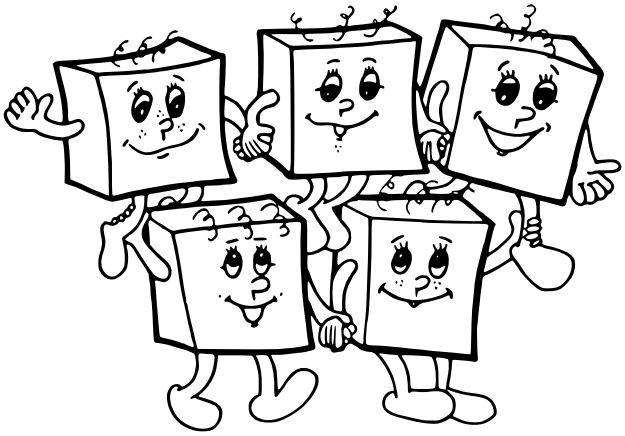
OLYMPIÁDA V INFORMATIKE

NA STREDNÝCH ŠKOLÁCH

http://oi.sk/



dvadsiaty piaty ročník
školský rok 2009/10

riešenia celoštátneho kola
kategória A

1. súťažný deň

A-III-1 Čokoláda je tu zas

Najskôr ukážeme riešenie s časovou zložitosťou $O(R^2S)$. Bude založené na jednoduchej myšlienke: vyskúšame všetky dvojice riadkov, a pre každú dvojicu v $O(S)$ spočítame všetky obdĺžniky, ktoré práve tam majú svoj horný a dolný riadok.

Keď sme si už zvolili horný a dolný riadok, máme pás políčok. Niektoré jeho stĺpce sú celé biele, tie môžeme použiť. A niektoré obsahujú aspoň jedno sivé políčko, a tie použiť nemôžeme.

Ak by sme vedeli, ktoré stĺpce použiť môžeme, a ktoré nie, dostávame vlastne jednorozmernú verziu pôvodnej úlohy: máme riadok s nulami a jednotkami a chceme spočítať počet úsekov, ktoré sú tvorené len jednotkami.

To spravíme tak, že pre každé miesto zistíme počet úsekov jednotiek, ktoré na tom mieste končia, a všetky tieto počty sčítame. Počet úsekov jednotiek, ktoré na danom mieste končia, je zjavne rovný počtu jednotiek, ktoré uvidíme, keď pôjdeme z daného miesta doľava po najbližšiu nulu (alebo začiatok). A tento počet si vieme priebežne počítať, ako spracúvame čísla v riadku – vždy, keď spracujeme jednotku, zvýšime si počítadlo, a vždy, keď spracujeme nulu, počítadlo vynulujeme.

Takto teda celý riadok spracujeme v čase lineárnom od jeho dĺžky, čiže $O(S)$.

Zostáva doriešiť, odkiaľ vezmeme informáciu o tom, ktoré stĺpce môžeme použiť a ktoré nie. Na to nám stačí spracúvať dvojice riadkov v systematickom poradí. Pre dvojicu (r_1, r_1) túto informáciu máme „zadarmo“ priamo vo vstupe. A keď pre dvojicu (r_1, r_2) vieme, ktoré stĺpce sa ešte dajú použiť, pre dvojicu $(r_1, r_2 + 1)$ túto informáciu ľahko zistíme – sú to tie stĺpce, ktoré sa dali použiť pre (r_1, r_2), a zároveň majú jednotku aj v riadku $r_2 + 1$.

