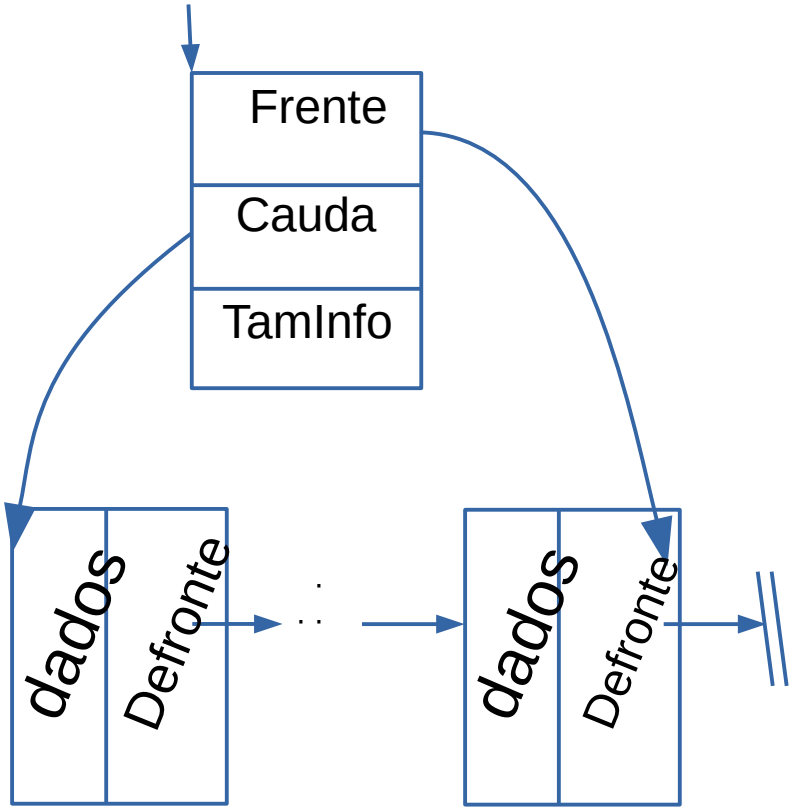
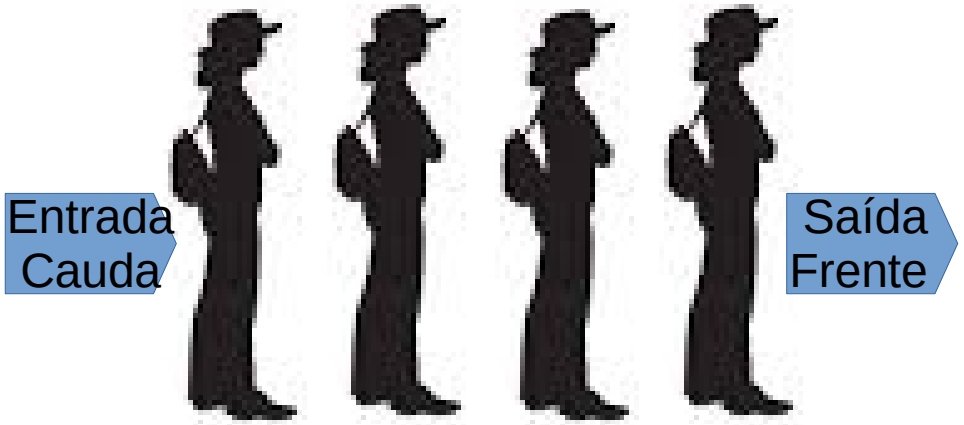
Fila Simplesmente Encadeada

