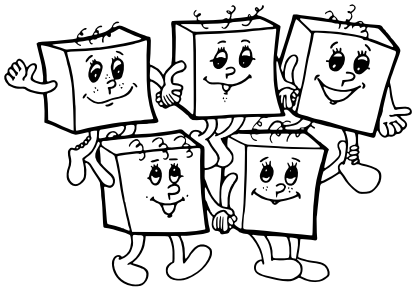
# OLYMPIÁDA V INFORMATIKE

## NA STREDNÝCH ŠKOLÁCH

http://oi.sk/



**tridsiaty prvý ročník**
školský rok 2015/2016

**zadania krajského kola**
**kategória A**

### Priebeh krajského kola

Krajské kolo 31. ročníka Olympiády v informatike, kategória A, sa koná 19. januára 2016 v dopoludňajších hodinách. Na riešenie úloh majú súťažiaci **4 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

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 úloh uvádzame časť „Hodnotenie“, v ktorej nájdete približné limity na veľkosť vstupných údajov. Pod pojmom „efektívne vyriešiť“ chápeme to, že váš program spustený na modernom počítači by mal dať odpoveď nanajvýš do niekoľkých sekúnd.
Údaje z tejto časti zadania by mali slúžiť hlavne na to, aby ste o riešení, ktoré vymyslíte, vedeli približne povedať, koľko bodov zaň dostanete.

## A-II-1 Hotel 2

Primomeňme si, že Aliho hotel má $p$ poschodí (očíslovaných od 1 po $p$) a na každom je $n$ jednolôžkových izieb. Jedného dňa bol celý hotel prázdny, keď tu zrazu sa na recepcii objavilo naraz $h$ hostí. O každom z nich vieme, aký je vysoký, a teda na aké najvyššie poschodie ho možno ubytovať. Okrem toho sa ale dá od pohľadu rozoznať aj to, koľko ktorý človek v hoteli utratí, a teda aký z neho bude mať Ali zisk. No a o ten samozrejme Alimu ide.

### Súťažná úloha

Na vstupe sú čísla $p$ a $n$ popisujúce hotel. Naraz nám prišlo $h$ hostí, ktorí sa chcú ubytovať. Hosťa číslo $i$ môžeme ubytovať len na poschodia 1 až $v_i$ a ak ho ubytujeme, tak zarobíme $q_i$ queliarov. Ktorých ľudí ubytujeme a ktorých pošleme preč, ak chceme maximalizovať Aliho zisk?
Dôležitou súčasťou riešenia je **dôkaz správnosti** vami navrhnutého algoritmu.

### Formát vstupu a výstupu

V prvom riadku sú čísla $p$, $n$, $h$. V druhom riadku sú čísla $v_1, \ldots, v_h$. V treťom riadku sú čísla $q_1, \ldots, q_h$.
V prvom riadku vypíšte, koľko queliarov Ali zarobí. V druhom riadku vypíšte $h$ medzerami oddelených celých čísel: poschodia na ktoré treba jednotlivých hostí (v poradí, v akom sú na vstupe) ubytovať, alebo -1 ak daného hosťa ubytovať nechceme. Ak existuje viac možností čo vypísať v tomto riadku, vyberte si ľubovoľnú.

### Obmedzenia a hodnotenie

Vzorové riešenie (hodné 10 bodov) si na bežnom počítači za sekundu poradí so vstupmi, v ktorých má hotel milióny izieb a milióny hostí.
Za 8 bodov je riešenie použiteľné pre $np \leq 5000$ a $h \leq 5000$.
Maximálne 6 bodov môžu získať riešenia s horšou polynomiálnou časovou zložitosťou.
Maximálne 3 body môžu získať riešenia s exponenciálnou alebo ešte horšou časovou zložitosťou.

