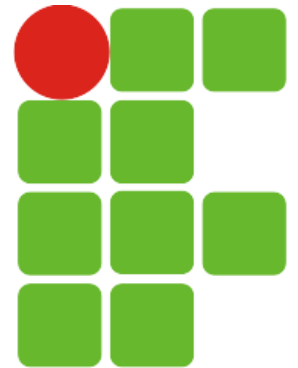


Estrutura de Dados

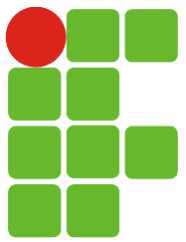
Profa. Marta Talitha Carvalho

Aula 5: FILA

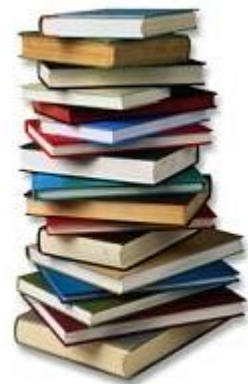


INSTITUTO FEDERAL
ESPÍRITO SANTO

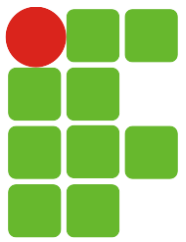
FILA



- O que diferencia a fila da pilha é a **ordem de saída dos elementos**, enquanto na pilha “O último que entra é o primeiro que sai.” , na fila “O primeiro que entra é o primeiro que sai.”
- Conhecido como FIFO – First In First Out

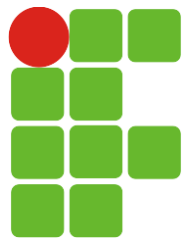


FILA



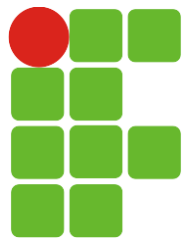
- A ideia fundamental da fila é que só podemos inserir um novo elemento no final da fila e só podemos retirar o elemento no início.
- A estrutura de fila é uma analogia natural com o conceito de fila que usamos no nosso dia a dia.

FILA



- **Um exemplo** de utilização na computação é a implementação de uma **fila de impressão**. Se uma impressora é compartilhada por várias máquinas, deve-se adotar uma estratégia para determinar que documento será impresso primeiro. A estratégia mais simples é tratar todas as requisições com a mesma prioridade e imprimir na ordem em que foram submetidos – O primeiro submetido é o primeiro a ser impresso.
- troca de mensagens entre computadores, atendimento de processos do sistema operacional, ordenação do encaminhamento dos pacotes de um roteador, buffer para gravação dos dados em mídia, etc.

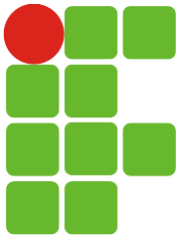
Qual é a diferença entre uma Lista e uma Fila?



- As operações de uma Fila são mais **restritas** do que as de uma Lista. Por exemplo, você pode adicionar ou remover um elemento em qualquer posição de uma Lista mas em uma Fila você só pode adicionar no fim e remover no início.
- Normalmente usa-se estrutura fila no problema quando o fator **tempo** influência nas operações.

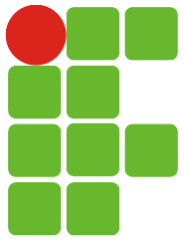
Lista é uma estrutura mais poderosa e mais genérica do que uma Fila. A Fila possui apenas um **subconjunto de operações** da Lista.

FILA



- Vamos discutir 2 estratégias para implementação de filas:
 - Usando vetor;
 - Usando lista encadeada;

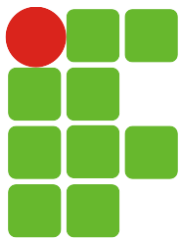
FILA



- Antes de discutir as estratégias de implementação, podemos definir quais as operações serão implementadas para manipular uma fila que armazena valores reais:
 - Criar uma estrutura fila;
 - Inserir um elemento no fim;
 - Retirar o elemento do início;
 - Verificar se a fila está vazia;
 - Liberar a fila;
- O arquivo fila.h, que representa a interface do tipo, pode conter o código:

