Informácie a pravidlá

Pre koho je súťaž určená?

Do **kategórie B** sa smú zapojiť len tí žiaci základných a stredných škôl, ktorí ešte ani v tomto, ani v nasledujúcom školskom roku nebudú končiť strednú školu.
Do **kategórie A** sa môžu zapojiť všetci žiaci (základných aj) stredných škôl.

Odovzdávanie riešení domáceho kola

Riešitelia domáceho kola odovzdávajú riešenia sami, v elektronickej podobe, a to priamo na stránke olympiády: http://oi.sk/. Odovzdávanie riešení bude spustené niekedy v septembri.

Riešenia kategórie A je potrebné odovzdať najneskôr **15. novembra 2015**.
Riešenia kategórie B je potrebné odovzdať najneskôr **30. novembra 2015**.

Priebeh súťaže

Za každú úlohu domáceho kola sa dá získať od 0 do 10 bodov. Na základe bodov domáceho kola stanoví Slovenská komisia OI (SK OI) pre každú kategóriu bodovú hranicu potrebnú na postup do **krajského kola**. Očakávame, že táto hranica bude približne rovná **tretine maximálneho počtu bodov**.

V krajskom kole riešitelia riešia štyri teoretické úlohy, ktoré môžu tematicky nadväzovať na úlohy domáceho kola. V kategórii B súťaž týmto kolom končí.

V kategórii A je približne najlepších 30 riešiteľov krajského kola (podľa počtu bodov, bez ohľadu na kraj, v ktorom súťažili) pozvaných do **celoštátneho kola**. V celoštátnom kole účastníci prvý deň riešia teoretické a druhý deň praktické úlohy. Najlepší riešitelia sú vyhlásení za víťazov. Približne desať najlepších riešiteľov následne SK OI pozve na týždňové výberové sústredenie. Podľa jeho výsledkov SK OI vyberie družstvá pre Medzinárodnú olympiádu v informatike (IOI) a Stredoeurópsku olympiádu v informatike (CEOI).

Ako majú vyzerať riešenia úloh?

V praktických úlohách je vašou úlohou vytvoriť program, ktorý bude riešiť zadanú úlohu. Program musí byť v prvom rade korektný a funkčný, v druhom rade sa snažte aby bol čo najefektívnejší.

V kategórii B môžete použiť ľubovoľný programovací jazyk.

V kategórii A musíte riešenia praktických úloh písať v jazyku Pascal, C, alebo C++. Odovzdaný program bude automaticky otestovaný na viacerých vopred pripravených testovacích vstupoch. Podľa toho, na koľko z nich dá správnu odpoveď, vám budú pridelené body. Výsledok testovania sa dozviete krátko po odovzdaní. Ak váš program nezíska plný počet bodov, budete ho môcť vylepšiť a odovzdať znova, až do uplynutia termínu na odovzdávanie.

Presný popis, ako majú vyzerať riešenia praktických úloh (napr. realizáciu vstupu a výstupu), nájdete na webstránke, kde ich budete odovzdávať.

Ak nie je v zadaní povedané ináč, riešenia teoretických úloh musia v prvom rade obsahovať **podrobný slovný popis použitého algoritmu, zdôvodnenie jeho správnosti** a diskusiu o efektivite zvoleného riešenia (t. j. posúdenie časových a pamäťových nárokov programu). Na záver riešenia uveďte program. Ak používate v programe netriviálne algoritmy alebo dátové štruktúry (napr. rôzne súčasti STL v C++), súčasťou popisu algoritmu musí byť dostatočný popis ich implementácie.

Usporiadateľ súťaže

Olympiádu v informatike (OI) vyhlasuje *Ministerstvo školstva SR* v spolupráci so *Slovenskou informatickou spoločnosťou* (odborným garantom súťaže) a *Slovenskou komisiou Olympiády v informatike*. Súťaž organizuje *Slovenská komisia OI* a v jednotlivých krajoch ju riadia *krajské komisie OI*. Na jednotlivých školách ju zaisťujú učitelia informatiky. Celoštátne kolo OI, tlač materiálov a ich distribúciu po organizačnej stránke zabezpečuje IUVENTA v tesnej súčinnosti so Slovenskou komisiou OI.

