Řešení úlohy č. 1
Darwinovy pěnkavy

Pro vyřešení úlohy bylo potřeba se věnovat třem dílčím částem. Jak načíst vstup, jak spočítat, že 2 pěnkavy jsou stejného druhu, a jak zjistit, které trojice jsou v řešení. Načtení vstupu je implementační záležitost, a tak se jí nebudeme věnovat. Pokud víme, které dvojice pěnkav jsou stejného druhu, tak zjistit, které trojice pěnkav jsou v řešení, lze udělat naivně vyzkoušením všech možných trojic pěnkav, což lze stihnout v čase $O(n^3)$. Zbytek textu je věnován efektivnímu zjištění, zda jsou 2 pěnkavy stejného druhu.

Editační vzdálenost

Abychom zjistili, zda jsou 2 pěnkavy stejného druhu, musíme být schopni převést DNA jedné pěnkavy na DNA druhé pěnkavy pomocí maximálně K operací přidání, odebrání a přepsání symbolu. Metrika, kolik takových operací je potřeba, se nazývá editační vzdálenost, konkrétně Levenštejnova vzdálenost (Levenshtein distance).

Když obdržíme 2 řetězce $A = a_1, a_2, \ldots, a_{x-1}, a_x, B = b_1, b_2, \ldots, b_{y-1}, b_y$ a ptáme se jaká je jejich editační vzdálenost $\delta(A, B)$, odpověď hnedka nevíme. Ale víme, že je to minimum ze 3 možností.

1. Přidání znaku: Do A přidáme na konec znak b_y, to lze chápat jako odebrání posledního b_y z B. Tedy editační vzdálenost je $1 + \delta(A, b_1, b_2, \ldots, b_{y-1})$.

2. Odebrání znaku: Z A odebereme poslední znak a_x, tedy editační vzdálenost je $1 + \delta(a_1, a_2, \ldots, a_{x-1}, B)$.

3. Přepsání znaku: Spočítáme vzdálenost řetězců A, B zkrácených o poslední symbol $= \delta(a_1, a_2, \ldots, a_{x-1}, b_1, b_2, \ldots, b_{y-1})$. a pokud se poslední dva znaky liší $a_x \neq b_y$ provedli jsme operaci přepsání znaku a k výsledku přičteme 1.

Musíme ještě vyřešit případ, kdy jeden z řetězců je prázdný, k tomu nám stačí operace `insert` nebo `delete`, a výsledek je tedy počet symbolů v neprázdném řetězci.

Editační vzdálenost s dynamickým programováním

Pokud bychom naivně implementovali tento předpis pomocí rekurzivní funkce, bude naše řešení velmi pomalé. V každém volání, které nedokážeme vyřešit triviálně, uděláme další 3 volání té samé funkce, v každém zmenšíme alespoň jeden řetězec o 1 znak. Tedy celkem je provedeno alespoň $3^{\min(x,y)}$ rekurzivních volání. Nicméně spousta těchto volání je se stejnými argumenty, a algoritmus tak odpoví vždy stejně. Řešením je si před vrácením odpovědi odpověď zapamatovat (memoizovat), a pokud příště přijde dotaz se stejnými argumenty, odpovědět zapamatovanou hodnotu. Této technice se říká dynamické programování.

Editační vzdálenost v závislosti na K

Odkládat zjištění, zda je editační vzdálenost 2 DNA řetězců maximálně K, až po spočítání editační vzdálenosti 2 řetězců, je neefektivní. Znamená to, že se musí provést zjištění editační vzdálenosti každého prefixu prvního řetězce s každým prefixem druhého řetězce. Přičemž často je výsledek jasný již z porovnání délek řetězců. Pokud `length(A) - length(B)` $> K$, tak je třeba více než k operací jenom na srovnání délek. Tedy do výpočtu $\delta(A, B)$ přidáme podmínku, která v případě, že je rozdíl délky řetězců $> K$, tak odpoví hodnotou větší než K, která znamená, že pěnkavy nejsou stejného druhu.