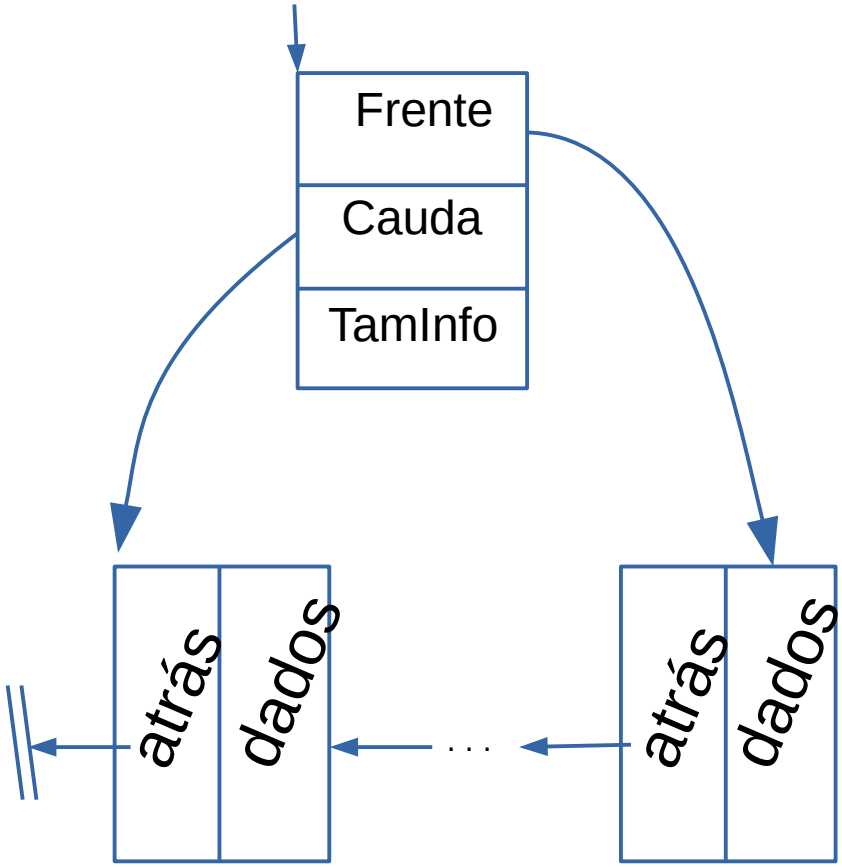
Cada nó sabe onde está seu sucessor

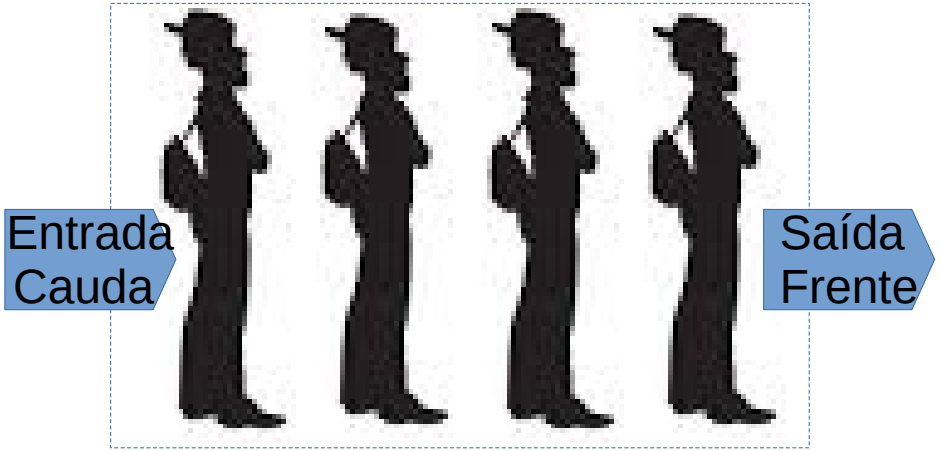




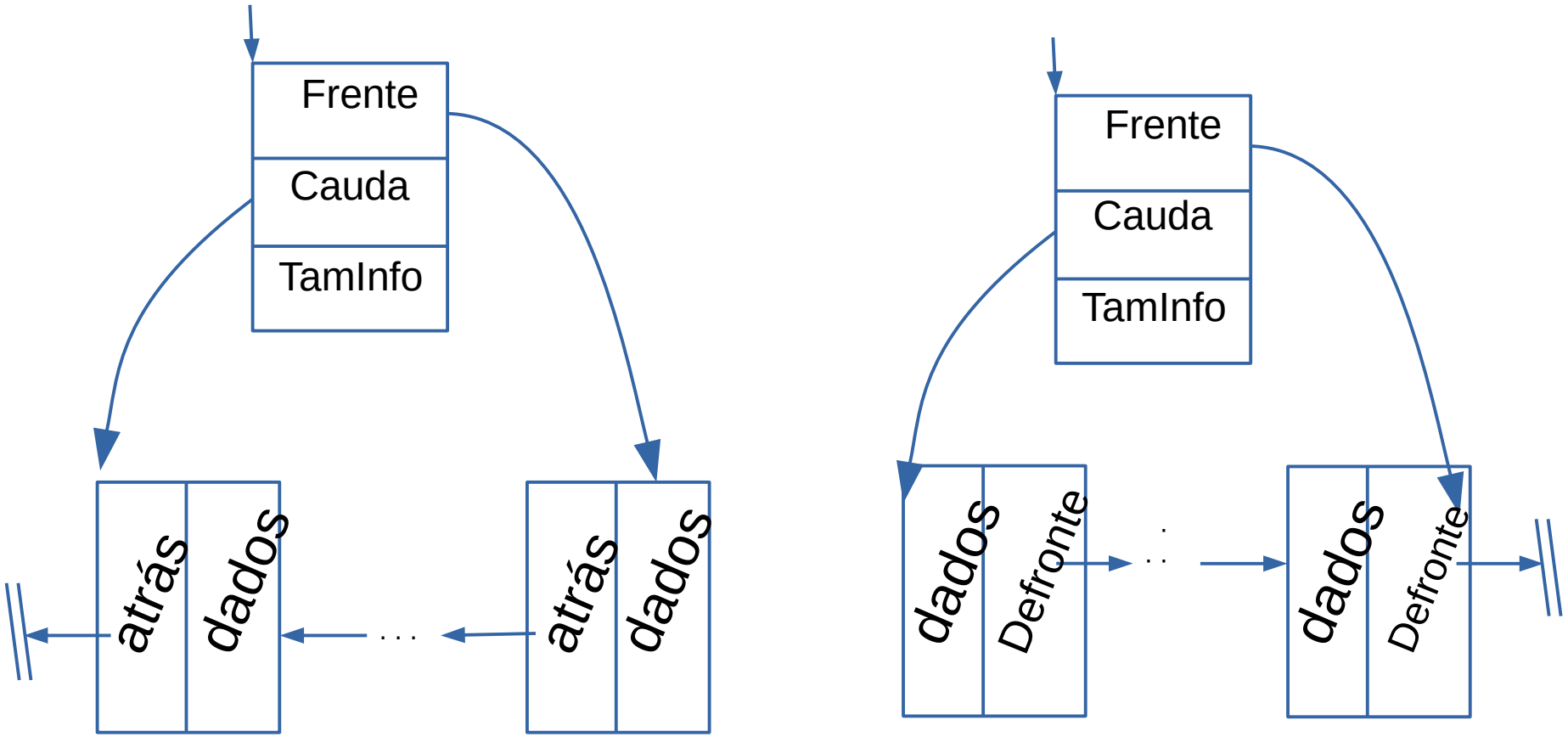
Fila Simplesmente Encadeada

Cada nó sabe onde está seu antecessor

