Estrutura de Dados

Hamilton José Brumatto

Bacharelado em Ciências da Computação - UESC

5 de fevereiro de 2024

Filas

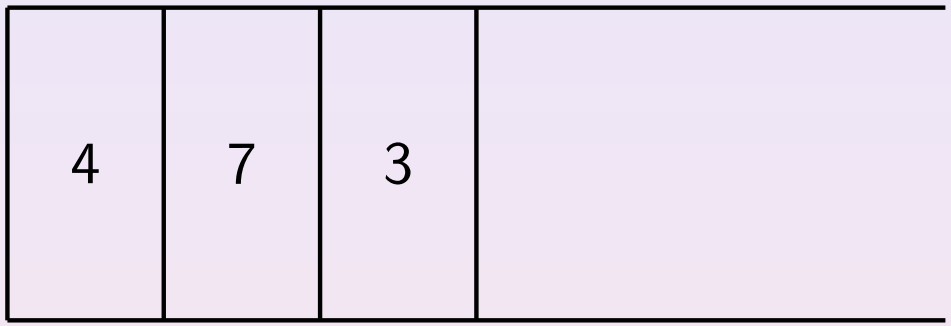
Fila:

- Um conjunto ordenado de itens
- Novos itens podem ser inseridos em uma extremidade: *Final* e excluídos da outra extremidade: *Início*
- Fila é uma estrutura de dados dinâmica: a operação de inserir e remover itens faz parte da estrutura.
- A representação de uma fila é um sequência de objetos, um objeto entra na fila, sendo inserido no final, o objeto que sai da fila é retirado de seu início.

Exemplo de Fila e operações:

- O que determina o estado da fila é o conjunto de objetos lá existentes.
- A operação que remove um item não especifica qual item será removido, pois será o item que está no início da fila.

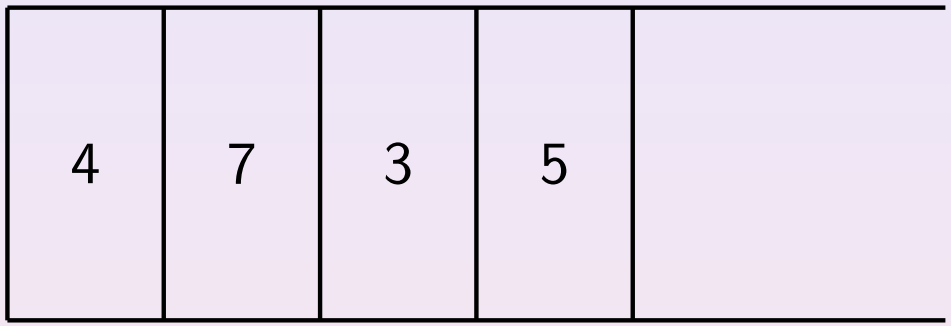
Exemplo de Fila e operações:



Inserindo 3

- O que determina o estado da fila é o conjunto de objetos lá existentes.
- A operação que remove um item não especifica qual item será removido, pois será o item que está no início da fila.

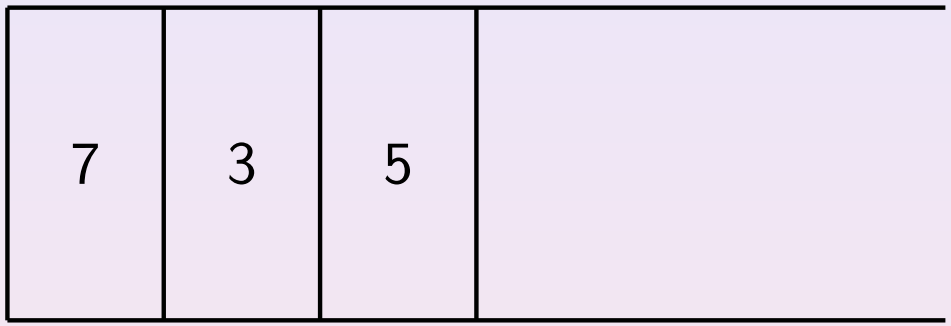
Exemplo de Fila e operações:



Inserindo 5

- O que determina o estado da fila é o conjunto de objetos lá existentes.
- A operação que remove um item não especifica qual item será removido, pois será o item que está no início da fila.