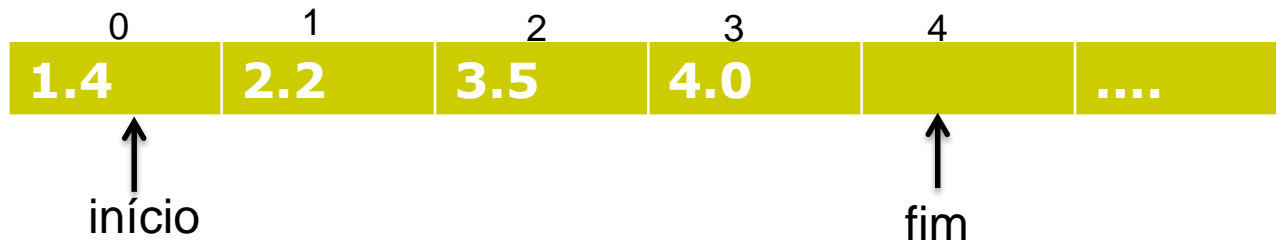
```
typedef struct fila Fila;  
Fila* cria();  
void insere(Fila* f, float v);  
float retira(Fila* f);  
int vazia(Fila* f);  
void libera(Fila* f);
```

FILA com vetor

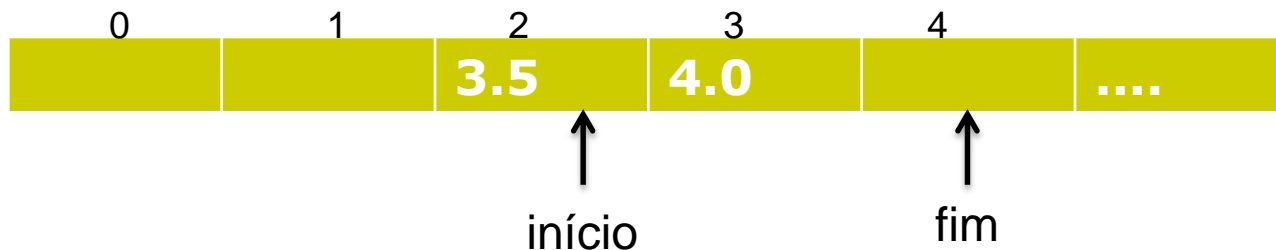


- Devemos ficar o número máximo N de elementos na fila. Podemos observar que processo de inserção e remoção em extremidades opostas fará com que a fila "ande" no vetor. Por exemplo, se inserirmos os elementos: 1.4, 2.2, 3.5, 4.0 e depois retirarmos dois elementos a fila não estará mais nas posições iniciais do vetor. Veja a figura seguinte:

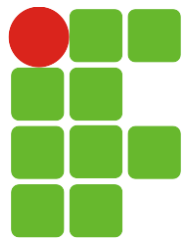
Após a inserção:



Após retirar elementos:

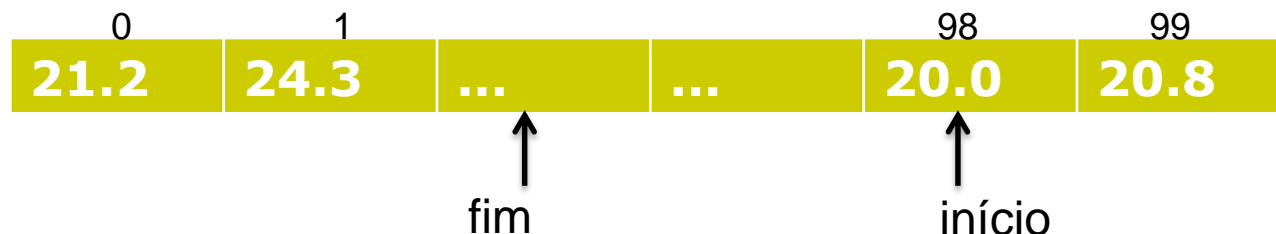


FILA com vetor

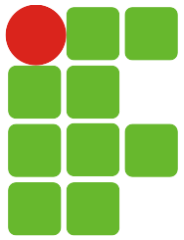


- Com essa estratégia, é fácil observar que, em dado instante, a parte ocupada do vetor pode chegar à última posição. Para reaproveitar as primeiras posições livres do vetor sem implementarmos uma re-arrumação trabalhosa dos elementos, podemos incrementar as posições do vetor de forma “circular” : se o último elemento da fila ocupa a última posição do vetor, inserimos os novos elementos a partir do início do vetor. Desta forma, em um dado momento poderíamos ter quatro elementos, 20.0, 20.8, 21.2 e 24.3 distribuídos. Veja a figura:

Fila com incremento circular;

