

Řešení první série začátečnické kategorie 38. ročníku KSP

38-Z1-1 Odchycení zpráv

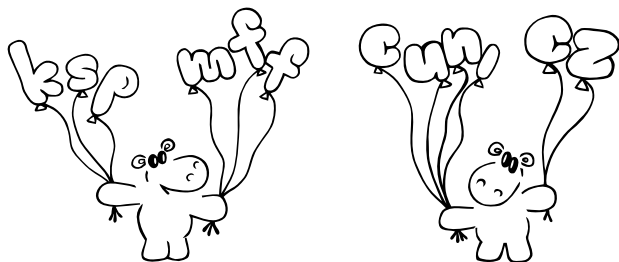
Zadání se ptalo, zda zašifrovaná zpráva obsahuje slovo `poklad` po dešifrování. To bychom mohli zjistit buď přímočaře dešifrováním zprávy a hledáním slova `poklad`, nebo se také nabízí možnost zašifrovat slovo `poklad` a hledat jeho zašifrovanou verzi ve stále zašifrované zprávě. Pokud to provedeme až při běhu programu, nebude mezi těmito způsoby příliš velký rozdíl, ale vzhledem k tomu, že hledané slovo ani způsob šifrování se nemění, nabízí se nám možnost *předem* zašifrovat `poklad`. Pak stačí pouze hledat `qplmbe` v textu.

Hledání v textu je v každém jazyce typicky nějakým způsobem velmi dobře implementované. Pythoní klíčové slovo `in` je pro obecné případy naprosto dostačující. Pokud byste si chtěli implementovat vlastní hledání v textu, nejspíš vymyslíte něco se složitostí $\mathcal{O}(NS)$, kde N je délka textu, ve kterém hledáme a S je délka slova, které hledáme. Existují však rychlejší algoritmy, které to zvládnou v čase $\mathcal{O}(N + S)$. Více o tomto tématu si můžete přečíst v naší kuchařce o vyhledávání v textu.¹

Máme tedy vyřešenou naši poměrně jednoduchou verzi úlohy, ale co kdyby naše šifrování nebylo posun o jeden znak, ale třeba posun o x znaků? Pak by se nám hodila nějaká obecná funkce na šifrování. Každý znak má nějakou číselnou hodnotu, určenou podle kódování ASCII (a Unicode). V Pythonu můžeme zjistit kódování nějakého znaku pomocí funkce `ord()` (převéde znak na hodnotu) a `chr()` (převéde číslo zpátky na znak). Také nám pomůže, že hodnoty pro celou anglickou abecedu jsou hned po sobě, tedy `a` má hodnotu 97, `b` má 98, a tak dále až k `z`, které má 122.

Stačí tedy jenom převést každý znak na číslo, přičíst posun, a pak převést zpátky na číslo. Nebo ne? Tady je potřeba dávat pozor na to, že hodnoty větší než `ord('z')` nejsou znaky anglické abecedy (třeba `chr('122')` = `{`). Toto můžeme jednoduše napravit tím, že můžeme získat index písmena v abecedě odečtením `ord('a')`, pak přičteme posun a dokud nám vyjde číslo větší než 25, odečteme 26. Poté v klidu přičteme zpátky `ord('a')` a získáme nový, poctivě zašifrovaný znak. Zašifrování jednoho znaku zvládneme v konstantním čase, asymptotická složitost našeho řešení tedy i v obecnější verzi zůstává nezměněna.

Úlohu připravili: Daniel Culliver, Riša Hladík



38-Z1-2 Zuzčín Standup

Na vstupu dostaneme informace o jednotlivých vystoupeních (počet vtipů v každém z nich) a požadavky jednotlivých pacientů. Naším úkolem je zjistit, kolik pacientů dokáže Zuzka vyléčit v daný den. Zadání říká, že se pacienti mohou během vystoupení střídat, což je klíčový poznatek, který využijeme.

Protože se pacienti mohou během vystoupení střídat, nezajímá nás počet vystoupení, ale celkový počet Zuzčíných vtipů. Nejprve spočítáme součet vtipů ze všech vystoupení a potom určíme pořadí, jak volat pacienty. Abychom maximalizovali počet vyléčených pacientů, vyplatí se nejdříve volat ty, kteří potřebují nejméně vtipů.

