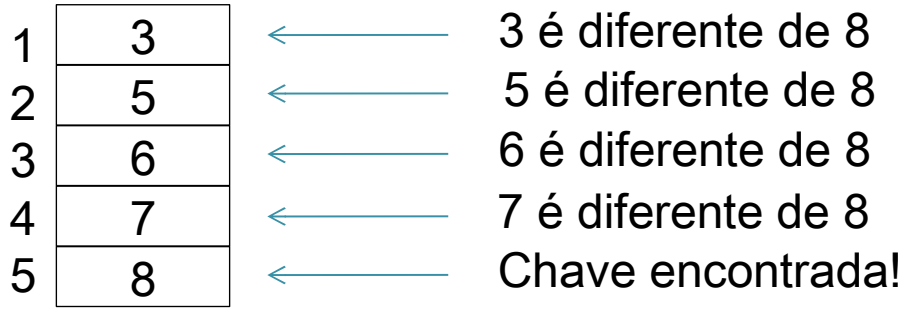
Busca Sequencial (Ordenada)

- Quando os elementos estão ordenados, a busca sequencial pode ser “otimizada” (acelerada)
- Exemplo 1: procurar pelo valor 4



Busca Sequencial (Ordenada)

- Quando os elementos estão ordenados, a busca sequencial pode ser “otimizada” (acelerada)
- Exemplo 2: procurar pelo valor 8



Busca Sequencial (Ordenada)

- A lista ordenada permite realizar buscas sequenciais mais rápidas uma vez que, caso a chave procurada não exista, pode-se parar a busca tão logo se encontre um elemento com chave maior que a procurada

```
1 int lista_busca_ordenada(LISTA *l, int x) {
2   int i=0;
3
4   if (l != NULL && !lista_vazia(l)){
5     while (item_get_chave(l->lista[i]) < x){
6       i++;
7       if (i >= l->fim)
8           break;
9     }
10  }
11  return (i); //retorna posição da chave ou posição onde a chave deveria estar
12 }
```

Qual o custo?

Busca Sequencial (Ordenada)

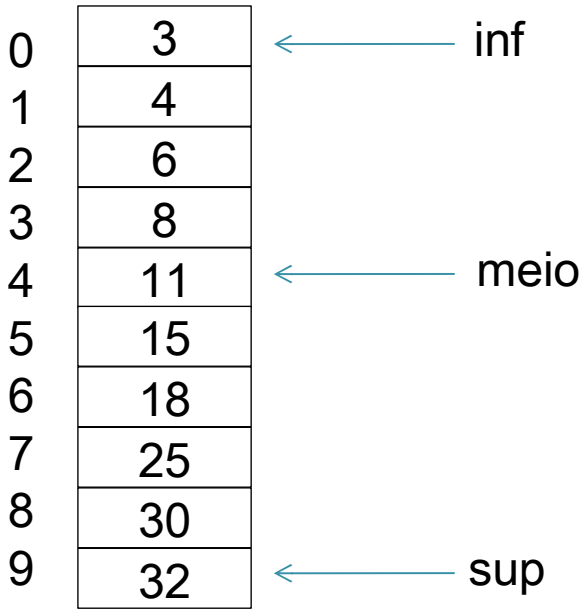
- Entretanto, essa melhoria não altera a complexidade da busca sequencial - ainda é *O(n).* Porque?

Busca Binária

- A busca binária é um algoritmo de busca mais sofisticado e bem mais eficiente que a busca sequencial
- Entretanto, a busca binária somente pode ser aplicada em estruturas que permitem acessar cada elemento em tempo constante, tais como os vetores
- A ideia é, a cada iteração, dividir o vetor ao meio e descartar metade do vetor

