B-II-3 Tajná firma

V tomto vzorovom riešení sa najskôr pozrieme na ľahšiu verziu úlohy: situáciu, keď máme len jednu jedinú vizitku. Toto riešenie potom zovšeobecníme.

Jedna vizitka

Máme text jednej vizitky a chceme z neho vybrať reťazec, ktorý bude čo najneskôr v abecednom poradí. Ako na to? Začneme tým, že si uvedomíme jednoduchú skutočnosť: Reťazec je tým neskôr v abecednom poradí, čím je neskôr jeho prvé písmeno v abecede. Ideálne by sme teda chceli reťazec začínajúci písmenom `z`. Kedy sa to dá? Zjavne práve vtedy, keď sa na vizitke aspoň jedno `z` nachádza.
Ak teda máme nejaké `z`, musíme naše riešenie začať ním. A ak na vizitke nemáme žiadne `z`, chceli by sme začať naše slovo písmenom `y`. A tak ďalej. Inými slovami: nájdeme na vizitke písmeno, ktoré je najneskôr v abecede. Tým písmenom musí nutne začínať optimálne riešenie. Napríklad z písmen slova `dromedarpodoknom` je posledné v abecede `r`, hľadané meno firmy teda bude začínať písmenom `r`.
Na našej vizitke sa dotyčné písmeno može nachádzať aj viackrát. Ktorý jeho výskyt chceme použiť? Správna odpoveď je zjavná: *všetky*. Napríklad v slove `dromedarpodoknom` sú dve `r`. Ak použijeme *na začiatku mena Sonnyho firmy* postupne obe `r`, dostaneme meno začínajúce `rr...`. Akékoľvek meno firmy, ktoré by začínalo len jedným `r`, by bolo v abecednom poradí isto skôr, keďže ostatné písmená sú v abecede skôr ako `r`.

A to už je vlastne celé riešenie. Keď chceme *na začiatku* riešenia použiť všetky výskyty písmena, ktoré je najneskôr v abecede, musíme začierniť všetky *ostatné* písmená nachádzajúce sa pred jeho posledným výskytom. V našom príklade by sme teda z `dromedarpodoknom` dostali `_r_____rpodoknom`.
No a teraz už len pokračujeme veselo ďalej: zoberieme zvyšok reťazca (v našom prípade `podoknom`) a preň zopakujeme celú úvahu. A tak dokola, až kým sa nám celý text vizitky neminie.

Toto isté riešenie si môžeme sformulovať aj nasledovne: Kým sa nám ešte neminul text vizitky, opakujeme:
1. Nájdi najväčšie písmeno na vizitke.
2. Zmaž všetko po jeho najľavejší výskyt vrátane neho.
3. Pridaj toto písmeno na koniec riešenia.

Keďže v každom kole aspoň niečo z vizitky zmažeme, nanajvýš sa zopakuje ℓ takýchto kôl. Krok 3 beží v konštantnom čase, takže ten je efektívny. Krok 2 je tiež efektívny: dokopy, počas všetkých kôl, v ňom postupne zmažeme celý text vizitky, a teda všetky vykonania kroku 2 majú spolu časovú zložitosť $O(\ell)$.
Problém je s krokom 1. Keby sme ho robili hlúpo (a zakaždým prechádzali celý zostávajúci text na vizitke), trvalo by každé vykonanie kroku 1 čas $O(\ell)$ a my by sme tak dostali celkovú časovú zložitosť $O(\ell^2)$.
Krok 1 ale vieme spraviť šikovnejšie. Stačí si pre každé písmeno pamätať, koľkokrát sa ešte na nespracovanej časti vizitky nachádza. Pomocou tejto informácie vieme ďalšie písmeno riešenia nájsť v konštantnom čase. No a počas kroku 2 keď škrtáme písmená tak si zároveň zmenšujeme príslušné počítadlá ich počtov.