Konečnost

V každém rekurzivním kroku počítání editační vzdálenosti je zmenšen alespoň jeden z porovnávaných řetězců. Řetězce mají konečnou délku. V obou úpravách základního algoritmu

jsou porovnávány jenom některé podřetězce, žádný není přidán navíc. Tedy jestli je první algoritmus konečný, tak jsou konečné i odvozené verze.

Správnost

Naivní verze výpočtu editační vzdálenosti, která vyzkouší všechny možnosti přepsání řetězce A na řetězec B, je korektní, protože vyzkouší všechny možnosti. Počítání editační vzdálenosti pomocí dynamického programování je korektní, protože na každý dotaz na editační vzdálenost řetězců odpoví stejně jako by odpověděla naivní verze. Třetí verze výpočtu editační vzdálenosti není korektní, protože pokud je výsledek větší než K, odpoví nějakou hodnotou větší než K, která nemusí být výsledkem. Ale pokud je editační vzdálenost řetězců menší nebo rovno K, tak algoritmus odpoví přesně toto číslo, jelikož posloupnost rekurzivních volání vedoucích k optimu (odpovídající optimálním operacím) je stejná - nebyla narušena přidanou podmínkou. Z předchozích 2 vět platí tvrzení $\forall A, B : R_2(A, B) \leq K \Leftrightarrow R_3(A, B, K) \leq K$, kde $R_2(A, B)$ je volání algoritmu s dynamickým programováním a $R_3(A, B, K)$ je volání algoritmu závislého na K. Tedy výpočet zda jsou 2 pěnkavy stejného druhu pomocí algoritmu s přidanou podmínkou na vzdálenost řetězců je i tak korektní.

Časová složitost

Naivní algoritmus v každém rekurzivním kroku provede 3 volání sám sebe s tím, že v nejhorším případě zmenší délku pouze jednoho argumentu o 1. Složitost algoritmu je tedy $O(3^{x+y})$.

Algoritmus s dynamickým programováním provede pouze $O(xy)$ různých dotazů, v každém stráví $O(1)$ času. Dohromady je tedy časová složitost $O(xy)$.

Přidáním podmínky v posledním algoritmu docílíme toho, že pro řetězec A délky x je třeba uvažovat jen řetězce B délek z intervalu $[x - K, x + K]$, tedy $2K + 1 \in O(K)$ možných délek, jinak je hned vyřešen pomocí přidané podmínky. Než se algoritmus dostane k triviálnímu případu, vyzkouší se x podřetězců řetězce A s podřetězci B, kterých je $O(K)$ pro fixní podřetězec A. Celkově se tedy provede $O(xK)$ rekurzivních volání.

Celková časová složitost

Pěnkav je celkem N, takže je potřeba porovnat $O(N^2)$ pěnkav. Délku každého DNA je možné omezit délkou nejdelšího DNA L_{MAX}. Výpočet všech editačních vzdáleností lze tak stihnout v čase $O(N^2 \cdot K \cdot L_{MAX})$. Vyzkoušet všechny trojice pěnkav jestli jsou ve výsledku zabere $O(N^3)$ času, výsledná časová složitost je tak $O(N^2 \cdot K \cdot L_{MAX} + N^3)$.

Poznámky k implementaci editační vzdálenosti

Ukládané výsledky editační vzdálenosti lze ukládat v poli o velikosti xy, do kterého se indexuje pomocí délek momentálních podřetězců. Není potřeba používat rekurzi, pole lze vyplňovat iterativně od jednoduchých případů až k výsledku.

Potřebná paměť se dá ušetřit tak, že se udržují uložené jen 2 řádky - ten z kterého čteme, a ten do kterého ukládáme nové hodnoty. To díky tomu, že poddotazy jsou vždy pouze na řetězce maximálně o 1 znak kratší.

S optimalizací dle K je vhodné nepostupovat v paměti s uloženými výsledky po řádcích nebo sloupcích, ale po diagonálách. Pokud zjistíme, že jsou všechny hodnoty na jedné diagonále větší než K, lze výpočet ukončit s výsledkem, že pěnkavy nejsou stejného druhu.