P-III-1 Půl království

Pokud v hledané polorovině neleží všechny zadané body, můžeme bez újmy na obecnosti předpokládat, že její hraniční přímka prochází jedním z těchto bodů (jinak ji můžeme posunovat směrem pryč od Honzovy poloroviny tak dlouho, dokud na některý z bodů nenarazí). Budeme tedy postupně zkoušet všechny body b a hledat nejlepší polorovinu, jejíž hraniční přímka tímto bodem prochází.

Pro každý další zadaný bod t si určeme úhel α_t (v radiánech), který svírá úsečka bt oproti polopřímce vycházející z b vodorovně doprava. Uvažme polorovinu, jejíž hraniční přímka prochází bodem b a svírá s polopřímkou vycházející z b vodorovně doprava úhel β. Pak t leží v této polorovině právě když buď α_t nebo $\alpha_t + 2\pi$ leží v intervalu $(\beta, \beta+\pi)$. Úlohu jsme si tedy převedli na následující: Máme $2n$ navzájem různých reálných čísel v intervalu $\langle 0, 4\pi)$ a ke každému z nich máme přiřazeno $+1$ nebo -1. Chceme najít interval $(\beta, \beta + \pi)$ takový, že $0 \leq \beta < 2\pi$ a součet čísel přiřazených k bodům v tomto intervalu je největší možný.

Nejprve si určíme, které body leží v intervalu $(0, \pi)$, a sečteme jim přiřazené hodnoty. Nyní budeme interval postupně posouvat doprava; pokaždé, když jeho pravý konec přejde nějaký bod, připočítáme příslušnou hodnotu, a když jeho levý konec přejde jeden z bodů, příslušnou hodnotu odečteme. Nakonec vrátíme maximum z těchto průběžných součtů. Samozřejmě si stačí simulaci posunování provést pouze v okamžicích, kdy se něco mění, tj. jeden z konců intervalu přechází přes bod. V každém kroku tedy potřebujeme rozhodnout, co se stane dřív: narazí pravý konec na bod za intervalem, nebo levý konec na bod uvnitř intervalu? To je snadné, jestliže si body nejprve setřídíme; pak stačí vždy porovnávat první bod v setříděném seznamu, který je uvnitř intervalu, s prvním bodem v seznamu, který je za koncem intervalu.

Časová složitost nalezení nejlepšího intervalu je dominována složitostí třídění, tedy $\mathcal{O}(n \log n)$. Jelikož nejlepší interval hledáme pro každý bod b, celková časová složitost je $\mathcal{O}(n^2 \log n)$. Paměťová složitost je $\mathcal{O}(n)$.

