

Strings

String e buscas em C++

Prof. Edson Alves - UnB/FGA

1. Algoritmos elementares
2. Busca em strings na STL

Algoritmos elementares

Métodos `find()` e `rfind()` da classe `string`

- A classe `string` da linguagem C++ oferece dois métodos de busca em strings

Métodos find() e rfind() da classe string

- A classe string da linguagem C++ oferece dois métodos de busca em strings
- O método find() procura pela substring str na substring $S[\text{pos}..(n - 1)]$

```
size_type find(const basic_string& str, size_type pos = 0) const;
```

Métodos find() e rfind() da classe string

- A classe string da linguagem C++ oferece dois métodos de busca em strings
- O método find() procura pela substring str na substring $S[\text{pos}..(n - 1)]$

```
size_type find(const basic_string& str, size_type pos = 0) const;
```

- O retorno é o índice da primeira ocorrência de str na substring em questão, ou `string::npos`, caso str não ocorra em nenhuma posição do intervalo especificado

Métodos find() e rfind() da classe string

- A classe string da linguagem C++ oferece dois métodos de busca em strings
- O método find() procura pela substring str na substring $S[\text{pos}..(n - 1)]$

```
size_type find(const basic_string& str, size_type pos = 0) const;
```

- O retorno é o índice da primeira ocorrência de str na substring em questão, ou `string::npos`, caso str não ocorra em nenhuma posição do intervalo especificado
- A complexidade assintótica é $O(nm)$, onde $n = |S|$ e $m = |\text{str}|$

Métodos find() e rfind() da classe string

- A classe string da linguagem C++ oferece dois métodos de busca em strings
- O método find() procura pela substring str na substring $S[\text{pos}..(n - 1)]$

```
size_type find(const basic_string& str, size_type pos = 0) const;
```

- O retorno é o índice da primeira ocorrência de str na substring em questão, ou `string::npos`, caso str não ocorra em nenhuma posição do intervalo especificado
- A complexidade assintótica é $O(nm)$, onde $n = |S|$ e $m = |\text{str}|$
- O método rfind() tem o mesmo comportamento e retorno, porém busca a última ocorrência de str em $S[0..\text{pos}]$

```
size_type rfind(const basic_string& str, size_type pos = npos) const;
```


Exemplo de uso dos métodos find() e rfind()

```
1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5 int main()
6 {
7     string S = "bananada", P = "ana";
8
9     auto x = S.find(P);           // x = 1
10
11     x = S.find(P, 2);             // x = 3
12     x = S.find(P, 4);            // x = npos
13
14     x = S.rfind(P);              // x = 3
15
16     return 0;
17 }
18
```

Método find_first_of()

- Outro método relacionado à busca de strings é o find_first_of(), cuja assinatura é

```
size_type find_first_of(const basic_string& str, size_type pos = 0) const;
```

Método find_first_of()

- Outro método relacionado à busca de strings é o `find_first_of()`, cuja assinatura é
`size_type find_first_of(const basic_string& str, size_type pos = 0) const;`
- Ele retorna a primeira posição i em S tal que $S[i]$ é igual a um dos caracteres de `str`, ou `string::npos`, caso não encontre nenhum correspondente de `str` em S

Método find_first_of()

- Outro método relacionado à busca de strings é o find_first_of(), cuja assinatura é

```
size_type find_first_of(const basic_string& str, size_type pos = 0) const;
```
- Ele retorna a primeira posição i em S tal que $S[i]$ é igual a um dos caracteres de str , ou $string::npos$, caso não encontre nenhum correspondente de str em S
- A complexidade é a mesma do método find(): $O(nm)$

Método find_first_of()

- Outro método relacionado à busca de strings é o `find_first_of()`, cuja assinatura é
`size_type find_first_of(const basic_string& str, size_type pos = 0) const;`
- Ele retorna a primeira posição i em S tal que $S[i]$ é igual a um dos caracteres de `str`, ou `string::npos`, caso não encontre nenhum correspondente de `str` em S
- A complexidade é a mesma do método `find()`: $O(nm)$
- O método `find_first_not_of()` é semelhante, porém retorna o primeiro caractere de S que é diferente de todos os caracteres de `str`

Método find_first_of()

- Outro método relacionado à busca de strings é o `find_first_of()`, cuja assinatura é

```
size_type find_first_of(const basic_string& str, size_type pos = 0) const;
```
- Ele retorna a primeira posição i em S tal que $S[i]$ é igual a um dos caracteres de `str`, ou `string::npos`, caso não encontre nenhum correspondente de `str` em S
- A complexidade é a mesma do método `find()`: $O(nm)$
- O método `find_first_not_of()` é semelhante, porém retorna o primeiro caractere de S que é diferente de todos os caracteres de `str`
- Os métodos `find_last_of()` e `find_last_not_of()` são equivalentes a ambos, porém iniciando sua busca no sentido oposto

