

Filas



Uma fila é uma estrutura de dados dinâmica que admite **remoção** de elementos e **inserção** de novos objetos. Mais especificamente, uma **fila** (*= queue*) é uma estrutura sujeita à seguinte regra de operação: sempre que houver uma remoção,

- o elemento removido é o que está na estrutura há *mais* tempo.

Em outras palavras, o primeiro objeto inserido na fila é também o primeiro a ser removido. Essa política é conhecida pela sigla FIFO (*= First-In-First-Out*).

Implementação em um vetor

Suponha que nossa fila mora em um vetor `fila[0..N-1]`. (A natureza dos elementos do vetor é irrelevante: eles podem ser inteiros, bytes, ponteiros, etc.) Digamos que a parte do vetor ocupada pela fila é

`fila[p..u-1]`.

O primeiro elemento da fila está na posição `p` e o último na posição `u-1`. A fila está **vazia** se `p == u` e **cheia** se `u == N`. A figura mostra uma fila que contém os números 111, 222, ..., 666:

0		p						u				N-1
		111	222	333	444	555	666					

Para **limar**, ou **remover** (*= delete = de-queue*), um elemento da fila basta fazer

```
x = fila[p++];
```

Isso **equival**e ao par de instruções "`x = fila[p]; p += 1;`", nesta ordem. É claro que você só deve fazer isso se tiver certeza de que a fila não está vazia. Para **colocar**, ou **inserir** (*= insert = enqueue*), um objeto `y` na fila basta fazer

```
fila[u++] = y;
```

Isso equivale ao par de instruções "`fila[u] = y; u += 1;`", nesta ordem. Note como esse código funciona corretamente mesmo quando a fila está vazia. É claro que você só deve inserir um objeto na fila se ela não estiver cheia; caso contrário, a fila *transborda* (ou seja, ocorre um *overflow*).

Para ajudar o leitor humano, podemos embalar as operações de remoção e inserção em duas pequenas funções. Se os objetos da fila forem números inteiros, podemos escrever

```
int tiradafila (void) {
    return fila[p++];
}

void colocanafila (int y) {
    fila[u++] = y;
}
```

Estamos supondo aqui que as variáveis `fila`, `p`, `u` e `N` são **globais**, isto é, foram definidas fora do código das funções. (Para completar o pacote, precisaríamos de mais três funções: uma que crie uma fila, uma que verifique se a fila está vazia e uma que verifique se a fila está cheia; veja exercício [abaixo](#).)

Exercícios 1

- MÓDULO DE IMPLEMENTAÇÃO DE FILA (VERSÃO 1). Escreva um **módulo** `filadeints.c` que implemente uma fila de números inteiros em um vetor alocado estaticamente. O módulo deve conter as funções `criafile`, `colocanafila`, `tiradafila`, `filavazia` e `filacheia`. O vetor que abriga a fila bem como os índices que indicam o início e o fim da fila devem ser variáveis globais do módulo. Escreva também uma `interface` `filadeints.h` para o módulo. [\[Solução\]](#)
- Escreva uma função que devolva o comprimento (ou seja, o número de elementos) da fila.
- Suponha que, diferentemente da convenção adotada acima, a parte do vetor ocupada pela fila é `fila[p..u]`. Escreva o código das funções `colocanafila`, `tiradafila`, `filavazia` e `filacheia`.
- Suponha que, diferentemente da convenção adotada acima, a parte do vetor ocupada pela fila é `fila[0..u]`. Escreva o código das funções `colocanafila`, `tiradafila`, `filavazia` e `filacheia`.

