Filas

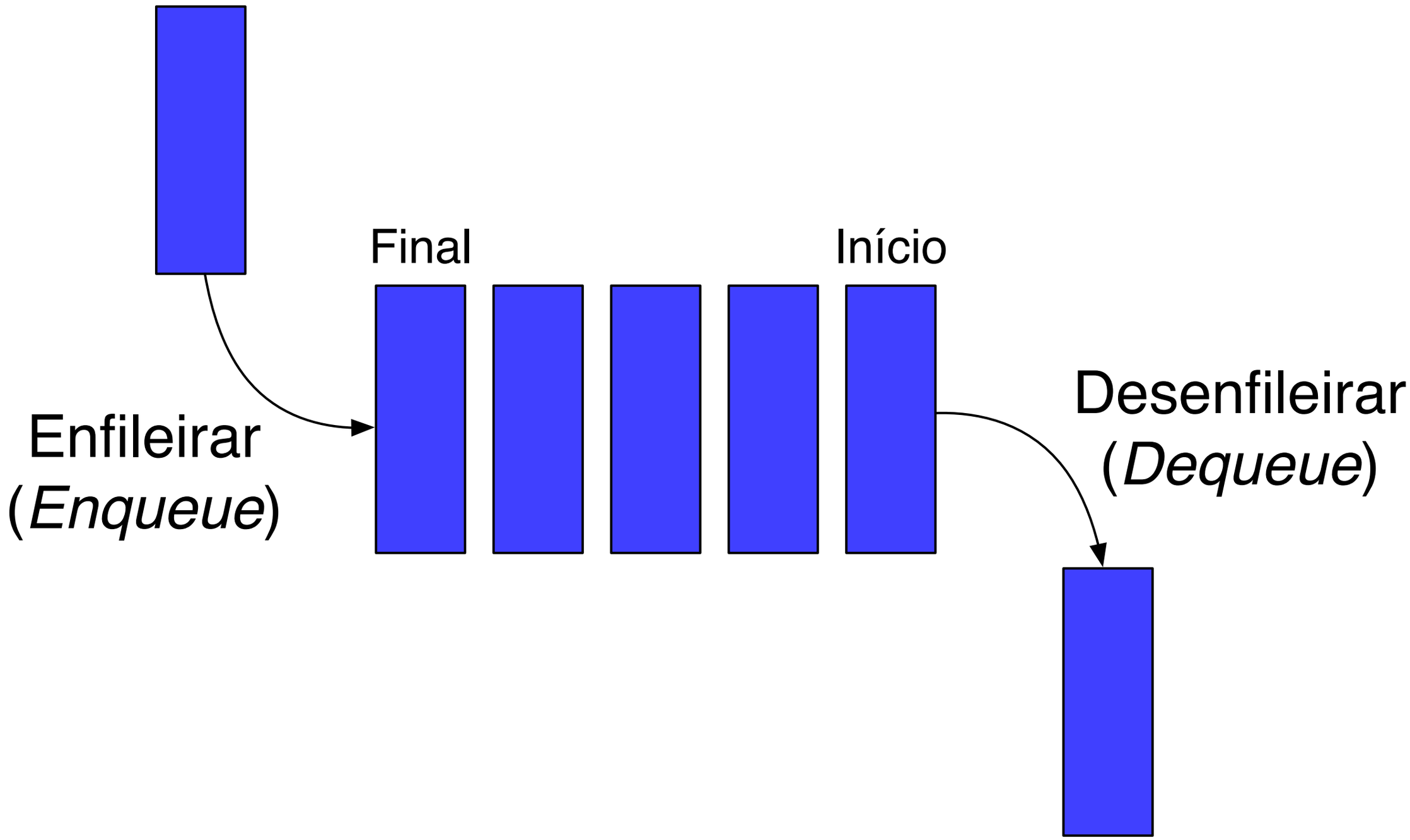
Algoritmos e Programação 2
Prof. Dr. Anderson Bessa da Costa
Universidade Federal de Mato Grosso do Sul

Introdução

- **Filas** são conjuntos dinâmicos que seguem uma política pré-especificada para inserção e remoção de elementos