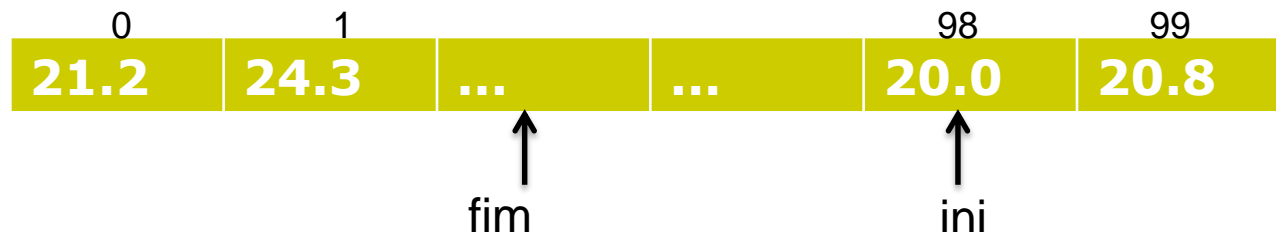


FILA com vetor

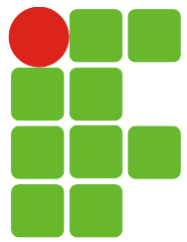


- Para essa implementação os índices do vetor são incrementados de maneira que seus valores progridam “circularmente”. Desta forma, se temos 100 posições no vetor, os valores dos índices assumem os seguintes valores:
- 0,1,2,3,...,98,99, 0,1,2,3,...,98,99,0,1....

Fila com incremento circular;



FILA com vetor



- Podemos definir uma função para auxiliar o valor do índice. Essa função recebe o valor do índice atual e fornece como valor de retorno o índice incrementado usando o incremento circular. Uma possível implementação dessa função é:

```
int incr( int i){  
    if (i==N-1)  
        return 0;  
    else  
        return i+1;  
}
```

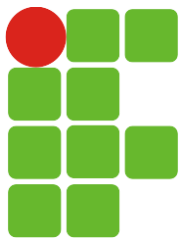


De uma forma mais compacta

```
int incr( int i){  
    return (i+1)%N;  
}
```

Fila com incremento circular:

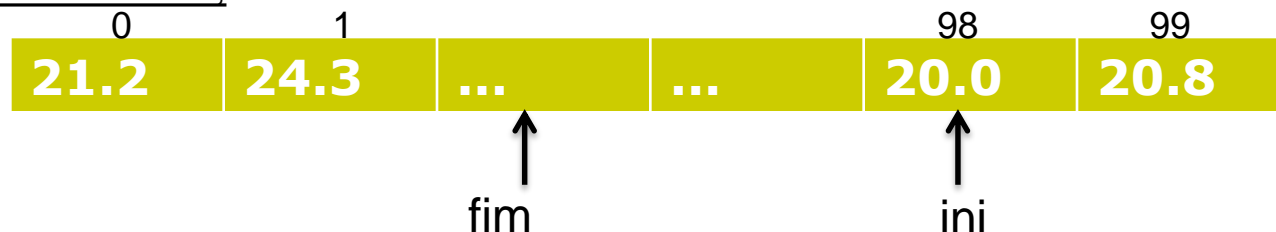


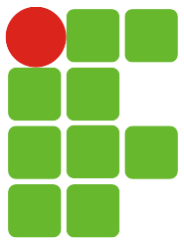


FILA com vetor

- Podemos declarar o tipo fila como sendo uma estrutura com 3 componentes:
 - Um vetor vet;
 - De tamanho N;
 - Um índice INI;
 - Um índice FIM;
- ENTÃO:
 - INI – marca a posição do próximo elemento a ser retirado da fila.
 - FIM – marca a posição, onde será inserido o próximo elemento.

Fila com incremento circular;