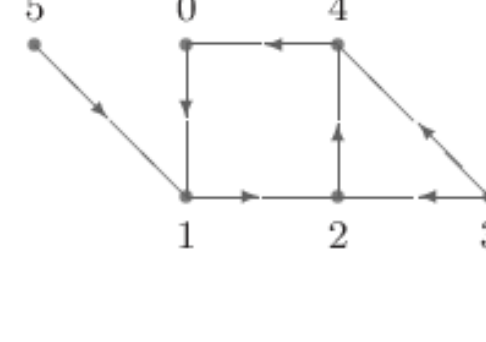
Aplicação: distâncias

A ideia de fila aparece naturalmente no cálculo de **distâncias em um grafo**. (Uma versão simplificada do problema é a *varredura por níveis* de uma *árvore binária*.) 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* da seguinte maneira:

`A[i][j]` vale 1 se existe estrada de *i* para *j*

e vale 0 em caso contrário. Suponha que a matriz tem zeros na diagonal, embora isso seja irrelevante. Segue um exemplo em que *N* vale 6:

	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 *c* a uma cidade *j* é o menor número de estradas que precisamos percorrer para ir de *c* a *j*. (A distância de *c* a *j* é, em geral, diferente da distância de *j* a *c*.) Nosso **problema**: dada uma cidade *c*,

- determinar a distância de *c* a cada uma das demais cidades.

As distâncias podem ser armazenadas em um vetor `dist`: a distância de *c* a *j* é `dist[j]`. É preciso tomar uma decisão de projeto para cuidar do caso em que é impossível ir de *c* a *j*. Poderíamos dizer que `dist[j]` é infinito nesse caso; mas é mais limpo e prático dizer que `dist[j]` vale *N*, pois nenhuma distância "real" pode ser maior que *N*-1. Se tomarmos *c* igual a 3 no exemplo acima, teremos

```
i      0 1 2 3 4 5
dist[i] 2 3 1 0 1 6
```

Eis a ideia de um algoritmo que usa uma *fila de cidades ativas* para resolver nosso problema. Uma cidade *i* é *ativa* se `dist[i]` já foi calculada mas as estradas que começam em *i* ainda não foram todas exploradas. Em cada **iteração**, o algoritmo

- tira da fila uma cidade *i* e coloca na fila todas as cidades vizinhas a *i* cujas distâncias ainda não foram calculadas.

Segue o código que implementa a ideia. (Veja antes um [rascunho em pseudocódigo](#).) Para simplificar, as variáveis `fila`, `p`, `u` e `dist` são **globais**, ou seja, são definidas fora do código das funções.

```
#define N 100
int fila[N], int p, u;
int dist[N];

void criafile (void) {
    p = u = 0;
}

int filavazia (void) {
    return p >= u;
}

int tiradafila (void) {
    return fila[p++];
}

void colocanafila (int y) {
    fila[u++] = y;
}

// Esta função recebe uma matriz A
// que representa as interligações entre
// cidades 0..N-1 e preenche o vetor dist
// de modo que dist[i] seja a distância
// da cidade c à cidade i, para cada i.

void distancias (int A[][N], int c) {
    for (int j = 0; j < N; ++j) dist[j] = N;
    dist[c] = 0;
    criafile ();
    colocanafila (c);

    while (! filavazia ()) {
        int i = tiradafila ();
        for (int j = 0; j < N; ++j)
            if (A[i][j] == 1 && dist[j] >= N) {
                dist[j] = dist[i] + 1;
                colocanafila (j);
            }
    }
}
```

(Poderíamos operar a fila diretamente, sem invocar as funções de manipulação da fila. O resultado *seria mais curto e compacto*, mas um pouco menos legível.)

Exercícios 2

- TAMANHO DA FILA. O espaço alocado para o vetor `fila` é suficiente? A instrução `colocanafila (j)` não provocará o transbordamento da fila?
- ALOCACÃO DINÂMICA. Escreva uma versão da função `distancias` que tenha o número de cidades como um dos parâmetros. (Portanto, *N* não é definido por um `#define`.)
- ☆ O ALGORITMO ESTÁ CORRETO. Prove que a função `distancias` está correta, ou seja, que calcula as distâncias corretas a partir de `c`. [\[Dicas\]](#)
- Faça uma versão da função `distancias` que devolva a distância de uma cidade *a* a outra *b*.
- Imagine um tabuleiro quadriculado com *m*×*n* casas dispostas em *m* linhas e *n* colunas. Algumas casas estão livres e outras estão bloqueadas. As casas livres são marcadas com "." e as bloqueadas com "#". Há um robô na casa (1,1), que é livre. O robô só pode andar de uma casa livre para outra. Em cada passo, só pode andar para a casa que está "ao norte", "a leste", "ao sul" ou "a oeste". Ajude o robô a encontrar a saída, que está na posição (*m*,*n*). (Sugestão: Faça uma moldura de casas bloqueadas.)

