PRASK
V. ročník, 2018/19
Katedra základov a vyučovania informatiky FMFI UK,
Mlynská Dolina, 842 48 Bratislava

Vzorové riešenia 1. kola zimnej časti

vzorák napísal(a) Jitka+Emo
(max. 15 b za riešenie)

1. Podivný zošit

Všeobecne sa pri úlohách, kde treba odhaliť skryté pravidlo, oplatí postupne skúšať malé čísla. Ľahko si pri nich totiž overíme vzniknuté hypotézy. A až keď nám nič nedáva zmysel, skúšame väčšie čísla.

Level 1

Po pár pokusoch sme si mohli všimnúť, že vstupné číslo sa umocňuje na druhú. Na dosiahnutie čísla 35473936 teda stačilo zadať jeho odmocninu, t.j. 5956.

Level 2

Zošit nám v tejto úlohe nevypisoval číslo, ale písmená, prípadne dvojice písmen oddelených čiarkou. Tieto písmená nám sú povedomé z hodín chémie. Vieme, že prvý prvok v periodickej sústave prvkov je vodík, teda `H` a rovnako pre vstup 1 dostaneme výstup `H`. Na dosiahnutie `Al, I, Ca` trebalo zadať protónové čísla týchto prvkov, teda 135320.

Level 3

Rýchlo si môžeme všimnúť, že všetky vypísané čísla sú prvočísla. Číslo 7907 je 999. prvočíslo v poradí, ale kedže indexujeme od 0, tak správny výsledok je 998.

Level 4

Nenecháme sa zastrašiť chceným výstupom a skúsime zadať nejaké čísla. Už po pár pokusoch si šikovný pozorovateľ a znalec kalendáru všime, že zisťujeme, kto má meniny v daný deň roka. `Oto` má meniny 5.12. To je 339. deň v roku. Opäť ale začíname postupnosť od 0, preto je výsledok 338.

Level 5

Pri zadávaní za sebou idúcich čísel dostávame tú istú hodnotu. Kľúčové pozorovanie však je, že keď zadáme už zadané číslo, dostaneme inú hodnotu ako predtým. Prečo sme pre ten istý vstup dostali rôzne výstupy? Prejdeme očami po obrazovke a všimneme si počet doterajších pokusov. Hneď vidíme, že výstupom je (absolútny) rozdiel vstupného čísla a doterajšieho počtu pokusov. Správne riešenie teda závisí od počtu predchádzajúcich pokusov.

Level 6

S jednocifernými číslami sa nič nedeje, skúsime teda väčšie čísla. Počet cifier ostáva rovnaký ako na vstupe, no niečo sa s ciframi predsalen deje. Každá sa o niekoľko znížila. Môže sa zdať, že niektoré sa zvýšili, to však len prešli cez desiatku pri znižovaní. Každá cifra sa znížila o toľko, koľko cifier ju delí od prvej cifry, teda postupne o nula, jedna, dva... Správne riešenie je teda číslo 3684591.

Level 7

Keď postupne vyskúšame čísla od 1 do 10, všimneme si, že niektoré čísla dávajú veľmi podobný výstup (niekoľko 0 a potom jedna 1). Všimneme si, že čísla ktoré dávajú takýto výstup sú práve **prvočísla**. Čo sa stane, ak skúsime nejaké zložené číslo? Dostaneme jeho **prvočíselný rozklad**, pričom jednotlivé pozície predstavujú postupne prvočísla. Napríklad, ak zadáme číslo 24, dostaneme výstup 31, pretože $2^3 \cdot 3^1 = 24$.

Požadovaný vstup je teda čisło $2^2 \cdot 3^4 \cdot 5^0 \cdot 7^3 \cdot 11^2 = 13446972$.

(Ak ste boli dostatočne trpezlivý a skúšali ste aj vyššie čísla, isto ste si všimli, že ste častokrát dostali výstup `BLBE_CISLO`. To preto, že toto číslo bolo deliteľné priveľkým prvočíslom a výstup by sa nezmestil do odpoveďovového rámčeka).

Level 8

Všimneme si, že výstup je dlhší ako vstup. Vo veľa prípadoch dokonca až 2 krát dlhší. Menej ako 2 krát dlhší je len v prípade, že vo vstupe sú úseky rovnakých cifier. Zistíme, že výstup je o toľko dlhší koľko úsekov rovnakých cifier sa nachádza vo vstupe. Pre každý úsek rovnakých cifier nám výstup podáva nejakú informáciu.

