A-III-1 ... a princeznú za ženu

Predstavme si, že máme nejaké optimálne riešenie, teda nejakú konkrétnu priamku. Ak na nej neleží žiaden zo zadaných bodov, sú len dve možnosti. Prvá je, že všetky body sú vybrané. Túto možnosť v riešení ošetríme ako špeciálny prípad.
Druhá je, že nejaké body vybrané nie sú. V takomto prípade musí medzi nevybranými bodmi byť aspoň jedna baňa – ak by tam boli samé krčmy, vedeli by sme priamku posunúť a vybrať ich a tak zlepšiť doterajšie riešenie. Predstavme si teraz, že našu priamku začneme posúvať (v smere kolmom na ňu a preč od vybranej polroviny) až kým nenarazí na prvú baňu. Táto nová priamka definuje aspoň tak dobré riešenie ako tá predchádzajúca, a teda je tiež optimálna.
Naše riešenie teda bude vyzerať nasledovne: Ak $n = 1$ alebo sú všetky body krčmy, ľahko nájdeme optimálne riešenie a skončíme. Vo zvyšných prípadoch si najskôr zistíme, ako dobré riešenie je zobrať všetky body a potom nájdeme najlepšiu hraničnú priamku prechádzajúcu nejakou baňou. Optimálnym riešením bude lepšia z týchto dvoch možností.

Majme konkrétnu baňu B. Chceme nájsť najlepšiu z priamok prechádzajúcich cez B.
Posuňme si súradnicovú sústavu tak, aby bod B bol jej začiatkom. Pre každý iný bod X teraz určíme α_X: veľkosť uhlu od kladnej poloosi osi x proti smeru ručičiek po polpriamku BX. Všetky α_X sú z intervalu $[0, 2\pi)$. Uvažujme polrovinu, ktorej hranica prechádza bodom B a má smer, ktorý s kladnou poloosou osi x zviera uhol β. Kedy bod X leží vo vybranej polrovine? Vtedy, keď (počítajúc modulo 2π) uhol α_X leží v intervale $(\beta, \beta+\pi)$. Inými slovami, keď budeme postupne spojito meniť β od 0 po 2π, bod X prestane byť vo výbere tesne pred $\beta = \alpha_X$ a začne byť vo výbere tesne po $\beta = (\alpha_X + \pi) \bmod 2\pi$. Ak ide o krčmu, jej pridanie do výberu nám zvýši profit a odstránenie z výberu nám ho zníži. Bane robia presný opak.
Teraz teda vieme postupne simulovať otáčanie našej priamky okolo B. Dokopy takto vyskúšame $4(n-1)$ rôznych možností: $2(n-1)$ takých, kde priamka prechádza cez druhý daný bod, a $2(n-1)$ takých, kde priamka vedie pomedzi dva z nich.
Presnejšie to bude celé vyzerať nasledovne: Najskôr si spočítame, aký profit zodpovedá priamke s $\beta = 0$. Potom si usporiadame všetkých $2(n-1)$ vyššie spomínaných uhlov podľa veľkosti a ku každému z nich si poznačíme, ako sa v jeho okolí mení hodnota optimálneho riešenia. Potom vieme každé otočenie priamky do ďalšej zaujímavej polohy odsimulovať v konštantnom čase.
Pre konkrétny bod B takto vyskúšame všetky možné deliace priamky v čase $O(n \log n)$, čo nám dáva celkovú časovú zložitosť $O(n^2 \log n)$.

Na záver poznamenáme, že vyššie popísaný postup vieme implementovať úplne exaktne, len použitím celočíselných premenných. Totiž nepotrebujeme poznať presné hodnoty uhlov, len ich relatívne poradie. Namiesto uhlov si teda budeme pamätať vektory a v rámci intervalu $[0, 2\pi)$ ich potom podľa smeru usporiadame podľa polroviny, v ktorej ležia, a v rámci nej podľa znamienka vektorového súčinu.