Listing programu:

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

int R, S, A[5012][5012], zije[5012];

int main() {
  cin >> R >> S;
  for (int r=0; r<R; r++) for (int s=0; s<S; s++) cin >> A[r][s];
  long long result = 0;
  for (int r1=0; r1<R; r1++) {
    for (int s=0; s<S; s++) zije[s]=1;
    for (int r2=r1; r2<R; r2++) {
      for (int s=0; s<S; s++) zije[s] &= A[r2][s];
      int run=0;
      for (int s=0; s<S; s++) if (zije[s]) result += (++run); else run=0;
    }
  }
  cout << result << endl;
}
```

Vzorové riešenie

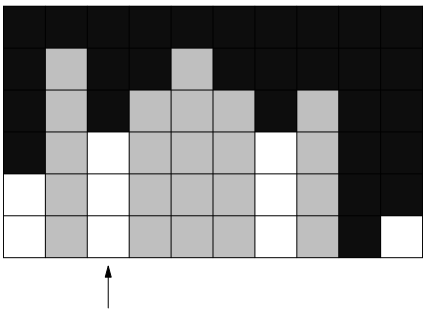
Teraz ukážeme jedno možné riešenie s optimálnou časovou zložitosťou $O(RS)$.

Naše riešenie bude založené na nasledujúcej základnej myšlienke: Postupne pre každý riadok spočítame všetky obdĺžniky, ktoré práve v ňom majú svoj dolný riadok.

Keď spracúvame nejaký riadok, budeme o každom jeho políčku potrebovať vedieť, akú má *výšku* – t. j. aká dlhá dohora idúca súvislá postupnosť jednotiek na ňom začína. Výšky pre nový riadok si vieme spočítať z výšok pre o jedno vyšší riadok rovnako, ako sme si počítali v predchádzajúcom riešení to, ktoré stĺpce ešte môžeme použiť.

Predstavme si, že teraz zoberieme všetkých S stĺpcov a utriedime ich od najvyššieho po najnižší. V tomto poradí ich teraz budeme spracúvať. Vždy, keď spracujeme stĺpec, zarátame všetky obdĺžniky, ktoré práve pribudli – teda tie, ktoré majú dolnú stranu na práve spracúvanom riadku, obsahujú aspoň jedno políčko tohto stĺpca, a celé ležia v už spracovaných stĺpcoch.

Zjavne takto každý obdĺžnik zarátame práve raz – vtedy, keď spracujeme posledný zo stĺpcov, v ktorých leží. Zostáva už len jediné – prísť na to, ako tieto počty obdĺžnikov efektívne určiť.



Situácia uprostred spracúvania riadku s výškami $2, 5, 3, 4, 5, 4, 3, 4, 0, 1$.
(Svetlou sivou sú už spracované stĺpce, šípkou je označený práve spracúvaný.)

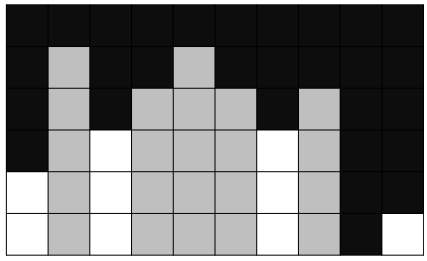
Keď spracúvame nejaký stĺpec, potrebujeme vedieť povedať počet jemu priradených obdĺžnikov. Keďže každý z nich má už pevne zvolenú spodnú stranu, je každý určený tromi hodnotami: výškou a tým, ako ďaleko doľava a ako ďaleko doprava od práve spracúvaného stĺpca siaha.

To, ako ďaleko doľava a doprava môže siahať, je samozrejme nanajvýš rovné počtu už spracovaných stĺpcov bezprostredne naľavo a napravo. Napríklad v situácii na obrázku by pri spracúvaní stĺpca označeného šípkou mohli obdĺžniky siahať až 1 políčko doľava a 3 doprava.

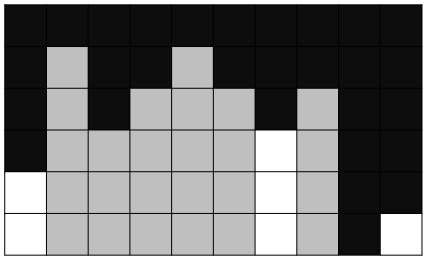
Uvedomme si teraz, že všetky už spracované stĺpce sú aspoň tak vysoké ako ten aktuálny. A teda každej prípustnej kombinácii hodnôt (výška, počet políčok doľava, počet políčok doprava) naozaj zodpovedá platný obdĺžnik. (Toto je dôvod, prečo stĺpce spracúvame práve usporiadané podľa výšky a nie v inom poradí.) A teda počet obdĺžnikov priradených aktuálnemu stĺpcu vieme zistiť jednoducho vynásobením týchto troch hodnôt.