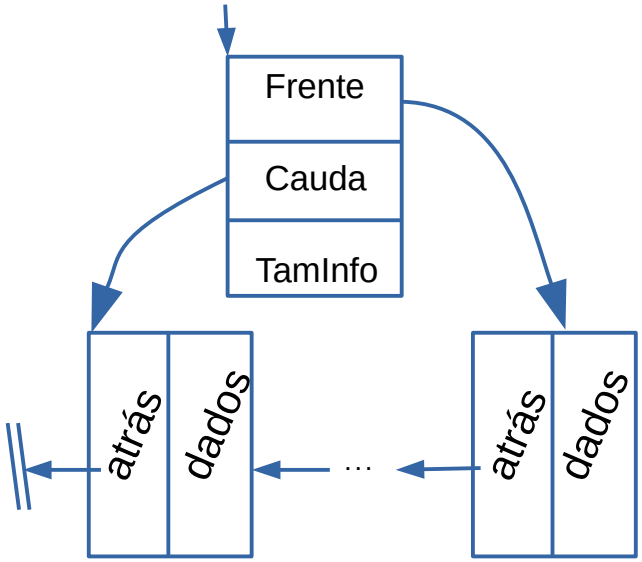


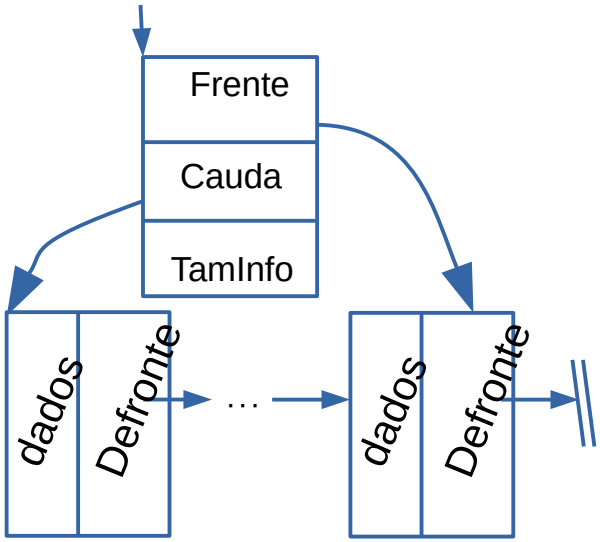


Fila Simplesmente Encadeada



Compare as duas implementações em termos das operações usuais, qual delas é a mais eficiente especialmente em relação a inserção e remoção.



