Dawid Pawliczek

Notatka – *Longest Increasing Subsequence*

Treść

Dla danej tablicy $A[1 \dots n]$ należy znaleźć *najdłuższy ściśle rosnący podciąg* (nie koniecznie spójny!) jej elementów.

Rozwiązanie $O(n^2)$ – dynamiczne

Definicja stanu.

$$\mathrm{dp}[i] \;=\; \text{długość LIS kończącego się dokładnie na pozycji } i.$$

$$\mathrm{dp}[i] = 1 + \max_{\substack{j<i \\ A[j]<A[i]}} \mathrm{dp}[j], \quad \mathrm{dp}[i] = 1 \text{ jeśli brak takiego } j.$$

Algorithm 1 NaiveLIS

Require: $A[1 \dots n]$
Ensure: długość LIS i jeden taki podciąg

```
for i ← 1 to n:
    dp[i] ← 1, prev[i] ← 0
    for j ← 1 to i − 1:
if A[j] < A[i] and dp[j] + 1 > dp[i] then
            dp[i] ← dp[j] + 1
            prev[i] ← j
end if
best ← arg max_i dp[i]
odtwarzaj ciąg cofając się po wskaźnikach prev
return (dp[best], ciąg)
```

Odzyskiwanie podciągu. Tablica `prev` wskazuje poprzedni wybrany indeks. Startujemy od `best` i idziemy wstecz, odwracając otrzymaną listę – otrzymujemy poprawny LIS.

Rozwiązanie $O(n \log n)$ – patience sorting

Idea. Utrzymujemy tablicę tail[l] – najmniejszą możliwą *wartość końcową* rosnącego podciągu długości l. Wektor `tail` jest rosnący, więc aktualizacje wykonujemy wyszukiwaniem binarnym.

Algorithm 2 FastLIS

Require: $A[1 \dots n]$
Ensure: długość LIS i jeden taki podciąg

```
1:  tail[0] ← −∞;  pos[0] ← 0                          ▷ wartość i pozycja końca
2:  len ← 0
3:  for i ← 1 to n do
4:      l ← najmniejsze p ∈ [1, len] z tail[p] ≥ A[i] (binarne szukanie)
5:      if nie znaleziono then l ← len + 1
6:          tail[l] ← A[i];  pos[l] ← i                 ▷ aktualizacja wartości
7:          pred[i] ← pos[l − 1]                        ▷ poprzednik w LIS
8:          len ← max(len, l)
9:
10:         rekonstrukcja:
            k ← pos[len];  while k ≠ 0: wypisz A[k], k ← pred[k]
            odwróć otrzymaną listę
11:         return (len, ciąg w poprawnej kolejności)
```

Algorytm.

Dlaczego działa? Jeśli mamy dwa podciągi tej samej długości, ten z mniejszym elementem końcowym jest lepszy, bo daje większą szansę przedłużenia. Stąd zawsze warto nadpisać pierwszy element w `tail` *większy równy* od $A[i]$.

Rekonstrukcja. `pos` zapamiętuje indeks w tablicy A, na którym dany ogon `tail` powstał. `pred`[i] wskazuje poprzedni element podciągu kończącego się w $A[i]$. Śledząc wskazania od ostatniego indeksu podciągu długości `len` otrzymujemy LIS w odwrotnym porządku.

Złożoność

$$O(n^2) \text{ — wersja DP}, \qquad O(n \log n) \text{ — wersja szybka}$$

z pamięcią $O(n)$ (obie metody).