### Príklady

vstup

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

výstup

```
14
1 -1 1 2 2
```

*Hotel má $p = 5$ poschodí a na každom $n = 2$ izby. Prišlo nám $h = 5$ hostí. Nie sú príliš vysokí: prví dvaja môžu bývať len na prvom poschodí, ďalší traja len na prvom alebo druhom.*

*Príklad výstupu ukazuje jedno optimálne riešenie. Za ubytovaných hostí zarobíme 2+4+5+3 = 14 queliarov.*

## A-II-2 Lyžovačka

Dano má rád rôzne extrémne športy, no nepohrdne ani obyčajnou lyžovačkou. Dnes sa vybral lyžovať do jedného malého lyžiarskeho strediska. V tomto lyžiarskom stredisku sa nachádza jedna sedačková lanovka a $n$ lokalít, očíslovaných od 0 po $n-1$. Spodná stanica lanovky je v lokalite 0, horná stanica je v lokalite 1. Všetky lokality majú navzájom rôzne nadmorské výšky, tie však nepoznáte. V niektorých lokalitách stoja bufety, kde si Dano môže dať čaj s rumom.
V lyžiarskom stredisku je $z$ zjazdoviek. Každá zjazdovka vedie smerom dole kopcom z nejakej lokality do nejakej inej. Občas sa môže stať, že sa nejaké dve zjazdovky križujú, ale všetky takéto križovatky sú riešené mimoúrovňovo (pomocou tunelov). Prejsť z jednej zjazdovky na druhú sa teda dá len v lokalite, kde prvá končí a druhá začína.

### Súťažná úloha

**Podúloha A (3 body).** Dano chce v lokalite 0 nastúpiť na sedačku, vyviezť sa ňou na lokalitu 1 a odtiaľ zísť nejakou postupnosťou zjazdoviek naspäť na lokalitu 0. Po ceste chce navštíviť práve jeden bufet. (Môže cestou prejsť aj okolo iných, v ktorých sa nezastaví.) Napíšte program, ktorý vypočíta, koľko bufetov má na výber.
**Podúloha B (7 bodov).** Dano chce spraviť rovnakú jazdu ako v podúlohe A, tentokrát sa však chce cestou postupne zastaviť v čo najviac bufetoch. Napíšte program, ktorý vypočíta, koľko najviac bufetov vie jednou cestou navštíviť.

V oboch podúlohách sa snažte, aby váš program bol efektívny pre čo najväčšie hodnoty $n$ a $z$.

### Formát vstupu a výstupu

V prvom riadku vstupu sú čísla $n$, $z$ a $b$: počet lokalít, zjazdoviek a bufetov.
V druhom riadku vstupu sú čísla $\ell_0, \ldots, \ell_{b-1}$: čísla lokalít, kde stoja bufety. Všetky tieto čísla sú z rozsahu od 2 po $n-1$ a sú navzájom rôzne.
Zvyšok vstupu tvorí $z$ riadkov, v každom z nich je popis jednej zjazdovky v tvare „odkiaľ kam“. Všetky dvojice „odkiaľ kam“ sú navzájom rôzne. Zjazdovky sú konzistentné s vyššie uvedeným popisom lyžiarskeho strediska. Navyše platí, že existuje aspoň jeden spôsob, ako sa po zjazdovkách dostať z lokality 1 na lokalitu 0.
Na výstup vypíšte jedno celé číslo: riešenie príslušnej podúlohy. V nižšie uvedenom príklade uvádzame vo výstupe postupne riešenia oboch podúloh.

### Príklady

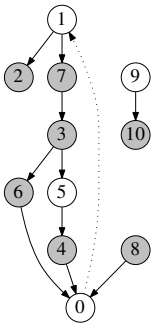
vstup

```
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
```

výstup

```
4
3
```

*Ak sa chce Dano zastaviť pri jednom bufete, má štyri možnosti: bufet v lokalite 3, 4, 6, alebo 7.*
*Jednou cestou vie navštíviť nanajvýš tri bufety: buď* $(7, 3, 4)$ *alebo* $(7, 3, 6)$.



