

# Strings

## Algoritmos Elementares

---

Prof. Edson Alves - UnB/FGA

2018

1. Palíndromos
2. Histograma
3. Strings e programação funcional

# Palíndromos

---

# Definição de palíndromos

- Palíndromos são strings que são idênticas quando lidas tanto do início para o fim quanto do fim para o início
- Por exemplo, “MUSSUM”, “SAIAS” e “HANNAH” são palíndromos
- Mais formalmente, um palíndromo  $P$  pode ser definido como

$$P[1..N] = \text{“”} \mid P[1..1] \mid c + P[2..N - 1] + c$$

ou seja, strings vazias, strings com um único caractere ou strings resultantes da concatenação de um mesmo caractere  $c$  no início e no fim de um palíndromo resulta em palíndromos

# Identificação de palíndromos

- Uma maneira de se verificar se uma string  $s$  é ou não um palíndromo é checar se o primeiro caractere coincide com o último, o segundo com o penúltimo, e assim por diante
- Este algoritmo tem complexidade  $O(|s|)$
- Embora ele identifique corretamente se  $s$  é ou não um palíndromo, é possível torná-lo mais eficiente ao observar que só é necessário fazer tal verificação até a metade de  $s$
- Isto ocorre pois se  $i \geq |s|/2$ , temos  $i = N - 1 - j, j < |s|/2$  e a comparação de  $s[i]$  com  $s[N - 1 - i]$  equivale a comparação de  $s[N - 1 - j]$  com  $s[N - 1 - (N - 1 - j)]$ , isto é, de  $s[N - 1 - j]$  com  $s[j], j < |s|/2$
- Mesmo que a complexidade permaneça em  $O(|s|)$ , esta segunda versão abaixo executa aproximadamente duas vezes mais rápido que a anterior

# Implementação da rotina de identificação de palíndromos

```
1 // Observe que a função abaixo identifica s corretamente
2 // mesmo nos casos onde |s| é ímpar
3 bool is_palindrome(const string& s)
4 {
5     size_t N = s.size();
6
7     for (size_t i = 0; i < N/2; ++i)
8         if (s[i] != s[N - 1 - i])
9             return false;
10
11     return true;
12 }
```

# Histograma

---

# Definição de histograma

- Uma técnica, oriunda da estatística, que permite identificar características importantes de uma string é a construção de um histograma
- Um histograma consiste em um mapeamento entre os caracteres do alfabeto e o número de ocorrências dos mesmos em uma string  $s$  dada
- Por exemplo, para a string “abacaxi”, teríamos um histograma  $h$  com  $h['a'] = 4, h['b'] = h['c'] = h['x'] = h['i'] = 1$  e  $h[y] = 0$ , se  $y \notin \{'a', 'b', 'c', 'x', 'i'\}$
- Há 3 técnicas para a construção de histogramas
- A primeira delas é utilizar a classe map do C++, que permite uma construção bastante intuitiva e fácil de histogramas
- Os reveses são a quantidade de memória necessária (o que, em geral, não chega a ser um problema) e a complexidade dos acessos ( $O(\log N)$  para leitura e escrita, onde  $N$  é o número de caracteres distintos presentes na string  $s$ )



# Construção de histogramas usando o map de C++

```
1 #include <map>
2
3 std::map<char, int> histogram(const string& s)
4 {
5     std::map<char, int> h;
6
7     for (auto c : s)
8         ++h[c];
9
10    return h;
11 }
```

# Construção de histogramas usando arrays estáticos

- Uma segunda forma de gerar o histograma é utilizar um array estático com 256 posições
- Estas posições cobrem todos os possíveis valores de um **char** em C/C++
- Tal abordagem assume que a string *s* contém apenas caracteres listados na tabela ASCII
- Também é preciso inicializar todas as posições deste *array* com o valor zero
- Esta construção permite atualizar/consultar os valores em  $O(1)$ , mas a identificação dos caracteres cujos valores associados são diferentes de zero tem que percorrer todas as 256 posições do histograma
- A abordagem anterior traria tais caracteres diretamente, sendo eles as chaves do mapa

# Construção de histogramas usando arrays estáticos

```
1 #include <cstring>
2
3 void histogram(int h[256], const string& s)
4 {
5     memset(h, 0, sizeof h);
6
7     for (auto c : s)
8         ++h[c];
9 }
```

