Najdlhšia vybraná rastúca podpostupnosť (Dynamické programovanie)

Matej Perejda

Úloha: Daná je postupnosť N čísel: $P[0]$, $P[1]$,...,$P[N-1]$. Nájdite takú podpostupnosť (nemusí byť súvislá), aby všetky hodnoty vybranej podpostupnosti tvorili rastúcu postupnosť hodnôt a vybraná postupnosť bola najdlhšia možná.

Definícia: Vstupom P je postupnosť N čisel $P[0]$, $P[1]$,...,$P[N-1]$. Pod najdlhšou vybranou rastúcou podpostupnosťou budeme rozumieť ľubovoľnú podmnožinu danej postupnosti v ktorej sú hodnoty zoradené v poradí od najmenšej po najväčšiu a v ktorej je podpostupnosť najdlhšia možná. Táto podpostupnosť nemusí byť súvislá a taktiež jediná, ktorá existuje.

Príklad:
Vstup P: 10, 22, 9, 33, 21, 50, 41, 60, 80
Rastúca podpostupnosť A: 9, 21, 50, 60, 80
Najdlhšia rastúca podpostupnosť B: 10, 22, 33, 50, 60, 80

Našli sme rastúcu podpostupnosť A dĺžky 5. Táto podpostupnosť nie je najdlhšia možná. Všimnime si, že existuje rastúca podpostupnosť B dĺžky 6, ktorá je zároveň najdlhšou rastúcou podpostupnosťou postupnosti P. Žiadna dlhšia rastúca podpostupnosť vstupnej postupnosti P neexistuje.
Nie je to jediné existujúce riešenie tejto dlžky. Ďalšia (6 prvková) vybraná rastúca podpostupnosť je napríklad 10, 22, 33, 41, 60, 80.

Riešenie: Jedným zo spôsobov ako túto úlohu riešiť je využiť dynamické programovanie.
Nech P je postupnosť N čísel: $P[0]$, $P[1]$,...,$P[N-1]$. Definujme $D[i]$ ako premennú, ktorá reprezentuje dĺžku najdlhšej rastúcej podpostupnosti danej postupnosti P, ktorá končí vo zvolenom i-tom prvku.

D[i] - dĺžka najdlhšej vybranej rastúcej podpostupnosti, ktorá končí v i-tom prvku

Zo vstupu P vyberieme ľubovoľnú rastúcu podpostupnosť končiacu na indexe i (zvolíme si napr. $i = 6$). Podpostupnosť je vyznačená červenou farbou, prvok na i-tom indexe modrou farbou. Uvažujme teda o najdlhšej vybranej rastúcej podpostupnosti, ktorá končí vo zvolenom prvku postupnosti na i-tej pozícii (a obsahuje ho).

Vstup P: 10, 22, 9, 33, 21, 50, 41, 60, 80
Najdlhšia vybraná rastúca podpostupnosť končiaca v $P[i] = P[6]$: 10, 22, 33, 41

Týmto sme dostali optimálne riešenie, ktoré predĺžuje o jeden prvok (41) nejakú **najdlhšiu vybranú rastúcu podpostupnosť**. Nevieme, ktorú podpostupnosť predlžujeme, ale určite vieme, že sa končí v nejakom prvku nachádzajúcom sa vľavo od 41 a logicky(kedže ide o **rastúcu** podpostupnosť) menšom ako 41.

Následne, najdlhšiu rastúcu podpostupnosť dokážeme nájsť dynamickým programovaním v čase $O(N^2)$ pomocou nasledujúcej relácie:

$$D[i] = 1 + \max\{0, D[j] | 0 \leq j < i \wedge P[j] < P[i]\}$$

K určeniu hodnoty $D[i]$ dospejeme vypočítaním hodnôt $D[j]$ pre $j < i$ postupne pre $D[0]$, $D[1]$, $D[2]$...