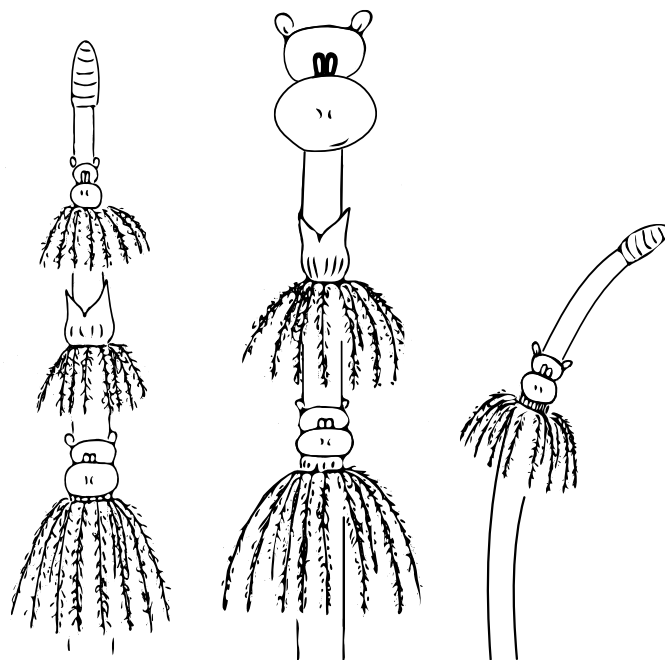
Pokud bychom místo toho vybrali pacienta s větším požadavkem, ubrali bychom si vtipy, které bychom mohli použít na pacienty s menšími požadavky. Tady může nastat situace, kdy už nebudeme mít dostatek vtipů na vyléčení dalších pacientů.

Tedy setřídíme pacienty vzestupně podle počtu potřebných vtipů a budeme odečítat jejich požadavky od Zuzčiny nabídky vtipů. Když Zuzčina nabídka klesne pod nulu, máme výsledný počet pacientů, které dokáže vyléčit.

Složitost algoritmu

Nejprve spočítáme celkový počet vtipů ze všech Zuzčíných vystoupení a pak setřídíme pacienty. Algoritmus má tedy časovou složitost $\mathcal{O}(N + M \log M)$, kde N je počet Zuzčíných vystoupení a M je počet pacientů.

Úlohu připravili: Riša Hladík, Vojta Káně, Vladimír Sklenár



¹ <https://ksp.mff.cuni.cz/viz/kucharky/hledani-v-textu/>

Úlohu stačilo odsimulovat. Při načtení vstupu ho projdeme řádek po řádku, abychom našli polohu a rotaci všech věží. Vstup si také někým uložíme, abychom mohli za běhu kontrolovat, zda je dané políčko volné. Pak stačí provádět jednotlivé simulační kroky. V každém kroku nejdřív projdeme seznam všech věží a s každou nezávisle na sobě pohneme – buď o jedno políčko dopředu, je-li to možné (kromě překážek nesmíme zapomenout kontrolovat ještě hranice mapy), případně věž otočíme o 90 stupňů doleva, jak káže zadání. Po pohnutí se všemi věžemi nastává jediný netriviální krok: slučování věží. To můžeme implementovat následovně: pořídíme si slovník, který pro každé políčko bude obsahovat seznam věží na tomto políčku. Ten umíme vyrobit prostou iterací přes všechny věže. Nyní vytvoříme nový seznam věží: projdeme postupně všechna políčka ve slovníku. Ta obsahující jen jednu věž prostě vložíme do seznamu věží nezměněna, a za ta, která obsahují věží více, vytvoříme novou věž, jejíž výška je součet všech věží na tomto políčku, a která je orientovaná nahoru. Po konci simulačního kroku můžeme zapomenout starý seznam věží a slovník políček a do dalšího kroku přeneseme jen nový seznam věží.