A-I-1 Hotel

Toto je praktická úloha. Pomocou webového rozhrania odovzdajte ***funkčný, odladený program****.*

Arabský šejk Ali si postavil najvyšší hotel na svete. Hotel má p poschodí, tie sú očíslované od 1 po p. Na každom poschodí je n izieb. Všetky izby sú „jednotky“, t.j. do každej izby môže ubytovať len jedného hosťa.
Keď už však Ali hotel dostaval a začal ubytovávať hostí, zistil, že má problém s výťahmi. V každom výťahu bolo nainštalovaných $p+1$ tlačidiel (jedno pre prízemie a po jednom pre každé z poschodí). No a keďže p bolo veľké, tlačidlá pokrývali celú stenu výťahu. To by samo o sebe až tak nevadilo. Horšie však bolo, že hostia nižšieho vzrastu často nedočiahli na tlačidlo svojho poschodia.
Aby tomu Ali zabránil, vydal nariadenie: keď na recepcii ubytúvajú hosťa, musia odhadnúť, aký je vysoký, a ubytovať ho na takom poschodí, na ktoré sa ešte bude vedieť dostať.

Súťažná úloha

Na vstupe sú čísla p a n popisujúce hotel. Postupne príde h hostí, ktorí sa chcú ubytovať. Hosťa číslo i môžete ubytovať len na poschodia 1 až v_i. Hostí musíte ubytúvať v poradí, v akom sú daní na vstupe. Koľko najviac ich viete ubytovať skôr než budete niekoho musieť poslať preč?

Formát vstupu a výstupu

V prvom riadku vstupu sú čísla p, n a h. V druhom riadku sú čísla $v_1, \ldots, v_h$.

V prvom riadku výstupu vypíšte najväčšie k také, že existuje spôsob ako ubytovať prvých k hostí zo vstupu.
V druhom riadku vypíšte k medzerami oddelených celých čísel: poschodia, na ktoré jednotlivých hostí (v poradí, v akom sú uvedení na vstupe) ubytovať. (Ak existuje viac možností čo vypísať v tomto riadku, vyberte si ľubovoľnú z nich.)

Obmedzenia a hodnotenie

Je 10 sád testovacích vstupov, za každú môžete získať 1 bod.
V jednotlivých sadách celkový počet izieb v hoteli neprevýši nasledovnú hodnotu: 10, 100, 500, 1000, 50 000, 250 000, 500 000, 500 000, 1 000 000 a 1 000 000.
V sadách s nepárnym číslom je $n = 1$, teda na každom poschodí je práve jedna izba. V sadách s párnym číslom platí $2 \leq n \leq 10$.
V každom vstupe platí $0 \leq h \leq pn$ a pre každé i platí $1 \leq v_i \leq p$.

Príklad

vstup	výstup
10 1 8 3 1 9 4 7 3 4 10	6 2 1 6 4 7 3

Hotel má 10 poschodí a na každom 1 izbu. Postupne príde 8 ľudí. Vieme ubytovať prvých 6 z nich, a to napr. vyššie uvedeným spôsobom. Všimnite si, že pri príchode siedmeho hosťa sú už všetky izby, do ktorých sa tento hosť vie dostať, plné.

A-I-2 Žabka

Toto je praktická úloha. Pomocou webového rozhrania odovzdajte ***funkčný, odladený program****.*

Žabka Šandyna rada skáče po kameňoch na rybníku. V jej rybníku ich je n a sú očíslované od 1 po n. Kamene sú maličké, takže si ich predstavíme ako body. Kameň i leží na súradniciach (x_i, y_i).
Šandyna dnes preskákala z kameňa 1 na kameň n. Cestou mohla niektoré kamene (vrátane kameňov 1 aj n) navštíviť aj viackrát. Šandyna vie spraviť skok ľubovoľnej dĺžky. Skákanie ju však unavuje, a tak každý jej skok okrem prvého bol ostro kratší ako bezprostredne predchádzajúci.

Súťažná úloha

Pre dané polohy kameňov vypočítajte, koľko najviac skokov mohla Šandyna spraviť počas cesty z kameňa 1 na kameň n.

Formát vstupu a výstupu

V prvom riadku vstupu je počet kameňov n. V i-tom z nasledujúcich n riadkov sú súradnice kameňa číslo i.

