Universidade do Estado do Amazonas
Escola Superior de Tecnologia - EST
Núcleo de Computação

Algoritmos e Estruturas de Dados I

Listas com implementação estática

Prof. Flávio José M. Coelho

fcoelho@uea.edu.br

Objetivos

1. Entender o que é uma **lista estática**.
2. Entender o **TAD lista estática** e suas operações.
3. Entender a **implementação** do TAD lista.

Coisas a fazer hoje

- Tomar café no Everest com a Louis. ✓
- Ir para o Planeta Diário.
- Evitar ataque terrorista nuclear. ✓
- Evitar invasão alien (desistiram). ✗
- Dar uma surra no Batman.

Relembrando...

Conjuntos manipulados por algoritmos são denominados **conjuntos dinâmicos**: podem sofrer mudanças após certas operações.

Operações básicas de um conjunto dinâmico

$\textsc{Cria}(S)$

Cria o conjunto S vazio.

Operações básicas de um conjunto dinâmico

$$\textsc{Busca}(k, S)$$

Consulta que retorna um ponteiro para o item em S, tal que $x.chave = k$, ou NIL se x não pertence à S.

Operações básicas de um conjunto dinâmico

$\textsc{Insere}(x, S)$

Operação de modificação que aumenta o conjunto S com o item apontado por x. Os atributos de x já estão inicializados.

Operações básicas de um conjunto dinâmico

$$\textsc{Remove}(x, S)$$

Operação de modificação que, dado um ponteiro x para um elemento em S, remove x de S.

Operações básicas de um conjunto dinâmico

$\textsc{Minimo}(S)$

Consulta sobre um conjunto totalmente ordenado S que retorna um ponteiro para o elemento de S com a chave mínima.

Operações básicas de um conjunto dinâmico

$\textsc{Maximo}(S)$

Consulta sobre um conjunto totalmente ordenado S que retorna um ponteiro para o elemento de S com a chave máxima.

Operações básicas de um conjunto dinâmico

$\textsc{Sucessor}(x, S)$

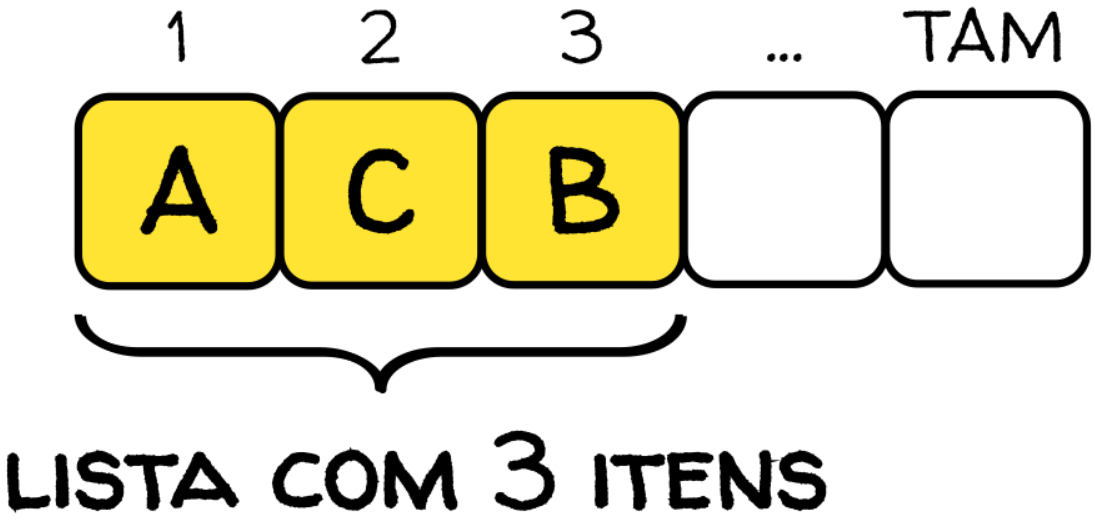
Consulta que, dado um elemento x cuja chave está em um conjunto totalmente ordenado S, retorna um ponteiro para o próximo maior elemento em S, ou NIL se x for o elemento máximo.

