TDA - Fila Dinâmica (comprimento dinâmico/variável)

Fila Dinâmica:

- FIFO: entrada por uma extremidade (cauda), saída pela outra (frente);

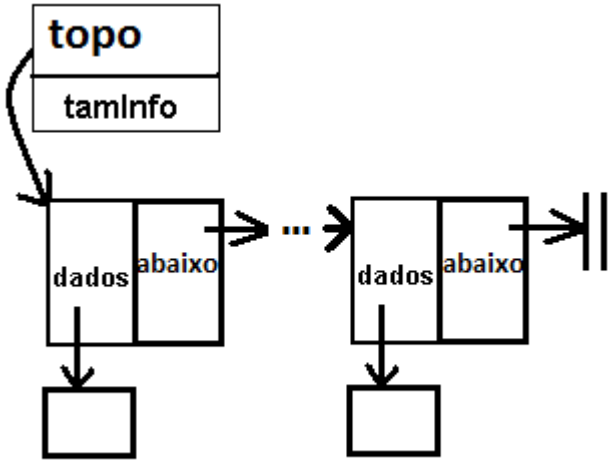
- O comprimento máximo não é estático/pré-determinado;

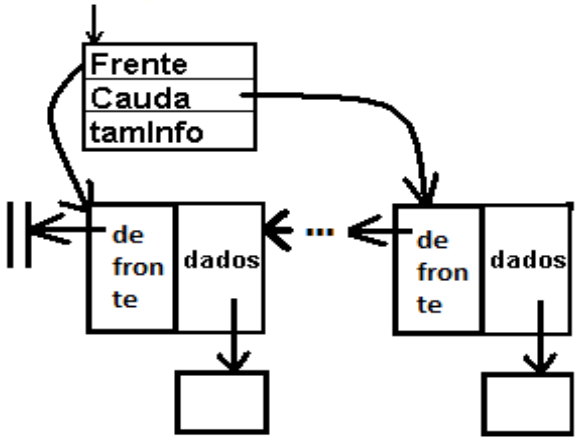
- As inserções exigem alocação (malloc) do novo nó;

- Remoções exigem liberação (free) do item removido;

- FDSE (FDDE) facilmente adaptada a partir da PDSE (PDDE):
  Pilha Dinâmica → Fila Dinâmica
    - Descritor possui cauda e frente;
    - Inserções pela cauda da fila;
    - Remoções pela frente da fila;

PDSE → FDSE






```
typedef struct noPDSE
{void *dados;
 struct noPDSE *abaixo;
}NoPDSE,*pNoPDSE;

typedef struct PDSE
{ int tamInfo;
  pNoPDSE topo;
}PDSE;
```

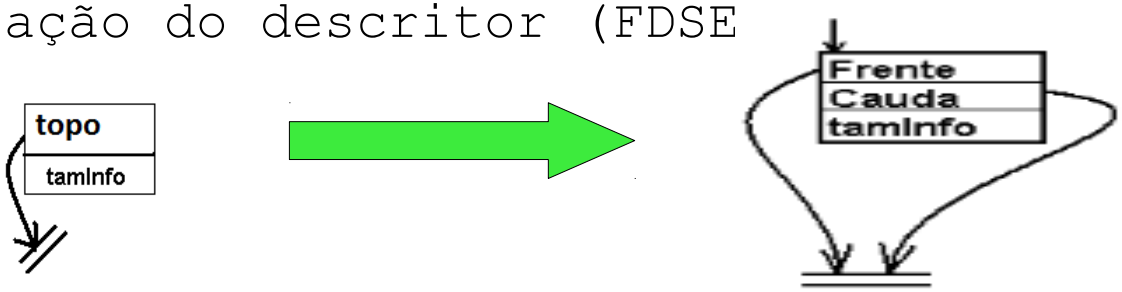
```
typedef struct noFDSE
{void *dados;
 struct noPDSE *defronte;
}NoFDSE,*pNoFDSE;

typedef struct FDSE
{ int tamInfo;
  pNoFDSE cauda;
  pNoFDSE frente;
}FDSE;
```

PDSE → FDSE

1) Criação: o mesmo código da PDSE com pequenas alterações, basicamente:

1 Alocação do descritor (FDSE);

2 Inicialização do descritor (FDSE



2) Busca na frente e busca na cauda: basicamente o mesmo código da busca no topo, substituindo identificadores de maneira adequada.

```
SE (fila != VAZIA)
   memcpy(destino,..., p->taminfo)
   return(SUCESSO)
```

