62. ročník Matematické olympiády – 2012/2013

Řešení úloh ústředního kola kategorie P – 2. soutěžní den

P-III-4 Volby

Obdobně jako v domácím kole si špióny přečíslujeme tak, aby každý z nich znal nejvýše 6 špiónů s vyššími čísly – tj. vybereme špióna znajícího nejvýše 6 jiných špiónů, přiřadíme mu číslo 1 a odstraníme ho; dále vybereme špióna znajícího nejvýše 6 ze zbylých špiónů a přiřadíme mu číslo 2, a tento postup opakujeme, dokud všichni špióni nemají přiřazené číslo. Tento postup lze implementovat v čase $\mathcal{O}(n)$.

Pro špióna s a kandidáta k si jako $N_{s,k}$ označme počet špiónů s čísly menšími než s, kteří znají špióna s a volí kandidáta k. Jako v_s si označme kandidáta, kterého volí špión s. Počet bodů kandidáta k je tedy

$$B_k = \sum_{s:v_s=k} N_{s,k}.$$

V programu si budeme průběžně udržovat hodnoty $N_{s,k}$ a B_k pro všechny kombinace s a k; tyto hodnoty lze na začátku snadno určit v čase $\mathcal{O}(n)$ průchodem přes dvojice špiónů, kteří se vzájemně znají (připomeňme, že celkem je těchto dvojic nejvýše $3n$).

Co se stane, když špión s změní názor a začne volit kandidáta k', místo jeho současné volby k? Zřejmě se o jedna sníží hodnota $N_{x,k}$ a o jedna zvýší hodnota $N_{x,k'}$ pro všechny špióny x, kteří znají špióna s a jejichž číslo je větší než s; množinu těchto špiónů si označme jako X. Dle volby číslování špiónů máme $|X| \leq 6$, proto hodnoty $N_{x,k}$ a $N_{x,k'}$ lze opravit v konstantním čase. Hodnotu B_k opravíme tak, že od ní odečteme $N_{s,k}$ plus počet špiónů v X, kteří volí k. Obdobně, novou hodnotu $B_{k'}$ dostaneme tak, že k ní přičteme $N_{s,k'}$ plus počet špiónů v X, kteří volí k'. Tyto hodnoty tedy také lze přepočítat v konstantním čase.

Celkově tedy potřebujeme čas $\mathcal{O}(n)$ na inicializaci a $\mathcal{O}(1)$ na každý dotaz.

```
#include <vector>
#include "spioni.h"
#define DEGEN 6
#define KAND 5
using namespace std;

/* Počet špíónů */
static int n;

struct spion
{
  vector<int> zna;              /* Seznam známých špiónů */
  vector<int> zamnou;           /* Seznam těch, co jsou za mnou */
  int kand;                     /* Aktuálně volený kandidát */
  int pred_pro_kand[KAND];  /* Počet špiónů přede mnou, volících daného kandidáta */

  spion(void) : zna(), zamnou(), kand(0)
    {
```

```
      int i;
      for (i = 0; i < KAND; i++)
        pred_pro_kand[i] = 0;
    }
};

/* Seznam špiónů */
static vector<spion> sezsp;

/* Počet bodů pro daného kandidáta */
static int bodu[KAND];

/* Nalezne pořadí takové, aby každý znal nanejvýš DEGEN špiónů za ním */
static void urci_poradi()
{
  bool odebran[n];              /* Zda už byl špión odebrán */
  int znamych[n];               /* Počet zbývajících známých */
  vector<int> malo;             /* Seznam špiónů s nanevýš DEGEN známými */
  int i;

  for (i = 0; i < n; i++)
    {
      odebran[i] = false;
      znamych[i] = sezsp[i].zna.size();
      if (znamych[i] <= DEGEN)
        malo.push_back(i);
    }

  while (!malo.empty())
    {
      int aktualni = malo.back();
      malo.pop_back();

      odebran[aktualni] = true;
      vector<int>::iterator s = sezsp[aktualni].zna.begin();
      vector<int>::iterator se = sezsp[aktualni].zna.end();
      for (; s < se; ++s)
        if (!odebran[*s])
          {
            sezsp[aktualni].zamnou.push_back(*s);
            znamych[*s]--;
            if (znamych[*s] == DEGEN)
              malo.push_back(*s);
          }
    }
}

static void inicializuj_pocty()
{
  int s;
  vector <int>::iterator sous;

  for (s = 0; s < n; s++)
    for (sous = sezsp[s].zamnou.begin(); sous != sezsp[s].zamnou.end(); ++sous)
      sezsp[*sous].pred_pro_kand[sezsp[s].kand]++;

  for (s = 0; s < n; s++)
    bodu[sezsp[s].kand] += sezsp[s].pred_pro_kand[sezsp[s].kand];
}
```