A-III-2 Podpostupnosti

Priamočiary algoritmus, ktorým vieme jednorazovo skontrolovať, či je reťazec Z podpostupnosťou reťazca T, má časovú zložitosť $O(z + t)$ – presnejšie, keďže pre $z > t$ rovno vieme, že odpoveď je „nie“, môžeme odhad časovej zložitosti zjednodušiť na $O(t)$.
Spomedzi všetkých možných spôsobov, ako vyrobiť Z z T, budeme hľadať ten „najľavejší“. V premennej *stav* si budeme pamätať, koľko znakov Z sme už vyrobili. Postupne čítame T. Ak vidíme znak, ktorý nechceme, zmažeme ho, a ak vidíme znak, ktorý práve potrebujeme pridať do Z, necháme ho a zvýšime premennú *stav*.

```
def je_podpostupnost(T,Z):
    stav = 0
    for c in T:
        if Z[stav] == c:
            stav += 1
            if stav == len(Z): return True
    return False
```

Ak po každej zmene spustíme tento algoritmus, dostaneme časovú zložitosť $O(nt)$. Ako ju vieme zlepšiť?

Asi najľahší spôsob, ako vyššie popísaný algoritmus urýchliť, je predstaviť si ho v trochu inej podobe. To, čo v ňom robíme, vieme ekvivalentne popísať nasledovne: Pozri sa na text T. Nájdi v ňom prvý výskyt prvého znaku Z. Od tej pozície choď ďalej a nájdi prvý výskyt druhého znaku Z. Odtiaľ ďalej hľadáme tretí, atď.
Toto hľadanie vieme urýchliť použitím vhodnej dátovej štruktúry.
Jedno pomerne ľahko implementovateľné riešenie vyzerá nasledovne: pre každé písmeno abecedy si budeme pamätať jeden vyvažovaný binárny vyhľadávací strom (BST), v ktorom budú uložené indexy všetkých výskytov tohto písmena. Keďže konkrétny strom bude mať vždy nanajvýš t prvkov, operáciu „nájdi prvý výskyt tohto písmena ktorého index je väčší ako i“ vieme spraviť v čase $O(\log t)$.
Inicializácia: Pre každé písmeno zvlášť si v čase $O(s+t)$ zozbierame usporiadaný zoznam jeho indexov v zadanom texte. Z usporiadaného zoznamu vieme v čase priamo úmernom jeho dĺžke vyrobiť vyvážený BST, takže výroba stromov nám tiež zaberie len čas $O(t)$.
Vykonanie algoritmu: z-krát potrebujeme nájsť nasledujúci výskyt nejakého písmena, dokopy teda vieme za pomoci našich BST v čase $O(z \log t)$ overiť, či aktuálny text T obsahuje podpostupnosť Z.
Zmena textu: keď zmeníme znak $T[i]$ z x na y, musíme zo stromu pre x vyhodiť index i a naopak ho pridať do stromu pre y. Obe tieto operácie vieme spraviť v čase $O(\log t)$, čo je zanedbateľné oproti následnému spúšťaniu algoritmu.
Celková časová zložitosť tohto riešenia je teda $O(s + t + nz \log t)$. Pamäťová zložitosť je $O(s + z + t)$.

Listing programu (C++)

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

const int s = 26;

void check(const string &Z, const vector<set<int> > &indexy) {
    int kde = 0;
    for (char c : Z) {
        auto it = indexy[c-'a'].upper_bound(kde-1);
        if (it == indexy[c-'a'].end()) { cout << "nie\n"; return; }
        kde = *it + 1;
    }
    cout << "ano\n";
}

int main() {
    string T, Z;
    cin >> T >> Z;

    vector< vector<int> > uvodne_indexy(s);
    for (int i=0; i<int(T.size()); ++i) uvodne_indexy[ T[i]-'a' ].push_back(i);

    vector< set<int> > indexy(s);
    for (int i=0; i<s; ++i) indexy[i] = set<int>( uvodne_indexy[i].begin(), uvodne_indexy[i].end() );

    check(Z, indexy);

