65. ročník Matematické olympiády – 2015/2016

Úlohy krajského kola kategorie P

Krajské kolo 65. ročníku MO kategorie P se koná v úterý 19. 1. 2016 v dopoledních hodinách. Na řešení úloh máte 4 hodiny čistého času. V krajském kole MO-P se neřeší žádná praktická úloha, pro zajištění rovných podmínek řešitelů ve všech krajích je použití počítačů při soutěži zakázáno. Zakázány jsou rovněž jakékoliv další pomůcky kromě psacích potřeb (např. knihy, výpisy programů, kalkulačky, mobilní telefony).

Řešení každé úlohy vypracujte na samostatný list papíru.

Řešení každé úlohy musí obsahovat:

- **Popis řešení,** to znamená slovní popis principu zvoleného algoritmu, *argumenty zdůvodňující jeho správnost* (případně důkaz správnosti algoritmu), diskusi o efektivitě vašeho řešení (časová a paměťová složitost). Slovní popis řešení musí být jasný a srozumitelný i bez nahlédnutí do samotného zápisu algoritmu (do programu). Není vhodné odkazovat se na vaše řešení úloh domácího kola, opravovatelé je nemusí mít k dispozici; na autorská řešení domácího kola se odkazovat můžete.
- **Zápis algoritmu.** V úlohách **P-II-1**, **P-II-2** a **P-II-3** je třeba uvést zápis algoritmu v nějakém dostatečně srozumitelném pseudokódu (případně v programovacím jazyce Pascal nebo C/C++). Nemusíte detailně popisovat jednoduché operace jako vstupy, výstupy, implementaci jednoduchých matematických vztahů, vyhledávání v poli, třídění apod. V řešení úlohy **P-II-4** využijte sufixové stromy takovým způsobem, jak je ukázáno ve studijním textu; nemusíte se starat o jejich implementaci.

Za každou úlohu můžete získat maximálně 10 bodů. Hodnotí se nejen správnost řešení, ale také kvalita jeho popisu a efektivita zvoleného algoritmu. Algoritmy posuzujeme podle jejich časové složitosti, tzn. závislosti doby výpočtu na velikosti vstupních dat. Záleží přitom pouze na řádové rychlosti růstu této funkce. V zadání každé úlohy najdete přibližné limity na velikost vstupních dat. Efektivním vyřešením úlohy rozumíme to, že váš program spuštěný s takovými daty na současném běžném počítači dokončí výpočet během několika sekund.

Vzorová řešení úloh naleznete krátce po soutěži na webových stránkách olympiády *http://mo.mff.cuni.cz/*. Na stejném místě bude zveřejněn i seznam úspěšných řešitelů postupujících do ústředního kola a také popis prostředí, v němž se budou v ústředním kole řešit praktické úlohy.

P-II-1 Hotel 2

Připomeňme si, že Aliho hotel má p poschodí (očíslovaných od 1 do p) a na každém z nich je n jednolůžkových pokojů.

Jednoho dne byl celý hotel prázdný. Najednou se na recepci objevilo h hostů, kteří se chtěli ubytovat. O každém z nich víme, jak je vysoký, a tedy ve kterém poschodí nejvýše ho můžeme ubytovat. Kromě toho lze od pohledu poznat, kolik peněz který člověk v hotelu utratí, tzn. jaký z něho bude mít Ali zisk. No a o ten samozřejmě Alimu jde.

Soutěžní úloha

Znáte čísla p a n popisující hotel. Ubytovat se chce h hostů. Hosta číslo i můžeme ubytovat pouze na poschodích 1 až v_i a pokud ho ubytujeme, vyděláme na jeho pobytu v hotelu q_i korun. Určete, které hosty ubytujeme a které pošleme pryč, abychom tak maximalizovali Aliho zisk.

Důležitou součástí řešení je *důkaz správnosti* vámi navrženého algoritmu.

Formát vstupu a výstupu

Na prvním řádku vstupu jsou zadána čísla p, n, h. Na druhém řádku jsou čísla $v_1, \ldots, v_h$. Na třetím řádku jsou čísla $q_1, \ldots, q_h$.