Operações básicas de um conjunto dinâmico

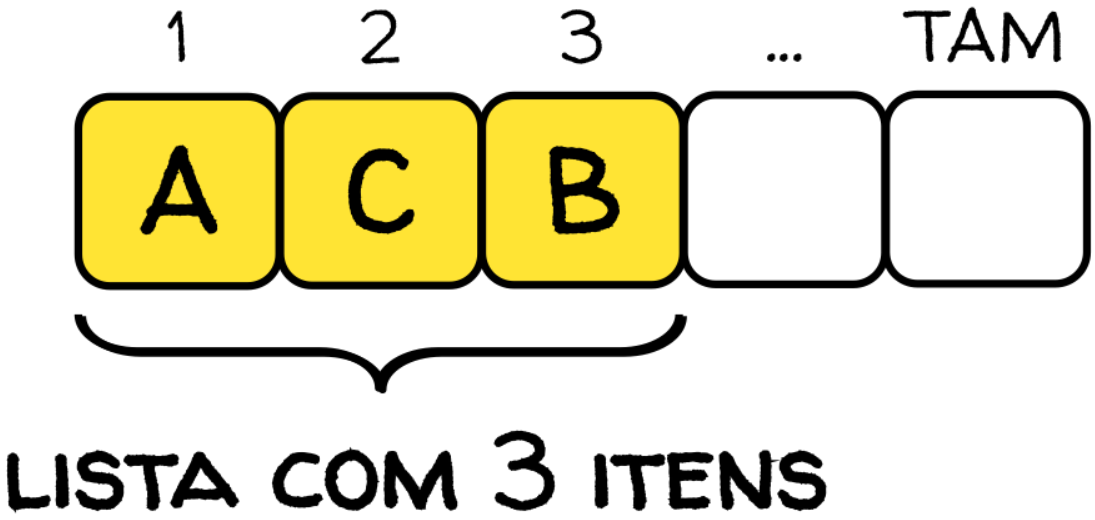
$\textsc{Predecessor}(x, S)$

Consulta que, dado um elemento x cuja chave está em um conjunto totalmente ordenado S, retorna um ponteiro para o próximo menor elemento em S, ou NIL se x for o elemento mínimo.

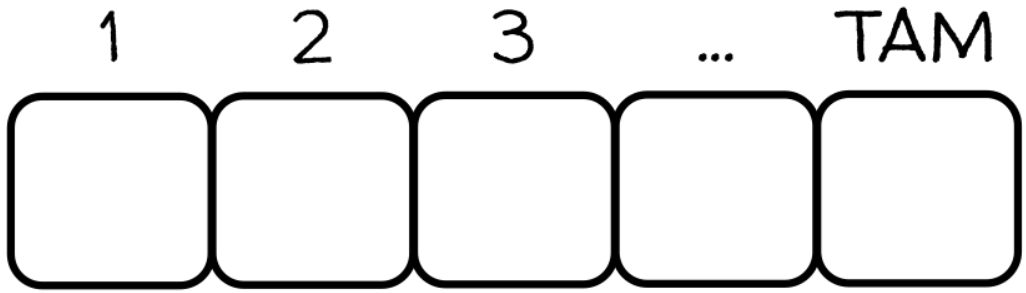
Listas estáticas



Uma **lista estática** é implementada por um **vetor** $S[1..\text{TAM}]$, onde **TAM** é uma constante inteira, não negativa.

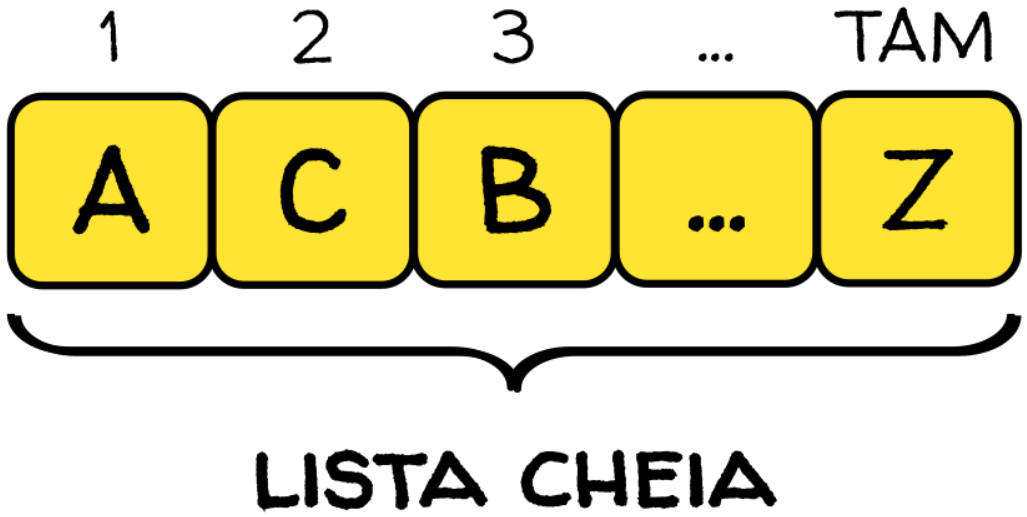


A lista é **dinâmica**; seu tamanho $0 \leq n \leq$TAM varia com o tempo. Estática é a estrutura de dados subjacente da lista - o vetor.