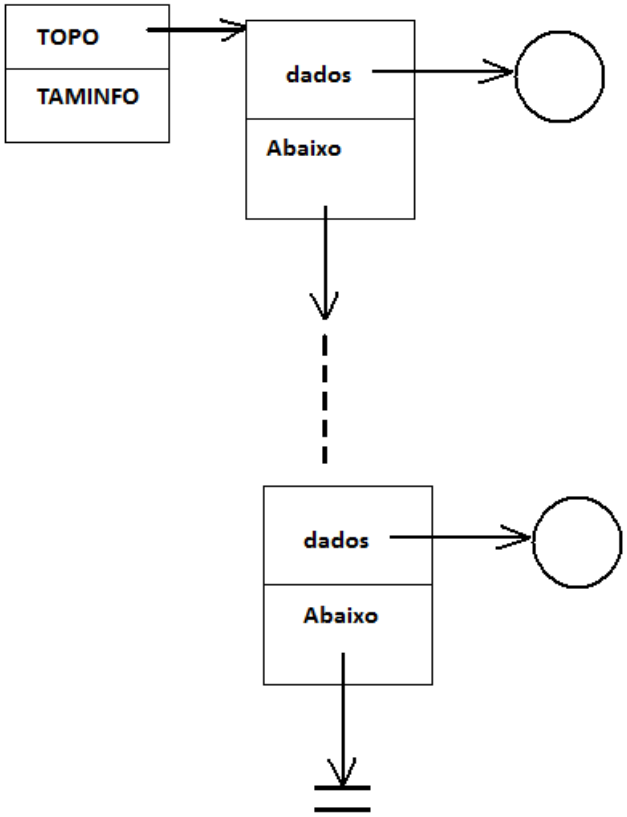
```
p->cauda->dados
ou
p->frente->dados
```

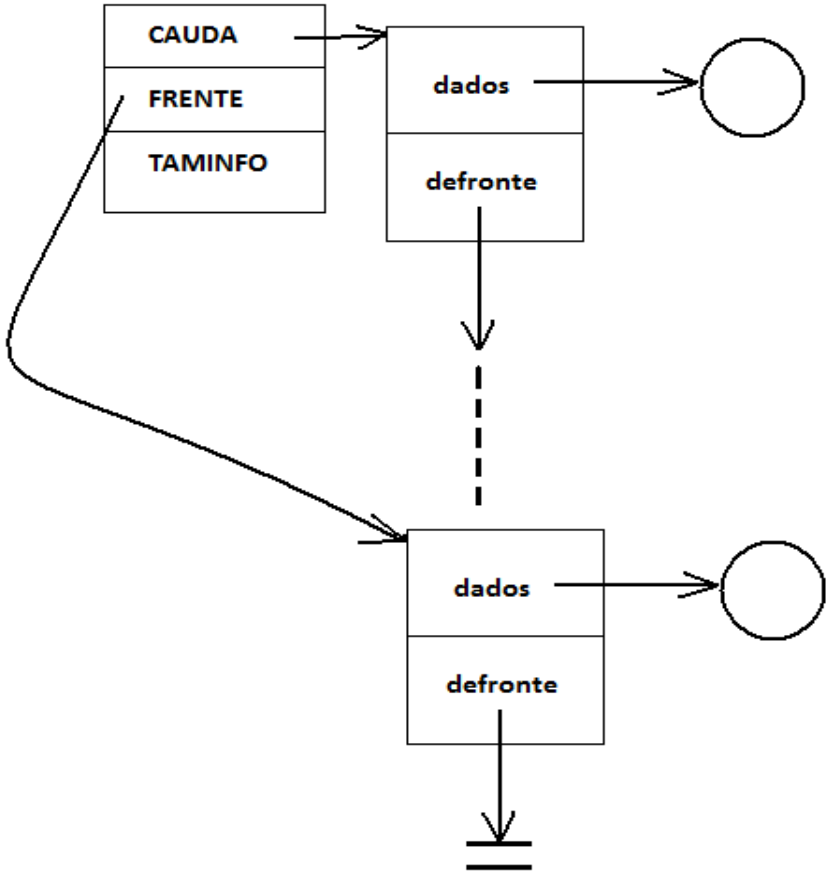
3) Testa condição de fila vazia: verificar se "cauda" e "frente" são nulos. Similar ao teste de PDSE vazia.

PDSE → FDSE

4) Inserção:

Note a similaridade PDSE/FDSE.





PDSE → FDSE

4) Inserção:

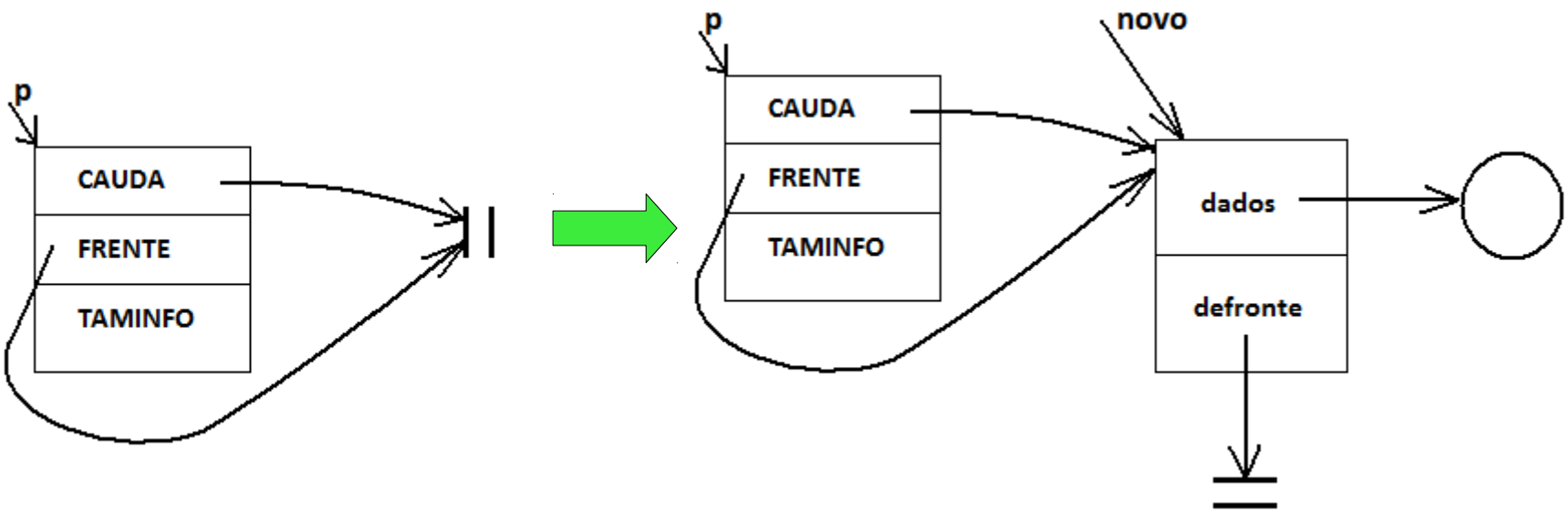
Basicamente trata-se do mesmo código de inserção. As alterações principais são:

a) Alterar identificadores

b) Substituir "topo" por "cauda" e...

b.1)Caso seja a primeira inserção, a "cauda" e a "frente" devem apontar para o mesmo nó.

```
novo->defronte=p->cauda;
p->frente=p->cauda=novo;
```



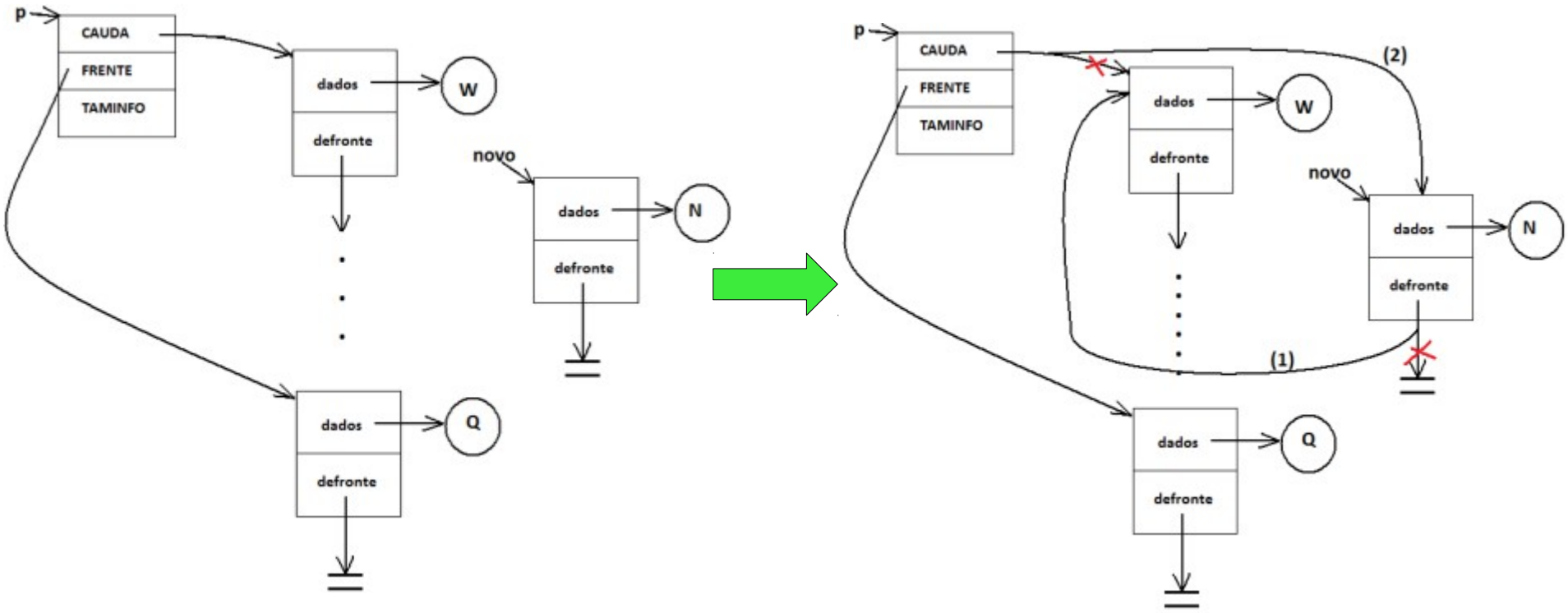
PDSE → FDSE

4) Inserção:

b.2) Caso não seja a 1a inserção:

(1)novo->defronte = p->cauda;

(2)p->cauda=novo;



