1. Fila linear

Definição: Em uma fila, os novos itens são colocados atrás do último, portanto o primeiro item inserido está na frente e o último item inserido está atrás. O modelo intuitivo de uma fila é uma estrutura de espera para atendimento ou de reserva.

Caso particular de lista no qual as operações são feitas em uma extremidade denominada R (rear) e as exclusões são feitas na extremidade oposta chamada F (front).

Estratégia: FIFO (first in first out).

Quando utilizar: quando a ordem de chegada precisa ser preservada

Tipos abstratos: representação interna + operações de manipulação:

- **incluir nó na fila**

  depois do último
- **excluir nó da pilha**

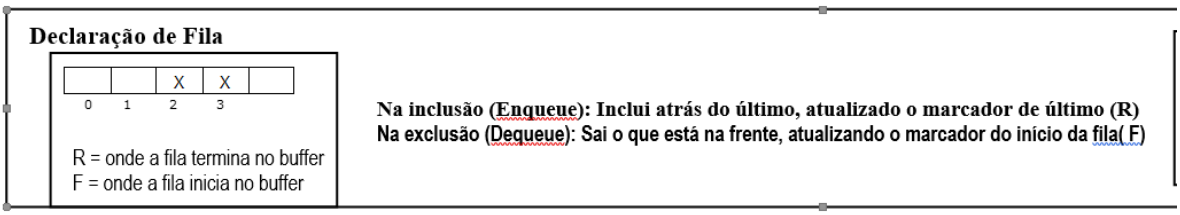
  o nó que está na frente(não tem busca)
- **acessar nó da pilha**

  o nó que está na frente (não tem busca!!!)

```
/* Primeiro, a definição de um tipo para o elemento */

typedef struct {
    int info;
} TNO;


/* Em seguida, um tipo para a pilha  */
typedef struct  {
    TNO vnos[MAX];
    int F,R;
    int maximo;
} TFILA;
```



Primeira Forma:

- Para criar fila vazia temos:

  F = -1 e R = -1

- Representação de uma fila vazia::

  F = R = -1 -> quando a fila está virgem

  F > R -> após operações

Implicações na inclusão (Enqueue):

> Se a fila é virgem, inclui na primeira posição e atualiza F e R
>
> Se a fila não é mais virgem, inclui atrás do R e incrementa o R

Implicações na exclusão (Dequeue):

> Sai o que está em F e F é incrementado
>
> Se for o último R também é atualizado

```
/*cria fila vazia*/
  void cria_fila_vazia(TFILA *pfila){
    pfila -> F = pfila -> R = -1;
}
```

```
//fila está vazia?
int fila_vazia(TFILA *fila){
    if ( (fila->F == -1)
        return 1;
    if  (fila->F > fila->R)
        return 1;
    return 0;
}
```

```
//fila está cheia?
int fila_cheia(TFILA *fila){
     return (fila -> R == MAX - 1)
}
```

```
// Enfileira ou enqueue
int enqueue (TFILA*pfila, TNO no){
    if (fila_cheia (pfila))
        return 0;
    (pfila ->R)++;
    if (pfila -> F == -1)
        (pfila -> F)++;
    pfila -> vnos[pfila -> R] = no
    return 1;
}
```

```
// Desenfileira ou Dequeue
int dequeue(TFILA*pfila,TNO *no){
    if (fila_vazia(pfila)
        return 0;
    *no = pfila -> vnos[pfila -> F]
    (pfila -> F)++;
    return 1;
}
```

Segunda Forma:

Problemas da Primeira Forma:

Ao criar uma fila vazia por F = -1 e R = -1 implica na ocorrência de dois problemas:

- A primeira inclusão em uma fila virgem cria a situação particular de atualizar o F
- O reconhecimento de uma fila vazia envolve reconhecer se ela é virgem (F = R = -1) ou não.

Para não termos o problema da "virgem"e da exclusão do último, devemos alterar a representação (e criação) da fila vazia: como a sequencialidade é dada pela vizinhança, sabe-se que o primeiro nó ficará na pos 0 do vetor, portanto, o F pode iniciar apontando para o slot 0.