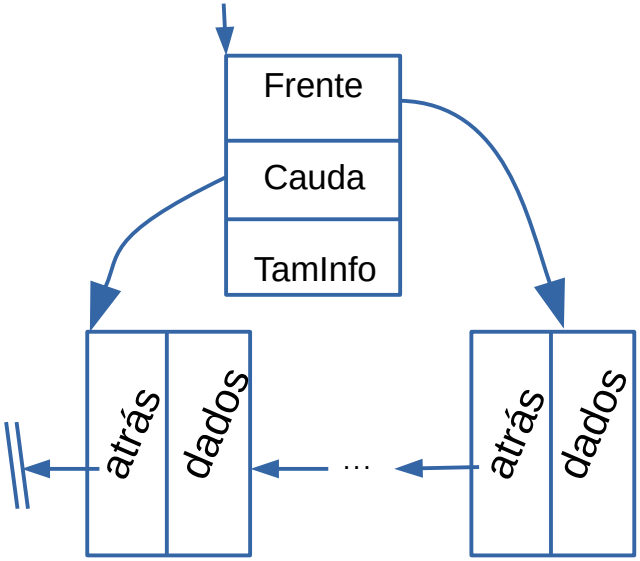


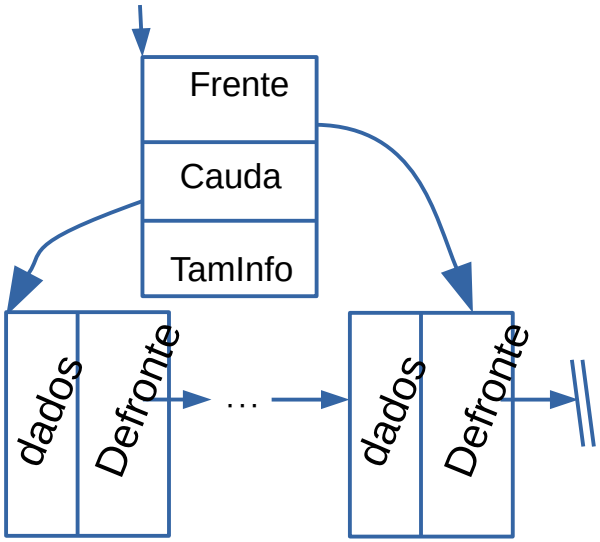
Faremos uma análise comparativa usando a interface básica:

```
struct descF * cria(int tamInfo);
int buscaNaCauda(info *reg, struct descF *p);
int buscaNaFrente(info *reg, struct descF *p);
int testaVazia(struct descF *p);

int reinicia(struct descF *p);
struct descF * destroi(struct descF *p);

int insere(info *novo,struct descF *p);
int remove_(info *reg, struct descF *p);
```





struct descF * cria(int tamInfo):

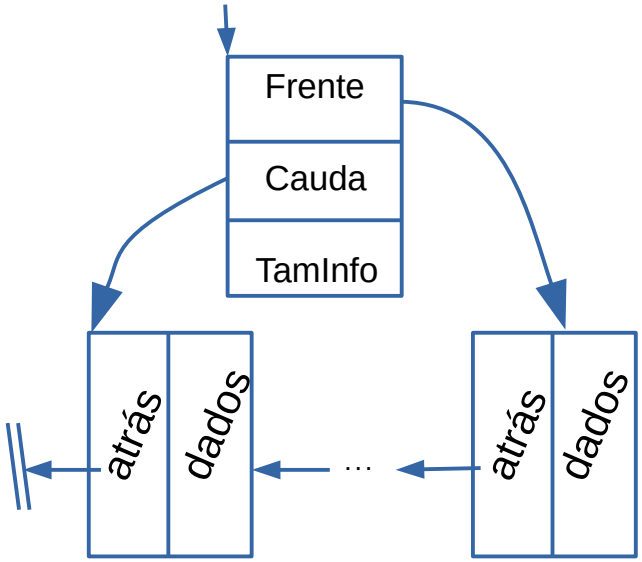
- Nos dois casos a instanciação da FES envolve o mesmo custo computacional O(1): alocação e inicialização do descritor

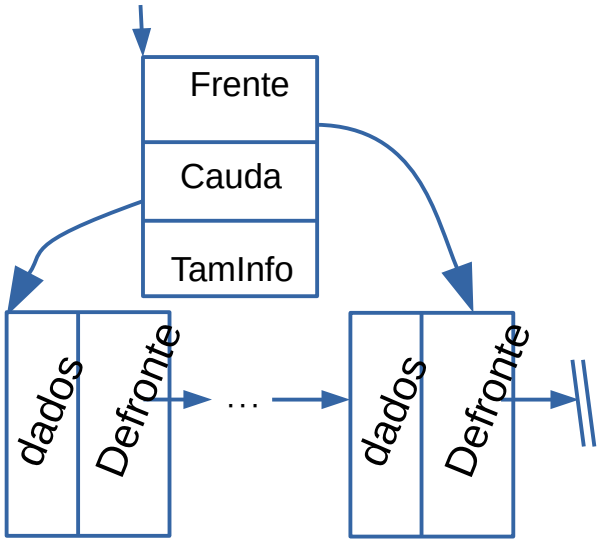
int buscaNaCauda(info *reg, struct descF *p)
int buscaNaFrente(info *reg, struct descF *p)

