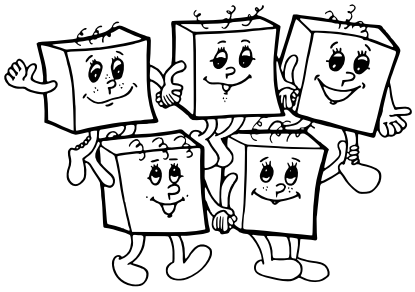
OLYMPIÁDA V INFORMATIKE

NA STREDNÝCH ŠKOLÁCH

http://oi.sk/



tridsiaty prvý ročník

školský rok 2015/2016

zadania celoštátneho kola, deň 1
kategória A

Priebeh celoštátneho kola

Celoštátne kolo 31. ročníka Olympiády v informatike, kategórie A, sa koná v dňoch 6. – 9. apríla 2016. Na riešenie úloh prvého, teoretického dňa majú súťažiaci 4,5 hodiny čistého času. Rôzne úlohy riešia súťažiaci na samostatné listy papiera. Akékoľvek pomôcky okrem písacích potrieb (napr. knihy, výpisy programov, kalkulačky) sú zakázané.

Čo má obsahovať riešenie úlohy?

- Slovne popíšte algoritmus.
  Slovný popis riešenia musí byť jasný a zrozumiteľný i bez nahliadnutia do samotného algoritmu/programu.
- Zdôvodnite správnosť vášho algoritmu.
- Uveďte a zdôvodnite jeho časovú a pamäťovú zložitosť.
- Podrobne uveďte dôležité časti algoritmu, ideálne vo forme programu v Pascale alebo C/C++.
- V prípade, že používate vo svojom programovacom jazyku knižnice, ktoré obsahujú implementované dátové štruktúry a algoritmy (napr. STL pre C++), v popise algoritmu stručne vysvetlite, ako by ste napísali program s rovnakou časovou zložitosťou bez použitia knižnice.

Hodnotenie riešení prvého (teoretického) dňa

Za každú úlohu môžete získať od 0 do 10 bodov.
Pokiaľ nie je v zadaní povedané ináč, najdôležitejšie dve kritériá hodnotenia sú v prvom rade **správnosť** a v druhom rade **efektívnosť** navrhnutého algoritmu. Na výslednom počte bodov sa môže prejaviť aj kvalita popisu riešenia a zdôvodnenie tvrdení o jeho správnosti a efektívnosti.
Efektívnosť algoritmu posudzujeme vypočítaním jeho časovej zložitosti – funkcie, ktorá hovorí, ako dlho vykonanie algoritmu trvá v závislosti od veľkosti vstupných parametrov. Nezávisí pri tom na konštantných faktoroch, len na rádovej rýchlosti rastu tejto funkcie.
V zadaní úlohy môžu byť uvedené limity na veľkosť premenných. Tieto môžete použiť na odhad toho, ako dobré vaše riešenie je. Na počítači, ktorý vykoná miliardu inštrukcií za sekundu, vyrieši vzorové riešenie ľubovoľný povolený vstup nanajvýš za niekoľko sekúnd.

A-III-1 Trojnohý beh

Štvrtáci sa už tešia na najbližší piatok. Namiesto vyučovania je vtedy totiž športový deň. Najsledovanejšou disciplínou je trojnohý beh. V trojnohom behu medzi sebou súťažia dvojice chlapec-dievča. Každá dvojica je pri behu spolu zviazaná: pravá noha chlapca je priviazaná k ľavej nohe dievčaťa, takže musia bežať obaja tým istým tempom.

Súťažná úloha

V škole je n chlapcov a n dievčat. O každom z nich vieme, akú má rýchlosť. Kvôli trojnohému behu ich chceme všetkých popárovať – teda vyrobiť n dvojíc. Rýchlosť dvojice bude pomalšou z rýchlostí detí, ktoré ju tvoria.

a) (5 bodov) Napíšte čo najefektívnejší program, ktorý zistí, aký *najväčší* môže byť súčet rýchlostí dvojíc.
b) (5 bodov) Napíšte čo najefektívnejší program, ktorý zistí, aký *najmenší* môže byť súčet rýchlostí dvojíc.

V oboch podúlohách je dôležitou súčasťou riešenia *dôkaz správnosti* použitého algoritmu.

Formát vstupu a výstupu

V prvom riadku vstupu je číslo n. V druhom riadku je n čísel: rýchlosti chlapcov. V treťom riadku je tiež n čísel: rýchlosti dievčat.
Vypíšte dva riadky: v prvom najväčší a v druhom najmenší možný súčet rýchlostí dvojíc.

Príklad

vstup

```
3
5 1 4
2 3 10
```

výstup

```
9
6
```

Jedno možné riešenie:

Najväčší súčet rýchlostí vieme dosiahnuť napr. tak, že chlapec s rýchlosťou 5 pobeží s dievčaťom s rýchlosťou 10, chlapec s rýchlosťou 1 s dievčaťom s rýchlosťou 2 a chlapec s rýchlosťou 4 s dievčaťom s rýchlosťou 3. Rýchlosti týchto dvojíc sú 5, 1 a 3, súčet rýchlostí je teda 9.

Najmenší súčet rýchlostí vyrobia napr. dvojice 5-2, 4-3 a 1-10. Rýchlosti týchto dvojíc sú 2, 3 a 1.

A-III-2 Elektromobil

Jano si nedávno kúpil nové moderné auto na elektrinu. Auto poslúcha, jazdí dobre, ale jeden problém tu predsa je: v našich končinách je pomerne málo dobíjacích staníc. Dostať sa z miesta na miesto môže teda občas byť netriviálne.

