Filas

Estrutura de Dados — QXD0010

Prof. Atílio Gomes Luiz
gomes.atilio@ufc.br

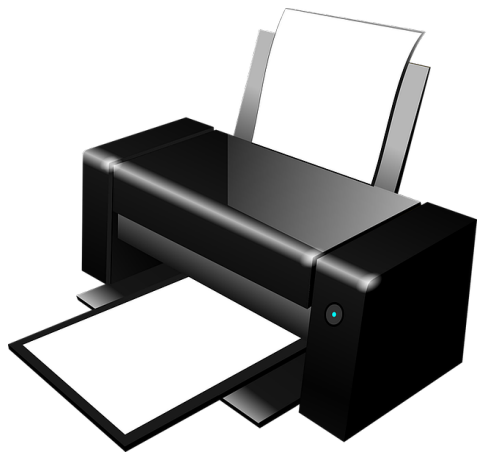
Universidade Federal do Ceará

2º semestre/2025

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

Filas

- Uma impressora é compartilhada em um laboratório
- Alunos enviam documentos quase ao mesmo tempo



Como gerenciar a lista de tarefas de impressão?

Filas

Definição

Uma **fila** é uma sequência dinâmica de objetos, todos do mesmo tipo, sujeita às seguintes regras de comportamento:

(1) sempre que solicitamos a remoção de um elemento, o elemento removido é o primeiro da sequência e

(2) sempre que solicitamos a inserção de um novo objeto, o objeto é inserido no fim da sequência.

Filas

- As **filas** são `listas lineares` que adotam a política FIFO para a manipulação de elementos.
- FIFO (*first-in first-out*): o primeiro que entra é o primeiro que sai. Removemos primeiro os objetos inseridos há mais tempo.

Filas

- As **filas** são `listas lineares` que adotam a política FIFO para a manipulação de elementos.
- FIFO (*first-in first-out*): o primeiro que entra é o primeiro que sai. Removemos primeiro os objetos inseridos há mais tempo.

Operações básicas:

- Enfileira (*push*): adiciona item no **fim**

Filas

- As **filas** são `listas lineares` que adotam a política FIFO para a manipulação de elementos.
- FIFO (*first-in first-out*): o primeiro que entra é o primeiro que sai. Removemos primeiro os objetos inseridos há mais tempo.

Operações básicas:

- Enfileira (*push*): adiciona item no **fim**
- Desenfileira (*pop*): remove item do **início**

Filas

- As **filas** são `listas lineares` que adotam a política FIFO para a manipulação de elementos.
- FIFO (*first-in first-out*): o primeiro que entra é o primeiro que sai. Removemos primeiro os objetos inseridos há mais tempo.

Operações básicas:

- Enfileira (*push*): adiciona item no **fim**
- Desenfileira (*pop*): remove item do **início**
- A consulta na fila é feita desenfileirando elemento a elemento até encontrar o elemento desejado ou chegar ao final da fila.

Operações de Fila

- Construir uma fila vazia
- Testar se a fila está vazia
- Retornar o número de elementos na fila
- Acessar o primeiro da fila
- Acessar o último elemento da fila
- Inserir um elemento
- Remover o próximo elemento

Exemplos de aplicações de filas

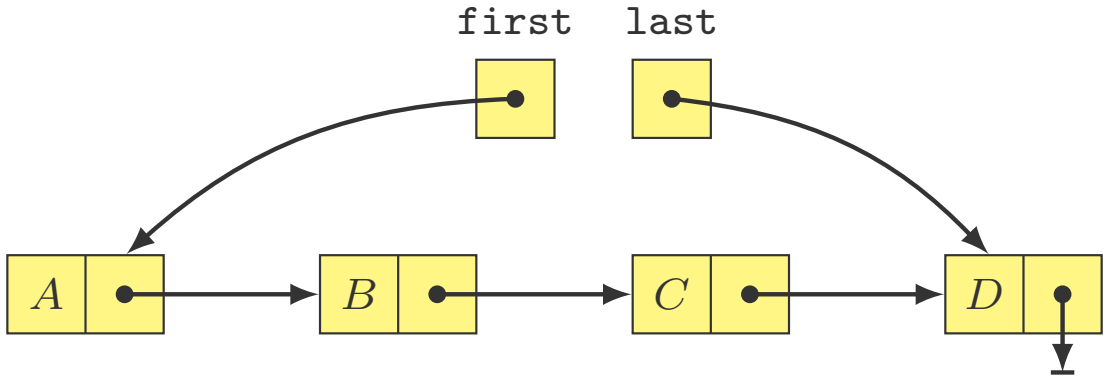
Algumas aplicações de filas:

- Gerenciamento de fila de impressão
- Buffer do teclado
- Escalonamento de processos
- Comunicação entre aplicativos/computadores
- Percurso de estruturas de dados complexas (grafos etc.)