- Lista vazia: returna “fracasso” → mesmo custo para ambas: O(1)
- Lista não vazia: mesmo custo O(1) (acesso direto via descritor e execução de um memcpy)

int testaVazia(struct descF *p)

- mesmo custo O(1) (acesso direto via descritor à frente e à cauda verificando se são ambos setados em NULL)





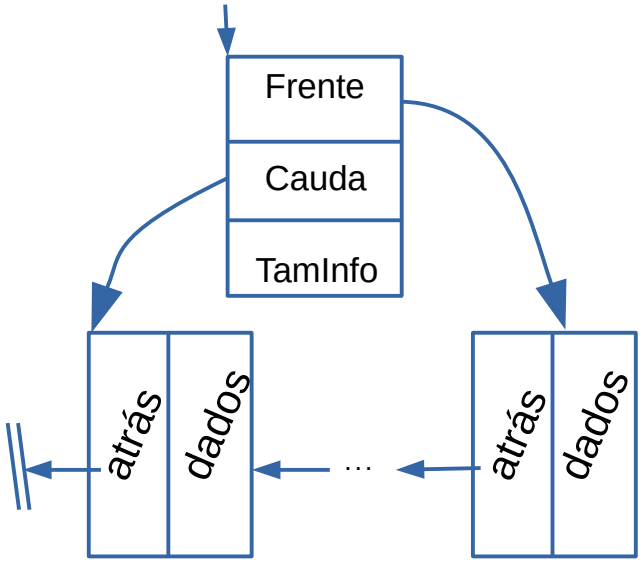
int reinicia(struct descF *p)
Para as duas estratégias acima:

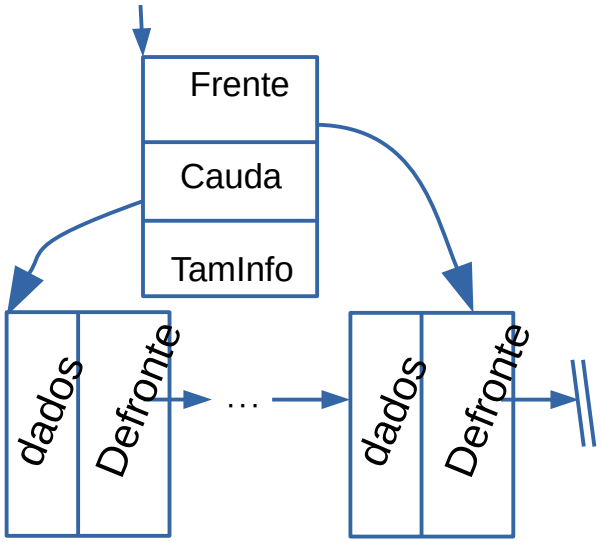
- Implica no “reset” de dados da FSE, a fila retorna ao status “vazia”
- Considerando o pior caso, no qual a fila contenha N elementos (uma significativamente grande, ainda que finita) o eset implica nos passos:
  - Liberação (free) de cada nó de dados seguida da “anulação” dos ponteiros frente e cauda no descritor;
    - A liberação dos N nós demanda um laço → custo computacional O(N)

struct descF * destroi(struct descF *p);
Para as duas estratégias acima:

- Implica no “reset” de dados da FSE seguido da liberação do descritor
  - A operação dominante é o laço do reset de dados → custo computacional O(N)

Nos dois casos as duas estratégia de estruturas equivalem em termos de custo computacional.





int insere(info *novo,struct descF *p);

Alocação de memória para o novo nó:
aux=malloc(sizeof(p->taminfo))

Cópia da informação para o novo nó:
memcpy(&(aux→dados),novo,p→taminfo)

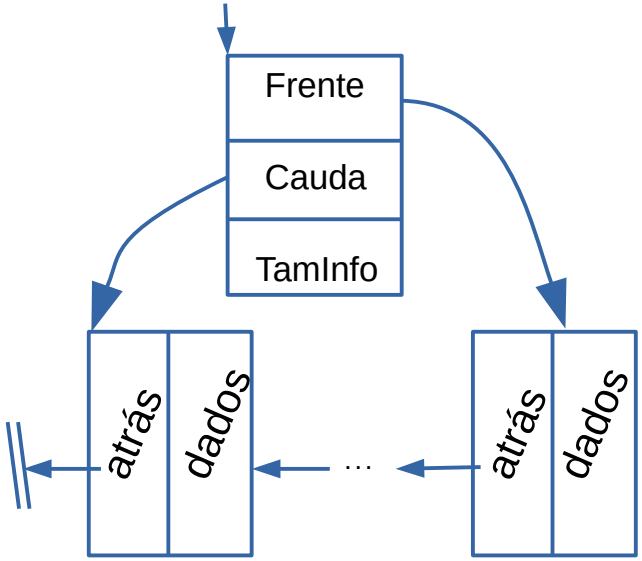
Inserção do novo nó na cauda da fila:
fila vazia:
p->frente=p->cauda=aux
aux->atras=NULL
fila não vazia:
p->cauda->atras=aux
p->cauda=aux
aux->atras=NULL

