Paradigmas de Resolução de Problemas

Programação Dinâmica: Maior Subsequência Crescente

Prof. Edson Alves - UnB/FGA

Sumário

- 1. Definição
- 2. Solução do problema da maior subsequência crescente
- 3. Variantes

Definição

Definição

Problema da Maior Subsequência Crescente

Considere uma sequência $a = \{a_1, a_2, \dots, a_N\}.$

Uma subsequência

$$b = \{b_1, b_2, \dots, b_k\} = \{a_{i_1}, a_{i_2}, \dots, a_{i_k}\}$$

de a é a maior subsequência crescente (longest increasing subsequence – LIS) se valem as seguintes condições:

- i. $i_1 < i_2 < \ldots < i_k$,
- ii. $b_i < b_j$ se i < j, e
- iii. k é máximo.

Características da maior subsequência crescente

- O problema da maior subsequência crescente tem solução para qualquer sequência a, uma vez que qualquer subsequência composta por um único elemento é uma subsequência crescente (não necessariamente a maior)
- A maior subsequência não é única: por exemplo, a sequência $a=\{4,1,5,2,6,3\}$ tem várias subsequências com três elementos ($b_1=\{4,5,6\}$ e $b_2=\{1,2,3\}$ são duas delas)
- $\bullet\,$ Os elementos de a pode ser de qualquer tipo, desde que o operador < esteja definido

Solução do problema da maior

subsequência crescente

Solução quadrática para a LIS

- Uma sequência $a=\{a_1,a_2,\ldots,a_N\}$ tem 2^N subsequências distintas, de modo que um algoritmo de busca completa só seria efetivo para valores de N pequenos
- A LIS de uma sequência a pode ser determinada por meio de um algoritmo de programação dinâmica
- ullet Seja lis(i) o tamanho da maior subsequência de a cujo último elemento é a_i
- O caso base acontece quando i = 1: lis(1) = 1
- \bullet A transição deve avaliar todas as subsequências anteriores que podem eventualmente serem estendidas por a_i

Solução quadrática para a LIS

Deste modo,

$$lis(i) = \max\{1, lis(a_k) + 1\},\,$$

para todo $k \in [1,i)$ tal que $a_k < a_i$

- ullet Assim, cada transição é feita em O(N) e há O(N) estados distintos
- ullet Portanto esta solução tem complexidade $O(N^2)$
- $\bullet \ \ {\rm A \ complexidade \ de \ mem\'oria \ \'e} \ O(N)$

Implementação da solução quadrática da LIS

```
5 int LIS(int N, const vector<int>& xs)
6 {
     vector<int> lis(N, 1);
7
8
     for (int i = 1; i < N; ++i)
9
10
          for (int j = i - 1; j >= 0; --j)
12
              if (xs[i] > xs[j])
13
                  lis[i] = max(lis[i], lis[j] + 1);
14
16
      return *max_element(lis.begin(), lis.end());
18
19 }
```

Recuperação dos elementos da LIS

- ullet É possível explicitar os elementos da LIS por meio de O(N) de memória adicional
- ullet Seja ps(i) o índice do penúltimo elemento da sequência terminada em a_i
- Se a sequência terminada em a_i contém um único elemento, faça ps(i)=-1 (ou qualquer outro valor sentinela)
- Assim, caso a_i possa estender uma sequência, o valor de ps(i) deve ser devidamente atualizado
- ullet O vetor ps poderá ser utilizado para recuperar os elementos de uma LIS

Recuperação dos elementos da LIS

```
1 #include <bits/stdc++.h>
₃ using namespace std;
5 vector<int> LIS(int N, const vector<int>& xs)
6 {
      vector\langle int \rangle lis(N, 1), ps(N, -1);
7
      for (int i = 1; i < N; ++i)
10
          for (int j = i - 1; j >= 0; --j)
               if (xs[i] > xs[j] and lis[j] + 1 > lis[i]) {
                   lis[i] = lis[j] + 1;
14
                   ps[i] = j;
16
1.8
      int best = 0. k = -1:
20
```

Recuperação dos elementos da LIS

```
for (int i = 0; i < N; ++i)
22
23
          if (lis[i] > best)
24
25
              best = lis[i];
26
              k = i:
27
28
29
30
      vector<int> ans:
31
32
      do {
33
          ans.push_back(xs[k]);
34
          k = ps[k];
35
      } while (k != -1);
36
37
      reverse(ans.begin(), ans.end());
38
39
40
      return ans;
41 }
```

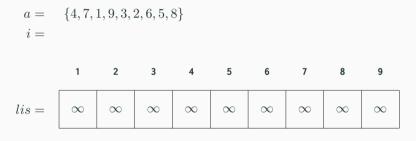
Variantes

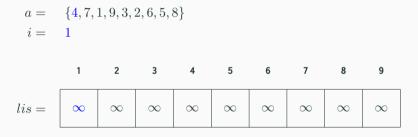
Solução linearítmica

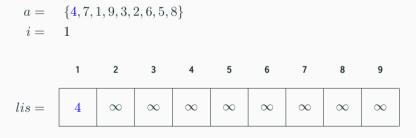
- A LIS pode ser encontrada por um algoritmo de programação dinâmica linearítmico
- Este algoritmo se baseia em um estado diferente do utilizado no algoritmo quadrático e em uma transição mais eficiente
- Seja lis(k,i) o menor elemento que finaliza uma subsequência crescente de $\{a_1,a_2,\ldots,a_i\}$ de tamanho k
- A segunda dimensão deste estado será implícita, sem necessidade de alocação de memória (caso contrário, a alocação da tabela de memória já tornaria este algoritmo quadrático)
- ullet Os casos bases acontecem com i=0, isto é, antes de se considerar qualquer elemento da sequência