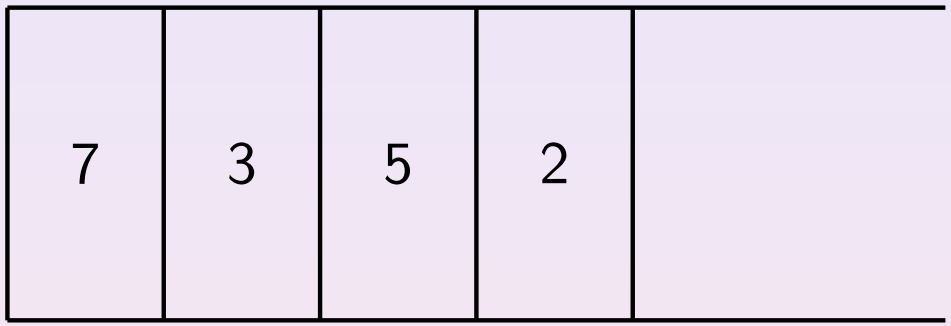
Exemplo de Fila e operações:



Removendo

- O que determina o estado da fila é o conjunto de objetos lá existentes.
- A operação que remove um item não especifica qual item será removido, pois será o item que está no início da fila.

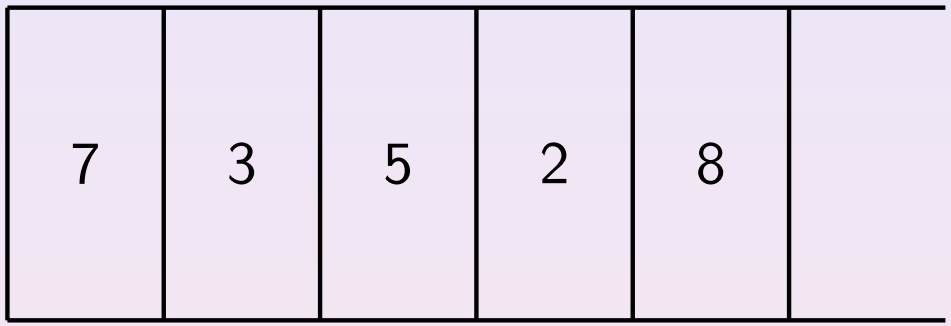
Exemplo de Fila e operações:



Inserindo 2

- O que determina o estado da fila é o conjunto de objetos lá existentes.
- A operação que remove um item não especifica qual item será removido, pois será o item que está no início da fila.

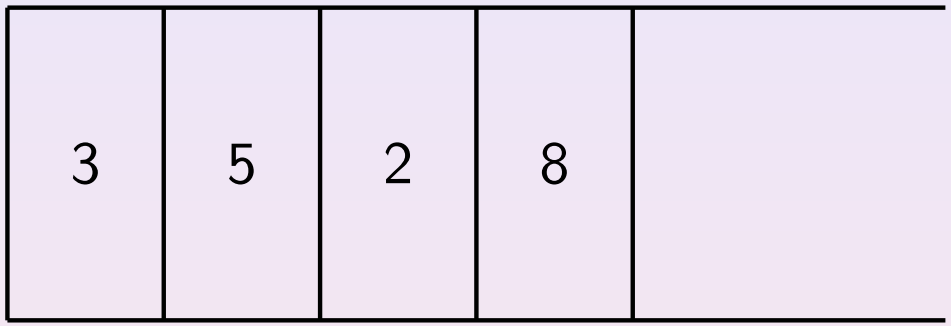
Exemplo de Fila e operações:



Inserindo 8

- O que determina o estado da fila é o conjunto de objetos lá existentes.
- A operação que remove um item não especifica qual item será removido, pois será o item que está no início da fila.

