Strings e Programação Dinâmica

Maior Subsequência Comum

Prof. Edson Alves - UnB/FGA

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 $k = 1, 2, \ldots, |b|$ onde $i_1 < i_2 < \ldots < i_{|b|}$ e $j_1 < j_2 < \ldots < j_{|b|}$

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 $k = 1, 2, \ldots, |b|$ onde $i_1 < i_2 < \ldots < i_{|b|}$ e $j_1 < j_2 < \ldots < j_{|b|}$

- Por exemplo, se $S =$ "casa" e $T =$ "nevasca", então

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

são subsequências comuns de S e T

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 $k = 1, 2, \ldots, |b|$ onde $i_1 < i_2 < \ldots < i_{|b|}$ e $j_1 < j_2 < \ldots < j_{|b|}$

- Por exemplo, se $S =$ "casa" e $T =$ "nevasca", então

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

são subsequências 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

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

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 $k = 1, 2, \ldots, |b|$ onde $i_1 < i_2 < \ldots < i_{|b|}$ e $j_1 < j_2 < \ldots < j_{|b|}$

- Por exemplo, se $S =$ "casa" e $T =$ "nevasca", então

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

são subsequências 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

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

- 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*

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

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 para a opção de manter os caracteres iguais, a LCS surge como o caminho com maior custo na tabela da $edit(S,T)$

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 para a 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, será mantido o maior número de caracteres comuns entre ambas possível, e as inserções e remoções, de custo zero, completarão a transformação

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 para a 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, será 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, 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

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 $O(nm)$

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 índices onde os caracteres foram mantidos

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 índices 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++

```
string 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;
        ps[i][0] = 'r';
    }

    for (int j = 1; j <= n; ++j) {
        st[0][j] = j*c_i;
        ps[0][j] = 'i';
    }

    for (int i = 1; i <= m; ++i)
        for (int j = 1; j <= n; ++j) {
            int insertion = st[i][j - 1] + c_i, 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 });
```

Identificação da LCS em C++

```cpp
            ps[i][j] = (insertion >= deletion and insertion >= change) ?
                'i' : (deletion >= change ? 'r' : 's');
        }

    int i = m, j = n;
    string b;

    while (i or j) {
        switch (ps[i][j]) {
        case 'i':
            --j;
            break;

        case 'r':
            --i;
            break;

        case 's':
            if (s[i - 1] == t[j - 1])
                b.push_back(s[i - 1]);
```

Identificação da LCS em C++

```
            --i;
            --j;
        }
    }

    reverse(b.begin(), b.end());

    return b;
}
```

LCS e LIS

- Quando todos os caracteres de S e de T são distintos (isto é, $S[i] \neq S[j]$ se $i \neq j$, o mesmo para T), o problema de se determinar a LCS pode ser reduzido ao problema de se determinar a maior sequência crescente (*Longest Increasing Subsequence – LIS*)

LCS e LIS

- Quando todos os caracteres de S e de T são distintos (isto é, $S[i] \neq S[j]$ se $i \neq j$, o mesmo para T), o problema de se determinar a LCS pode ser reduzido ao problema de se determinar a maior sequência crescente (*Longest Increasing Subsequence – LIS*)
- Para tal, seja $\{a_i\}$ a sequência crescente de índices de S tais que $S[a_i] = T[j]$ para algum $j \in [1, m]$

LCS e LIS

- Quando todos os caracteres de S e de T são distintos (isto é, $S[i] \neq S[j]$ se $i \neq j$, o mesmo para T), o problema de se determinar a LCS pode ser reduzido ao problema de se determinar a maior sequência crescente (*Longest Increasing Subsequence – LIS*)
- Para tal, seja $\{a_i\}$ a sequência crescente de índices de S tais que $S[a_i] = T[j]$ para algum $j \in [1, m]$
- Em outras palavras, $\{a_i\}$ é sequência crescente de índices de caracteres de S que coincidem com algum dos caracteres de T

LCS e LIS

- Quando todos os caracteres de S e de T são distintos (isto é, $S[i] \neq S[j]$ se $i \neq j$, o mesmo para T), o problema de se determinar a LCS pode ser reduzido ao problema de se determinar a maior sequência crescente (*Longest Increasing Subsequence – LIS*)
- Para tal, seja $\{a_i\}$ a sequência crescente de índices de S tais que $S[a_i] = T[j]$ para algum $j \in [1, m]$
- Em outras palavras, $\{a_i\}$ é sequência crescente de índices de caracteres de S que coincidem com algum dos caracteres de T
- Seja $\{b_k\}$ a sequência tal que $b_k = j_k$, onde $S[a_k] = T[j_k]$

LCS e LIS

- Quando todos os caracteres de S e de T são distintos (isto é, $S[i] \neq S[j]$ se $i \neq j$, o mesmo para T), o problema de se determinar a LCS pode ser reduzido ao problema de se determinar a maior sequência crescente (*Longest Increasing Subsequence – LIS*)
- Para tal, seja $\{a_i\}$ a sequência crescente de índices de S tais que $S[a_i] = T[j]$ para algum $j \in [1, m]$
- Em outras palavras, $\{a_i\}$ é sequência crescente de índices de caracteres de S que coincidem com algum dos caracteres de T
- Seja $\{b_k\}$ a sequência tal que $b_k = j_k$, onde $S[a_k] = T[j_k]$
- A LIS de $\{b_k\}$ corresponderá a LCS entre as duas strings

