1. *Nejdelší rostoucí posloupnost: NRP nemusí být jednoznačně určena. Jak spočítat, kolik různých NRP obsahuje zadaná posloupnost?*

   Upravíme algoritmus NRP2 z Průvodce[1] tak že si pořídíme pole $D[1 \dots n]$, kde $D[i]$ bude počet nejdelších rostoucích posloupností začínajících v x_i (tedy počet rostoucích posloupností délky $T[i]$, které začínají v x_i). Kdykoliv budeme vylepšovat hodnotu $T[i]$ – tedy najdeme $x_j > x_i$ takové že $T[i] < 1 + T[j]$ – pak nastavíme $D[i] = D[j]$. Kdykoliv ale narazíme na j' takové, že přes $x_{j'}$ získáme posloupnost stejné délky – tj. $T[i] = 1 + T[j]$ – tak k $D[i]$ přičteme $D[j]$. Algoritmus stále poběží v čase $O(n^2)$

   Jde vlastně o úplně stejný postup jako počítání nejdelších (případně nejkratších) cest v acyklickém orientovaném grafu, jen adaptované pro náš problém.

2. *Kopec: Kopcem nazveme podposloupnost, která nejprve roste a pak klesá. Vymyslete algoritmus, který v zadané posloupnosti nalezne nejdelší kopec.*

   Nechť $x_1, x_2, \dots, x_n$ je vstupní posloupnost. Všimněme si, že libovolný kopec je svým maximálním prvkem rozdělený na rostoucí a klesající posloupnost. Pokud bychom tedy věděli, které x_i bude tvořit vrchol nejdelšího kopce, stačí najít největší rostoucí podposloupnost končící v x_i a nejdelší klesající podposloupnost začínající v x_i (což jsou symetrické případy k $T[i]$ z předchozího příkladu).

   V první fázi tedy spočítáme hodnoty $A[i]$ – nejdelší rostoucí podposloupnost končící v x_i, a $B[i]$ – nejdelší klesající podposloupnost začínající v x_i. To zvládneme dvěma symetrickými verzemi algoritmu NPR2 (jednou hledáme rostoucí posloupnost zleva místo zprava, a jednou hledáme klesající posloupnost - vyměníme podmínku $x_i < x_j$ za $x_i > x_j$). Poté už jen najdeme i takové že $A[i] + B[i] - 1$ je maximální (1 odečítáme jelikož x_i jsme započítali dvakrát). Časová složitost tedy zůstane $O(n^2)$, případně $O(n \log n)$, podle toho jak budeme počítat $A[i]$ a $B[i]$.

3. *Knihovna: Mějme posloupnost n knih. Každá kniha má nějakou šířku s_i a výšku v_i. Knihy chceme naskládat do knihovny s nějakým počtem polic tak, abychom dodrželi abecední pořadí. Prvních několik knih tedy půjde na první polici, další část na druhou polici, a tak dále. Máme zadanou šířku knihovny S a chceme rozmístit police tak, aby se do nich vešly všechny knihy a celkově byla knihovna co nejnižší. Tloušťku polic a horní a spodní desky přitom zanedbáváme.*

   Nechť $V[i]$ udává minimální výšku knihovny, která pojme knihy $K_i, K_{i+1}, \dots, K_n$. Stačí umět rekurzivně spočítat tyto hodnoty, jelikož $V[1]$ je hledaná odpověď. Řekněme, že $V[n+1] = 0$ jelikož se ptáme na knihovnu pro prázdnou množinu knížek. Pokud už máme spočítané $V[j]$ pro $j > i$, vyzkoušíme všechny možné podoby první poličky. Tj. $V[i]$ bude minimum přes výšku poličky pro $K_i, K_{i+1}, \dots, K_l$ sečtenou s $V[l+1]$ pro všechna l taková, že šířka knížek $K_i, K_{i+1}, \dots, K_l$ je nejvýše S. Počítáním každého $V[i]$ trávíme nejvýše $O(n)$ času a celkem počítáme n hodnot - časová složitost $O(n^2)$.

4. *Edit v malém prostoru: Algoritmus na výpočet editační vzdálenosti z Průvodce zabere $\Theta(nm)$ buněk paměti na uložení tabulky. Ukažte, jak spotřebu paměti snížit na $\Theta(n + m)$.*

   Algoritmus EDIT2[2] z Průvodce počítá editační vzdálenost vyplňováním tabulky T. Všimněte si, že pro spočítání buňky $T[i, j]$ potřebujeme znát jen buňky $T[i+1, j], T[i+1, j+1]$ a $T[i, j+1]$. Stačí si tedy pamatovat poslední dva řádky na což nám stačí $\Theta(n + m)$ paměti.

[1] s. 287
[2] s. 292

5. *Edit explicitně: Kromě editační vzdálenosti můžeme chtít spočítat i příslušnou nejkratší posloupnost editačních operací. V grafové interpretaci našeho algoritmu je to triviální – prostě vypíšeme nalezenou nejkratší cestu. Ukažte, jak to udělat bez explicitního sestrojení grafu, třeba přímou úpravou algoritmu z Průvodce. Nemusíte se snažit zachovat paměťovou složitost z předchozího příkladu.*

   Krom tabulky $T[i, j]$ si pořídíme tabulku $P[i, j]$ kam si pro každou buňku zapíšeme jak vznikla její hodnota - tj. jestli z $T[i + 1, j]$, $T[i + 1, j + 1]$ nebo z $T[i, j + 1]$. Pak můžeme začít v buňce $(1, 1)$ a následováním $P[i, j]$ postupně projdeme celou větev výpočtu, která vede na optimální hodnotu $T[1, 1]$ – tj. editační operace, které vedou na optimální editační vzdálenost.

   Tohle je velmi užitečný a obecný trik u dynamického programování - pořídit si pro každý stav informaci odkud do něj vede optimální cesta. Poté můžeme zpětným průchodem vypsat celou optimální cestu od začátku do konce.

Poslední příklad jsme na cvičení nestihli.