Listing programu (Python)

```
def ries_jednu(vizitka):
    # zmenime pismena na cisla od 0 do 25
    vizitka = [ ord(pismeno)-97 for pismeno in vizitka ]

    # zistime si pocty pismen
    pocty_pismen = [ 0 for i in range(26) ]
    for pismeno in vizitka: pocty_pismen[pismeno] += 1

    # najdeme najvacsie pismeno na vizitke
    najvacsie = 25
    while pocty_pismen[najvacsie] == 0: najvacsie -= 1

    # dokola: kym nie sme na najvacsom pismene, skrtame
    riesenie = ''
    for pismeno in vizitka:
        pocty_pismen[pismeno] -= 1
        if pismeno == najvacsie:
            riesenie += chr( 97+pismeno )
            # mozno sa nam minuli najvacsie pismena
            while najvacsie>0 and pocty_pismen[najvacsie]==0:
```

```
                najvacsie -= 1
    return riesenie
```

Ľubovoľne veľa vizitiek

Vďaka nadhľadu, ktorý sme získali pri riešení ľahšej verzie tejto úlohy, teraz už hravo vyriešime aj všeobecnú verziu. Čo sa zmení ak máme namiesto jednej vizitky tých vizitiek viac? Dokopy vlastne nič. Stále funguje tá istá úvaha: na začiatku riešenia chceme čo najväčšie písmeno. V našom všeobecnom prípade teda hľadáme písmeno, ktoré je čo najneskôr v abecede a ktoré sa vyskytuje na úplne všetkých vizitkách.
A to vieme efektívne spraviť presne rovnakým spôsobom. Na začiatku si pre každú vizitku a každé písmeno abecedy spočítame, koľkokrát sa na nej nachádza. Následne si predpočítame pre každé písmeno abecedy minimum z počtov jeho výskytov. (Ak máme napr. tri vizitky pričom na jednej je 5 **z**, na druhej sú 2 a na tretej 4, tak si pre **z** budeme pamätať, že minimum z počtov jeho výskytov je 2.)
Pomocou týchto údajov vieme rýchlo zistiť, akým písmenom bude začínať naše riešenie: najväčším z písmen, ktoré majú na každej vizitke aspoň jeden výskyt.
No a keď vieme, ktorým písmenom začína naše riešenie, tak na každej vizitke začiernime všetko po jeho *najľavejší* výskyt. (Prečo ten? Lebo zjavne nič nestratíme – keby sme použili iný výskyt, začiernili by sme toho viac, a teda by nám toho ostalo na výber menej.) A popri čiernení si nezabudneme zmenšovať príslušné počty písmen a tiež kontrolovať, či sa nám nemenia minimá ich počtov.
Každé písmeno každej vizitky raz začiernime alebo raz zoberieme do riešenia. Celková časová zložitosť tohto riešenia je teda $O(n\ell)$.

Listing programu (Python)

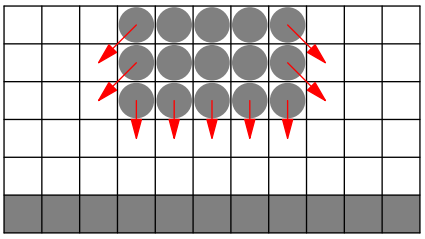
```
from queue import Queue                 # fronta: vieme v O(1) vybrat a spracovat prvy prvok
from collections import Counter         # pocitadlo: pamata si pocty vyskytov nejakych objektov
from string import ascii_lowercase      # konstanta obsahujuca nasu abecedu

# nacitame retazce

n = int( input() )
vizitky = [ input() for i in range(n) ]

# pre kazdy retazec si spocitame pocty pismen v nom
# napr. pocty_pismen[3]['a'] bude pocet 'a' v retazci vizitky[3]

pocty_pismen = [ Counter(vizitky[i]) for i in range(n) ]

# pre kazde pismeno si budeme pamatat minimum z jeho poctov vyskytov v jednotlivych slovach

minimalne_pocty_pismen = {}
for pismeno in ascii_lowercase: minimalne_pocty_pismen[pismeno] = min( pocty_pismen[i][pismeno] for i in range(n) )

# pismena kazdej vizitky si dame do fronty aby sme vedeli efektivne ich zahadzovat

for i in range(n):
    Q = Queue()
    for x in vizitky[i]: Q.put(x)
    vizitky[i] = Q

# postupne pre kazde pismeno abecedy:
# kym ho este mame na vsetkych vizitkach tak na kazdej pouzijeme jeho najlavejsi
# este nezacierneny vyskyt

riesenie = ''

for pismeno in ascii_lowercase[::-1]:
    while minimalne_pocty_pismen[pismeno] > 0:
        # pridame do riesenia
        riesenie += pismeno
        # z kazdej vizitky skrtame pismena kym nenajdeme zelane
        for i in range(n):
            while True:
                x = vizitky[i].get()
                pocty_pismen[i][x] -= 1
                minimalne_pocty_pismen[x] = min( minimalne_pocty_pismen[x], pocty_pismen[i][x] )
                if x == pismeno:
                    break

print(riesenie)
```