Výstup si rozdelíme na dvojice cifier a priradíme ich úsekom rovnakých cifier vstupného čísla podľa poradia. Prvé číslo z dvojice hovorí o akú cifru v danom úseku ide, druhé číslo z dvojice hovorí koľko sa daných cifier v úseku nachádza. Už si len správne prečítame, čo máme vypísať a to jednu 2, tri 4 a päť 6, teda 244466666.

Level 9

Vidíme, že výstup je vždy tvaru {`číslo`}{`L alebo P`}, {`cislo`}{`H alebo D`}, prípadne ich kombinácia {`číslo`}{`L alebo P`}{`cislo`}{`H alebo D`}. Čo by mohli znamenať tie písmenká? Sú to smery *(L = ľavo, P = pravo, H = hore, D = dole)*. Keď si začneme kresliť na papier tieto pozície, zistíme, že tvoria peknú špirálu.

16	**15**	**14**	**13**	**12**
17	**4**	**3**	**2**	**11**
18	**5**	**0**	**1**	**10**
19	**6**	**7**	**8**	**9**
20	**21**	**22**	**23**	**24**

2L2H	**1L2H**	**2H**	**1P2H**	**2P2H**
2L1H	**1L1H**	**1H**	**1P1H**	**2P1H**
2L	**1L**	—	**1P**	**2P**
2L1D	**1L1D**	**1D**	**1P1D**	**2P1D**
2L2D	**1L2D**	**2D**	**1P2D**	**2P2D**

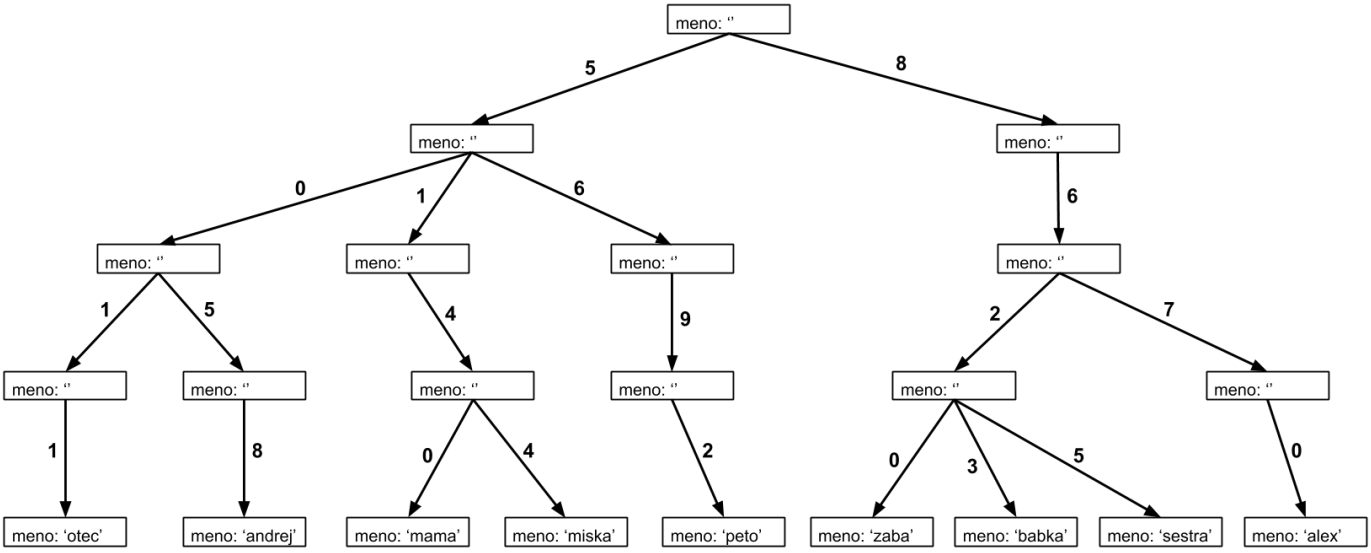
Výsledok buď odhadneme a postupným skúšaním doladíme alebo ho vypočítame priamo *(odporúčame vyskúšať ;))*. Pri odhadovaní si vieme všimnúť, že na diagonále sa vyskutujú druhé mocniny *(ľavá horná diagonála obsahuje mocniny párnych čísel, pravá dolná mocniny nepárnych -1)*. Všimnime si, že ak máme zistiť odpoveď pre `31L17D`, tak táto pozícia bude ležat na nejakom štvorci s rozmermi 62, pretože pozície `31L31D`, `31L31H`, `31P31D` a `31P31H` ležia na tom istom štvorci. Teda vieme, že výsledok bude aspoň $(31 \cdot 2)^2 = 3844$, následne už len správne doladíme pozíciu a nájdeme správne číslo 3892.