Zostáva doriešiť, odkiaľ efektívne zistíme, ako ďaleko doľava a doprava môžu siahať obdĺžniky pre aktuálny stĺpec. To sa dá spraviť jednoducho. Všimnime si, že v každom okamihu tvoria už spracované stĺpce niekoľko súvislých úsekov. Ku každému súvislému úseku si budeme pamätať dve čísla: na jeho ľavom okraji ako ďaleko doprava, a na jeho pravom okraji ako ďaleko doľava siaha. Takto sa pri spracúvaní stĺpca v konštantnom čase dozvieme informácie, ktoré potrebujeme (sú uložene v jeho bezprostredných susedoch), a takisto po jeho spracovaní vieme tieto informácie v konštantnom čase upraviť – poznáme začiatok aj koniec úseku obsahujúceho práve spracovaný stĺpec, tak do jeho koncov zapíšeme jeho novú dĺžku.

Na nasledujúcich obrázkoch nájdete pamätané informácie pred spracovaním a po spracovaní šípkou označeného stĺpca z predchádzajúceho obrázku.



dolava		1			3		1		
doprava		1		3			1		



dolava						5		1	
doprava		5						1	

Ešte raz teda zhrnieme naše riešenie. Postupne pre každý z R riadkov spravíme nasledujúce veci: Najskôr v čase $O(S)$ spočítame výšky pre tento riadok. Potom utriedime všetky stĺpce podľa výšky – keďže výšky sú celé čísla od 0 po R, toto vieme spraviť v čase $O(R+S)$ napríklad priehradkovým triedením (bucket sort). Následne pre každý stĺpec v konštantnom čase zistíme počet jemu priradených obdĺžnikov, a tiež v konštantnom čase upravíme pamätané dĺžky súvislých úsekov. Aj táto fáza spracovania riadku teda beží v čase $O(S)$.

Celková časová zložitosť je teda $O(R(R+S))$. A keďže môžeme predpokladať, že $R \leq S$ (inak vstup transponujeme, čo počet obdĺžnikov nezmení), je toto rovné $O(RS)$. Táto zložitosť je zjavne optimálna, keďže musíme načítať takto veľký vstup.

Listing programu:

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

int R, S;
int A[5012][5012], vedierka[5012][5012];
int vyska[5012], dolava[5012], doprava[5012], pocty[5012];

int main() {
  scanf("%d %d ",&R,&S);
  for (int r=1; r<=R; r++) for (int s=1; s<=S; s++) scanf("%d",&A[r][s]);
  long long result = 0;
  for (int r=1; r<=R; r++) {
    for (int s=1; s<=S; s++) if (A[r][s]) vyska[s]++; else vyska[s]=0;
    fill(dolava,dolava+S+2,0);
    fill(doprava,doprava+S+2,0);
    fill(pocty,pocty+r+2,0);
    for (int s=1; s<=S; s++) vedierka[ vyska[s] ][ pocty[vyska[s]]++ ] = s;
    for (int v=r; v>0; v--) for (int ss=0; ss<pocty[v]; ss++) {
      int s = vedierka[v][ss];
      result += v * (dolava[s-1]+1) * (doprava[s+1]+1);
      int vlavo=s-dolava[s-1], vpravo=s+doprava[s+1], dlzka=vpravo-vlavo+1;
      doprava[vlavo]=dolava[vpravo]=dlzka;
    }
  }
  printf("%Ld\n",result);
  return 0;
}
```