Obmedzenia a hodnotenie

Je 10 sád testovacích vstupov, za každú môžete získať 1 bod.
V jednotlivých sadách je maximálna hodnota n nasledovná: 2, 3, 7, 18, 50, 100, 200, 1000, 2000, 3000.
Všetky súradnice sú z rozsahu od 0 do 10^9, vrátane. V prvých piatich sadách vstupov dokonca žiadna súradnica neprekročí 10^4.

Príklad

vstup

```
6
0 1
5 0
8 3
3 3
3 5
3 2
```

výstup

```
7
```

Jedna optimálna postupnosť 7 skokov:
$1 \to 3 \to 5 \to 1 \to 6 \to 5 \to 4 \to 6$

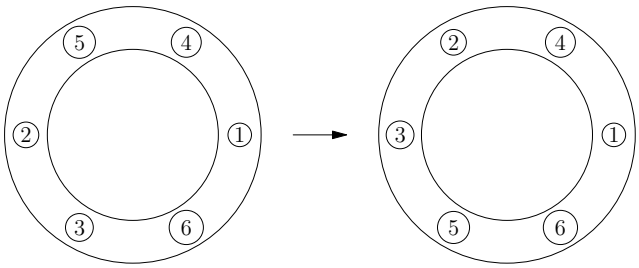
Tieto skoky majú postupne nasledovné dĺžky:
$2\sqrt{17} > \sqrt{29} > 5 > \sqrt{10} > 3 > 2 > 1$

A-I-3 Triediaca hra

Toto je teoretická úloha. Pomocou webového rozhrania odovzdajte súbor vo formáte PDF, obsahujúci riešenie, spĺňajúce požiadavky uvedené v pravidlách.

Marienka sa hrá na počítači. Dnes ju zaujala jednoduchá logická hra. Do kruhu je rozložených n kamienkov očíslovaných od 1 po n. Úlohou hráčky je kamienky preusporiadať tak, aby boli v usporiadanom poradí. (Presnejšie, keď začneme pri kamienku 1 a pôjdeme **v smere** hodinových ručičiek, musíme postupne stretnúť kamienky s číslami 2, 3, ..., n.
Hráčka môže meniť poradie kamienkov len jedným spôsobom: tak, že na nejaký klikne. Označený kamienok sa presunie po obvode kruhu o k pozícii **proti smeru** hodinových ručičiek. Relatívne poradie ostatných kamienkov sa nezmení.

Na obrázku je príklad platného ťahu pre $k = 2$. V situácii vľavo Marienka klikla na kamienok s číslom 5.



Súťažná úloha

Na vstupe sú čísla n, k, a zoznam čísel kamienkov v poradí, v akom ich stretneme idúc po obvode kruhu **v smere** hodinových ručičiek. Môžete predpokladať, že $k \in \{1, 2, 3\}$ a že $n > k + 1$.
Napíšte program, ktorý zistí, či sa z danej situácie dá hra vyhrať – teda či existuje postupnosť platných ťahov, ktorá kamienky usporiada.

Hodnotenie

Za program fungujúci pre $n \leq 10$ môžete získať až 4 body.
Za program fungujúci pre $n \leq 1000$ môžete získať až 8 bodov.
Vzorové, 10-bodové riešenie by na bežnom počítači do sekundy vyriešilo ľubovoľný vstup s $n \leq 100\,000$.
Dôležitou súčasťou ľubovoľného efektívneho riešenia je **dôkaz jeho správnosti**.
Čiastočné body môžete dostať aj za riešenia ktoré budú fungovať len pre niektoré hodnoty k.

Príklady

vstup

```
7 1
6 7 1 2 5 4 3
```

výstup

```
ano
```

Napríklad dvakrát po sebe klikne na kamienok s číslom 3 a potom raz na kamienok s číslom 4.

vstup

```
5 2
1 4 3 2 5
```

výstup

```
nie
```

vstup

```
5 3
1 4 3 2 5
```

výstup

```
ano
```

Napr. klikne na kamienok s číslom 3 a potom dvakrát po sebe na kamienok číslo 4.

A-I-4 Sufixové stromy

Toto je teoretická úloha. Pomocou webového rozhrania odovzdajte súbor vo formáte PDF, obsahujúci riešenie, spĺňajúce požiadavky uvedené v pravidlách. K tejto úlohe patrí študijný text uvedený na nasledujúcich stranách. Odporúčame najskôr prečítať ten a až potom sa vrátiť k samotným súťažným úlohám. Jednotlivé podúlohy spolu nesúvisia, môžete ich riešiť v ľubovoľnom poradí.

