Strings

String e Programação Dinâmica – Maior Subsequência Palíndroma

Prof. Edson Alves - UnB/FGA

2019

Sumário

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

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

- 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:

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

Implementação *top-down* da LPS

```
5  const int MAX { 1001 };
6  int st[MAX][MAX];
7
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

```cpp
#include <bits/stdc++.h>

using namespace std;

const int MAX { 1001 };
int st[MAX][MAX];
char ps[MAX][MAX];

int dp(const string& s, int i, int j)
{
    if (i > j)
        return 0;

    if (i == j)
    {
        ps[i][j] = 'K';
        return 1;
    }

    if (st[i][j] != -1)
        return st[i][j];
```

Implementação da identificação da LPS

```
23      st[i][j] = max(dp(s, i + 1, j), dp(s, i, j - 1));
24      ps[i][j] = dp(s, i + 1, j) > dp(s, i, j - 1) ? 'L' : 'R';
25
26      if (s[i] == s[j])
27      {
28          st[i][j] = max(st[i][j], dp(s, i + 1, j - 1) + 2);
29          ps[i][j] = st[i][j] > dp(s, i + 1, j - 1) + 2 ? ps[i][j] : 'B';
30      }
31
32      return st[i][j];
33  }
34
35  string lps(const string& s)
36  {
37      memset(st, -1, sizeof st);
38      memset(ps, 0, sizeof ps);
39
40      int n = s.size();
41
42      dp(s, 0, n - 1);
43
```

Implementação da identificação da LPS

```
44      int i = 0, j = n - 1;
45      string L = "", R = "";
46
47      while (i <= j)
48      {
49          auto p = ps[i][j];
50
51          switch (p) {
52          case 'L':
53              ++i;
54              break;
55
56          case 'R':
57              --j;
58              break;
59
60          case 'K':
61              L += s[i];
62              ++i;
63              break;
64
```

Implementação da identificação da LPS

```
65          default:
66              L += s[i];
67              R = s[i] + R;
68              ++i;
69              --j;
70              break;
71          }
72      }
73
74      return L + R;
75  }
76
77  int main()
78  {
79      string s;
80      cin >> s;
81
82      cout << lps(s) << '\n';
83
84      return 0;
85  }
```

Referências

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