A-III-2 Odveta

Ako každá matematická hra[1], aj naša hra s koníkmi je v každom momente v nejakej pozícii (stave). Pozíciu vieme jednoznačne popísať tak, že uvedieme aktuálne súradnice všetkých koníkov.

Pod *vyhávajúcou stratégiou* budeme rozumieť postup, ktorý hráčovi zaručí víťazstvo bez ohľadu na ťahy protihráča. Pozícia je *vyhávajúca*, ak hráč, ktorý je na ťahu, má vyhávajúcu stratégiu. Pozícia, ktorá nie je vyhávajúca, je *prehrávajúca*. Zjavne každá pozícia je buď vyhrávajúca alebo prehrávajúca.

Pozíciu, z ktorej neexistuje ťah, nazývame *koncová*. V našej hre sú podľa pravidiel všetky koncové pozície prehrávajúce.

Keďže v súťažnej úlohe máme proti sebe optimálne ťahajúceho protihráča, zaujíma nás presne to, či je začiatočná pozícia pre nás vyhrávajúca, alebo nie. Ako to určiť?

Pri charakterizácii pozícii si môžeme pomôcť nasledovnými myšlienkami:

1. Ak je pozícia koncová, je prehrávajúca.
2. Ak z danej pozície všetky ťahy vedú do vyhrávajúcich pozícií, tak je táto pozícia prehrávajúca.
3. Ak z danej pozície existuje ťah do prehrávajúcej, tak je táto pozícia vyhrávajúca.

(Ak všetky ťahy vedú do vyhrávajúcich pozícií, nech si vyberieme ktorýkoľvek, vždy tým dostaneme súpera do vyhrávajúcej pozície. A ak sa potom bude súper držať nejakej vyhrávajúcej stratégie, hru prehráme. Preto takáto pozícia je prehrávajúca. Naopak, ak existuje ťah do prehrávajúcej pozície, spravíme ho, a tým dostaneme súpera do tejto, pre neho prehrávajúcej pozície.)

Túto myšlienku ľahko prepíšeme do rekurzívnej funkcie, ktorá nám o pozícii povie, či je vyhrávajúca alebo prehrávajúca.

Hra s jedným alebo dvomi koníkmi

Podľa nenápadnej rady v zadaní sa teraz zamyslime nad jednoduchšou verziou hry a uvažujme, že na šachovnici je len jeden koník.

Problém predchádzajúceho prístupu spočíva v tom, že je príliš pomalý. Hlavný dôvod je ten, že pri rekurzívnych volaniach vlastne skúša všetky možné priebehy hry, a pri tom mnohé pozície vyhodnotí veľa krát.

Tu si môžeme pomôcť takzvanou memoizáciou – akonáhle o nejakej pozícii zistíme, či je vyhrávajúca alebo prehrávajúca, zapíšeme si to do pomocného poľa. Takto dosiahneme to, že každú pozíciu budeme spracovávať práve raz a riešenie vykoná pre každú spracovávanú pozíciu konštantný počet operácii.

A koľko pozícií budeme spracovávať? Všimnime si, že každým ťahom sa priblížime k políčku $(0, 0)$. Presnejšie, súčet súradníc sa nám zníži aspoň o jedna. To znamená, že počet spracovávaných stavov môžeme zhruba zhora odhadnúť počtom bodov, ktoré majú súčet súradníc menší alebo rovnaký ako počiatočná pozícia. V prípade, že počiatočná pozícia je (X, Y), potom je týchto pozícií $O((X+Y)^2)$, a taká je aj časová zložitosť tohto algoritmu.

Uvedené myšlienky sa dajú veľmi jednoducho rozšíriť aj viacero koníkov. Ak máme N koníkov a každý začína na súradniciach so súčtom nanajvýš S, tak je počet dosiahnuteľných pozícií rádovo rovný S^{2N}. Takéto riešenie je teda použiteľné len pre veľmi malé hodnoty N.

Rýchlejšia charakterizácia pozícií pre jedného koníka

Vráťme sa k najjednoduchšej možnej hre s jediným koníkom.

