ESTRUTURAS DE DADOS

Fila (Vetores)

Roteiro

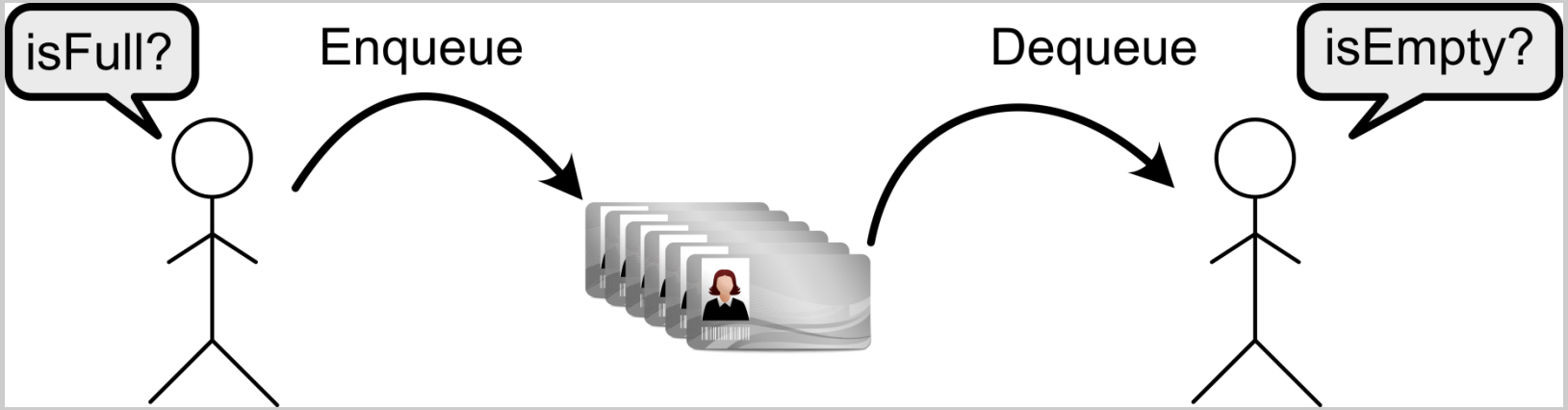
- **Motivação**
- **Tipo Abstrato de Dados**
- **Aplicações da Estrutura**
- **Detalhes de Implementação**

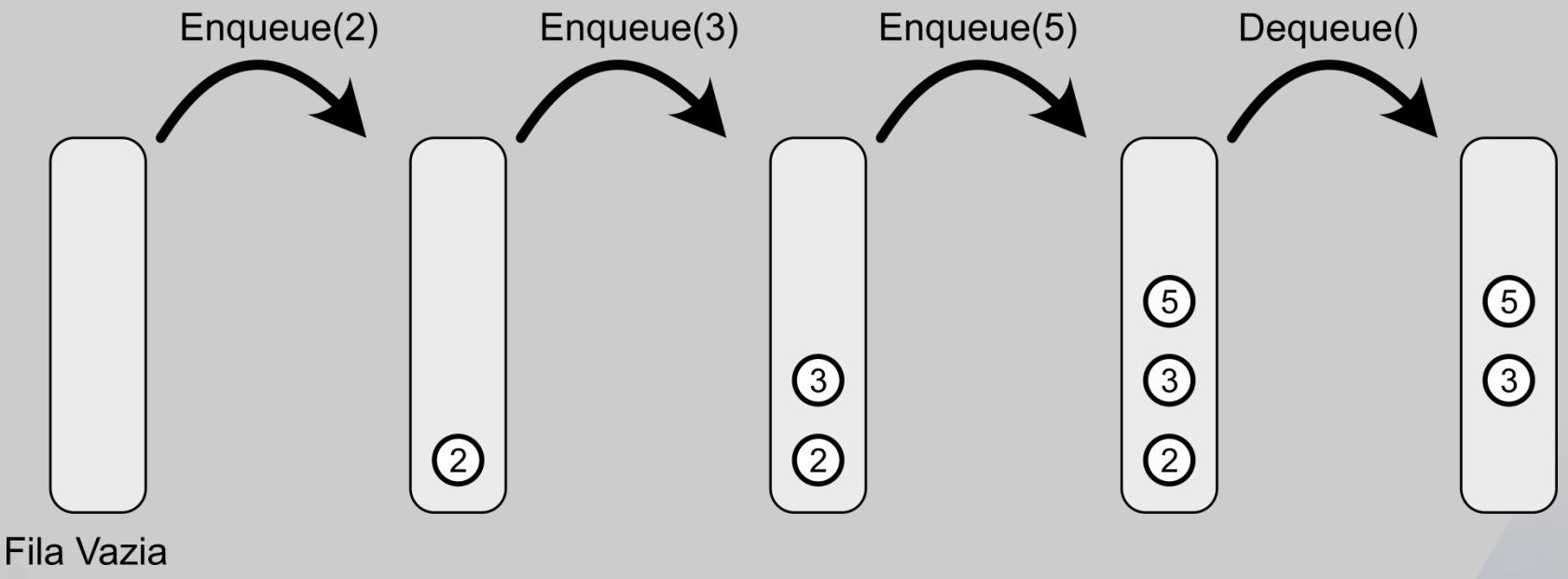
Motivação

Uma fila é uma estrutura linear na qual as inserções ocorrem no final e as exclusões ocorrem no início.