Súťažná úloha, podúloha A (2 body)

V premennej `strom` je sufixový strom nejakého neznámeho reťazca. Napíšte čo najefektívnejší program, ktorý zistí, koľko *navzájom rôznych* písmen tento reťazec obsahuje.

Príklad: Ak ide o reťazec `macka`, odpoveďou by malo byť číslo 4.

Súťažná úloha, podúloha B (4 body)

Na vstupe je daný reťazec S. Napíšte program, ktorý v optimálnej časovej zložitosti nájde (jeden ľubovoľný) najdlhší podreťazec T, ktorý sa v S vyskytuje aspoň dvakrát. Výskyty T v S sa môžu čiastočne prekrývať.

Príklady: Pre $S =$ `rokoko` je riešením $T =$ `oko`. Pre $S =$ `macka` je riešením $T =$ `a`. Pre $S =$ `pes` je riešením prázdny reťazec T.

Súťažná úloha, podúloha C (4 body)

Na vstupe je daný reťazec S. Napíšte program, ktorý v optimálnej časovej zložitosti spočíta, koľko má S navzájom rôznych podreťazcov.

Príklad: Pre $S =$ `rokoko` je správna odpoveď 15. V abecednom poradí sú to nasledovné podreťazce: `k`, `ko`, `kok`, `koko`, `o`, `ok`, `oko`, `okok`, `okoko`, `r`, `ro`, `rok`, `roko`, `rokok` a `rokoko`. (Niektoré z nich sa v S vyskytujú viackrát.)

Študijný text: sufixové stromy

V tomto študijnom texte sa zoznámime s užitočnou dátovou štruktúrou pre prácu s reťazcami: *sufixovým stromom*. Dozvieš sa, ako takýto strom *vyzerá*. Nedozvieš sa, ako takýto strom *efektívne vyrobiť* – ale to na riešenie súťažných úloh ani vedieť nepotrebuješ. Plne bude stačiť, ak budeš vedieť tento strom *použiť ako nástroj* pri návrhu nových algoritmov.
Ale skôr, než sa dostaneme k samotným sufixovým stromom, začneme jednoduchšími vecami.

Abeceda

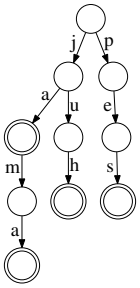
Vstupom pre všetky súťažné úlohy budú reťazce tvorené malými písmenami anglickej abecedy. Okrem nich sa nám občas oplatí interne použiť aj nejaké iné symboly. Budeme však predpokladať, že všetky použité symboly majú ASCII hodnoty od 33 po 126. Veľkosť abecedy teda budeme považovať za konštantu a nebudeme ju brať do úvahy pri odhadoch časovej zložitosti.

Písmenkový strom

Písmenkový strom (po anglicky *trie*) je jednoduchá dátová štruktúra, ktorú vieme použiť na uloženie množiny reťazcov. Funguje nasledovne: písmenkový strom je zakorenený strom, v ktorom platí:

- Každá hrana má k sebe priradené písmeno.
- Pre každý vrchol platí, že hrany vedúce z neho majú navzájom rôzne písmená.
- Niektoré vrcholy sú označené.

Každému vrcholu v písmenkovom strome vieme priradiť reťazec, ktorý mu zodpovedá: postupnosť písmen, ktoré prečítame na hranách cestou z koreňa do dotyčného vrcholu. Písmenkový strom predstavuje množinu tých reťazcov, ktoré zodpovedajú označeným vrcholom.

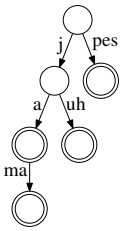


Obr. 1: Písmenkový strom predstavujúci množinu reťazcov {`ja`, `jama`, `juh`, `pes`}. Označené vrcholy sú znázornené dvojitým krúžkom.

Písmenkový strom reprezentujúci danú množinu reťazcov vieme ľahko vyrobiť v čase priamo úmernom súčtu ich dĺžok: Začneme s prázdnym stromom, tvoreným len neoznačeným koreňom. Postupne po jednom pridávame reťazce, ktoré chceme uložiť. Pridávanie reťazca vyzerá tak, že sa z koreňa dodola vyberieme cestou určenou jeho písmenami. Vo všeobecnosti prvých niekoľko krokov povedie cez už existujúce vrcholy a následne budeme nútení niekoľko nových vrcholov a hrán pridať. Vrchol, v ktorom cestu dodola skončíme, označíme.

