Listas dinâmicas

Conceitos e implementação

Prof. Marcelo de Souza

45EST – Algoritmos e Estruturas de Dados
Universidade do Estado de Santa Catarina

Material de apoio

Leitura principal:

- Capítulo 7 de Goodrich et al. (2014)[1] – Listas e iteradores.

Leitura complementar:

- Capítulo 5 de Lafore e Machado (2004)[2] – Listas encadeadas.
- Capítulo 7 de Pereira (2008)[3] – Listas encadeadas.

[1] Michael T Goodrich et al. (2014). *Data structures and algorithms in Java*. 6ª ed. John Wiley & Sons.

[2] Robert Lafore e Eveline Vieira Machado (2004). *Estruturas de dados & Algoritmos em Java*. Ciência Moderna.

[3] Silvio do Lago Pereira (2008). *Estruturas de Dados Fundamentais: Conceitos e Aplicações*.

Listas dinâmicas

Ideia geral

Uma **lista dinâmica** permite operações (inserção e remoção) em posições arbitrárias.

- i.e., não somente nas extremidades, mas também no meio da estrutura.

Listas dinâmicas

Ideia geral

Uma **lista dinâmica** permite operações (inserção e remoção) em posições arbitrárias.

- i.e., não somente nas extremidades, mas também no meio da estrutura.

Operações:

- `get(i)`: retorna o `i`-ésimo elemento da lista.
- `set(i, e)`: substitui o `i`-ésimo elemento por `e`.
- `add(i, e)`: adiciona `e` na posição `i`, alterando a posição dos elementos seguintes.
- `remove(i)`: remove o `i`-ésimo elemento da lista.

Listas dinâmicas

Ideia geral

Uma **lista dinâmica** permite operações (inserção e remoção) em posições arbitrárias.

- i.e., não somente nas extremidades, mas também no meio da estrutura.

Operações:

- `get(i)`: retorna o `i`-ésimo elemento da lista.
- `set(i, e)`: substitui o `i`-ésimo elemento por `e`.
- `add(i, e)`: adiciona `e` na posição `i`, alterando a posição dos elementos seguintes.
- `remove(i)`: remove o `i`-ésimo elemento da lista.

Aplicações:

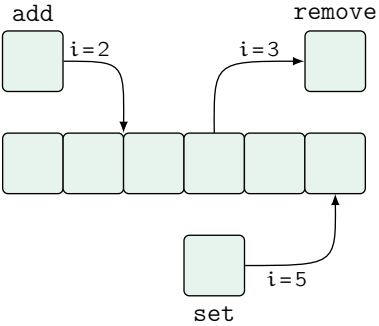
- Listas com ordenação (e.g., inserção ordenada).
- Listagem de arquivos em um diretório.

Listas dinâmicas

Exemplo de funcionamento

Seja uma lista dinâmica implementada usando uma lista sequencial simples, inicialmente vazia.

Método	**Retorno**	**Conteúdo da lista**
`add(0, A)`	–	(A)
`add(0, B)`	–	(B, A)
`get(1)`	A	(B, A)
`set(2, C)`	*erro*	(B, A)
`add(2, C)`	–	(B, A, C)
`add(4, D)`	*erro*	(B, A, C)
`remove(1)`	A	(B, C)
`add(1, D)`	–	(B, D, C)
`add(1, E)`	–	(B, E, D, C)
`get(4)`	*erro*	(B, E, D, C)
`add(4, F)`	–	(B, E, D, C, F)
`set(2, G)`	D	(B, E, G, C, F)
`get(2)`	G	(B, E, G, C, F)



(representação gráfica)

Listas dinâmicas

Implementação (interface)

Interface `List`:

```
1 public interface List<E> {
2   int size();
3   boolean isEmpty();
4   E get(int i);
5   E set(int i, E e);
6   void add(int i, E e);
7   E remove(int i);
8 }
```

Usamos uma interface para definir nosso TAD lista. A interface só define a API (métodos), a qual pode ser implementada de diversas formas (e usando diferentes estruturas).