PDSE → FDSE

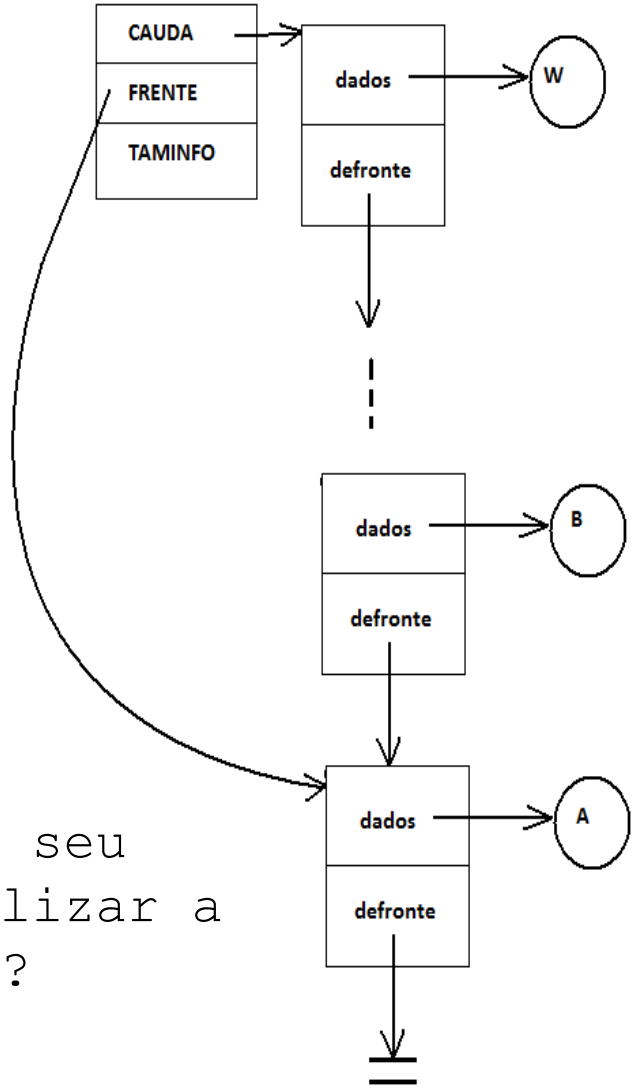
5) Remoção: adaptação mais trabalhosa

A remoção da fila é feita pela sua "frente":

No exemplo ao lado, a remoção eliminará o item 'A' e em seguida "frente" será redirecionada para 'B';

Nessa fila cada nó de dados só "enxerga" o seu vizinho de frente (defronte);

O nó frontal (A) não "enxerga" o seu vizinho de trás (B)... como atualizar a "frente" redirecionando-a para B?

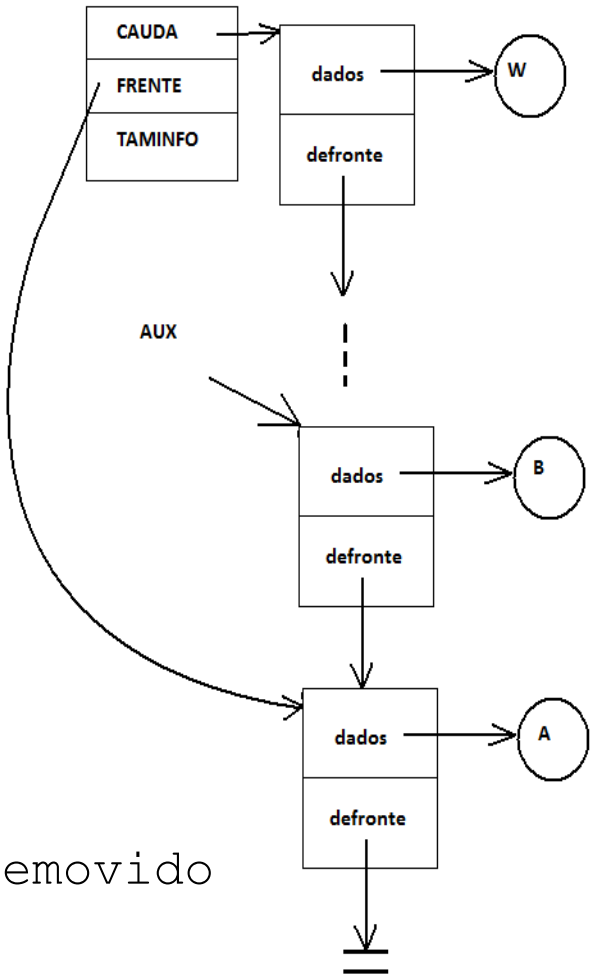


PDSE → FDSE

5) Remoção: adaptação mais trabalhosa

… é necessário um ponteiro auxiliar que, a partir da cauda, alcance o penúltimo item de saída (item B):

```
SE (fila != VAZIA)
{ aux=p->cauda;
  enquanto(aux != nulo E
           aux-> defronte != p->frente)
    { aux=aux->defronte;
    }
  free(p->frente->dados);
  free(p->frente);
  p->frente=aux;
  SE (aux == nulo)
    // havia um único elemento que foi removido
    {p->cauda=aux;
    }
}
```