Exemplo de uso dos métodos find_first_of() e find_last_of()

```
1 #include <iostream>
2
3 using namespace std;
4
5 int main()
6 {
7     string S { "exemplo" }, P { "abcde" }, Q { "xyz" };
8
9     auto x = S.find_first_of(P);           // x = 0
10    x = S.find_first_not_of(P);            // x = 1
11    x = S.find_last_of(P);                 // x = 2
12    x = S.find_last_not_of(P);             // x = 6
13
14    auto y = S.find_first_of(Q);           // y = 1
15    y = S.find_first_not_of(Q);            // y = 0
16    y = S.find_last_of(Q);                 // y = 1
17    y = S.find_last_not_of(Q);             // y = 6
18
19    return 0;
20 }
```

Busca em strings na STL

Função search() da STL

- A função search() da STL busca a primeira ocorrência da sequência de elementos [a, b) no intervalo [first, last):

```
template<class ForwardIt1, class ForwardIt2>
```

```
ForwardIt1 search(ForwardIt1 first, ForwardIt1 last, ForwardIt2 a, ForwardIt2 b);
```

Função search() da STL

- A função search() da STL busca a primeira ocorrência da sequência de elementos [a, b) no intervalo [first, last):

```
template<class ForwardIt1, class ForwardIt2>
```

```
ForwardIt1 search(ForwardIt1 first, ForwardIt1 last, ForwardIt2 a, ForwardIt2 b);
```

- Sendo uma função paramétrica, ela pode ser aplicada no contexto de busca em strings

Função search() da STL

- A função search() da STL busca a primeira ocorrência da sequência de elementos [a, b) no intervalo [first, last):

```
template<class ForwardIt1, class ForwardIt2>
```

```
ForwardIt1 search(ForwardIt1 first, ForwardIt1 last, ForwardIt2 a, ForwardIt2 b);
```

- Sendo uma função paramétrica, ela pode ser aplicada no contexto de busca em strings
- Por exemplo, para procurar a primeira ocorrência do padrão P em S a chamada seria

```
1 auto it = search(S.begin(), S.end(), P.begin(), P.end());
```

Função search() da STL

- A função search() da STL busca a primeira ocorrência da sequência de elementos [a, b) no intervalo [first, last):

```
template<class ForwardIt1, class ForwardIt2>
```

```
ForwardIt1 search(ForwardIt1 first, ForwardIt1 last, ForwardIt2 a, ForwardIt2 b);
```

- Sendo uma função paramétrica, ela pode ser aplicada no contexto de busca em strings
- Por exemplo, para procurar a primeira ocorrência do padrão P em S a chamada seria

```
1 auto it = search(S.begin(), S.end(), P.begin(), P.end());
```

- A string S tem tamanho n e o padrão P tem tamanho m , a complexidade será $O(nm)$

Função search() da STL

- A versão C++17 da STL trouxe uma assinatura adicional para a função search():

```
template<class ForwardIt, class Searcher>
```

```
ForwardIt search(ForwardIt first, ForwardIt last, const Searcher& searcher);
```

Função search() da STL

- A versão C++17 da STL trouxe uma assinatura adicional para a função search():

```
template<class ForwardIt, class Searcher>
```

```
ForwardIt search(ForwardIt first, ForwardIt last, const Searcher& searcher);
```

- Deste modo, é possível especificar o algoritmo de busca a ser utilizado para o localizar o padrão indicado no construtor de search na string delimitada pelo intervalo [begin, last)

Função search() da STL

- A versão C++17 da STL trouxe uma assinatura adicional para a função search():

```
template<class ForwardIt, class Searcher>
```

```
ForwardIt search(ForwardIt first, ForwardIt last, const Searcher& searcher);
```

- Deste modo, é possível especificar o algoritmo de busca a ser utilizado para o localizar o padrão indicado no construtor de search na string delimitada pelo intervalo [begin, last)
- A biblioteca padrão fornece três algoritmos:
 1. default_searcher
 2. boyer_moore_searcher
 3. boyer_moore_horspool_searcher

Função search() da STL

- A versão C++17 da STL trouxe uma assinatura adicional para a função search():

```
template<class ForwardIt, class Searcher>
```

```
ForwardIt search(ForwardIt first, ForwardIt last, const Searcher& searcher);
```

- Deste modo, é possível especificar o algoritmo de busca a ser utilizado para o localizar o padrão indicado no construtor de search na string delimitada pelo intervalo [begin, last)
- A biblioteca padrão fornece três algoritmos:
 1. default_searcher
 2. boyer_moore_searcher
 3. boyer_moore_horspool_searcher
- É possível implementar um Searcher customizado