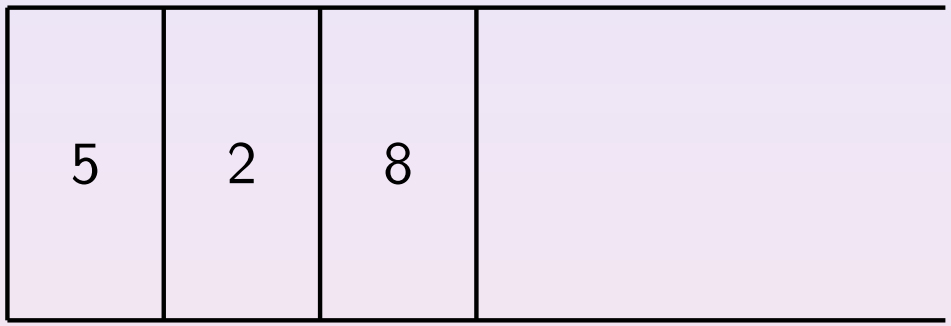
Exemplo de Fila e operações:



Removendo

- O que determina o estado da fila é o conjunto de objetos lá existentes.
- A operação que remove um item não especifica qual item será removido, pois será o item que está no início da fila.

Exemplo de Fila e operações:



Removendo

- O que determina o estado da fila é o conjunto de objetos lá existentes.
- A operação que remove um item não especifica qual item será removido, pois será o item que está no início da fila.

Enfileirar / Desenfileirar

- Duas operações são realizadas sobre a fila:
  - A operação de inserir na fila é: *Enfileirar(Obj)*.
  - A operação de remover da pilha é: *Desenfileirar()* $\rightarrow$ *Obj*
- Estas operações são conhecidas, também, como *Enqueue(Obj)* e *Dequeue()*, respectivamente.
- Nos slides anteriores, a sequência de operações foram:
  Enfileirar(3)
  Enfileirar(5)
  Desenfileirar()
  Enfileirar(2)
  Enfileirar(8)
  Desenfileirar()
  Desenfileirar()

FIFO ou LILO

- Dada as operações sobre a fila e como são inseridos os objetos, tem-se:
- O primeiro objeto a ser inserido será o primeiro a ser removido.
- *FIFO* - First In, First Out.
- Ou então, o último objeto que foi inserido na fila será o último a ser removido.
- *LILO* - Last In, Last Out. (pouco usado)

Aplicações de Filas

- Filas de escalonamento de processos: Um processo por vez usa o processador.
- Filas de lotes batchs de processamento. Por exemplo: Slurm (Simple Linux Utility for Resource Management) em HPC (High Performance Cluster).
- Spool de impressão para Impressoras compartilhadas
- Fila de instruções na microarquitetura e em algoritmos superescalares.
- Fila de pacotes numa transmissão de rede...

Implementação da Fila

- Acompanhando a implementação das outras estruturas, teremos as seguintes funções:
  - *Enfileirar* - inclui um objeto no final da fila.
  - *Desenfileirar* - remove um objeto do início da fila.
  - *Frente* - mostra qual o primeiro objeto da fila.
  - *ehVazia* - informa se a fila está vazia.
  - *Tamanho* - indica quantos elementos a fila possui.

Implementando Fila: Vetores

- A implementação de fila necessita de dois marcadores: Início e Fim.
- O ponteiro início aponta para o primeiro elemento da fila.
- O ponteiro final aponta para a primeira posição livre para inserir elemento.
- É mais prático andar os marcadores do que andar com os dados no vetor.
- O vetor é usado de forma circular:
- A implementação de fila em vetores também precisa de dois marcadores: vazia e cheia.
- Nas duas situações, a relação entre Início e Fim são idênticas, logo são necessários os marcadores para diferenciar as operações permitidas e proibidas.

Implementando Fila

- As funções para a fila:
  - A função *Enfileirar(fila,obj)*: Insere um objeto no final e atualiza a posição do final da fila. Retorna se houve sucesso.
  - O ponteiro de final de fila é incrementado, chegando no final do vetor, ele retoma o início.
  - A função *Desenfileirar(fila)*: Remove um objeto do início da fila e atualiza a posição do início. Retorna se houve sucesso.
  - O ponteiro de início de fila é incrementado, chegando no final do vetor, ele retoma o início.
  - Se o ponteiro de início atingir a posição do ponteiro de final da fila, a fila está vazia.
  - Se o ponteiro de final de fila atingir a posição do ponteiro de início da fila, a fila está cheia.
  - A função *Frente(fila)*: Devolve o primeiro elmento da fila a ser retirado.

