

Fila e Deque

Estruturas de Dados I

Departamento de Computação

Universidade Federal de São Carlos (UFSCar)

Sumário

1 Filas

2 Deques

Sumário

1 Filas

2 Deques

TAD Fila

- Filas são estruturas lineares nas quais as inserções são feitas em um extremo (final) e remoções são feitas no outro extremo (início)
- Seguem a 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 início de F , e retorna esse elemento. Retorna **true** se foi possível remover **false** caso contrário

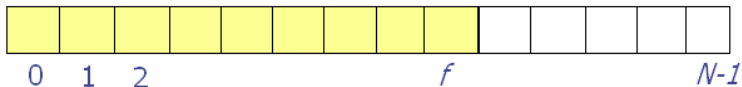
TAD Fila

Operações auxiliares

- $frente(F)$: retorna o elemento no início de F , sem remover
- $contar(F)$: 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

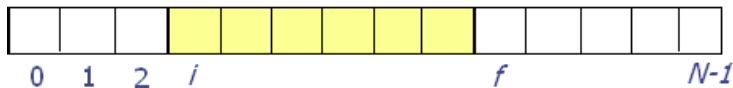
- Uma maneira simples de implementar uma fila é fixar o início da fila na primeira posição do vetor



- 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

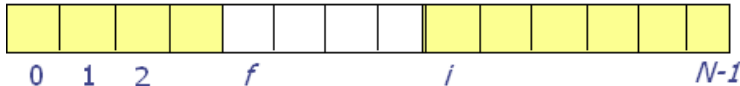
- A solução é fazer com que o início não seja fixo na primeira posição do vetor
- Portanto, deve-se manter dois contadores para o início (i) e final da fila (f)



- 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  #define TAM 100
2
3  typedef struct {
4      int valor;
5  } ITEM;
6
7  typedef struct {
8      ITEM itens[TAM];
9      int fim;
10     int inicio;
11 } FILA_ESTATICA;
```

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 está cheia ou 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 void criar(FILA_ESTATICA *fila) {  
2     fila->fim = 0;  
3     fila->inicio = 0;  
4 }  
5  
6 int vazia(FILA_ESTATICA *fila) {  
7     return (fila->inicio == fila->fim);  
8 }  
9  
10 int cheia(FILA_ESTATICA *fila) {  
11     return (((fila->fim+1) % TAM) == fila->inicio);  
12 }
```

Implementação Estática Circular

```
1  int enfileirar(FILA_ESTATICA *fila, ITEM *item) {  
2      if (!cheia(fila)) {  
3          fila->itens[fila->fim] = *item; //insiro o item  
4          fila->fim = (fila->fim+1) % TAM; //avanço o fim para a próxima ←  
           posição vazia  
5          return 1;  
6      }  
7      return 0;  
8  }
```

Implementação Estática Circular

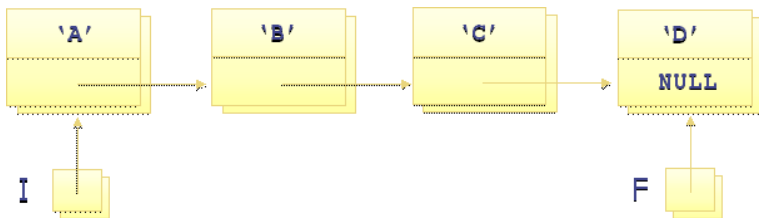
```
1  int desenfileirar(FILA_ESTATICA *fila, ITEM *item) {  
2      if (!vazia(fila)) {  
3          *item = fila->itens[fila->inicio]; //recupero o primeiro item  
4          fila->inicio = (fila->inicio+1) % TAM; //avanço o fim para a próxima←  
              posição vazia  
5          return 1;  
6      }  
7      return 0;  
8  }
```

Implementação Estática Circular

```
1 int contar(FILA_ESTATICA *fila) {  
2     if (fila->fim >= fila->inicio) {  
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 vazia

Estática versus Dinâmica

Operação	Estática	Dinâmica
Cria	$O(1)$	$O(1)$
Enfileira	$O(1)$	$O(1)$
Desenfileira	$O(1)$	$O(1)$
Frente	$O(1)$	$O(1)$
Vazia	$O(1)$	$O(1)$
Conta	$O(1)$	$O(1)$ (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

1 Filas

2 Deques

TAD Deques

- Deques são estruturas lineares que permitem inserir e remover de ambos os extremos
- 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 início de D , e retorna esse elemento. Retorna **true** se foi possível remover **false** caso contrário
 - *remover_fim(D)*: remove o elemento no final de D , e retorna esse elemento. Retorna **true** se foi possível remover **false** caso contrário

TAD Deques

- Operações auxiliares
 - *primeiro*(D): retorna o elemento no início de D . Retorna **true** se o elemento existe **false** caso contrário
 - *ultimo*(D): retorna o elemento no final de D . Retorna **true** se o elemento existe **false** caso contrário
 - *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 deque 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;  
4          if (deque->inicio < 0) {  
5              deque->inicio = TAM - 1;  
6          }  
7  
8          deque->itens[deque->inicio] = *item; //insiro o item  
9  
10         return 1;  
11     }  
12  
13     return 0;  
14 }
```

Implementação - Remover do Fim

```
1  int remover_fim(DEQUE_ESTATICA *deque, ITEM *item) {  
2      if (!vazia(deque)) {  
3          deque->fim = deque->fim-1;  
4          if (deque->fim < 0) {  
5              deque->fim = TAM - 1;  
6          }  
7  
8          *item = deque->itens[deque->fim]; //recupero o primeiro item  
9  
10         return 1;  
11     }  
12  
13     return 0;  
14 }
```

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)