    int n;
    cin >> n;
    while (n--) {
        int i;
        string x;
        cin >> i >> x;
        indexy[ T[i]-'a' ].erase(i);
        T[i] = x[0];
        indexy[ T[i]-'a' ].insert(i);
        check(Z, indexy);
    }
}
```

Poznámky na záver:
Iné dobré riešenie sa dá založiť na myšlienke, že si nad textom T postavíme intervalový strom. Každý vrchol tohto stromu predstavuje nejaký úsek textu T. V každom vrchole si budeme pamätať informáciu o tom, akú hodnotu bude mať premenná *stav* na konci spracovania jeho úseku v závislosti od toho, akú ju mala na začiatku.

Existujú aj riešenia, ktoré sú ešte o čosi efektívnejšie ako to, ktoré sme si vyššie popísali. Dá sa totiž využiť pozorovanie, že údaje, ktoré sme ukladali do BST, nie sú všeobecné, ale ide o prirodzené čísla z malého rozsahu. Namiesto BST môžeme potom použiť tzv. Van Emde Boasov strom, ktorý vie všetky potrebné operácie robiť v čase $O(\log \log t)$. Viac detailov o tomto strome nájdete napr. v riešeniach domáceho kola 26. ročníka OI.

A-III-3 Online algoritmy 3

Ak hra trvá n kôl, optimálne hrajúci jasnovidec bude mať na konci hry nanajvýš $n + 2$ bodov.
Nech by sme zbierali kartičky podľa akejkoľvek stratégie, nazbierame ich priebežne n, čiže z niektorého druhu určite zoberieme aspoň $n/5$ kartičiek. Na konci hry sa teda nedá mať menšie skóre ako $n/5 + 2$.
Keď si vyjadríme pomer optimálneho a nášho riešenia, dostávame, že platí $\frac{n+2}{(n/5)+2} = \frac{5n+10}{n+10} < 5$.
Každá možná stratégia je teda dokonca 5-kompetitívna.

Prvá 2-kompetitívna stratégia

Optimálna stratégia hráča, ktorý pozná budúcnosť, je zjavná: stačí sa pozrieť, ktorú kartičku mu súper koľkokrát dá na výber, zvoliť si tú, ktorá mu bude ukázaná najviackrát, a zakaždým ju zobrať.
Základná myšlienka 2-kompetitívnej stratégie bude jednoduchá: budeme hrať tak, aby sme kariet *každého* typu (zhruba) polovicu zobrali a polovicu odmietli. Ak sa nám toto podarí zabezpečiť, tak na konci hry bude platiť, že máme aspoň (zhruba) polovicu optimálneho skóre – lebo budeme mať aspoň polovicu kariet toho typu, ktorý si vybral optimálny hráč. Dá sa to ale, a ak áno, ako?

Definujme, že *balans* typu karty je dvakrát počet zobraných mínus počet ukázaných kariet toho typu. Balans nula teda znamená, že sme zobrali presne polovicu možného počtu kariet tohto typu. Záporný balans znamená, že ich máme menej ako polovicu, a teda by sme radi ešte nejaké vzali.
Naša stratégia bude nasledovná: Vždy, keď nám súper ukáže dve karty, zoberieme tú, ktorá má menší balans. V prípade rovnosti zoberieme ľubovoľnú z nich.
Nižšie si ukážeme, že balans žiadneho typu nikdy neklesne pod -2. Z toho už bude vyplývať 2-kompetitívnosť našej stratégie. Totiž nech si optimálne riešenie vybralo typ karty, ktorého bolo ukázaných n kusov. Keďže každý typ karty má balans -2 alebo viac, znamená to, že my sme zobrali aspoň $(n - 2)/2$ kariet tohto typu. Dokopy je teda optimálne skóre presne $n + 2$ a naše skóre aspoň $(n - 2)/2 + 2 = (n + 2)/2$, čiže sme nahrali aspoň polovicu optimálneho skóre.
Dôkaz tvrdenia o minimálnom balanse môžeme spraviť rozborom všetkých prípadov. Ľahko overíme, že pätica balansov (pričom nezáleží na ich poradí) môže nadobúdať len nasledovné hodnoty: $(0, 0, 0, 0, 0)$, $(-1, 0, 0, 0, 1)$, $(-1, -1, 0, 1, 1)$, $(-1, -1, 0, 0, 2)$, $(-1, -1, -1, 1, 2)$, $(-2, 0, 0, 1, 1)$, $(-2, 0, 0, 0, 2)$, $(-2, -1, 1, 1, 1)$, $(-2, -1, 0, 1, 2)$.