Level 10

Pre mocniny dvojky vždy dostaneme na výstupe číslo 1. V akom zápise čísla mocniny dvojky vyzerajú ako 1? V binárnom. Tam však potrebujeme aj nejaké 0. Výstupom teda nebude celý binárny zápis vstupného čísla, ale len počet 1 v tomto zápise. Riešením je ľubovolné číslo, ktoré v binárnom zápise obsahuje práve osem 1, takýmto číslom je napríklad 32385.

2. Reprezentácia telefónnych čísel

vzorák napísal(a) Žaba
(max. 15 b za riešenie)



Podúloha a.

Zistiť, ktoré kontakty sú v telefónne nie je ťažké, stačí prečítať všetky mená v spodnom riadku obrázku. Ak k nim však chceme priradiť čísla, musíme si uvedomiť, ako táto reprezentácia funguje. Pomôcť nám pri tom môže kontakt `babka` so známym číslo `8623`.

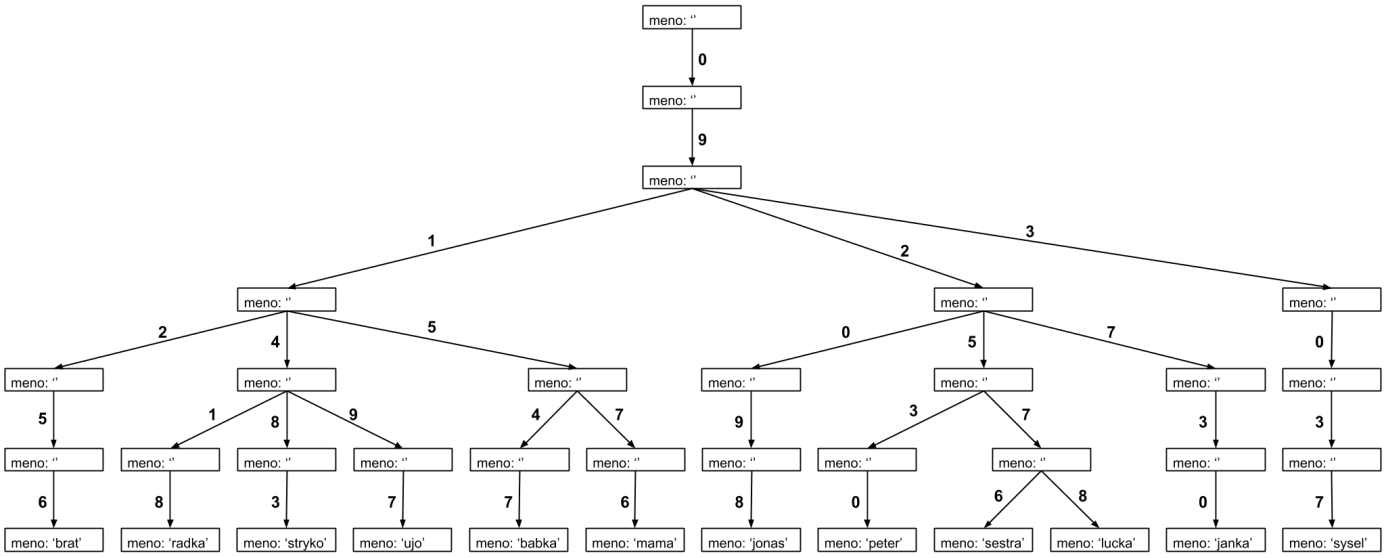
Keď tieto čísla hľadáme na obrázku, ľahko si všimneme, že zodpovedajú šipkám, ktoré vedú z vrchu až ku kontaktu s menom `babka`. Je preto prirodzené, že rovnaká vlastnosť bude platiť aj pre zvyšné kontakty. Ak teda chceme zistiť telefónne číslo pre zadaný kontakt, prejdeme po šipkách od vrchu obrázku k jeho spodku. Dostávame nasledovné kontakty:

```
otec:   5011      miska:  5144      babka:  8623
andrej: 5058      peto:   5692      sestra: 8625
mama:   5140      zaba:   8620      alex:   8670
```

Podúloha b.

Našou úlohou je nakresliť obrázok zo zadaných kontaktov. V predchádzajúcej podúlohe sme si všimli, že šipky od vrchu obrázku ku konkrétnemu kontaktu majú označenie podľa jednotlivých cifier príslušného čísla. Pri kreslení je však dôležitá ešte jedna vlastnosť, ktorú si ľahko všimneme. Z každého obdĺžnika vychádza **najviac jedna šipka s konkrétnou cifrou**. Vďaka tomu spájame dokopy čísla s rovnakým začiatkom a rozdeľujeme ich až keď je to naozaj nevyhnutné.