Na první řádek výstupu napište, kolik nejvýše korun může Ali vydělat ubytováním těchto hostů. Na druhý řádek vypište h mezerami oddělených celých čísel – jsou to čísla poschodí, kde ubytujeme jednotlivé hosty (v tom pořadí, v jakém jsou na vstupu), nebo -1, jestliže příslušného hosta ubytovat nechceme. Pokud existuje více řešení, vypište jedno libovolné z nich.

Omezení a hodnocení

Plných 10 bodů získáte za správné řešení, které efektivně vyřeší libovolný vstup s milióny hotelových pokojů a milióny hostů.

Až 8 bodů dostanete za správné řešení, které efektivně vyřeší libovolný vstup s $np \leq 5\,000$ a $h \leq 5\,000$.

Až 6 bodů obdržíte za správné řešení s horší polynomiální časovou složitostí.

Nejvýše 3 body získáte za správné řešení s exponenciální nebo ještě horší časovou složitostí.

Příklad

Vstup:	*Výstup:*
`5 2 5`	`14`
`1 1 2 2 2`	`1 -1 1 2 2`
`2 1 4 5 3`	

Hotel má $p = 5$ poschodí a na každém $n = 2$ pokoje. Přišlo $h = 5$ hostů a nejsou moc vysocí: první dva mohou bydlet jen v prvním poschodí, další tři v prvním nebo ve druhém poschodí.

Příklad výstupu ukazuje optimální řešení. Ubytováním hostů vyděláme $2 + 4 + 5 + 3 = 14$ korun.

P-II-2 Lyžování

Dan se vypravil lyžovat do jednoho malého lyžařského střediska. Toto středisko se skládá z n lokalit očíslovaných od 0 do $n-1$ a je zde také jedna sedačková lanovka. Spodní stanice lanovky je umístěna v lokalitě 0, horní stanice je v lokalitě 1. Všechny lokality mají navzájem různé nadmořské výšky, tyto výšky ale neznáte. V některých lokalitách jsou bufety, kde si Dan může koupit čaj s rumem.

V lyžařském středisku je z sjezdovek. Každá sjezdovka vede směrem dolů z jedné lokality do druhé. Občas se mohou některé sjezdovky křížit, ale všechna taková křížení jsou řešena mimoúrovňově (pomocí tunelů). Přejet z jedné sjezdovky na druhou je tedy možné jedině v lokalitě, kde první sjezdovka končí a druhá začíná.

Soutěžní úloha

Úkol A: *(3 body)* Dan chce v lokalitě 0 nasednout na lanovku, nechat se vyvézt na lokalitu 1 a odtud sjet nějakou posloupností sjezdovek zpět do lokality 0. Cestou chce navštívit právě jeden bufet. (Může ovšem projet okolo jiných bufetů, v nichž se nezastaví.) Napište program, který určí, kolik bufetů má Dan na výběr.

Úkol B: *(7 bodů)* Dan chce podniknout stejnou jízdu jako v úkolu A, tentokrát se ale chce cestou postupně zastavit v co nejvíce bufetech. Napište program, který určí, kolik nejvýše bufetů může Dan při jedné cestě navštívit.

Omezení a hodnocení

Plných 10 bodů získáte za správné řešení, které v obou úkolech efektivně vyřeší libovolný vstup s $n \leq 100\,000$ a $z \leq 250\,000$.

Za libovolné polynomiální řešení dostanete až 2 body v úkolu A a až 5 bodů v úkolu B.

Za libovolné správné řešení bez ohledu na jeho efektivitu můžete získat 1 bod v úkolu A a 3 body v úkolu B.

Formát vstupu a výstupu

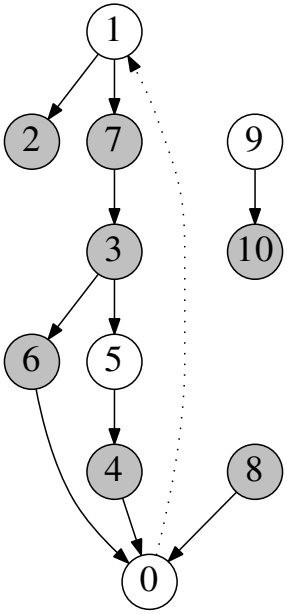
Na prvním řádku vstupu jsou zadána čísla n, z a b: počet lokalit, sjezdovek a bufetů. Na druhém řádku vstupu jsou čísla $\ell_0, \ldots, \ell_{b-1}$: čísla lokalit, kde jsou bufety. Všechna tato čísla jsou z rozsahu hodnot od 2 do $n-1$ a jsou navzájem různá. Zbytek vstupu tvoří z řádků, na každém z nich je popis jedné sjezdovky ve tvaru *odkud kam.* Všechny dvojice *odkud kam* jsou navzájem různé. Sjezdovky zadané na vstupu odpovídají výše uvedenému popisu lyžařského střediska.

