

Strings

String e buscas em C++

Prof. Edson Alves - UnB/FGA

2019

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
- 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 }
```

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);
```

- 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);
```

- 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()`
- 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,
6                  const function<void(string,string)>& f)
7 {
8     auto start = chrono::high_resolution_clock::now();
9     f(S, P);
10    auto end = chrono::high_resolution_clock::now();
11    chrono::duration<double> d = end - start;
12
13    return d.count();
14 }
15
16 int main()
17 {
18     string S(1000000, 'a'), P { string(1000, 'a') + 'b' };
19
20     auto base = benchmark(S, P, [](auto s, auto p)
21                             { for (size_t i = 0; i < s.size() + p.size(); ++i) {} });
```

Teste de performance dos algoritmos de busca da STL

```
23     cout.precision(6);
24     cout << "Empty loop: \t\t" << base << " ms\t\t-\n";
25
26     auto runtime = benchmark(S, P, [](auto s, auto p) { s.find(p); });
27
28     cout << "find(): \t\t" << runtime << " ms\t\ttx"
29           << (int) round(runtime/base) << "\n";
30
31     runtime = benchmark(S, P, [](auto s, auto p)
32     {
33         search(s.begin(), s.end(), p.begin(), p.end());
34     });
35
36     cout << "search(): \t\t" << runtime << " ms\t\ttx"
37           << (int) round(runtime/base) << "\n";
38
39     runtime = benchmark(S, P, [](auto s, auto p)
40     {
41         search(s.begin(), s.end(),
42               boyer_moore_searcher(p.begin(), p.end()));
43     });
```

Teste de performance dos algoritmos de busca da STL

```
44
45     cout << "boyer_moore: \t\t" << runtime << " ms\t\t\tx"
46         << (int) round(runtime/base) << "\n";
47
48     runtime = benchmark(S, P, [](auto s, auto p)
49     {
50         search(s.begin(), s.end(),
51             boyer_moore_horspool_searcher(p.begin(), p.end()));
52     });
53
54     cout << "boyer_moore_horspool: \t" << runtime << " ms\t\t\tx"
55         << (int) round(runtime/base) << "\n";
56
57     // Possível saída:
58     // Algoritmo                Runtime (em ms)                runtime/empty
59     // Empty loop:                0.000537077 ms                -
60     // find():                    0.027087 ms                x50
61     // search():                  0.589431 ms                x1097
62     // boyer_moore:                0.00420558 ms                x8
63     // boyer_moore_horspool:       0.00327 ms                x6
64
```

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.