Komprimovaný písmenkový strom

Písmenkový strom často zaberá zbytočne veľa pamäte. V každom vrchole v si totiž treba pamätať pre každé písmeno x abecedy či a kam vedie z v na x hrana. Ako to zlepšiť? Kompresiou hrán: jednoducho vynecháme tie vrcholy, kde sa nič nedeje – inými slovami, neoznačené vrcholy v ktorých sa písmenkový strom nevetví. V komprimovanom písmenkovom strome teda platí, že každá hrana má k sebe priradený neprázdny *reťazec*. A následne pre každý vrchol platí, že hrany vedúce z neho majú navzájom rôzne *prvé* písmená.



Obr. 2: Komprimovaná verzia písmenkového stromu z predchádzajúceho obrázku.

Za zmienku stojí, že aj komprimovanú verziu písmenkového stromu vieme vyrábať podobne ako pôvodnú, len implementácia je o čosi zložitejšia. Keby sme napríklad do stromu z obrázka išli pridať nový reťazec `pluh`, museli by sme súčasnú hranu označenú `pes` rozdeliť novým vrcholom v na dve kratšie: hranu označenú `p` vedúcu z koreňa do v, a hranu označenú `es` vedúcu z v ďalej. A následne by sme z v pridali druhú hranu označenú `luh`.

Prefixy, sufixy a podreťazce

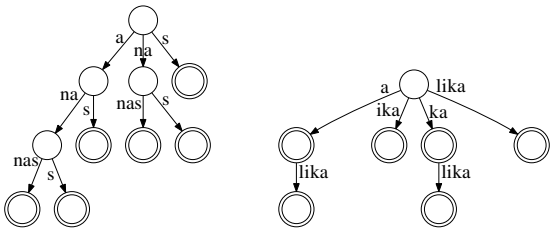
Vo viacerých úlohách sa budeme zaoberať podreťazcami daného reťazca. Tu považujeme za potrebné podotknúť, že pod slovom *podreťazec* budeme vždy rozumieť súvislý podreťazec, teda úsek po sebe nasledujúcich písmen v pôvodnom reťazci. Teda napr. reťazec `ace` nie je podreťazcom reťazca `abcde`.
Podreťazce začínajúce na začiatku reťazca voláme *prefixy* a podreťazce končiace na jeho konci voláme *sufixy*. Teda napríklad reťazec `abcde` má sufixy `abcde`, `bcde`, `cde`, `de` a `e`. (Niekedy za sufix považujeme aj prázdny reťazec – teda sufix nulovej dĺžky.)

Všimnite si užitočnú vec: nech si zvolíme akýkoľvek podreťazec daného reťazca, vždy existuje nejaký sufix, ktorý týmto podreťazcom začína. Napríklad ak máme reťazec `abcde` a zvolíme si podreťazec `bc`, tak ide o sufix `bcde`.

Načo je toto pozorovanie dobré? Hovorí nám, že keď vieme nejakú informáciu o sufixoch daného reťazca, môžeme z nej často ľahko vedieť odvodiť takú istú informáciu o ľubovoľnom jeho podreťazci. A zatiaľ čo počet podreťazcov závisí od dĺžky daného reťazca kvadraticky, počet jeho sufixov je len lineárny, takže ich vieme spracovať efektívnejšie.
A práve na tomto pozorovaní je založená hlavná dátová štruktúra, ktorú si v tomto študijnom texte predstavujeme.

Sufixový strom

Sufixový strom si vieme definovať veľmi jednoducho: sufixový strom zodpovedajúci reťazcu S je komprimovaný písmenkový strom obsahujúci množinu všetkých neprázdnych sufixov reťazca S.
Teda napríklad sufixový strom zodpovedajúci reťazcu `abcde` je vlastne komprimovaný písmenkový strom obsahujúci reťazce `abcde`, `bcde`, `cde`, `de` a `e`.



Obr. 3: Vľavo sufixový strom pre reťazec `ananas`, vpravo pre reťazec `kalika`.