Na výstup vypište jedno celé číslo – řešení příslušného úkolu A nebo B. V následujícím příkladu uvádíme na výstupu řešení obou úkolů.

Příklad

Vstup:	*Výstup:*
11 10 7 2 3 4 6 7 8 10 1 2 1 7 3 5 3 6 4 0 5 4 6 0 7 3 8 0 9 10	4 3

Když se Dan chce zastavit v jednom bufetu, má čtyři možnosti: bufet v lokalitě 3, 4, 6 nebo 7. Při jedné cestě může Dan navštívit nejvýše tři bufety: buď $(7, 3, 4)$ *nebo* $(7, 3, 6)$.



P-II-3 Míchání karet

Kleofáš postavil stroj na míchání karet. Stroj má nahoře otvor, do kterého Kleofáš vloží balíček karet. Potom Kleofáš zatočí klikou, stroj karty nějak promíchá a vypadne z něj zamíchaný balíček. Stroj míchá karty pokaždé stejně: pro každou pozici v balíčku je jednoznačně určeno, kam se po zamíchání dostane karta, která byla před mícháním na této pozici.

Kleofáš si chce zahrát poker se svým kamarádem Leonardem. Budou používat balíček n karet, z nichž právě čtyři jsou esa. Aby Leonard nepodezíral Kleofáše z podvádění, dohodli se, že před začátkem hry Kleofáš několikrát po sobě promíchá balíček karet na svém stroji. Kleofáš ale přesně ví, jakým způsobem stroj míchá karty. A nejen to, ví také, kde jsou v balíčku před prvním mícháním esa. Po zamíchání balíčku dostane Kleofáš vrchních pět karet. Nyní by ho zajímalo, kolik nejvýše es může být po skončení míchání mezi pěti vrchními kartami, když vhodně zvolí počet míchání. Kromě toho by chtěl vědět, jaký je nejmenší počet míchání, po kterém bude mezi vrchními pěti kartami tento maximální možný počet es.

Soutěžní úloha

Míchání karet Kleofášovým strojem můžeme formálně popsat pomocí čísla n a permutace čísel 1 až n, tedy takové posloupnosti $a_1, a_2, \ldots, a_n$ čísel od 1 do n, v níž se každé z čísel 1 až n vyskytuje právě jednou. Pro každé x od 1 do n potom platí, že karta, která byla před zamícháním x-tá shora, bude po zamíchání a_x-tá shora.

Na vstupu dostanete číslo n a čísla $a_1, a_2, \ldots, a_n$ popisující Kleofášův stroj. Dále dostanete čtyři čísla r_1, r_2, r_3, r_4 – pozice čtyř es v balíčku. Pozice v balíčku jsou číslovány shora dolů, tzn. pozice 1 znamená vrchní kartu balíčku a pozice n označuje spodní kartu balíčku.

Zjistěte, kolik nejvýše es může být po několika (možná i nula nebo jedna) mícháních mezi vrchními pěti kartami a po kolika mícháních bude mezi vrchními pěti kartami tento počet es poprvé.

Ve svém řešení můžete předpokládat, že celočíselné proměnné mají dostatečný rozsah na uložení správného výstupu.

Omezení a hodnocení

Za jakékoliv správné řešení bez ohledu na efektivitu dostanete až 3 body (v závislosti na kvalitě popisu).

Až 5 bodů můžete získat za správné řešení, které dokáže na běžném počítači vyřešit za méně než sekundu libovolný vstup s $n \leq 20$.

Až 7 bodů dostanete za správné řešení, které vyřeší za méně než sekundu libovolný vstup s $n \leq 100$.

Řešení za plných 10 bodů musí správně vyřešit za méně než sekundu libovolný vstup s $n \leq 1\,000\,000$.