Na obrázku zo zadania napríklad vidíme, že z vrchného obdĺžnika nevychádza 8 šipiek, jedna pre každý kontakt, ale iba 2, pretože všetky kontakty v Adamovom telefónne začínajú číslom 5 alebo 8. Keď sa pozrieme do zadaných kontaktov, všimneme si, že všetky začínajú na `09`. To znamená, že z najvrchnejších dvoch obdĺžnikov bude vychádzať iba jedna šipka a až na tretej úrovni sa to začne rozdeľovať.



Podúloha c.

Skôr ako sa pustíme do riešenia úlohy, zaveďme si nejaké označenie, aby sa nám o ňom ľahšie rozprávalo. Celý náš obrázok vyzerá ako *strom* (obrátený hore nohami), budeme preň preto používať toto označenie. Jednotlivé obdĺžniky nazveme *vrcholy* a šipky z nich *hrany*. Najvrchnejší vrchol má naviac špeciálny význam a preto aj názov – *koreň*.

Pre zadaný začiatok čísla máme nájsť ľubovoľný kontakt, ktorého telefónne číslo má rovnaký začiatok. Kde by sa však mohol nachádzať? Ako sme si všimli už v podúlohe b., všetky kontakty s rovnakým začiatkom sa v našom strome nachádzajú pokope. Naviac, ich umiestnenie vieme ľahko nájsť, stačí ísť od koreňa stromu po hranách s príslušnými ciframi.

Zoberme teda zadaný začiatok čísla. Začnúc v koreni stromu sa postupne posúvame po hranách, ktorých označenie zodpovedá ďalšej cifre tohto čísla. Uvedomme si, že v každom momente sa pod aktuálnym vrcholom nachádza časť stromu (*podstrom*), v ktorom sú všetky kontakty začínajúce sa na už spracovaný úsek čísla. Keď teda spracujeme celé zadané číslo, budeme vo vrchole, pod ktorým sú všetky kontakty, ktoré majú takýto začiatok telefónneho čísla. My si môžeme vybrať ľubovoľný z nich. To znamená, že odtiaľto je už jedno po ktorých hranách prejdeme, stačí že pôjdeme dodola až kým sa nedostaneme k vrcholu, ktorý obsahuje `meno` niektorého kontaktu a toto meno vypíšeme.

Uvedomme si ešte, že sa mohlo stať, že **žiadny** kontakt nezačínal na zadané číslo. To znamená, že v niektorom momente nášho postupu sme chceli použiť nejak označenú hranu, ktorá ale z aktuálneho vrcholu nevychádzala. V tom momente sme teda mohli postup ukončiť a vyhlásiť, že taký kontakt v mobile nie je.

Nami navrhnutý postup je navyše pomerne efektívny. V každom jeho kroku sa totiž posunieme o jednu úroveň nižšie, spracujeme jednu cifru telefónneho čísla. Bez ohľadu na to, koľko kontaktov má Adam v telefónne bude počet krokov tohto algoritmu úmerný dĺžke telefónnych čísel.

Podúloha d.

Po stlačení tlačidla "Zoptimalizuj kontakty" sa nám do každého vrchola pridala položka `dopln`. Z obrázku ľahko vydedukujeme, že táto hodnota zodpovedá jednému menu kontaktu, ktoré sa nachádza v podstrome príslušného vrcholu.

Riešenie z prechádzajúcej podúlohy bude teraz zrazu oveľa jednoduchšie. Tak ako predtým pôjdeme z koreňa stromu po hranách označených príslušnými ciframi. Keď však spracujeme celé zadané číslo, nemusíme v našom zostupe pokračovať. Jednoducho sa pozrieme na hodnotu `dopln` vo vrchole, kde sme zastavili a použijeme tú. Presne to totiž táto hodnota predstavuje – ľubovoľný kontakt začínajúci správnym číslom.

O niečo komplikovanejšie je vymyslieť, ako tieto hodnoty spočítať. Prvé riešenie, ktoré nám môže napadnúť je pomerne priamočiare. Pre každý vrchol pôjdeme dodola po ľubovoľných hranách až kým sa nedostaneme k nejakému kontaktu a tento kontakt do tohto vrcholu zapíšeme. To je však strašne pomalé, pre každý vrchol totiž musíme spraviť zhruba toľko operácií, ako dlhé sú čísla.

