

Pilhas e Filas

Filas

Prof. Edson Alves - UnB/FGA

2018

1. Filas
2. Filas com Prioridades
3. Heaps

Filas

Definição de fila

- Uma fila é um tipo de dados abstrato cuja interface define que o primeiro elemento inserido na pilha é o primeiro a ser removido
- Esta estratégia de inserção e remoção é denominada FIFO – *First In, First Out*
- De acordo com sua interface, uma fila não permite acesso aleatório ao seus elementos (apenas o elemento do topo da fila pode ser acessado)
- As operações de inserção e remoção devem ter complexidade $O(1)$

Interface de uma fila

Método	Complexidade	Descrição
clear(F)	$O(N)$	Esvazia a fila F, removendo todos os seus elementos
empty(F)	$O(1)$	Verifica se a fila F está vazia ou não
push(F, x)	$O(1)$	Insere o elemento x no final da fila F
pop(F)	$O(1)$	Remove o elemento que está no início da fila F
front(F)	$O(1)$	Retorna o elemento que está no início da fila F
size(F)	$O(1)$	Retorna o número de elementos armazenados na fila F

Exemplo dos métodos da interface de uma fila

Método

Retorno

empty(F)

Fila

Exemplo dos métodos da interface de uma fila

Método

Retorno

empty(F)

True

Fila

Exemplo dos métodos da interface de uma fila

Método

Retorno

push(F, 5)

Fila

Exemplo dos métodos da interface de uma fila

Método

Retorno

push(F, 5)

Fila



Exemplo dos métodos da interface de uma fila

Método

Retorno

push(F, 11)

Fila



Exemplo dos métodos da interface de uma fila

Método

Retorno

`push(F, 11)`

Fila



Exemplo dos métodos da interface de uma fila

Método

Retorno

`push(F, 7)`

Fila



Exemplo dos métodos da interface de uma fila

Método

Retorno

`push(F, 7)`

Fila



Exemplo dos métodos da interface de uma fila

Método

Retorno

pop(F)

Fila



Exemplo dos métodos da interface de uma fila

Método

Retorno

pop(F)

Fila



Exemplo dos métodos da interface de uma fila

Método

Retorno

`size(F)`

Fila



Exemplo dos métodos da interface de uma fila

Método

Retorno

size(F)

2

Fila



Exemplo dos métodos da interface de uma fila

Método

Retorno

front(F)

Fila



Exemplo dos métodos da interface de uma fila

Método

Retorno

front(F)

11

Fila



Implementação de uma fila

- Como uma pilha é um tipo de dados abstrato, ela não impõe nenhuma restrição quanto à sua implementação
- É possível implementar uma pilha por composição, usando listas encadeadas ou um deque
- A estratégia FIFO precisa de operações de inserção e remoção eficientes nos dois extremos do contêiner, o que inviabiliza o uso do vector e da `forward_list`
- Se há uma estimativa do tamanho máximo de elementos a serem inseridos na fila, é possível usar um *array* estático e o mesmo princípio de uma lista circular para implementar uma fila

Exemplo de implementação de fila em C++

```
1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5 template<typename T, size_t N>
6 class Queue {
7 public:
8     Queue() : first(0), last(0), _size(0) {}
9
10    void clear() { first = last = _size = 0; }
11    bool empty() const { return _size == 0; }
12    size_t size() const { return _size; }
13
14    void push(const T& x)
15    {
16        if (_size == N) throw "Fila cheia";
17
18        elems[last] = x;
19        last = (last + 1) % N;
20        _size++;
21    }
```

Exemplo de implementação de fila em C++

```
22
23     void pop()
24     {
25         if (_size == 0) throw "Lista vazia";
26
27         first = (first + 1) % N;
28         _size--;
29     }
30
31     const T& front() const
32     {
33         if (_size == 0) throw "Lista vazia";
34
35         return elems[first];
36     }
37
38 private:
39     array<T, N> elems;
40     int first, last;
41     size_t _size;
42 };
```