Ak by sme chceli sufixový strom daného n-znakového reťazca vyrobiť priamo podľa našej definície, potrebovali by sme na to $\Theta(n^2)$ krokov: postupne po jednom by sme doň vkladali všetky sufixy, no a tie majú súčet dĺžok až $n(n+1)/2$.
Všimnite si ale, že výsledný strom má nanajvýš n listov (jeden pre každý sufix) a teda má len $O(n)$ vrcholov a tiež len $O(n)$ hrán. Teoreticky by teda mohol ísť vyrobiť aj v lepšom ako kvadratickom čase. A skutočne: Existujú šikovné algoritmy, ktoré k danému reťazcu vyrobia jeho sufixový strom dokonca v čase $\Theta(n)$. Tieto algoritmy svojou náročnosťou presahujú rámec tohto textu a nebudeme sa nimi zaoberať.

Sufixový strom so zarážkou

Sufixový strom pre reťazec `ananas` mal peknú vlastnosť: každému sufixu zodpovedal jeden z listov tohto stromu. Sufixový strom pre reťazec `kalika` túto vlastnosť nemal. Totiž napríklad sufix `a` je prefixom sufixu `alika`.
Tomu však vieme ľahko pomôcť: namiesto reťazca `kalika` si spravíme sufixový strom pre reťazec `kalika#` (pričom vo všeobecnosti `#` predstavuje ľubovoľný symbol, ktorý sa v pôvodnom reťazci nenachádza). V novom sufixovom strome už naozaj každý sufix zodpovedá inému listu – inými slovami, po pridaní „zarážky“ `#` na koniec reťazca už zjavne nemôže byť jeden sufix prefixom iného.

Detaily reprezentácie sufixového stromu v pamäti

Skôr, než sa pustíš do riešenia súťažných úloh, ešte sa potrebujeme dohodnúť na niekoľkých technických detailoch, aby sa nám vaše riešenia lepšie čítali.

- Sufixový strom je objekt. Obsahuje premennú `retazec` v ktorej je uložený reťazec ktorého sufixy sú uložené v strome. Ďalej obsahuje premennú `koren`: ukazovateľ na koreň samotného stromu.
- Každý vrchol stromu je objekt, ktorý obsahuje tri premenné:
  - premennú `data`, do ktorej si môžete ukladať údaje ľubovoľného typu (pričom tento typ si môžete vy zvoliť aký potrebujete)

  - premennú `deti`: pole, do ktorého indexujeme písmenkom (prvým písmenkom reťazca priradeného hrane). Každé políčko tohto poľa obsahuje ukazovateľ na príslušnú hranu. Ak taká hrana neexistuje, je napr. príslušný ukazovateľ nulový.
  - premennú `koniec` v ktorej je hodnota true alebo false podľa toho, či tu končí nejaký sufix

- Každá hrana stromu je tiež objekt. Obsahuje tri premenné: číselné premenné `od` a `po` a ukazovateľ na vrchol `kam`. Premenná `kam` ukazuje na vrchol do ktorého hrana vedie. Číselné premenné hovoria, že reťazec napísaný na tejto hrane je podreťazcom pôvodného reťazca (toho uloženého v premennej `retazec` pre celý strom) tvorený znakmi na pozíciách `od` až `po-1`, vrátane.

  (Prečo sme použili premenné `od` a `po` namiesto toho aby sme priamo pre každú hranu uložili reťazec ktorý jej zodpovedá? Rozmyslite si, že keby sme dotyčné reťazce zapisovali priamo, dostali by sme v najhoršom možnom prípade až strom, na ktorého uloženie je potrebnej kvadraticky veľa pamäte.)

- Vo svojom riešení môžeš použiť funkciu `vyrob_strom(r)` ktorej dáš ako jediný parameter reťazec `r`, ktorého sufixový strom chceš vyrobiť. Funkcia tento strom (v lineárnom čase od dĺžky zadaného reťazca) vyrobí a vráti ho ako návratovú hodnotu.