Vetores - Estrutura e tipos para a fila

```
#define TAMANHOFILA 5
typedef int obj_t;
typedef int boolean;
enum{falso, verdade};

typedef struct fila {
  obj_t itens[TAMANHOFILA];
  int inicio, fim;
  boolean cheia, vazia;
} fila;

boolean enfileirar(fila *p, obj_t obj);
boolean desenfileirar(fila *p);
boolean ehvazia(fila p);
obj_t frente(fila p);
int tamanho(fila p);
```

Operações: Enfileirar

```
boolean enfileirar(fila *p, obj_t obj) {
  boolean ret = falso;
  if(!p->cheia) {
    p->itens[p->fim++]=obj;
    if(p->fim == TAMANHOFILA) p->fim=0;
    if(p->inicio == p->fim) p->cheia=verdade;
    p->vazia=falso;
    ret = verdade;
  }
  return ret;
}
```

Operações: Desenfileirar

```
boolean desenfileirar(fila *p) {
 boolean ret=falso;
 if(!p- >vazia) {
  p- >inicio++;
  if(p- >inicio == TAMANHOFILA) p- >inicio = 0;
  if(p- >inicio == p- >fim) p- >vazia = verdade;
  p- >cheia = falso;
  ret = verdade;
 }
 return ret;
}
```

Operações: ehVazia, Frente e Tamanho

```
boolean ehvazia(fila p) {
  return (p.vazia);
}

obj_t frente(fila p) {
  obj_t o = 0;
  if(!p.vazia) o = p.itens[p.inicio];
  return o;
}

int tamanho(fila p) {
  int tam = TAMANHOFILA;
  if(!ehcheia)
    tam = (p.fim-p.inicio >= 0 ? p.fim-p.inicio : TAMANHOFILA + p.fim - p.inicio);
  return tam;
}
```

Testando

```
int main() {
  fila f;
  f.inicio = f.fim = 0;
  f.cheia=falso;
  f.vazia=verdade;
  enfileirar(&f,5); printf("%d -- > %d\n",tamanho(f),frente(f));
  enfileirar(&f,4); printf("%d -- > %d\n",tamanho(f),frente(f));
  enfileirar(&f,3); printf("%d -- > %d\n",tamanho(f),frente(f));
  desenfileirar(&f); printf("%d -- > %d\n",tamanho(f),frente(f));
  enfileirar(&f,2); printf("%d -- > %d\n",tamanho(f),frente(f));
  enfileirar(&f,1); printf("%d -- > %d\n",tamanho(f),frente(f));
  enfileirar(&f,7); printf("%d -- > %d\n",tamanho(f),frente(f));
  enfileirar(&f,8); printf("%d -- > %d\n",tamanho(f),frente(f));
  while(!ehvazia(f)) {
    desenfileirar(&f); printf("%d -- > %d\n",tamanho(f),frente(f));
  }
  return 0;
}
```

Saída

```
1 --> 5
2 --> 5
3 --> 5
2 --> 4
3 --> 4
4 --> 4
5 --> 4
5 --> 4
4 --> 3
3 --> 2
2 --> 1
1 --> 7
0 --> 0
```

Implementando via Lista Ligada

- Como os itens são enfileirados de um lado e desenfileirados de outro, a melhor solução é manter dois ponteiros na fila, o cabeça (início da fila) e o cauda (fim da fila).
- Lista Ligada oferece duas vantagens:
  - Não há limite de enfileiramento (exceto o próprio limite de memória).
  - Não precisa deslocar ponteiros ou dados, insere no final e retira no início.
- A fila está vazia quando a cabeça e cauda são null

Tipos e Protótipos

```
typedef int obj_t;
typedef int bool;
enum{false,true};

typedef struct fila_item {
  obj_t item;
  struct fila_item *prox;
} fila_item;
typedef struct fila {
  fila_item *inicio, *fim;
} fila;

void iniciar(fila *);
bool enfileirar(fila *f, obj_t obj);
bool desenfileirar(fila *f);
obj_t frente(fila *f);
bool ehvazia(fila *f);
int tamanho(fila *inicio);
```