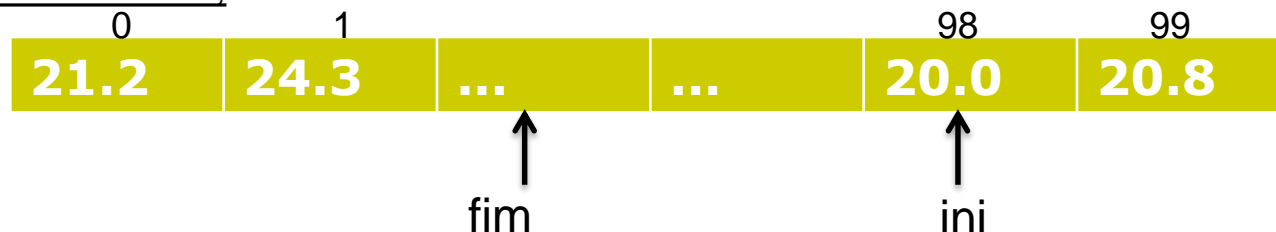




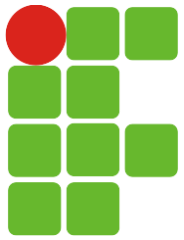
FILA com vetor

- Como sabemos que a fila está cheia ou vazia?
 - $\text{Ini} == \text{fim} \Rightarrow$ fila vazia
 - $\text{incr}(\text{fim}) == \text{ini} \Rightarrow$ fila cheia . Fim e ini em posições consecutivas.
- Note que , com essas convenções a posição indicada por fim permanece sempre vazia, de forma que o número máximo de elementos na fila é N-1. Isto é necessário porque a inserção de mais de um elemento, faria $\text{ini} == \text{fim}$, e haveria uma ambiguidade entre fila cheia e fila vazia. Outra estratégia possível consiste em armazenar uma informação adicional, n, que indicaria explicitamente o número de elementos armazenados na fila. Assim a fila estaria vazia se $n == 0$ e cheia se $n == N-1$. Nos exemplos que se seguem, optamos por não armazenar n explicitamente.

Fila com incremento circular;



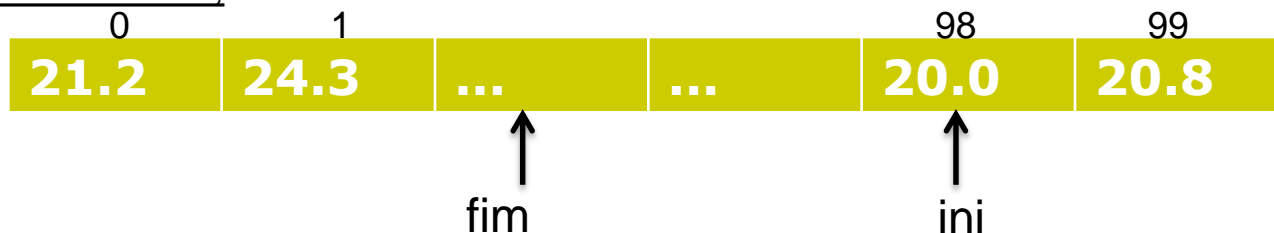
FILA com vetor



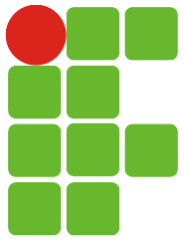
- A estrutura da fila:

```
#define N 100  
struct fila {  
    int ini, fim;  
    float vet[N];  
}
```

Fila com incremento circular;



FILA com vetor



- Fila* cria (void)

{

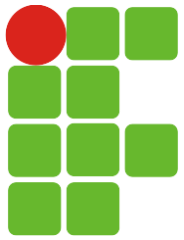
Fila* f = (Fila*) malloc(sizeof(Fila));

f->ini = f->fim = 0; /* inicializa fila vazia */

return f;

}

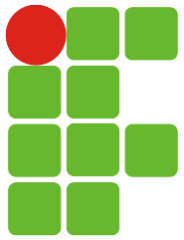
FILA com vetor



```
void insere (Fila* f, float v){
    /* fila cheia: capacidade esgotada*/
    if (incr(f->fim) == f->ini) {
        printf("Capacidade da fila estourou.\n");
        exit(1); /* aborta programa */
    }

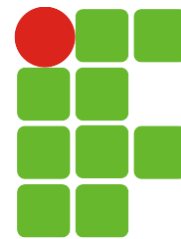
    /* insere elemento na próxima posição livre */
    f->vet[f->fim] = v;
    f->fim = incr(f->fim);
}
```


FILA com vetor



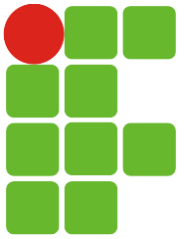
```
float retira (Fila* f)
{
    float v;
    if (vazia(f)) {
        printf("Fila vazia.\n");
        exit(1); /* aborta programa */
    }
    /* retira elemento do início */
    v = f->vet[f->ini];
    f->ini = incr(f->ini);
    return v;
}
```

Fila com vetor



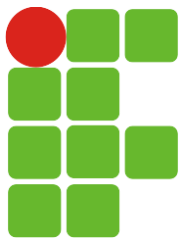
```
int vazia (Fila* f)
{
    return (f->ini == f->fim);
}
```

Fila com vetor



```
void libera (Fila* f)
{
    free(f);
}
```

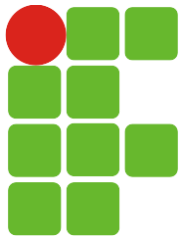
FILA com lista encadeada



Vamos agora ver como implementar uma fila através de uma lista encadeada, que será, como nos exemplos anteriores, uma lista simplesmente encadeada, em que cada nó guarda um ponteiro para o próximo nó da lista. Como teremos que inserir e retirar elementos das extremidades opostas da lista, que representarão o início e o fim da fila, teremos que usar dois ponteiros, ini e fim, que apontam respectivamente para o primeiro e para o último elemento da fila. Essa situação é ilustrada a seguir:



FILA com lista encadeada

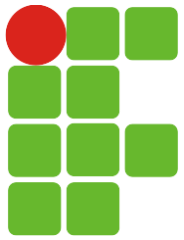


A operação para retirar um elemento se dá no início da lista(fila) e consiste essencialmente em fazer com que, após a remoção, ini aponte para o sucessor do nó retirado.