LCS e LIS

- Quando todos os caracteres de S e de T são distintos (isto é, $S[i] \neq S[j]$ se $i \neq j$, o mesmo para T), o problema de se determinar a LCS pode ser reduzido ao problema de se determinar a maior sequência crescente (*Longest Increasing Subsequence – LIS*)
- Para tal, seja $\{a_i\}$ a sequência crescente de índices de S tais que $S[a_i] = T[j]$ para algum $j \in [1, m]$
- Em outras palavras, $\{a_i\}$ é sequência crescente de índices de caracteres de S que coincidem com algum dos caracteres de T
- Seja $\{b_k\}$ a sequência tal que $b_k = j_k$, onde $S[a_k] = T[j_k]$
- A LIS de $\{b_k\}$ corresponderá a LCS entre as duas strings
- A vantagem desta abordagem é que, enquanto a LCS tem implementação $O(nm)$, a LIS pode ser implementada em $O(n \log n)$

Visualização de $LCS(S, T)$ como LIS

i	1	2	3	4	5	6	7	8	9	10
S	't'	'r'	'a'	'p'	'e'	'z'	'i'	'o'		
T	'r'	'e'	't'	'i'	'c'	'u'	'l'	'a'	'd'	'o'
a_i										
b_i										

Visualização de $LCS(S, T)$ como LIS

i	1	2	3	4	5	6	7	8	9	10
S	't'	'r'	'a'	'p'	'e'	'z'	'i'	'o'		
T	'r'	'e'	't'	'i'	'c'	'u'	'l'	'a'	'd'	'o'
a_i	1									
b_i	3									

Visualização de $LCS(S, T)$ como LIS

i	1	2	3	4	5	6	7	8	9	10
S	't'	'r'	'a'	'p'	'e'	'z'	'i'	'o'		
T	'r'	'e'	't'	'i'	'c'	'u'	'l'	'a'	'd'	'o'
a_i	1	2								
b_i	3	1								

Visualização de $LCS(S, T)$ como LIS

i	1	2	3	4	5	6	7	8	9	10
S	't'	'r'	'a'	'p'	'e'	'z'	'i'	'o'		
T	'r'	'e'	't'	'i'	'c'	'u'	'l'	'a'	'd'	'o'
a_i	1	2	3							
b_i	3	1	8							

Visualização de $LCS(S, T)$ como LIS

i	1	2	3	4	5	6	7	8	9	10
S	't'	'r'	'a'	'p'	'e'	'z'	'i'	'o'		
T	'r'	'e'	't'	'i'	'c'	'u'	'l'	'a'	'd'	'o'
a_i	1	2	3	5						
b_i	3	1	8	2						

Visualização de $LCS(S,T)$ como LIS

i	1	2	3	4	5	6	7	8	9	10
S	't'	'r'	'a'	'p'	'e'	'z'	'i'	'o'		
T	'r'	'e'	't'	'i'	'c'	'u'	'l'	'a'	'd'	'o'
a_i	1	2	3	5	7					
b_i	3	1	8	2	4					

Visualização de $LCS(S,T)$ como LIS

i	1	2	3	4	5	6	7	8	9	10
S	't'	'r'	'a'	'p'	'e'	'z'	'i'	'o'		
T	'r'	'e'	't'	'i'	'c'	'u'	'l'	'a'	'd'	'o'
a_i	1	2	3	5	7	8				
b_i	3	1	8	2	4	10				

Visualização de $LCS(S,T)$ como LIS

i	1	2	3	4	5	6	7	8	9	10
S	't'	'r'	'a'	'p'	'e'	'z'	'i'	'o'		
T	<u>'r'</u>	<u>'e'</u>	't'	<u>'i'</u>	'c'	'u'	'l'	'a'	'd'	<u>'o'</u>
a_i	1	2	3	5	7	8				
b_i	3	$\boxed{1}$	8	$\boxed{2}$	$\boxed{4}$	$\boxed{10}$				

$LIS(b_k) = \{1, 2, 4, 10\}$

$LCS(S,T) =$"reio"

Implementação da LCS como LIS em C++

```cpp
int lis(const vector<int>& as)
{
    vector<int> st(as.size(), -1);
    int n = 0;

    for (auto x : as)
    {
        if (x > st[n])  // Aumenta a LIS, quando possível
            st[++n] = x;
        else {          // Melhora os elementos já encontrados
            auto it = lower_bound(st.begin() + 1, st.begin() + n, x);
            *it = min(*it, x);
        }
    }

    return n;
}
```

Implementação da LCS como LIS em C++

```cpp
int lcs(const string& S, const string& T)
{
    map<char, int> idx;

    for (size_t i = 0; i < T.size(); ++i)
        idx[T[i]] = i;

    vector<int> bs;

    for (const auto& c : S)
    {
        auto it = idx.find(c);

        if (it != idx.end())
            bs.push_back(it->second);
    }

    return lis(bs);
}
```

Referências

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