# Construção de histogramas com mapeamento do alfabeto

- A terceira e última abordagem é uma otimização, em espaço, da segunda
- Aqui o vetor  $h$  deve ter o tamanho  $M$  do alfabeto, e os caracteres do alfabeto devem ser indexados de 0 a  $M - 1$
- Se, por exemplo, o alfabeto for constituído das letras maiúsculas e minúsculas, esta indexação é feita de forma direta, em  $O(1)$
- Caso contrário, é preciso procurar pelo caractere no alfabeto em  $O(M)$  (ou  $O(\log M)$ , se o alfabeto estiver ordenado)
- Neste cenário a perda de performance é compensada pela redução da memória necessária
- Esta é a abordagem mais econômica em termos de memória

# Construção de histogramas com mapeamento do alfabeto

```
1 // Esta implementação assume que o alfabeto é composto
2 // pelas 26 letras maiúsculas do alfabeto
3 void histogram(int h[26], const string& s)
4 {
5     memset(h, 0, sizeof h);
6
7     for (auto c : s)
8         ++h[c - 'A'];
9 }
```

# Strings e programação funcional

---

- Mapas, filtros e reduções são técnicas de programação funcional que permitem alterar os elementos de um vetor, gerar um novo vetor a partir da seleção de elementos específicos de um vetor dado ou gerar um único objeto ou elemento a partir dos elementos de um vetor
- Sendo uma string um vetor de caracteres, estas técnicas podem ser adaptadas para o contexto da manipulação de textos e caracteres
- A vantagem de tal abordagem é a redução do tamanho do código, evitando laços e variáveis temporárias explícitas
- Outro aspecto importante é que o uso de tais conceitos permitem simplificar a notação e descrição de problemas de strings

- Um mapa (ou mapeamento) consiste em uma função  $m_f : S_N \rightarrow S_N$ , onde  $S_N$  é o conjunto de todas as strings de tamanho  $N$  e  $f : A \rightarrow A$  é uma função cujo domínio é o alfabeto  $A$  tal que se  $y = m_f(s)$ , então  $y[i] = f(s[i])$
- Em termos mais simples,  $m_f$  mapeia cada caractere de  $s$  de acordo com a função  $f$
- Por exemplo, se  $A$  é formado pelas letras alfabéticas maiúsculas e minúsculas e  $f$  é a função `toupper()`, o mapeamento  $m_f$  tornaria maiúsculas todas as letras de uma string  $s$  dada
- A implementação do mapeamento pode ser apenas conceitual, usando uma função padrão do C/C++, ou utilizar a função `transform()` da STL da linguagem C++



# Exemplo de implementação de um mapa usando funções

```
1 #include <functional>
2 #include <iostream>
3 #include <cctype>
4
5 using namespace std;
6
7 // O nome smap ('string map') foi utilizado para evitar
8 // confusão com a classe map da STL
9 string smap(const string& s, const function<char(char)>& f)
10 {
11     string y;
12
13     for (const auto& c : s)
14         y.push_back(f(c));
15
16     return y;
17 }
18
```

# Exemplo de implementação de um mapa usando funções

```
19 int main()
20 {
21     string s = "Teste de mapeamento";
22     auto y = smap(s, [](char c) { return (char) toupper(c); });
23
24     cout << y << '\n';
25
26     return 0;
27 }
```

# Implementação da cifra de César usando a função transform()

```
1 #include <iostream>
2 #include <algorithm>
3
4 using namespace std;
5
6 int main()
7 {
8     string text { "cesar cipher xyz" };
9     string cipher;
10
11     transform(text.begin(), text.end(), back_inserter(cipher),
12               [](char c) {
13                 return c == ' ' ? c : (((c - 'a') + 3) % 26) + 'a';
14             });
15
16     cout << cipher << endl;
17
18     return 0;
19 }
20 }
```

1. CppReference. [Accumulate](#), acesso em 24/03/2017.
2. CppReference. [Copy](#), acesso em 24/03/2017.
3. CppReference. [Transform](#), acesso em 24/03/2017.
4. **CROCHEMORE**, Maxime; **RYTTER**, Wojciech. *Jewels of Stringology: Text Algorithms*, WSPC, 2002.
5. **HALIM**, Steve; **HALIM**, Felix. *Competitive Programming 3*, Lulu, 2013.
6. RegexOne. [Lesson 1: An Introduction, and the ABCs](#), acesso em 15/03/2019.
7. Wikipedia. [Tokenização](#), acesso em 22/01/2017.