Ako to teda vyriešiť jednoduchšie? Možno neriešme všetko naraz, ale pozrime sa na tie vrcholy, pre ktoré je odpoveď triviálna. To sú všetky tie v najnižšej úrovni nášho stromu. Pre každý vrchol, ktorý má priradené `meno` totiž môžeme rovno doplniť aj rovnakú hodnotu `dopln`. V tomto podstrome je totiž iba jeden vyhovujúci kontakt a to práve tento.

No a keď už máme najspodnejšie vrcholy spracované, ktoré ďalšie vieme doplniť veľmi jednoducho? Predsa tie nad nimi! Každý z vrcholov nad nimi si vyberie jedného svojho *syna* (vrchol priamo pod ním) a použije jeho hodnotu `dopln`, je pritom jedno, ktorého syna si vyberie. Vďaka tomu ale vieme rýchlo vyriešiť aj tretiu vrstvu odspodu. Všetci synovia vrcholu na tejto vrstve už majú doplnenú hodnotu `dopln`. Opäť mu teda stačí si vybrať ľubovoľného syna a hodnotu `dopln` skopírovať.

Všimnite si ten výrazný rozdiel týchto dvoch riešení. V prvom sme začínali z vrchu a zakaždým sme museli ísť až po spodok, aby sme sa dostali k relevantnej informácii. V druhom riešení však začíname už známymi informáciami a postupne ich posúvame dohora. Pre každý vrchol nám teda stačí pozrieť sa na jedného z jeho synov a iba z neho skopírovať `dopln`, čo je rýchlejšie, ako keď sme museli ísť úplne dodola. Pre každý vrchol spravíme len jedinú operáciu.

Podúloha e.

Keď rozumieme riešeniu podúlohy d., vyriešiť podúlohu e. je jednoduché. Použijeme rovnaký prístup. Do každého vrcholu si pridáme novú hodnotu `pocet`. Táto hodnota označuje počet kontaktov v podstrome určenom daným vrcholom. Táto hodnota je jasne určená pre spodnú vrstvu vrcholov, kde platí, že `pocet: 1`.

No a počet kontaktov v podstrome nejakého vyššieho vrchola vieme vypočítať cez súčet počtov kontaktov v jeho synoch. Opäť teda budeme postupovať od spodku stromu a postupne upravovať informáciu v jednotlivých vrcholoch. Pre každý vrchol budeme musieť spočítať hodnoty `pocet` všetkých vrcholov priamo pod ním. Vďaka tomu zistíme, koľko kontaktov začína na zadané číslo.

Postup vypisovania tohto čísla je už potom rovnaký ako v podúlohe d., jednoducho prejdeme po hranách, ktoré zodpovedajú číslam zadaným používateľom a vypíšeme hodnotu `pocet` z posledného takto navštíveného vrcholu.

Písmenkový strom

Štruktúra, ktorú sme si predstavili v tejto úlohe sa volá **písmenkový strom** (po anglicky tiež trie) a väčšinou sa používa na ukladanie slov, ktoré sú tým pádom tiež zoskupené podľa ich začiatkov.

S pomocou písmenkových stromov vieme počítať množstvo zaujímavých informácií o uložených slovách alebo číslach a aj keď reálne použitie je častokrát komplikovanejšie, základná myšlienka ostáva rovnaká. Pri hľadaní postupuj od vrchu po zodpovedajúcich hranách a pri predpočítavaní posúvaj informáciu od spodu stromu ku koreňu.

3. Audio Práskači

vzorák napísal(a) Roman
(max. 15 b za riešenie)

K jednotlivým podúlohám sme pre vás pripravili videovzoráky.

4. Skriňa plná tričiek

vzorák napísal(a) Mišof
(max. 15 b za riešenie)

V našom prvom riešení využijeme, že počty nosení tričiek sú rozumne malé (neprevyšujú 10^6). Namiesto jednej obrovskej kopy si spravíme veľa menších kôp: pre každé x si spravíme kopu, na ktorej budú všetky tričká, ktoré mal Žaba zatiaľ na sebe presne x-krát. Na začiatku tam tričká rozmiestnime tak, aby ich poradie zhora dole zodpovedalo poradiu zhora dole v pôvodnej kope.

Navyše si ešte spravíme jednu pomocnú premennú, v ktorej si budeme pamätať, ktoré tričko je práve na vrchu celej kopy.

Ako bude vyzerať deň podľa Lucky? Ráno Žaba nájde najmenšie x, ktorému zodpovedá neprázdna kopa tričiek. Z tejto kopy si zoberie vrchné tričko – teda to, ktoré bolo najbližšie k vrchu pôvodnej kopy. No a keď ho večer operie a usuší, položí ho na vrch kopy s číslom $x+1$. (Všimnite si, že aj po tomto úkone platí, že tričká na kôpke $x+1$ sú uložené v tom poradí, v ktorom by v skutočnosti boli v pôvodnej kope – práve nosené tričko totiž šlo na vrch úplne celej pôvodnej kopy.) No a na záver si už len práve donosené tričko uložíme do premennej pre vrch kopy.