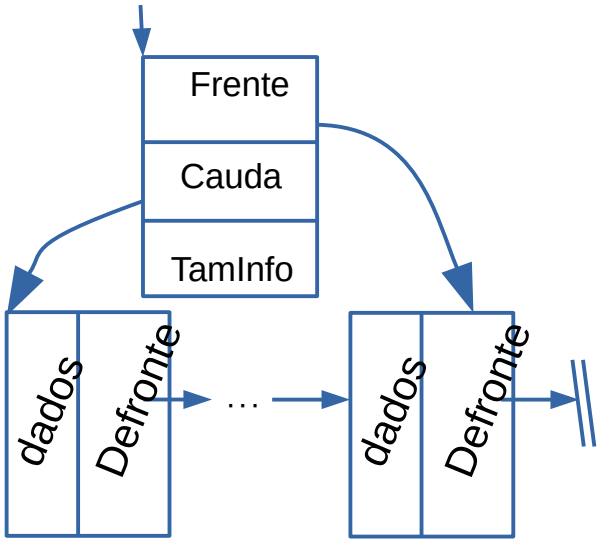
Alocação de memória para o novo nó:
aux=malloc(sizeof(p->taminfo))

Cópia da informação para o novo nó:
memcpy(&(aux→dados),novo,p→taminfo)

Inserção do novo nó na cauda da fila:
fila vazia:
p->frente=p->cauda=aux
aux->defronte=NULL
fila não vazia:
aux->defronte=p->cauda
p->cauda=aux
aux->atras=NULL

Ambos com custo computacional de ordem O(1)





int remove_(info *reg, struct descF *p);

Fila vazia: returna fracasso

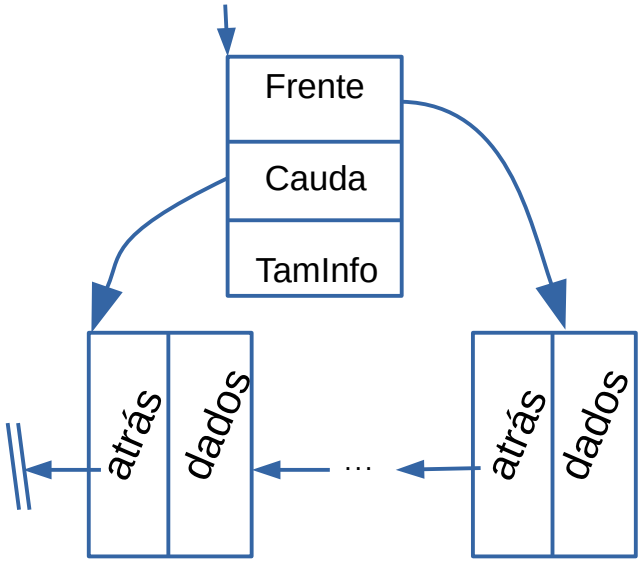
Fila não vazia com 1 único elemento:
- Cópia da informação frontal para o chamador:
  memcpy(reg,&(p->frente->dados),p->taminfo)
- Liberação do único nó:
  free(p->frente)

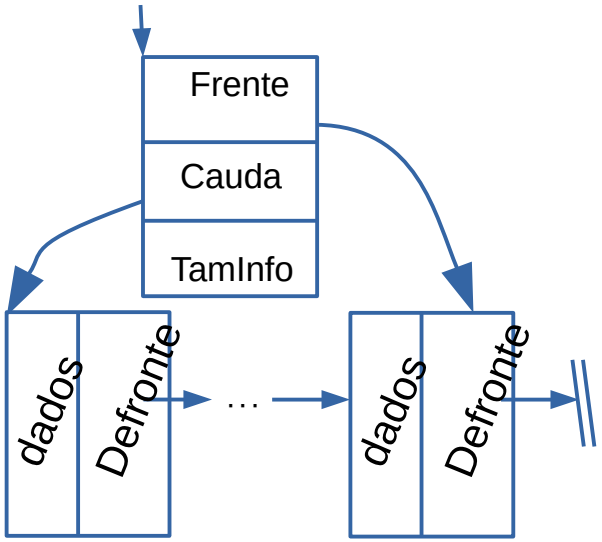
Fila vazia: returna fracasso

Fila não vazia com 1 único elemento:
- Cópia da informação frontal para o chamador:
  memcpy(reg,&(p->frente->dados),p->taminfo)
- Liberação do único nó:
  free(p->frente)