Vstup D: 10, 22, 9, 33, 21, 50, 41, 60, 80

Krok 1: $D[i] \mid i{=}0$ $\mathbf{D[0] = 1}$ (triviálny prípad)

Krok 2: $D[i] \mid i = 1$

$D[1] = 1 + \max\{0, D[0] | 0 \leq j < i \wedge P[j] < P[i]\}$
$D[1] = 1 + \max\{0, 1\}$
$D[1] = 1 + 1 \Rightarrow \mathbf{D[1] = 2}$

Krok 3: $D[i] \mid i = 2$

$D[2] = 1 + \max\{0, D[1] | 0 \leq j < i \wedge P[j] < P[i]\}$
Nie je splnená podmienka P[2] > P[1] ani P[2] > P[0]
$\mathbf{D[2]= 1}$

Krok 4: $D[i] \mid i = 3$

$D[3] = 1 + \max\{0, D[1], D[2] | 0 \leq j < i \wedge P[j] < P[i]\}$
$D[3] = 1 + \max\{0, 1, 2\}$
$D[1] = 1 + 2 \Rightarrow \mathbf{D[3] = 3}$

Krok 5: $D[i] \mid i = 4$

$D[4] = 1 + \max\{0, D[0], D[2] | 0 \leq j < i \wedge P[j] < P[i]\}$
Platí len pre D[0] a D[2], pretože P[0] < P[4] a P[2] < P[4]
$D[4] = 1 + \max\{0, 1, 1\}$
$D[4] = 1 + 1 \Rightarrow \mathbf{D[4] = 2}$

Krok 6: $D[i] \mid i = 5$

$D[5] = 1 + \max\{0, D[0], D[1], D[2], D[3], D[4] | 0 \leq j < i \wedge P[j] < P[i]\}$
$D[5] = 1 + \max\{0, 1, 2, 1, 3, 2\}$
$D[5] = 1 + 3 \Rightarrow \mathbf{D[5] = 4}$

Krok 7: $D[i] \mid i = 6$

$D[6] = 1 + \max\{0, D[0], D[1], D[2], D[3], D[4] | 0 \leq j < i \wedge P[j] < P[i]\}$
Platí pre D[0], D[1], D[2], D[3], D[4]
$D[6] = 1 + \max\{0, 1, 2, 1, 3, 2\}$
$D[6] = 1 + 3 \Rightarrow \mathbf{D[6] = 4}$

Krok 8: $D[i] \mid i = 7$

$D[7] = 1 + \max\{0, D[0], D[1], D[2], D[3], D[4], D[5], D[6]\}$
$D[7] = 1 + \max\{0, 1, 2, 1, 3, 2, 4, 4\}$
$D[7] = 1 + 4 \Rightarrow \mathbf{D[7] = 5}$

Krok 9: $D[i] \mid i = 8$

$D[8] = 1 + \max\{0, D[0], D[1], D[2], D[3], D[4], D[5], D[6], D[7]\}$
$D[8] = 1 + \max\{0, 1, 2, 1, 3, 2, 4, 4, 5\}$
$D[8] = 1 + 5 \Rightarrow \mathbf{D[8] = 6}$

Najdlhšia vybraná rastúca podpostupnosť postupnosti D má dĺžku 6.

Ak už vieme ako vypočítať dĺžku najdlhšej vybranej rastúcej podpostupnosti (NVRP), bude nás zaujímať jej optimálne riešenie. **Ako teda zrekonštruovať NVRP ?** Zadefinujme premenné, ktoré nám dopomôžu pri jej hľadaní.

- $maxDlzka$ - dĺžka doteraz zatiaľ najdlhšej nájdenej vybranej rastúcej podpostupnosti | hodnota na začiatku nastavená ako $maxDlzka = 1$
- $najlepsiKoniec$ - index prvku v ktorom končí doteraz najdlhšia nájdená rastúca podpostupnosť | hodnota na začiatku $najlepsiKoniec = 0$
- $D[\ ]$ - pole celých čísel dĺžky N, ktoré podľa vzťahu na výpočet dĺžky NVRP na každom indexe tohto poľa uchováva dĺžku NVRP končiacu na indexe i. | triviálny prípad $D[0] = 1$
- $predchodca[\ ]$ - pole celých čísel dĺžky N, ktoré na danom indexe i tohto poľa uchováva index predchodcu prvku NVRP, ktorý sa nachádza na indexe i | ak prvok nemá predchodcu $predchodca[0]$ = -1