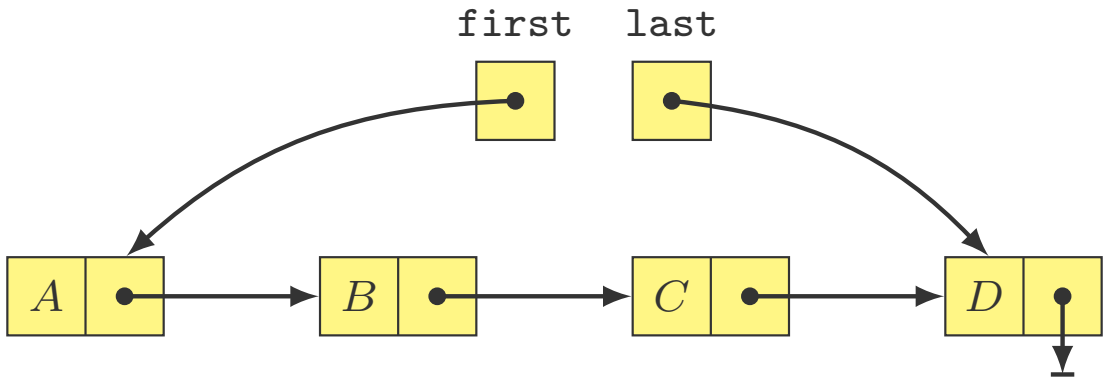
UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

Implementação

Implementação de uma Fila



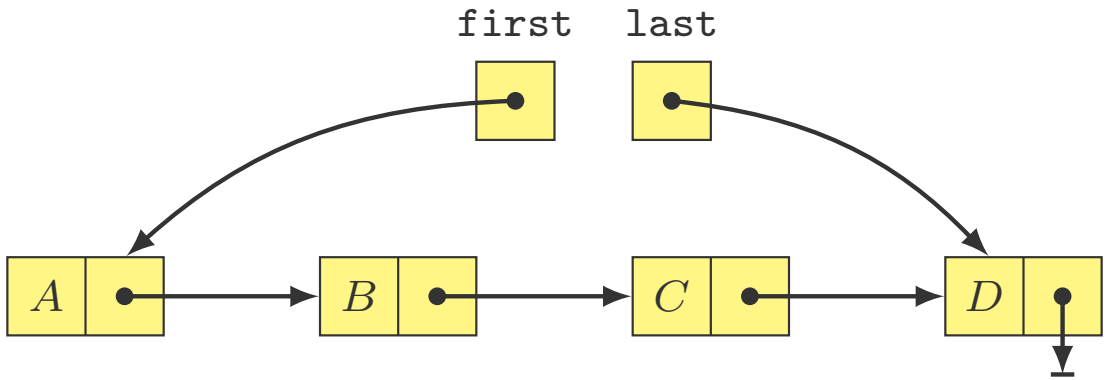
Implementação de uma Fila



- Com relação à alocação de memória, o modo mais natural de implementar uma fila é usando alocação dinâmica.

Implementação de uma Fila

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

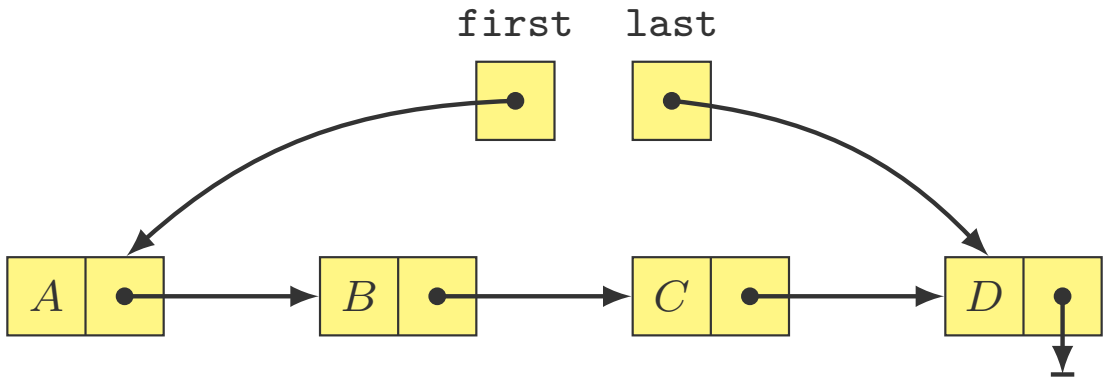


- Com relação à alocação de memória, o modo mais natural de implementar uma fila é usando alocação dinâmica.