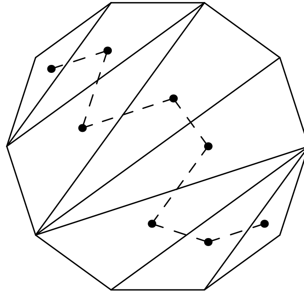
```
void spioni(int inpn, int m, int znaji_se[][2], int voli[])
{
  int i;

  n = inpn;
  sezsp.resize(n + 1);
  for (i = 0; i < n; i++)
    sezsp[i].kand = voli[i];
  for (i = 0; i < m; i++)
    {
      int s1 = znaji_se[i][0];
      int s2 = znaji_se[i][1];
      sezsp[s1].zna.push_back(s2);
      sezsp[s2].zna.push_back(s1);
    }
  urci_poradi();
  inicializuj_pocty();
}

void zmen_nazor(int spion, int voli)
{
  int puvodni = sezsp[spion].kand;
  int pro_puv = 0, pro_nov = 0;
  vector<int>::iterator sous;

  if (puvodni == voli)
    return;
  sezsp[spion].kand = voli;

  for (sous = sezsp[spion].zamnou.begin(); sous != sezsp[spion].zamnou.end(); ++sous)
    {
      int skand = sezsp[*sous].kand;

      if (skand == puvodni)
        pro_puv++;
      else if (skand == voli)
        pro_nov++;

      sezsp[*sous].pred_pro_kand[puvodni]--;
      sezsp[*sous].pred_pro_kand[voli]++;
    }

  bodu[puvodni] -= sezsp[spion].pred_pro_kand[puvodni] + pro_puv;
  bodu[voli] += sezsp[spion].pred_pro_kand[voli] + pro_nov;
}

void pocet_bodu(int pocet[KAND])
{
  int i;
  for (i = 0; i < KAND; i++)
    pocet[i] = bodu[i];
}
```

P-III-5 Gamesa

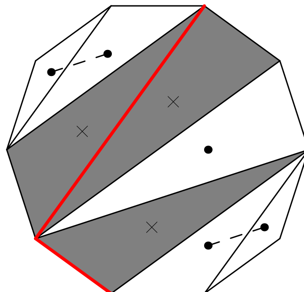
Nejprve si povšimněme, že když některý z hráčů obarví hranu trojúhelníka t, pak již nikdo nebude chtít barvit jeho další dvě hrany – kdyby např. Honzík jednu z nich obarvil, Mařenka okamžitě obarví třetí hranu trojúhelníka t a vyhraje. V optimální strategii tedy hra bude probíhat tak, že hráči střídavě barví hrany disjunktních trojúhelníků a hráč prohraje, jestliže je na tahu a všechny trojúhelníky již mají obarvenou hranu.

Když někdo obarví hranu na hranici mnohoúhelníka, vyřadí tak ze hry jeden trojúhelník, který s ní sousedí. Pokud obarví uhlopříčku, vyřadí oba sousedící trojúhelníky. Abychom se v aktuálním stavu hry lépe orientovali, zaveďme si následující reprezentaci herního plánu.