Função `search()` da STL

- O primeiro é o algoritmo utilizado nas demais versões da função `search()`

Função `search()` da STL

- O primeiro é o algoritmo utilizado nas demais versões da função `search()`
- O segundo implementa o algoritmo de Boyer-Moore, cuja complexidade assintótica é $O(n + m)$ no pior caso

Função `search()` da STL

- O primeiro é o algoritmo utilizado nas demais versões da função `search()`
- O segundo implementa o algoritmo de Boyer-Moore, cuja complexidade assintótica é $O(n + m)$ no pior caso
- O terceiro algoritmo é uma versão simplificado do algoritmo de Boyer-Moore, que exige menos memória

Função `search()` da STL

- O primeiro é o algoritmo utilizado nas demais versões da função `search()`
- O segundo implementa o algoritmo de Boyer-Moore, cuja complexidade assintótica é $O(n + m)$ no pior caso
- O terceiro algoritmo é uma versão simplificado do algoritmo de Boyer-Moore, que exige menos memória
- Esta redução de memória, porém, implica em uma complexidade $O(nm)$ no pior caso

Função `search()` da STL

- O primeiro é o algoritmo utilizado nas demais versões da função `search()`
- O segundo implementa o algoritmo de Boyer-Moore, cuja complexidade assintótica é $O(n + m)$ no pior caso
- O terceiro algoritmo é uma versão simplificado do algoritmo de Boyer-Moore, que exige menos memória
- Esta redução de memória, porém, implica em uma complexidade $O(nm)$ no pior caso
- Embora o algoritmo de Boyer-Moore tenha sido proposto inicialmente como um algoritmo de busca em strings, no caso da STL ele pode ser utilizado em um contêiner que armazena um tipo `T` arbitrário

Teste de performance dos algoritmos de busca da STL

```
1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5 double benchmark(const string& S, const string& P, const function<void(string,string)>& f)
6 {
7     auto start = chrono::high_resolution_clock::now();
8     f(S, P);
9     auto end = chrono::high_resolution_clock::now();
10    chrono::duration<double> d = end - start;
11
12    return d.count();
13 }
14
15 int main()
16 {
17     string S(1'000'000, 'a'), P { string(1'000, 'a') + 'b' };
18
19     auto base = benchmark(S, P, [](auto s, auto p)
20         { for (size_t i = 0; i < s.size() + p.size(); ++i) {} });
```

Teste de performance dos algoritmos de busca da STL

```
22  cout.precision(6);
23  cout << "Empty loop: \t\t" << base << " s\t\t-\n";
24
25  auto runtime = benchmark(S, P, [](auto s, auto p) { (void) s.find(p); });
26
27  cout << "find(): \t\t" << runtime << " s\t\ttx" << (int) round(runtime/base) << "\n";
28
29  runtime = benchmark(S, P, [](auto s, auto p)
30  {
31      search(s.begin(), s.end(), p.begin(), p.end());
32  });
33
34  cout << "search(): \t\t" << runtime << " s\t\ttx" << (int) round(runtime/base) << "\n";
35
36  runtime = benchmark(S, P, [](auto s, auto p)
37  {
38      search(s.begin(), s.end(), boyer_moore_searcher(p.begin(), p.end()));
39  });
40
41  cout << "boyer_moore: \t\t" << runtime << " s\t\ttx" << (int) round(runtime/base) << "\n";
```

Teste de performance dos algoritmos de busca da STL

```
43 runtime = benchmark(S, P, [])(auto s, auto p)
44 {
45     search(s.begin(), s.end(), boyer_moore_horspool_searcher(p.begin(), p.end()));
46 };
47
48 cout << "boyer_moore_horspool: \t" << runtime << " s\t\tx"
49      << (int) round(runtime/base) << "\n";
50
51 // Possível saída:
52 // Algoritmo           Runtime (em s)           runtime/empty
53 // Empty loop:         0.00378854                -
54 // find():             0.0930658                x25
55 // search():           4.07972                  x1077
56 // boyer_moore:        0.0121259                x3
57 // boyer_moore_horspool: 0.00646972             x2
58
59 return 0;
60 }
```


1. Bartek's coding blog. [Speeding up Pattern Searches with Boyer-Moore Algorithm from C++17](#), acesso em 21/08/2019.
2. CppReference. [std::basic_string](#), acesso em 21/08/2019.
3. CppReferenc. [std::search](#), acesso em 22/08/2019.
4. **CROCHEMORE**, Maxime; **RYTTER**, Wojciech. *Jewels of Stringology: Text Algorithms*, WSPC, 2002.
5. **HALIM**, Steve; **HALIM**, Felix. *Competitive Programming 3*, Lulu, 2013.
6. Wikipédia. [Boyer-Moore string-search algorithm](#), acesso em 22/08/2019.