PDSE → FDSE

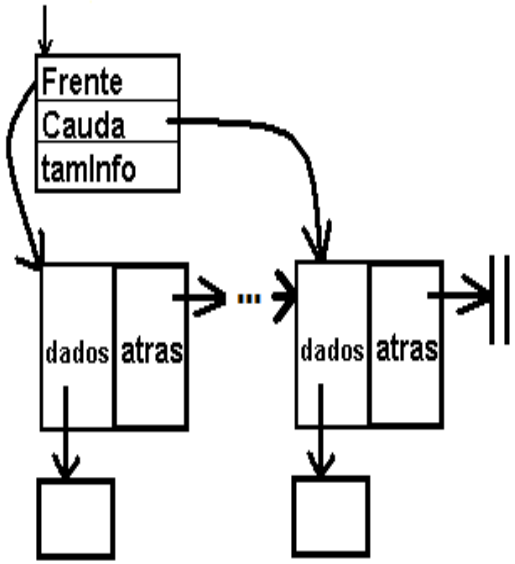
6) **Reinicialização:**

   A mesma lógica que reinicia a PDSE, porém o "topo" é substituído pela "cauda" e ao final é preciso atualizar (anular) o campo "frente" da fila.

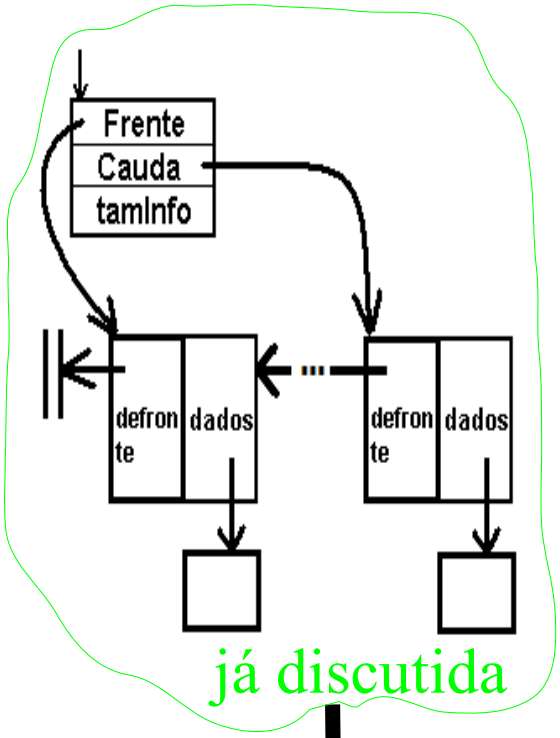
7) **Destruição:**
   A mesma lógica que destrói a PDSE.

Filas de crescimento dinâmico:

FDSEs



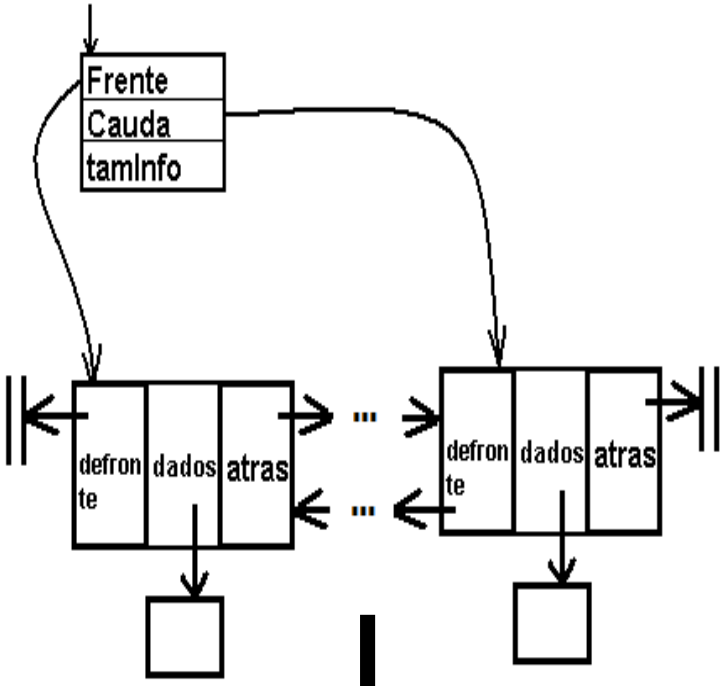
cada nó enxerga seu vizinho detrás



já discutida

cada nó enxerga seu vizinho da frente

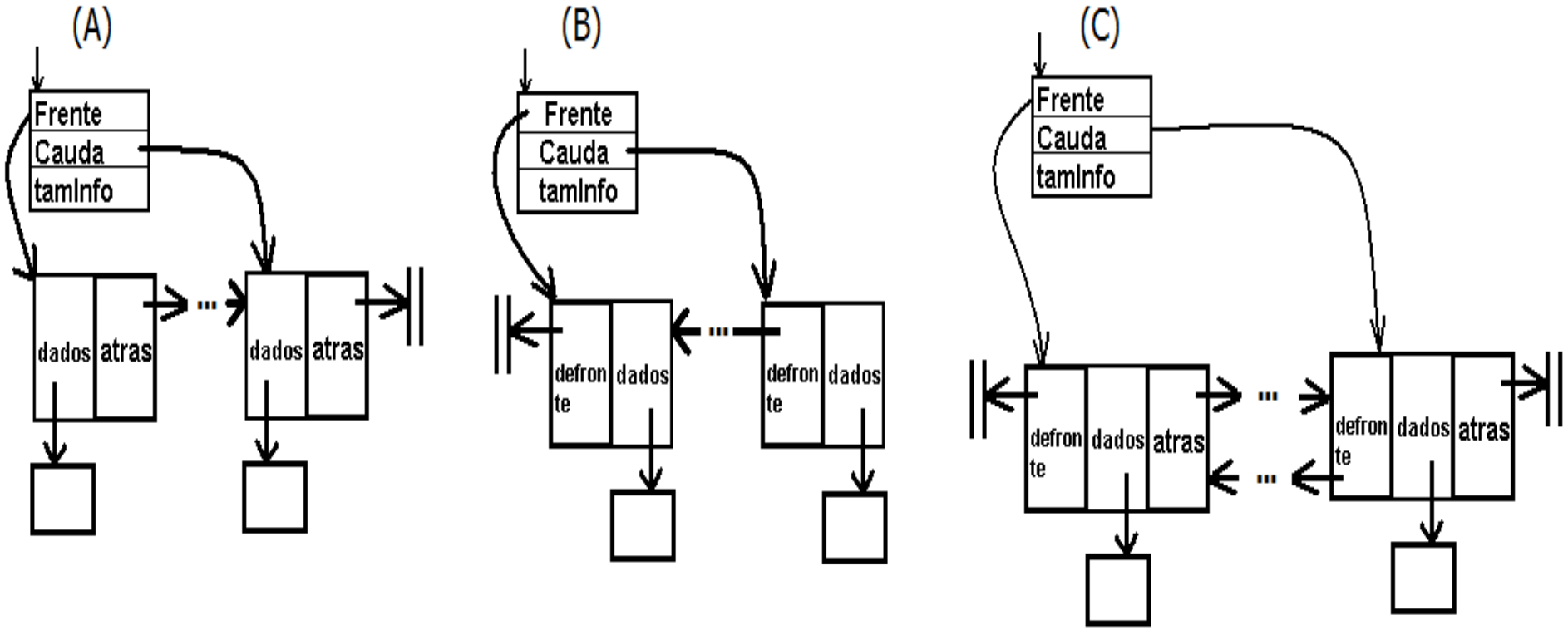
FDDE



cada nó enxerga seus vizinhos detrás e da frente

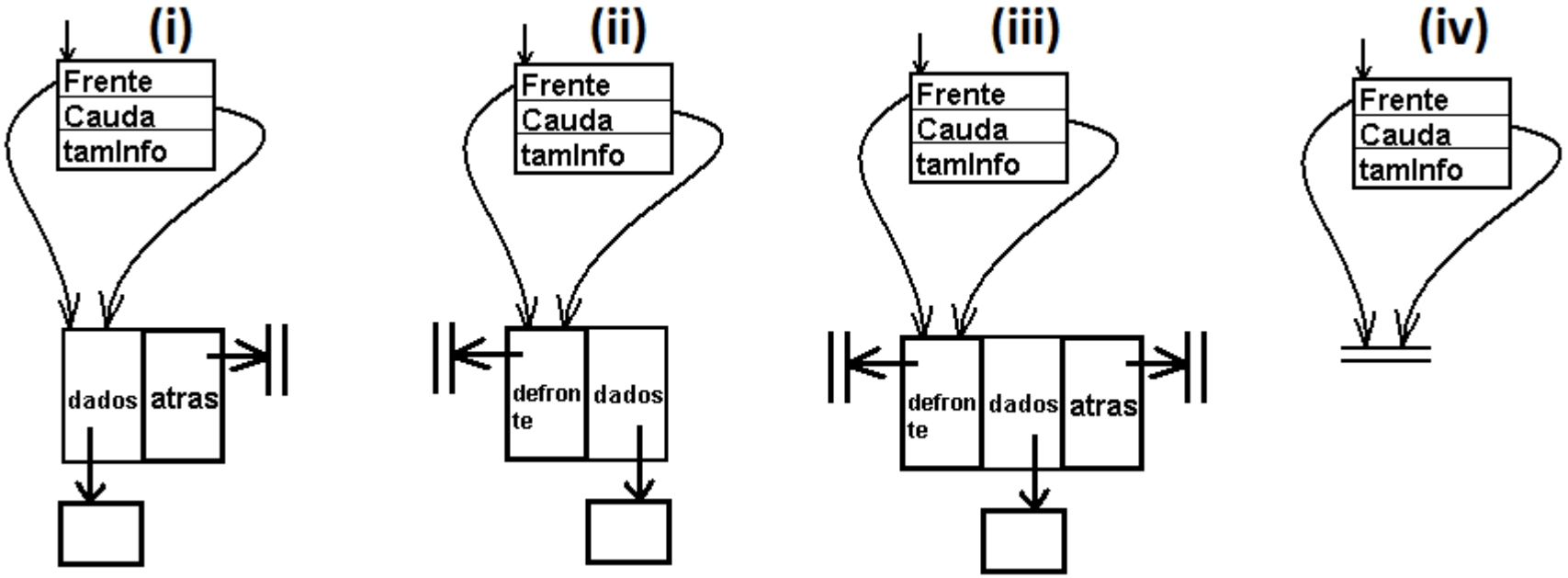
Fila Dinâmica:

- Em A, B e C temos filas com $n > 1$ elementos
- Esses estados devem ser considerados, especialmente, durante os processos de remoção/inserção.