Ambos com custo computacional de ordem O(1)

Mas, o que ocorre se houver N elementos na FSE?





int remove_(info *reg, struct descF *p);

N elementos na FSE:

- Cópia da informação frontal para o chamador:
  memcpy(reg,&(p->frente->dados),p->taminfo)

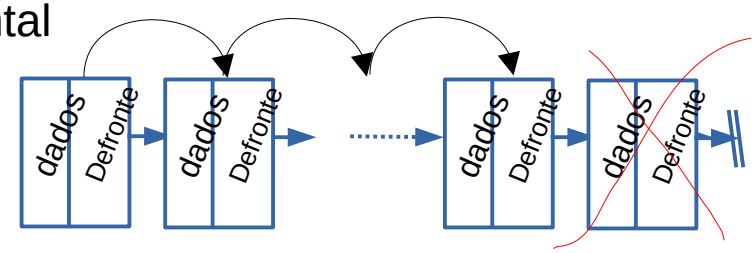
- Remoção:
Atualização da frente da fila com a liberação do nó frontal:
  aux=p→frente
  p->frente=aux->atras;
  free(aux)

Custo computacional de ordem O(1)

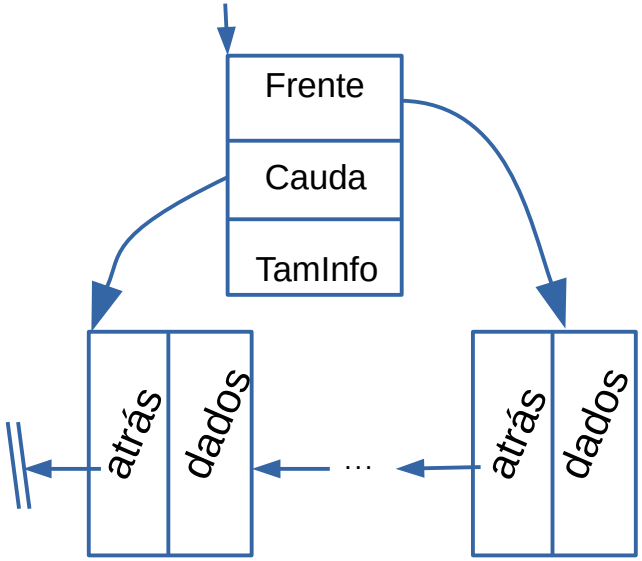
N elementos na FSE:

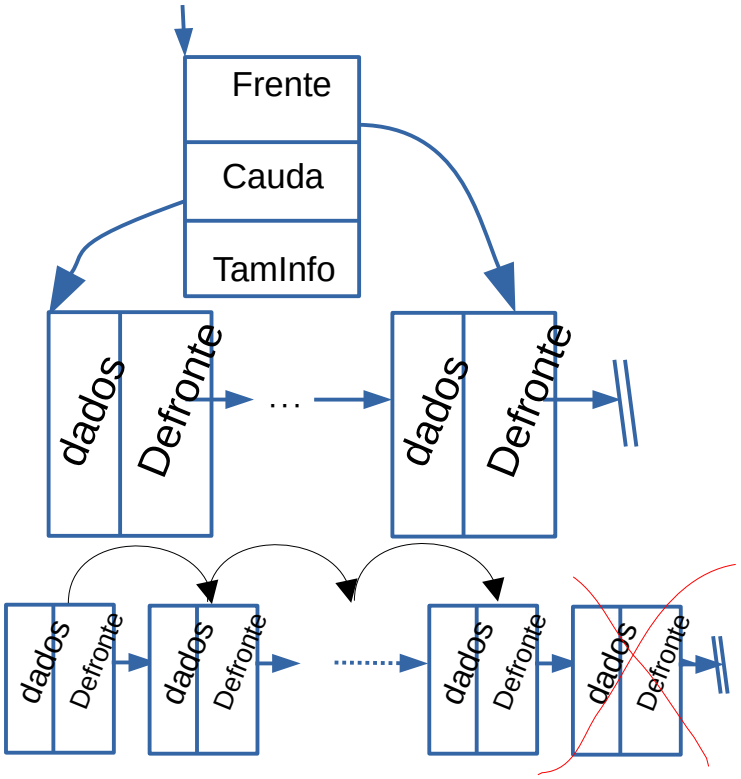
- Cópia da informação frontal para o chamador:
  memcpy(reg,&(p->frente->dados),p->taminfo)