## A-II-3 Miešanie kariet

Kleofáš vymyslel stroj na miešanie kariet. Stroj má v hornej časti otvor, do ktorého Kleofáš strčí balíček $n$ kariet. Potom potočí kľukou, stroj karty nejako zamieša a vypľuje zamiešaný balíček. Stroj pri tom karty mieša vždy rovnako: pre každú pozíciu v balíčku je jednoznačne určené, kam sa po zamiešaní dostane karta, ktorá bola pred zamiešaním na tejto pozícii.
Kleofáš ide hrať poker proti svojmu kamarátovi Leonardovi. Pri tom budú používať balíček $n$ kariet, z ktorých práve štyri sú esá. Aby Leonard Kleofáša nepodozrieval z podvádzania, dohodli sa, že pred hrou Kleofáš niekoľkokrát po sebe premieša balíček kariet na svojom stroji. Kleofáš však presne vie, akým spôsobom stroj mieša karty. A nielen to: vie aj kde v balíčku sú pred prvým zamiešaním esá. Po zamiešaní dostane Kleofáš vrchných 5 kariet z balíčka. Teraz by ho zaujímalo, koľko najviac es môže byť na konci celého miešania medzi piatimi vrchnými kartami, ak vhodne zvolí počet premiešaní. Okrem toho by chcel vedieť, aký je najmenší počet premiešaní, po ktorom bude medzi vrchnými piatimi kartami tento maximálny možný počet es.

### Súťažná úloha

Miešanie Kleofášovým strojom môžeme formálne popísať pomocou čísla $n$ a permutácie čísel 1 až $n$, teda takej postupnosti $a_1, a_2, \ldots, a_n$ čísel od 1 do $n$, v ktorej sa každé z čísel 1 až $n$ vyskytuje práve raz. Pre každé $x$ od 1 do $n$ potom platí, že karta, ktorá bola pred zamiešaním $x$-tá zhora, bude po zamiešaní $a_x$-tá zhora.
Na vstupe dostanete číslo $n$ a čísla $a_1, a_2, \ldots, a_n$ popisujúce Kleofášov stroj. Ďalej dostanete 4 čísla $r_1, r_2, r_3, r_4$ – pozície štyroch es v balíčku. Pozície v balíčku sú číslované zhora nadol, teda pozícia 1 znamená vrchnú kartu balíčka a pozícia $n$ znamená spodnú kartu balíčka.
Zistite, koľko najviac es môže byť po niekoľkých (možno nula) zamiešaniach medzi vrchnými piatimi kartami a po koľkých zamiešaniach bude medzi vrchnými piatimi kartami tento počet es prvý raz.
Pri písaní riešenia môžete predpokladať, že celočíselné premenné ktoré používate majú dostatočný rozsah na uloženie správneho výstupu.

### Hodnotenie

Akékoľvek korektné riešenie môže (v závislosti od kvality popisu) získať 3 body. Až 5 bodov môžu získať korektné riešenia, ktoré dokážu na bežnom počítači vyriešiť ľubovoľný vstup, kde $n \leq 20$ za menej ako sekundu. Až 7 bodov môžu získať riešenia, ktoré dokážu za menej ako sekundu vyriešiť ľubovoľný vstup, kde $n \leq 100$. Riešenie za plných 10 bodov by malo za menej ako sekundu vyriešiť ľubovoľný vstup, kde $n \leq 1\,000\,000$.

### Príklady

vstup

```
10
3 4 2 10 7 6 9 1 8 5
7 8 1 2
```

výstup

```
4
3
```

*Pre potreby tohoto vstupu si esá označme písmenami A, B, C, D a zvyšné karty zhora nadol číslami 1 až 6.*
*Pred miešaním sú teda karty (zhora nadol) v takomto poradí:* `A B 1 2 3 4 C D 5 6`
*Po prvom zamiešaní sú v takomto poradí:* `D 1 A B 6 4 3 5 C 2`
*Po druhom zamiešaní sú v takomto poradí:* `5 A D 1 2 4 6 C 3 B`
*Po treťom zamiešaní sú v takomto poradí:* `C D 5 A B 4 2 3 6 1`
*Viac es už medzi vrchnými piatimi kartami byť nemôže, teda Kleofáš vie získať najviac 4 esá a to najskôr po 3 zamiešaniach.*