Příklady

Vstup:	*Výstup:*
10 3 4 2 10 7 6 9 1 8 5 7 8 1 2	4 3

Pro potřeby tohoto vstupu si esa označme písmeny A, B, C, D a zbývající karty shora dolů čísly 1 *až* 6. *Před mícháním jsou tedy karty seřazeny (shora dolů) v tomto pořadí:* `A B 1 2 3 4 C D 5 6`

Po prvním míchání jsou karty seřazeny takto: `D 1 A B 6 4 3 5 C 2`

Po druhém míchání jsou karty seřazeny takto: `5 A D 1 2 4 6 C 3 B`

Po třetím míchání jsou karty seřazeny takto: `C D 5 A B 4 2 3 6 1`

Více es už mezi vrchními pěti kartami nemůže být. Kleofáš tedy dokáže získat všechna čtyři esa, a to nejdříve po 3 mícháních.

Vstup:	*Výstup:*
10 2 3 4 5 6 7 8 9 1 10 1 2 3 10	3 0

Vidíme, že eso na spodku balíčku za z tohoto svého místa nikdy nepohne, proto mezi vrchními pěti kartami mohou být nejvýše tři esa. Tento stav poprvé nastává ještě před prvním mícháním.

P-II-4 Sufixové stromy

K této úloze se vztahuje studijní text uvedený na následujících stranách. Studijní text je identický se studijním textem z domácího kola.

Úkol A: *(5 bodů)* Na vstupu jsou zadány řetězce S a T tvořené malými písmeny anglické abecedy. Napište program s optimální časovou složitostí, který najde jeden jejich nejdelší společný podřetězec. Jinými slovy, hledáme nejdelší řetězec R, který se nachází jako souvislý podřetězec jak v řetězci S, tak i v řetězci T. Například pro $S =$ `kalerab` a $T =$ `prales` je jediným správným řešením řetězec `ale`.

Úkol B: *(5 bodů)* Řekneme, že řetězec A má periodu délky p, jestliže pro všechna relevantní i platí $A[i] = A[p + i]$. Například řetězec $A =$ `mamam` má periodu délky 2, neboť platí $A[0] = A[2]$, $A[1] = A[3]$, $A[2] = A[4]$. Další příklady: řetězec `eeee` má periodu délky 1, `hahaha` délky 2, `kolecko` délky 5, `banan` délky 5.

V proměnné `strom` je uložen sufixový strom nějakého neznámého řetězce. Slovně popište algoritmus, který přímo z tohoto stromu vypočítá délku nejkratší periody dotyčného řetězce.

Studijní text

V tomto studijním textu se seznámíme s jednou užitečnou datovou strukturou pro práci se znakovými řetězci: *sufixovým stromem*. Dozvíte se, jak tento strom *vypadá*. Nedozvíte se, jak takový strom *efektivně sestrojit* – ale to při řešení soutěžních úloh nebudete potřebovat. Úplně vám bude stačit, když dokážete tento strom *použít jako nástroj* při návrhu nových algoritmů.

Dříve, než se dostaneme k samotným sufixovým stromům, zavedeme si některé užitečné pojmy.

Abeceda

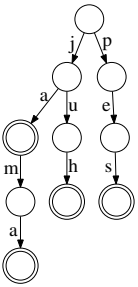
Vstupem všech soutěžních úloh budou znakové řetězce tvořené malými písmeny anglické abecedy. Kromě nich se nám občas bude hodit použít pracovně i některé další symboly. Budeme ale předpokládat, že všechny použité znaky mají ASCII hodnoty z rozmezí od 33 do 126. Velikost abecedy proto můžeme považovat za konstantní a nebudeme ji uvažovat při odhadech časové složitosti.

Písmenkový strom

Písmenkový strom (anglicky *trie*) je jednoduchá datová struktura, kterou můžeme použít pro uložení množiny řetězců. Je to zakořeněný strom, v němž platí:

- Každá hrana má přiřazeno jedno písmeno.
- Pro každý vrchol platí, že z něho vedoucí hrany mají navzájem různá písmena.
- Některé vrcholy jsou označeny.

Každému vrcholu v písmenkovém stromu odpovídá řetězec tvořený posloupností písmen, která přečteme na hranách stromu cestou z kořene do dotyčného vrcholu. Písmenkový strom představuje množinu těch řetězců, které odpovídají označeným vrcholům.

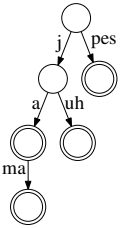


Písmenkový strom představující množinu řetězců {`ja`, `jama`, `juh`, `pes`}. Označené vrcholy jsou znázorněny dvojitým kroužkem.