A remoção também é simples, pois basta acrescentar à lista um sucessor para o último nó, apontando para fim, e fazer com que fim aponte para este novo nó.



FILA com lista encadeada



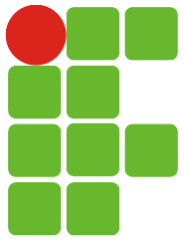
O nó da lista para armazenar valores reais pode ser:

```
struct no{  
    float info;  
    struct no* prox;  
};  
typedef struct no No;
```

```
struct fila {  
    No* ini;  
    No* fim;  
}  
  
typedef struct fila Fila;
```



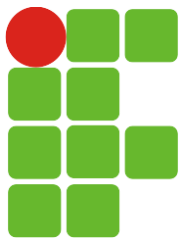
FILA com lista encadeada



A função cria aloca a estrutura da fila e inicializa a lista vazia.

```
Fila* inicializa(){  
    Fila* f = (Fila*) malloc(sizeof(Fila));  
    f->ini = f->fim = NULL;  
    return f;  
}
```





FILA com lista encadeada

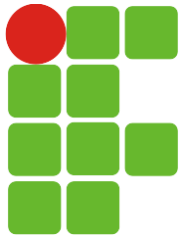
Cada novo elemento é inserido no fim da lista e, sempre que solicitado, retiramos o elemento do início da lista. Dessa forma, precisamos de 2 funções auxiliares de lista:

- Para inserir no fim
- Para remover do início

A função para inserir no fim deve retornar o novo “fim”. A função para retirar do início é idêntica à função usada na implementação da pilha.



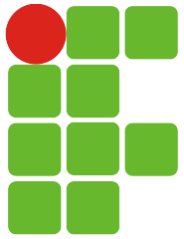
FILA com lista encadeada



```
No* ins_fim(No* fim, float v){
    No* p = (No*) malloc(sizeof(No));
    p->info = v;
    p->prox = NULL;
    if(fim!=NULL){
        /*verifica se a lista não está vazia*/
        fim->prox = p;
    }
    return p;
}

void insere(Fila* f, float v){
    f->fim = ins_fim(f->fim, v);
    if ( f->ini==NULL)
        f->ini= f->fim;
}
```

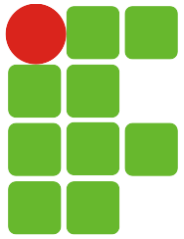
FILA com lista encadeada



```
No* ret_ini(No* ini){
    No* p = ini->prox;
    free(ini);
    return p;
}

Float retira(Fila* f){
    float v;
    if( vazia(f)){
        printf("Fila vazia \n");
        exit(1);
    }
    v = f->ini->info;
    f->ini = ret_ini(f->ini);
    if (f->ini == NULL) { //removeu o último
        f->fim = NULL;
    }
    return v;
}
```

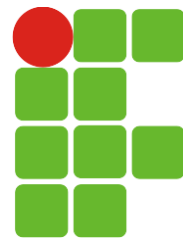
FILA com lista encadeada



```
/*Verifica se a fila está vazia*/  
int vazia(Fila* f){  
    return (f->ini == NULL);  
}
```

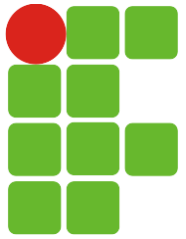
```
/*libera todos os elementos*/  
void libera(Fila* f) {  
    No* q = f-> ini;  
    While (q!=NULL) {  
        No* t = q->prox;  
        free(q);  
        q =t;  
    }  
    free(f);  
}
```

FILA com lista encadeada



```
/*Imprime a fila*/  
void imprime(Fila* f) {  
    No* q;  
    for(q=f->ini; q!=NULL; q= q->prox){  
        printf("%f\n", q->info);  
    }  
}
```