Ak chceme vedieť rýchlejšie povedať, ktorá pozícia je vyhrávajúca a ktorá prehrávajúca, potom by sme mali nájsť charakterizáciu pozícii, ktorá nevyužíva vedomosti o pozíciach naokolo. Veľmi často pomáha skúsiť si vypočítať pre niekoľko najmenších pozícií, či sú vyhrávajúce, a hľadať nejakú závislosť.

Ak by sme v našom prípade vyhodnotili všetky pozície, ktorých súčet súradníc nepresahuje 13, dostali by sme nasledujúcu tabuľku:

[1] Presnejšie, konečná kombinatorická hra s úplnou informáciou.

```
0
0 0
1 1 1
1 1 1 1
0 0 1 1 0
0 0 1 1 0 0
1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
0 0 1 1 0 0 1 1 0
0 0 1 1 0 0 1 1 0 0
1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1
0 0 1 1 0 0 1 1 0 0 1 1 0
0 0 1 1 0 0 1 1 0 0 1 1 0 0
```

Políčko $(0, 0)$ sa nachádza v ľavom dolnom rohu. Nulou sme označili prehrávajúce pozície, jednotkou vyhrávajúce. Nie je obtiažne spozorovať vzorku – celá šachovnica je zložená z dlaždíc veľkosti 4×4, na ktorých sú 4 políčka vľavo dole prehrávajúce a zvyšné vyhrávajúce. Preto môžeme vysloviť nasledujúcu hypotézu:

Pozícia (X, Y) je prehrávajúca práve vtedy, ak $X \bmod 4 \in \{0, 1\}$ a $Y \bmod 4 \in \{0, 1\}$.

Túto hypotézu dokážeme matematickou indukciou podľa súčtu $X+Y$. Pre malé hodnoty sme už toto tvrdenie dokázali zostrojením vyššie uvedenej tabuľky, stačí sa teda sústrediť na indukčný krok.

Majme teda nejakú pozíciu (X, Y), pričom o všetkých pozíciách, ktoré majú súčet súradníc menší ako $X+Y$, už vieme, že naša hypotéza pre ne platí. Dokážeme, že platí aj pre (X, Y). Rozoberieme niekoľko možností podľa toho, aké zvyšky po delení 4 dávajú X a Y.

- Obe súradnice dávajú zvyšok 0 alebo 1:

  Nech potiahneme ľubovoľne, jednu súradnicu vždy zmeníme o 2 a druhú o 1. No a tá súradnica, ktorú sme zmenili o 2, bude teraz dávať po delení 4 zvyšok 2 alebo 3. A teda podľa indukčného predpokladu bude táto nová pozícia pre hráča na ťahu vyhrávajúca.

  Tým sme zdôvodnili, že nech potiahneme akokoľvek, vždy dostaneme súpera do vyhrávajúcej pozície. Naša pozícia je teda prehrávajúca, čo sme aj chceli dokázať.

- Jedna súradnica dáva zvyšok 2 alebo 3, druhá 0 alebo 1:

  Potiahneme tak, že prvú súradnicu zmenšíme o 2.
  Ak druhá dáva zvyšok 1, tak ju o 1 zmenšíme, inak ju o 1 zväčšíme.

- Obe súradnice dávajú zvyšok 2: Zmenšíme X o 2 a Y o 1.

- Obe súradnice dávajú zvyšok 3: Zmenšíme X o 2 a zväčšíme Y o 1.

- Jedna súradnica dáva zvyšok 3 a druhá 2: Prvú súradnicu zmenšíme o 2 a druhú o 1.

  Vo všetkých štyroch prípadoch sme ukázali ťah, ktorým súpera dostaneme do pozície, ktorá je podľa indukčného predpokladu prehrávajúca. Z čoho vyplýva, že naša pozícia musela zakaždým byť vyhrávajúca, č.b.t.d.

Pre hru s jedným koníkom teda vieme v konštantnom čase povedať, či je pozícia vyhrávajúca alebo nie.