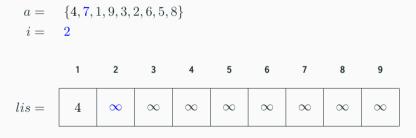
Solução linearítmica

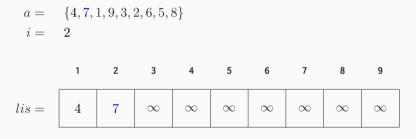
- Para simplificar a implementação, pode-se fazer $lis(k,0)=\infty$, para todo $k\in[1,N]$ e lis(0,0)=0 (ou qualquer outro valor sentinela, desde que seja estritamente menor do que qualquer a_i)
- ullet Para cada i, apenas um dos lis(k,i) será atualizado
- Importante notar que os elementos da sequência $lis(1,i), lis(2,i), \ldots$ estarão em ordem crescente
- Por meio de uma busca binária, deve-se identificar o primeiro índice j tal que lis(j,i-1) seja estritamente maior do que a_i
- Daí, $lis(j,i) = a_i$
- O tamanho da LIS será igual ao maior índice j tal que $lis(j,N)<\infty$

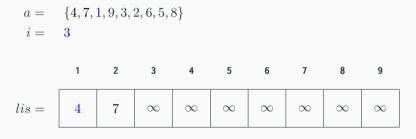


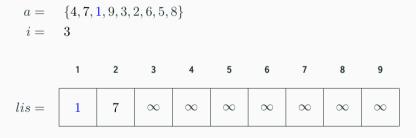


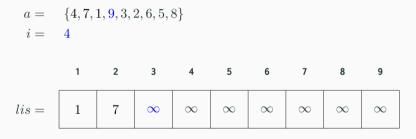


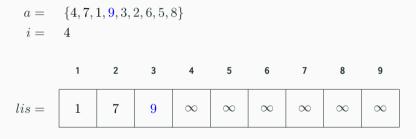


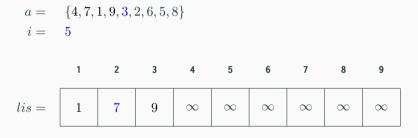


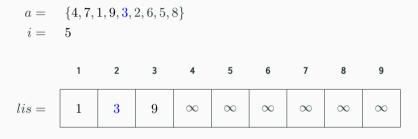


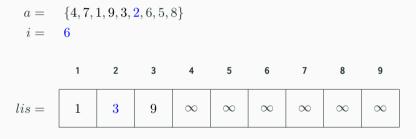


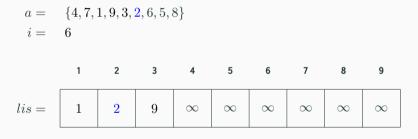


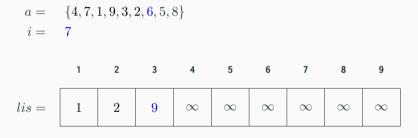


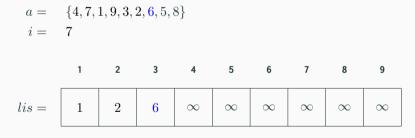


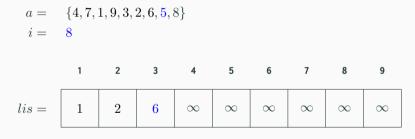


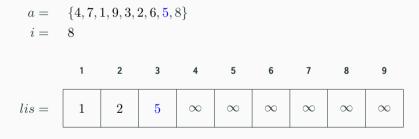


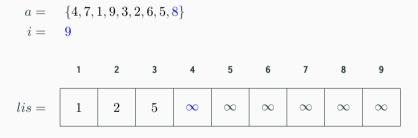


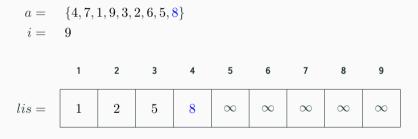












Implementação linearítmica da LIS

```
5 const int oo { 2000000010 };
7 int LIS(int N, const vector<int>& as)
8 {
      vector<int> lis(N + 1, oo);
9
      lis[0] = -oo;
10
      auto ans = 0;
      for (int i = 0; i < N; ++i)
14
15
          auto it = lower_bound(lis.begin(), lis.end(), as[i]);
16
          auto pos = (int) (it - lis.begin());
1.8
          ans = max(ans, pos);
19
          lis[pos] = as[i];
20
21
23
      return ans;
24 }
```

Referências

- 1. HALIM, Steve; HALIM, Felix. Competitive Programming 3, Lulu, 2013.
- 2. **LAARKSONEN**, Antti. Competitive Programmer's Handbook, 2017.