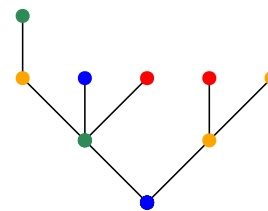
Je-li K počet věží, pak jeden simulační krok zabere $\mathcal{O}(K)$ času – s každou věží nejdříve jednou pohneme a krok slučování věží zabere jen lineární čas. Celková časová složitost T kroků simulace je tak $\mathcal{O}(TK + MN)$ – nesmíme zapomenout na načítání vstupu (v $\mathcal{O}(MN)$) a seřazení věží podle výšky ve finálním výstupu (výšky jsou čísla od 1 do K , takže stačí si ke každé výšce poznamenat, kolik věží s touto výškou máme, a pak výšky projít od 1 do K a věže vypsat – tomuto algoritmu se taky říká *counting sort* a můžete si o něm přečíst v naší kuchařce).²

Poznámka na závěr: existuje ještě jeden přístup k simulaci, který bohužel nebyl dostatečně rychlý. Můžeme napsat funkci, která vezme popis hracího pole a vrátí popis hracího pole o jeden krok později. Pokud tedy na hrací ploše vidíme $>.$, můžeme dané místo nahradit za $.>$. Tento přístup má ale samé nevýhody: jednak je zralý na opomenutí okrajových případů, zvlášť pokud spolu nějaké věže interagují (třeba $>>.$ se má přepsat na $.>>.$, ale naivní řešení ho může přepsat na $>.>.$ nebo $>.>$), a pak taky v každém kroku musíme projít celou mapu v čase $\Theta(MN)$, přestože zajímavá jsou jen políčka s věžemi. Na naše řešení jde nahlížet jako na zrychlenou verzi takovéto simulace, kde se díváme jen na zajímavá políčka.

Úlohu připravili: Daniel Culliver,
Kačka Doubková, Riša Hladík

V této úloze hrají důležitou roli jednotlivé hladiny, na kterých se květy mohou nacházet. Hledané sousední květy totiž musejí růst vždy na stejné hladině. Ideálně bychom chtěli umět stromem projíždět postupně po hladinách od nejnižší po nejvyšší a v každé hladině projíždět květy zleva doprava dle jejich uspořádání na dané hladině. Když toto budeme umět, tak na každé hladině už v podstatě jen řešíme problém hledání dvou po sobě jdoucích totožných prvků v posloupnosti. A to už je znatelně lehčí úloha. Jak ale docílíme tohoto kýženého způsobu procházení stromem?

Využijeme prohledávání stromu do šířky (BFS), které započneme v jediném vrcholu na první hladině. (Více o grafech, stromech a BFS si můžeš přečíst v naší základní³ a grafové⁴ kuchařce.) Označme tento vrchol m . BFS má totiž tu hezkou vlastnost, že prvky navštívuje v pořadí odpovídajícímu vzdálenosti od m , která zas přesně odpovídá jednotlivým hladinám. Konkrétně tedy nejprve vložíme m do fronty a následně v každém dalším kroku algoritmu vyjeme první květ ze začátku fronty a označíme si ho v . Vzpomeňme si, že každý květ, a tedy i v , má dán uspořádaný seznam květů z něj vyrůstajících. Pro květ v si tento seznam pojmenujeme ℓ_v . Následně na konec fronty přidáváme všechny květy ze seznamu ℓ_v přesně v tom pořadí v jakém se v něm nacházejí.



Tímto způsobem budeme navštěvovat jednotlivé květy na každé hladině zleva doprava, jak jsme chtěli. Ještě však potřebujeme umět rozpoznat, na jaké se zrovna nacházíme hladině. Jinak by se nám mohlo stát, že prohlásíme za sousední poslední květ z nějaké hladiny s prvním květem z bezprostředně následující hladiny. Tomuto můžeme zabránit tak, že si na začátek fronty umístíme zvláštní symbol – záračku. Vždy, když pak z fronty odebereme záračku, tak si zvýšíme počítadlo hladiny a vložíme záračku na konec fronty.

Časová složitost bude úměrná součtu počtu vrcholů (květů) a hran (spojnic mezi květy) stromu. Jelikož je ale ve stromu hran o jedna méně než vrcholů ($E = V - 1$), bude časová složitost v $\mathcal{O}(V)$. Prostorová složitost bude stejná, jelikož si musíme pamatovat celý strom.

Úlohu připravil: Janek Hartman

² <http://ksp.mff.cuni.cz/viz/kucharky/trideni>

³ <http://ksp.mff.cuni.cz/viz/kucharky/zakladni-algoritmy>

⁴ <http://ksp.mff.cuni.cz/viz/kucharky/grafy>