Implementação circular

Na implementação que adotamos acima, a fila "anda para a direita" dentro do vetor que a abriga. Isso pode tornar difícil prever o valor que o **parâmetro** *N* deve ter para evitar o transbordamento da fila. Uma implementação "circular" pode ajudar a adiar o transbordamento, como mostraremos a seguir.

Suponha que os elementos da fila estão dispostos no vetor `fila[0..N-1]` de uma das seguintes maneiras: `fila[p..u-1]` ou `fila[p..N-1] fila[0..u-1]`.

0		p						u				N-1
		111	222	333	444	555	666					

0										p		N-1
444	555	666								111	222	333

Teremos sempre $0 \leq p < N$ e $0 \leq u < N$, mas não podemos supor que $p \leq u$. A fila está

- vazia* se `u == p` e
- cheia* se `u+1 == p` ou `u+1 == N e p == 0` (ou seja, se `(u+1) % N == p`).

A posição anterior a `p` ficará sempre desocupada para que possamos distinguir uma fila cheia de uma vazia. Com essas convenções, a remoção de um elemento da fila pode ser escrita assim:

```
int tiradafila (void) {
    int x = fila[p++];
    if (p == N) p = 0;
    return x;
}
```

(desde que a fila não esteja vazia). A inserção de um objeto `y` na fila pode ser escrita assim:

```
void colocanafila (int y) {
    fila[u++] = y;
    if (u == N) u = 0;
}
```

(desde que a fila não esteja cheia).

Exercícios 3

- Imagine uma implementação circular de fila em um vetor `fila[0..9]` que contém

16 17 18 19 20 11 12 13 14 15

Suponha que o primeiro elemento da fila está na posição de índice 5 e o último está na posição de índice 4. Essa fila está cheia?
- Considere a implementação circular de uma fila em um vetor. Escreva o código das funções `colocanafila`, `tiradafila`, `filavazia` e `filacheia`. Escreva uma função que devolva o comprimento (ou seja, o número de elementos) da fila.
- MÓDULO DE IMPLEMENTAÇÃO DE FILA (VERSÃO 2). Escreva um **módulo** `filadeints.c` que faça uma implementação circular de uma fila de números inteiros em um vetor. O módulo deve conter as funções `criafile`, `colocanafila`, `tiradafila`, `filavazia` e `filacheia`. O vetor e as variáveis que indicam o início e o fim da fila devem ser globais no módulo. Escreva também uma `interface` `filadeints.h` para o módulo. (Inspire-se num dos [exercícios acima](#).)

Implementação em vetor com redimensionamento

Nem sempre é possível prever a quantidade de espaço que deve ser reservada para a fila de modo a evitar transbordamentos. Se o vetor que abriga a fila foi alocado **dinamicamente** (com a função `malloc`), é possível resolver essa dificuldade *redimensionando* o vetor: toda vez que a fila ficar cheia, alogue um vetor maior e transfira a fila para esse novo vetor. Para evitar redimensionamentos frequentes, convém que o novo vetor seja pelo menos duas vezes maior que o original.

Eis um exemplo para o caso em que a fila contém números inteiros (e as variáveis `fila`, `p`, `u` e *N* são **globais**):

```
void redimensiona (void) {
    N *= 2;
    fila = realloc (fila, N * sizeof (int));
}
```

Uma versão *ad hoc* poderia ser escrita assim sem usar `realloc`:

```
void redimensiona (void) {
    N *= 2;
    int *novo;
    novo = malloc (N * sizeof (int));
    for (int i = p; i < u; i++)
        novo[i] = fila[i];
    free (fila);
    fila = novo;
}
```

Melhor ainda seria transferir `fila[p..u-1]` para `novo[0..u-p-1]` e reajustar as variáveis `p` e `u` de acordo.

