

# Strings e Programação Dinâmica

Maior Subsequência Palíndroma

---

Prof. Edson Alves - UnB/FGA

1. Maior subsequência palíndroma

## Maior subsequência palíndroma

---

# Definição

- Uma variante da maior sequência comum é o problema de se encontrar a maior subsequência de uma string  $S$  que forma um palíndromo (*Longest Palindrome Subsequence – LPS*)
- Uma maneira de se enunciar este problema é a seguinte: qual é o maior palíndromo que pode ser formado removendo  $m$  ( $0 \leq m \leq n$ ) caracteres, de quaisquer posições, de uma string  $S$  de tamanho  $n$ ?
- Este problema sempre tem solução, pois uma string com apenas um caractere é um palíndromo (o mesmo vale para strings vazias)
- O tamanho do maior palíndromo, ou o palíndromo em si, pode ser determinado por meio de programação dinâmica

## Formulação em programação dinâmica da LPS

- O fato de uma string vazia ou com um único caractere serem palíndromos fornecem os casos bases
- Se  $LPS[i, j]$  é o tamanho da maior subsequência palíndroma da substring  $S[i..j]$ , então

$$LPS[i, i] = 1, \quad LPS[i, j] = 0, \quad \text{se } i > j$$

- São três transições possíveis: a primeira é remover o caractere mais à esquerda de  $S[i..j]$ , isto é,

$$LPS[i, j] = LPS[i + 1, j]$$

- A segunda transição remove o caractere mais à direita  $S[i..j]$ :

$$LPS[i, j] = LPS[i, j - 1]$$

- No último caso, casos os caracteres que estão nos extremos da strings sejam iguais, eles podem ser parte do palíndromo:

$$LPS[i, j] = LPS[i + 1, j - 1] + 2, \quad \text{se } S[i] = S[j]$$

## Implementação *top-down* da LPS

```
8 int dp(const string& s, int i, int j)
9 {
10     if (i > j)
11         return 0;
12
13     if (i == j)
14         return 1;
15
16     if (st[i][j] != -1)
17         return st[i][j];
18
19     st[i][j] = max(dp(s, i + 1, j), dp(s, i, j - 1));
20
21     if (s[i] == s[j])
22         st[i][j] = max(st[i][j], dp(s, i + 1, j - 1) + 2);
23
24     return st[i][j];
25 }
```

# Implementação *top-down* da LPS

```
27 int lps(const string& s)
28 {
29     memset(st, -1, sizeof st);
30
31     return dp(s, 0, s.size() - 1);
32 }
33
34 int main()
35 {
36     string s;
37     cin >> s;
38
39     cout << lps(s) << '\n';
40
41     return 0;
42 }
```

# Identificação da maior subsequência palíndroma

- O algoritmo proposto para a LPS tem complexidade  $O(n^2)$  tanto para a execução quanto para a memória
- Para recuperar a string que corresponde à LPS, é preciso manter o registro das operações utilizadas em cada transição:
  - 'B': os caracteres dos extremos são mantidos
  - 'K': o único caractere da string é mantido
  - 'L': remover o primeiro caractere
  - 'R': remover o último caractere
- Usando um valor sentinela (zero) para estados que não forem atingidos, é possível remontar o palíndromo em  $O(n^2)$



# Implementação da identificação da LPS

```
9 int dp(const string& s, int i, int j)
10 {
11     if (i > j)
12         return 0;
13
14     if (i == j) {
15         ps[i][j] = 'K';
16         return 1;
17     }
18
19     if (st[i][j] != -1)
20         return st[i][j];
21
22     st[i][j] = max(dp(s, i + 1, j), dp(s, i, j - 1));
23     ps[i][j] = dp(s, i + 1, j) > dp(s, i, j - 1) ? 'L' : 'R';
24
25     if (s[i] == s[j]) {
26         st[i][j] = max(st[i][j], dp(s, i + 1, j - 1) + 2);
27         ps[i][j] = st[i][j] > dp(s, i + 1, j - 1) + 2 ? ps[i][j] : 'B';
28     }
```

# Implementação da identificação da LPS

```
30     return st[i][j];
31 }
32
33 string lps(const string& s)
34 {
35     memset(st, -1, sizeof st);
36     memset(ps, 0, sizeof ps);
37
38     int n = s.size();
39
40     dp(s, 0, n - 1);
41
42     int i = 0, j = n - 1;
43     string L = "", R = "";
44
45     while (i <= j)
46     {
47         auto p = ps[i][j];
```

# Implementação da identificação da LPS

```
49     switch (p) {  
50     case 'L':  
51         ++i;  
52         break;  
53  
54     case 'R':  
55         --j;  
56         break;  
57  
58     case 'K':  
59         L += s[i];  
60         ++i;  
61         break;  
62  
63     default:  
64         L += s[i]; R = s[i] + R;  
65         ++i; --j;  
66         break;  
67     }  
68 }
```

# Implementação da identificação da LPS

```
70     return L + R;  
71 }  
72  
73 int main()  
74 {  
75     string s;  
76     cin >> s;  
77  
78     cout << lps(s) << '\n';  
79  
80     return 0;  
81 }
```

1. **HALIM**, Steve; **HALIM**, Felix. *Competitive Programming 3*, Lulu, 2013.
2. **CROCHEMORE**, Maxime; **RYTTER**, Wojciech. *Jewels of Stringology: Text Algorithms*, WSPC, 2002.