vstup

```
10
2 3 4 5 6 7 8 9 1 10
1 2 3 10
```

výstup

```
3
0
```

*Vidíme, že eso na spodku balíčka za z tohto miesta nikdy nepohne, preto medzi vrchnými piatimi kartami môžu byť najviac tri esá. Toto prvý raz nastane ešte pred prvým premiešaním.*

## A-II-4 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 kola.*

### Súťažná úloha, podúloha A (5 bodov)

Na vstupe sú dané reťazce $S$ a $T$ tvorené malými písmenami anglickej abecedy. Napíšte program s optimálnou časovou zložitosťou, ktorý nájde jeden ich najdlhší spoločný podreťazec. Inými slovami, hľadáme najdlhší reťazec $R$ ktorý sa vyskytuje ako (súvislý) podreťazec aj v $S$, aj v $T$. Napr. pre $S =$ `kalerab` a $T =$ `prales` je jediným správnym riešením reťazec `ale`.

### Súťažná úloha, podúloha B (5 bodov)

Hovoríme, že reťazec $A$ má periódu dĺžky $p$, ak pre všetky relevantné $i$ platí $A[i] = A[p + i]$. Napríklad reťazec $A =$ `mamam` má periódu dĺžky 2, lebo platí $A[0] = A[2]$, $A[1] = A[3]$ aj $A[2] = A[4]$.

Ďalšie príklady: Najkratšia perióda reťazca `eeee` je 1. Najkratšia perióda reťazca `hahaha` je 2. Najkratšia perióda reťazca `koliesko` je 6. Najkratšia perióda reťazca `banan` je 5.

V premennej `strom` je sufixový strom nejakého neznámeho reťazca. Slovne popíšte algoritmus, ktorý priamo z tohto stromu vypočíta dĺžku najkratšej periódy dotyčného reťazca.

## 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

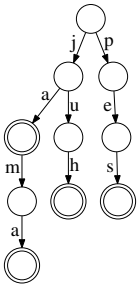
Vstupom pre všetky súťažné úlohy budú reťazce tvorené malými písmenami anglickej abecedy. Okrem nich sa nám občas oplatí interne použiť aj nejaké iné symboly. Budeme však predpokladať, že všetky použité symboly majú ASCII hodnoty od 33 po 126. Veľkosť abecedy teda budeme považovať za konštantu a nebudeme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

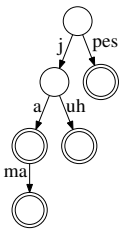



Obr. 1: Písmenkový strom predstavujúci množinu reťazcov {`ja`, `jama`, `juh`, `pes`}. Označené vrcholy sú znázornené dvojitým krúžkom.

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

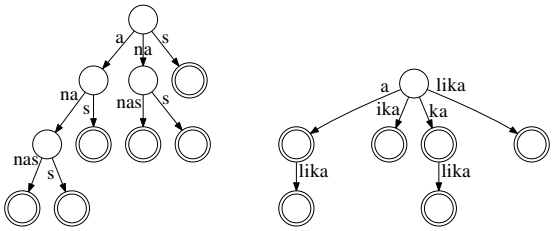
Vo viacerých úlohách sa budeme zaoberať podreťazcami daného reťazca. Tu považujeme za potrebné podotknúť, že pod slovom *podreťazec* budeme vždy rozumieť súvislý podreťazec, teda úsek po sebe nasledujúcich písmen v pôvodnom reťazci. Teda napr. reťazec `ace` nie je podreťazcom reťazca `abcde`.
Podreťazce 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
Recenzia: Michal Forišek
Slovenská komisia Olympiády v informatike
Vydal: IUVENTA – Slovenský inštitút mládeže, Bratislava 2015