Deste modo, retira-se os casos particulares que surgiram por causa da inicialização

```
/*cria fila vazia*/
  void cria_fila_vazia(TFILA *pfila){
    pfila -> F = 0;
    pfila -> R = -1;
}
```

```
//fila está vazia?
int fila_vazia(TFILA *fila){
    return (fila->F > fila->R)
}
```

```
//fila está cheia?
int fila_cheia(TFILA *fila){
     return (fila -> R == MAX - 1)
}
```

```
// Enfileira ou enqueue
int enqueue (TFILA*pfila, TNO no){
    if (fila_cheia (pfila))
        return 0;
    (pfila ->R)++;
    pfila -> vnos[pfila -> R] = no
    return 1;
}
```

```
// Desenfileira ou Dequeue
int dequeue(TFILA*pfila,TNO *no){
    if (fila_vazia(pfila)
        return 0;
    *no = pfila -> vnos[pfila -> F]
    (pfila -> F)++;
    return 1;
}
```

Terceira Forma:

Problemas da Primeira e Segunda Formas:

A medida que inclusões e exclusões vão sendo realizadas, a fila vai andando no buffer em direção à extremidade direita, o que pode implicar em um **overflow** (fila cheia) de uma fila que está vazia.

Solução:

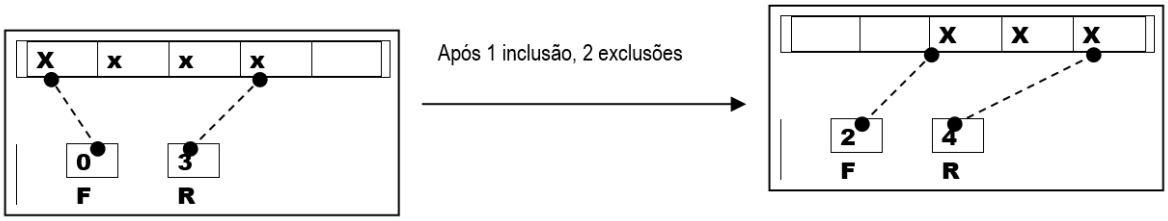
Quando o único nó (o último nó) for retirado, devemos "recriá-la" como vazia.

```
// Modificação na exclusão:Desenfileira ou Dequeue
int dequeue (TFILA *pfila, Tno * pno){
    if (fila_vazia(pfila)
        return 0;
    *pno = pfila -> vnos[pfila -> F]
    (pfila -> F)++;
    if (pfila->F == pfila->R)     /* o úlltimo nó está saindo, portanto a
                                     fila  ficará vazia*/
        cria_fila_vazia(pfila);
    else
        (pfila -> F)++;
    return 1;
}
```

2. Fila circular

Problemas da fila linear:

Dada a seguinte fila linear:



Uma nova inclusão, embora haja espaço no buffer(vetor), acarretará em overflow **(** tentativa de inclusão em uma estrutura cheia). Como as inclusões só podem ser realizadas nos slots vazios após o último nó da fila (visualizamos o buffer como uma reta), os slots liberados pelas exclusões não podem ser reaproveitados pois estão no início do mesmo. Para reutilizar as posições iniciais, devemos imaginá-lo como um círculo. Com isso, a posição sucessora dos marcadores F e R dependerá da atual. Neste caso teremos:

1. *Próximo R será R + 1 caso o limite físico do buffer não seja atingido e 0 caso o seja.*
2. *Próximo F será F + 1 caso o limite físico do buffer não seja atingido e 0 caso o seja.*

```
int sucessor ( int max,int atual){
    if (atual == max - 1)
        return 0;
    else
        return atual + 1;
}
```