Exercícios 4

- MÓDULO DE IMPLEMENTAÇÃO DE FILA (VERSÃO 3). Escreva um **módulo** `filadeints.c` que implemente uma fila de números inteiros num vetor com redimensionamento. O módulo deve conter as funções `criafile`, `colocanafila`, `tiradafila`, `filavazia`, `liberafile`. (Nessa versão, a função `filacheia` não faz sentido.) Trate os parâmetros da fila como variáveis globais do módulo. Escreva também uma `interface` `filadeints.h` para o módulo. [\[Solução\]](#)
- MÓDULO DE IMPLEMENTAÇÃO DE FILA (VERSÃO 4). Escreva um módulo `filadechars.c` que implemente uma fila de **bytes**. Veja o [exercício anterior](#).

Fila implementada em uma lista encadeada

Como administrar uma fila armazenada em uma **lista encadeada**? Digamos que as células da lista são do tipo `celula`:

```
typedef struct reg {
    int      conteudo;
    struct reg *prox;
} celula;
```

É preciso tomar algumas decisões de projeto sobre como a fila vai morar na lista. Vamos supor que nossa lista encadeada é **circular**: a última célula aponta para a primeira. Vamos supor também que a lista tem uma **célula-cabeça**; essa célula não é removida nem mesmo se a fila ficar vazia. O primeiro elemento da fila fica na *segunda* célula e o último elemento fica na célula *anterior à cabeça*.

Um ponteiro `fi` aponta a célula-cabeça. A fila está *vazia* se `fi->prox == fi`. Uma fila vazia pode ser criada e inicializada assim:

```
celula *fi;
fi = malloc (sizeof (celula));
fi->prox = fi;
```

Podemos agora definir as funções de manipulação da fila. A remoção é fácil:

```
// Tira um elemento da fila fi e devolve
// o conteúdo do elemento removido.
// Supõe que a fila não está vazia.
```

```
int tiradafila (celula *fi) {
    celula *p;
    p = fi->prox; // o primeiro da fila
    int x = p->conteudo;
    fi->prox = p->prox;
    free (p);
    return x;
}
```

A inserção usa um truque sujo: armazena o novo elemento na célula-cabeça original e cria uma nova célula-cabeça:

```
// Coloca um novo elemento com conteúdo y
// na fila fi. Devolve o endereço da
// cabeça da fila resultante.
```

```
celula *colocanafila (int y, celula *fi) {
    celula *nova;
    nova = malloc (sizeof (celula));
    nova->prox = fi->prox;
    fi->prox = nova;
    fi->conteudo = y;
    return nova;
}
```

Exercícios 5

- Implemente uma fila em uma lista encadeada circular *sem* célula-cabeça. (Basta manter o endereço u da última célula; a primeira célula será apontada por `u->prox`. A lista encadeada estará vazia se e somente se `u == NULL`.)
- Implemente uma fila em uma lista encadeada não circular com célula-cabeça. Será preciso manter o endereço `c` da célula-cabeça e o endereço `u` da última célula.
- Implemente uma fila em uma lista encadeada não circular sem célula-cabeça. Será preciso manter um ponteiro `p` para a primeira célula e um ponteiro `u` para a última.
- MÓDULO DE IMPLEMENTAÇÃO DE FILA (VERSÃO 5). Escreva um **módulo** `filadeints.c` que implemente uma fila de números inteiros numa lista encadeada (escolha lista circular ou não circular, com ou sem cabeça). O módulo deve conter as funções `criafile`, `colocanafila`, `tiradafila`, `filavazia`, `liberafile`. Trate os parâmetros da fila como variáveis globais do módulo. Escreva também uma `interface` `filadeints.h` para o módulo. (Inspire-se num dos [exercícios acima](#).)
- LISTA DUPLAMENTE ENCADEADA. Implemente uma fila em uma **lista duplamente encadeada** sem célula-cabeça. Use um ponteiro `p` para a primeira célula e um ponteiro `u` para a última.
- Dique! Uma fila *dupla* (*= deque*, pronuncia-se *deck*) permite inserção e remoção em qualquer das duas extremidades da fila. Implemente uma fila dupla (em um vetor ou uma lista encadeada) e escreva as funções de manipulação da estrutura.

Veja o verbete *Queue* na Wikipédia.

Atualizado em 2018-08-29
<https://www.imc.usp.br/~pf/algoritmos/>
 Paulo Freyfloff
 DCC-IME-USP