A ako bude vyzerať deň podľa Žabu? Jeho spôsob vyberania trička sa simuluje ľahko: stačí sa pozrieť do pomocnej premennej na tričko na vrchu kopy. Potom, ako ho donosí, ho však nesmieme zabudnúť presunúť „o kôpku vyššie“ – teda ak doteraz bolo na kôpke pre y nosení, odteraz bude na kôpke pre $y+1$.

Na záver zopár implementačných detailov: * Keďže žiadnemu tričku počet nosení nemôže ubudnúť, nemusí Žaba hľadať najmenšiu neprázdnu kopu vždy odznova. Namiesto toho môžeme šikovne využiť, že ak naposledy bral tričko z kôpky x, najbližšie ho bude brať z kôpky s číslom aspoň x. * Na zapamätanie kôpky nám stačí ľubovoľná dátová štruktúra, z ktorej vieme na jednom konci efektívne vyberať prvky a aj na ten istý koniec

nové vkladať. Základnou takouto štruktúrou je zásobník. Môžeme však použiť aj všeobecnejšie dátové štruktúry (ako napr. vector v C++, ArrayList v Jave, prípadne list v Pythone). * Pri šikovnej implementácii dostávame časovú zložitosť $O(n+q+m)$, kde n je počet tričiek, q je počet dní, ktoré treba odsimulovať, a $m = \max t_i$ je najväčší počet oblečení trička doteraz.

Listing programu (Python)

```
N, Q = [ int(_) for _ in input().split() ]
T = [ int(_) for _ in input().split() ]
M = max(T)
T = [ None ] + T
na_vrchu = 1

# spravíme si prázdne kôpky
kopky = [ [] for _ in range(M+Q+1) ]

# rozmiestnime tričká na kôpky podľa počtu nosení
for n in reversed(range(1,N+1)):
    kopky[ T[n] ].append(n)

odkial_ber = 0

# čítame otázky a simulujeme
for q in range(Q):
    if input() == 'Z':
        print(na_vrchu)
        kde = T[na_vrchu]
        kopky[kde].pop()
        kopky[kde+1].append(na_vrchu)
        T[na_vrchu] += 1
    else:
        while kopky[odkial_ber] == []:
            odkial_ber += 1
        nosil = kopky[odkial_ber].pop()
        print(nosil)
        na_vrchu = nosil
        kopky[odkial_ber+1].append(nosil)
        T[nosil] += 1
```

Iné efektívne riešenie môžeme založiť na použití pokročilejšej dátovej štruktúry: usporiadanej množiny. Presnejšie, dvoch takýchto množín. V jednej si budeme udržiavať všetky tričká usporiadané podľa toho, kedy boli naposledy nosené (čiže podľa poradia vo veľkej kope zhora dole) a v druhej budú usporiadané primárne podľa počtu nosení a až sekundárne podľa poradia vo veľkej kope.

Podľa toho, kto si v ten deň vyberá, vyberieme záznam o tričku zo začiatku jednej alebo druhej usporiadanej množiny. Následne tento záznam z oboch zmažeme a namiesto neho tam vložíme nový záznam, v ktorom už má toto tričko o jedno nosenie viac a navyše dostalo nový vyšší timestamp (čiže je aj naposledy nosené zo všetkých).

Každá operácia s usporiadanou množinou, v ktorej je n tričiek, nám trvá $O(\log n)$, preto má toto riešenie časovú zložitosť $O((n+q)\log n)$.

Listing programu (C++)

```
#include <bits/stdc++.h>
using namespace std;

struct tricko { int cislo, timestamp, pocet_noseni; };

struct porovnaj_timestamp {
    bool operator() (const tricko &A, const tricko &B) const { return A.timestamp > B.timestamp; }
};

struct porovnaj_pocet {
    bool operator() (const tricko &A, const tricko &B) const {
        if (A.pocet_noseni != B.pocet_noseni) return A.pocet_noseni < B.pocet_noseni;
        return A.timestamp > B.timestamp;
    }
};

int main() {
    int N, Q;
    cin >> N >> Q;

    // spravime si dve mnoziny triciek: jednu usporiadanu primarne podla poctu noseni,
    // druhu podla toho kedy bolo naposledy nosene
    set<tricko,porovnaj_pocet> najmenej_nosene;
    set<tricko,porovnaj_timestamp> naposledy_nosene;

    // ulozime tricka do oboch mnozin
    for (int n=0; n<N; ++n) {
        int t;
        cin >> t;
        najmenej_nosene.insert( {n+1, N-1-n, t} );
        naposledy_nosene.insert( {n+1, N-1-n, t} );
    }

    // spracuvame jednotlive dni
```

```
    for (int q=0; q<Q; ++q) {
        string kto;
        cin >> kto;
        tricko dnes;
        if (kto == "Z") dnes = *naposledy_nosene.begin(); else dnes = *najmenej_nosene.begin();
        cout << dnes.cislo << endl;
        najmenej_nosene.erase(dnes);
        naposledy_nosene.erase(dnes);
        dnes.timestamp = N+q;
        ++dnes.pocet_noseni;
        najmenej_nosene.insert(dnes);
        naposledy_nosene.insert(dnes);
    }
}
```