- Remoção:
Atualização da frente da fila com a liberação do nó frontal:
  Necessariamente demanda um laço de ordem N para determinação do endereço do novo nó frontal



Custo computacional de ordem O(N)

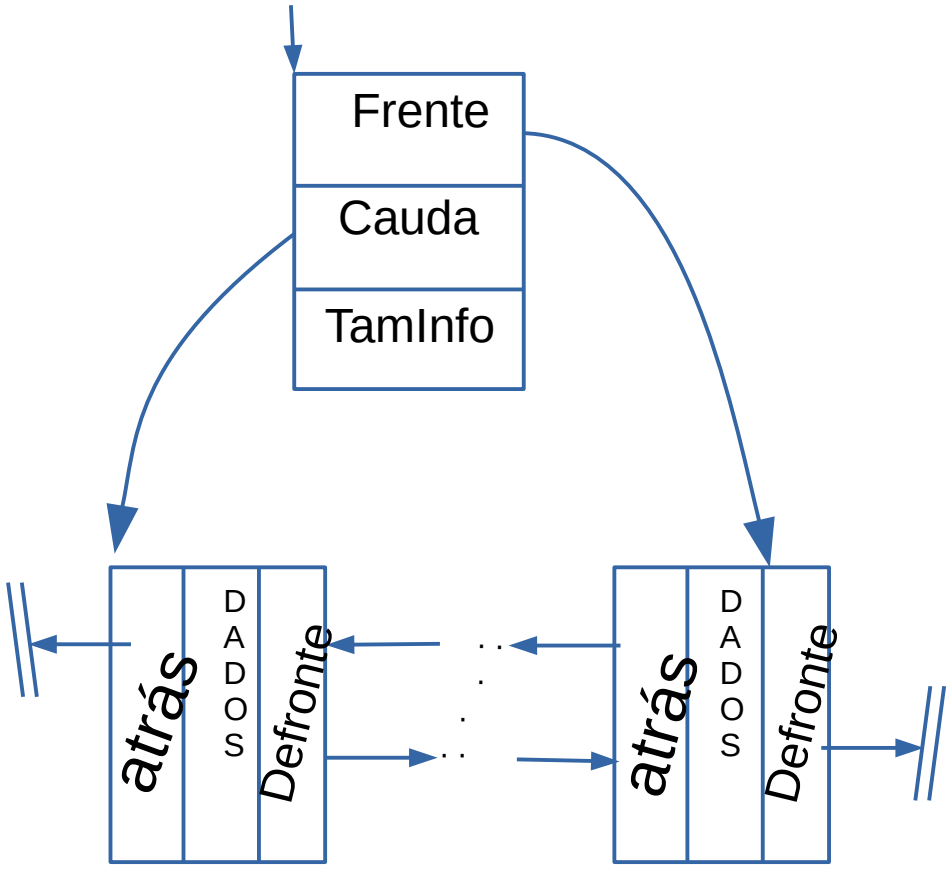


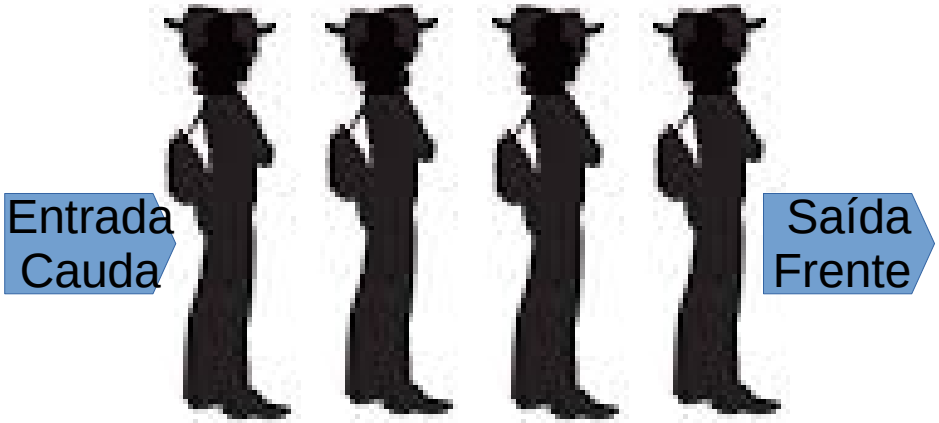


Dentro dos parâmetros de comparação utilizados na nossa análise, dentre as duas, a implementação da direita é a desvantajosa, devido ao seu desempenho na remoção ser O(N) ao passo que a da esquerda é O(1).

Em todas as outras operações utilizadas, as duas equivalem.

Fila Duplamente Encadeada





Implemente esta fila a partir do que você entendeu das duas anteriores.