Poznamenejme ještě, že výše popsaný postup, kde si počítáme úhly, je potenciálně problematický kvůli zaokrouhlovacím chybám. Tomu je samozřejmě možné se vyhnout, jelikož nepotřebujeme znát hodnoty úhlů, ale jen je umět porovnávat, tj. dokázat pro dvě úsečky se stejným počátkem rozhodnout, která z nich svírá s s polopřímkou vycházející z jejich počátku vodorovně doprava větší úhel. To lze provést pomocí vektorového součinu příslušných vektorů, jehož znaménko nám říká, zda je druhý z vektorů v levé či pravé polorovině, jejíž hranice obsahuje první z vektorů. Pro přehlednost toto vylepšení v autorském řešení neimplementujeme.

```
#include <cstdio>
#include <cmath>
#include <vector>
```

```
#include <algorithm>
using namespace std;

struct osoba
{
  int x, y, p;
  osoba(int _x, int _y, int _p) : x(_x), y(_y), p(_p) {}
};

struct bod
{
  double v;
  int p;

  bod(double _v, int _p) : v(_v), p(_p) {}
  bool operator<(const bod &b) const { return v < b.v; }
};

static vector<osoba> kralovstvi;
static int n;

/* Určí nejlepší polorovinu procházející B-tým z bodů, do UHEL uloží úhel její
   hraniční přímky, a vrátí rozdíl počtu přátel a nepřátel v této polorovině. */
static int polorovina_skrz(int b, double &uhel)
{
  vector<bod> body;
  const osoba &bo = kralovstvi[b];

  for (int i = 0; i < n; i++)
    if (i != b)
      {
        const osoba &io = kralovstvi[i];
        double uhel = atan2(io.y - bo.y, io.x - bo.x);
        if (uhel < 0)
          uhel += 2 * M_PI;
        body.emplace_back(uhel, io.p);
      }
  sort(body.begin(), body.end());
  for (int i = 0; i < n - 1; i++)
    body.emplace_back(body[i].v + 2 * M_PI, body[i].p);
  /* Zarážky */
  body.emplace_back(4 * M_PI, 0);
  body.emplace_back(5 * M_PI, 0);

  vector<bod>::iterator levy = body.begin();
  if (levy->v == 0)
    ++levy;
  vector<bod>::iterator pravy = levy;
  int rozdil = 0;
  for (; pravy->v < M_PI; ++pravy)
    rozdil += pravy->p;
  double uh = 0;

  uhel = 0;
  int nejlepsi_rozdil = rozdil;

  while (true)
    {
      if (levy != pravy && levy->v < pravy->v - M_PI)
```

```
        {
          rozdil -= levy->p;
          uh = levy->v;
          ++levy;
        }
      else
        {
          rozdil += pravy->p;
          double puh = pravy->v;
          uh = puh - M_PI;
          ++pravy;

          /* Aby se pravý bod dostal do intervalu, musíme interval ještě kousek
             posunout; ale ne tolik, aby z něj vypadl první bod uvnitř nebo aby
             se dovnitř dostal následující bod. */
          uh += min(levy->v - uh, pravy->v - puh) / 2;
        }

      if (uh >= 2 * M_PI)
        break;

      if (rozdil > nejlepsi_rozdil)
        {
          nejlepsi_rozdil = rozdil;
          uhel = uh;
        }
    }

  return nejlepsi_rozdil;
}

int main(void)
{
  int nejlepsi_rozdil = 0;
  /* Hranice zatím nejlepší poloroviny prochází bodem (xn,yn) pod úhlem nej_uhel. */
  double xn = 0, yn = 0, nej_uhel = 0;

  scanf("%d", &n);
  for (int i = 0; i < n; i++)
    {
      int x, y;
      char c;
      scanf("%d%d %c", &x, &y, &c);
      int p = (c == 'P' ? 1 : -1);
      kralovstvi.emplace_back(x, y, p);

      /* Hodnoty nejlepsi_rozdil a yn inicializujeme dle poloroviny
         obsahující všechny body. */
      nejlepsi_rozdil += p;
      if (y <= yn)
        yn = y - 1;
    }

  for (int i = 0; i < n; i++)
    {
      double uhel;
      int rozdil = polorovina_skrz(i, uhel);
      if (rozdil > nejlepsi_rozdil)
        {
```

```
          nejlepsi_rozdil = rozdil;
          xn = kralovstvi[i].x;
          yn = kralovstvi[i].y;
          nej_uhel = uhel;
        }
    }
  printf("%.2f %.2f %.2f %.2f\n", xn, yn, xn + cos(nej_uhel), yn + sin(nej_uhel));
  return 0;
}
```

P-III-2 Pomluvy

Nejprve si rozmysleme, jak poznat, zda se heslo h dá dostat z textu smazáním nějakých písmen. K tomu stačí v textu najít první výskyt prvního písmene z h, za tímto výskytem najít první výskyt druhého písmene z h, atd., dokud buď nenajdeme všechna písmena z h (a ostatní můžeme smazat), nebo nedojdeme na konec textu (v kterémžto případě se h v textu najít nedá).

Pro efektivní implementaci stačí použít datovou strukturu, která nám pro zadané písmeno a pozici v textu umožní rychle najít nejbližší výskyt daného písmene za touto pozicí a také umožní v textu měnit písmena. Jednou z možností je si pro každé písmeno ve vyvažovaném vyhledávacím stromu* udržovat seznam pozic, na kterých se toto písmeno nachází. Hledáme-li výskyt písmene za danou pozicí, v příslušném stromu najdeme nejmenší prvek větší než tato pozice (to lze v čase $\mathcal{O}(\log m)$). Chceme-li změnit písmeno na pozici p z a na b, pak ze stromu pro a smažeme p a do stromu pro b vložíme p; to také zabere čas $\mathcal{O}(\log m)$. Vyhledávací strom pro předem danou setříděnou množinu hodnot se dá postavit v lineárním čase. Celková časová složitost tedy bude $\mathcal{O}(m)$ na inicializaci stromů, $\mathcal{O}(\log m)$ na úpravy stromů v každém dotazu a $\mathcal{O}(|h| \log m)$ na každé ověření výskytu h v textu, celkem tedy $\mathcal{O}(m + n|h| \log m)$. Paměťová složitost je $\mathcal{O}(m)$, což odpovídá součtu velikostí všech reprezentovaných množin.