Fila Indiana

Fila TAD



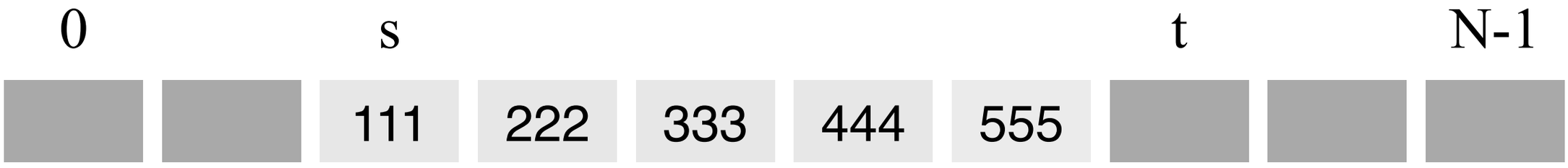
Tipo Abstrato de Dados

- São um **Tipo Abstrato de Dados** (**TAD**, em inglês ADT – *Abstract Data Type*)
  - Descrição lógica
  - vs estrutura de dados que são representações concretas em nível de implementação

Definição

- **Fila** (*Queue*) é um **TAD** que serve como uma coleção de elementos
- Duas operações principais:
  - **enfileirar** (enqueue), que adiciona um elemento para coleção
  - **desenfileirar** (dequeue) que remove o elemento que a mais tempo foi adicionado
- A política de inserção e remoção **FIFO** (*First In, First Out*), em português Primeiro a Entrar, Primeiro a Sair

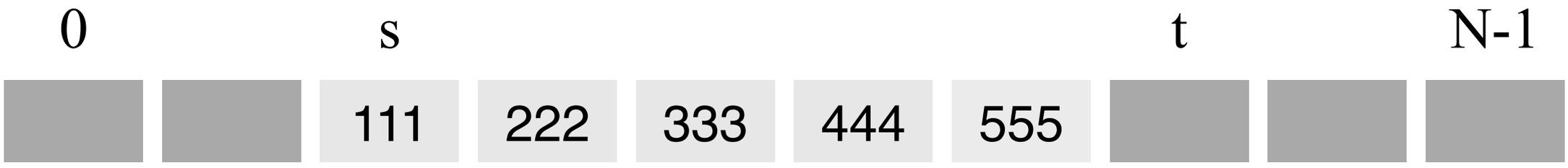
Implementação em Vetor



O vetor $f[s\,..\,t-1]$ armazena uma fila.

- Uma fila pode ser armazenada em um segmento $f[s\,..\,t-1]$ de um vetor $f[0..N-1]$
- É claro que devemos ter $0 \leq s \leq t \leq N$
- Primeiro elemento da fila está na posição s
- Último elemento na posição $t-1$

Implementação em Vetor (cont.)



O vetor $f[s\,..\,t-1]$ armazena uma fila.

- A fila está **vazia** se s é igual a t
- A fila está **cheia** se t é igual a N

Remover

- Para **desenfileirar** (remover) um elemento da fila basta dizer:

```
x = f[s];
s += 1;
```

- É claro que o programador não deve fazer isso se a fila estiver vazia

Inserir

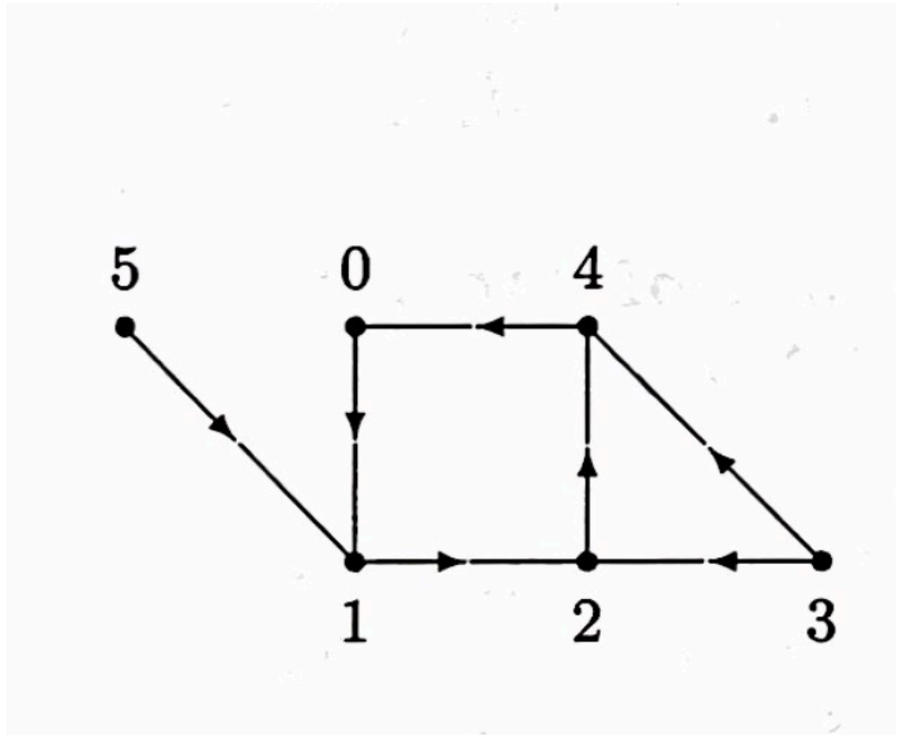
- Para **enfileirar** (inserir) um objeto y na fila basta dizer:

```
f[t] = y;
t += 1;
```