Rozšírený sufixový strom

Občas by sme chceli vedieť spraviť sufixový strom aj pre viac ako jeden reťazec. Presnejšie, chceli by sme napríklad zobrať reťazce `A` a `B` a vyrobiť strom, v ktorom budú aj sufixy reťazca `A` aj sufixy reťazca `B`.
Na vyriešenie tejto úlohy stačí šikovne využiť funkciu `vyrob_strom`. Tej na vstup podstrčíme reťazec `A#B#`, kde `#` („zarážka“) je nový znak nevyskytujúci sa ani v `A` ani v `B`. V strome, ktorý takto dostaneme, už len jednoducho odignorujeme (alebo dokonca zmažeme) všetko čo sa nachádza pod nejakým výskytom znaku `#`.
Napr. ak by sme mali reťazce `macka` a `pes`, zostrojíme sufixový strom pre reťazec `macka#pes#`. V tomto strome bude napr. uložený sufix `es#` (zodpovedajúci sufixu `es` reťazca `pes`) ale aj sufix `cka#pes#` (zodpovedajúci sufixu `cka` reťazca `macka`).
Občas je navyše užitočné použiť navzájom rôzne zarážky: napr. ak vyrobíme sufixový strom pre reťazec `macka$pes#` tak vieme spoznať, či sufix patrí prvému alebo druhému reťazcu podľa toho, na ktorú zarážku skôr narazíme pri jeho čítaní.

Príklad 1

Úloha: Na vstupe je dlhý reťazec `T`. Potom bude prichádzať veľa ďalších reťazcov. O každom z nich zistite, či sa v `T` nachádza ako podreťazec.

Riešenie: Zostrojíme si sufixový strom pre `T`. Následne pre každý reťazec `S` začneme v koreni stromu a snažíme sa ísť dodola cestou, ktorá zodpovedá `S`. Ak sa nám to podarí, vieme, že `S` sa v `T` nachádza, ak sa niekde zasekneme, vieme, že nastal opačný prípad.
Každý reťazec takto spracujeme v čase lineárnom od jeho dĺžky.

```
def zisti_ci_sa_nachadza(strom,slovo):
    # zisti, ci sa retazec "slovo" nachadza v retazci T, ktoreho sufixovy strom je "strom"

    kde = strom.koren  # zacneme v koreni stromu
    i = 0              # ideme spracovat i-te pismeno retazca slovo
    while i < len(slovo):
        # pozrieme, ci z aktualneho vrcholu ide hrana na spravne pismeno
        if slovo[i] not in kde.deti: return False
        hrana = kde.deti[ slovo[i] ]

        # ak ide, skontrolujeme, ci je cely text hrany spravny
        dlzka = min( hrana.po - hrana.od, len(slovo) - i )

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

        # ak text sedel, posunieme sa o vrchol nizsie
        i += dlzka
        kde = hrana.kam
    return True
```

```
T = input()
strom = vyrob_strom(T)

Q = int( input() ) # pocet otazok
for q in range(Q):
    slovo = input() # nacitame otazku
    print( zisti_ci_sa_nachadza( strom, slovo ) )
```

Príklad 2

Úloha: Na vstupe je dlhý reťazec `T`. Potom bude prichádzať veľa ďalších reťazcov. O každom z nich zistite, **koľkokrát** sa v `T` nachádza ako podreťazec.

Riešenie: Upravíme predchádzajúce riešenie. Po tom, ako strom zostrojíme, ho rekurzívne prejdeme a v každom vrchole si spočítame, koľko sufixov pod ním končí – teda koľko vrcholov pod ním (vrátane jeho samotného) má premennú `koniec` nastavenú na true.
Rozmyslite si, že keď máme v našom sufixovom strome vrchol r zodpovedajúci reťazcu `R`, tak každý koniec sufixu v podstrome s koreňom r zodpovedá jednému výskytu reťazca `R` v pôvodnom texte. Namiesto true/false teda na zadanú otázku odpovieme našou predpočítanou hodnotou.
V nasledujúcom listingu uvádzame len časti v ktorých sa líši od predchádzajúceho.

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

def kolkokrat_sa_nachadza(strom,slovo):
    # zisti, kolkokrat sa retazec "slovo" nachadza v retazci T, ktoreho 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 = vyrob_strom(T)
spocitaj_konce( strom.koren ) # <--- pred spracuvanim otazok raz predpocitame odpovede

Q = int( input() ) # pocet otazok
for q in range(Q):
    slovo = input() # nacitame otazku
    print( kolkokrat_sa_nachadza( strom, slovo ) )
```

TRIDSIATY PRVÝ ROČNÍK OLYMPIÁDY V INFORMATIKE

Príprava úloh: Michal Anderle, Michal Forišek, Askar Gafurov
Recenzia: Michal Forišek
Slovenská komisia Olympiády v informatike
Vydal: IUVENTA – Slovenský inštitút mládeže, Bratislava 2015