Listas dinâmicas

Implementação usando arranjos

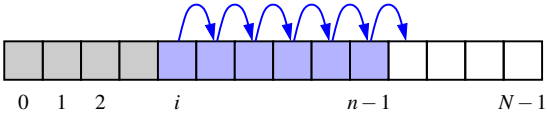
Problema: realocação de elementos nas operações.

Listas dinâmicas

Implementação usando arranjos

Problema: realocação de elementos nas operações.

- Na **inserção**, os elementos seguintes são movidos pra **trás**.

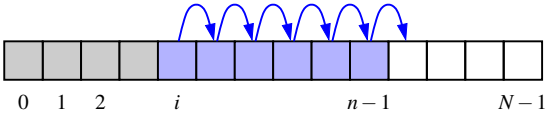


Listas dinâmicas

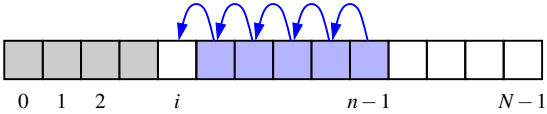
Implementação usando arranjos

Problema: realocação de elementos nas operações.

- Na **inserção**, os elementos seguintes são movidos pra **trás**.



- Na **remoção**, os elementos seguintes são movidos pra **frente**.



Listas dinâmicas

Implementação usando arranjos

Problema: capacidade limitada devido à estrutura fixa do arranjo.

Listas dinâmicas

Implementação usando arranjos

Problema: capacidade limitada devido à estrutura fixa do arranjo.

Solução:

- Mantém uma lista de **tamanho fixo**.
- Quando precisar de mais espaço, **dobra o tamanho** da lista.
  - Cria uma nova lista com o dobro de capacidade.
  - Copia os elementos para a nova lista.
  - Atualiza a referência.

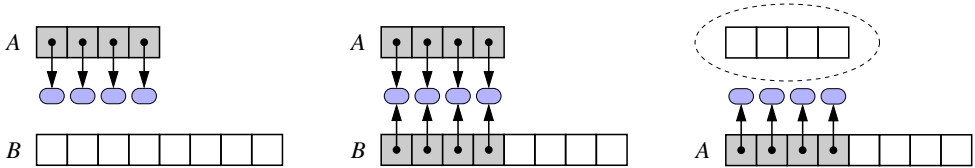
Listas dinâmicas

Implementação usando arranjos

Problema: capacidade limitada devido à estrutura fixa do arranjo.

Solução:

- Mantém uma lista de **tamanho fixo**.
- Quando precisar de mais espaço, **dobra o tamanho** da lista.
  - Cria uma nova lista com o dobro de capacidade.
  - Copia os elementos para a nova lista.
  - Atualiza a referência.



Listas dinâmicas

Implementação usando arranjos

```java
public class ArrayList<E> implements List<E> {
  public static final int CAPACITY = 16;
  private E[] data;
  private int size = 0;

  public ArrayList() { this(CAPACITY); }

  public ArrayList(int capacity) {
    data = (E[]) new Object[capacity];
  }

  public int size() { return size; }
  public boolean isEmpty() { return size == 0; }

  //...
}
```

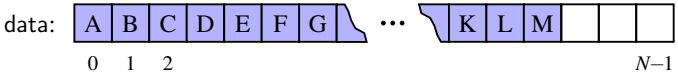
Listas dinâmicas

Implementação usando arranjos

```java
public class ArrayList<E> implements List<E> {
  public static final int CAPACITY = 16;
  private E[] data;
  private int size = 0;

  public ArrayList() { this(CAPACITY); }

  public ArrayList(int capacity) {
    data = (E[]) new Object[capacity];
  }

  public int size() { return size; }
  public boolean isEmpty() { return size == 0; }

  //...
}
```

Detalhes:

- `ArrayList` implementa a interface `List`.
- O *array* possui estrutura fixa (capacidade definida), apesar de usarmos a estratégia apresentada para contornar essa limitação.
- Não há referência para a "extremidade" da estrutura, pois os dados sempre estarão armazenados a partir do início do *array*. Logo, o último elemento estará em `size - 1`.



Listas dinâmicas

Implementação usando arranjos

Método `checkIndex`:

```
protected void checkIndex(int i, int n) {
  if (i < 0 || i >= n)
    throw new IndexOutOfBoundsException("Illegal index: " + i);
}
```

- Para se certificar de não acessar uma posição inválida.