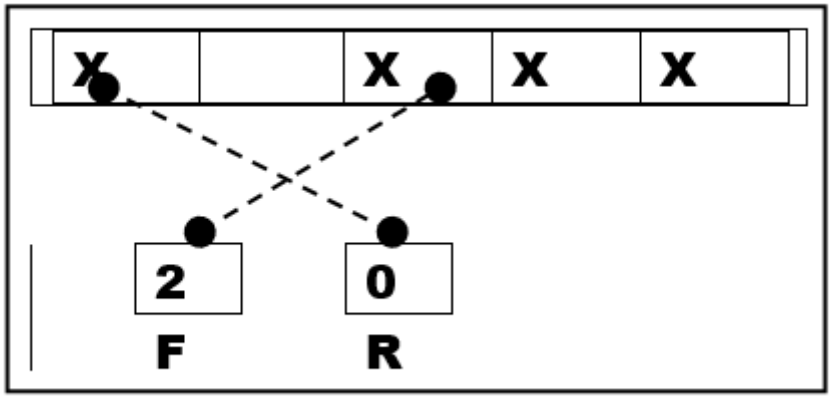
ou

```
int sucessor (int max,int atual){
    return (atual == max - 1)?  0 : atual + 1;
}
```

ou

```
int sucessor (int max, int atual){
    return (atual + 1)%max;
}
```

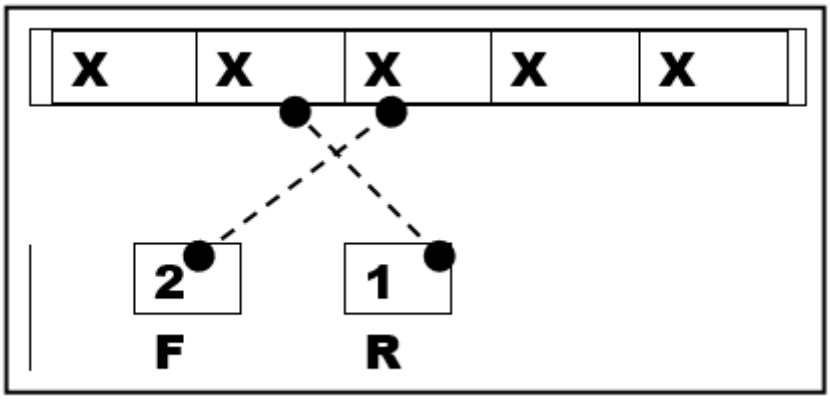
Desta forma, uma nova inclusão ficaria:



Embora tenhamos conseguido reaproveitar o espaço livre, introduzimos um problema: *o reconhecimento da fila vazia e da fila cheia:*

- **fila vazia**: na fila linear, quando o F > R a fila está vazia. Agora, no entanto, basta circular uma vez para o F ficar maior que o R mesmo a fila não estando vazia.
- **fila cheia**: na fila linear, se o R está na extremidade do buffer (limite físico) a fila está cheia. Agora, só o estará se não houver espaço livre no buffer

Como reconhecer uma fila cheia:



Se a próxima posição do R, isto é, onde o novo elemento da fila deve ser posicionado (atrás do atual último) está o F (atual primeiro) então a fila está cheia

```
int fila_cheia(const TFILA * pfilacirc){
    int proxR;

    proxR = sucessor(pfilacir->max,pfilacirc->R);
    return ((proxR == pfilacirc->F) && (pfilacirc->R != -1));
}
```

Como reconhecer uma fila vazia:

O único modo de reconhecer uma fila vazia (já que as posições relativas de F e R não indicam que está vazia) é quando a mesma é “virgem” , portanto, ao excluir o último elemento da fila, esta deve ser reinicializada

```
int fila_vazia(const TFILA * pfilacirc){
    return (pfilacirc->R == -1);
}
```

```
int Enqueue (TFILA *pfila, TINF * pinf){
    if (fila_cheia (pfila))
        return 0;

    pfila -> R =  sucessor(pfila->max,pfila->R);
    pfila-> vnos[pfila -> R] = *pinf;
    return 1;
}
```

```
int Dequeue (TFILA *pfila, TINF *pinf){
    if (fila_vazia(pfila))
        return 0;

    *pinf = pfila -> vnos[pfila -> F];

    if (pfila -> F == pfila -> R)
        cria_fila_vazia(pfila);
    else
        pfila -> F = sucessor(pfila->max,pfila -> F )
    return 1;
}
```

Fila circular com contador:

Para diminuir a complexidade do reconhecimento de fila cheia e vazia, poderíamos criar mais um campo no tipo TFILA que armazena com a quantidade de nós. Este campo deve ser atualizado a cada inclusão e exclusão.