FILA DUPLA

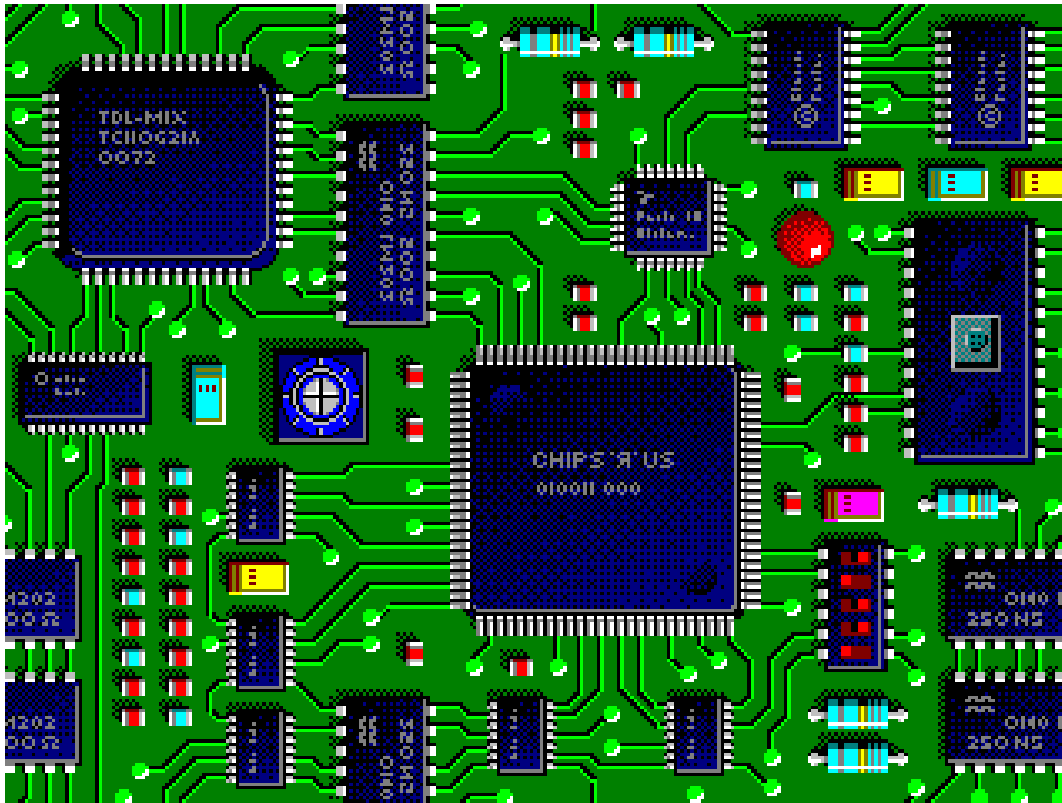
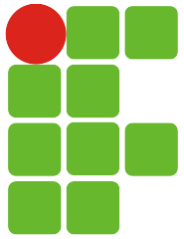


Estrutura de uma fila na qual é possível inserir novos elementos em ambas as extremidades no início e fim. Consequentemente permite-se também retirar elementos de ambos os extremos.