- Se isso for feito quando a fila já está cheia, dizemos que a fila transbordou (*overflow*)

Aplicação: Distâncias em uma rede

- Imagine n cidades numeradas de 0 a $n-1$ e interligadas por estradas de mão única

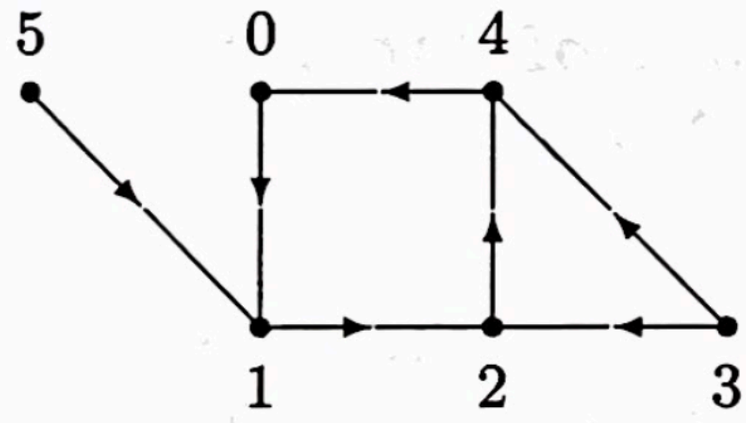


Estradas de mão única representadas como um grafo.

Matriz de Adjacência

- As ligações entre as cidades são representadas por uma matriz a definida da seguinte maneira:
  - $a[x][y]$ vale 1 se existe estrada da cidade x para y
  - Vale 0 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

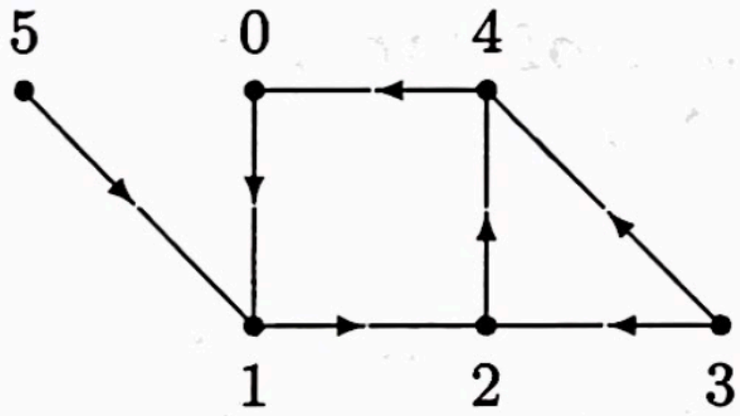


A matriz representa as cidades $0,\ldots,5$ interligadas por estradas de mão única.

Distância

- Distância de uma cidade o a cidade x é o menor número de estradas que é preciso percorrer para ir de o a x
- Problema: *determinar a distância de uma dada cidade o a cada uma das outras cidades*

	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



	0	1	2	3	4	5
d	2	3	1	0	1	6

A matriz representa as cidades $0,\ldots,5$ interligadas por estradas de mão única. O vetor d dá as distâncias da cidade 3 a cada uma das demais.

```c
#include <stdlib.h>

int * distancias(int **a, int n, int o) {
    int *d, x, y;
    int *f, s, t;

    d = (int *) malloc (n * sizeof(int));
    for (x = 0; x < n; x++)
        d[x] = -1;
    d[o] = 0;

    f = (int *) malloc (n * sizeof(int));
    s = 0; t = 1; f[s] = o;

    while (s < t) {
        /* f[s..t-1] e uma fila de cidades */
        x = f[s];
        s += 1;
        for (y = 0; y < n; y++)
            if (a[x][y] == 1 && d[y] == -1) {
                d[y] = d[x] + 1;
                f[t] = y;
                t += 1;
            }
    }
    free(f);
    return d;
}
```

Execução Algoritmo

- $f[s\,..\,t-1]$ armazena a fila das cidades
- $f[0..s-1]$ armazena as cidades que já saíram da fila
  - Para cada v no vetor $f[0..t-1]$, existe um caminho de o a v, de comprimento $d[v]$, cujas cidades estão todas no vetor $f[0..t-1]$;
  - Para cada v no vetor $f[0..t-1]$, todo caminho de o a v tem comprimento pelo menos $d[v]$;
  - Toda estrada que começa em $f[0..s-1]$ termina em $f[0..t-1]$