Písmenkový strom reprezentující danou množinu řetězců dokážeme snadno sestrojit v čase přímo úměrném součtu jejich délek. Začneme s prázdným stromem, který je tvořen pouze neoznačeným kořenem. Postupně do stromu přidáváme jednotlivé řetězce. Přidání jednoho řetězce vypadá tak, že se z kořene stromu vydáme dolů po cestě, která je určena znaky tvořícími řetězec. Několik našich prvních kroků může vést přes již existující vrcholy, následně budeme nuceni několik nových vrcholů a hran do stromu přidat. Nakonec ještě označíme ten vrchol, v němž jsme naši cestu zakončili.

Komprimovaný písmenkový strom

Písmenkový strom často zabírá zbytečně mnoho paměti. V každém vrcholu v si totiž musíme pamatovat pro každé písmeno x abecedy, zda a kam vede z v hrana označená x. Zlepšení lze dosáhnout kompresí hran. Jednoduše vynecháme ty vrcholy, kde se nic neděje – tedy neoznačené vrcholy, v nichž se písmenkový strom nevětví. V komprimovaném písmenkovém stromu tedy platí, že každá hrana má přiřazen neprázdný *řetězec*. Následně pro každý vrchol platí, že hrany z něj vedoucí mají navzájem různá *první* písmena.



Komprimovaná verze písmenkového stromu z předcházejícího obrázku.

Komprimovanou verzi písmenkového stromu dokážeme sestrojit podobně jako tu původní, jenom implementace je o něco složitější. Kdybychom například do stromu na obrázku chtěli přidat nový řetězec `pluh`, museli bychom současnou hranu označenou `pes` rozdělit novým vrcholem v na dvě kratší: hranu označenou `p` vedoucí z kořene do v, a hranu označenou `es` vedoucí z v dále. Následně bychom z v přidali druhou hranu označenou `luh`.

Prefixy, sufixy a podřetězce

Ve více úlohách se budeme zabývat podřetězci daného řetězce. Slovem *podřetězec* budeme vždy rozumět souvislý podřetězec, tedy úsek po sobě následujících písmen v původním řetězci. Tedy například řetězec `ace` není podřetězcem řetězce `abcde`.

Podřetězce začínající na začátku řetězce nazýváme *prefixy* a podřetězce končící na jeho konci nazýváme *sufixy*. Například řetězec `abcde` má sufixy `abcde`, `bcde`, `cde`, `de` a `e`. (Někdy za sufix považujeme i prázdný řetězec – tedy sufix nulové délky.)

Všimněte si užitečné vlastnosti: ať si zvolíme jakýkoliv podřetězec daného řetězce, vždy existuje sufix, který tímto podřetězcem začíná. Například máme-li řetězec `abcde` a zvolíme si podřetězec `bc`, pak se jedná o sufix `bcde`.

K čemu je toto pozorování dobré? Říká nám, že když známe nějakou informaci o sufixech daného řetězce, můžeme z ní často snadno odvodit obdobnou informaci o libovolném jeho podřetězci. Zatímco počet podřetězců závisí na délce daného řetězce kvadraticky, počet jeho sufixů je jen lineární, takže je dokážeme zpracovat efektivněji.

Na tomto pozorování je založená hlavní datová struktura, kterou si v tomto studijním textu ukážeme.

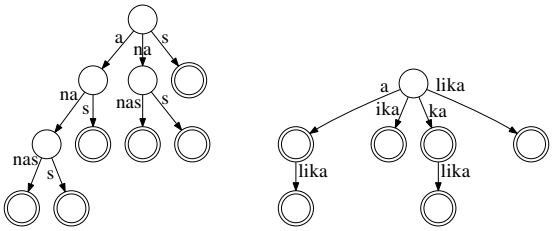
Sufixový strom

Sufixový strom odpovídající řetězci S je komprimovaný písmenkový strom obsahující množinu všech neprázdných sufixů řetězce S.

Například sufixový strom odpovídající řetězci `abcde` je vlastně komprimovaný písmenkový strom obsahující řetězce `abcde`, `bcde`, `cde`, `de` a `e`.

Kdybychom chtěli sufixový strom daného n-znakového řetězce sestrojit přímo podle naší definice, potřebovali bychom na to $\Theta(n^2)$ kroků: postupně po jednom bychom do něj vkládali všechny sufixy, jejichž součet délek je $n(n+1)/2$.