5. Korálkový náhrdelník

vzorák napísal(a) Paulinka
(max. 15 b za riešenie)

Riešenie za 3 body

Poďme sa pustiť do riešenia. Pre každý začiatok náhrdelníku máme n možných koncov, kde môže končiť. Napríklad pre náhrdelník `3 1 2 1 3 2` a začiatok na tretej korálke (korálke `2`) chceme postupne vyskúšať náhrdelníky (`2`), (`2 1`), (`2 1 3`), (`2 1 3 2`), (`2 1 3 2 3`), (`2 1 3 2 3`) a (`2 1 3 2 3 1`). Ako vieme overiť, či náhrdelník spĺňa zadanú vlastnosť? Môžeme si pamätať, ktoré farby korálok sa v ňom nachádzajú. Dokonca nám postačia dve farby, pretože ak by sme mali nájsť tretiu farbu, vieme, že daný úsek nebude vhodný.

Pre každý možný náhrdelník, ktorých je n^2 (pre každý začiatok mám n koncov) prejdeme týmto náhrdelníkom a skontrolujeme, či obsahuje najviac dve farby. Na pamätanie farieb použijeme dve premenné, v ktorých si na začiatku budeme pamätať hodnotu `-1` značiacu, že sme ešte nevideli obe farby.

Takéto riešenie má časovú zložitosť $O(n^3)$

Listing programu (Python)

```
n = int(input())
koralky = list(map(int, input().split()))
koralky += koralky; # skopirujeme si pole za seba, aby sa nam jednoduchsie implementovalo to, ze koralky su v kruhu

najlepsia = 0
kde = 0
for i in range(n):
    for j in range(n):
        prva, druha = -1, -1
        moze = True
        for l in range(j + 1):
            if prva == -1: # toto je prva koralka
                prva = koralky[i + l]
            elif prva != koralky[i + l] and druha == -1:
                druha = koralky[i + l]
            elif prva != koralky[i + l] and druha != koralky[i + l]: # mame privela farieb
                moze = False
        if moze and najlepsia < j + 1:
            najlepsia = j + 1
            kde = i

print(najlepsia)
print(kde + 1, n if (kde + najlepsia) % n == 0 else ((kde + najlepsia) % n))
```

Zlepšenie na 9 bodov

Všimnime si, že ak pre daný začiatok nefunguje úsek s dĺžkou k, nebude preň fungovať ani žiaden dlhší úsek. Napríklad v príklade vyššie vidíme, že už úsek (`2 1 3`) obsahuje priveľa farieb. Je preto prirodzené, že keď tento úsek predĺžime o ďalšie korálky, počet farieb sa nezníži.

Na druhej strane si však môžeme uvedomiť, že ak sme pred chvíľou spracovali úsek dlhý k, tak pri spracovávaní úseku dĺžky $k+1$ nemusíme začínať odznova. Stačí pridať len tú jednu korálku navyše, skontrolovať či má jednu z dvoch farieb v úseku dĺžky k, poprípade ak bola v úseku dĺžky k len jedna farba poznačiť si farbu korálky $k+1$ (ak je náhodou rôzna od tých predtým).

Toto riešenie je rýchlejšie, pretože pre každý začiatok nám stačí iba raz prejsť celým náhdrelníkom. Pri tomto prechode postupne zisťujeme vhodosť každého konca a to vždy iba pridaním jednej korálky. Časová zložitosť je preto $O(n^2)$.