- Vamos implementar a fila usando uma lista simplesmente encadeada sem nó cabeça com um ponteiro para o início e outro para o fim da lista.

Implementação de uma Fila

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ



- Com relação à alocação de memória, o modo mais natural de implementar uma fila é usando alocação dinâmica.
- Vamos implementar a fila usando uma lista simplesmente encadeada sem nó cabeça com um ponteiro para o início e outro para o fim da lista.
- Vamos supor que o primeiro elemento da fila ficará no primeiro nó e que o último elemento da fila ficará no último nó.

Implementação de uma Fila

- Nossa fila armazenará inteiros.

Implementação de uma Fila

- Nossa fila armazenará inteiros.

- A nível de implementação, cada nó da lista simplesmente encadeada será representado como uma estrutura (struct) que possui apenas dois campos:

Implementação de uma Fila

- Nossa fila armazenará inteiros.

- A nível de implementação, cada nó da lista simplesmente encadeada será representado como uma estrutura (struct) que possui apenas dois campos:
  - `value`: guarda o valor inteiro.

Implementação de uma Fila

- Nossa fila armazenará inteiros.

- A nível de implementação, cada nó da lista simplesmente encadeada será representado como uma estrutura (struct) que possui apenas dois campos:
  - `value`: guarda o valor inteiro.
  - `next`: ponteiro que aponta para o nó seguinte na lista.

Implementação de uma Fila

- Nossa fila armazenará inteiros.

- A nível de implementação, cada nó da lista simplesmente encadeada será representado como uma estrutura (struct) que possui apenas dois campos:
  - `value`: guarda o valor inteiro.
  - `next`: ponteiro que aponta para o nó seguinte na lista.

- Definição do nó da lista:

```
1 struct node {
2     int value;
3     struct node *next;
4 };
5
6 typedef struct node Node;
```

Implementação de uma Fila

- Também teremos uma outra estrutura (struct queue) que agrupa os ponteiros para o início e o fim da lista, além do tamanho da lista:

```
1 struct queue {
2     Node *first;
3     Node *last;
4     size_t size;
5 };
6
7 typedef struct queue Queue;
```

Queue.h — Tipo Abstrato de Dado Fila

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

```
1  #ifndef QUEUE_H
2  #define QUEUE_H
3  #include <stdio.h>
4  #include <stdbool.h>
5
6  typedef struct node Node;
7  typedef struct queue Queue;
8
9  Queue* queue_create(void);
10 size_t queue_size(Queue *q);
11 bool queue_empty(Queue *q);
12 void queue_push(Queue *q, int value);
13 void queue_pop(Queue *q);
14 int queue_front(Queue *q);
15 int queue_back(Queue *q);
16 void queue_swap(Queue *q1, Queue *q2);
17 void queue_free(Queue *q);
18
19 #endif
```

Queue.c: Criando fila vazia


```
1 Queue* queue_create(void) {
2     Queue *q = (Queue*) malloc(sizeof(Queue));
3     q->first = q->last = NULL;
4     q->size = 0;
5     return q;
6 }
```

- A fila vazia tem tamanho igual a zero e os ponteiros `first` e `last` são nulos.

Queue.c: A fila está vazia?


```
1 bool queue_empty(Queue *q) {
2     if(q == NULL) {
3         printf("fail: null pointer.\n");
4         exit(1);
5     }
6     return q->size == 0;
7 }
```

- A função checa se o ponteiro é válido e, se não for, aborta o programa.
- Se o ponteiro não for nulo, retorna `true` se e somente se a fila estiver vazia.

Queue.c: Tamanho da fila


```
1 size_t queue_size(Queue *q) {
2     if(q == NULL) {
3         printf("fail: null pointer.\n");
4         exit(1);
5     }
6     return q->size;
7 }
```

- A função checa se o ponteiro é válido e, se não for, aborta o programa.
- Se o ponteiro não for nulo, então retorna o tamanho da fila.

Queue.c: Inserir na fila


```
1  void queue_push(Queue *q, int value) {
2      if(q == NULL) {
3          printf("fail: null pointer.\n");
4          exit(1);
5      }
6      Node *novo = (Node*) malloc(sizeof(Node));
7      novo->value = value;
8      novo->next = NULL;
9      if(q->last != NULL) {
10         q->last->next = novo;
11     } else {
12         q->first = novo;
13     }
14     q->last = novo;
15     q->size++;
16 }
```

- Sempre inserimos no final da fila.