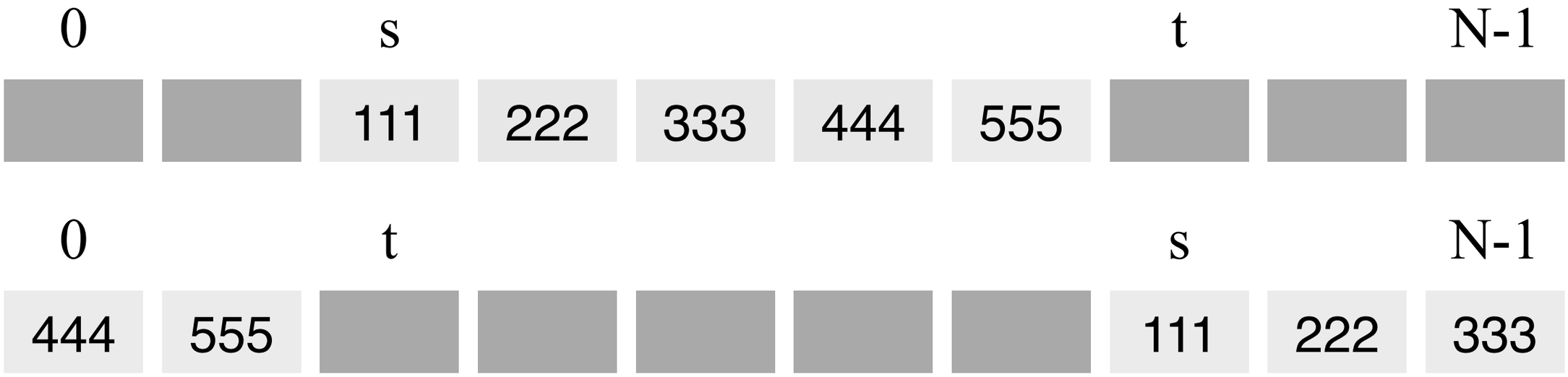
Implementação Circular

- No problema discutido, o vetor que abriga a fila não precisa ter mais componentes que o número total de cidades
  - Cada cidade entra na fila no máximo uma vez
- Em geral, entretanto, é difícil prever o espaço necessário para abrigar a fila
- Nesses casos, é mais seguro implementar a fila de maneira **circular**

Fila Circular

- Suponha que os elementos da fila estão dispostos no vetor $f[0..N-1]$ de uma das seguintes maneiras:
  - $f[s\,..\,t-1]$ ou $f[s\,..\,N-1]$ $f[0..t-1]$

Figura: Fila circular



Fila circular. Na primeira parte da figura, a fila está armazenada no vetor $f[s\,..\,t-1]$. Na segunda parte, a fila está armazenada no vetor $f[s\,..\,N-1]$ concatenado com $f[0..t-1]$.

Fila Circular (cont.)

- Teremos sempre $0 \leq s < N$ e $0 \leq t < N$, mas não podemos supor que $s \leq t$
- Fila **vazia** se $t = s$ e
- Fila **cheia** se $t + 1 = s$ ou $t + 1 = N$ e $s = 0$, ou seja, se $(t + 1) \,\%\, N = s$
- A posição t ficará sempre desocupada, para que possamos distinguir uma fila cheia de uma vazia

Inicializar Fila Circular

- Inicialmente devemos sinalizar que a fila está vazia. Uma ideia consiste em inicializar $s = 0$ e $t = 0$

Algoritmo Enfileire

```c
/* Insere o elemento x no final da fila (política FIFO).
Sem verificação de overflow. */
void enfileire (int f[], int *s, int *t, int x) {
    // insere elemento ao final da fila
    f[*t] = x;

    // incremente indice t
    *t = *t + 1;

    // circular
    if (*t == N)
        *t = 0;
  }
}
```

Algoritmo Desenfileire

```c
/* Remove e retorna o elemento mais antigo da fila (politica FIFO).
Sem verificação de underflow. */
int desenfileire (int f[], int *s, int *t) {
    int x;

    // retire o elemento do inicio
    x = f[*s];

    // incremente de maneira circular
    *s = (*s + 1) % N;

    return x;
}
```

Referências

- FEOFILOFF, P. Algoritmos em Linguagem C, 1. ed. Rio de Janeiro: Elsevier, 2008.
- TENENBAUM, Aaron M; ANGENTEIN, Moshe; LANGSAM, Yedidyah. Estruturas de dados usando C. Sao Paulo, SP: Pearson, 1995. 884p.
- CORMEN, T. H. [et al]. Algoritmos: teoria e prática. 3a ed. Rio de Janeiro: Elsevier, 2012.