Dovnitř každého trojúhelníku t hracího plánu si dokresleme vrchol v_t a dva takovéto vrcholy v_t a $v_{t'}$ spojme čarou, právě když trojúhelníky t a t' sdílí hranu. Podle pravidel z každého vrcholu vychází nejvýše dvě čáry. Také nahlédněme, že ze dvou vrcholů vede jen jedna čára (ať už Mařenka rozdělila mnohoúhelník jakkoliv, dva trojúhelníky sdílí dvě hrany s hranicí mnohoúhelníka) a že po nakreslených čarách lze přejít mezi libovolnými dvěma trojúhelníky. Proto nakreslené čáry tvoří cestu procházející všemi vrcholy. Pro ilustraci viz následující obrázek.



Obarvíme-li nyní hranu mezi trojúhelníkem t a hranicí mnohoúhelníka, v této nové reprezentaci tomu odpovídá smazání vrcholu v_t a sousedících čar (cesta se nám tak rozpadne na dvě, pokud mazaný vrchol není na jejím konci). Obarvíme-li hranu mezi trojúhelníky t a t', pak musíme odebrat oba vrcholy v_t a $v_{t'}$. Následující obrázek ukazuje situaci po dvou takových tazích.



Hru si tedy můžeme přeformulovat takto: na začátku je herní plán cesta. Honzík a Mařenka se střídají na tahu a každý z nich může odebrat buď jeden vrchol, nebo dva sousedící vrcholy. Vyhrává ten, kdo smaže poslední vrchol. Nyní již snadno najdeme vyhrávající strategii pro Honzíka: ve svém prvním tahu smaže buď prostřední vrchol (má-li cesta lichý počet vrcholů) nebo dva prostřední vrcholy (má-li cesta sudý počet vrcholů). Tím cestu rozdělí na dvě stejně dlouhé. V dalších tazích bude pouze zrcadlit tahy Mařenky – pokud Mařenka smaže jeden či dva vrcholy v první cestě, Honzík smaže odpovídající vrchol či dvojici vrcholů v druhé cestě, a naopak. Takto se nemůže stát, že by po tahu Mařenky neměl jak hrát. Po počátečním předzpracování hracího plánu (v čase $\mathcal{O}(n)$) tak lze každý správný tah Honzíka určit v čase $\mathcal{O}(1)$.

```
#include <unordered_map>
#include <vector>
#include <cstdio>
#include "gamesa.h"
using namespace std;

struct hrana
{
  unsigned vs[2];

  hrana(void) { vs[0] = vs[1] = 0; }

  hrana(int v1, int v2)
    {
      if (v1 < v2)
        {
          vs[0] = v1;
          vs[1] = v2;
        }
      else
        {
          vs[0] = v2;
          vs[1] = v1;
        }
    }

  bool operator==(const hrana &s) const
    {
      return vs[0] == s.vs[0] && vs[1] == s.vs[1];
    }
};

static unsigned n;
namespace std {
template <> class hash< hrana > : public unary_function< hrana, size_t >
{
  public:
  size_t operator()(const hrana &h) const
    {
      return h.vs[0] * n + h.vs[1];
    }
};}

// Nezáporná čísla odpovídají vrcholům v cestě (od 0 do n - 3).
// Záporná odpovídají dvojici vrcholů -i - 1 a -i (kde i je od -1 do -(n-3)).
```

```
static unordered_map<hrana,int> vceste;
static unordered_map<hrana,bool> obarveny;
static vector<hrana> vrcholy;
static vector<bool> smazany;
static vector<hrana> dvojice;
static bool prvni_tah = true;
static int martah;

static unsigned pred(unsigned k)
{
  return k == 1 ? n : k - 1;
}

static unsigned nasl(unsigned k)