Suponha que pacientes para obter atendimento médico precisam entregar a identidade para uma secretária:

- **A secretária deve receber a identidade e colocar o paciente recém-chegado no final da fila.**
- **Se tem muito paciente, a secretária faz uma pausa.**
- **O médico chamará para atendimento o paciente no início da fila (seria um caos se usasse uma pilha).**
- **Se a fila estiver vazia, o médico faz uma pausa.**


isFull?
Enqueue
Dequeue
isEmpty?


Enqueue(2)
Enqueue(3)
Enqueue(5)
Dequeue()
2
3
2
5
3
2
5
3
Fila Vazia

Resumindo

- **O primeiro elemento a entrar na estrutura tem que ser o primeiro a sair.**
- **O último elemento a entrar tem que ser o último a sair.**
- **Comportamento parecido com a comunicação de processos ou acesso a algum recurso.**
- **Inserções ocorrem no final e remoções ocorrem no início.**

Tipo Abstrato de Dados

```
class Queue
{
 public :
  Queue(); // Constructor
  ~Queue(); // Destrutor
  bool isEmpty() const;
  bool isFull() const;
  void print() const;

  void enqueue(ItemType);
  ItemType dequeue();
 private:
  int front;
  int back;
  ItemType* structure;
};
```

Aplicações da Estrutura

Uma fila é uma estrutura bastante útil, principalmente quando precisamos garantir que processos acessarão recursos compartilhados de uma maneira justa.

- **Documentos enviados para a impressão.**
- **Troca de mensagens entre processos em um Sistema Operacional.**

Detalhes de Implementação

Implementaremos uma fila como um vetor.

A posição do elemento na frente da fila será indicada por uma variável inteira.

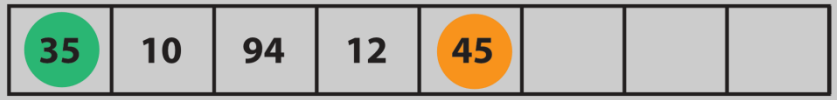
A posição do elemento atrás da fila será indicada por uma segunda variável inteira.

Queremos que inserções e remoções ocorram em tempo constante.

Enqueue: 35

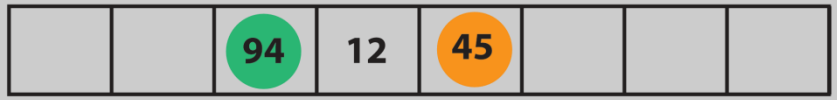
Enqueue: 10

Enqueue: 94, 12, 45

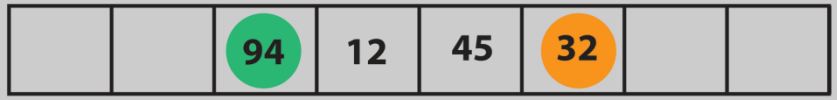


Dequeue

Dequeue



Enqueue: 32



A fila caminha pelo vetor, ocupando o final do vetor e desocupando o início

O tamanho do vetor pode ser obtido subtraindo o índice do elemento da frente pelo índice do elemento que está atrás.

Podemos imaginar o vetor como circular.

O índice "real" pode ser obtido com o resto da divisão pelo tamanho do vetor.

Implementando essa ideia, temos:

- **Construtor e Destrutor**

```
Queue::Queue()
{
  front = 0;
  back  = 0;
  structure = new ItemType[MAX_ITEMS];
}

Queue::~Queue()
{
  delete [] structure;
}
```

Implementando essa ideia, temos:

- **Verificação de cheio ou vazio.**

```
bool Queue::isEmpty() const
{
  return (front == back);
}

bool Queue::isFull() const
{
  return (back - front == MAX_ITEMS);
}
```

Implementando essa ideia, temos:

- **Inserindo elementos**

```
void Queue::enqueue(ItemType item)
{
  if (!isFull()){
    structure[back % MAX_ITEMS] = item;
    back++;
  } else {
    throw "Queue is already full!";
  }
}
```

Implementando essa ideia, temos:

- **Removendo elementos**

```
ItemType Queue::dequeue()
{
  if (!isEmpty()){
    front++;
    return structure[(front-1) % MAX_ITEMS];
  } else {
    throw "Queue is empty!";
  }
}
```

Implementando essa ideia, temos:

- **Imprimindo a fila na saída padrão**

```
void Queue::print() const
{
  cout << "Fila = ";
  for (int i = front; i < back; i++) {
    cout << structure[i % MAX_ITEMS];
  }
  cout << endl;
}
```

Usando a estrutura:

```
char character;
Queue queue;
char queueChar;
cout << "Enter a string; press return." << endl;
cin.get(character);
while (character != '\n' and !queue.isFull())
  {
    queue.enqueue(character);
    cin.get(character);
  }
while (!queue.isEmpty())
  {
    queueChar = queue.dequeue();
    cout << queueChar;
  }
cout << endl;
```

ESTRUTURAS DE DADOS

Fila (Vetores)