Fila e Deque

SCC0202 - Algoritmos e Estruturas de Dados I

Prof. Fernando V. Paulovich
**Baseado no material do Prof. Gustavo Batista*
http://www.icmc.usp.br/~paulovic
paulovic@icmc.usp.br

Instituto de Ciências Matemáticas e de Computação (ICMC)
Universidade de São Paulo (USP)

23 de outubro de 2017

Sumário

Sumário

TAD Fila

- Filas são estruturas nas quais as inserções são feitas em um extremo (final) e remoções são feitas no outro extremo (início)
  - Política "First-in, First-out" (FIFO)
- Modelos intuitivos de filas são as linhas para comprar bilhetes de cinema e de caixa de supermercado

Aplicações

- Exemplos de aplicações de filas
  - Filas de espera e algoritmos de simulação
  - Controle por parte do sistema operacional a recursos compartilhados, tais como impressoras
  - Buffers de Entrada/Saída
  - Estrutura de dados auxiliar em alguns algoritmos como a busca em largura

TAD Fila

Operações principais

- *enfileirar(F,x)*: insere o elemento x no final da fila F. Retorna **true** se foi possível inserir **false** caso contrário
- *desenfileirar(F)*: remove o elemento no inicio de F, e retorna esse elemento. Retorna **null** se não foi possível remover

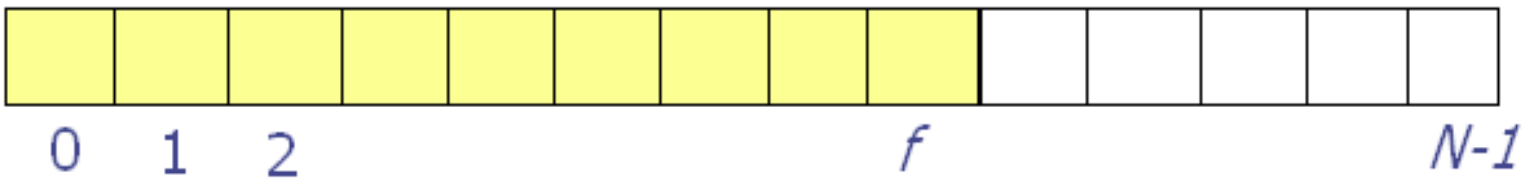
TAD Fila

Operações auxiliares

- *frente(F)*: retorna o elemento no início de F, sem remover
- *tamanho(F)*: retorna o número de elementos em F
- *vazia(F)*: indica se a fila F está vazia
- *cheia(F)*: indica se a fila F está cheia (útil para implementações estáticas)

Implementação Estática

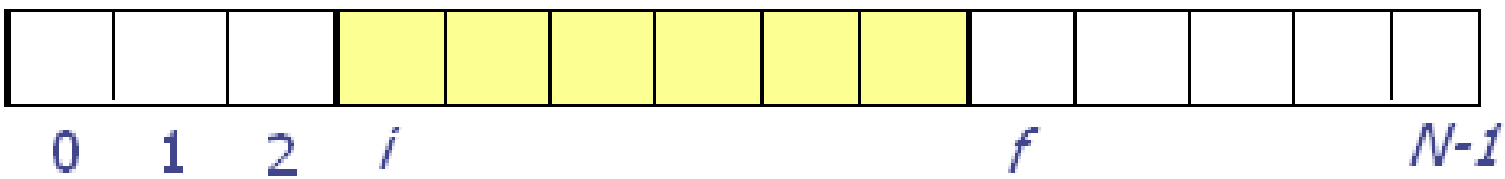
- Uma maneira simples de implementar uma fila é fixar o início da fila na primeira posição do vetor



- As inserções (enfileira) fazem com que o contador *f* seja incrementado ($O(1)$)
- Mas as remoções (desenfileira) requerem deslocar todos os elementos ($O(n)$)
- É possível melhorar isso?

Implementação Estática Circular

- A solução é fazer com que o início não seja fixo na primeira posição do vetor
- Portanto, deve-se manter dois contadores para o início (i) e final da fila (f)



- Mas o que deve ser feito quando $f = N$ e deseja-se inserir mais um elemento?