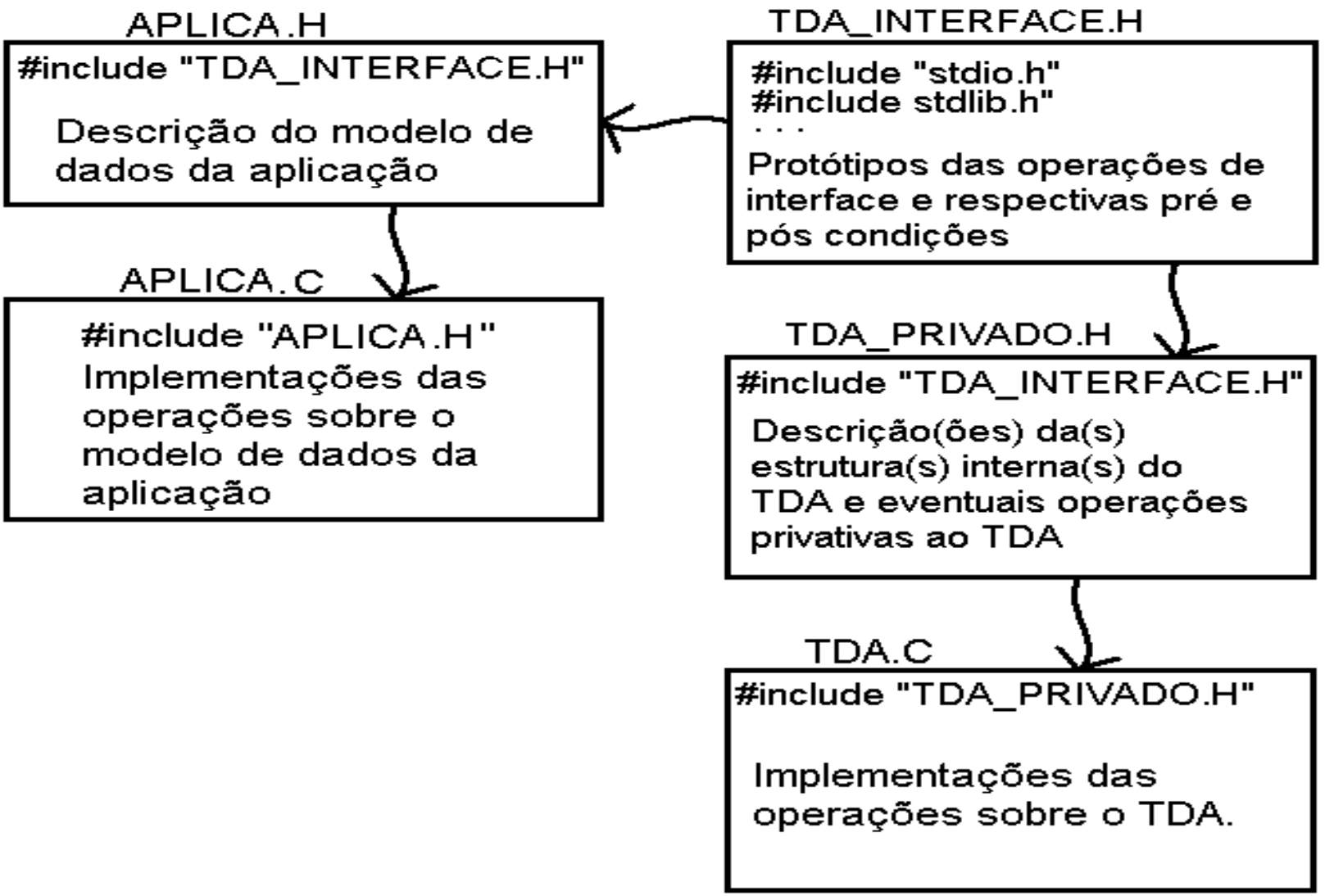


Fila Dinâmica:

- Em i, ii e iii temos filas com um único elemento.
- Em iv a configuração geral da fila vazia.
- Esses estados da fila devem ser considerados, especialmente, durante o processo de remoção/inserção.