LISTA VAZIA

Quando $n = 0$ a lista está vazia.



Quando $n =$ TAM a lista está cheia.

TAD Lista

Uma **lista** é composta por um conjunto de **itens**.

Vamos criar uma **Lista de Tarefas** (*To-Do list*).

Neste caso, as **tarefas** são os itens da lista.

Vamos dividir o código em três arquivos:

- **arquivo de cabeçalho Lista.h**: conterá, includes de bibliotecas, constantes, definições de estruturas de dados (typedefs, enums, structs, classes, etc.) e assinatura de métodos/funções.

Vamos dividir o código em três arquivos:

- **arquivo de implementação Lista.cpp**: conterá o código de cada método/função definido no Lista.h. Precisaremos fazer um include do Lista.h.

Vamos dividir o código em três arquivos:

- **arquivo Main.cpp**: contém a função main, o include do arquivo Lista.h, outros includes necessários, e a implementação de funções auxiliares.

No ANSI C...

Arquivo Lista.h

```
typedef string Chave;
typedef string Hora;
// Item: tarefa
struct Tarefa {
    Chave desc; // descricao
    Hora hora;
    bool feita;
};
```

No ANSI C...

Arquivo Lista.h

```
struct Lista {
    Tarefa *tarefa; // vetor de itens
    int TAM; // tamanho do vetor
    int n; // tamanho atual da lista
};

// Assinaturas das operações da Lista
```

No ANSI C...

Arquivo Lista.cpp

```
#include "Lista.h"

// Implementa operação Cria
// Implementa operação Busca, etc.
```

No C++...

Arquivo Lista.h

```
typedef string Chave;
typedef string Hora;
// Item: tarefa
class Tarefa {
private:
    Chave desc; // descricao
    Hora hora;
    bool feita;
    // métodos básicos };
```

No C++...

Arquivo Lista.h

```
class Lista {
private:
    Tarefa *tarefa; // vetor de itens
    int TAM; // tamanho do vetor
    int n; // tamanho atual da lista
public:
// definições de métodos da lista
};
```

No C++...

Arquivo Lista.cpp

```
#include "Lista.h"

// Implementa métodos classe Tarefa
// Implementa métodos classe Lista
```

No C++...

Arquivo Main.cpp

```
#include "Lista.h"

// Outros includes necessários
// Implementa funções auxiliares
// Implementa main.
```

Operações do TAD

CRIA(S)

1 Aloca TAM células de memória para S

2 $S.n = 0$

```
BUSCA(k, S)
1    S[0].chave = k // sentinela
2    i = S.n
3    enquanto k != s[i].chave
4        i = i - 1
5    se i > 0 retorne S[i]
6    senão retorne NIL
```

```
INSERE(x, S)
1    se S.n < TAM
2        S.n = S.n + 1
3        S[S.n] = x
4    senão
5        lista cheia
```

```
REMOVE(pos, S)
1    se S.n > 0 e 1 ≤ pos ≤ S.n
2        para i = pos até S.n − 1
3            S[i] = S[i + 1]
4        S.n = S.n − 1
5    senão
6        lista vazia ou posição inválida
```

```
MINIMO(S)
1    min = 1
2    para i = 1 até S.n
3        se S[i].chave < S[min].chave
4            min = i
5    retorne S[min]
```

Conclusão

#1

Busca tem eficiência $O(n)$. Se a lista for ordenada, podemos usar **busca binária** que tem eficiência $O(\log n)$ (mais o custo da ordenação).

Conclusão

#2

Inserção no fim consome $O(1)$. Se for em posição arbitrária, o deslocamento, no pior caso, consome $O(n)$ passos.

Conclusão

#3

Remoção em posição arbitrária consome $O(n)$ (pior caso).

Conclusão

#4

Lista estática tem implementação **simples**, **eficiente** mas tem tamanho **fixo**. E se fosse preciso incluir mais itens?

Referências

- T. H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein, Introduction to Algorithms, 3rd edition, MIT Press, 2010
- N. Ziviani. Projeto de Algoritmos com Implementação em Pascal C. Cengage Learning, 2012.

Onde obter este material:

est.uea.edu.br/fcoelho