Implementação Estática Circular

- Pode-se permitir que f volte para o início do vetor quando esse contador atingir o final do vetor
- Essa implementação é conhecida como fila circular
- Na figura abaixo a fila circular possui 4 posições vagas e $f < i$

Definição de Tipos

```
1  typedef struct fila_estatica FILA_ESTATICA;
2
3  #define TAM 100
4
5  struct fila_estatica {
6    ITEM * vetor[TAM];
7    int inicio;
8    int fim;
9  };
```

Implementação Estática Circular

- Caso o **inicio** aponte para o primeiro elemento inserido e o **fim** aponte para o último, não é possível distinguir se uma fila tem um elemento ou está vazia
- Uma estratégia seria apontar o **fim** para a próxima posição de inserção
- Assim, se a próxima posição do **fim** for igual ao **inicio**, a fila está cheia
    - Perde-se uma posição no vetor (a fila terá **TAM-1** posições de inserção)

Implementação Estática Circular

```
1  FILA_ESTATICA *criar_fila() {
2    FILA_ESTATICA *fila = (FILA_ESTATICA*)malloc(sizeof(FILA_ESTATICA));
3
4    if (fila != NULL) {
5      fila->fim = 0;
6      fila->inicio = 0;
7    }
8
9    return fila;
10 }
11
12 int cheia(FILA_ESTATICA *fila) {
13   return ((fila->fim + 1) % TAM == fila->inicio);
14 }
15
16 int vazia(FILA_ESTATICA *fila) {
17   return (fila->inicio == fila->fim);
18 }
```

Implementação Estática Circular

```
1  int enfileirar(FILA_ESTATICA *fila, ITEM *item) {
2    if (!cheia(fila)) {
3      fila->vetor[fila->fim] = item;
4      fila->fim = (fila->fim + 1) % TAM;
5    }
6    return 0;
7  }
```

Implementação Estática Circular

```
1  ITEM *desenfileirar(FILA_ESTATICA *fila) {
2    if (!vazia(fila)) {
3      ITEM *ret = fila->vetor[fila->inicio];
4      fila->inicio = (fila->inicio + 1) % TAM;
5      return ret;
6    }
7    return NULL;
8  }
```

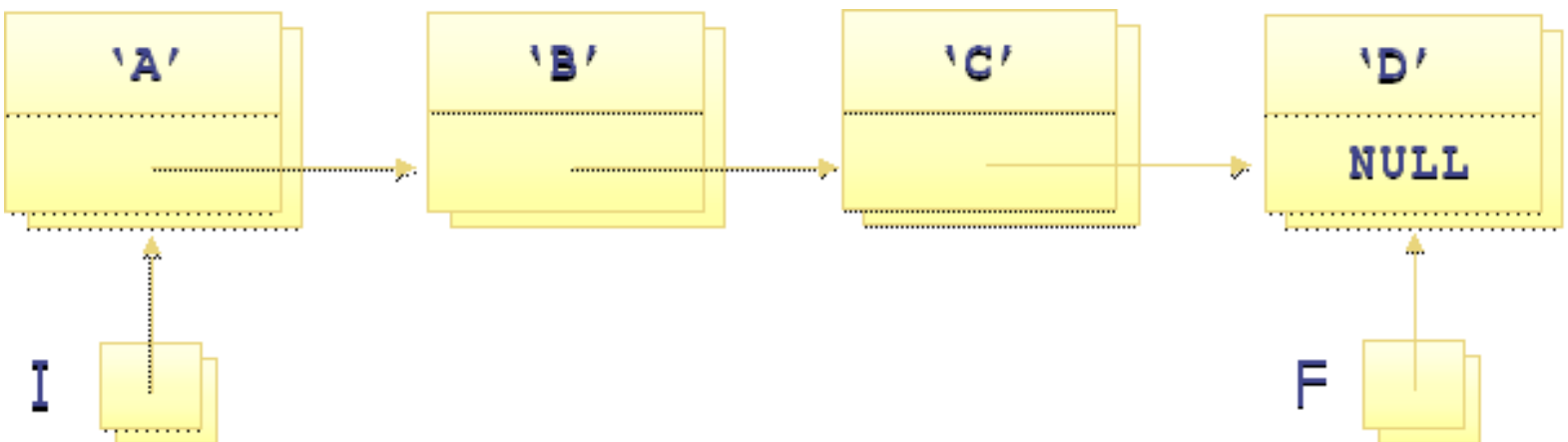
Implementação Estática Circular

```
1  int tamanho(FILA_ESTATICA *fila) {
2    if (fila->inicio <= fila->fim) {
3      return fila->fim - fila->inicio;
4    } else {
5      return TAM - fila->inicio + fila->fim;
6    }
7  }
```

.