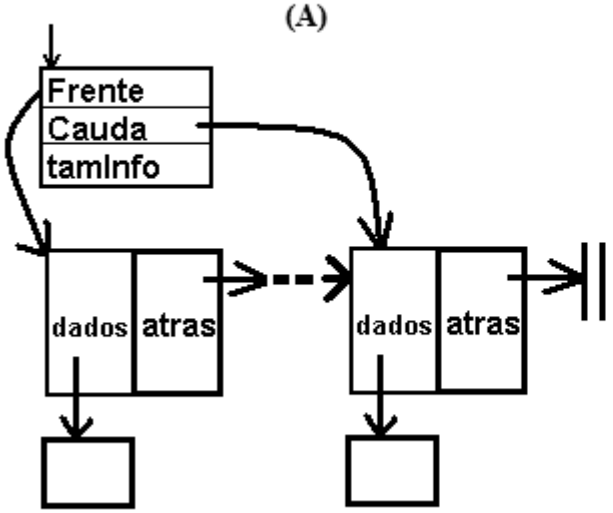


Utiliza a mesma arquitetura de TDA que vem sendo aplicada:

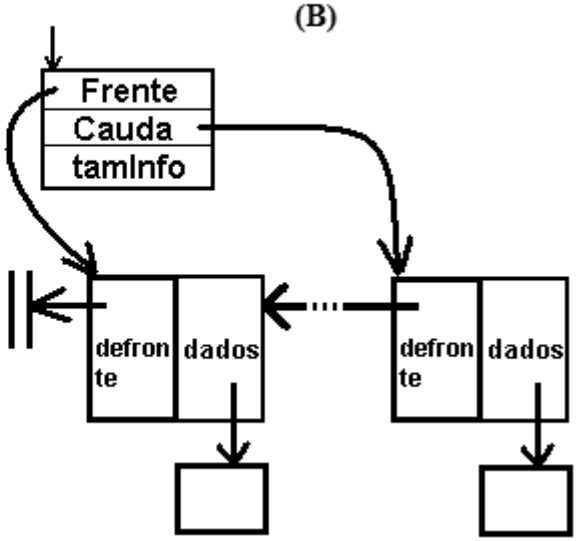


Variações de FDSE's

(A)

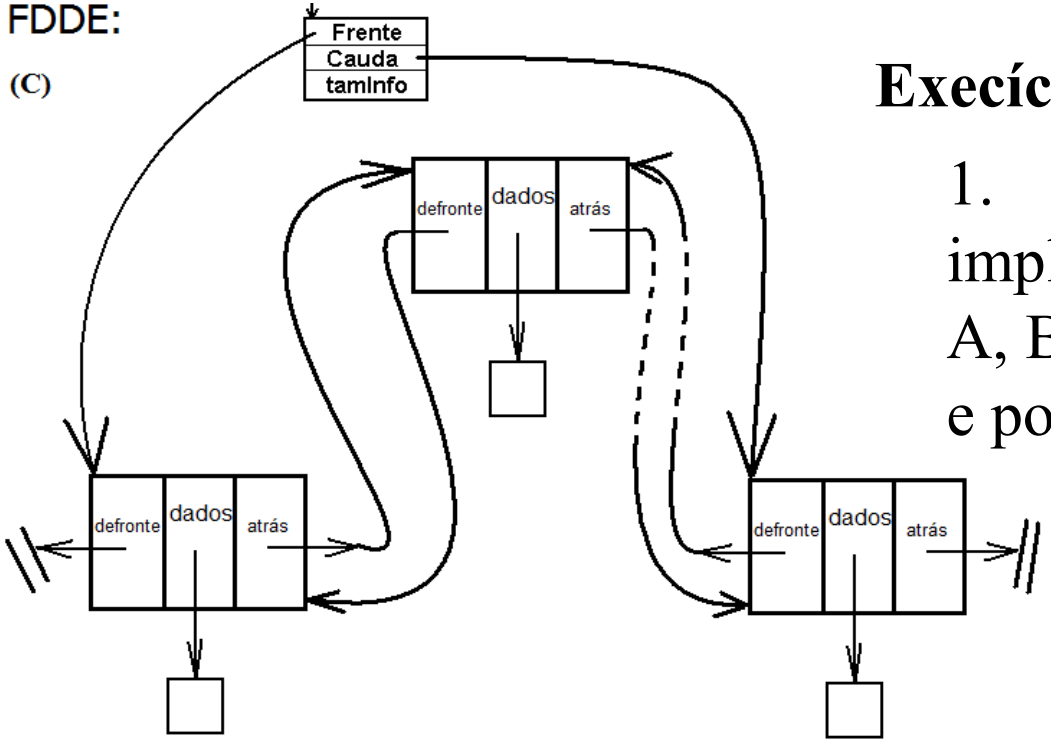


(B)



FDDE:

(C)



Execícios:

1. Discuta e compare as implementações apresentadas em A, B e C. Qual a mais desvantajosa e porque?

2. Implemente a FDSE e a FDDE com a seguinte funcionalidade:

```
int cria(ppFila pp, int tamVet, int tam info);

int destroi(ppFila pp);

int buscaNaFrente(pFila p, void *reg);

int buscaNaCauda(pFila p, void *reg);

int testaVazia(pFila p);

int reinicializa(pFila p);

int enfileira(pFila p, void *novo);

int desenfileira(pFila p, void *reg);
```