Queue.c: Remover da fila


```
1  void queue_pop(Queue *q) {
2      if(queue_empty(q)) {
3          printf("fail: empty queue.\n");
4          exit(1);
5      }
6      Node *temp = q->first;
7      q->first = temp->next;
8      if(q->first == NULL) {
9          q->last = NULL;
10     }
11     free(temp);
12     q->size--;
13 }
```

- Sempre removemos do início da fila.

Queue.c: Acessar valor do primeiro elemento

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

```
1  int queue_front(Queue *q) {
2      if(q == NULL) {
3          printf("fail: null pointer.\n");
4          exit(1);
5      }
6      if(q->first == NULL) {
7          printf("fail: empty queue.\n");
8          exit(1);
9      }
10     else
11         return q->first->value;
12 }
```

Queue.c: Acessar valor do último elemento

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

```
1  int queue_back(Queue *q) {
2      if(q == NULL) {
3          printf("fail: null pointer.\n");
4          exit(1);
5      }
6      if(q->last == NULL) {
7          printf("fail: empty queue.\n");
8          exit(1);
9      }
10     else
11         return q->last->value;
12 }
```

Queue.c: Trocar duas filas

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

```
1 void queue_swap(Queue *q1, Queue *q2) {
2     if(q1 == NULL || q2 == NULL) {
3         printf("fail: null pointer.\n");
4         exit(1);
5     }
6     Queue aux = *q1;
7     *q1 = *q2;
8     *q2 = aux;
9 }
```

Queue.c: Liberar fila

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

```
1  void queue_free(Queue *q) {
2      if(q == NULL) return;
3      Node *aux = q->first;
4      while(aux != NULL) {
5          Node *temp = aux->next;
6          free(aux);
7          aux = temp;
8      }
9      free(q);
10 }
```

Exercícios

Exercício 1 (Intercalação de Filas)

Considere o tipo abstrato de dados Queue como definido nesta aula:

- Implemente uma função que receba duas filas `f1` e `f2`, e transfira alternadamente os elementos de `f1` e `f2` para uma nova fila `f_res` e retorne essa nova fila.

- Note que, ao final dessa função, as filas `f1` e `f2` vão estar vazias, e a fila `f_res` vai conter todos os valores originalmente em `f1` e `f2`.

- Exemplo: suponha que `f1 = [1,2,3,4,5,6]` e `f2 = [20,21,22]`. Então, `fres = [1,20,2,21,3,22,4,5,6]`

- Essa função deve obedecer ao protótipo:
```
void intercala(Queue *f1, Queue *f2, Queue *f_res);
```

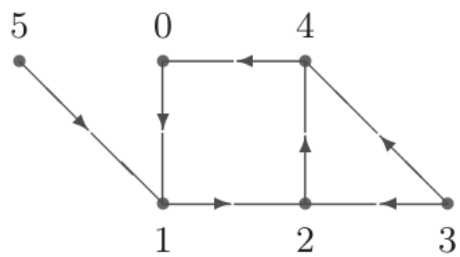
Uma aplicação de fila

Aplicação: distâncias em uma rede

Imagine n cidades numeradas de 0 a $n-1$ e interligadas por estradas de mão única. As ligações entre as cidades são representadas por uma matriz A definida da seguinte maneira:

- $A[i][j] = 1$ se existe estrada da cidade i para a cidade j e $A[i][j] = 0$ em caso contrário.

	0	1	2	3	4	5
0	0	1	0	0	0	0
1	0	0	1	0	0	0
2	0	0	0	0	1	0
3	0	0	1	0	1	0
4	1	0	0	0	0	0
5	0	1	0	0	0	0



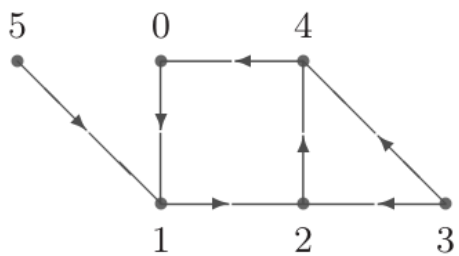
Aplicação: distâncias em uma rede

Universidade Federal do Ceará
Campus Quixadá

Imagine n cidades numeradas de 0 a $n-1$ e interligadas por estradas de mão única. As ligações entre as cidades são representadas por uma matriz A definida da seguinte maneira:

- $A[i][j] = 1$ se existe estrada da cidade i para a cidade j e $A[i][j] = 0$ em caso contrário.