Operações: Enfileirar

```
void enfileirar(fila *f, obj_t obj) {
  bool res = false;
  fila_item *o = malloc(sizeof(fila_item));
  if(o) {
    o->item = obj;
    o->prox = NULL;
    if(f->fim) f->fim->prox = o;
    else f->inicio = o;
    (f->fim) = o;
    res = true;
  }
  return res;
}
```

Operações: Desenfileirar

```
bool desenfileirar(fila *f) {
  bool res = false;
  if(f->inicio) {
    fila_item *o = f->inicio;
    f->inicio = f->inicio->prox;
    if(!f->inicio) f->fim = NULL;
    free(o);
    res = true;
  }
  return res;
}
```

Operações:Iniciar, Frente, ehVazia e Tamanho

```
void iniciar(fila *f) {
 f->inicio = f->fim = NULL;
}
obj_t frente(fila *f) {
 obj_t o;
 if(f->inicio) o = f->inicio->item;
 return o;
}
bool ehvazia(fila *f) {
 return (!f->inicio);
}
int tamanho(fila *f) {
 int cont = 0;
 for(fila_item *p = f->inicio; p; p=p->prox, cont++);
 return cont;
}
```

Testando

```
int main(int argc, char **args) {
  fila *f; iniciar(&f);
  enfileirar(&f, 5); printf("%d -- > %d\n",tamanho(&f),frente(&f));
  enfileirar(&f, 4); printf("%d -- > %d\n",tamanho(&f),frente(&f));
  enfileirar(&f, 3); printf("%d -- > %d\n",tamanho(&f),frente(&f));
  desenfileirar(&f); printf("%d -- > %d\n",tamanho(&f),frente(&f));
  enfileirar(&f, 2); printf("%d -- > %d\n",tamanho(&f),frente(&f));
  enfileirar(&f, 1); printf("%d -- > %d\n",tamanho(&f),frente(&f));
  enfileirar(&f, 7); printf("%d -- > %d\n",tamanho(&f),frente(&f));
  enfileirar(&f, 8); printf("%d -- > %d\n",tamanho(&f),frente(&f));
  while(!ehvazia(&f)) {
    printf("%d -- > %d\n",tamanho(&f),frente(&f));desenfileirar(&f);
  }
  return 0;
}
```

Resultado

```
1 --> 5
2 --> 5
3 --> 5
2 --> 4
3 --> 4
4 --> 4
5 --> 4
6 --> 4
6 --> 4
5 --> 3
4 --> 2
3 --> 1
2 --> 7
1 --> 8
```

Filas com prioridades

- Fila com prioridades representa uma sequência ordenada e classificada.
- O elemento que é retirado da fila sempre é o elemento de maior prioridade.
- Se dois elementos de mesma prioridade estão para ser retirado, primeiro é retirado aquele que entrou primeiro.
- Implementação: duas opções.
  - Classificar a fila na entrada (insert sort)
  - Retirar o elemento de maior prioridade (busca sequencial)

Comparando as implementações

- Classificar na entrada:
  - Inserir o elemento implica em classificar a sequência.
  - Na média é necessário mover 50% da fila para encaixar o elemento. (Pior caso: custo $O(n)$)
  - Retirar o elemento representa retirar o elemento do início da fila. (Custo $O(1)$)
- Retirar o de maior prioridade:
  - Inserir o elemento representa inserir no fim da fila. (Custo $O(1)$)
  - Para retirar o elemento é necessário verificar toda a fila para achar o elemento de maior prioridade (Sempre é custo $O(n)$).

ED e Linguagens

- As estruturas de dados que vimos são tradicionais e usadas em muita ocasiões.
  - Quando formos trabalhar com árvores usaremos fila para realizar os percursos em largura, por exemplo.
  - Para construir uma árvore de expressão usamos pilhas, bem como para construir a árvores do código de Huffmann.
- É importante então ter pronta uma biblioteca que contemplem estas estruturas.
- Em C++ que utiliza o tipo genérico em templates e Python que não é fortemente tipada, as estruturas estão prontas para uso, e implementadas de forma eficiente.
- Outras estruturas e algoritmos também já estão prontos para uso, como um algoritmo de ordenação.