12. cvičení

Úloha 1 (Sufixové a LCP pole v praxi)
Společně si ukážeme jak sestavit obě pole na řetězci `CAGTAGCTGTA`.
Zkuste si to potom sami na řetězci `TATGTCAGTATCTC`.

Řešení

i	SA[i]	LCP[i]	Suffix	RA[SA[i]]	RA[SA[i]+k]	tempRA[SA[i]]
0	11	-	$	0	0	0
1	10	-	A$	1	0	1
2	4	-	AGCTGTA$	2	7	2
3	1	-	AGTAGCTGTA$	3	6	3
4	0	-	CAGTAGCTGTA$	4	2	4
5	6	-	CTGTA$	5	1	5
6	5	-	GCTGTA$	6	9	6
7	8	-	GTA$	7	0	7
8	2	-	GTAGCTGTA$	8	5	8
9	9	-	TA$	9	0	9
10	3	-	TAGCTGTA$	10	11	10
11	7	-	TGTA$	11	0	11

i	SA[i]	LCP[i]	Sorted Suffix T[SA[i]:]	Phi[i]	PLCP[i]	Positional Suffix T[i]
0	11	0	$	1	0	CAGTAGCTGTA$
1	10	0	A$	4	2	AGTAGCTGTA$
2	4	1	AGCTGTA$	8	3	GTAGCTGTA$
3	1	2	AGTAGCTGTA$	9	2	TAGCTGTA$
4	0	0	CAGTAGCTGTA$	10	1	AGCTGTA$
5	6	1	CTGTA$	6	0	GCTGTA$
6	5	0	GCTGTA$	0	1	CTGTA$
7	8	1	GTA$	3	1	TGTA$
8	2	3	GTAGCTGTA$	5	1	GTA$
9	9	0	TA$	2	0	TA$
10	3	2	TAGCTGTA$	11	0	A$
11	7	1	TGTA$	-1	0	$

i	SA[i]	LCP[i]	Suffix	RA[SA[i]]	RA[SA[i]+k]	tempRA[SA[i]]
0	12	-	$	0	0	0
1	6	-	AGTATC$	1	10	1
2	9	-	ATC$	2	0	2
3	1	-	ATGTCAGTATC$	3	5	3
4	11	-	C$	4	0	4
5	5	-	CAGTATC$	5	2	5
6	7	-	GTATC$	6	4	6
7	3	-	GTCAGTATC$	7	6	7
8	8	-	TATC$	8	0	8
9	0	-	TATGTCAGTATC$	9	11	9
10	10	-	TC$	10	0	10
11	4	-	TCAGTATC$	11	8	11
12	2	-	TGTCAGTATC$	12	1	12

i	SA[i]	LCP[i]	Sorted Suffix T[SA[i]:]	Phi[i]	PLCP[i]	Positional Suffix T[i]
0	12	0	$	8	3	TATGTCAGTATC$
1	6	0	AGTATC$	9	2	ATGTCAGTATC$
2	9	1	ATC$	4	1	TGTCAGTATC$
3	1	2	ATGTCAGTATC$	7	2	GTCAGTATC$
4	11	0	C$	10	2	TCAGTATC$
5	5	1	CAGTATC$	11	1	CAGTATC$
6	7	0	GTATC$	12	0	AGTATC$
7	3	2	GTCAGTATC$	5	0	GTATC$
8	8	0	TATC$	3	0	TATC$
9	0	3	TATGTCAGTATC$	6	1	ATC$
10	10	1	TC$	0	1	TC$
11	4	2	TCAGTATC$	1	0	C$
12	2	1	TGTCAGTATC$	-1	0	$

Úloha 2 (Skoro advent of code)
Ze sopky se vám podařilo zachránit hromadu slonů[1]. Díky vašim expertním znalostem různých sloních druhů jste zpozorovali, že ve skutečnosti máte slony dvou různých druhů, a protože oba druhy mají jiné preference potravy, chcete je od sebe rozdělit, ať se na vás některý ze slonů nenaštve. Bohužel je od sebe nejste schopni odlišit přímo, ale víte, že se liší svou DNA. Navíc jako správní matfyzáci s sebou máte přístroj na sekvenování DNA, který zároveň automaticky vygeneruje i sufixové a LCP pole daného řetězce DNA.
Každému slonovi tedy (s jejich souhlasem) vysekvenujete relevantní kousek DNA, který si můžeme představit jako řetězec. Po důkladném prostudování příručky *Jak se liší sloni* jste zjistili, že hlavní rozdíl jejich DNA je v délce nejdelší opakující se podposloupnosti bází DNA. Konkrétně, pokud je délka takového řetězce lichá, jedná se o slona indického, zatímco slon africký má nejdelší takovou podposloupnost sudé délky. Pro každého slona určete, zda se jedná o slona afrického či slona indického.
Pokud vás nezajímají příběhy: nalezněte v řetězci, když máte jeho sufixové i LCP pole, délku nejdelšího opakujícího se podřetězce.