Počas algoritmu počítania dĺžky NVRP sa tieto hodnoty ukladajú do premenných a postupne sa aktualizujú. Pomocou týchto údajov vieme spätne zrekonštruovať presnú podobu NVRP. Na príklade z predchadzajúcej strany si ukážeme aké hodnoty sa uložia do týchto premenných.

```
Maximálna dĺžka NVRP je:      6
NVRP končí v prvku na indexe: 8

Pole D:            {1, 2, 1, 3, 2, 4, 4, 5, 6}
Pole predchodcov:  {-1, 0, -1, 1, 2, 3, 3, 6, 7}
NVRP:               10 22 33 41 60 80
```

V prvých dvoch riadkoch vidíme, že algoritmus vypočítal dĺžku NVRP a index ($najlepsiKoniec$) na ktorom sa NVRP končí. V kóde za týmto textom, v časti "Rekonštruovanie NVRP" môžeme vidieť ako celá NVRP pomocou týchto pomocných premenných vzniká.

Do zásobníka je najprv vložený prvok, ktorý sa nachádza vo vstupe na indexe $najlepsiKoniec$. Týmto prvkom sa bude končiť celá nájdená NVRP. Nasleduje while cyklus, ktorý hovorí: "kým mám správneho predchodcu (predchodca má kladný index), pridá sa ku mne"– predchodca je pridaný do zásobníka a $najlepsiKoniec$ je aktualizovaný na index tohto pridaného predchodcu. Celý tento cyklus trvá až kým v poli $predchodca[\ ]$ nenarazíme na index rovný -1. Napokon nám v zásobníku ostanú uložené prvky zrekonštruovanej NVRP dlžky, ktorú sme vypočitali. Prvky vypíšeme.

```java
import java.util.Stack;

public class NajdlhsiaVybranaRastucaPodpostupnost {
	// Main
	public static void main(String[] args) {
		int[ ] vstup = { 10, 22, 9, 33, 21, 50, 41, 60, 80 };
		najdiNVRP(vstup);
	}
	// Najde NVRP
	public static void najdiNVRP(int[] vstupnePole) {
		int maxDlzka = 1;
		int najlepsiKoniec = 0;
		int[] D = new int[vstupnePole.length];
		int[] predchodca = new int[vstupnePole.length];
		D[0] = 1;
		predchodca[0] = -1;

		// Relacia na vypocet dlzky
		for (int i = 1; i < vstupnePole.length; i++) {
			D[i] = 1;
			predchodca[i] = -1;
			for (int j = i - 1; j >= 0; j--) {
			    if ((D[j] + 1 > D[i]) && vstupnePole[j] <= vstupnePole[i]) {
					D[i] = 1 + D[j];
					predchodca[i] = j; // uchovavanie predchodcu
				}
			}
			// Aktualizacia dlzky NVRP
			if (D[i] > maxDlzka) {
				maxDlzka = D[i];
				najlepsiKoniec = i;
			}
		}
		// Orientacne vypisy
		System.out.println("Maximalna dlzka NVRP je: " + maxDlzka);
		System.out.println("NVRP konci v prvku na indexe: " + najlepsiKoniec +
		    "\n");
		System.out.print("Pole D: " + "       ");
		vypisPole(D);
		System.out.print("Pole predchodcov: " + " ");
		vypisPole(predchodca);

		// Rekonstruovanie NVRP
		Stack<Integer> stack = new Stack<Integer>();
		System.out.print("NVRP:           ");
		stack.push(vstupnePole[najlepsiKoniec]);
		while (predchodca[najlepsiKoniec] >= 0) {
			stack.push(vstupnePole[predchodca[najlepsiKoniec]]);
			najlepsiKoniec = predchodca[najlepsiKoniec];
		}
		while (!stack.isEmpty()) {
			System.out.print(stack.pop() + " ");
		}
	}
```

```
        private static void vypisPole(int[] poleNaVypis) {
                System.out.print("{");
                for (int i = 0; i < poleNaVypis.length; i++) {
                        if (i != poleNaVypis.length - 1)
                                System.out.print(poleNaVypis[i] + ", ");
                        else
                                System.out.print(poleNaVypis[i]);

                }
                System.out.print("}");
                System.out.println();
        }
}
```