Busca Binária

- Exemplo 1
  - Para procurar pelo valor 30

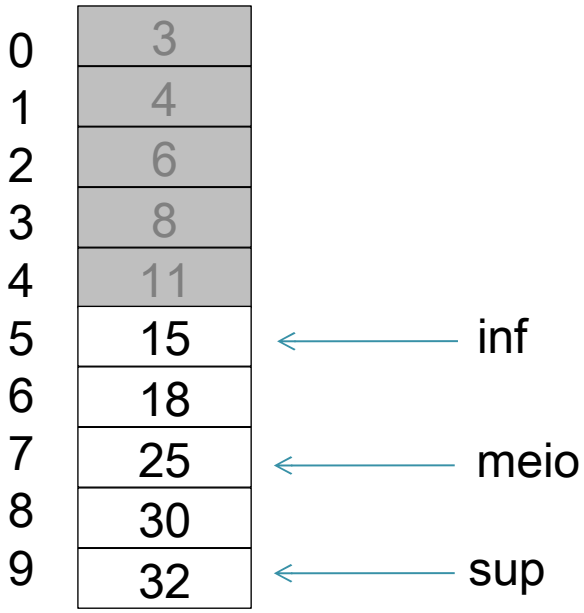


meio = (inf+ sup) /2
9/2 = 4

lista[meio] < 30
inf = meio +1

Busca Binária

- Exemplo 1
  - Para procurar pelo valor 30



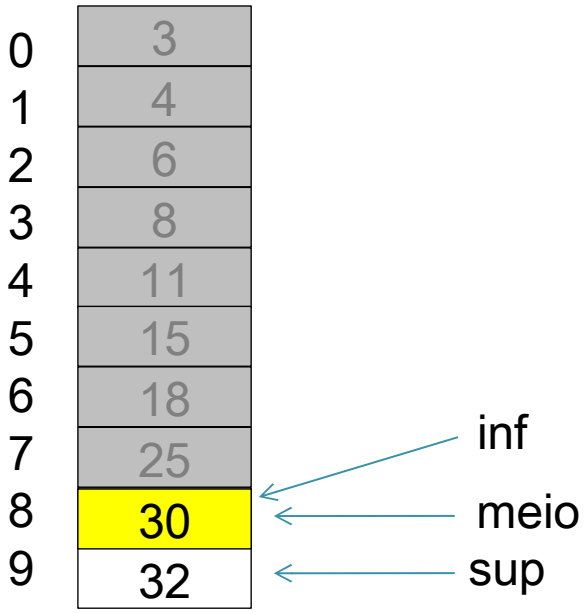
lista[meio] < 30
Inf = meio +1

meio = (inf+ sup) /2
14/2 = 7

lista[meio] < 30
inf= meio +1

Busca Binária

- Exemplo 1
  - Para procurar pelo valor 30



lista[meio] < 30
inf= meio +1

meio = (inf+ sup) /2
17/2 = 8

lista[meio] igual a 30

Chave Encontrada

Busca Binária

- É importante lembrar que
  - Busca binária somente funciona em vetores ordenados
  - Busca sequencial funciona com vetores ordenados ou não
- A busca binária é muito eficiente. Ela é $O(log_2 n)$. Para $n = 1.000.000$, aproximadamente 20 comparações são necessárias

Busca Binária

- Sugestão
  - Tentem implementar a Busca Binária para o TAD Lista Sequencial.

E a Remoção?

- TAD Lista Ordenada:
- Remover item (por chave)
  - Pré-condição: uma chave válida de item é informada; a lista não está vazia
  - Pós-condição: o item com a chave fornecida é removido da lista, a lista é mantida ordenada

E a Remoção?

- Em Listas não ordenadas
  - Remover do fim
    - Custo constante. Simples!
  - Remover do início
    - Requer deslocamentos.
  - Remover do meio
    - Item qualquer, desde que esteja no meio exato da lista?
      - Requer deslocamentos
    - Item com chave específica, que pode estar em qualquer posição da lista?
      - Requer uma busca
      - Pode requerer deslocamentos (se o item não estiver no fim)

E a Remoção?

- Em Listas ordenadas: remoção não altera ordenação
  - Remover do fim
    - Custo constante. Simples!
  - Remover do início
    - Requer deslocamentos.
  - Remover do meio
    - Item qualquer, desde que esteja no meio exato da lista?
      - Requer deslocamentos
    - Item com chave específica, que pode estar em qualquer posição da lista?
      - Requer uma busca
      - Pode requerer deslocamentos (se o item não estiver no fim)

Conclusão Lista Linear Sequencial

- Pontos Fortes
  - Tempo constante de acesso aos dados
  - Simplicidade
- Pontos Fracos
  - Custo para inserir e retirar elementos da lista dada uma posição
    - O problema dos deslocamentos
  - Lista ordenada:
    - Busca é facilitada
    - Inserções e remoções implicam em deslocamentos
  - Tamanho máximo da lista é (dependendo da linguagem) definido em tempo de compilação

Quando utilizar...

- Essa implementação simples é mais comumente utilizada em certas situações:
  - Listas pequenas
  - Tamanho máximo da lista é conhecido
  - Poucas ocorrências de utilização dos métodos de inserção e remoção.

Exercício de Prova

- Implemente uma função de Busca em lista sequencial ordenada que retorne a posição do item buscado na lista. Exemplo, seja a lista: L = [1, 4, 5, 9, 15]. Considere que L é um vetor de itens tal qual definimos em nossas aulas para o TAD Lista Sequencial e que a chave do item buscado é 5. A função retornaria 2.

Exercício de Prova 2

- Implemente uma função para Remover um item (dada uma chave) em lista sequencial ordenada do modo mais eficiente possível.

Exercício de Prova 3

- Implemente uma função para Inserir um item em lista sequencial ordenada. A interface definida para o TAD Lista não pode ser alterada.

Bibliografia

- Livro texto do Ziviani (Projeto de Algoritmos. Nívio Ziviani, 2004).
  - Qualquer um dos livros-texto.