Poznamenejme, že tento postup lze ještě trochu vylepšit, uvědomíme-li si, že neudržujeme seznamy libovolných prvků, ale přirozených čísel v intervalu od 1 do m. Pro tuto situaci existuje efektivnější datová struktura: van Emde Boasův strom,** který umožňuje hledání i změny provádět v čase $\mathcal{O}(\log \log m)$, a s jeho pomocí tedy lze časovou složitost zlepšit na $\mathcal{O}(m + n|h| \log \log m)$.

```
#include <cstdio>
#include <cstring>
#include <set>
#include <list>
using namespace std;

#define POCET_PISMEN ('z' - 'a' + 1)
static set<int> *pozice[POCET_PISMEN];
```

* Viz *https://ksp.mff.cuni.cz/kucharky/vyhledavaci-stromy/*

** Viz *http://mo.mff.cuni.cz/p/60/reseni-1.html*, úloha P-I-3.

```
static bool najdi_heslo(const char *h)
{
  int a = -1;

  for (const char *p = h; *p; p++)
    {
      set<int> *pos = pozice[*p - 'a'];
      set<int>::iterator n = pos->upper_bound(a);
      if (n == pos->end())
        return false;
      a = *n;
    }

  return true;
}

int main(void)
{
  char *h, *t;
  scanf("%ms%ms", &h, &t);
  int m = strlen(t);

  list<int> pocatecni_pozice[POCET_PISMEN];
  for (int i = 0; i < m; i++)
    pocatecni_pozice[t[i] - 'a'].push_back(i);
  for (int p = 0; p < POCET_PISMEN; p++)
    pozice[p] = new set<int>(pocatecni_pozice[p].begin(), pocatecni_pozice[p].end());

  int n;
  scanf("%d", &n);

  for (int i = 0; i < n; i++)
    {
      int p;
      char c;

      scanf("%d %c", &p, &c);
      p--;
      pozice[t[p] - 'a']->erase(p);
      t[p] = c;
      pozice[c - 'a']->insert(p);

      printf("%s\n", najdi_heslo(h) ? "ano" : "ne");
    }

  return 0;
}
```

P-III-3 Kartičky

Nejprve si povšimněme, že počet bodů podvodníka je přesně roven 2 plus maximum z počtů výskytů jednotlivých barev na soupeřem nabízených kartičkách; podvodník si může jednoduše vždy kdy to jde brát kartičku nejčastěji se vyskytující barvy.

Použijme následující strategii. Představme si, že z barev A, ..., E vytvoříme cyklus. Z dvojice barev prohlásíme za *důležitější* tu, pro kterou platí, že druhá z těchto barev se po ní na cyklu vyskytuje o jednu nebo dvě pozice dále. Tedy ve dvojici A B je důležitější A, ve dvojici E B je důležitější E, atd. Pro každou dvojici

barev si kartičky budeme brát nastřídačku: když nám tuto dvojici soupeř nabídne poprvé, vybereme si důležitější z barev, podruhé si vybereme méně důležitou, pak opět důležitější, atd.

Řekněme, že nejčastěji vyskytující se barva je A (analýza pro ostatní barvy je symetrická). Nechť nám během hry nabídne soupeř b-krát A B, c-krát A C, d-krát A D a e-krát A E. Pak skóre podvodníka je $b+c+d+e+2$. Oproti tomu výše popsaná strategie zaručuje, že my budeme mít alespoň $\lceil b/2 \rceil + \lceil c/2 \rceil + \lfloor d/2 \rfloor + \lfloor e/2 \rfloor + 2 \geq b/2 + c/2 + (d-1)/2 + (e-1)/2 + 2 = (b+c+d+e+2)/2$ kartiček barvy A, a naše skóre tedy bude alespoň polovina skóre podvodníka.

Uvažujme nyní libovolnou strategii. Soupeř si vybere libovolné přirozené číslo n a začne tím, že $2n$-krát nabídne A B. Nechť si uvažovaná strategie $(n+a)$-krát zvolí A a $(n-a)$-krát B, kde bez újmy na obecnosti $a \geq 0$ a zjevně $a \leq n$. Soupeř poté $2a$-krát nabídne B C. Ať už se pro tyto nabídky strategie rozhodne libovolně, A bude na konci zvoleno $(n+a)$-krát, B bude zvoleno nejvýše $(n-a+2a) = (n+a)$-krát a C bude zvoleno nejvýše $2a \leq (n+a)$-krát. Naše skóre dle uvažované strategie bude tedy $n+a+2$. Oproti tomu podvodník může v každém kole brát B a jeho skóre tedy bude $2n+2a+2$. Uvažovaná strategie tedy získá pouze

$$\frac{n+a+2}{2n+2a+2} = \frac{1}{2} + \frac{1}{2n+2a+2} < \frac{1}{2} + \frac{1}{2n}$$

krát tolik bodů, co podvodník. Pro dostatečně velké n bude tento poměr menší než jakékoliv pevné $r > 1/2$.