Lista P2

1)

{Enquanto o índice for menor que o limite lógico}

{Enquanto elem. já ordenado for maior que elemento a ordenar e existir elem. já ordenado a ser considerado}

Inicializar índice

ordenar por inserçao

armazenar indice maior elem. já ordenado

ordenar

seta elem. a ser reposicionado

reposiciona elem.

atualiza quantos foram ordenados

encontra pos. correta do elem.

reposiciona elem. já ordenado maior que o a ordenar

diminui qtd. de elem. já ordenados a serem considerados

Conjunto solução de ordenar por inserção: inicializar índice; ordenar.

Componente abstrato: ordenar

Componente concreto: inicializar índice

2) e 3)

**AE**

busca\_bin(Lista \* lst, void \*valor)

{

inf = ObtemLimInf(lst);

AI1

sup = ObtemLimSup(lst);

AI2

while(inf<=sup)

{

meio = (inf+sup)/2;

AI3

comp = comparar(valor,ObterValor(meio));

AI4

if(comp == IGUAL)

{

CNT\_Conta( "achou elemento" );

return 1;

}

AI5

if(comp == MENOR)

{

CNT\_Conta( "comp menor" );

sup = meio-1;

}

else

{

CNT\_Conta( "comp maior" );

inf = meio+1;

}

AI6

}

CNT\_Conta( "nao achou" );

return 0;

}

**AS**

Sequência:

**AE:** - a lista existe e não está vazia

- a lista está ordenada crescentemente

- Valem as assertivas estruturais da lista duplamente encadeada

**AS:** - se o elemento existe, o elemento corrente aponta para ele e a função retorna 1. Se não existe, a função retorna 0 e o elemento corrente está em um elemento aleatório.

- a lista está ordenada

- valem as assertivas estruturais da lista duplamente encadeada

AI1: - inf recebe índice do primeiro elemento a ser considerado

AI2: - sup recebe índice do último elemento a ser considerado

Repetição

AE = AI2

AS = AS

AINV: - existem dois conjuntos: diferentes do pesquisado e a pesquisar.

- inf e sup são índices de elementos a pesquisar.

1 ) AE → AINV

Pela AE, inf é o índice do primeiro elemento a ser pesquisado, e sup do último. Existem dois conjuntos: a liberar e liberados. O primeiro contém todos os elementos e o segundo está vazio. Como inf e sup são índices de elementos da lista, necessariamente estes são do conjunto a pesquisar, valendo AINV.

2) AE && (C == F) → AS // ele sempre entra no while, só ignoro isso?

Pela AE, pElem aponta para o primeiro elemento da lista, mas para que (C==F) o elemento precisa ser NULL, significando que a lista está vazia. Vale então a AS, pois não existem elementos alocados.

3) AE && (C==T) + B → AINV

Pela AE, inf é o índice do primeiro elemento a ser pesquisado, e sup do último. Como (C==T), o Bloco B passa pelo menos um elemento de a pesquisar para o conjunto diferente do pesquisado. Continuam existindo os dois conjuntos e inf ou sup passam a ser o índice de outro elemento de a pesquisar, valendo AINV

4) AINV && (C==T) + B → AINV

Para que AINV continue valendo a cada ciclo, B deve garantir que pelo menos um elemento passa do conjunto a pesquisar para diferente do pesquisado e inf ou sup seja reposicionado.

5) AINV && (C==F) → AS

Pela AINV, há dois conjuntos, sendo que todos os elementos com índice entre inf e sup estão em diferentes do pesquisado. A C==F indica que o limite inferior é maior que o superior. Todos os elementos foram pesquisados, e o elemento não existe, indicando AS válida.

6) Término

Como cada ciclo retira pelo menos um elemento do conjunto a liberar, e a quantidade de elementos é finita, então a repetição termina num número finito de passos.

Sequência

AE = AINV

AS = AINV

AI3: - meio é o índice do elemento a ser considerado

AI4: - comp possui o resultado da comparação entre o elemento a ser considerado e o elemento buscado

AI5: - o elemento considerado não é igual ao buscado

AI6: - se o elemento considerado é menor do que o buscado o sup recebe o índice do elemento após o considerado, se o elemento considerado é menor do que o buscado o inf recebe o índice do elemento antes do considerado.

Seleção

AE = AI4

AS = AI5 ou AS

1) AE && (C==T) + B → AS

Pela AE, comp possui o resultado da comparação entre o elemento considerado e o buscado. Como (C==T), o elemento considerado é igual ao buscado. Neste caso B retorna 1, e o elemento corrente aponta para o elemento buscado, valendo AS.

2) AE && (C==F) → AI1

Pela AE, comp possui o resultado da comparação entre o elemento considerado e o buscado. Como (C==F), o elemento é diferente do buscado, valendo AI1.

Seleção

AE = AI5

AS = AI6

1) AE && (C==T) + B1 → AI1

Pela AE, comp possui o resultado da comparação entre o elemento considerado e o buscado. Como (C==T), o elemento considerado é menor que buscado. Neste caso B1 muda sup para o índice do elemento anterior ao considerado, valendo AI1.

2) AE && (C==F) + B2 → AI1

Pela AE, comp possui o resultado da comparação entre o elemento considerado e o buscado. Como (C==F), o elemento é maior que o buscado. Neste caso B2 muda inf para o índice do elemento após o considerado, valendo AI1.

4)

int assertiva\_de\_entrada (Lista \* lst)

{

/\*\*\*\*\*

falta conferir assertivas de uma lista

\*\*\*\*/

if (lst == NULL)

return LISTA\_NAO\_EXISTE;

if (lst->elemcorrente == NULL)

return LISTA\_VAZIA;

ant = lst->prim->valor;

aux = lst->prim->prox;

while (aux!=NULL)

{

if(compara(aux->valor,ant)==MENOR)

reutrn NAO\_ORDENADA;

ant=aux->valor;

aux=aux->prox;

}

return ASSERTIVAS\_OK;

}

int assertiva\_de\_saida (Lista \* lst, int retorno, void \* valor)

{

/\*\*\*\*\*

falta conferir assertivas de uma lista

\*\*\*\*/

if (lst == NULL)

return LISTA\_NAO\_EXISTE;

if (lst->elemcorrente == NULL)

return LISTA\_VAZIA;

ant = lst->prim->valor;

aux = lst->prim->prox;

while (aux!=NULL)

{

if(compara(aux->valor,ant)==MENOR)

reutrn NAO\_ORDENADA;

ant=aux->valor;

aux=aux->prox;

}

if(retorno == 1)

{

if(compara(valor,lst->elemcorrente->valor)!=IGUAL)

return ELEM\_CORR\_NAO\_E\_O\_PROCURADO;

}

return ASSERTIVAS\_OK;

}