Řešení
Prostě najdeme největší hodnotu v LCP poli, a ta odpovídá nejdelšímu opakujícímu se podřetězci.

Úloha 3 (Sloni jsou nějak zvědaví)
Když se sloni dozvěděli, co všechno váš sekvencer umí, zajímala by je ještě další věc: přečetli si, že podřetězec P délky n odpovídá zvýšené inteligenci, a tak by je zajímalo, jestli mají takové predispozice. Když máte relevantní část jejich genomu G délky m, nejprve si připomeňte triviální algoritmus hledání jehly v seně v čase $\mathcal{O}(n \log m)$ (můžete předpokládat, že sufixové pole máte už sestavené).
Dále si představte, že pro libovolné dva řetězce x, y umíte spočítat $\mathrm{LCP}(x, y)$ v konstantním čase. Upravte triviální algoritmus tak, aby trval jenom $\mathcal{O}(\log m)$.

Řešení
Triviální: binárně vyhledáváme nad sufixovým polem a porovnáváme řetízky, abychom zjistili, kterým směrem se vydat.
Lepší: vždycky se zeptáme na LCP(jehla, prostřední sufix sena), a pak se podíváme na další znak, a podle toho jdeme doleva nebo doprava.

Bonusové úlohy

Úloha 4 (Advent of code intensifies)
Sloni jsou ale realisté, takže si jsou vědomi toho, že obecné LCP nejde spočítat v konstantním čase. Na druhou stranu ale objevili hromadu dalších podřetězců, které údajně odpovídají všem možným vlastnostem, takže by byli moc rádi, kdyby algoritmus byl co nejrychlejší.
Co ale kdybychom uměli v konstantním čase spočítat LCP libovolných dvou *sufixů* v genomu (seně)?[2] Nalezněte v tom případě algoritmus na hledání jehly délky n v seně délky m běžící v čase $\mathcal{O}(n + \log m)$.

[1] Viz příběh advent of code 2022, dny 16-18: `https://adventofcode.com/2022/day/16`.
[2] Tento předpoklad je silnější než jenom mít LCP pole, protože umíme počítat i LCP lexikograficky nesousedních sufixů.

Hint: *V sufixovém poli máme všechny sufixy G lexikograficky seřazené, takže v něm můžeme binárně vyhledávat a zajímá nás takový sufix, jehož prefixem je právě P. Jenom je potřeba při porovnání řetězců porovnávat jenom ty správné části, aby nás všechna porovnání dohromady stále $\mathcal{O}(n)$. Cílem je tedy využívat informace o společných prefixech mezi prvky sufixového pole k udržování informací o společném prefixu P s relevantními řetězci v S.*

Řešení
Během binárního vyhledávaní máme vždy minimální i_l a maximální i_r index v S, které nám určují zmenšující se oblast, v níž ještě vyhledáváme, a střední index $i_m = \lfloor \frac{i_l + i_r}{2} \rfloor$, podle kterého dělíme. Zároveň si udržujeme délku společného prefixu P s hraničními sufixy $d_l = \mathrm{LCP}(P, S[i_l]), d_r = \mathrm{LCP}(P, S[i_r])$, tu na začátku spočítáme porovnáním znak po znaku.
Během každého kroku binárního vyhledávání vybereme hranici s delším společným prefixem s P podle d_l vs d_r, zjistíme LCP této hranice se středem $S[i_m]$. Teď z výsledku buď umíme určit $\mathrm{LCP}(P, S[i_m])$: BÚNO $d_l \geq d_r$, pak pokud výsledné $\mathrm{LCP}(S[i_l], S[i_m]) > d_l$, pokračujeme v pravé části. Pokud je výsledné LCP $\mathrm{LCP}(S[i_l], S[i_m]) < d_l$, pokračujeme v pravé části. Pokud došlo k rovnosti, pokračujeme porovnáváním dalšího znaku.

Úloha 5 (Vlastně můžeme trošku zeslabit předpoklady)
Víme, že v lineárním čase lze k S dopočítat pole LCP (Kasaiovým algoritmem). Ukažte, že potom je v lineárním čase možné dopočítat všechny hodnoty, které algoritmus v předchozí úloze potřebuje.

Řešení
Strom binárního vyhledávání je fixní a má lineární velikost, takže můžeme počítat hodnoty odpovídající jednotlivým stavům binárního vyhledávání bottom-up: listy máme okamžitě z LCP pole, vnitřní vrcholy pak jsou minimum jejich synů.