Listas dinâmicas

Implementação usando arranjos

Método `checkIndex`:

```
protected void checkIndex(int i, int n) {
  if (i < 0 || i >= n)
    throw new IndexOutOfBoundsException("Illegal index: " + i);
}
```

- Para se certificar de não acessar uma posição inválida.

Método `resize`:

```
protected void resize(int capacity) {
  E[] temp = (E[]) new Object[capacity];
  for (int k=0; k < size; k++)
    temp[k] = data[k];
  data = temp;
}
```

- Para implementar tamanho dinâmico.

Listas dinâmicas

Implementação usando arranjos

Método `get`:

```
1 public E get(int i) {
2   checkIndex(i, size);
3   return data[i];
4 }
```

Listas dinâmicas

Implementação usando arranjos

Método get:

```
1 public E get(int i) {
2   checkIndex(i, size);
3   return data[i];
4 }
```

Método set:

```
1 public E set(int i, E e) {
2   checkIndex(i, size);
3   E temp = data[i];
4   data[i] = e;
5   return temp;
6 }
```

Listas dinâmicas

Implementação usando arranjos

Método get:

```
1 public E get(int i) {
2   checkIndex(i, size);
3   return data[i];
4 }
```

Método set:

```
1 public E set(int i, E e) {
2   checkIndex(i, size);
3   E temp = data[i];
4   data[i] = e;
5   return temp;
6 }
```

- Tal como feito no método get, a posição a ser acessada é verificada.
- O elemento substituído é retornado.

Listas dinâmicas

Implementação usando arranjos

Métodos add:

```java
public void add(int i, E e) {
  checkIndex(i, size + 1);
  if (size == data.length)
    resize(2 * data.length);
  for (int k = size - 1; k >= i; k--)
    data[k + 1] = data[k];
  data[i] = e;
  size++;
}

public void add(E e) {
  add(size, e);
}
```

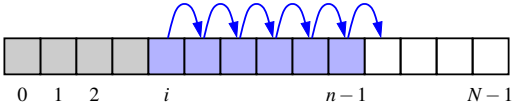
Listas dinâmicas

Implementação usando arranjos

Métodos `add`:

```
1  public void add(int i, E e) {
2    checkIndex(i, size + 1);
3    if (size == data.length)
4      resize(2 * data.length);
5    for (int k = size - 1; k >= i; k--)
6      data[k + 1] = data[k];
7    data[i] = e;
8    size++;
9  }
10
11 public void add(E e) {
12   add(size, e);
13 }
```

- A posição de inserção deve estar entre 0 e `size + 1`.
- Se o limite do *array* foi atingido, sua capacidade é dobrada (linhas 3 e 4).
- As linhas 5 e 6 fazem a realocação dos elementos da posição `i` em diante.
- O método `add` sem posição definida insere o elemento no final da lista (i.e., no índice `size`).



Listas dinâmicas

Implementação usando arranjos

Método `remove`:

```
public E remove(int i) {
  checkIndex(i, size);
  E temp = data[i];
  for (int k=i; k < size-1; k++)
    data[k] = data[k+1];
  data[size-1] = null;
  size--;
  return temp;
}
```

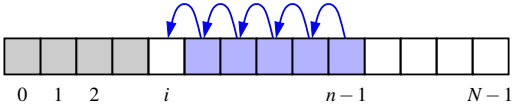
Listas dinâmicas

Implementação usando arranjos

Método `remove`:

```
1 public E remove(int i) {
2   checkIndex(i, size);
3   E temp = data[i];
4   for (int k=i; k < size-1; k++)
5     data[k] = data[k+1];
6   data[size-1] = null;
7   size--;
8   return temp;
9 }
```

- A posição de remoção deve estar entre 0 e `size`.
- As linhas 4 e 5 fazem a realocação dos elementos para cobrir o espaço deixado pela remoção.
- A linha 6 limpa a antiga última posição.
- O elemento removido é retornado.



45EST – Algoritmos e Estruturas de Dados
Prof. Marcelo de Souza