O reconhecimento de:

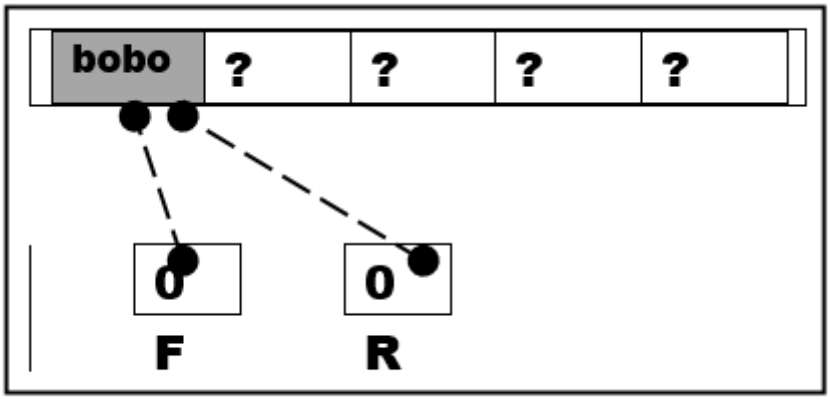
fila vazia: quantidade de nós ==0;

fila cheia: quantidade de nós == MAX.

Fila circular com nó bobo:

Na solução básica de fila circular embora o buffer seja reaproveitado, as posições relativas de F e R não indicam mais que a fila está vazia, obrigando a rotina de exclusão a recriar uma fila virgem quando retira o último elemento.

Para solucionar este problema, isto é, F > R não representar mais uma fila vazia, devemos representar a fila vazia de tal modo que não possa haver dúvida na posição dos indicadores (F e R). Isto só é possível se F e R apontarem para uma mesma posição do buffer, portanto, se F e R apontam para uma posição e a fila está vazia este nó inexiste e é chamado de **Bobo** ou **Dummy**



Há dois modos de implementarmos a fila circular com dummy: F marcando o bobo ou com R marcando o bobo.

1. Quando o R fica apontando para o dummy, incluímos o novo elemento na posição atual apontada por R, e R é então atualizado. Portanto, o **nó bobo** está sempre **no fim da Fila**, marcado por R

2. Quando o F fica apontando para o dummy, incluímos o novo elemento na posição sucessora do atual R, e R é então atualizado. Portanto, o **nó bobo,** está sempre no início da fila

IMPLEMENTAÇÃO COM DUMMY EM F

```
void cria_fila_vazia (TFILA *pfila){
    pfila -> F = pfila -> R = 0;
}
```

```c
int fila_vazia(const TFILA  * pfila){
    return (pfila->R == pfila->F)
}
```

```c
int fila_cheia(const TFILA  * pfilacirc){
    return (sucessor(pfila->max, pfilacirc->R) == pfilacirc->F)
}
```

```c
int Enqueue_Dummy (TFILA *pfila, TINF *pinf){
    if (fila_cheia (pfila))
        return 0;
    pfila -> R = sucessor((pfila->max,pfila -> R );
    pfila -> vnos[pfila -> R] = *pinf;
    return 1;
}
```

```c
int Dequeue_Dummy (TFILA *pfila, TINF *pinf){
    if (fila_vazia (pfila))
        return 0;

    pfila -> F = sucessor(pfila->max,pfila -> F);
    *pinf = pfila -> vnos[pfila -> F];
    return 1;
}
```