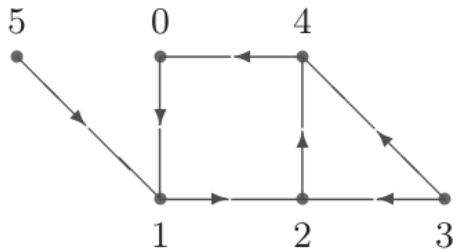
	0	1	2	3	4	5
0	0	1	0	0	0	0
1	0	0	1	0	0	0
2	0	0	0	0	1	0
3	0	0	1	0	1	0
4	1	0	0	0	0	0
5	0	1	0	0	0	0



A **distância** de uma cidade i para uma cidade j é o menor número de estradas que é preciso percorrer para ir de i a j. Nosso problema: determinar a distância de uma dada cidade i a cada uma das outras cidades.

Aplicação: distâncias em uma rede

	0	1	2	3	4	5
0	0	1	0	0	0	0
1	0	0	1	0	0	0
2	0	0	0	0	1	0
3	0	0	1	0	1	0
4	1	0	0	0	0	0
5	0	1	0	0	0	0



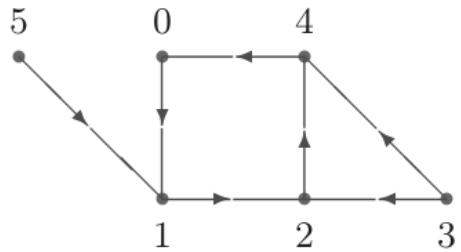
Distância da cidade 3 a cada uma das demais:

`dist = {2,3,1,0,1,-1}`

As distâncias serão armazenadas em um vetor `dist` de tal modo que `dist[x]` seja a distância de i a x. Se for impossível sair de i e chegar em x, dizemos que `dist[x] = -1`.

Aplicação: distâncias em uma rede

	0	1	2	3	4	5
0	0	1	0	0	0	0
1	0	0	1	0	0	0
2	0	0	0	0	1	0
3	0	0	1	0	1	0
4	1	0	0	0	0	0
5	0	1	0	0	0	0



Distância da cidade 3 a cada uma das demais:

`dist = {2,3,1,0,1,-1}`

As distâncias serão armazenadas em um vetor `dist` de tal modo que `dist[x]` seja a distância de i a x. Se for impossível sair de i e chegar em x, dizemos que `dist[x] = -1`.

A seguir mostramos um algoritmo que usa o conceito de fila para resolver nosso problema das distâncias.

Uma cidade é considerada **ativa** se já foi visitada mas as estradas que nela começam ainda não foram exploradas. O algoritmo mantém as cidades ativas numa fila. Em cada iteração, o algoritmo remove da fila uma cidade x e insere na fila todas as vizinhas de x que ainda não foram visitadas.

Programa Principal

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

```
1  #include <stdio.h>
2  #include "Queue.h"
3
4  void distancias(int n, int mat[n][n], int origem, int dist[]);
5  void print(int array[], int n);
6
7  int main() {
8      int matrix[6][6] =
9      {
10         {0,1,0,0,0,0},
11         {0,0,1,0,0,0},
12         {0,0,0,0,1,0},
13         {0,0,1,0,1,0},
14         {1,0,0,0,0,0},
15         {0,1,0,0,0,0}
16     };
17
18     int origem = 3;
19     int vdist[6];
20     distancias(6, matrix, origem, vdist);
21     print(vdist, 6);
22     return 0;
23 }
```

Algoritmo


```
1  void distancias(int n, int mat[n][n], int origem, int dist[])
2  {
3      for(int i = 0; i < n; i++)
4          dist[i] = -1;
5      dist[origem] = 0;
6
7      Queue *fila_cidades = queue_create();
8      queue_push(fila_cidades, origem);
9
10     while(!queue_empty(fila_cidades)) {
11         int cidade = queue_front(fila_cidades);
12         queue_pop(fila_cidades);
13         for(int i = 0; i < n; i++) {
14             if(mat[cidade][i] == 1 && dist[i] == -1) {
15                 dist[i] = dist[cidade] + 1;
16                 queue_push(fila_cidades, i);
17             }
18         }
19     }
20
21     queue_free(fila_cidades);
22 }
```

Função que imprime vetor


```
1 void print(int array[], int n) {
2     printf("[ ");
3     for(int i = 0; i < n; i++) {
4         printf("%d ", array[i]);
5     }
6     printf("]\n");
7 }
```

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

FIM