Všeobecná hra

Čo ale v prípade, ak máme koníkov viac? Pri hľadaní všeobecného riešenia pomôže, ak sa budeme pozerať na vzor vyššie a skúšať sa po ňom hýbať dvoma, prípadne iným malým počtom koníkov. Takto môžeme objaviť nasledujúcu charakterizáciu pozícií:

- Ak sú všetky koníky na políčku s nulou, tak je pozícia prehrávajúca.
- Ak existuje koník, ktorý je na políčku s jednotkou, tak je pozícia vyhrávajúca.

Dôkaz: Pozíciu, kde sú všetky koníky na políčkach s nulou, budeme volať modrá, ostatné pozície budeme volať červené. Ľahko nahliadneme, že platia nasledujúce tri tvrdenia:

1. Všetky koncové pozície hry sú modré.
2. Ak sme v modrej pozícii, tak ľubovoľným ťahom z nej vyrobíme červenú pozíciu.

   Musíme totiž pohnúť aspoň jedného koníka. A keďže v hre s jedným koníkom všetky ťahy z prehrávajucej pozície (t. j. políčka s nulou) vedú do vyhrávajúcich pozícií (na políčka s jednotkou), koník, ktorého pohneme, skončí na políčku s jednotkou, a teda nová pozícia bude červená.
3. Ak sme v červenej pozícii, tak z nej vieme platným ťahom vyrobiť modrú pozíciu.

   Vyberieme všetky koníky, ktoré stoja na políčkach s jednotkou. Keďže tieto políčka dostali jednotky, z každého z nich existuje skok na nejaké políčka s nulou. Takéto skoky porobíme, čím dosiahneme, že výsledná pozícia bude modrá.

Teraz vidíme, že modré a červené pozície presne spĺňajú našu definíciu vyhrávajúcich a prehrávajúcich pozícií. Nutne sú teda všetky modré pozície prehrávajúce a všetky červené vyhrávajúce.

Keď máme situáciu s N koníkmi, potom stačí len pre každého v konštantnom čase zistiť, či je na políčku s nulou alebo na políčku s jednotkou. V prípade, že existuje koník na políčku s jednotkou, pozícia je vyhrávajúca. V takom prípade vieme o každom jazdcovi na jednotke zistiť v konštantom čase, kam ho treba presunúť. Časová zložitosť tohto riešenia je teda $O(N)$. Všimnite si, že vieme koníky spracúvať po jednom, a teda nám stačí konštantná pamäť.

Listing programu:

```
#include <iostream>
using namespace std;
int N,x,y;
int dx[4] = {-2,-2,-1,1}, dy[4] = {1,-1,-2,-2}; // povolene tahy kona

//metoda zisti, ci je konik na prehravajucej pozicii v hre s 1 konom
bool prehravajuci(int x, int y){
  return  (x%4 < 2) && (y%4 < 2);
}

int main(){
  cin >> N;
  int prehra=1;
  for (int i=0;i<N;i++){
    cin >> x >> y;
    if ( !prehravajuci(x,y) ){
      if (prehra == 1) cout << "Cedecko" << endl;
      prehra = 0;
      //treba pohnut kazdym konikom, ktory nie je na prehravajucej pozicii
      cout << x << " " << y << " ";
      for (int j=0;j<4;j++){
        int nx = x + dx[j], ny = y + dy[j];
        if ( nx < 0 || ny < 0 ) continue;
        if ( prehravajuci(nx,ny) ){
          cout << nx << " " << ny << endl;
          break;
        }
      }
    }
  }
  if (prehra == 1) cout << "Petrzlen" << endl;
  return 0;
}
```

A-III-3 Počítač s gumenou rúrou

Podúloha a: Kontrola, či je postupnosť rastúca

Začneme obľúbeným trikom: na koniec postupnosti si ako zarážku vložíme 0.

Teraz načítame do registra a prvé číslo postupnosti, a za zarážku vložíme číslo 1.