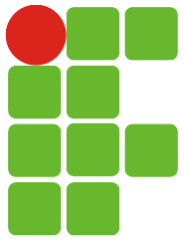
Outros tipos de fila

- Fila circulares
- Fila de prioridade

Problema: Wire Routing

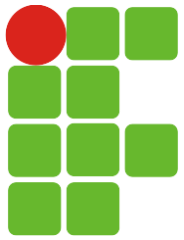




Usando Fila

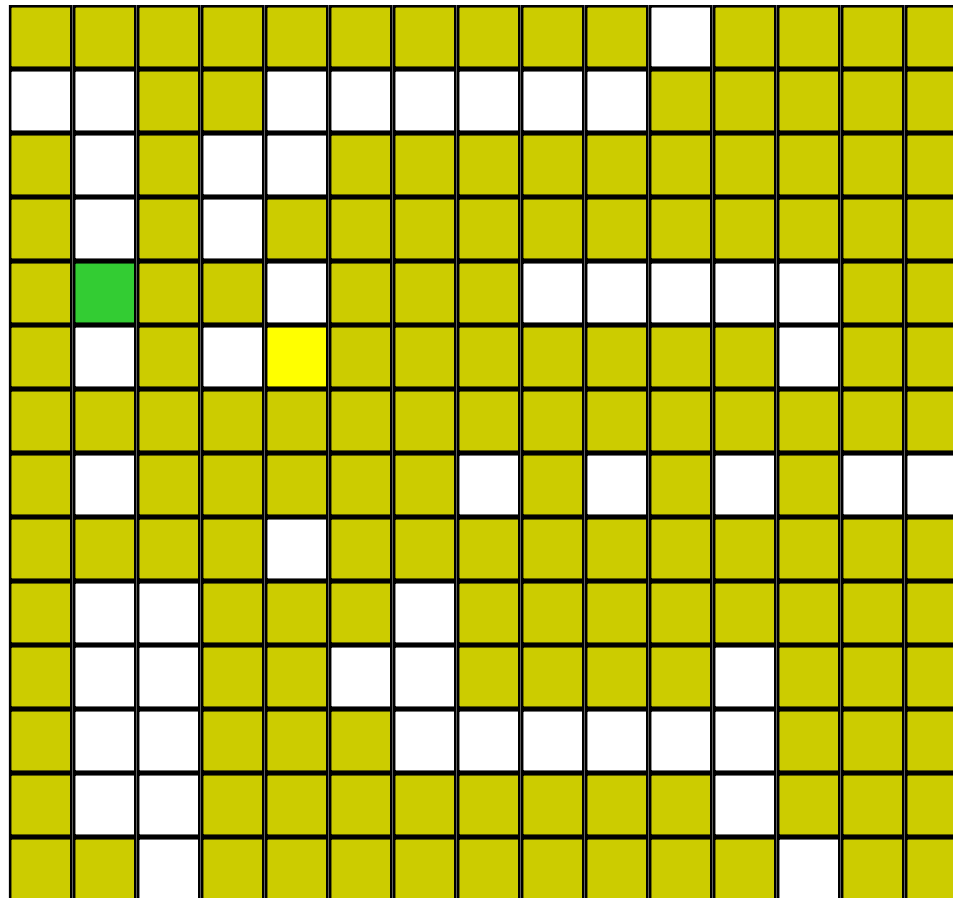


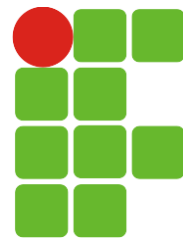
1. Uma fila de “quadrados livres” é usado.
2. A fila é inicialmente vazia, e a célula “Inicio” é examinada.
3. As células adjacentes são examinadas são marcadas com suas distâncias e adicionadas a fila.
4. Enquanto a fila não estiver vazia e não se tiver atingido o ponto de destino:
 - Então a célula é removida da fila e feito uma nova análise da célula. Numerar os seus vizinhos livres com o número do ponto acrescido de 1;
 - Colocar os novos pontos (vizinhos) na fila;
5. Este processo é repetido até que a célula “Fim” é alcançada ou a fila fique vazia.