Totiž nech v ľubovoľnej z týchto situácií nám súper ukáže ľubovoľnú ďalšiu dvojicu kariet, vždy tým vznikne opäť jedna z týchto situácií.

Iný dôkaz: Všimnime si, že súčet všetkých balansov je vždy nula, lebo na začiatku sú všetky nulové a v každom kole jeden stúpne a jeden klesne. Ukážeme, že sa nedá vyrobiť balans -3 ani $+3$, lebo keď skúsime vyrobiť hociktorý jeden z nich, zistíme, že najskôr potrebujeme vyrobiť ten druhý. Nový balans $+x$ (resp. $-x$) môže vzniknúť len tak, že nám súper ukáže dve karty, ktoré majú obe balans presne o jedna menší (resp. väčší). Preto ak napríklad chceme vyrobiť balans -3, musíme v nejakom okamihu mať typy kariet s balansom -2, -1, 0, 0. Ale potom posledný typ kariet už musí mať balans $+3$, aby sedel súčet nula.

Druhá 2-kompetitívna stratégia

Iná 2-kompetitívna stratégia vyzerá nasledovne: Existuje 10 dvojíc typov rôznych kariet. Označme si ich AB, BC, CD, DE, EA, AC, CE, EB, BD a DA. Každú dvojicu si budeme všímať samostatne. Keď nám ju súper ukáže prvýkrát, vyberieme si prvú kartu v nej, druhýkrát druhú, tretíkrát znova prvú, a tak ďalej.
Bez ujmy na všeobecnosti predpokladajme, že najčastejšia karta bola napr. karta E. Nech bolo dokopy ukázaných a dvojíc EA, b dvojíc EB, c dvojíc CE a d dvojíc DE. Optimálne hrajúci hráč nahral skóre $2 + a + b + c + d$. My máme $2 + \lceil a/2 \rceil + \lceil b/2 \rceil + \lfloor c/2 \rfloor + \lfloor d/2 \rfloor$ kariet typu E. No a keďže $\lceil x/2 \rceil \geq x/2$ a $\lfloor x/2 \rfloor \geq x/2 - 1/2$, platí $2 + \lceil a/2 \rceil + \lceil b/2 \rceil + \lfloor c/2 \rfloor + \lfloor d/2 \rfloor \geq 1 + a/2 + b/2 + c/2 + d/2$, a teda sme zaručene nahrali aspoň polovicu optimálneho skóre.
Všimnite si, že dvojice kariet sme si označili tak, aby každý typ karty bol dvakrát prvý a dvakrát druhý. Vďaka tomu nám potom vo vyššie uvedenom vzorci pre každý typ karty vyšli dve horné a dve dolné celé časti. Keby sme si na toto nedali pozor a pre niektorý typ karty dostali viac ako dve celé časti, tvrdenie o 2-kompetitívnosti by už neplatilo.

Neexistencia lepšej stratégie

Uvažujme súpera, ktorý hrá túto hru nasledovne: Na začiatku si zvolí číslo n a v prvých $2n$ kolách nám ponúkne vždy dvojicu AB. Predpokladajme, že sme zobrali $n+a$ kariet A a $n-a$ kariet B, pričom bez ujmy na všeobecnosti je a nezáporné. Súper potom $2a$-krát ponúkne dvojicu BC a hru ukončí.
Naša stratégia vie nahrať nanajvýš skóre $n + a + 2$, lebo vieme zobrať nanajvýš $n + a$ kariet A, nanajvýš $(n - a) + 2a = n + a$ kariet B a nanajvýš $2a \leq n + a$ kariet C. Optimálna stratégia by však úplne od začiatku brala všetky karty B a nahrala skóre $2n + 2a + 2$.
Vidíme, že platí nasledovná nerovnosť:

$$\frac{2n + 2a + 2}{n + a + 2} = 2 - \frac{2}{n + a + 2} > 2 - \frac{2}{n}$$

S rastúcim n preto pomer medzi optimálnym a najlepším naším skóre konverguje ku dvom. Nemôže teda existovať žiadna stratégia, ktorá by bola c-kompetitívna pre nejaké $c < 2$. Ku každému takému c by totiž súper vedel zvoliť dostatočne veľké n, pre ktoré nás už donúti vyrobiť riešenie, ktoré je horšie ako optimálne deleno c.

TRIDSIATY ŠTVRTÝ ROČNÍK OLYMPIÁDY V INFORMATIKE

Príprava úloh: Michal Anderle, Bui Truc Lam, Michal Forišek, Samuel Sládek
Recenzia: Michal Forišek
Slovenská komisia Olympiády v informatike
Vydal: IUVENTA – Slovenský inštitút mládeže, Bratislava 2019