Universidade Federal de São João del Rei
Departamento de Ciência da Computação
Curso de Ciência da Computação

Roteiro 4

Rodrigo José Zonzin
212050002

1 Lista Sequencial Estática

1.1

Implementado.

1.2

Operação Procura

```
1   int Procura(Lista *li, int x){
2     for(int i = 0; i<= li->qtd; i++){
3       if(li->dados[i] == x) return i;
4     }
5
6     return -1;
7   }
```

1.3

Inserção ordenada.

```
1  int novoInsere(Lista* li, int elem){
2    if(li == NULL) return 0;
3    if(!listaCheia(li)){
4      li->dados[li->qtd] = elem;
5      li->qtd++;
6
7      qsort(li->dados, li->qtd, sizeof(int), comparador);
8      return 1;
9    }
10   else return 0;
11 }
```

1.4

Remoção de um elemento $x \in$ *Lista*

```
1   void removeLi(Lista *li, int item) {
2     for(int i = 0; i < li->qtd; i++){
3       if(li->dados[i] == item){
4         if (i < li->qtd - 1){
5           for (int j = i; j < li->qtd - 1; j++){
6             li->dados[j] = li->dados[j + 1];
7           }
8         }
9         li->qtd--;
10        return;
11      }
12    }
13  }
```

2 Lista Simplesmente Encadeada

2.1

O TAD foi reimplementado. O desenho rastreio não foi incluído já que para este autor ele é uma instância pertencente ao reino das ideias, na analogia platônica.

2.2

Tamanho e Procura

```
1   int tamanhoLi(Lista *li){
2     struct NO *corrente = (*li)->prox;
3     if(corrente == NULL) return 0;
4
5     int tam = 1;
6     while(corrente != NULL){
7       tam++;
8       corrente = corrente->prox;
9     }
10    return tam;
11  }
12
13  int procuraLi(Lista *li, int x){
14    struct NO *corrente = (*li)->prox;
15    if(corrente == NULL) return -1;
16
17    int i = 1;
18    while(corrente != NULL){
19      if(corrente->info == x) return i;
20      corrente = corrente->prox;
21      i++;
22    }
23    return -1;
24  }
```

2.3

Inserção ordenada

```
1   void insereOrdenado(Lista* lista, int valor){
2     NO* novo = alocarNO();
3
4     novo->info = valor;
5     novo->prox = NULL;
6
7     NO* atual = *lista;
8     NO* anterior = NULL;
9
10    while(atual != NULL && atual->info < valor){
11      anterior = atual;
12      atual = atual->prox;
13    }
14
15    if(anterior == NULL){
16      novo->prox = *lista;
17      *lista = novo;
18    }
19
20    else{
21      anterior->prox = novo;
22      novo->prox = atual;
23    }
24  }
```

2.4

Remoção de um elemento.

```
1   void removerElemento(Lista* lista, int valor) {
2     NO* atual = *lista;
3     NO* anterior = NULL;
4
5     while(atual != NULL && atual->info != valor){
6       anterior = atual;
7       atual = atual->prox;
8     }
9
10    if(atual == NULL) return;
11
12    if(anterior == NULL) *lista = atual->prox;
13    else anterior->prox = atual->prox;
14
15    free(atual);
16  }
```

3 Lista Duplamente Encadeada

3.1

Mesma situação da seção 2.1.

3.2

Tamanho e Procura

```
int tamanho(Lista* lista){
  int tamanho = 0;
  NO* atual = *lista;

  while(atual != NULL){
    tamanho++;
    atual = atual->prox;
  }

  return tamanho;
}

int procura(Lista* lista, int x){
  NO* atual = *lista;

  while(atual != NULL){
    if (atual->info == x){
      return 1;
    }
    atual = atual->prox;
  }

  return 0;
}
```

3.3

Inserção ordenada

```
void inserirOrdenado(Lista* lista, int x){
  NO* novoNo = alocarNO();
  novoNo->info = x;

  if(*lista == NULL){
    novoNo->prox = NULL;
    novoNo->ant = NULL;
    *lista = novoNo;
  }
  else if(x <= (*lista)->info){
    novoNo->prox = *lista;
    novoNo->ant = NULL;
    (*lista)->ant = novoNo;
    *lista = novoNo;
  }
  else{
    NO* atual = *lista;

    while(atual->prox != NULL && atual->prox->info < x){
      atual = atual->prox;
    }

    novoNo->prox = atual->prox;
    novoNo->ant = atual;

```

```
26        if(atual->prox != NULL){
27          atual->prox->ant = novoNo;
28        }
29
30        atual->prox = novoNo;
31      }
32    }
```

3.4

Remoção

```
1   void removerElemento(Lista* lista, int valor) {
2     if (*lista == NULL) return;
3
4     NO* atual = *lista;
5
6     if(atual->info == valor){
7       *lista = atual->prox;
8       if(*lista != NULL){
9         (*lista)->ant = NULL;
10      }
11      free(atual);
12      return;
13    }
14
15    while(atual != NULL && atual->info != valor){
16      atual = atual->prox;
17    }
18
19    if(atual == NULL) return;
20    if(atual->ant != NULL) atual->ant->prox = atual->prox;
21    if(atual->prox != NULL) atual->prox->ant = atual->ant;
22
23    free(atual);
24  }
```

4 Lista Circular Simplesmente Encadeada

4.1

Desenho mental.

4.2

Tamanho e Procura.

```
1   int tamanho(Lista* li){
2     if(li == NULL || *li == NULL) return 0;
3
4     int tamanho = 0;
5     NO* aux = *li;
6
7     do{
8       tamanho++;
```

```
9         aux = aux->prox;
10      }while (aux != *li);
11
12      return tamanho;
13    }
14
15    int procura(Lista* li, int elem){
16      if(li == NULL || *li == NULL) return -1;
17
18      int posicao = 0;
19      NO* aux = *li;
20
21      do{
22        if(aux->info == elem) return posicao;
23
24        posicao++;
25        aux = aux->prox;
26      }while(aux != *li);
27
28      return -1;
29    }
```

4.3 Código

https://github.com/RodrigoZonzin/labaeds2