Handout 10

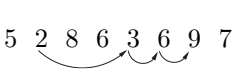
Sebastian Millius, Sandro Feuz, Daniel Graf

Thema: Longest Increasing Subsequence

Longest Increasing Subsequence

Problem: Gegeben ist ein Array A von Zahlen $A[1], ..., A[n]$. Eine *Subsequence* ist eine Teilmenge dieser Zahlen $A[i_1], A[i_2], ..., A[i_k]$ mit $1 \leqslant i_1 < i_2 < ... < i_k \leqslant n$.

Eine *Increasing Subsequence* ist eine Sequenz in der die Zahlen strikt grösser werden. z.B. die längste aufsteigende Sequenz in 5, 2, 8, 6, 3, 6, 9, 7 ist 2, 3, 6, 9



Wir suchen nun die längste aufsteigende Sequenz in einem Array.

1. Struktur

Denken wir darüber nach, was eine optimale Lösung charakterisiert. Betrachten wir das letzte Element (sei dies das Element i im Array, die 9 im obigen Beispiel) in einer längsten aufsteigenden Sequenz. Das vorletzte Element in der Sequenz (die 6 im obigen Beispiel) ist eines, dass im Array vor i kommt, und der Teil der optimalen Lösung ohne das letzte Element ist *eine längste aufsteigende Sequenz*, die beim vorletzten Element in der Sequenz endet.

2. Rekursion

Versuchen wir obige Überlegungen zusammenzufassen.

Sei $lis(i)$ die längste aufsteigende Sequenz die beim Element $A[i]$ endet.

Die Lösung wäre also die längste davon, d.h. $\max_{i \in \{1,..,n\}} lis(i)$ (wobei das i, welches das maximum erreicht, gerade das letzte Element der Lösung ist).

Formuliere $lis(i)$ rekursiv:

$lis(i) = 1 + \min_{j \in \{1,..,i-1\} \text{ mit } A[j]<A[i]} lis(j)$

Wir gehen also alle Elemente vor dem Element i durch und wählen von denen die kleiner als das i-te Element sind, dasjenige aus, bei dem die längste bisherige aufsteigende Sequenz endet. Daraus können wir eine um eins längere aufsteigende Sequenz bis zum i-ten Element bilden (wir hängen das i-te Element einfach hinten dran).

3. Bottom-Up

```
LIS(A, n)
    // Berechne die Länge der longest increasing subsequence

    for i = 1 to n
        lis[i] = 1                  // Finde längste Sequenz mit Schlusselement i
        pre[i] = −1                 // Vorgänger in der Sequenz
        for j = 1 to i − 1
            if A[j] < A[i] and lis[j] + 1 > lis[i]
                lis[i] = lis[j] + 1
                pre[i] = j

    return max_i lis[i]
```

Die Laufzeit ist (*think about it!*) $O(n^2)$

4. Lösung berechnen

Das letzte Element der Lösung ist gerade dasjenige mit dem grössten $lis[]$ Wert, also $\arg\max_i lis[i]$. Es ist dann sehr einfach die Sequenz zu rekonstruieren, wenn wir das Feld $pre[i]$ nutzen, in welchem jeweils der direkte Vorgänger von i in der längsten Sequenz mit Endelement i steht.

LIS2(lis, pre, n)
  // Berechne eine Instanz der longest increasing subsequence

  $path = ""$
  $i = \arg\max_i\{lis[i]\}$
  while $i \neq -1$
    $path = i + "," + path$ // i vorne an den Pfad hängen
    $i = pre[i]$

  return $path$

Lösung in $\mathcal{O}(n \log n)$

Der Algorithmus kann auf eine Laufzeit von $\mathcal{O}(n \log n)$ verbessert werden mit Hilfe von folgenden Beobachtungen: Es sei t_j^i der kleinste Wert bei dem eine Increasing Subsequence der Länge j endet auf der Sequenz $A[1], \ldots, A[i]$.

Beobachtung: Für jedes i gilt, dass $t_1^i < t_2^i < \ldots < t_j^i$, d.h. diese Werte liegen immer sortiert vor! Dies legt nahe, dass wenn wir die längste aufsteigende Teilsequenz finden möchten, die mit $A[i]$ endet, wir nach dem j suchen müssen, so dass $t_j^{i-1} < A[i] < t_{j+1}^{i-1}$ ist! Die Länge der längsten aufsteigenden Teilsequenz, die bei $A[i]$ endet, ist damit $j + 1$.

Dann ist $t_{j+1}^i = A[i]$ und $t_k^i = t_k^{i-1}$ für alle $k \neq j + 1$. Da die t_j^i immer in sortierter Reihenfolge vorliegen, und eine Änderung die Sortierheit bewahrt, kann in jedem Schritt eine binäre Suche durchgeführt werden, um j zu bestimmen.

Die Laufzeit ist damit $\mathcal{O}(n \log n)$ (vgl. Musterlösung zur Programmieraufgabe).

Siehe http://www.algorithmist.com/index.php/Longest_Increasing_Subsequence (http://goo.gl/1s87F) für eine Implementierung in C++.