Listing programu (Python)

```
n = int(input())
koralky = list(map(int, input().split()))
koralky += koralky; # skopirujeme si pole za seba, aby sa nam jednoduchsie implementovalo to, ze koralky su v kruhu
```

```
najlepsia = 0
kde = 0
for i in range(n):
    prva, druha = -1, -1
    for j in range(n):
        if prva == -1:
            prva = koralky[i + j]
        elif prva != koralky[i + j] and druha == -1:
            druha = koralky[i + j]
        elif prva != koralky[i + j] and druha != koralky[i + j]: # mame privela farieb
            if najlepsia < j:
                najlepsia = j
                kde = i
            break
        if j == n - 1:
            najlepsia = n
            kde = i

print(najlepsia)
print(kde + 1, n if (kde + najlepsia) % n == 0 else ((kde + najlepsia) % n))
```

Vzorové riešenie

V predchádzajúcom riešení sme využili fakt, že úseky s rovnakým začiatkom a takmer rovnakým koncom sa líšia minimálne. Dokonca ak je rozdiel ich dĺžok jedna, tak sa líšia iba pridaním jedinej guličky. Toto pozorovanie je však symetrické a rovanko dobre vieme povedať, že úseky s rovnakým koncom a začiatkom líšiacim sa o jedna sú rozdielne iba v začiatočnej guličke.

Toto pozorovanie teraz využijeme. Zoberme si nejaký fixný začiatok a postupom vyššie nájdime najdlhší vhodný náhrdelník s týmto začiatkom, nech má dĺžku l. Keďže dlhší náhrdelník neexistuje, musíme sa posunúť na nový začiatok. Uvedomme si však, že nemusíme s koncom začínať opäť od začiatku. Aktuálny koniec nám totiž určite vyhovuje. Na pozíciách od 0 po l boli najviac dve rôznofarebné guličky. Z toho ale plynie, že aj na pozíciách 1 až l budú najviac dve farby korálie. A možno dokonca menej, ak bola na pozícii 0 jediná korálka príslušnej farby v úseku. Po posunutí začiatku preto budeme len ďalej pokračovať v hľadaní konca tohto úseku bez toho, aby sme začínali odznova.

Ostáva nám toto riešenie implementovať. Hlavný rozdiel oproti predchádzajúcim dvom riešeniam je, že prestaneme pouzívať premenné `prva` a `druha`. Miesto nich budeme mať dictionary (slovník resp. map pre C++ programátorov). V nej si budeme pamätať všetky korálky, ktoré sú aktuálne v nami vybranom úseku. Toto potrebujeme spraviť preto, lebo korálky už nebude iba pridávať, ale aj odoberať pri posúvaní začiatku. Potrebujeme teda vedieť zistiť, keď odoberieme poslednú korálku nejakej farby, aby sme vedeli, že môžeme pridávať nové korálky.

V našom slovníku si budeme pre každú farbu korálok pamätať, koľko daných korálok sa v našom úseku nachádza. Naviac, ak z danej farby nie je v úseku žiadna korálka, túto hodnotu v slovníku nebudeme mať. Vďaka tomu sa budeme môcť pýtať na veľkosť tohto slovníka (či je menší ako 3), teda počet farieb v aktuálnom úseku. Pri každom posune začiatku odstránime začiatočnú korálku zo slovníka a následne budeme vo while cykle zisťovať, nakoľko vieme náš úsek predĺžiť. Pri predlžovaní budeme samozrejme korálky do slovníka pridávať.

Aká je časová zložitosť takéhoto riešenia? Predstavme si dva indexy – začiatok a koniec úseku – ktoré v našom algoritme posúvame. Index začiatku posunieme práve n krát, raz pre každý začiatok. A index konca nemôže predbehnúť index začiatka (nezabúdajme, že sme na kruhu), preto ho môžeme posunúť dokopy najviac $2n$ krát. To vedie k celkovej zložitosti $O(n)$. Všimnite si ešte v riešení, že namiesto toho, aby sme pracovali s kruhovým náhrdelníkom, tak si náhrdelník nakopírujeme dvakrát za seba. Takéto predĺženie sa chová podobne ako keby sme boli na kruhu, akurát sa nám s tým lepšie pracuje.

Listing programu (Python)

```
n = int(input())

koralky = list(map(int, input().split()))
koralky += koralky

maxi = 1
od = -1
it = -1
aktualne = {}
for i in range(n):
    if i > 0:
        aktualne[koralky[i - 1]] -= 1;
        if aktualne[koralky[i - 1]] < 1:
            aktualne.pop(koralky[i - 1]);
    while it < i + n and len(aktualne) < 3:
        it += 1
        if not koralky[it] in aktualne:
            aktualne[koralky[it]] = 0
        aktualne[koralky[it]] += 1
    maxi = max(maxi, it - i)
    if maxi == it - i:
        od = i
```

```
print(maxi)
print(od + 1, n if (od + maxi) % n == 0 else ((od + maxi) % n))
```