Všimněte si ale, že výsledný strom má nejvýše n listů (jeden pro každý sufix). Má tedy jenom $\mathcal{O}(n)$ vrcholů a také pouze $\mathcal{O}(n)$ hran. Zdá se proto, že bychom ho mohli sestrojit i v lepším čase než kvadratickém. Skutečně existují šikovné algoritmy, které k danému řetězci postaví jeho sufixový strom dokonce v čase $\Theta(n)$. Tyto algoritmy ovšem svou náročností přesahují rámec tohoto textu a nebudeme se jimi zde zabývat.



Vlevo sufixový strom pro řetězec `ananas`*, vpravo pro řetězec* `kalika`*.*

Sufixový strom se zarážkou

Sufixový strom pro řetězec `ananas` měl jednu pěknou vlastnost: každému sufi-

xu odpovídal jeden z listů tohoto stromu. Sufixový strom pro řetězec `kalika` tuto vlastnost neměl, jelikož třeba sufix `a` je prefixem sufixu `alika`.

Tomu však můžeme snadno pomoci: namísto řetězce `kalika` sestrojíme sufixový strom pro řetězec `kalika#` (přičemž obecně `#` představuje libovolný symbol, který se v původním řetězci nevyskytuje). V novém sufixovém stromu už skutečně každý sufix odpovídá jinému listu, neboť po přidání „zarážky“ `#` na konec řetězce už zjevně nemůže být jeden sufix prefixem jiného.

Detaily reprezentace sufixového stromu v paměti

Než se pustíme do řešení soutěžních úloh, musíme se ještě domluvit na některých technických detailech.

- Sufixový strom je objekt. Obsahuje proměnnou `retezec`, v níž je uložen znakový řetězec, jehož sufixy jsou uloženy ve stromu. Dále obsahuje proměnnou `koren`, která představuje ukazatel na kořen samotného stromu.
- Každý vrchol stromu je objekt, který obsahuje tři proměnné:
  - proměnnou `data`, do níž si můžete ukládat údaje libovolného typu (tento typ si můžete sami zvolit, jak potřebujete)
  - proměnnou `deti`, což je pole indexované písmeny (prvním písmenem řetězce přiřazeného hraně). Každý prvek tohoto pole obsahuje ukazatel na příslušnou hranu. Pokud taková hrana neexistuje, je příslušný ukazatel nulový.
  - proměnnou `konec`, v níž je uložena hodnota true nebo false podle toho, zda tu končí nějaký sufix
- Každá hrana stromu je také objekt. Obsahuje tři proměnné: číselné proměnné `od` a `do` a ukazatel na vrchol `kam`. Proměnná `kam` ukazuje na vrchol, do něhož hrana vede. Číselné proměnné říkají, že řetězec přiřazený této hraně je podřetězec původního řetězce (toho, který je uložen v proměnné `retezec` pro celý strom) tvořený znaky na pozicích `od` až `do-1` včetně.

  (Proč jsme použili proměnné `od` a `do` místo toho, abychom pro každou hranu přímo uložili její řetězec? Rozmyslete si, že kdybychom dotyčné řetězce zapisovali přímo, potřebovali bychom na uložení stromu v nejhorším možném případě kvadraticky mnoho paměti.)
- Ve svých řešeních můžete používat funkci `vytvor_strom(r)`, které předáte jako jediný parametr řetězec `r`, jehož sufixový strom chcete sestrojit. Funkce tento strom (v lineárním čase vzhledem k délce zadaného řetězce) postaví a vrátí ho jako návratovou hodnotu.

Rozšířený sufixový strom

Občas potřebujeme sufixový strom pro více než jeden řetězec. Máme například řetězce `A` a `B` a chceme sestrojit strom, který bude obsahovat sufixy řetězce `A` i sufixy řetězce `B`.

K tomu stačí šikovně využít funkci `vytvor_strom`. Na vstup jí předložíme řetězec `A#B#`, kde `#` („zarážka“) je nový znak nevyskytující se ani v `A`, ani v `B`. Ve stromu,

který takto získáme, budeme ignorovat (nebo dokonce smažeme) všechno, co se nachází pod nějakým výskytem znaku `#`.

