Strings

String e Programação Dinâmica – Maior Subsequência Comum

Prof. Edson Alves - UnB/FGA

2019

Sumário

Maior Subsequência Comum

Definição

- Uma subsequência comum $b = sc(S, T)$ entre duas strings S e T é uma sequência de pares de índices (i_k, j_k) tais que

$$1 \leq i_k \leq |S|, 1 \leq j_k \leq |T| \quad \text{e} \quad S[i_k] = T[j_k],$$

  para todo $k = 1, 2, \ldots, |b|$
- Por exemplo, se $S =$ "casa" e $T =$ "nevasca", então

$$b_1 = \{(3, 5)\}, b_2 = \{(1, 6), (2, 7) \quad \text{e} \quad b_3 = \{(2, 4), (3, 5), (4, 7)\}$$

  são subsquências comuns de S e T
- O problema de se determinar a maior subsequência comum entre S e T (*Longest Common Subsequence – LCS*) consiste em determinar o maior elemento do conjunto dos tamanhos substrings comuns, isto é,

$$LCS(S, T) = \max\{|b| \mid b = sc(S, T)\}$$

- Observe que pode existir duas (ou mais) subsequências comuns b_1 e b_2 de S e T tais que $|b_1| = |b_2| = LCS(S, T)$

LCS e *edit distance*

- O LCS pode ser interpretado como uma variante do *edit distance*
- Basta notar que uma sequência b de índices tal que $|b| = LCS(S, T)$ é formada por caracteres comuns às duas strings
- Se atribuídos pesos iguais a zero às operações de inserção e remoção, peso infinitamente negativo à substituição e peso 1 à opção de manter os caracteres iguais, a LCS surge como o caminho com maior custo na tabela da $edit(S, T)$
- Isto porque, com a atribuição de pesos descrita, serão mantido o maior número de caracteres comuns entre ambas possível, e as inserções e remoções, de custo zero, completarão a transformação
- Esta abordagem tem complexidade $O(nm)$

Visualização de $LCS(S,T)$

$LCS(i,j)$		'n'	'e'	'v'	'a'	's'	'c'	'a'
	0	0	0	0	0	0	0	0
'c'	0	0	0	0	0	0	1	1
'a'	0	0	0	0	1	1	1	2
's'	0	0	0	0	1	2	2	2
'a'	0	0	0	0	1	2	2	3

Implementação da LCS em C++

```cpp
int LCS(const string& s, const string& t)
{
    const int c_i = 0, c_r = 0, c_s = 1;      // Custos modificados
    int m = s.size(), n = t.size();

    for (int i = 0; i <= m; ++i)
        st[i][0] = i*c_r;

    for (int j = 1; j <= n; ++j)
        st[0][j] = j*c_i;

    for (int i = 1; i <= m; ++i)
        for (int j = 1; j <= n; ++j) {
            int insertion = st[i][j - 1] + c_i;
            int deletion = st[i-1][j] + c_r;
            int change = st[i-1][j-1] + c_s*(s[i-1] == t[j-1] ? 1 : -oo);
            st[i][j] = max({ insertion, deletion, change });
        }

    return st[m][n];
}
```

Variantes da LCS

Identificação da LCS

- Assim como o problema de *edit distance*, uma variante comum do LCS é determinar a sequência de operações que leva à maior subsequência comum
- A implementação é idêntica à proposta para $edit(S, T)$, uma vez aplicada a modificação dos pesos e a alteração da operação `min()` por `max()`, de modo que a complexidade permanece sendo $O(nm)$
- A maior subsequência comum corresponde aos caracteres onde os caracteres foram mantidos
- Assim esta rotina pode ser modificada para exibir a sequência, e não as operações que levaram a ela

Identificação da LCS em C++

```
74 // -         Deletion
75 // c         Insertion of char c
76 // =         Keep
77 // [c->d]    Change (c to d)
78 string LCS(const string& s, const string& t)
79 {
80     const int c_i = 0, c_r = 0, c_s = 1;      // Custos modificados
81     int m = s.size(), n = t.size();
82
83     for (int i = 0; i <= m; ++i)
84     {
85         st[i][0] = i*c_r;
86         ps[i][0] = 'r';
87     }
88
89     for (int j = 1; j <= n; ++j)
90     {
91         st[0][j] = j*c_i;
92         ps[0][j] = 'i';
93     }
94
```

Identificação da LCS em C++

```
95      for (int i = 1; i <= m; ++i)
96          for (int j = 1; j <= n; ++j) {
97              int insertion = st[i][j - 1] + c_i;
98              int deletion = st[i-1][j] + c_r;
99              int change = st[i-1][j-1] + c_s*(s[i-1] == t[j-1] ? 1 : -oo);
100             st[i][j] = max({ insertion, deletion, change });
101
102             ps[i][j] = (insertion >= deletion and insertion >= change) ?
103                 'i' : (deletion >= change ? 'r' : 's');
104         }
105
106     int i = m, j = n;
107     string b;
108
109     while (i or j)
110     {
111         switch (ps[i][j]) {
112         case 'i':
113             --j;
114             break;
115
```

Identificação da LCS em C++

```
116             case 'r':
117                 --i;
118                 break;
119
120             case 's':
121                 if (s[i-1] == t[j-1])
122                     b.push_back(s[i-1]);
123
124                 --i;
125                 --j;
126             }
127         }
128
129         reverse(b.begin(), b.end());
130
131         return b;
132 }
```

Referências

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