Solução: Lee's Wire Router




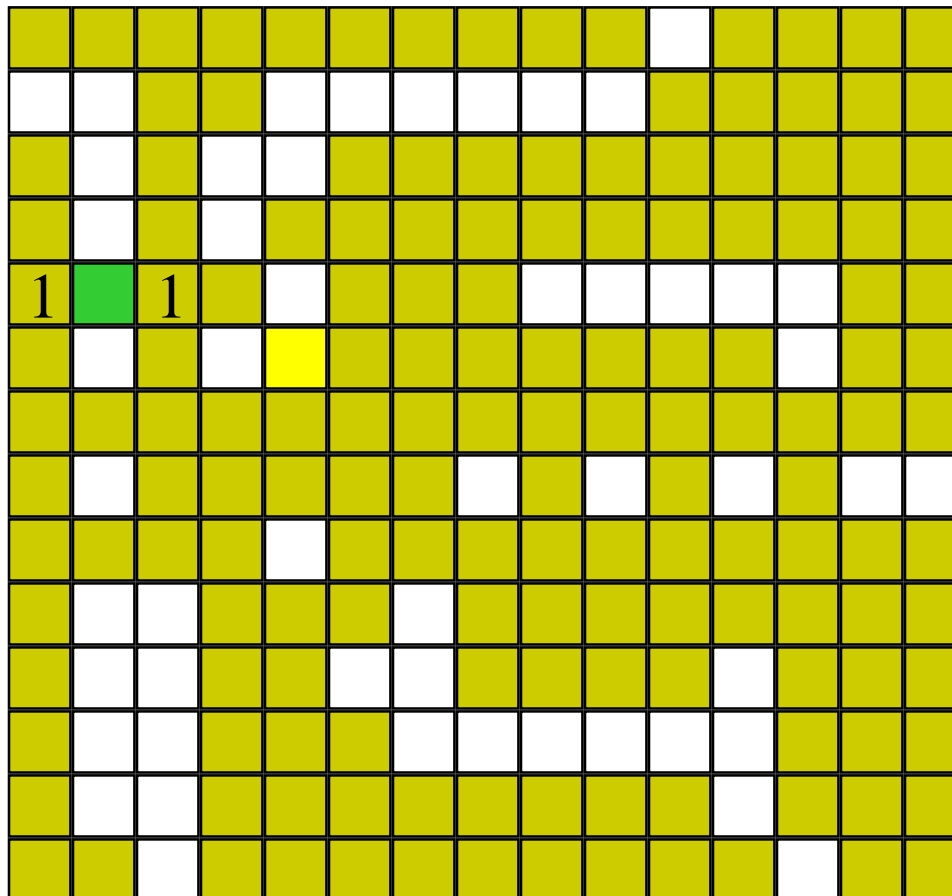
-  Início
-  Fim
-  Bloqueado
-  Livre





Lee's Wire Router

-  Início
-  Fim
-  Bloqueado
-  Livre

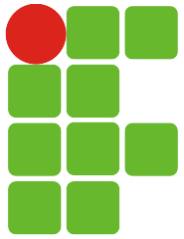


[illegible]

[illegible]

[illegible]

Lee's Wire Router



Início



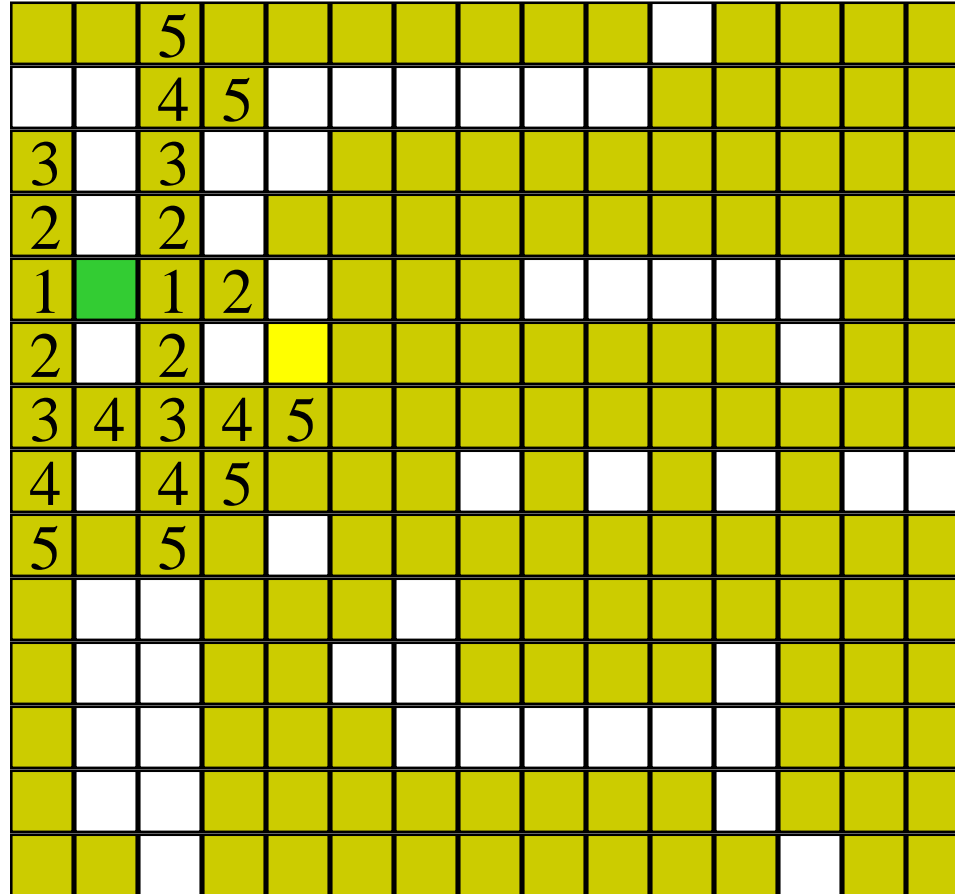
Fim

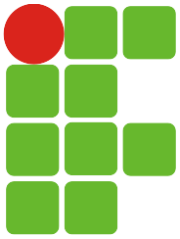


Bloqueado







Livre



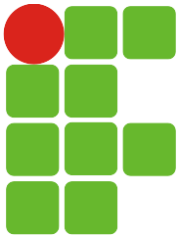


Lee's Wire Router





-  Início
-  Fim
-  Bloqueado
-  Livre

		6	5	6										
			4	5										
3			3											
2			2											
1			1	2										
2			2		6									
3	4		3	4	5	6								
4			4	5	6									
5	6		5	6										
6														

End pin reached. Traceback.



Lee's Wire Router

-  Início
-  Fim
-  Bloqueado
-  Livre

	6	5	6												
		4	5												
3		3													
2		2													
1		1	2												
2		2		6											
3	4	3	4	5	6										
4		4	5	6											
5	6	5	6												
6															

End pin reached. Traceback.