Implementação Dinâmica

- A implementação dinâmica pode ser realizada mantendo-se dois ponteiros, um para o início e outro para o final da fila
- Com isso pode-se ter acesso direto às posições de inserção e remoção



Implementação Dinâmica

- As operações enfileira e desenfileira implementadas dinamicamente são bastante eficientes
- Deve-se tomar cuidado apenas para que os ponteiros para início e final da fila tenham valor *NULL* quando ela estiver fazia

Estática versus Dinâmica

Operação	Estática	Dinâmica
Criar	$O(1)$	$O(1)$
Apagar	$O(n)$	$O(n)$
Enfileirar	$O(1)$	$O(1)$
Desenfileirar	$O(1)$	$O(1)$
Frente	$O(1)$	$O(1)$
Vazia	$O(1)$	$O(1)$
Cheia	$O(1)$	$O(1)$
Tamanho	$O(1)$	$O(1)$ (c/ contador)

Estática versus Dinâmica

Estática

- Implementação simples
- Tamanho da fila definido a priori

Dinâmica

- Alocação dinâmica permite gerenciar melhor estruturas cujo tamanho não é conhecido a priori ou que variam muito de tamanho

Sumário

TAD Deques

- Deques são estruturas que permitem inserir e remover de ambos os extremos

TAD Deques

- Operações principais
  - *inserir_inicio(D,x)*: insere o elemento x no início do deque D. Retorna **true** se foi possível inserir **false** caso contrário
  - *inserir_fim(D,x)*: insere o elemento x no final do deque D. Retorna **true** se foi possível inserir **false** caso contrário
  - *remover_inicio(D)*: remove o elemento no inicio de D, e retorna esse elemento. Retorna **null** se não foi possível remover
  - *remover_fim(D)*: remove o elemento no final de D, e retorna esse elemento. Retorna **null** se não foi possível remover

TAD Deques

- Operações auxiliares
  - *primeiro(D)*: retorna o elemento no início de D. Retorna **null** se o elemento não existe
  - *ultimo(D)*: retorna o elemento no final de D. Retorna **null** se o elemento não existe
  - *contar(D)*: retorna o número de elementos em D
  - *vazia(D)*: indica se o deque D está vazio
  - *cheia(D)*: indica se o deque D está cheio (útil para implementações estáticas)

TAD Deques

- Como deques requerem inserir e remover elementos em ambos os extremos
  - Implementação estática circular
  - Implementação dinâmica duplamente encadeada
- Nesses casos, as operações do TAD são $O(1)$

Implementação Estática Circular

- Aproveita-se a implementação de uma lista circular estática e acrescenta as duas operações que faltam: (1) remover do fim; e (2) inserir no início
- As outras operações são as mesmas

Implementação - Inserir no Início

```
1  int inserir_inicio(DEQUE_ESTATICA *deque, ITEM *item) {
2    if (!cheia(deque)) {
3      deque->inicio = (deque->inicio - 1 + TAM) % TAM;
4      deque->vetor[deque->inicio] = item;
5      return 1;
6    }
7    return 0;
8  }
```

- Se $x \geq 0$, então $(x + N)\ \%\ N = x$
- Se $x = -1$, então $(x + N)\ \%\ N = N - 1$

Implementação - Remover do Fim

```
1  ITEM *remover_fim(DEQUE_ESTATICA *deque) {
2    if (!vazia(deque)) {
3      deque->fim = (deque->fim - 1 + TAM) % TAM;
4      return deque->vetor[deque->fim];
5    }
6    return NULL;
7  }
```

- Se $x \geq 0$, então $(x + N)\ \%\ N = x$
- Se $x = -1$, então $(x + N)\ \%\ N = N - 1$

Exercícios

- Implemente uma fila dinâmica
- Implemente uma deque dinâmica

- Implemente um procedimento recursivo capaz de esvaziar uma fila
- Implemente um procedimento para inverter uma fila (o primeiro elemento se tornará o último e vice-versa)