Od tohto okamihu ďalej budeme čísla z rúry striedavo načítavať do registrov b a a. Vždy, keď načítame nové číslo, porovnáme ho s predchádzajúcim. Ak je to nové väčšie, vložíme do rúry ďalšie číslo 1. Ak nie, našli sme práve hľadanú prvú chybu. Všimnime si, že počet 1, ktoré sme doteraz do rúry vložili, je presne rovný indexu, ktorý máme vypísať. Stačí teda vyprázdniť z rúry všetko po zarážku a následne všetky 1 sčítať, aby sme dostali odpoveď, a túto vypíšeme.

V opačnom prípade, teda ak sa nám podarí dočítať celú zadanú postupnosť až po zarážku, jednoducho program ukončíme.

Celý program môže vyzerať napr. nasledovne:

```
get a ; put 0
jz a koniec
jump citaj_b

label citaj_b
  put 1 ; get b
  jz b koniec
  jgt b a citaj_a
jump vyprazdni

label citaj_a
  put 1 ; get a
  jz a koniec
  jgt a b citaj_b
jump vyprazdni

label vyprazdni
  get a ; jz a scitaj
jump vyprazdni

label scitaj
  get a ; jempty vypis ; put a ; add
jump scitaj

label vypis
  put a ; print
jump koniec

label koniec
```

Podúloha b: Hľadanie majoritného prvku

Prvé jednoducho naprogramovateľné riešenie bude založené na nasledujúcej myšlienke: Nech a a b sú dva prvky zadanej postupnosti, ktoré sú navzájom rôzne. Potom ak oba tieto prvky vynecháme, dostaneme opäť postupnosť, ktorá má majoritný prvok (a je to ten istý prvok ako predtým).

Prečo? Jednoducho preto, že sme vyhodili nanajvýš jeden z prvkov, ktorých bola väčšina, a aspoň jeden z prvkov, ktorých väčšina nebola.

Náš program bude teda dokola opakovať nasledujúci postup: Číta postupnosť a overuje, či sú všetky prvky rovnaké. Ak sa dočíta na koniec a zistí že sú, jeden z nich vypíše a skončí. V opačnom prípade nájde dvojicu rôznych prvkov. Túto odstráni, postupnosť dočíta do konca a začne odznova.

Tento program bude mať časovú zložitosť kvadratickú od dĺžky zadanej postupnosti. Totiž každú iteráciu uvedeného postupu vieme vykonať v čase lineárnom od aktuálnej dĺžky postupnosti a platí, že jedným jej vykonaním zmenšíme dĺžku postupnosti o 2.

(Síce by sme mohli postupne vyhadzovať aj viac dvojíc, ak ich nájdeme postupne, ale v najhoršom možnom prípade bude aj tak takéto riešenie potrebovať kvadraticky veľa krokov – napr. pre postupnosť obsahujúcu k jednotiek a následne $k+1$ dvojok. Preto radšej zostaneme pri ľahšie napísateľnom programe.)

```
put 0

label kolo
  get a
  label loop
    get b ; jz b koniec
    jeq a b dalej
  jump docitaj

  label dalej
    put b
  jump loop

  label docitaj
    get a ; jz a docital ; put a
  jump docitaj

  label docital
    put 0
jump kolo

label koniec
put a ; print
```

Existuje však aj lepšie riešenie. Uvažujme najskôr situáciu, kedy je celková dĺžka n našej postupnosti párna. Rozdelíme našu postupnosť na $n/2$ párov. V niektorých z nich sú obe čísla rovnaké, v niektorých sú rôzne. Už sme si zdôvodnili, že každý pár, kde sú čísla rôzne, môžeme zahodiť. Zostane nám teda niekoľko párov, v ktorých sú čísla rovnaké. Čo s nimi? Z každého páru jedno číslo necháme a to druhé zahodíme.

Rozmyslite si, že touto operáciou nič nepokazíme – ak mal nejaký prvok väčšinu pred ňou, tak to znamená, že nadpolovičný počet dvojíc bol tvorený týmto prvkom. A teda po „vydelení dvoma“ bude mať tento prvok stále nadpolovičný počet výskytov. Tým sme teda ukázali, že ak je n párne, vieme v lineárnom čase našu úlohu previesť na úlohu nanajvýš polovičnej veľkosti.