Súťažná úloha

Krajinu tvorí n miest a m ciest. Každá cesta je obojsmerná a spája niektoré dve mestá. Pre jednoduchosť budeme predpokladať, že všetky cesty majú rovnakú dĺžku.
V niektorých d mestách sú dobíjacie stanice. Auto má batériu, ktorej kapacita stačí na prejdenie k ciest.
Pre dané dve mestá a a b zistite, či sa dá dostať do mesta b, ak začíname v meste a a máme do plna nabitú batériu v aute.

Formát vstupu a výstupu

V prvom riadku vstupu sú čísla n, m, d, k, a, b. Mestá sú očíslované od 0 po $n-1$. V druhom riadku je d čísel: čísla miest, v ktorých sú dobíjacie stanice. Zvyšok vstupu tvorí m riadkov. Každý z týchto riadkov obsahuje čísla dvoch miest, ktoré sú spojené cestou.
Na výstup vypíšte „`ano`“ alebo `nie` podľa toho, či sa vieme dostať z a do b.

Obmedzenia a hodnotenie

Plný počet bodov môžu dostať riešenia, ktoré efektívne vyriešia vstupy, v ktorých máme $n \leq 100\,000$ miest a $m \leq 500\,000$ ciest (pričom d aj k môžu byť ľubovoľné).
Značný počet bodov dostanú riešenia, ktoré si poradia s podobne veľkými n a m, ale len za predpokladu, že niektorý z parametrov d a k je malý.
Ľubovoľné riešenie bežiace v polynomiálnom čase od n a m môže získať aspoň 4 body.

Príklad

vstup

```
9 8 2 3 0 7
3 5
0 1
1 2
2 3
2 4
4 5
4 6
6 7
7 8
```

výstup

```
ano
```

Odvezieme sa z 0 cez 1 a 2 do 3, kde dobijeme auto. Odtiaľ pôjdeme cez 2 a 4 do 5, znova dobijeme auto. Na záver sa z 5 odvezieme cez 4 a 6 do 7.

```
      5
      |
0-1-2-4-6-7-8
    |
    3
```

Keby namiesto $b = 7$ bolo $b = 8$, odpoveď by bola nie.

A-III-3 Sufixové stromy

V tomto zadaní nájdete najskôr zadanie súťažnej úlohy a následne študijný text o sufixových stromoch. Študijný text je identický s tým, ktorý bol v zadaniach domáceho a krajského kola.

Súťažná úloha, podúloha A (4 body)

Určite poznáte osemsmerovku: logickú úlohu, pri ktorej lúštení je potrebné nájsť nejaké slová v danej tabuľke písmen. Možno ste si ale neuvedomili, že množina slov, ktoré treba v osemsmerovke vyškrtať, nemôže byť len tak hocijaká. Okrem iného musí platiť, že žiadne z vyškrtávaných slov nie je podreťazcom iného z nich. Napríklad v dobrej osemsmerovke nebudú naraz slová `tvor` a `potvora`. Totiž potom by sa mohlo stať, že v osemsmerovke nájdem slovo `tvor`, vyškrtnem ho, ale potom sa ukáže, že to nebol ten správny `tvor` ale časť dlhšieho slova `potvora`.

Na vstupe sú dané reťazce $S_1, \ldots, S_n$. Napíšte program s optimálnou časovou zložitosťou, ktorý zistí, či je niektorý z týchto reťazcov podreťazcom iného.

Súťažná úloha, podúloha B (6 bodov)

Cyklický posun reťazca je každý reťazec, ktorý vieme vyrobiť tak, že pôvodný reťazec rozdelíme na dve časti a druhú presunieme pred prvú. Napr. všetky cyklické posuny reťazca `zabka` sú `zabka`, `abkaz`, `bkaza`, `kazab` a `azabk`. Keby sme tieto cyklické posuny usporiadali podľa abecedy, dostaneme nasledovné poradie: `abkaz`, `azabk`, `bkaza`, `kazab`, `zabka`.

Daný je reťazec S dĺžky n a číslo k, pre ktoré platí $1 \leq k \leq n$. Napíšte program s optimálnou časovou zložitosťou, ktorý vypíše k-ty najmenší cyklický posun reťazca S. Napr. pre $S =$ `zabka` a $k = 4$ by mal byť výstupom reťazec `kazab`. Pre $S =$ `baba` by aj pre $k = 1$ aj pre $k = 2$ mal byť výstupom reťazec `abab`.

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.

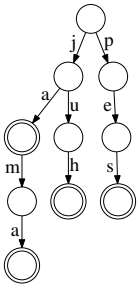
Abeceda

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.

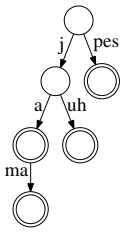


Obr. 1: Písmenkový strom predstavujúci množinu reťazcov {`ja`, `jama`, `juh`, `pes`}. Označené vrcholy sú znázornené dvojitým krúžkom.

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.
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`.

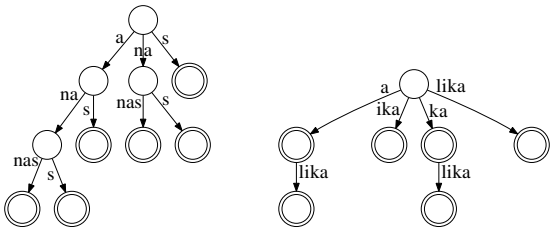
Prefixy, sufixy a podreťazce

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, Eduard Batmendijn, Michal Forišek, Marián Horňák, Jaroslav Petrucha
Recenzia: Michal Forišek
Slovenská komisia Olympiády v informatike
Vydal: IUVENTA – Slovenský inštitút mládeže, Bratislava 2016