Exemplo de implementação de fila em C++

```
44 int main()
45 {
46     Queue<int, 10> q;
47
48     cout << "Empty? " << q.empty() << '\n';
49
50     for (int i = 1; i <= 10; ++i) q.push(i);
51
52     cout << "Size = " << q.size() << '\n';
53     cout << "Front = " << q.front() << '\n';
54
55     for (int i = 0; i < 5; ++i) q.pop();
56
57     q.push(11);
58     q.push(12);
59
60     cout << "Size = " << q.size() << '\n';
61     cout << "Front = " << q.front() << '\n';
62
63     return 0;
64 }
```

- A STL do C++ oferece uma implementação de fila: a classe `queue`
- Assim como no caso das pilhas, o contêiner usado na composição é, por padrão, o deque
- Este contêiner pode ser substituído por qualquer contêiner que contenha os métodos `pop_front()`, `push_back()` e `size()`, dentre outros
- O método `swap()` também está disponível

Filas com Prioridades

Definição de filas com prioridades

- As filas com prioridades são variações da fila onde os elementos são acessados ou inseridos de acordo com a prioridade estabelecida
- Assim, a estratégia FIFO não se mantém: o primeiro elemento a sair não é mais o primeiro a entrar, e sim o elemento com maior prioridade ainda na fila
- O desafio é encontrar uma implementação eficiente
- Se os elementos são inseridos ordenadamente na fila, de acordo com a prioridade, a complexidade do método `push()` é $O(N)$, e do método `pop()` é $O(1)$
- Se os elementos são inseridos no final da fila, e a prioridade é avaliada no momento do acesso, a complexidade do método `push()` é $O(1)$, e do método `pop()` é $O(N)$

Filas com prioridades em C++

- A STL do C++ oferece um contêiner que implementa uma fila com prioridades: a `priority_queue`, que faz parte da biblioteca `queue`
- A implementação usa, por padrão, o contêiner `vector`, através do qual é implementada uma *heap* binária
- Esta estratégia permite que os métodos `push()` e `pop()` sejam implementados com complexidade $(\log N)$
- Diferente da fila, para acessar o próximo elemento, segundo a prioridade estabelecida, é utilizado o método `top()`
- Por padrão o maior elemento, segundo o critério de comparação, é o de maior prioridade
- Este comportamento pode ser modificado através da reescrita do critério de comparação

Exemplo de filas com prioridades

```
1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5 struct Paciente {
6     string nome;
7     int idade;
8
9     Paciente(const string& n, int a) : nome(n), idade(a) {}
10
11     bool operator<(const Paciente& p) const {
12         // Idosos tem maior prioridade, mais velhos primeiro
13         if (idade >= 65) return idade < p.idade;
14
15         // Depois crianças, mais novas primeiro
16         if (idade <= 6)
17             return p.idade >= 65 or (p.idade <= 6 and idade > p.idade);
18
19         // Os demais por idade
20         return p.idade >= 65 or p.idade <= 6 or p.idade > idade;
21     }
```

Exemplo de filas com prioridades

```
22 };
23
24 int main()
25 {
26     priority_queue<Paciente> pq;
27
28     pq.push(Paciente("Maria", 25));
29     pq.push(Paciente("Carlos", 70));
30     pq.push(Paciente("Neto", 4));
31     pq.push(Paciente("Laura", 25));
32     pq.push(Paciente("Beatriz", 32));
33     pq.push(Paciente("Pedro", 5));
34     pq.push(Paciente("Beto", 68));
35
36     while (not pq.empty()) {
37         auto p = pq.top(); pq.pop();
38         cout << p.nome << ": " << p.idade << " anos\n";
39     }
40
41     return 0;
42 }
```

Heaps

1. **DROZDEK**, Adam. *Algoritmos e Estruturas de Dados em C++*, 2002.
2. **KERNIGHAN**, Bryan; **RITCHIE**, Dennis. *The C Programming Language*, 1978.
3. **STROUSTROUP**, Bjarne. *The C++ Programming Language*, 2013.
4. C++ Reference¹.

¹<https://en.cppreference.com/w/>