Ako to bude vyzerať pre nepárne n? Označme x prvok, ktorý má v našom poli väčšinu. Dvojicu prvkov nazveme dobrá, ak sú oba rovné x, a zlá, ak sú oba rovnaké, ale iné od x. Posledný prvok označme p a zatiaľ ho odložme nabok. Ostatných $2m$ prvkov rozdeľme rovnako ako predtým na m dvojíc.

Ak $p \neq x$, sú všetky výskyty x (ktorých je aspoň $m+1$) rozdelené v týchto dvojiciach. To je presne predchádzajúci prípad, vieme teda, že dobrých dvojíc máme viac ako zlých.

Ak $p = x$, najhoršie, čo sa môže stať, je že máme presne m výskytov x v dvojiciach a že tie sú rozložené tak, že je dobrých a zlých dvojíc presne rovnako. (Dôkaz že dobrých nemôže byť menej ako zlých: Predstavme si, že zoberieme ľubovoľné nie-x a vymeníme ho s p. Dostaneme situáciu, v ktorej je ostro viac dobrých ako zlých dvojíc, a pritom sme zmenili len jednu dvojicu.)

Rozlíšme teraz dva prípady. Ak je dokopy dobrých a zlých dvojíc nepárny počet, už sme vyhrali. Totiž spolu s informáciou, že dobrých je aspoň toľko ako zlých, z toho vieme povedať, že dobrých je viac. Môžeme teda spraviť to isté ako pre párne n (a posledný prvok p jednoducho zahodiť).

Ak je dobrých a zlých dvojíc dokopy párny počet, necháme z každej z nich po jednom prvku *a navyše* ku nim pridáme posledný prvok p. Čo tým dostaneme? Ak bolo dobrých dvojíc viac ako zlých, bolo ich nutne viac aspoň o dve (aby bol celkový počet párny), a teda pridanie p nič nepokazí. A ak bolo dobrých dvojíc rovnako ako zlých, tak vieme, že $p = x$, a teda po pridaní p bude mať x opäť väčšinu.

Ešte raz teda zhrnieme celý algoritmus pre $n = 2m + 1$:

Postupne vyrobíme m dvojíc. Každú, v ktorej sú prvky rôzne, zahodíme, a z každej, v ktorej sú prvky rovnaké, jeden necháme a druhý zahodíme. Pritom si pamätáme paritu počtu prvkov, ktoré sme už nechali. Keď sa dostaneme k poslednému prvku p, ktorý už nemá pár, podľa zapamätanej parity sa rozhodneme, či ho nechať alebo zahodiť.

Vyššie popísaný algoritmus teda v lineárnom čase spracuje postupnosť, ktorú má v rúre, a vyrobí v nej novú postupnosť približne polovičnej dĺžky. Pritom platí, že nová postupnosť má ten istý majoritný prvok ako tá pôvodná. Ľahko spočítame, že opakovaným použitím tohto algoritmu (až kým nám nezostane jediný prvok) dostaneme odpoveď v lineárnom čase.

```
label kolo
get a ; jempty koniec
put a ; put 0
jump even

label even
  get a ; jz a kolo
  get b ; jz b posledne
  jeq a b pis_odd
jump even

label odd
  get a ; jz a kolo
  get b ; jz b kolo
  jeq a b pis_even
jump odd

label pis_odd  ; put a ; jump odd
label pis_even ; put a ; jump even
label posledne ; put a ; jump kolo
label koniec   ; put a ; print ; stop
```

SLOVENSKÁ KOMISIA OLYMPIÁDY V INFORMATIKE

DVADSIATY PIATY ROČNÍK OLYMPIÁDY V INFORMATIKE

Vydala IUVENTA s finančnou podporou Ministerstva školstva SR
Náklad: 40 výtlačkov
Zodpovedný redaktor: Michal Forišek
Sadzba programom LaTeX

© Slovenská komisia Olympiády v informatike, 2010