

# Heaps binárias

STL

---

Prof. Edson Alves - UnB/FGA

2020

1. Filas com Prioridades
2. Ordenação e heaps na STL

# 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 presente 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`
- Esta fila com prioridades é implementada através de uma *heap* binária
- Esta estratégia permite que os métodos `push()` e `pop()` sejam implementados com complexidade  $O(\log N)$
- 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 operador 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 }
```

# Min heap na STL

- A `priority_queue` da STL é uma *max heap*
- Ela pode ser transformada em uma *min heap* de duas maneiras
- A primeira dela é útil quando a fila armazena tipos numéricos
- Neste caso, basta inserir o simétrico de cada elemento na fila, o que inverterá o critério de comparação
- Deve-se, contudo, tomar o cuidado de utilizar o simétrico do elemento máximo, quando este for extraído
- A segunda maneira é utilizar a estrutura `greater()` da STL, a qual faz com que o operador `>` seja utilizado nas comparações de ordenação
- Se a classe ou estrutura não tiver tal operador por padrão, o mesmo deve ser implementado



# Exemplo de min heap na STL

```
1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5 int main()
6 {
7     vector<int> xs { 40, 68, 15, 99, 24, 6, 51, 77 };
8
9     priority_queue<int> p;
10
11     for (const auto& x : xs)
12         p.push(-x);
13
14     while (not p.empty())
15     {
16         cout << -p.top() << ' ';
17         p.pop();
18     }
19
20     cout << '\n';           // 6 15 24 40 51 68 77 99
21
```

## Exemplo de min heap na STL

```
22     priority_queue<int, vector<int>, greater<int>> q(xs.begin(), xs.end());
23
24     while (not q.empty())
25     {
26         cout << q.top() << ' ';
27         q.pop();
28     }
29
30     cout << '\n';           // 6 15 24 40 51 68 77 99
31
32
33     return 0;
34 }
```

## **Ordenação e heaps na STL**

---

# Heaps e ordenação

- A biblioteca `algorithm` da linguagem C++ contém três rotinas de ordenação, a saber: `sort()`, `stable_sort()` e `partial_sort()`
- As *heaps* estão presentes em dois deles
- A função `sort()` é implementada através de uma estratégia mista: ela começa com o *Introsort* (que combina o *Quicksort* com o *Heapsort*) e finaliza com o *Insert sort*
- Já o `partial_sort()` é implementada por meio do *Heapsort*: é mantida uma *max heap* com exatamente  $k$  elementos, removendo o  $(k + 1)$ -ésimo elemento sempre que o tamanho da *heap* for maior do que  $k$
- Em seguida, os elementos são ordenados, retirando os elementos máximo, um por vez, e colocando-os nas posições apropriadas

## Exemplo de uso das funções de ordenação em C++

```
1 #include <bits/stdc++.h>
2
3 int main()
4 {
5     std::vector<int> xs { 99, 40, 68, 15, 24, 77, 6, 51 };
6     int k = 3;
7
8     std::partial_sort(xs.begin(), xs.begin() + k, xs.end());
9
10    // xs = {6, 15, 24, 99, 68, 77, 40, 51}
11    for (size_t i = 0; i < xs.size(); ++i)
12        std::cout << xs[i] << (i + 1 == xs.size() ? "\n" : " ");
13
14    std::sort(xs.begin(), xs.end(), std::greater<int>());
15
16    // xs = {99, 77, 68, 51, 40, 24, 15, 6}
17    for (size_t i = 0; i < xs.size(); ++i)
18        std::cout << xs[i] << (i + 1 == xs.size() ? "\n" : " ");
19
20    return 0;
21 }
```

1. CppReference. [Priority Queue](#), acesso em 22/04/2019.
2. Geeks for Geeks. [C qsort\(\) vs C++ sort\(\)](#), acesso em 22/04/2019.
3. Quora. [What algorithm do popular C++ compilers use for std::sort\(\) and std::stable\\_sort\(\)](#), acesso em 22/04/2019.
4. Wikipedia. [Introsort](#), acesso em 22/04/2019.