B-I-4 Piesok

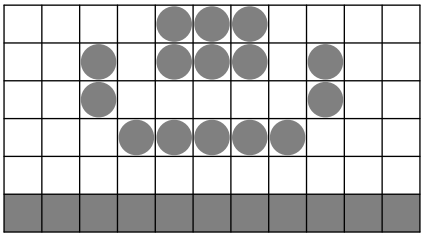
Postupne uvedieme riešenia všetkých troch podúloh v poradí, v akom boli uvedené v zadaní.

Podúloha A (3 body)

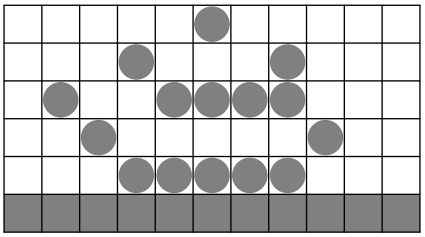
V tejto podúlohe stačilo len dôsledne aplikovať pravidlá pohybu zrniek. Pre rozloženie zo zadania sa zrnká chcú pohnúť nasledovne:



Po prvom takte teda dostaneme nasledujúce rozloženie:



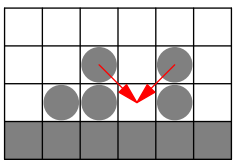
Z neho po druhom takte vznikne táto nesymetrická hrôza:



(Vyšla vám symetrická? To ste pravdepodobne prehliadli, že zrnko piesku, ktoré bolo po prvom takte v najpravejšom stĺpci najvyššie, nepadne v druhom takte doprava dodola ale doľava dodola.)
Pre zaujímavosť, hoci piesok robí počas pádu takéto roztodivné veci, o ďalšie tri takty sa dostane do stabilnej konfigurácie a tá bude opäť krásne symetrická.

Podúloha B (3 body)

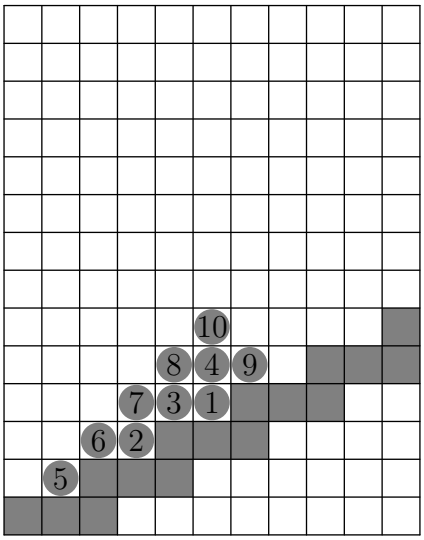
Pravidlo 2 treba použiť vtedy, keď na to isté miesto chcú isť práve dve zrnká: jedno šikmo zľava zhora a druhé šikmo sprava zhora. Volajme ich ľavé a pravé zrnko.
Pravé zrnko vyrobíme ľahko: stačí postaviť dve zrnká na seba, to horné bude chcieť spadnúť doľava dodola. Ľavé zrnko vyrobíme takmer tak isto, budeme však potrebovať až tri zrnká: samotné ľavé zrnko, zrnko pod ním (aby nemohlo padnúť rovno dole) a zrnko od neho vľavo dole (aby ľavé zrnko nemohlo padnúť ani šikmo doľava dodola). Výsledné rozloženie piesku teda vyzerá nasledovne:



Je zároveň jasné, že menej zrniek piesku nestačí: musíme mať aspoň dve, ktoré naozaj chcú padnúť na to isté políčko, a pod nimi aspoň tri ďalšie, aby tie dve padali tými správnymi smermi.

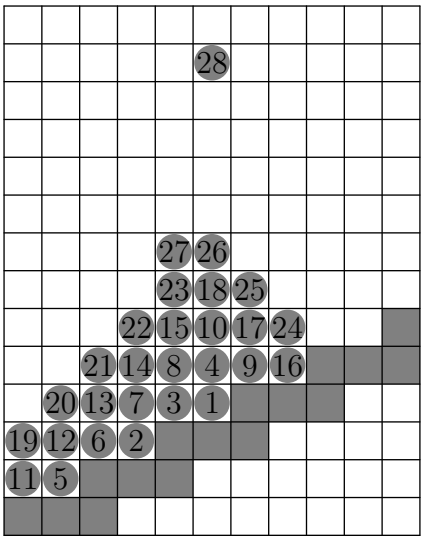
Podúloha C (4 body)

V domácom kole sme piesok sypali na rovnú dosku a ten tam postupne začal vyrábať „pyramídu“. A ľahko sa presvedčíme, že presne rovnako tomu bude aj v tomto prípade: pri pohľade zhora bude kopa piesku pyramídového tvaru. Len jej podstava nebude vodorovná, a preto ľavá hrana pyramídy bude dlhšia od pravej.
Na nasledujúcom obrázku sú záverečné polohy prvých 10 zrniek.



Všimnite si, že každá nová „vrstva“ pyramídy vzniká tak, že najskôr niekoľko zrniek odíde doľava, potom toľko isto odíde doprava, a na záver nasledujúce zrnko ostane sedieť v strede a vytvorí tak nový hrot pyramídy.
Najbližších päť zrniek (čísla 11 až 15) teda pôjde doľava, ďalšie dve (16 a 17) doprava a potom zrnko číslo 18 utvorí nový hrot o jedna vyššej pyramídy.

Pozorovanie, ktoré sme práve spravili, môžeme využiť na to, aby sme bez simulácie priamo zostrojili stav po 136 taktoch. Prvá vec, ktorú ľahko nahliadneme, je, že máme presne 28 zrniek: zrnko číslo 28 sa zjavilo po 135 taktoch. Toto posledné zrnko teda stihlo spadnúť len o jedno políčko dodola, a teda je ešte vo vzduchu.
Poloha zrnka číslo 27 je zatiaľ neznáma. Ale zrnko číslo 26 už bolo na svete 11 taktov, a teda už určite spokojne leží na svojom záverečnom mieste. Aby sme zistili, kde sa nachádza zrnko 27, potrebujeme zistiť, ako vyzerá pyramída tvorená prvými 26 zrnkami.
Zatiaľ vieme, že po zrnku 18 sme mali peknú kompletnú pyramídu. Zrnká 19 až 23 potom po náraze na jej špic pôjdu doľava (pričom zrnko 19 sa zastaví o okraj obrazovky), zrnká 24 a 25 pôjdu doprava, no a zrnko 26 bude novým špicom o jedno poschodie vyššej pyramídy.
Teraz už ľahko nájdeme aktuálnu polohu zrnka 27 a tým sme hotoví. Výsledná konfigurácia je na nasledujúcom obrázku:



TRIDSIATY ROČNÍK OLYMPIÁDY V INFORMATIKE

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