{
  return k == n ? 1 : k + 1;
}

void herni_plan(int inpn, int uhlopricky[][2])
{
  unsigned i, cv;
  int h, d;
  int deg[inpn + 1];

  n = inpn;
  vrcholy.resize(n);
  smazany.resize(n);
  dvojice.resize(n);

  for (i = 0; i < n - 2; i++)
    smazany[i] = false;

  for (i = 1; i <= n; i++)
    deg[i] = 0;

  for (i = 0; i < n - 3; i++)
    {
      deg[uhlopricky[i][0]]++;
      deg[uhlopricky[i][1]]++;
      vceste[hrana(uhlopricky[i][0], uhlopricky[i][1])] = 0;
    };
  for (i = 1; deg[i] > 0; i++)
    continue;

  h = nasl(i);
  d = pred(i);
  vceste[hrana(i, h)] = 0;
  vceste[hrana(i, d)] = 0;
  dvojice[0] = hrana(i, h);
  vrcholy[0] = hrana(i, d);

  for (cv = 1; cv < n - 3; cv++)
    {
      int nh = nasl(h), nd = pred(d);

      vceste[hrana(d, h)] = -cv;
      dvojice[cv] = hrana(d, h);
      if (vceste.count(hrana(d, nh)))
        {
          vceste[hrana(h, nh)] = cv;
```

```
          vrcholy[cv] = hrana(h, nh);
          h = nh;
        }
      else
        {
          vceste[hrana(d, nd)] = cv;
          vrcholy[cv] = hrana(d, nd);
          d = nd;
        }
    }

  if (n > 3)
    {
      i = nasl(h);
      vceste[hrana(d, h)] = -cv;
      dvojice[cv] = hrana(d, h);
      vceste[hrana(h, i)] = cv;
      vceste[hrana(d, i)] = cv;
      vrcholy[cv] = hrana(h, i);
      dvojice[cv + 1] = hrana(d, i);
    }
  else
    {
      vceste[hrana(d, h)] = 0;
      dvojice[1] = hrana(d, h);
    }

  for (i = 0; i < n - 2; i++)
    obarveny[vrcholy[i]] = false;
  for (i = 0; i < n - 1; i++)
    obarveny[dvojice[i]] = false;
}

static void zvitezit(int trojuhelnik, int hr[2])
{
  hrana h[3];
  int i;

  h[0] = vrcholy[trojuhelnik];
  h[1] = dvojice[trojuhelnik];
  h[2] = dvojice[trojuhelnik + 1];
  for (i = 0; obarveny[h[i]]; i++)
    continue;

  hr[0] = h[i].vs[0];
  hr[1] = h[i].vs[1];
}

static void tahni(int tah, int hr[2])
{
  hrana h = tah >= 0 ? vrcholy[tah] : dvojice[-tah];
  obarveny[h] = true;

  if (tah >= 0)
    smazany[tah] = true;
  else
    {
      smazany[-tah - 1] = true;
```

```
      smazany[-tah] = true;
    }
  hr[0] = h.vs[0];
  hr[1] = h.vs[1];
}

void tah_honzika(int hr[2])
{
  if (prvni_tah)
    {
      if (n % 2 == 0)
        tahni(-((n - 2) / 2), hr);
      else
        tahni((n - 3) / 2, hr);
      prvni_tah = false;
      return;
    }

  if (martah >= 0)
    {
      if (smazany[martah])
        {
          zvitezit(martah, hr);
          return;
        }

      smazany[martah] = true;
      tahni(n - 3 - martah, hr);
    }
  else
    {
      if (smazany[-martah - 1])
        {
          zvitezit(-martah - 1, hr);
          return;
        }
      if (smazany[-martah])
        {
          zvitezit(-martah, hr);
          return;
        }

      smazany[-martah - 1] = true;
      smazany[-martah] = true;
      tahni(-(n - 2) - martah, hr);
    }
}

void tah_marenky(int hr[2])
{
  hrana h(hr[0], hr[1]);
  martah = vceste[h];
  obarveny[h] = true;
}
```