Například máme-li řetězce `macka` a `pes`, sestrojíme sufixový strom pro řetězec `macka#pes#`. V tomto stromu bude uložen třeba sufix `es#` (odpovídající sufixu `es` řetězce `pes`), ale také sufix `cka#pes#` (odpovídající sufixu `cka` řetězce `macka`).

Někdy je navíc užitečné použít navzájem různé zarážky. Když sestrojíme sufixový strom pro řetězec `macka$pes#`, můžeme pak rozlišit, zda sufix patří prvnímu nebo druhému řetězci podle toho, na kterou zarážku dříve narazíme při jeho čtení.

Příklad 1

Úloha: Na vstupu je zadán dlouhý řetězec `T`. Poté bude přicházet mnoho dalších řetězců. O každém z nich zjistěte, zda se v `T` nachází jako podřetězec.

Řešení: Sestrojíme si sufixový strom pro `T`. Následně pro každý řetězec `S` začneme v kořeni stromu a snažíme se sestupovat dolů cestou, která odpovídá řetězci `S`. Když se nám to podaří, řetězec `S` se v `T` nachází. Když někde cestou uvázneme a nemůžeme pokračovat dále, nastal opačný případ.

Každý řetězec takto zpracujeme v čase lineárním vzhledem k jeho délce.

```
def zjisti_zda_se_nachazi(strom, slovo):
    ''' Zjistí, zda se řetězec "slovo" nachází v řetězci T,
        jehož sufixový strom je "strom". '''

    kde = strom.koren  # začneme v kořeni stromu
    i = 0              # zpracujeme i-té písmeno řetězce "slovo"
    while i < len(slovo):
        # Zkontrolujeme, zda z aktuálního vrcholu vede hrana pro správné písmeno.
        if slovo[i] not in kde.deti: return False
        hrana = kde.deti[ slovo[i] ]

        # Pokud vede, zkontrolujeme, zda je celý text hrany správný.
        delka = min( hrana.do - hrana.od, len(slovo) - i )

        text_hrana = strom.retezec[ hrana.od : hrana.od + delka ]
        text_slovo = slovo[ i : i+delka ]
        if text_hrana != text_slovo: return False

        # Když text odpovídal, posuneme se o vrchol níže.
        i += delka
        kde = hrana.kam
    return True

T = input()
strom = vytvor_strom(T)

Q = int( input() )      # Počet otázek
for q in range(Q):
    slovo = input()     # Přečteme otázku
    print( zjisti_zda_se_nachazi( strom, slovo ) )
```

Příklad 2

Úloha: Na vstupu je zadán dlouhý řetězec `T`. Poté bude přicházet mnoho dalších řetězců. O každém z nich zjistěte, *kolikrát* se v `T` nachází jako podřetězec.

Řešení: Upravíme předchozí řešení. Až sestrojíme strom, rekurzívně ho projdeme a v každém vrcholu si spočítáme, kolik sufixů pod ním končí – tedy kolik vrcholů pod ním (včetně jeho samotného) má proměnnou `konec` nastavenu na true.

Rozmyslete si, že máme-li v našem sufixovém stromu vrchol r odpovídající řetězci `R`, potom každý konec sufixu v podstromu s kořenem r odpovídá jednomu výskytu řetězce `R` v původním textu. Namísto true/false tedy na zadanou otázku odpovíme naší spočítanou hodnotou.

V následujícím výpisu programu uvádíme jen ty části, v nichž se liší od předcházejícího.

```
def spocitej_konce(kde):
    kde.data = 0
    if kde.konec:
        kde.data = 1
    for x in kde.deti:
        kde.data += spocitej_konce( kde.deti[x].kam )
    return kde.data

def kolikrat_se_nachazi(strom,slovo):
    ''' zjistí, kolikrát se řetězec "slovo" nachází v řetězci T,
        jehož sufixovy strom je "strom" '''

    # ...
        if slovo[i] not in kde.deti: return 0
    # ...
        if text_hrana != text_slovo: return 0
    # ...
    return kde.data

T = input()
strom = vytvor_strom(T)
spocitej_konce( strom.koren ) # <--- před zpracováním otázek
                              # jednou spočítáme odpovědi

Q = int( input() )      # Počet otázek
for q in range(Q):
    slovo = input()     # Přečteme otázku
    print( kolikrat_se_nachazi( strom, slovo ) )
```